Can Incremental Learning help with KG Completion?

Mayar Osama
Mervat Abu-Elkheir

Faculty of Media Engineering and Technology, German University in Cairo, Egypt

Abstract. Knowledge Graphs (KGs) are a type of knowledge representation that gained a lot of attention due to their ability to store information in a structured format. This structure representation makes KGs naturally suited for search engines and NLP tasks like question-answering (QA) and task-oriented systems; however, KGs are hard to construct. While QA datasets are more available and easier to construct, they lack structural representation. This availability of QA datasets made them a rich resource for machine learning models, but these models benefit from the implicit structure in such datasets. We propose a framework to make this structure more pronounced and extract KG from QA datasets in an end-to-end manner, allowing the system to learn new knowledge in incremental learning with a human-in-the-loop (HITL) when needed. We test our framework using the SQuAD dataset and our incremental learning approach with two datasets, YAGO3-10 and FB15K237, both of which show promising results.

Keywords: Knowledge Graphs, Question Answering, Incremental Learning, Human in the loop

1 Introduction

Task-oriented dialogue systems have been a big part of our modern life and an active area for research and industry. They aim to chat with the user to understand their needs and achieve a specific task for them, and this could be an actual task like setting the alarm, answering a question, or recommending something for the user. These systems could work as multi-domain systems, covering more than one domain, or closed-domain systems, covering only a single domain. They could achieve the goal by conversing with the user over multiple or single turns.

Task-oriented dialogue systems, in general, either follow a modular pipeline approach consisting of four main modules; the first one is a Natural Language Understanding (NLU) module: which is responsible for extracting the information from the user utterance into the knowledge format the model understands, the second one is a Dialogue State Tracking (DST) module: that is responsible for updating the current state of the dialogue, the third module is the Dialogue Policy (DP): which decides for the following action based on the current state, and the last one is a Natural Language Generator (NLG): that takes action decided upon and generates the response to the user in the form of natural language. Sometimes, the DST and DP are referred to as the Dialogue Manager. The other approach is for the system to work end-to-end.[1]

How the knowledge is represented and understood affects the architecture and the techniques used to reach the objective goal. Consequently, many approaches were proposed in this area, which motivated the construction of many datasets. The knowledge representation of the training dataset affects the techniques used to process and extract the needed information. Some of the most common data representations are:

- Questions and Answers pairs; are the most accessible form of knowledge to construct a dataset, as we only need to collect the previous logs of conversations when it's available.
- Documents; in which the knowledge is stored in paragraphs containing the information needed about the system.

David C. Wyld et al. (Eds): CCSEA, AIFU, EMSA, NLCAI, NCOM, SIPRO, SEA, DKMP, BDML, BIoT, CLOUD - 2023
pp. 103-122, 2023. CS & IT - CSCP 2023 DOI: 10.5121/csit.2023.130510

- Intents and slots values representation is a catalog-like representation to store the information, where the intent detected from the user's query is used to fill its corresponding slot. The slots are the labels each word token has in the text.
- Knowledge Graphs, at which the knowledge is represented as facts, where each fact is a triplet *(h, r, t)* where the head entity h is connected to the tail entity t by relation r.

The most expensive and time-consuming forms of knowledge representation are knowledge graphs or intents and slot values; although they provide the best results since they are handcrafted, especially in closed domain systems, they require experts to tailor the knowledge with the required format. In comparison, having the knowledge represented as pairs of questions and answers is easier to be collected from existing systems' logs. Same for having multi-paragraph documents, which are easier to construct.

Constructing a new dataset is costly and time-consuming. Hence, we propose the integration of existing datasets and trying to represent the same knowledge with different representation forms. This paper focuses on extracting knowledge from question-answer pairs and constructing a graph containing the equivalent knowledge.

Knowledge graphs (KGs) represent the data formally as a set of facts; a fact triplet represents the relation between two entities, so these entities could be viewed as the nodes of the graph and the relations to be the edges connecting them. A fact is also referred to as a triplet (h, r, t), where the head entity h is connected to the tail entity t by relation r. Thanks to their structural representation, KGs are used in many NLP task-oriented tasks, e.g., information retrieval, search agents, question answering, conversational recommender systems, etc...

As for every knowledge representation, KGs have limitations beyond being more expensive to construct. The main task of extracting knowledge from KGs is a link prediction task. Link prediction is the task of predicting the missing entity in a given query depending on whether it's a *(h,r,?)* tail prediction or a *(?,r,t)* head prediction, the model should predict the missing entity to complete the required fact. This link prediction could be done straightforwardly or require multiple hops between a few facts in the KG until we reach the missing entity. The issue of not being able to retrieve the missing entity right away motivated another active task in the field of KGs, which is the task of reasoning over KGs. Reasoning over KGs aims to obtain new facts from existing knowledge. Chen et al. [2] reviewed over 11 different approaches for reasoning over KGs, which is an active task for KGs because KGs suffer the limitation of incompleteness.

The incompleteness issue exists in most datasets as it's hard to capture all the needed knowledge of a given topic, especially when designing an open-domain system; it's almost impossible to cover everything when creating the knowledge base at the beginning. Although reasoning over KGs helps reduce that gap, all the data to be added would be at some point, and no new knowledge could be added. Hence, a dynamic system is needed to add new knowledge when needed. We address this issue by having a human in the loop.

A human in the loop is used in different fields to achieve different tasks [3,4,5,6]. The idea behind it is to deploy the model and let it work as it should, and when the model needs new knowledge or makes a mistake, the human/expert could interfere and update the model with the needed knowledge. In our approach, we achieve this with an incremental learning module. But having a human in the loop pops out an important question, does this contradict the main aim of artificial intelligence and machine learning to automate a given task and remove the human from the loop? And the answer is it's a trade-off. For a particular closed-domain system, it is very doable to construct a complete knowledge base from the beginning that the model could be trained on and answer any given ques-

tion accordingly. But in open-domain systems, constructing a complete knowledge base is almost impossible. And assuming there was a way to collect all the relevant knowledge from multiple sources would require intensive training, which would be very expensive. It might also be that not all the knowledge gathered is needed, which might backfire and affect the model's performance. Hence, having a human in the loop only when needed is an acceptable middle ground in our approach, as we start by extracting the knowledge in an end-to-end manner and only refer to the human in the loop when needed to reduce the gap between the knowledge collected. Eventually, the dependency on the human in the loop decreases till it's not needed anymore.

