ConvQASARE : End-to-End Cognitive Conversational Question Answering using GraphSAGE and RotatE

Marina Sruthi Maria Michael

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2022

Abstract

Conversational Question Answering (ConvQA) is a critical Natural Language Processing (NLP) problem and a long-standing artificial intelligence milestone. A multi-turn ConvQA has proven to be more challenging than a regular QA since it requires learning the history of each conversation. However, the history of the conversation is not efficiently captured by most of the current approaches, thus failing on questions involving coreferences. Additionally, most of the graph-based approaches fail to capture rich word attributes and long-range dependency along with the global structure when reasoning over a context. In this project, we propose a cognitive way of an end-to-end model (ConvQASARE) for ConvQA that jointly performs Knowledge Graph Completion (KGC) and Question-Graph reasoning. Our model delivers an end-to-end model that learns history by completing a knowledge graph using a simple yet effective translational model proposed in RotatE. We leverage the inductive representation learning technique proposed in GraphSAGE to model long-range dependency and rich node attribute information. Our model, ConvQASARE, mimics human cognitive reasoning which gets reliable performance on a ConvQA benchmark: Question Answering in Context (QuAC). Additionally, experiments show that ConvQASARE model can offer good scalability for the reasoning process and efficient history selection.

Acknowledgements

This work is mainly supported by my supervisor, Prof. Jeff Pan, who gave me a lot of help and mentoring on this thesis. I also need to thank the School of Informatics, University of Edinburgh for the provided computation resources, Eddie, for completing the experiments. As well, thanks to my family, Mr Maria Michael, Dr L Josephine Mary, Mr Cyril Bastin and Mrs Nisha John, who provided emotional support and guidance throughout this work.

Table of Contents

Chapter 1

Introduction

Conversational Question Answering (ConvQA) is one of the fundamental problems in Natural Language Processing (NLP), which involves predicting an answer to a query in context of previous conversations. When the ConvQA task involves machines to extract the start and end span from the context passage as an answer to the question, the ConvQA follows the conversational machine reading comprehension (CMRC) framework [26]. Each conversation consists of multiple questions, thus making it a multi-turn conversational QA [26].

One of the primary challenges of multi-turn ConvQA which makes it more challenging than a regular QA is learning the history of each conversation. Each question lacks the context that is important to understand them completely and are referenced from any of the previous question-answer pair (history) from the conversation [44]. For example, in Figure 1.1, the *"there"* in the sample question *"What languages are spoken there"*, is referenced from the previous answer *"Kerala"*. Another important challenge is that questions can be unanswerable due to ambiguity or insufficient context [60]. Due to these challenges, many deep learning models fail to achieve human performance, especially on ConvQA benchmark datasets for English like Question Answering in Context (QuAC) [10] which is a 6.7 F1 score behind human performance. Thus it is a widely researched field in NLP currently.

Many recent models targeting ConvQA are mostly pre-trained language models (PLM) and very rarely graph-based models. The standard state-of-the-art model is a transformer-based PLM using BERT [13] and RoBERTa [61] which understand conversation context better. These models combine history, current context, and query as an input sequence. As these models are limited to only 512 sequences, [44], [54] show that combining questions with previous conversation history and context becomes very

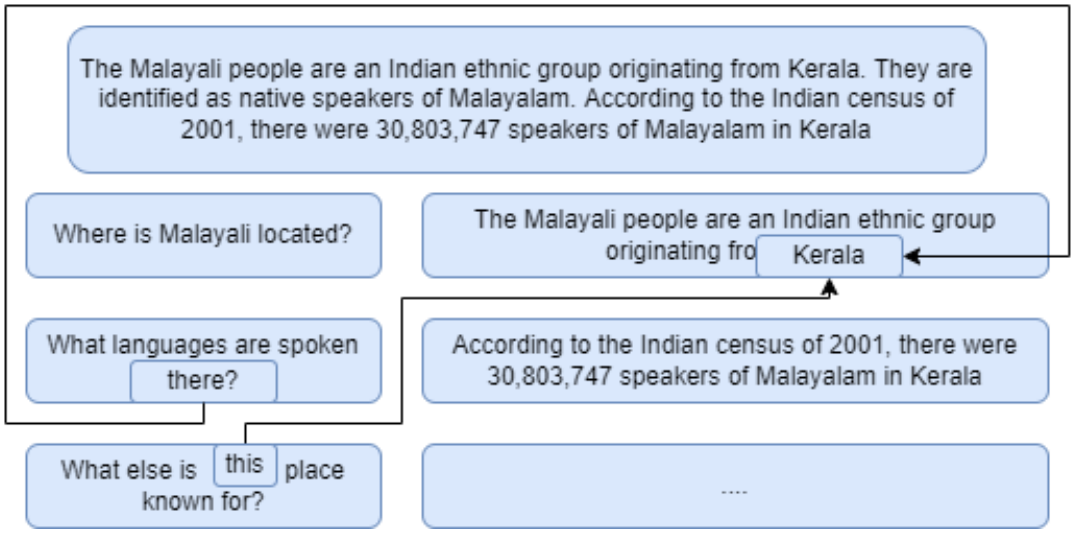

Figure 1.1: Sample question-answer pair from a ConvQA benchmark (QuAC [10]) along with its coreferences.

long and difficult to maintain. For example, the RoBERTa-based model (125M parameters with 768 embedding size) on QuAC needs 40 hours of training on 8 Nvidia V100 32GB GPUs.

These issues lead to inefficient training on ConvQA tasks and promote more research on graph-based models which use Graph Neural Networks (GNN). It is easy to maintain previous conversation context and history in Knowledge Graphs (KGs) [24]. Graph-based models like [8], [27], and [14] use Gated Graph Neural Network (GGN) [32], Graph Attention Network (GAT) [48], and Graph Convolution Network (GCN) [28] to achieve considerable performance on ConvQA tasks. However, they commonly suffer from the following issues:

i) Most of the KG-based QA systems are attempted on publicly available KG (triplets or graph) datasets rather than free-form text datasets. That is because manually constructing a KG can accumulate errors and be carried forward from the construction stage to the learning stage [40]. Also, the primary reason for the accumulation of errors while constructing KG is that it suffers from a poor understanding of an entity in a context as they are co-referenced from previous sentences (Figure 1.2). For example, constructing a KG from the context in Figure 1.2 results in *("They", "identified", "speakers")* which makes the error to accumulate, resulting in difficulty in reasoning for any GNN.

ii)As mentioned previously, the main issue with PLM in ConvQA is that it combines previous questions and answers to the current question as history. This history,

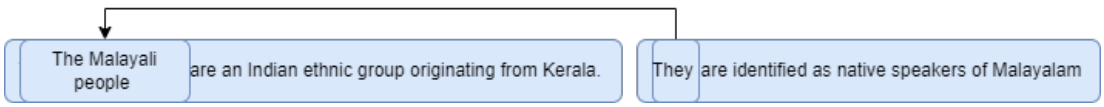

Figure 1.2: Sample context from a ConvQA benchmark (QuAC [10]) along with its coreferences.

can extend very long, more than 512 sequences. Even most graph-based models like [14] and [27] follow the same methods to handle history. For example, for a question in a conversation that has 10 previous visited questions, GraphFlow [8] combines all the 10 previous questions and answers to the current question as history therefore resulting in inefficient training.

iii)In NLP, the representation of words as entities results in long and complex KGs. This can increase exponentially and is referred to as the Combinatorial explosion of knowledge [7]. Graph Neural Networks (GNN) such as GAT [48] and GGN [32] constructs an adjacency matrix and holds them completely in memory during training. Thus when the KGs become long and complex, these models fail to capture the global semantic structure of a graph. These models also requires several linguistic features like Parts of Speech (POS) tags and Named Entity Recognition (NER) to capture better word order and long-range dependency effectively.

Despite efficient training compared to PLM's, these issues restrict graph-based models from achieving state-of-the-art performance in multi-turn ConvQA, especially in the QuAC dataset. Therefore, to fill the gaps above, this work proposed a cognitive way of an end-to-end model for ConvQA that jointly learns Knowledge Graph Completion (KGC) and question-graph reasoning. This work leverages inductive representation learning proposed in GraphSAGE [20] and Knowledge Graph Embedding (KGE) proposed in RotatE [45]. The above architecture follows dual process theory [4] to mimic the human way of reasoning. Taking after the architecture, the solution is constructed in two phases. The first phase is the dynamic node embedding construction phase using GraphSAGE while the second phase is a joint reasoning phase where KGC is used as history and reasoned jointly with question and graph embedding. The main aim of this project is to implement the above model to fill the gap and investigate if the proposed architecture increases the performance of ConvQA task compared to the best graph-based model in the QuAC dataset. The experiments show that the proposed method overcomes all challenges while achieving better performance in ConvQA task.

The major results and contributions of this research will be to:

- Propose an end-to-end architecture with knowledge graph completion and question-graph reasoning in a cognitive way that is scalable and captures better global semantics, rich node attributes while at the same time inductive i.e., generalizes better for unseen nodes.
- Construct a baseline model that creates a base framework for our proposal using an unscalable, less efficient GNN and without a knowledge graph based history selection module.
- Construct an end-to-end model with scalable, inductive GNN GraphSAGE and KGC as history in advance of the baseline model, which follows the proposed architecture to overcome the issues mentioned above.
- Experiment with the proposed architecture on conversational question answering benchmark, and get good performance in F1 metrics.
- Investigate the proposed architecture with the best Graph-based model in the QuAC benchmark dataset.

The remainder of this dissertation is organised as follows. **Chapter 2** briefly introduces the background and related works concerning this thesis. **Chapter 3** describes the basic modules used, the baseline model, and the ConvQASARE models, in detail. **Chapter 4** reports the results of the baseline and ConvQASARE on various experiments including the analysis of these results. **Chapter 5** discusses some common findings in the experiment part, which is the extended analysis part for **Chapter 4**. **Chapter 6** reports the conclusion of this work, including limitation of this work, and future works related to the limitation.

Chapter 2

Related Works

2.1 Knowledge Graphs

Knowledge Graphs (KGs) are knowledge representations of data inspired by how humans solve their problems. They consist of nodes and edges in which nodes are connected together to form an edge. When the nodes and edges are represented in the form of (Subject, Relation, Object) triplet, KGs are also referred to as Knowledge Base (KBs), where Subject and Object are nodes and Relation is an edge between subject and object. In NLP, entities of a given text are represented as nodes and the semantic relationship between them are represented as edges. For example, from Figure 1.1 *"Malayali People"*, *"Indian ethnic group"* and *"Kerala"* are all entities/nodes, and the semantic relationship/edges between the entities *"Malayali people"* and *"Kerala"* is *"originating from"*. This is represented as triplets *("Malayali people", "originating from", "Kerala")*. Few models [35], [50] also treat every word as a node with 'neighbors' as its edge. The construction of a knowledge graph happens in two ways: i)Manual and ii)Dynamic. Manual construction of KG is carried out by a professional annotator which is a time-consuming process [58]. While dynamically constructing KGs from the text data is achieved by extracting relations between entities in text and forming triplets. Extraction of relations and resolving entities is easily achieved with many open-source libraries like nltk [36] and spacy [11] which in turn use varying neural network methods. This can also be achieved by using more advanced deep learning methods.

2.2 Embedding

Representing words as low-dimensional vector spaces that reflect their syntactic and semantic information is an important part of NLP research. In the early stage of NLP research, these representations were static i.e., a single vector for a word ignoring the meaning of words in different contexts. Word2Vec [37] or Glove [39] are some example methods for obtaining static word embedding. As these ignore the variability of word meaning in different linguistic contexts, vectors injected with varying context information was required [23]. Therefore contextualised word embeddings were introduced. Transformer [47] based PLM like BERT [13], RoBERTa [61] produces contextualised word embedding which has become a commonly used embedding in most NLP-related areas.

Similar to contextualized word embedding, several approaches have proposed the projection of knowledge graphs (KGs) into lower embedding vector space while preserving the semantic information of KG. The primary challenge in projecting the KGs into embedding space, is to encode a node's global information and its neighbor's information. To achieve this, in the early stages of research, various statistics like node degrees, clustering coefficients, kernel functions, or hand-engineered features were used [29]. However, an end-end learning method cannot be applied to these statistics resulting in graph representational learning. The main idea behind graph representation learning is to map semantic information of KGs to low-dimensional vector space. These representations make the numerical computation of KG easier, while preserving and encoding the geometric and global information of the knowledge graph [49]. Models like TransE [5], DistMult [53], and RotatE [45] achieve these representations by defining different score functions. These score functions measure the distance between two entities relative to their relation in a low-dimensional vector space, thus forming knowledge graph embeddings (KGE). The TransE model aims to achieve inversion and composition patterns while DistMult achieves a symmetry pattern. But all these older models are incapable of inferring all three patterns together. On the other hand, RotatE [45] maps each relation as a rotation from source to target entity achieving all three patterns together. These representations are used in a wide range of downstream tasks like Link Prediction [33], [46], Relation extraction [19], [36], and knowledge graph completion [41]. Knowledge Graph Completion (KGC) is a graph-based task that automatically finds the missing subject or object entity in a given triplet. For example, consider the triplet *("Malayali People", "originating from ", "?")*, this

can be completed by filling the missing object entity as *"Kerala"*, where *"?"* is the missing entity. In this work, the representations generated using RotatE are used in a KGC task which acts as a history to the ConvQA task and BERT [13] based contextual word embedding are used as the question and context features.

2.3 Dual Process Theory

Researchers found that humans use a dual process approach when they reason over large-capacity memory for a given question about a context already present in their memory. Derived from cognitive science, a dual process theory [4] assumes that decision-making is achieved in a two-stage processes in humans. The unconscious system 1 that eliminates irrelevant information and provides a highly contextualized representation of the problem. The conscious system 2 decides or provides an answer by extreme analysis on the obtained representation. Researchers used this theory in various AI tasks like question answering [16], [15], and pattern recognition [30]. In our work, the entire system is built to mimic the dual process theory, thus achieving a cognitive way of answering questions.

2.4 Knowledge Graph Question Answering System

In the early stages, traditional approaches of graph-based QA translate a natural language question into structured queries such as SPARQL and return entities from KGs as answers [21]. [25], [21] finds the answer entity by finding the closest subject and relation to a simple NLP question using GNN like GCN [28], GAT [28]. Then the corresponding object entity is returned as the answer. The above approach is applied to problems when the dataset is structured or if the required answer is a single entity from the triplet. However, the above framework could not be directly used in ConvQA when the required answer is to be extracted from the context [18]. Therefore a graph-based CMRC task framework was introduced to predict answer spans based on natural language questions and KG . [18]defines this framework into four main modules:

History Selection Module: This module selects the history of a conversation. Some models like GraphFlow [8], [43] concatenate previous questions and answers to the current question as a history. Some models like [10] also concatenates the turn number of each question and location of the answer as history features.

Encoding: In this module, the Context passage $C^{(i)}$, History $H^{(i)}$ and Question $Q^{(i)}$ are encoded into input embedding based on various approaches defined in section 2.2. Some models like [52], [24] use lexical embedding and question aware and intra sequence contextual embedding. Some methods like GraphFlow [8] uses nine types of features like Parts of Speech (POS), Named Entity Recognition (NER), Glove and BERT Embedding, Question Aligned Context embedding, Question marker, Answer marker, number of question words matching with context words, and context word frequency. All this embeddings are concatenated to get encoded features.

Reasoning: This module involves the combined reasoning of questions and graphs. There are two layers involved in this module. The first layer is the question understanding layer which is usually carried out using an encoder-decoder architecture similar to [54] or [38], which skips the encoding module for questions and directly uses PLM models like BERT [13] and RoBERTa [61]. In both types, the hidden states of the encoder or the PLM models are used as output to jointly reason with graphs for predicting answers.

The second layer is the Graph Reasoning layer, which involves graph learning and reasoning over the generated knowledge graph using Graph Neural Network (GNN). GNN has become an important research field in AI. Recently, GNNs are applied to various question answering tasks like MRC [16], [30] and knowledge base question answering (KBQA) [25] and have achieved better performances than traditional approaches. GNN learns individual node embedding that is fed into downstream tasks as node features. Network like GCN [28], GGN [32], GAT [45] distills high-dimensional information about a node and its neighborhood thus forming node embedding. GraphFlow [8] first constructs a adjacency matrix treating every word as node and performs knn-style graph sparsification to get a normalised adjacency matrix. It applies Gated Graph Neural Network (GGN) to aggregate plenty of information from its surroundings on the weighted normalized adjacency matrix. And then on the output of the GNN cell applies Recurrent Graph Neural Network to get sequential output. Cognitive Graph QA [16] extracts possible answers and next-hop entities as clues and passes question-clue-context as input to the BERT system. The output of BERT system is passed as latent representations to a GAT layer which fetches node embedding. But these models cannot scale when the size of the KG increases, thus failing to capture global semantics, long-range dependencies and rich node attribute information. Therefore, models like GraphSAGE [20] and GIN [51] were introduced. They use inductive

learning to generate low-dimensional vector representations for graph nodes and highly scalable. GraphSAGE [20] uses an aggregator function that helps in better capturing rich node attribute information while scaling for larger contexts. [49]uses GraphSAGE to overcome the above challenge in a question correction and generation task. In this work, GraphSAGE is used as the GNN reasoning layer with single and multiple hops.

Prediction: This module is used to predict answers using the reasoning layer output. A major challenge of this module is to handle abstractive answers rather than to predict the most relevant entity candidate from the graph. [31] proposes extracting answers from the encodings using augmented text generators. [8] reasons the question using an encoder and self-attention module and then predicts an answer by combining the question reasoning output and graph node embedding. [16] finds the probability of a Yes/No question and extracts answers only if Yes/No was not selected.

In our work, the above framework was adopted with different variations in each module which will be explained in detail in the next chapter.

Chapter 3

Methodology

In this chapter, the basic modules that are necessary to our proposal, such as Spacy Neural Co-Reference, Recurrent Neural Network, Attention Mechanism and Graph Neural Networks are introduced. The baseline model, the proposed (ConvQASARE) model and its overall architecture are also explained in detail.

3.1 Notations

This section briefs out the various notations used as part of this thesis. In a conversational question answering system, the number of turns in a conversation is given as i with j^{th} context word as C_j and k^{th} question word as Q_k with history as $H_k^{(i)}$. x_t and h_t at timestamp t represents the input and hidden state to recurrent models (e.g. Bi-Directional Long Short Memory (BiLSTM)). h_t serves as an input and c_t as an output to attention mechanism. Concatenation of two vectors a and b are indicated by [a, b]. A knowledge graph G is represented as $\{v, e\}$ where v is the node and e is the relation/edge between two nodes. A knowledge base is given in the form of a triplet T which is represented as (s, e, o) where s refers to the subject entity and o refers to an object entity. Node v can also be understood as a list of unique subject and object entities in a given context. The input to any GNN model is the knowledge graph G and a feature vector W_x^i, where x represents the input for which the feature vectors are generated. For example, a feature vector of subject and object entity is represented as W_s^i and W_o^i, whereas the feature vector of context and question is represented as W_{cj}^i and W_{qk}^i respectively. The output of a GNN network at k^{th} layer is represented as h_v^k where the final node embedding is given as z_v. The aggregator function used in the GraphSAGE model is given by agg and feature dropout is given by $featdrpt$.

3.2 Basic Modules

In this section, the five basic modules related to this work- Spacy Neural Co-Reference, Recurrent Neural Networks like Bi-Directional Long-Short Term Memory and Gated Recurrent Unit, Attention Mechanism, RotatE and GraphSAGE network that served as primary models are introduced.

3.2.1 Spacy Neural Co-Reference

Co-Reference Resolution aims to find all the text that refers to the same mentions in a given context. For example, consider *"The Malayali people are originated from Kerala, and they speak Malayalam"*, a co-reference resolution model that aims to find all the reference for the mention *"they"*. This is achieved by learning a scoring function for each span of the given context. The scoring function computes the probability of the given span referring to a particular mention of the text [57]. In specific, Spacy neural coref API[1] uses a two neural network approach to resolve co-references. The first net calculates the score for the mention having an antecedent in a text and the second net calculates the score for the mention having no antecedent. The highest of these gives the resolved text for the mention. For example, consider the above, the first neural net calculates the score of *"Malayali people"* being an antecedent of *"they"* and the second neural net calculates the score of *"they"* having no antecedent. *"Malayali people"* gets the highest score and the mention *"they"* is resolved for its co-reference text *"Malayali people"*. This method provides easy integration with any NLP processing pipeline. In this work, spacy neural coref resolution is used to achieve better contextual construction of knowledge graphs.

3.2.2 Recurrent Neural Network

A Recurrent Neural Network (RNN) [17] is a type of Artificial Neural Network, introduced primarily because a feed-forward neural network cannot handle sequential data and considers only current input without memorizing previous input. In NLP, text data are sequential and to understand the context of a sentence better, learning the sequence of word is very important. To achieve this, RNN was introduced as an added recurrent connection. This connection makes the information flow from one step to another, thus handling sequential data. A sequential input data, denoted as x_T, is passed to an RNN

[1] https://github.com/huggingface/neuralcoref

with initial hidden states h_0. The hidden state h_t for a given timestamp t is updated by the input x_t and previous hidden state h_{t-1} which is given by Equation 3.1 .

$$h(t) = (W_x x_t + W_h h_{t-1} + b_h) \tag{3.1}$$

where W_x, W_h and b_h are learned parameters. This type of RNN is also referred to as vanilla RNN and is trained through backpropagation [43]. However, as RNN processes multiple previous steps, it begins to lose previous information. This phenomenon is referred to as the gradient vanishing problem [3]. Long-Short Term Memory (LSTM) [22] was introduced to overcome the weakness of vanilla RNN. LSTM addresses this vanishing gradient problem by using a cell state c_t and gates. Gates are nothing but neural net layers that control the hidden state by deciding which information should flow through the cell. There are three types of gates used in LSTM. Cell state outputs hidden state by taking updates from input.