This paper aims to address the following three issues; 1) Constructing KGs from question-answering datasets in an end-to-end manner, 2) Having a dynamic knowledge base system to learn new facts and allow the model to learn new entities, and 3) Reducing the gap of having 1:n or n:1 relations in the KG for existing KG embedding models.

2 Background and Related Work

2.1 Knowledge Graph Embedding and Link Prediction

KGs could be formally represented as a set of facts, such that a fact consists of a triplet *(e_1, r, e_2)* or *(head, relation, tail)*; i.e. the subject e_1 is connected to the object e_2 through this relation r, where e_1 and e_2 belong to the set of possible entities and r belongs to the set of possible relations. *KG=*$\{$*(e1,r,e2)*$\|$ *e1,e2* $\in \boldsymbol{E}$ *and* $r \in \boldsymbol{R}\}$

Link prediction is the task of predicting the missing entity given a source entity and a relation. It could be a head prediction where the source entity is the tail entity, and the missing entity is the head *(?,r,t)*, or a tail prediction where the source entity is the head, and the missing entity is the tail *(h,r,?)*.

Many approaches have been proposed to achieve this link prediction task, some focused on observable features such as Rule Mining [17][16][38][24] or the Path Ranking Algorithm [31][32], and others focused on capturing latent features of the graph by using different embedding techniques. In our paper, we are mainly focusing on the KG embedding approaches.

In general, how the task of link prediction works with KG embedding models is by defining a scoring function ϕ that indicates the probability of the given fact being true. As shown in Equation 1, for a given tail prediction, the model should output the entity e, which returns the highest score from the scoring function.

$$t = \arg\max_{e \in \boldsymbol{E}} \ \phi(h, r, e) \tag{1}$$

Rossi et al.[7] provided a very useful comparative analysis for many of these link prediction approaches for KGs. They classified these models into three main categories:

1. Tensor Decomposition Models, where the task of LP is considered a tensor decomposition task, as these models process the KG as a 3D adjacency matrix or a 3-way tensor that is only partially observable due to the KG incompleteness. This tensor is then decomposed into low-dimensional vectors, which are used as the embeddings for entities and relations.
   (a) Bilinear Models: Given a head embedding $h \in R^d$ and a tail embedding $t \in R^d$, these models usually represent the relation embedding as a bi-dimensional matrix $r \in R^{d \times d}$, where the scoring function computes the result of the product of the three matrices $\phi(h, r, t) = h \times r \times t$. One of the most commonly used models of this class is ComplEX[8].

   (b) Non-Bilinear Models combine the head, relation, and tail embeddings of composition with approaches different from the strictly bilinear product. e.g., HolE[9] computes the circular correlation between the embeddings of head and tail entities and then applies the matrix multiplication with the relation embedding.

2. Geometric Models, on the other hand, view relations as geometric transformations in the latent space where the fact score is represented as the distance between a resulting vector of processing the head and the relation and the tail vector.
   (a) Pure Translational Models represent entities and relations as one-dimensional vectors of the same length, where the added distance between the head embedding and relation embedding should result in the position closest to the tail embedding. TransE[10] was the first proposed model using a pure translational approach, and due to this nature in calculating the score, TransE cannot correctly handle one-to-many and many-to-one relations, as well as symmetric and transitive relations.
   (b) Translational models with Additional Embeddings may associate more than one embedding to each KG element. For instance, CrossE[11] is considered one of the best models of this class. CrossE learns an additional relation-specific embedding with each relation c_r that is then combined with the head and the relation to be used in the translation.
   (c) Roto-Translational Models that perform rotation-like transformations either in combination or in alternative to translations, e.g., RotatE[12] represents relations as rotations in a complex latent space.

3. Deep Learning Models use deep neural network layers to extract the features from the input by fine-tuning the weights and the biases of the neurons of these layers along with learning the KG embeddings.
   (a) Convolutional Neural Networks may contain one or more convolutional layer(s). The task of those layers is to loop over the input with convolution techniques by applying low-dimensional filters to allow the model to extract the needed features during the training phase. Then a Dense layer is used to process the output of the convolution to get the score of a given fact. Examples of CNN models for KG embeddings are ConvE[13], ConvKB[14], and ConvR[15].
   (b) Capsule Neural Networks consist of capsules that are composed of groups of neurons that aim to encode specific features of the input. The main difference is that capsules allow the model to encode those features without losing spatial information, unlike convolutional networks. e.g. CapsE[16].
   (c) Recurrent Neural Networks (RNNs), which consist of recurrent layers that are known for their abilities to process and encode sequential data, e.g., RSN[17].

2.2 Knowledge Graph Construction

The task of constructing a knowledge graph is usually done by experts to ensure the right format and cover the correct information, which is a very time-consuming and expensive process. Hence, many approaches have been proposed to try and automate this process.

Some approaches focused on the potential of language models, as they get to learn linguistic knowledge during training and their ability to store relational knowledge between the training data. Language models are known to have the ability to implicitly encode massive amounts of knowledge, to be used for different tasks like question-answering, text summarizing, etc. One of the main advantages when using language models is that they don't require a fixed schema or human annotations, which allows them to support open-domain questions and the ability to extend to more data.

Language models could answer queries that are structured as "fill in the blank" cloze statements because of their masking mechanism during training, at which the model is required to learn to fill the word at the masked position with the correct word. Petroni et al[18] provided an analysis for language models to test their ability on factual and commonsense knowledge, where the facts they used were either triplet subject-relation-object or question-answer pairs. Each fact is converted to a cloze statement which is then queried to the language model for a missing token and accordingly evaluated that model. They used Google-RE[1], T-rex[19], ConceptNet[20], and SQuAD[21] datasets.

Their results showed the potential of BERT, as it performed well on open-domain questions, and results showed that it contains relational knowledge that could be competitive with traditional NLP methods. Traditional NLP methods are known to contain the best information extractions on fixed schema, which is not always available. Also, they work in a complex pipeline to achieve entity extraction, coreference resolution, entity linking, and relation extractions. The pipeline architecture makes them vulnerable to error propagation and accumulation.

Another approach that took advantage of language models in KG construction is MAMA[22]. They argued that language models are considered an open knowledge graph, as having a language model and a textual corpus, they could generate a knowledge graph relevant to that corpora through a two-staged architecture; Match and Map. At the match phase, the model generates a set of candidate facts from the corpus using Beam Search, where the goal is to match the knowledge stored in the pre-trained language model with the facts in the corpus. During the map phase, the generated facts are mapped to a fixed and open schema to generate the final knowledge graph. MAMA was not the only proposed approach to generate KGs using language models. Swamy et al.[23] followed the same property in language models to answer "fill in the blank queries" cloze statements as LAMA[18]. They first used this property to extract all statements that contain relevant knowledge using the masking property, then added an extraction step using a hybrid SpaCy[2] and Textacy[3] approach to extract the relevant triplets and construct the KG, illustrated in Figure 1. Since their approach works in an unsupervised end-to-end manner, it was vulnerable to inconsistency, as the output KG would depend on the structure of the statements in the corpus and the generated statements after the cloze querying to the language model.

BertNet[24] tried to address this issue along with the dependency of having existing massive data to learn from. They proposed to apply a paraphrasing stage before extracting the triplets, that way, there would be a more diverse set of alternatives to generate the entities and the triplets from. And to handle the resulting issue of having a large search space after the paraphrasing, they proposed a search and scoring strategy to balance the accuracy and coverage of the output.

Garg et al.[25] decided to go in a different direction and examine whether or not language models could capture graph semantics and if language models and graphs could work interchangeably. The objective was if a language model took a graph as an input can it output the same graph while maintaining the same semantics? They pointed out that due to the nature of language models, they take their input as a form of distributed representations or vectors, and to pass the graph as an input to a language model; it needs

[1] https://code.google.com/archive/p/relation-extraction-corpus/

[2] SpaCy is a free, open-source library for NLP in Python. It's written in Cython and is designed to build information extraction or natural language understanding systems

[3] Textacy is a Python library for performing a variety of natural language processing (NLP) tasks, built on the high-performance spaCy library. With the fundamentals — tokenization, part-of-speech tagging, dependency parsing, etc.

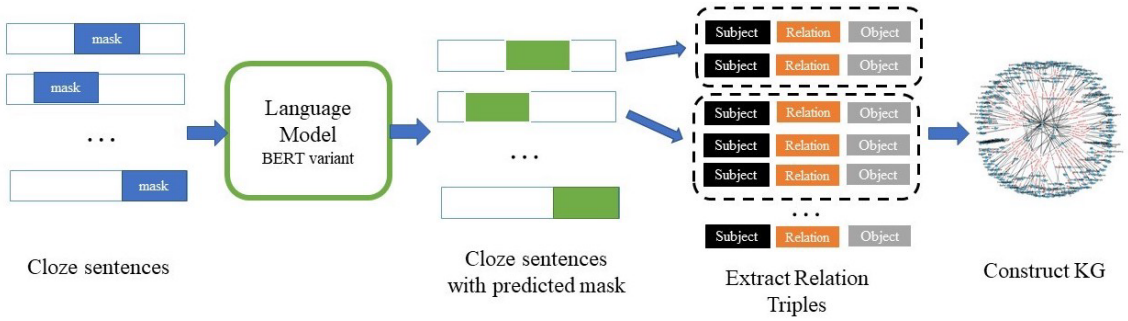

Fig. 1. Pipeline to generate knowledge graphs from language models from [23]

to be compressed into a vector representation which would affect its structural semantics. From this point, the experiments showed that transformer models could not express the full semantics of the input knowledge graph.

Language models have shown so much potential when it comes to NLP tasks, and they are viewed as more than just processing models but potentially a pre-trained knowledge base. However, they store the knowledge in a form of a black box which still makes it difficult to access or edit. AlKhamissi et al.[26] provided a full analysis of language models and whether or not they could be considered a walking database.

2.3 Incremental Learning

The aim of machine learning is to allow the system to behave as a human, and humans keep updating their knowledge either by trial and error, which is represented in the training phase, or by external knowledge and asking for the right answer, which is represented in the incremental learning phase of our approach.

Incremental learning aims to keep the system updated and gives it the ability to learn new knowledge without having to retrain it on all the previous knowledge. There are three main ways to apply incremental learning; the first one is to have an interactive environment in which the system can get feedback and change its behavior accordingly, the second approach is to allow the system to get its knowledge from an external source which in our scope is shown with the human/expert in the loop, and the third is using both approaches together. The concept of incremental learning has been adapted to different domains, and fields [3,4,5,6].

Wang, Weikang, et al. followed the idea of building a training dataset consisting of five sub-datasets (SubD1, SubD2, SubD3, SubD4, and SubD5). The model is first trained on these sub-datasets one at a time till it is ready for deployment. But instead of using a traditional task-oriented dialogue system, they proposed a very interesting approach which is the Incremental Dialogue System (IDS)[27].

IDS uses the concept of adding a human-in-the-loop; the motivation of this approach is to reduce the non-relevant responses of the dialogue system. One of the main issues in dialogue systems is the irrelevant replies; since the model is usually trained on certain dialogues, it usually remembers "most" of the replies, but in many cases, if the user asks a new question that the system didn't see before then the response would be irrelevant. To fix that, first, they calculate the confidence level between the model's reply and the user's query; if it's high, then respond with the generated system's reply, but if it has a low confidence level meaning the reply is irrelevant, then let a human expert reply this time to the human. After asking the human in the loop to reply, the model needs to learn the

answer to this user query, so they use incremental learning to do so, as shown in Figure 2. Their approach consists of mainly 3 modules:

1. Dialogue Embedding Module: at which the user utterance is embedded using Gated Recurrent Unit (GRU) based bidirectional Recurrent Neural Network (bi-RNN), and on top of it, they use self-attention layer to improve the encoding.
2. Uncertainty Estimation module: at which the confidence level between the user's utterance and the system response(s) is calculated.
3. Online Learning module: this module is only used when the confidence level from the second module is low, and none of the candidate responses are relevant to the user utterance. In this case, a human expert is involved to respond on this utterance, and the system should be updated with the given utterance and its proper response using incremental learning.