This can be mathematically explained as, for a given input x_t and previous hidden state h_{t-1}, the forget gate $f(t)$,(Equation 3.2), decides which information needs attention and which can be ignored by outputting values between 0 and 1. 0 represents completely getting rid of information and 1 represents complete flow. Next, the input gate $i(t)$ (Equation 3.3) decides which information to be updated. Both of these gates are combined together along with previous cell state c_{t-1} and new candidate values $\tilde{c}(t)$ to update the current cell state $c(t)$. This is portrayed in (Equation 3.5) which decides which information from the past to be included and which values from the current state is important. The final hidden state h_t (Equation 3.7) is obtained from the new cell state which is controlled by an output gate $o(t)$ (Equation 3.6). The complete flow of this process is given below:

$$f(t) = sigmoid(W_f[h_{t-1}, x_t] + b_f) \tag{3.2}$$

$$i(t) = sigmoid(W_i[h_{t-1}, x_t] + b_i) \tag{3.3}$$

$$\tilde{c}(t) = tanh(W_c[h_{t-1}, x_t] + b_c) \tag{3.4}$$

$$c(t) = f(t)c_{t-1} + i(t) * \tilde{c}(t) \tag{3.5}$$

$$o(t) = sigmoid(W_o[h_{t-1}, x(t)] + b_o) \tag{3.6}$$

$$LSTM(x_t, h_0) = h_t = o(t)tanh(c(t)) \tag{3.7}$$

The final hidden state h_t summarises the whole input sequence which acts as an input to either attention mechanism or any downstream tasks. There are variations to

this LSTM. When the input sequence flows in two directions then the LSTM is referred to as Bi-Directional LSTM(BiLSTM) [12]. In this case there are two hidden states fetched: one for past and one for future. Both are concatenated to get the final hidden state h_t (Equation 3.8). This helps in maintaining both past and future information.

$$BiLSTM(x_t, h_0) = [\overrightarrow{h_t}, \overleftarrow{h_t}] \tag{3.8}$$

Similar to LSTM, Gated Recurrent Unit (GRU) [9] is also a type of RNN which solves vanishing gradients by just using two gates: update and reset gate. In this work, the BiLSTM is used for reasoning questions and GRU is used for predicting answers.

3.2.3 Attention Mechanism

As mentioned in the previous section, the final hidden state h_t of any LSTM summarises the whole input sequence. One of the main drawbacks of these models are, when the sequence is too long, it cannot extract strong contextual relations [1]. For example, if a long text has some context within its substring, models like LSTM cannot identify those substring's context. To overcome this, an attention mechanism was introduced to learn a dynamic context vector c_t. An attention mechanism compares each word in a sentence with one another to find the word to be most attended. Based on this comparison it reweighs the input word embedding, thus forming more contextualised vector [34]. A self-attention mechanism [1] is an attention mechanism that compares every word in the sentence with every other word and with itself. The main reason behind comparing every word to itself is that a word can incur two different meanings based on the context.

Mathematically, a self attention mechanism involves three inputs: query q, key k_t and value v_t. The basic working of a self attention mechanism is given as, for a given encoded input h_t also referred to as keys k_t, and previous decoder state s_t,also referred to as the query q, the model calculates an alignment score u_{qt}. The alignment score u_{qt} for a word is a vector of values which indicates the similarity between the given word and every other word in a sentence and also with itself. The similarity between them is calculated by f_{score} which is a feed forward neural network or dot product (Equation 3.9). The alignment scores are then normalised by a *softmax* function (Equation 3.10).

$$u_{qt} = f_{score}(q, k_t) \tag{3.9}$$

$$\alpha_{qt} = softmax(u_{qt}) \tag{3.10}$$

Then for a given word, its complete alignment score is multiplied with the original value v_t. The final context vector c_t (Equation 3.11) for a word is calculated by summing the multiplied values which gives the weighted sum of vectors.

$$attention(q,k,v) = c_t = \sum \alpha_{qt} v_t \tag{3.11}$$

3.2.4 RotatE

As described in Section 2.2, most of the KGE models fail to capture relational patterns like symmetry, inversion and composition altogether. But, all the above patterns are required for a model to generate better KGE as they explain the structure and properties of a graph better. RotatE [45] can infer all three relational patterns by defining each relation as a rotational model in the complex vector space [45]. RotatE maps subject and object entities with the relation as a rotation. It is one of the few techniques which provide, negative sampling technique to efficiently and effectively train the nodes and edges. It maps each relation e as an element-wise rotation from the subject entity s to object entity o which is given in Equation 3.12.

$$o = s \cdot e \qquad s, o \in C_k \tag{3.12}$$

Here $\cdot$ is the Elementwise product. The basic working of RotatE is as follows. For a given triplet (s,e,o) , RotatE yields the object entity as element-wise product of subject and relation. This method constrains the modulus of each element of e, to be $|e_i| = 1$.The rotation function is the main method that differentiates it from the rest of the models, which is given by the form $r_i \theta e_i$. This form corresponds to counterclockwise rotation by θ and affects only the phases of the entity embedding in the complex vector space, thus defining the rotation function. This is nothing but the scoring function which is defined in Equation 3.13.

$$r(h,t) = RotatE = \mid h \cdot r - t \mid \tag{3.13}$$

3.2.5 Graph Neural Network

Neural Networks for graph data are handled by Graph Neural networks. They represent nodes in lower vector space by using message passing, aggregation, and updating techniques. Each GNN model is different in the way they handle these three functions. Message passing function f_{mp} sends message from node v to u which is nothing but a Multi-Layer Perceptron (MLP) and aggregate function f_{agg} specifies how to aggregate

these messages across nodes and their neighbors. Update function f_{update} updates the node representations based on these aggregates. When a GNN stacks multiple layers to aggregate information from higher-order neighbors it is referred to as k-hop, where k is the number of layers stacked up. Mathematically GNN is represented as shown in (Equation 3.14).

$$h_v^k = f_{update}(h_v^{k-1}, f_{agg}(f_{mp}(h_v^{k-1}, h_u^{k-1}, h_{uv}^{k-1}))) \tag{3.14}$$

Graph Attention Network [48] is an attention based GNN which aggregates the nodes and its neighbours by passing the nodes through an attention mechanism. The aggregated node values are updated based on the attention score and initial values h_v^0. The GAT model from the Deep Graph Library (DGL)[2] is used as the GNN cell in the baseline model of our work (Equation 3.15). DGL is an open source library that carefully handles sparse, irregular, big, and small graphs. It takes advantage of modern hardware with arbitrary message handling and flexible propagation rules.

$$h_v^k = GAT(G, h_v^0) \tag{3.15}$$

The above network is a very simple GNN which fails to capture global semantics, long-range dependency especially, when the KGs are long and complex. In order to overcome that, GraphSAGE was introduced, which is an inductive deep learning method developed by Hamilton, Ying, and Leskovec (2017) [20]. GraphSAGE generates low-dimensional vector representations for nodes. The primary goal of GraphSAGE is to learn node embedding using only a subset of neighboring node features instead of the whole graph. The main idea is to select a few nodes at random from the K-hop neighbors' rather than the entire K-hop neighbor of a target node. This way scalability of the GNN model is ensured. Instead of learning embedding for each node, it learns an aggregate function which includes information about its local neighborhoods [20]. In this way, the global semantics and long-range dependency is maintained. The entire sequence of operations in GraphSAGE 2-hop is given as:

$$h_v^0 = SageCONV(G, W_v, agg, featdrpt) \tag{3.16}$$

$$h_v^0 = dropout(ReLU(h_v^0)) \tag{3.17}$$

$$z_v = SageCONV(G, h_v^0, agg, featdrpt) \tag{3.18}$$

Here z_v is the final node embedding after k hops. The basic working of SageConv in the DGL library for a given graph G is that it initially initializes G to its original feature

[2]https://www.dgl.ai/

vectors W_v. Then at k hop for target node v, it aggregates its k hop neighboring node features into a single vector. An LSTM aggregator would perform an LSTM model on the nodes and their neighbor's hidden state. This process is repeated iteratively for k hops to get z_v and at the end, a nonlinear activation function is applied making it capable to learn and perform more complex tasks.

3.3 Baseline Model

To evaluate the work in this thesis, a baseline model is constructed. The first step in the baseline model is to choose a history selection module. The most common history selection method mentioned in Section 2.4 is to concatenate the previous question-answer pair to the current question. This method is adopted here for the history module. Next, a simple GNN network, Graph Attention Network (GAT) with minimal context features is used as the GNN layer. The primary purpose of creating such a minimal model as a baseline is to evaluate how GNN networks like GAT require more features to capture better word ordering and long-range dependency. And also, the baseline model does not fully follow our proposed-ConvQASARE architecture, whereas it uses a less scalable GNN model with an history selection module which is less efficient. It also does not mimic human cognitive ways of QA. The overall structure of the baseline model is given in Figure 3.1. Each module of baseline model is explained below:

History Selection Module: History in conversational question answering plays an important role. In the baseline model, previous question-answer pairs are joined together. This is used as history $H_k^{(i)}$ in the baseline model (Refer Equation 3.19). This history component is concatenated to the current question (Refer Equation 3.20). As previous question and answers are embedded to a current question, a marker $f_{kqmarker}^{(i)}$ is also created to indicate which turn each question belongs to after concatenation of history (Refer Equation 3.21).

$$H_k^{(i)} = < q > Q^1 < /q >< a > A^1 < /a > ..... < q > Q^i - 1 < /q >< a > A^i - 1 < /a > \quad (3.19)$$

$$Q^{(i)} = [H_k^{(i)}, Q^{(i)}] \quad (3.20)$$

$$f_{kqmarker}^{(i)} = < q > 1, 1, ...1 < /q >< a > 1, 1, ...1 < /a > ..... < q > i, , ..i < /q > \quad (3.21)$$

where Q^i and A^i are the questions and answers at i^{th} turn in a conversation.

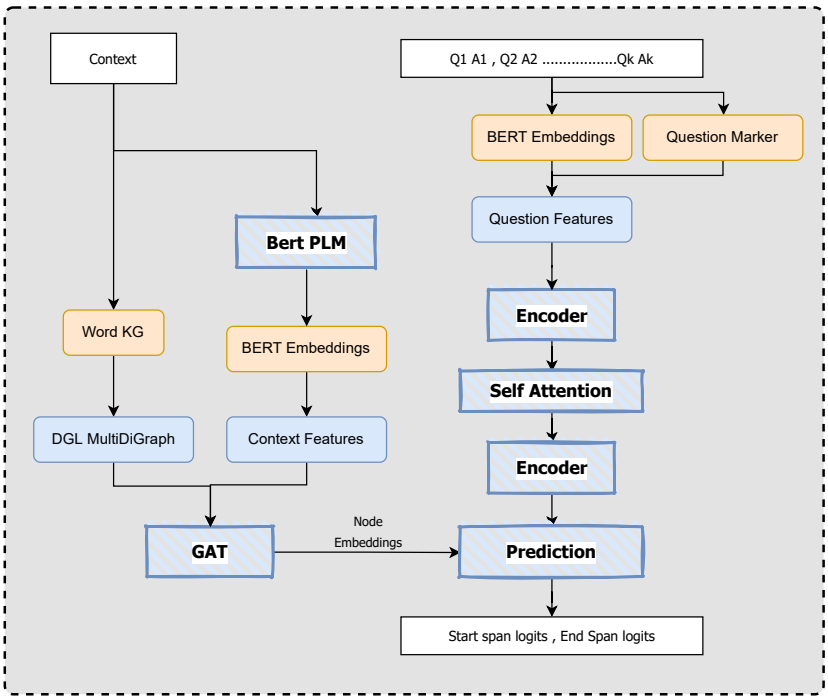

Figure 3.1: Overall structure of the baseline model, with GAT as the GNN layer.