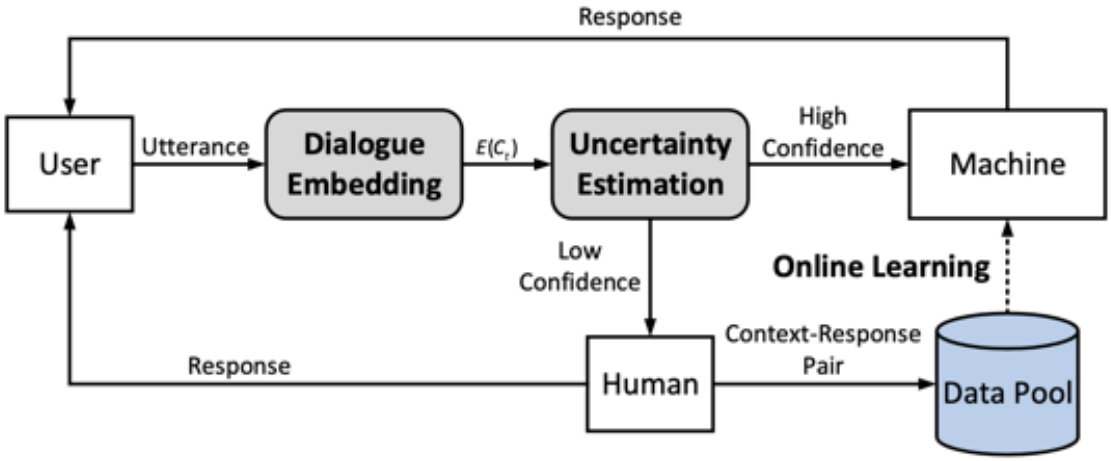

Fig. 2. Overview of IDS [27]

The same idea of adding a human-in-the-loop was discussed with a different approach by Rajendran et al. [28]. Their goal was to maximize task success in task-oriented dialogue systems while minimizing the involvement of a human expert in the loop. This is very similar to the previous approach except that here they used reinforcement learning instead of incremental learning.

The system has mainly three elements: the model M, a neural dialogue model which is trained for goal-oriented dialogues. The classifier C, which is a neural classifier that uses reinforcement learning to learn, and the human H, which is the expert in the loop. As discussed earlier, since it's very unlikely to get users' utterances to be similar to the ones in the training dataset, therefore the model's response might not be correct. The idea is that with every user utterance, the classifier gets to pick between the model and the human; this way, the classifier learns with trial and error, using a reward and punishment system.

If the classifier picked the model and the model answered correctly, the classifier gets rewarded with a high reward, but if the model answered incorrectly, then the punishment is high. Finally, if the classifier picked the human to answer, in this case, we always assume that the human response is correct, then the classifier is rewarded with a low reward. With trial and error, the classifier will try to maximize its reward by increasing the task success and minimizing the need for human.

Our incremental learning approach is mostly inspired by [27,28], both of them worked with the motivation of improving the neural network task-oriented dialogue system by adding a human in the loop to respond whenever the system outputs an invalid/incorrect answer to the user, and takes this expert's response and feeds it back to the system so it would learn the correct answer.

3 Proposed Method

Our approach consists mainly of four modules; *Knowledge Graph Extractor Module*, *Knowledge Graph Embedding Module*, *Incremental Learning Module*, and *Selection Module*. The framework takes place over two phases, the first phase is the training phase, in which we construct the KG and train the model with the first two modules, and the second phase is the deployment phase. First, we construct the KG from a question-answer dataset for the training phase by taking each question and answer and converting them into statements. Then we pass these statements to OpenIE[29], which extracts the facts/triplets from each statement. And finally, we use the constructed KG to train our model.

For the deployment phase, there are two scenarios; the first is that new knowledge needs to be added, so we use the *Incremental Learning Module* directly. The second is when interacting with the user; the user would ask a question, which we pass to the first module to convert into a statement to get the missing entity (*(h,r,?)* tail prediction or *(?,r,t)* head prediction), then we pass this required prediction to the model to retrieve the missing entity. Here we have the following cases;

- if the model could predict a link:
  - This link prediction is valid, so we output it to the user.
  - This link prediction is invalid, so we ask the human in the loop to answer and learn this new link.
- if the model could not predict a link, which might happen if the source entity is not in the entities list that the model is trained on (new entity). In that case, we redirect this task to the human in the loop and feed the model the new fact to update it.

3.1 Knowledge Graph Extractor (KGE) Module

This module aims to extract the KG facts from a Question-Answer dataset. Figure 3 illustrates an overview of this module. Having a question Q and answer A, we first pass Q to lexicalized PCFG parser[30] to extract the parse tree of the grammatical structure of the question.

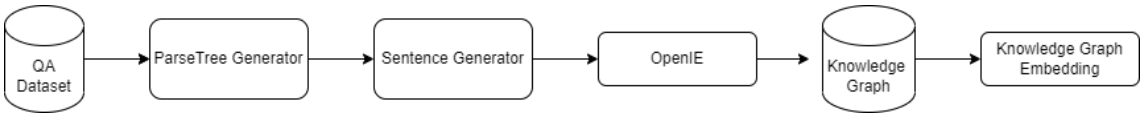

Fig. 3. Extracting Facts from QA dataset

Generating a parse tree transforms a natural language sentence/question into its equivalent syntactic tree form representing the grammatical structure. This process includes identifying groups of words (phrases), part of speech tags of these phases/words, and dependency labels. This is done in an unsupervised manner by using a pre-trained parser

provided by the Stanford Natural Language Processing Group[4], which follows the English Penn Treebank[5]. Example is shown in Figure 4.

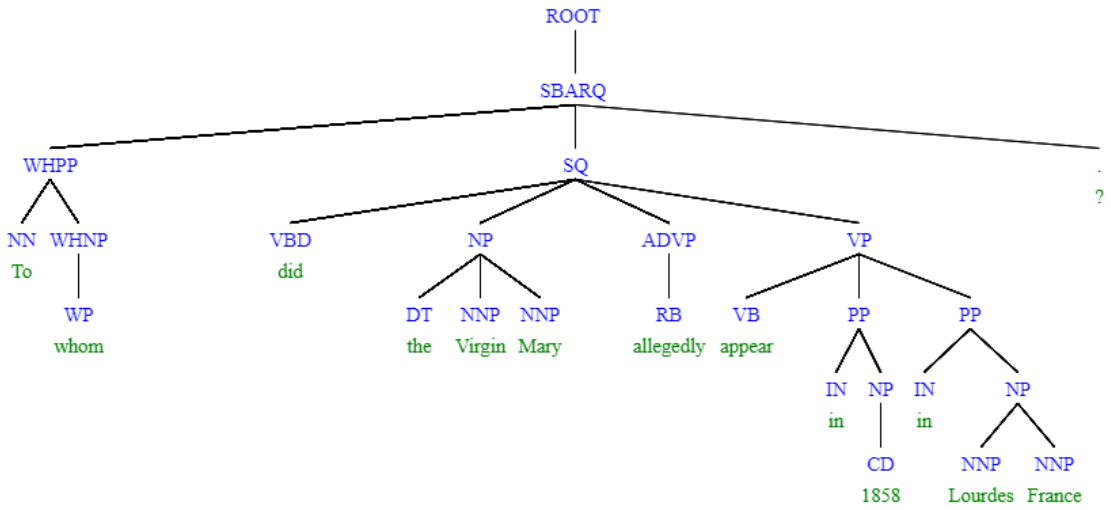

Fig. 4. Output parse tree for "To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France?"

After generating the parse tree, we use it to rewrite the question in the form of sentence S; by looping over the tree nodes. First, we decide where to place the given node in the sentence according to the node type. Then we remove the question header, and according to the type of question, we put the correct prepositions, if needed, before the answer.

This sentence S would contain a ***blank*** where the answer should be, so we simply take A and replace it with ***blank***. We use the same module for the user's question to extract the fact with the missing entity to pass to the model for the link prediction while leaving the ***blank*** placeholder to identify the prediction needed.

Once we have the sentence ready, we pass it to OpenIE[29] to extract all the possible fact triplets from the given sentence. The Open Information Extraction (OpenIE) is an unsupervised annotator that extracts the relation triples from a given system by splitting them into clauses. Each clause is used to generate a set of shorter sentence fragments. From these shorter sentences, it's easier to extract the triples [31,32,29,33]. One of the main advantages of OpenIE is that the extracted facts are humanly readable, which makes it easier to generate the answer from the knowledge graph. The result of this module should be a knowledge graph $KG=\{(e1,r,e2) \| \ e1,e2 \in \boldsymbol{E}$ *and* $r \in \boldsymbol{R}\}$ which contains the information presented in the question-answer pairs in the original dataset, which refers to the first contribution in the Introduction 1.

3.2 Knowledge Graph Embedding Module

We used TransE as our KG embedding model for our experiments. TransE constructs the embeddings for the entities $\boldsymbol{E}$ in $\|\boldsymbol{E}\|^k$, where k is the dimension of the embeddings which is passed as a hyperparameter to the model, and same for the relations $\boldsymbol{R}$ in $\|\boldsymbol{R}\|^k$. Initially, these embeddings are randomly initialized, and the model gets to learn and fine-tune these

[4] https://nlp.stanford.edu/software/lex-parser.html
[5] http://surdeanu.cs.arizona.edu//mihai/teaching/ista555-fall13/readings/PennTreebankConstituents.html

embeddings by minimizing the margin-based loss equation 2:

$$\mathcal{L} = \sum_{(e1,r,e2)\in S} \sum_{(e1',r,e2')\in S'_{(e1,r,e2)}} [\gamma + d(e1 + r, e2) - d(e1' + r, e2')]_+ \quad (2)$$

Where $\gamma > 0$ is a margin hyperparameter and the energy of a triplet $d(e1' + r, e2')$ is for some dissimilarity measured. The $[\gamma + d(e1 + r, e2) - d(e1' + r, e2')]_+$ donates only the resulted values that are positive. $S'_{(e1+r,e2)}$ is the set of corrupted triplets, where it takes triplets from the training set and replaces either the head or tail in the triplet by a random entity but not both at the same time. Shown in equation 3:

$$S'_{(e1+r,e2)} = \{(e1', r, e2)|e1' \in E\} \cup \{(e1, r, e2')|e2' \in E\}. \quad (3)$$

Because TransE is an energy-based model that uses a geometric interpretation of the latent space, it considers that a fact *(e1,r,e2)* holds when the embedding of the tail entity *e2* is close to the sum of the embedding of the head entity *e1* plus some vector that depends on the relationship *r*, i.e., *e1+r≈e2*. However, due to the nature of translation, TransE cannot correctly predict one-to-many and many-to-one relations; a selection mechanism is needed after the TransE prediction to help reduce this gap, further explained in module 3.4.

3.3 Incremental Learning Module

After training the model, we use this module in two scenarios; 1) if there is new information that should be added to the knowledge base, or 2) when the user asks the system a question and the system provides an invalid answer. An overview of this module is illustrated in Figure 5. In the second scenario, the system can not provide a correct answer; hence, we redirect to the human expert to answer this question. According to that answer, we also use it to update the model and KG.

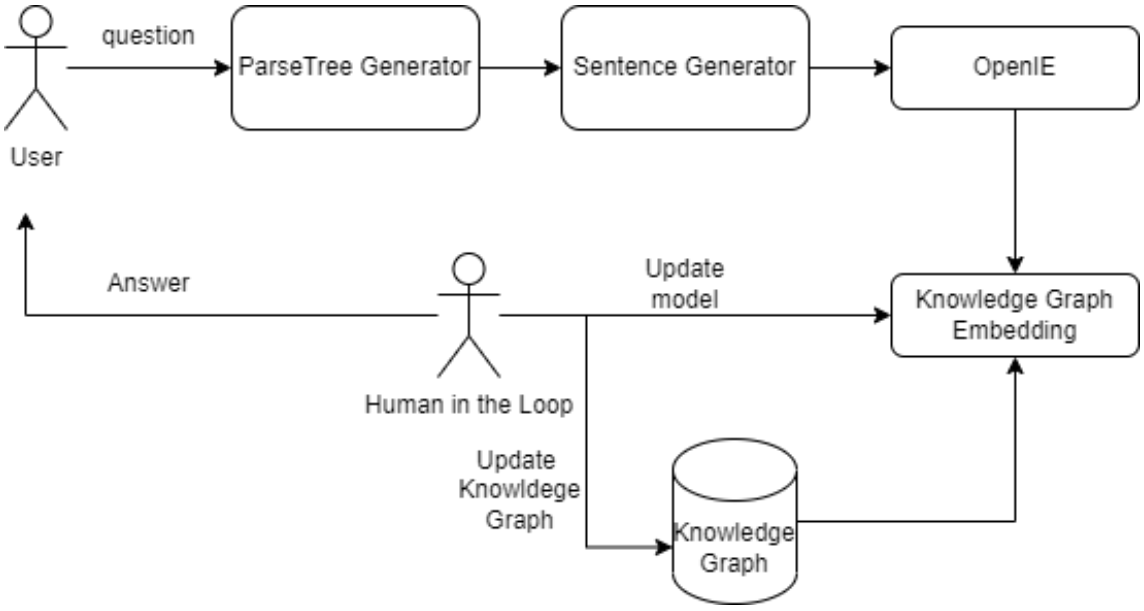

Fig. 5. Incremental Learning with HITL

As illustrated in Algorithm 1, first, we check if the 'new' fact contains any new entities. In case it includes an entity that is not in the entity list of the KG, we first add it to the list of entities by adding a new dimension to the model's entity parameter. Then, this new dimension is randomly initialized and updated by finetuning the model; by feeding it the new fact. The need for this step comes from the formal definition of the KG itself since the embeddings of the KG are dependent on the total number of entities and relations at the construction stage, as explained in subsection 3.2.