Encoding: The context passage C_j , History H_k and Question Q_k are encoded into input embedding. As mentioned in Section 2.3, the various embedding available were experimented with in this model. The preliminary results showed that BERT performed better than RoBERTa embedding, thus pre-trained word embedding generated using BERT [13] are utilized for both context B_j^C and question B_k^Q. The sliding window approach is used for generating the token's input id from text because BERT base model has a maximum sequence limit of 512. The sliding window approach extracts chunks of 512 lengths from the entire text and yields each chunk to get the input ids and attention mask. These values are passed as an input to the BERT pre-trained model inherited from "BERT-large-uncased". BERT Model is a 1024 embedding model with 24 hidden layers [61]. The layers are averaged to fetch word embedding. One main challenge in using BERT-based embedding is they use byte-level Byte Pair Encoding tokenization [6]. This method splits rarely occurring vocabulary into sub words. For example, "tokenization" is split into *"token"* and *"##ization"* tokens. To map the split tokens to original words, the offset mapping of the tokenizer adds the split hidden layer into one.

Along with BERT Question embedding B_k^Q, $f_{kqmarker}^{(i)}$ which was generated in the history selection module is passed to an Embedding layer to generate question marker

embedding $Q^{(i)}_{kqmark}$ which is done as:

$$Q^{(i)}_{kqmark} = Embedding(f^{(i)}_{kqmarker}) \tag{3.22}$$

Therefore, at the i^{th} turn in a conversation, for each context word C_j, BERT context embedding B^C_j is encoded as context features $W^{(i)}_{cj}$ which is given in Equation 3.23. And for each question word Q_k concatenation of BERT question embedding B^Q_k and question marker embedding $Q^{(i)}_{kqmark}$ is encoded as question features $W^{(i)}_{qk}$ which is given in Equation 3.24.

$$W^{(i)}_{cj} = B^C_j \tag{3.23}$$

$$W^{(i)}_{qk} = [B^Q_k, Q^{(i)}_{kqmark}] \tag{3.24}$$

Reasoning: There are two layers involved in this module- The Question Understanding layer and the Graph Reasoning Layer. In the question understanding layer, a BiLSTM layer is used on question features $W^{(i)}_{qk}$ to get more contextualized question embedding $Q^{(i)}_k$. Then a self-attention mechanism is applied to $Q^{(i)}_k$ to get weighted sum of question word vectors c_k. Finally, an LSTM layer is performed on c_k to capture the dependency among question history. From Equation 3.27, $h(i)$ is the final hidden state, the output of question understanding layer which is used along with graph reasoning output for predicting answers. The entire reasoning process is given as:

$$Q^{(i)}_k = BiLSTM(W^{(i)}_{qk}, h^0) \tag{3.25}$$

$$c_k = attention(Q^{(i)}_k) \tag{3.26}$$

$$h(i) = LSTM(c_k) \tag{3.27}$$

The next layer is the Graph Reasoning Layer. The first step in this layer is the KG construction. As the ConvQA task is treated as a CMRC task, using entity and its relationship as KG results in single entity answers, thus ignoring sequential dependency of words. Therefore to capture the sequential dependency of words, every word in the context is considered as a subject entity while its corresponding neighbor is considered as an object entity. With *"neighbors"* being the relation between them. This KG is referred to as Word KG. Sample example of the WordKG construction from QuAC dataset is given in Figure 3.2. The resulting triplets are transferred into the DGL Multi directional graph (Multi-DiGraph) to be used as an input to the GNN module.

The next step here is the GNN reasoning on the dgl graph. Graph Attention Network (GAT) is the GNN layer used in this model. This network passes messages and

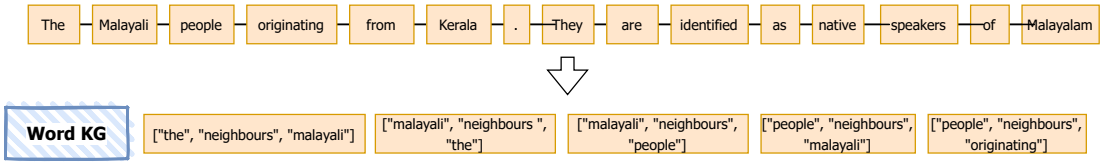

Figure 3.2: Sample example of the Word Knowledge Graph construction from QuAC.

aggregates the nodes and their neighbors on the dgl graph to generate node embedding (Refer Equation 3.28). The context features $W_{cj}^{(i)}$ fetched from Equation 3.23 are passed as node features. The output from GNN reasoning and the question understanding layer is combined to predict answers.

$$z_v = GAT(G, W_{cj}^{(i)}) \tag{3.28}$$

Prediction: The output from the question understanding layer $h^{(i)}$ and graph reasoning layer z_v from the previous module is used together for predicting answers. For j^{th} context word in t^{th} turn in a conversation, the probability of the context word C_j being the start is calculated by matrix multiplying the question output $h^{(i)}$ with transposed graph node embedding z_v. The start probability $start_j$ is given by,

$$start_j = exp(z_v^T W_s h^{(i)}) \tag{3.29}$$

where W_s is a d * d trainable weight and $h^{(i)}$ is the question representation obtained in (Equation 3.27). Next, end probability is obtained in similar way, the start probability was calculated. The only difference is that the question outputs $h^{(i)}$ are passed to a Gated Recurrent Unit cell before matrix multiplication. Then, the end probability end_i is calculated by matrix multiplying the GRU output and the node embedding. This is given by,

$$end_j = exp(z_v^T W_e GRU(h^{(i)})) \tag{3.30}$$

With the start and end probabilities of a context C_j, start and end index of an answer in a context for a given question is calculated by finding the argmax of the probabilities.

3.4 ConvQASARE

3.4.1 Overall Architecture

The proposed baseline model is a simple model that does not fill the gaps mentioned in Chapter 1. The first issue comes from the construction of knowledge graphs. The

errors while constructing KGs accumulate in the later stages resulting in inefficient modeling. This is overcome in our proposal by combining two types of knowledge graphs, Word KG and Entity KG. And also while constructing Entity KG, it is resolved for co-references. The second issue comes from the history component which is not efficient as the conversation increases. This history can extend to a very long sequence resulting in in-efficient training. Therefore, a knowledge graph-based solution using a translational model like RotatE is used due to its simplicity and effectiveness. The third issue directly comes from the GNN layer, GAT, which requires a lot of features to achieve a minimal result and is also inefficient as the size of the graph increases. It does not better capture the long-range dependencies, rich node attributes, and global structure of the graph. At the same time, it is also not scalable and inductive. So the first very straightforward idea comes out, to use GNN methods with an inductive and better scalable approach like GraphSAGE. Meanwhile, the solution should provide end-to-end learning and at the same time should mimic dual process theory. So, the overall structure of the proposed methods, an end-to-end cognitive conversational question answering system using GraphSAGE and RotatE (ConvQASARE) is introduced. The overall architecture is given in Figure 3.3.

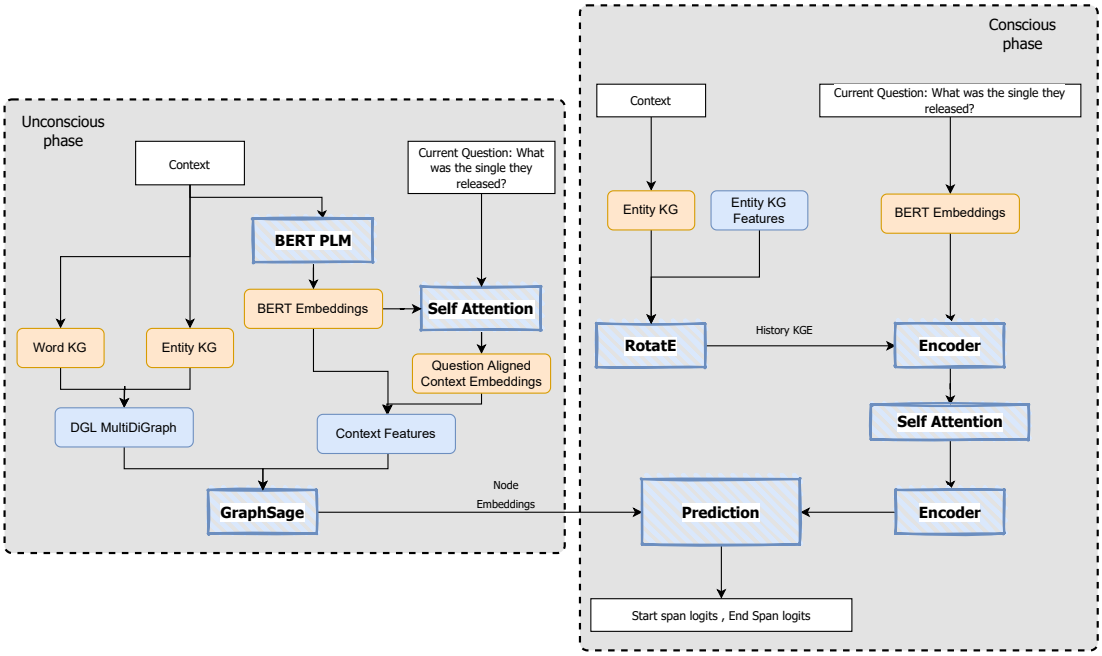

Figure 3.3: Overall Architecture of the proposed ConvQASARE model.

As described in Section 2.6, the architecture also follows a CMRC framework and is resolved mimicking dual process theory in two phases; Unconscious phase and Conscious phase. Both the phases are explained in detail in the upcoming sections.

3.4.2 Unconscious phase

This phase involves dynamic construction of question and conversation history aware node embedding using GraphSAGE. Unlike the baseline model that uses unscalable and less efficient GNN, our model uses GraphSAGE which is inductive and scalable.

Knowledge Graph Construction: The first primary issue overcome by our proposal is the error accumulation due to KG construction. This is overcome in two methods. One, the Word KG created in the baseline is used along with another KG referred to as Entity KG. Entity KG captures the relational facts in a context. In this way, both the relational and sequential dependencies of words are captured. This reduces the error in the construction phase to a limited extent. Second, prior to construction of the Entity KG, context involving co-references are resolved by applying Spacy Neural Co-reference Resolution. This reduces the error accumulation further. The overall structure of Entity KG construction is given in Figure 3.4. Initially, the context words are resolved for its coreferences. The resolved context words are solved for its dependency label using the Spacy dependency parse tree API[3] and the context words with dependency label *"subj"* are resolved as subject entities and *"obj"* are resolved as object entities. Relation objects are obtained based on the root's head of the sentence. Multi relation triplets are generated for a single sentence as each subject can have multiple objects in an English sentence. This type of KG is referred as Entity KG and helps in achieving more contextual KGs. The entity KG is used as an input in a knowledge graph completion task for RotatE and both Word KG and Entity KG are used as an input to the GraphSAGE model. To the best of my knowledge this thesis is the first model which uses both the Word KG and Entity KG in a Conversational QA task.

Context Encoding: Following the baseline model, in addition to the context BERT [13] word embedding B_j^C soft alignment between context words and question words is learnt using attention mechanism. This is referred to as question aligned context embedding. The attention score between context and question word is calculated as :

$$s_{j,k} = exp(ReLU(WB_j^C)^T ReLU(WB_k^Q)) \tag{3.31}$$

where $s_{j,k}$ is the attention score and B_j^C and B_k^Q are BERT embedding of context word C_j and question word Q_k. W is a d * 1024 trainable weight with d being hidden state

[3] https://spacy.io/usage/linguistic-featuresdependency-parse

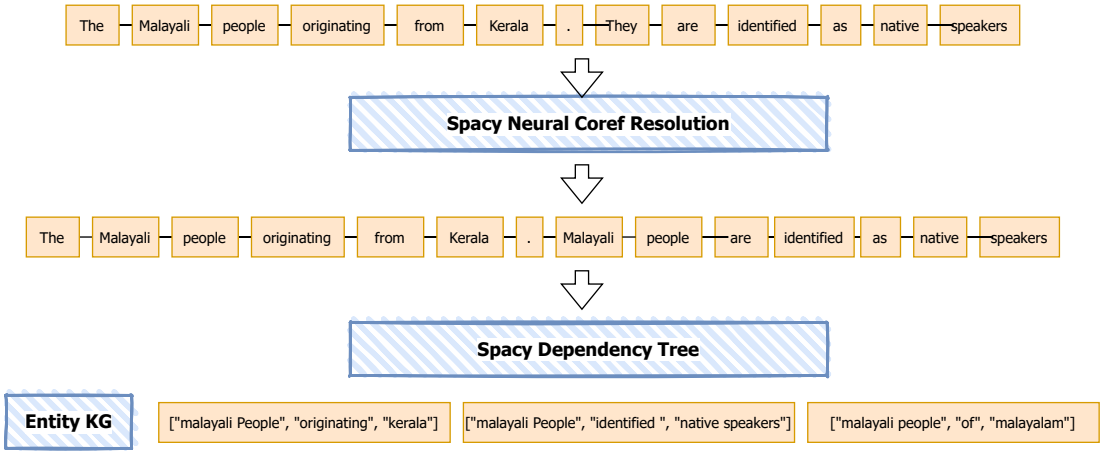

Figure 3.4: Sample example and structure of the Entity Knowledge Graph construction from QuAC.

size. Finally the attention score is multiplied with the BERT question embedding B_k^Q to get the question aligned context embedding (Equation 3.32).

$$F_{align} = \sum s_{j,k} B_k^Q \tag{3.32}$$

Therefore, at the i^{th} turn in a conversation, for each context word C_j both BERT context embedding B_j^C, and question aligned context embedding F_{align} are concatenated to get context features $W_{cj}^{(i)}$ is given by:

$$W_{cj}^{(i)} = [B_j^C, F_{align}] \tag{3.33}$$

Dynamic Node Embedding generation: The baseline model uses a GNN layer which is unscalable and less efficient. This kind of GNN layer fails to capture rich node attributes, long-range depdendency and the overall graph structure. It also requires large number of features to learn them. Thus a better scalable, inductive, and efficient GNN-GraphSAGE is used as our GNN reasoning layer. Entity KG and Word KG are converted into a sequence of context graphs using DGL MultiDiGraph. The DGL graph along with context features $W_{cj}^{(i)}$ fetched above is used as an input to the GraphSAGE module with *'lstm'* aggregator and dropout of 0.3 (Equation 3.34).

$$z_v = GraphSAGE(G, W_{cj}^{(i)}, 'lstm', 0.3) \tag{3.34}$$

Each context graph is processed using single and multi-hop with hops 1, 2, and 3. The GraphSAGE model passes the message to k-hop neighbors in every hop and aggregates the vector as a weighted neural net of all its neighboring node embedding. Later these

node vectors are updated for every node by incorporating the aggregation vectors. The updated node embedding are passed through a ReLU layer to get the final node embedding. This final node embedding is used in predicting answers in the conscious phase.

3.4.3 Conscious phase

This phase aims to jointly reason the contextualized graph node embedding generated in the previous phase along with question and history features. This phase uses the graph-based CMRC task framework to predict the answers.

History Selection Module: The second challenge mentioned in Chapter 1 is that concatenating previous question-answer pairs as history results in in-efficient training. In order to overcome this, a knowledge graph-based solution is used to resolve history. The basic idea is to concatenate the embedding of all the possible entities in a question instead of concatenating the entire previous question-answer pairs. But this process involves identifying the direct mentions of an entity because most of the question involves co references. For better understanding, consider an example, *"What languages are spoken there?"*. The possible entities of this question are *"languages" , "there"*. But *"there"* is not the direct entity but referenced based on any previous mention in the context. Therefore appending an embedding of an indirect entity like *"there"* will result in poor performance of the model. Thus, the primary goal of this KG-based solution is to find the best subject and object entity for a given question $Q^{(i)}$ by utilising knowledge graph completion task. Therefore, at the i^{th} turn in a conversation, for a given question $Q^{(i)}$, a likelihood score for each triplet in Entity KG is calculated using RotatE is given by:

$$kge^{(i)} = RotatE(W_s^{(i)}, W_e^{(i)}, W_o^{(i)}) \tag{3.35}$$

where $W_s^{(i)}, W_o^{(i)} and W_e^{(i)}$ are subject, object and relational embedding from Equation 3.33 and kge is the knowledge graph embedding, the score of all triplets. The maximum score gives the best triplet match for the given question. Thus the index $Hindex$ of the triplet with the maximum score is fetched and the subject and object embedding corresponding to the index are fetched and used as history H_k. The complete process is given as:

$$Hindex = INDEX(argmax(kge^{(i)})) \tag{3.36}$$

$$H_k = [W_s^{(Hindex)}, W_e^{(Hindex)}] \tag{3.37}$$

To the best of my knowledge, this is the first model to use the KGE as history in a ConvQA task.

Question Encoding: The History H_k and question Q_k are transformed into input embedding. The history H_k generated in the previous section is concatenated with the BERT question embedding B_k^Q to get question features which is given in Equation 3.38. This feature is used as an input for the question inference module.

$$W_{qk}^{(i)} = [B_k^Q, H_k] \tag{3.38}$$

Reasoning: As discussed in the baseline model, reasoning involves two layers- Question Understanding and Graph Reasoning. For question understanding layer, the same approach used in the baseline model is used here. The only difference between our model and the baseline model in this layer is that the question word features are changed due to the proposed history selection module. The question $h^{(i)}$ is generated based on the Equation 3.27. Next, the graph reasoning layer is performed in the unconscious phase. The inductive graph node embedding z_v (Equation 3.34) generated in the unconscious phase and the question understanding layer output $h^{(i)}$ are jointly used as input to the next (prediction) module.

Prediction: This module is same as the baseline model's prediction layer without any changes to its formula and mathematical calculations. The input to this module are from the unconscious phase graph node embedding z_v and question output $h^{(i)}$. Two probabilities: start and end are calculated to extract answers from the context words.

Table 3.1 shows the comparison between Baseline and ConvQASARE model. This table shows the equations used in every module of the baseline and ConvQASARE.

Comparison	**Baseline**	**ConvQASARE**
History Selection	$H_k^{(i)}$=<q>Q^1</q><a>A^1..... <q>Q^i-1</q><a>A^i-1	$kge^{(i)}$=$RotatE(W_s^{(i)},W_e^{(i)},W_o^{(i)})$ $Hindex$=$INDEX(argmax(kge^{(i)}))$ H_k=$[W_s^{(Hindex)},W_e^{(Hindex)}]$
Question	$Q^{(i)}$=$[H_k^{(i)},Q^{(i)}]$	$Q^{(i)}$
Question Encoding	$W_{qk}^{(i)}$=$[B_k^Q,Q_{kqmark}^{(i)}]$	$W_{qk}^{(i)}$=$[B_k^Q,H_k]$
Context Encoding	$W_{cj}^{(i)}$=B_j^C	$W_{cj}^{(i)}$=$[B_j^C,F_{align}]$
Question Reasoning	$Q_k^{(i)}$=$BiLSTM(W_{qk}^{(i)},h^0)$ c_k=$attention(Q_k^{(i)})$ $h(i)$=$LSTM(c_k)$	$Q_k^{(i)}$=$BiLSTM(W_{qk}^{(i)},h^0)$ c_k=$attention(Q_k^{(i)})$ $h(i)$=$LSTM(c_k)$
Graph Reasoning	$GAT(G,W_{cj}^{(i)})$	$GraphSAGE(G,W_{cj}^{(i)},'lstm',0.3)$
Prediction	$start_j$=$exp(z_v^T W_s h^{(i)})$ end_j=$exp(z_v^T W_e GRU h^{(i)})$	$start_j$=$exp(z_v^T W_s h^{(i)})$ end_j=$exp(z_v^T W_e GRU h^{(i)})$

Table 3.1: Comparison between equations of Baseline and ConvQASARE used in each module of our thesis

Chapter 4

Experiment

In this chapter, experiments which are designed to evaluate the proposed model is explained in detailed. There are two types of experiments here; conversational question answering task on the QuAC benchmark dataset and investigating our model with the best performing graph-based model on the QuAC benchmark.

4.1 Dataset

The main experiment of this thesis is a Conversational Question Answering task. There are two main benchmark datasets for this purpose : Conversational QA (CoQA) [42] and Question Answering in Context (QuAC) [10]. QuAC is chosen as the benchmark dataset primarily because it has 100K questions from 14K information-seeking dialogs making it more conversational. The main interaction in this dataset is driven by students, where a student asks a question based on the given Wikipedia section and background while the teacher provides a short evidence text as answers from the referenced Wikipedia passage [10].

The entire QuAC dataset is fetched from the hugging face library[1]. A pre-processing step similar to GraphFlow [8] was adopted and applied in this work as the dataset provides only the start character index of the answer feature. But the start and end token index are required to calculate the loss function and evaluate the model. This start and end token index is referred to as the target value in this thesis which is fetched using the offset generated using the StandfordCoreNLP API[2] annotator.

[1] https://huggingface.co/

[2] https://stanfordnlp.github.io/CoreNLP

Table 4.1 shows the data split of QuAC. To evaluate any model, QuAC uses macro averaged F1 score which is the arithmetic mean of all per class F1 scores [55]. There are also two other metrics QuAC uses for evaluation, as there can be multiple answers for a single question [10]. This includes human equivalence score (HEQ) in two variants. When the model exceeds or matches the F1 score, the percentage of questions that achieved this is HEQ-Q and percentage of dialogues that achieved this is HEQ-D. But to get the test metrics, the implemented system is submitted to CodaLab[3] which will evaluate the model on the test set, because the test set is not made available to the public to maintain integrity.

Features	Train	Validation	Test
Dialog	11567	1000	1002
Question	83568	7354	7353
% unanswerable	20.2	20.1	20.1

Table 4.1: Statistics of QuAC dataset for Conversational Question Answering.

4.2 Hyperparameters

For all experiments in this section, the model is trained for 100 epochs with a batch size of 10. The training of the model stops if it does not see any improvement in the validation F1 for more than 10 epochs. Adam optimizer [51] with a learning rate of 0.0001 is used. The learning rate is decreased when the model does not improve its validation performance for 2 epochs. In the question understanding part, 2048 (1024 BERT and 1024 for question aligned context) dimensional word embedding is used for the BiLSTM model. A 2-layer bidirectional LSTM with 1024 hidden units with 0.3 words and RNN dropout is used to avoid overfitting. A two-layer MLP with 1024 hidden units is used as the score function for all attention modules. For the GraphSAGE model, up to 3 layers with 2048-dimensional node embedding are used and 1024-dimensional knowledge graph embedding are generated. The Aggregator function for the GraphSAGE model is “LSTM”. Cross Entropy Loss is used as the loss function between the target’s start, and end values, and the model’s start, and end values. The complete setup can be found in the configuration file of our codebase.