If the new fact does not contain any new entity, then we move directly to feeding the model the 'new'[6] fact. This way, the model could keep learning new knowledge without affecting the previously learned knowledge and without retraining from the beginning, which refers to the second contribution in the Introduction Section1.

Algorithm 1 Incremental Learning with the HITL feeding the model (e1,r,e2)

if $e1 \notin E$ **then**
    $E = E \cup \{e1\}$
    $embed(e1) = random(size = k)$ ▷ Randomly initializing embedding for the new entity
    $embed(E) = append(embed(E), embed(e1))$ ▷ Add randomly initialized vector for the new embedding in the embeddings
if $e2 \notin E$ **then**
    $E = E \cup e2$
    $embed(e2) = random(size = k)$
    $embed(E) = append(embed(E), embed(e2))$
train for (e1,r,e2)

3.4 Selection Module

This module is responsible for verifying the answer and selecting the correct answer output by the model since most KG embedding models mispredict the 1:n and n:1 relations, which makes sense because, without any provided context, all the facts are 'valid'. We can address this issue by passing the user's question to the KGE module 3.1 and excluding the common facts between the questions and the valid facts predicted by the model, explained in Algorithm 2. This step reduces the gap of having n:1 or 1:n relations, which refers to the third contribution.

Algorithm 2 Selection Module Algorithm

Input: Question q

$tree = ParseTree(q)$ ▷ Generate the ParseTree q
$sentence\_q = tree_to_sentence(tree)$ ▷ Generate the sentence of q without the answer
$facts = OpenIE(sentence\_q)$ ▷ Extract the facts and the missing fact to predict
$missing\_fact = facts.contain$
$top\_k\_predictions = model.predict(missing\_fact)$ ▷ save top k predictions with the highest score
$output\_fact = top\_k\_predictions - facts$ ▷ Eliminate the common facts between the two sets

Output: $output\_fact$ ▷ Output the fact with the highest score

[6] The reason we put new between " is that the fact that is being fed to the model might not be a new fact, but a fact that the model mispredict.

This way, we have fewer predicted facts; if only one remains, the system outputs that to the user and waits for the user's feedback. If it is not the correct fact that the user is looking for, then the system moves to HITL with the incremental module 3.3. If there is more than one valid fact after the filtration, the model picks the fact with the highest score as the correct output. If it is not the correct answer, move on to the next one until the user finds what they want. Finally, if none of the predicted facts were valid or no facts were left after the filtration, then we move to the HITL with the incremental module 3.3.

For example:*Q: To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France?*

- from Q we form the statement S:
  *S: The Virgin Mary allegedly appeared in 1858 in Lourdes France to **blank***
- from which we extract the following triplets:
  (The Virgin Mary, appear, in 1858)
  (The Virgin Mary, appear, in Lourdes France)
  (The Virgin Mary, appear, ?)
- Then we pass this tail prediction to the model to get the missing entity, which would give us all the relevant facts; technically, all of them are correct/valid.
  (The Virgin Mary, appear, in 1858)
  (The Virgin Mary, appear, in Lourdes France)
  (The Virgin Mary, appear, Saint Bernadette Soubirous)
- Since we know that the first two facts are already mentioned in the question, this is our context to pick the right answer; hence the system would output *(The Virgin Mary, appear, Saint Bernadette Soubirous)* which is the correct answer.

4 Experiments and Results

In this section, we first start by introducing the datasets that we used in subsection 4.1, then we present the evaluation metrics that we used in subsection 4.2, and finally, we discuss our experiments and their results on the given datasets in subsection 4.3.

4.1 Datasets

To validate our approach, we used the SQuAD[21] dataset, which is considered one of the benchmark datasets for the Question-Answering tasks. The SQuAD is a collection of question-answer pairs created from Wikipedia articles by humans through crowd-sourcing, which makes it very diverse. We used 70,000 question-answer pairs for our approach. After running the parser and the sentence generation step, we ended up with 68,445 sentences, which resulted in 100500 entities and 14783 relations, and 71194 facts after running the KGE module[7]. We used the cross-validation function by sklearn[8] to split our dataset into training, validation, and testing, each containing 56955 facts, 7119 facts, and 7120 facts, respectively. For the unseen experiments, we saved 50 entities away from the model among the corresponding 58 facts from the training, validation, and testing sets.

We compared our approach with other approaches using language models from the literature review to construct the KG. We used the same framework from the methodology to convert the question-answer pairs into sentences. Still, instead of adding the answer to the sentence, they added '[MASK]' for the language model to learn how to fill it with the correct answer. We added this step to achieve the same task: constructing the KG

[7] We will refer to the KG dataset extracted from SQuAD with SQuADKG.
[8] https://scikit-learn.org/stable/

from question-answer pairs, which is the main focus of our paper. We used the approach presented by[23] for this experiment with Roberta with their custom SpaCy and Textacy method to clean the KG. Each approach has its pros and cons, which we consider a designer choice for this step. Table 1 shows the results for running both methods on 57,355 statements as the parser and sentence generator output. On average, the LM approach took 03:33:52 to process 10,000 sentences, using GPUs offered by Colab Pro at the time we conducted this experiment which was Tesla T4, while for OpenIE, we used average processing power; Intel(R) Core i7-8750H CPU. Despite the difference in the computation power, the LM approach took a significantly longer time to process all the sentences and still extracted significantly fewer facts than the OpenIE approach; details about the LM experiment are discussed in Appendix section5.1.

Using The LM approach would provide a cleaner graph in terms of entities and relations. On the other hand, it requires multiple hops to reach the answer, which would affect the generation of the answer, as we have to keep track of the path taken to generate the answer. Although using OpenIE would make it easier to generate the answer, we might find facts like *(The National Archives, make strides towards making its holdings more widely available In 2006)*. These facts make it harder for the embedding model to understand the features and have many relations and entities with the same semantic meaning. However, this issue could be fixed by adding another layer of filtering the entities and relations before adding them to the graph.

The last point of comparison is the execution time and resource needed for each, which is presented in the last row in Table 1; the time difference between the two methods is noticeable.

POC	LM	OpenIE
count entities	6,778	87,498
count relations	5,174	14,118
count facts	12,251	59,636
Execution time hh:mm:ss	+20:00:00	02:10:27

Table 1. Constructing the KG with different approaches

We continued the rest of the experiments with the dataset generated by OpenIE, as we wanted to reduce having implicit data learned by the language model and work with a more classical approach. In addition, we don't see any improvement done by the LM, so we go with OpenIE as it requires less time and resources and still extracts more facts/knowledge.