[3]https://worksheets.codalab.org/

4.3 ConvQA Task

This section describes the Conversation Question Answering experiment performed on the QuAC benchmark dataset. Two models - the baseline model, and the ConvQASARE model -proposed in this thesis are used in this experiment. To understand how our proposed GNN and history component affect the performance on the ConvQA task, we unplug our proposed history component from ConvQASARE and use the baseline model's history selection while the GNN layer and the rest of its components remain the same. This is referred to as *"Our Model"* in this thesis. This model is experimented in multiple hops. Next, the best performing model from the above mentioned models is plugged with our KG-based proposed history component. This is the proposed end-to-end ConvQASARE model. Therefore, in this way, three types of model were developed: The Baseline Model, Our Model, and the ConvQASARE model. All the above models were experimented on a 100% QuAC dataset. Each experiment takes up to 15 epochs. Along with these models, GraphFlow [8] model is included as part of the evaluation metric as it is the only graph-based model which exceeds the QuAC benchmark performance. QuAC does not provide the test set to the user to maintain integrity constraints on the test set. In order to get test metrics, the predicted answers are converted to JSON format and is uploaded to a submission portal[4] provided by QuAC, where live test scores were obtained[5]. Table 4.2 reports the train and test set metrics of the models.

Model	Train F1	Test F1	Test HEQ	Test DHEQ
Human	-	81.1	100	100
GraphFlow	-	64.9	**60.3**	5.1
Baseline	66.98	62.4	54.6	5.8
Our Model				
1 Hop	82.51	66.87	60.11	7.70
2 Hop	77.83	66.11	58.85	6.30
3 Hop	74.46	66.07	59.12	7.20
ConvQASARE	75.89	**67.01**	59.3	**8.30**

Table 4.2: Train-Test metrics of models on 100% QuAC,**bold** numeric means best result.

[4]https://worksheets.codalab.org/worksheets/0x06cbafcd9414478b8987955a6b94b2ee

[5]Refer Appendix for the screenshot of live metrics.

On experimenting with the baseline model, the results were up to expectations. It resulted in an F1 score of 62.4 on the test set, the lowest performance of all our models. This clearly shows how few GNN models like GAT, GGN fail to understand better long-range dependency and node attribute information when trained with minimal features. The lower performance shows that the baseline model fails to scale efficiently, thus proving our third hypothesis. As a solution to overcome this, our second model was implemented and evaluated. This model is experimented with single and multiple hops and the single hop model shows 7% increase in F1 compared to the baseline. From Figure 4.1, the results show that the train F1 score does not change much between different hops in every epoch. The model with 2-hop shows better test F1 than the 3-hop model whereas 3-hop shows better test HEQ and DHEQ than 2-hop. However, the single hop model performs slightly better in all the metrics with test F1 score of 66.87. While many researchers adopt multi-hop for graph reasoning, our results were slightly unexpected. A detailed discussion of this phenomenon is discussed in Chapter 5.

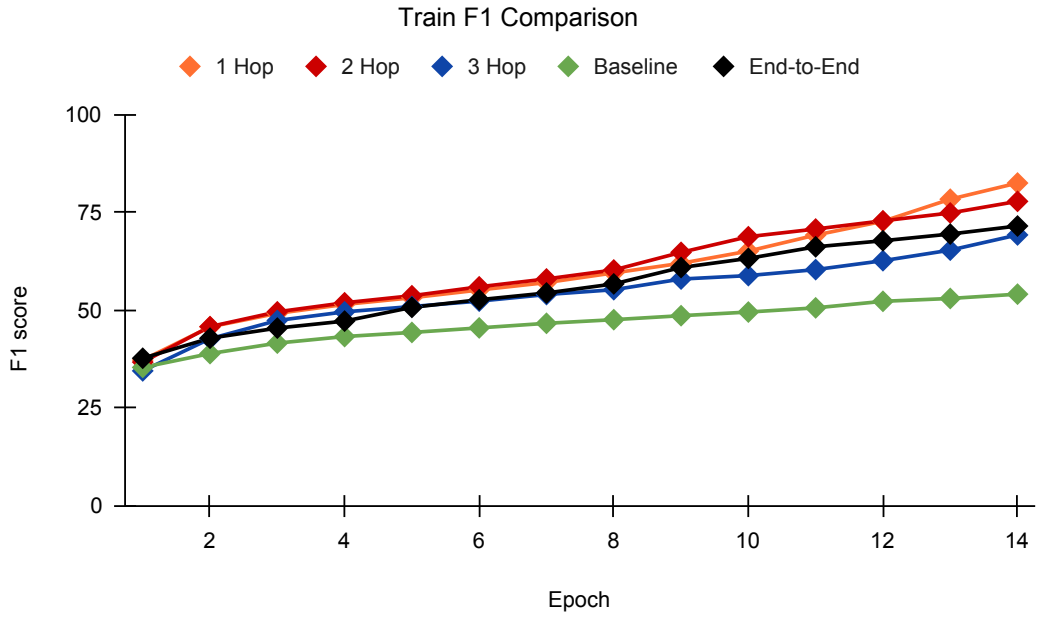

Figure 4.1: Train F1 metrics of all the experiments on behalf of epochs

Even though the history module used so far is effective, they are not efficient. Therefore an end-to-end model with knowledge graph-based history selection was constructed. Previously, the single-hop model performed better with an F1 score of 66.87. Therefore, the selection of history was changed from question-answer pair to a KGC

task on the single-hop model. Based on the results, our ConvQASARE model performed better than the previous models with a test F1 score of 67.01, a 0.2% increase in F1. This shows the efficiency and effectiveness of our history selection method. Even though the test HEQ was lower than our simple single-hop model, the test DHEQ is better in ConvQASARE model. This model also shows better performance, 2.1 F1 better than the only graph-based model, GraphFlow. The better performances prove that ConvQASARE resolves all the gaps discussed in Chapter 1.

Even though our models performs better than GraphFlow, results from Table 4.2 show that there is a huge gap between train and test F1 metrics. This is visualized in Figure 4.2 a,b,c showing the loss comparison between train and validation of our model in multihop. This shows that the model overfits badly resulting in better training and poor validation performance.

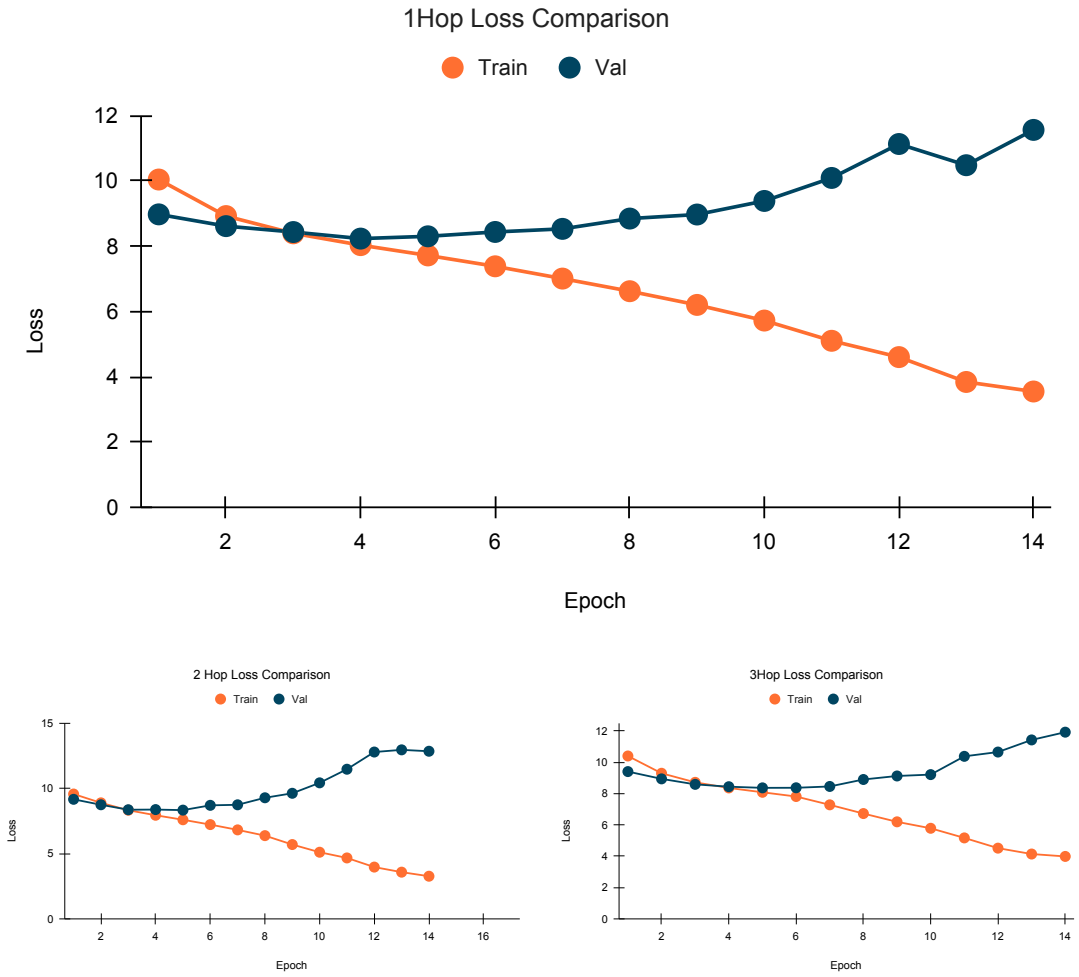

Figure 4.2: a)Top: Loss comparison of single-hop b)Bottom Left: Loss comparison of 2-hop c)Bottom Right: Loss comparison of 3-hop on behalf of epochs

To overcome this overfitting, a small regularisation was applied to the 1-hop model. A weight decay of 0.01 was applied to the ConvQASARE. Due to time constraint, the model was regularised only upto 40 epochs. Unfortunately, the result from 40 epochs showed that this method did not improve the validation performance. Figure 4.3 shows the regularised loss comparison between train and validation. This resulted in a test F1 score of 66.81 which is 0.3 lesser than the model before regularisation. As the regularisation was stopped in 40 epochs, further analysis could not be provided on this experiment. Therefore this is included as part of the future works in this work.

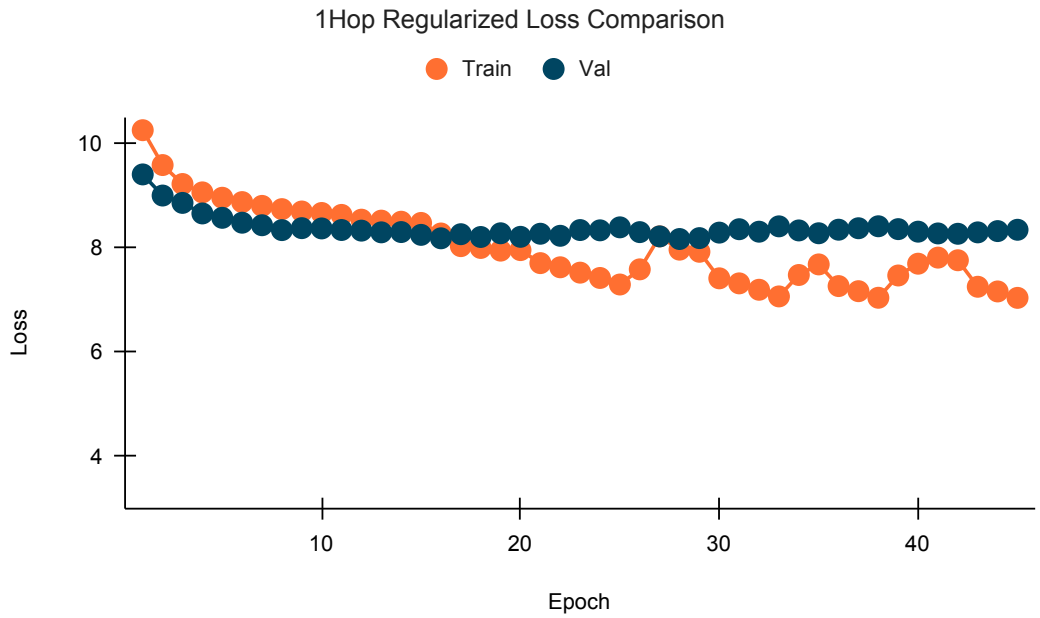

Figure 4.3: Loss comparison of Regularised 1 hop model on behalf of epochs

4.3.1 Summary

The ConvQA experiment conducted on our proposed model, the baseline model proves our third hypothesis, where GNN models like GAT, GGN require large features to achieve a considerable result. Our proposed ConvQASARE achieves better performance of 67.01 test F1 score than any of our other models and GraphFlow. Thus ConvQASARE further indicates the effectiveness and efficiency of our newly proposed, KG-based history selection and GraphSAGE network. Most importantly, better results were fetched by using a few features, a simple history selection module, and a single hop. At the same time, our model also shows the overfitting of the train set. The regularisation method did not improve the test score from its original test score.

4.4 Investigation of ConvQASARE with GraphFlow

The second main experiment in our thesis is to provide an overall comparison between our end-to-end ConvQASARE model and the best performing graph-based model in the QuAC benchmark dataset. Based on the published results[6] of QuAC benchmark dataset on leaderboard, GraphFlow was chosen as it is the only graph model which exceeds the QuAC benchmark performance. Table 4.3 shows various comparisons between the GraphFlow model and our proposed ConvQASARE model. The results of

Comparison	**GraphFlow [8]**	**ConvQASARE**
GNN Method	Recurrent GNN + Gated GNN	GraphSAGE
Graph Size	n^2	$2Xn$
Knowledge Graph	Word KG	Word KG and Entity KG
KG Construction	Dynamic GNN based adjacency matrix	Pattern based DGL MultiDiGraph
Features	POS tags for every token NER tags for every token Glove and BERT Embedding Answer marker, Question Marker, Question Aligned Context , Document Word Count	BERT Embedding, Question Aligned Context , KGE
History Selection	Concatenating previous question and answers	KG Completion
Hops	Multi Hop	Single Hop
F1	64.9	67.01

Table 4.3: Comparison of GraphFlow [8] and our proposed ConvQASARE model.

this comparison show that our model performs better and is more efficient than the GraphFlow primarily because of three main reasons. Firstly, a better scalable and efficient inductive learning GraphSAGE performs better in the ConvQA task than the Recurrent Graph Neural Network along with a Gated Graph Neural network used in GraphFlow. Moreover, our model achieves it by using few features compared to GraphFlow. Our assumption is that the inductive nature and better scalability of GraphSAGE is the primary reason for better performance. Secondly, GraphFlow dynamically constructs a context graph with every word as a node with 'neighbours' as its relation, referred to as Word KG, here. This ensures maintaining sequential dependency among

[6]https://quac.ai/

graph nodes. But it fails to handle relational dependencies among graph nodes. This was resolved in our model by combining Word KG and Entity KG. Thirdly, GraphFlow appends previous questions and answers as a history. However, our model uses efficient and effective history selection module and achieves 2.1 F1 score higher than the GraphFlow.

Along with this, GraphFlow constructs a dynamic adjacency matrix in the context of size n^2, where n = the number of word tokens in a given context. It updates the adjacency matrix on every turn whereas our model construct a dynamic MultiDiGraph of size $2Xn$. This results in efficient training of the model. Therefore this overall comparison experiment shows how our proposed architecture is efficient and results in better performance in the ConvQA tasks, especially on the QuAC benchmark.

4.4.1 Summary

On investigating ConvQASARE with GraphFlow, the third hypothesis where GNN models like GGN require large features to achieve a considerable result is again proved and overcome by our proposal. Our proposed model is efficient in three ways i)Highly scalable and efficient GNN layer ii)Knowledge Graph based history selection is very efficient and easy iii) Our model uses both Word KG and Entity KG as DGL MultiDiGraph of size just $2Xn$.

Chapter 5

Discussions

While researchers claim multi hop to give the best result in a ConvQA task, our experiments resulted in an unexpected outcome where single hop model performed better than any other proposed model. This chapter will discuss this phenomenon in detailed. Furthermore, all our models are said to mimic the human way of question answering. Additionally, a detailed discussion about the cognitive solution is also discussed here.

5.1 Best Single Hop Result

As shown in Chapter 4, single hop model has a good train and test F1 metrics on the dataset. Besides this, the model also beats every other model in all evaluation metrics. However, studies [56], [15] show that QA tasks involving graph reasoning perform better in a multi-hop approach than in a single hop. Generally, 3 hop models yield better results because understanding the long-range dependency and entire global structure of the graph is efficient when the GNN aggregates higher neighbours. This is also proven in CMRC-based QA tasks. Models like [8] and [14] show the best performance only on multi hops. Thus multi-hop is proven to fetch better results. But in our case, this is different. The single hop good performance comes from one very important source, which is the GNN layer GraphSAGE.

Apart from the several mentioned reasons on why our GNN reasoning performs better in this thesis, the single-hop result mainly comes from the GraphSAGE's aggregate function. The aggregate function includes deep neural network methods like *"LSTM"* while most other networks add or average the neighbor information. Another important reason is that the GraphSAGE is said to be permutation invariant to graph isomorphism. Graph isomorphism means that two graph structures are preserved even

after reversing them by an inverse mapping. This feature is very important when it comes to NLP. In NLP, extracting the answers from a long sentence involves understanding the important facts irrespective of sentence order while also capturing the word order within the sentence. As GraphSAGE's aggregate functions are invariant to long graph structures, the permutation invariant training largely improves the performance compared to other properties of the graph. While this assumption validates our single hop results, this does not quantify on multi-hop showing less performance. On further analysis, the aggregate function of GraphSAGE is found to be not injective i.e, distinct node features to different edge elements must generate different node embedding. Due to this feature, the GraphSAGE network starts to generate similar node embedding as the number of layers increases. This explains that the lesser hops showing better performance and increase in layers does not increase the performance of our solution.

5.2 Cognitive Solution for ConvQA

The primary reason to adopt human way of reasoning for question answering is to achieve human performance. One important process of humans during decision making in a conversation is to learn based on their previous mistakes [2]. For instance, when a human reasons out a wrong answer for a question or could not recollect the right answer for a question, they update their memory with the correct answer or add the answer to their memory [59]. This is achieved in dual process theory during the conscious phase of question answering. The knowledge representations i.e, the node embedding generated in the unconscious phase is added with new nodes for every unanswerable question. This solution of adding new nodes is not implemented as part of the ConvQASARE is because QuAC is a dataset which results in an *"unanswerable"* answer when the information is not available in the context passage. But in general, for a conversation question answering task, this would not be the case. Therefore a cognitive solution on a general ConvQA involving node addition would fetch better results. This is included as part of the future works in this work.

Chapter 6

Conclusions

This chapter concludes the works presented in this thesis as a detailed summary. The limitations and the future works of this thesis are also discussed below in detail.

6.1 Summary of this work

In this dissertation, an end-to-end model for Conversational Question Answering that fills the gap between the scalability, combinatorial explosion of graph-based models and high training cost of state-of-the-art PLM based models was proposed. First, a baseline model with simple GNN layer, knowledge graph construction and poor history selection module was constructed. Later, a second model with better scalable and efficient GNN layer but with the same history selection module was built. This model also included simple yet efficient knowledge graph construction to overcome the challenges of the baseline model. Finally, to overcome the poor history selection method which results in inefficient training, an end-to-end model was built on top of the best performing model built so far. To evaluate this proposal on the ConvQA task, two experiments on QuAC dataset was performed. The first experiment was to train and test these models on the QuAC dataset. Our end-to-end single hop model performed better than GraphFlow and all of our proposed models. This shows that permutation invariant training, scalability and better capturing of long-range dependency is important for better performance. In the second experiment, investigating our end-to-end model with GraphFlow, the proposed model shows efficiency and effectiveness of our ConvQASARE model.

6.2 Limitations and Future work

As discussed in the previous chapter, the primary reason for our model's single hop results were injectivity. This can be overcome by implementing new GNN methods or by changing GraphSAGE aggregator function. Moreover, our model perform pretty well on training data but they overfit, resulting in less validation performance and test score. Due to the limitations in time and training resources, more experiments could not be performed to mitigate this issue. Besides, for the same reason, this work is experimented only on single ConvQA benchmark, QuAC. This work was not evaluated on other ConvQA benchmarks like CoQA. Moreover, our model does not achieve state-of-the-art performance which is 9 points behind our F1 score. So these works are considered as the limitations of this thesis.

To fill the limitations of this work here are potential future works that will benefit the AI research field:

- Research on more suitable GNN methods which improves the injectivity of GraphSAGE and implement this on the proposed framework to check how it affects the performance of ConvQA task.
- Completely run the regularisation technique on our model or implement graph augmentation techniques to reduce overfitting of the model.
- Evaluation of our end-to-end model on other Conversational Question Answering benchmarks like CoQA. As our proposed model follows CMRC framework, evaluation can be performed on more downstream tasks like text summarisation, text generation.
- Apart from overcoming the limitations of our thesis, the proposed history selection method has never been applied to any other NLP downstream task. This type of history selection can be evaluated on other tasks like Entity Resolution.

Bibliography

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. volume 1409, 09 2014.

[2] Soham Banerjee, Pradeep Kumar Singh, and Jaya Bajpai. A comparative study on decision-making capability between human and artificial intelligence. In Bijaya Ketan Panigrahi, M. N. Hoda, Vinod Sharma, and Shivendra Goel, editors, *Nature Inspired Computing*, pages 203–210, Singapore, 2018. Springer Singapore.

[3] Francesco Bergadano and Daniele Gunetti. *Bibliography*. 1995.

[4] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, page 1247–1250, New York, NY, USA, 2008. Association for Computing Machinery.

[5] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[6] Kaj Bostrom and Greg Durrett. Byte pair encoding is suboptimal for language model pretraining. pages 4617–4624, 01 2020.

[7] Y. Chen, M.J. Zaki, and Rensselaer Polytechnic Institute. Dept. of Computer Science. *Question Answering and Generation from Structured and Unstructured Data*. Rensselaer Polytechnic Institute, 2020.