To validate the Incremental learning module on its own, we used YAGO3-10[34] and FB15K237[35] since these are the datasets that were included in the OpenKE[36] results. The YAGO3-10[34] consists of 123182 entities and 37 relations; it contains 1,079,040 facts for the training set and 5000 facts for each validation and testing set. We picked this dataset because its entities are associated with at least ten different relations. These relations describe human attributes like association, profession, gender, etc.

The FB15K237[35] consists of 14,541 entities and 237 relations, which form 149678 facts for training, 3992 facts for testing, and 543 facts for validation.

To simulate the incremental learning with human-in-the-loop, we removed **100** entities from each of the datasets to save them for the unseen dataset along with their relevant facts from the training, validation and testing sets. These facts are then used to simulate the unseen queries for the model and apply the incremental learning approach. In addition to those, we saved any mistakenly predicted link while testing the model after training

so that we get to add these incorrect facts to the testing set. For YAGO3-10, the total number of unseen facts was 1935 and 4992 mispredicted facts. For FB15K237, the total number of unseen facts was 77, and 20319 mispredicted facts.

These three datasets are different in size, i.e., the total number of entities, relations, and the number of 1:n or n:1 relations. Since SQuADKG was constructed end-to-end, the number of relations is significantly larger than the number of relations of the other two datasets.

4.2 Metric

Mean Rank (MR): It is the average of the obtained ranks. Its range is between 1 and $\|\boldsymbol{E}\|$. As the value gets closer to 1 than $\|\boldsymbol{E}\|$, this means that the performance is improved. Because this metric is very sensitive to outliers, it's usually not used just by itself.

Mean Reciprocal Rank (MRR): It's value ranges between 0 and 1. It is the average of the inverse of the obtained ranks, and hence the higher the value is, the better the model results.

Hits@K (H@K): It is the ratio of predictions for which the rank is equal or lesser than a threshold K; its range lies between 0 and 1, where closer to 1 is better. Common values for K are 1, 3, 5, and 10. The higher the H@K, the better the model results. We mainly focus on K=10, which shows the proportion of correct entities ranked in the top 10.

4.3 Results and Analysis

In Table 2, the *TransE* row shows the results of training TransE on the YAGO3-10 dataset. After saving some entities and facts for the incremental learning part, the model was trained for 500 epochs. The results of running the model on the given test file were 56.4% for hits@10.

We saved the mispredicted facts to apply the incremental learning part to the model; this experiment aimed to simulate the human in the loop when the model mispredicted a particular fact, so we saved the facts that we knew that the model needed to 'learn.' As expected, the results after this type of incremental learning improved the model's performance on these facts. The results are shown in Table 2 in the *TransE+* row.

For the main experiment to feed the model entities that it has not seen before, we applied the incremental learning approach to this setting while adding a new dimension for this entity and initially randomizing its values till the model learns it. Then we went back to test the model's performance on the unseen facts, as shown in Table 2 in the *TransE++ unseen entities* row that the model could recognize these entities and facts correctly after this learning.

For the final experiment, we wanted to test if adding these new entities would affect the model's performance on the original data of the training phase. So we merged all the facts from the three experiments (the original test set, the mispredicted facts, and the unseen facts). Row *TransE++ all* in Table 2 shows the results of this experiment that learning new entities does not affect TransE's ability to recognize the original trained entities and facts.

We repeated those four experiments with the same order for the FB15K237 dataset, and their results are shown in Table 3. *TransE* refers to training the model with the training dataset after removing the unseen facts and showing the results of running on the test set. *TransE+* refers to the model after the incremental learning on the mispredicted facts, which was tested on those mispredicted facts after the incremental learning. One of the reasons why the performance did not reach high results for YAGO3-10 is the number

Model	MRR	MR	hit@10	hit@3	hit@1
TransE	0.342691	1704.469727	**0.564413**	0.402080	0.226445
TransE+	0.895744	2.000801	**0.988391**	0.960268	0.828263
TransE++ unseen entities	0.788238	9.513178	**0.928165**	0.848579	0.707494
TransE++ all	0.796323	16.453110	**0.940918**	0.887318	0.697302

Table 2. Results of TransE on YAGO3-10.

of 1:n and n:1 facts. *TransE++* refers to the model after incrementally learning the unseen entities and their relevant facts. Here, we show the results of testing the model on just the unseen data, shown in row *TransE++ unseen entities*, and testing the model on all the data, shown in row *TransE++ all*.

Model	MRR	MR	hit@10	hit@3	hit@1
TransE	0.279723	233.155823	**0.466185**	0.317886	0.184267
TransE+	0.489348	125.041397	**0.622388**	0.527722	0.412699
TransE++ unseen entities	0.744715	10.077922	**0.831169**	0.805195	0.681818
TransE++ all	0.452894	243.779419	**0.576355**	0.487343	0.382188

Table 3. Results of TransE on FB15K237 for the four scenarios.

The drop-off in the performance of the model after training the model on new entities in some cases, for instance, in Table 3 when it dropped from 62% (*TransE+*) to 57% (*TransE++ all*), is acceptable in our approach because initially, the model cannot predict a link of an entity it has not been trained on, but now it has this capacity for these new entities. The objective of our approach is to allow the model to learn new facts and new entities while still being able to recognize the original facts with acceptable accuracy. So this is a trade-off between being unable to output a fact if it contains an entity it hasn't seen and slightly affecting the prediction of existing facts.

After constructing its knowledge graph entities and facts, we conducted the same experiments on the SQuADKG dataset. Results are shown in Table 4. The first scenario *TransE* was to train TransE while keeping the unseen entities and their corresponding facts outside the training set, the loss while training was 0.007913. We tested the trained model on the test set, which it had not seen before, not during the training or validation. The accuracy for hit@10 was almost 2.9%, which was perfect to showcase the effect of our incremental learning effect. The model could not predict the test facts correctly because we constructed the KG manually, which means that the graph would be vulnerable to being unconnected, especially after removing some for the unseen scenario and some for the test and validation sets.

Model	MRR	MR	hit@10	hit@3	hit@1
TransE	0.015392	42327.921875	**0.029234**	0.017569	0.007309
TransE+	0.559917	62.463951	**0.911174**	0.865777	0.245678
TransE++ unseen entities	0.893739	1.844828	**0.982759**	0.956897	0.844828
TransE++ all	0.562405	62.002300	**0.911683**	0.865886	0.250244

Table 4. Results of TransE on SQuADKG for the four scenarios.