[8] Yu Chen, Lingfei Wu, and Mohammed Zaki. Graphflow: Exploiting conversation flow with graph neural networks for conversational machine comprehension. 07 2019.

[9] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv, 2014.

[10] Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wen-tau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. QuAC: Question answering in context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2174–2184, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[11] Nicola Colic and Fabio Rinaldi. Improving spacy dependency annotation and pos tagging web service using independent ner services. volume 17, page e21, 06 2019.

[12] Zhiyong Cui, Ruimin Ke, Ziyuan Pu, and Yinhai Wang. Deep bidirectional and unidirectional lstm recurrent neural network for network-wide traffic speed prediction. arXiv, 2018.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[14] Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. Cognitive graph for multi-hop reading comprehension at scale. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2694–2703, Florence, Italy, July 2019. Association for Computational Linguistics.

[15] Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. Cognitive graph for multi-hop reading comprehension at scale. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2694–2703, Florence, Italy, July 2019. Association for Computational Linguistics.

[16] Zhengxiao Du, Chang Zhou, Ming Ding, Hongxia Yang, and Jie Tang. Cognitive knowledge graph reasoning for one-shot relational learning. arXiv, 2019.

[17] Jeffrey L. Elman. Finding structure in time. volume 14, pages 179–211, 1990.

[18] Somil Gupta, Bhanu Pratap Singh Rawat, and Hong Yu. Conversational machine comprehension: a literature review. In *Proceedings of the 28th International Conference on Computational Linguistics*. International Committee on Computational Linguistics, 2020.

[19] Dilek Hakkani-Tur, Celikyilmaz Asli, Larry Heck, G. Tur, and G. Zweig. Probabilistic enrichment of knowledge graph entities for relation detection in conversational understanding. pages 2113–2117, 01 2014.

[20] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[21] Yanchao Hao, Yuanzhe Zhang, Kang Liu, Shizhu He, Zhanyi Liu, Hua Wu, and Jun Zhao. An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 221–231, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. volume 9, pages 1735–1780, 11 1997.

[23] Valentin Hofmann, Janet Pierrehumbert, and Hinrich Schütze. Dynamic contextualized word embeddings. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6970–6984, Online, August 2021. Association for Computational Linguistics.

[24] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. arXiv, 2020.

[25] Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. Knowledge graph embedding based question answering. WSDM '19, page 105–113, New York, NY, USA, 2019. Association for Computing Machinery.

[26] Shaoling Jing, Shibo Hong, Dongyan Zhao, Haihua Xie, and Zhi Tang. Combining impression feature representation for multi-turn conversational question answering. In *Proceedings of the 19th Chinese National Conference on Computational Linguistics*, pages 863–873, Haikou, China, October 2020. Chinese Information Processing Society of China.

[27] Cheng Wenfang He Shuyun Jiang Hui JOUR, Miao Yalin. Research on visual question answering based on gat relational reasoning. Neural Processing Letters, 04 2022.

[28] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. arXiv, 2016.

[29] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. volume 5. Springer Science and Business Media LLC, jan 2020.

[30] Thao Minh Le, Vuong Le, Svetha Venkatesh, and Truyen Tran. Neural reasoning, fast and slow, for video question answering. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[31] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. arXiv, 2020.

[32] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. arXiv, 2015.

[33] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 2181–2187. AAAI Press, 2015.

[34] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. arXiv, 2017.

[35] Ziqing Liu, Enwei Peng, Shixing Yan, Guozheng Li, and Tianyong Hao. T-know: a knowledge graph-based question answering and infor-mation retrieval system for traditional Chinese medicine. In *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, pages 15–19, Santa Fe, New Mexico, August 2018. Association for Computational Linguistics.

[36] Edward Loper and Steven Bird. Nltk: the natural language toolkit. volume cs.CL/0205028, 07 2002.

[37] Tomas Mikolov, Kai Chen, G.s Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. volume 2013, 01 2013.

[38] Armin Oliya, Amir Saffari, Priyanka Sen, and Tom Ayoola. End-to-end entity resolution and question answering using differentiable knowledge graphs. 09 2021.

[39] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[40] Truong Phan and Phuc Do. Developing a bert based triple classification model using knowledge graph embedding for question answering system. volume 52, 01 2022.

[41] Wei Qian, Cong Fu, Yu Zhu, Deng Cai, and Xiaofei He. Translating embeddings for knowledge graph completion with relation attention mechanism. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, page 4286–4292. AAAI Press, 2018.

[42] Siva Reddy, Danqi Chen, and Christopher D. Manning. CoQA: A conversational question answering challenge. volume 7, pages 249–266, Cambridge, MA, 2019. MIT Press.

[43] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*. 1987.

[44] Wissam Siblini, Baris Sayil, and Yacine Kessaci. Towards a more robust evaluation for conversational question answering. pages 1028–1034, 01 2021.

[45] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2019.

[46] Yi Tay, Anh Luu, Siu Hui, and Falk Brauer. Random semantic tensor ensemble for scalable knowledge graph link prediction. pages 751–760, 02 2017.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. arXiv, 2017.

[48] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. arXiv, 2017.

[49] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, page 1112–1119. AAAI Press, 2014.

[50] Jiacheng Xu, Kan Chen, Xipeng Qiu, and Xuanjing Huang. Knowledge graph representation with jointly structural and textual encoding. arXiv, 2016.

[51] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[52] Weiwen Xu, Bowei Zou, Wai Lam, and Ai Ti Aw. Improving lexical embeddings for robust question answering. 2 2022.

[53] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. arXiv, 2014.

[54] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc Le. Xlnet: Generalized autoregressive pretraining for language understanding. 06 2019.

[55] Mark Yatskar. A qualitative comparison of CoQA, SQuAD 2.0 and QuAC. 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), pages 2318–2323, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[56] Jiayuan Zhang, Yifei Cai, Qian Zhang, Zehao Cao, Zhenrong Cheng, Dongmei Li, and Xianghao Meng. Multi-hop reasoning for question answering with knowledge graph. In *2021 IEEE/ACIS 19th International Conference on Computer and Information Science (ICIS)*, pages 121–125, 2021.

[57] Rui Zhang, Cícero Nogueira dos Santos, Michihiro Yasunaga, Bing Xiang, and Dragomir Radev. Neural coreference resolution with deep biaffine attention by joint mention detection and mention clustering. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 102–107, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[58] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J. Smola, and Le Song. Variational reasoning for question answering with knowledge graph. arXiv, 2017.

[59] Hong Zhao, Yao Fu, Weihao Jiang, Shiliang Pu, and Xiaoyu Cai. Simulate human thinking: Cognitive knowledge graph reasoning for complex question answering. In *Advances in Knowledge Discovery and Data Mining: 26th Pacific-Asia Conference, PAKDD 2022, Chengdu, China, May 16–19, 2022, Proceedings, Part I*, page 522–534, Berlin, Heidelberg, 2022. Springer-Verlag.

[60] Fengbin Zhu, Wenqiang Lei, Chao Wang, Jianming Zheng, Soujanya Poria, and Tat-Seng Chua. Retrieving and reading: A comprehensive survey on open-domain question answering. 01 2021.

[61] Liu Zhuang, Lin Wayne, Shi Ya, and Zhao Jun. A robustly optimized BERT pre-training approach with post-training. In *Proceedings of the 20th Chinese National Conference on Computational Linguistics*, pages 1218–1227, Huhhot, China, August 2021. Chinese Information Processing Society of China.

Appendix A

Result Screenshot

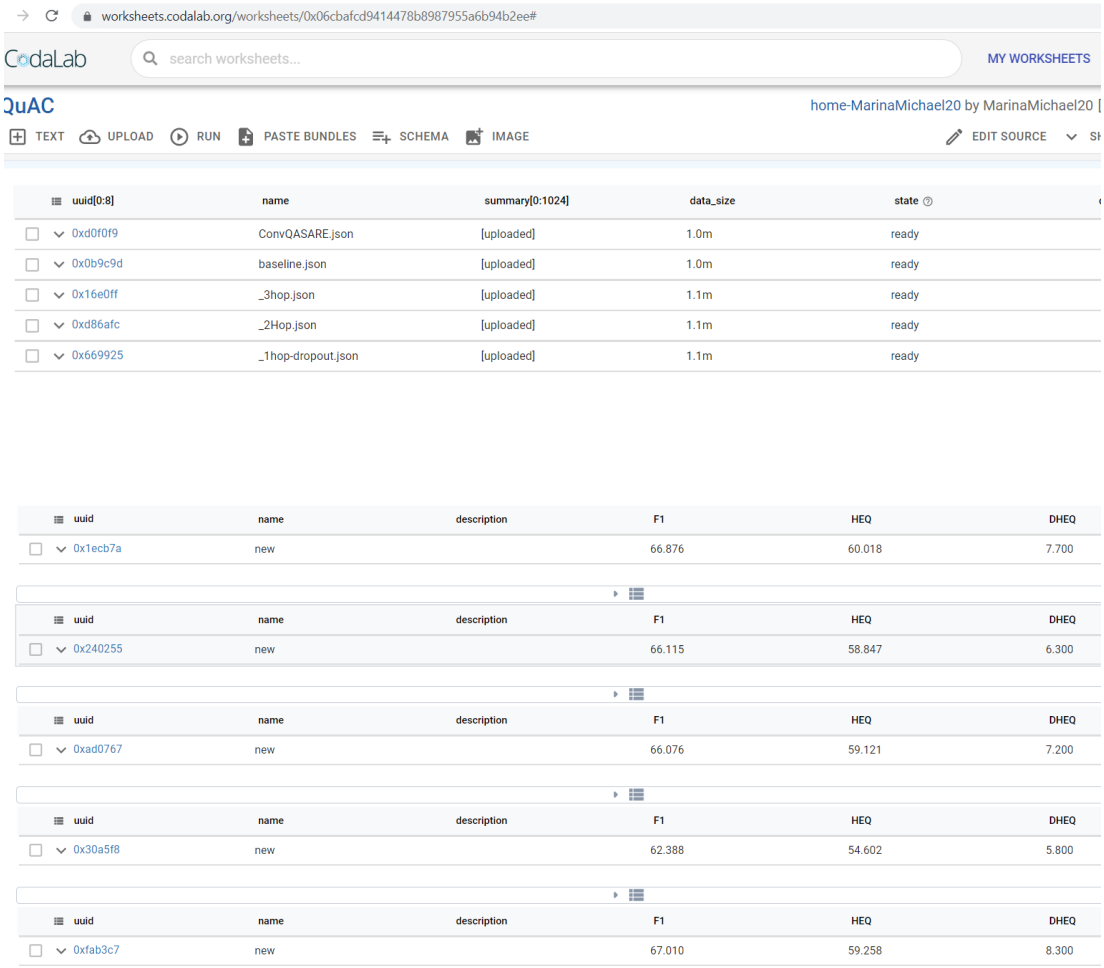

Figure A.1: Screenshot of results of ConvQA experiment on QuAC benchmark dataset. This is a screenshot from the live url https://worksheets.codalab.org/worksheets/0x06cbafcd9414478b8987955a6b94b2ee

The above images are the screenshot of the ConvQA experiment metrics on QuAC dataset. The first part of the image shows the list of models uploaded and the second part of the screenshot shows the corresponding scores. The scores in the second part of the image is read in the reverse order. For example, the F1 score *66.87* corresponds to the model *"_1hop-dropout.json"*.