The second scenario *TransE+* was to teach the model these mispredicted/new facts and test it on the same set; here, we notice that the model was able to recognize these facts while still recognizing the other facts correctly as the score of the metrics got better, as the accuracy improved to 91% for these new facts (but not new entities). And finally, in the *TransE++* where we teach the model entities and facts they have not seen before during the training phase, the prediction score when examining the model on the unseen facts was 98%, and to make sure that learning these new entities still did not affect its overall performance we tested the model on a merged test set that combines everything, at which it is shown that the model was still able to predict the facts with 91%.

When using ComplEx on SQuADKG, we noticed that it performed poorly with an accuracy of less than 1% (0.009979) on the test set with a loss of 8.81. This could be due to the fact that the KG is constructed in an end-to-end unsupervised manner which highlights the problem with incompleteness, in addition to removing over 20% of the facts for validation and training, which makes the data significantly incomplete. Another reason is that ComplEx requires the relation r between the entities to be diagonal, as discussed in [7], which is not guaranteed in our case. Although TransE initially did not perform well either on this set, the difference in the scoring function and embeddings for each of the models made TransE more convenient to approach.

5 Conclusion and Future Work

In this paper, we start by introducing our problem statement, which consists of two parts; the first is constructing a knowledge graph from a question-answering dataset since QA datasets are more available and easier to collect, unlike KGs. However, QA representations lack the structural representation that exists in data representations such as KGs. Another advantage of the KG representation is its ability to explicitly store the data, unlike deep language models, where data is stored implicitly and appears as a black box, making it harder to manipulate.

For this problem, we propose our Knowledge Graph Extraction Module 3.1, which takes the question and extracts its grammatical structure using a parse tree generator. Then, from the generated tree, we rearrange the question in sentence form and add the answer in the correct position in the sentence. And the last step is to use OpenIE to extract the fact triples from the generated sentence. For the experiments, we used question-answer pairs from the SQuAD[21] dataset; from 70,000 question-answer pairs, we extracted 100500 entities and 14783 relations, and 71194 facts.

The second part of our problem statement addresses the incompleteness problem, which is present in most of the knowledge representations but mainly highlighted in knowledge graphs because it is shown as missing links in the graph, making it harder for the model to reach the correct answer. For this problem, we propose an incremental learning approach that allows the model to learn from a human expert in the loop that would feed the model new knowledge when needed. This Incremental learning module also allows the model to learn new entities, not in the original training entity vocab list. The need for this step is shown when the user asks a link prediction query with an unseen entity or when the human-in-the-loop tries to feed the model a new fact that contains an unseen entity; in both cases, the model would typically raise an error.

We conducted the same experiments on three different datasets to verify our framework. The first dataset is the extracted KG from the SQuAD dataset, which we refer to as SQuADKG. The second dataset is the YAGO3-10[34] dataset which is one of the benchmark datasets that contains 123,182 entities, 37 relations, and 1,179,040 triples, and the

third dataset is the FB15K237[35] which has 310,079 triples with 14,505 entities and 237 relations. Finally, for the embedding model, we used TransE[10]. We removed 100 entities with their related facts from the training, testing, and validation set to create the unseen set for the incremental learning experiments. For YAGO3-10, after training the results on the test dataset, TransE's performance was 56.44%. Then, after the incremental learning on the mispredicted facts, it reached 98.83%. Finally, for the last experiment on the unseen entities facts and testing on the original test set, the performance was 94.09%.

For FB15K237, the results were different due to the difference in the structure of the dataset itself. The experiments results on the test dataset were: initially 46.61%, after the incremental learning on the mispredicted facts, it reached 62.23%, and for the unseen entities facts with the Incremental Learning module, it achieved 83.11% on the unseen data. But on the complete test set, including the unseen data, the performance was 57.63%. The dropout from 62% to 57% is acceptable in our approach as it's a trade-off between not having the needed vocab and mispredicting some facts since the incremental learning approach can handle this issue when required.

Lastly, the SQuADKG dataset shows the most significant results, going from 2.92% in the first experiment to 91.16% in the final experiment on the unseen data. This improvement makes sense since we constructed the KG and then removed over 20% of the dataset for the test, validation, and unseen data, leaving the training dataset significantly incomplete. That's why the new knowledge fills the missing gaps in the constructed graph.

Our final contribution was to address the 1:n and n:1 link prediction problem that is known to be one of TransE's limitations, along with other models. We propose a selection method that eliminates the facts mentioned in the question to reduce the number of possible answers and supposedly output the response with the highest score. The objective was to try and reduce the complexity of each module, aiming to save resources while maintaining acceptable results.

As for the limitations of our approach that we aim to target in our future work, starting with the resources and time efficiency, we found that the parser takes the most time. Although we tested this module on limited resources, we still aim to find a more efficient way to achieve the same results with a somewhat optimized approach. Regarding the embedding model, we want to experiment with different models and verify the effect of incremental learning, with the required tuning, on each of them. The incremental learning approach would show promising results with reinforcement learning models because they naturally learn by interacting with the environment. One of the models that we are currently experimenting with is the dual agent approach[37] to navigate the KG.

Finally, we aim to expand our approach to handle multi-turn conversations by adding another layer that would keep track of the current dialogue state and use it to filter the possible answers. That would help navigate the KG better without significantly increasing the complexity.

References

1. Zheng Zhang, Ryuichi Takanobu, Qi Zhu, MinLie Huang, and XiaoYan Zhu. Recent advances and challenges in task-oriented dialog systems. *Science China Technological Sciences*, 63(10):2011–2027, 2020.
2. Xiaojun Chen, Shengbin Jia, and Yang Xiang. A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications*, 141:112948, 2020.
3. Samuel Budd, Emma C. Robinson, and Bernhard Kainz. A survey on active learning and human-in-the-loop deep learning for medical image analysis. *Medical Image Analysis*, 71:102062, 2021.
4. Andreas Holzinger, Markus Plass, Katharina Holzinger, Gloria Cerasela Crişan, Camelia-M. Pintea, and Vasile Palade. Towards interactive machine learning (iml): Applying ant colony algorithms to

solve the traveling salesman problem with the human-in-the-loop approach. In Francesco Buccafurri, Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar Weippl, editors, *Availability, Reliability, and Security in Information Systems*, pages 81–95, Cham, 2016. Springer International Publishing.

5. Yiwei Yang, Eser Kandogan, Yunyao Li, Prithviraj Sen, and Walter S Lasecki. A study on interaction in human-in-the-loop machine learning for text analytics. In *IUI Workshops*, 2019.
6. Jiwei Li, Alexander H Miller, Sumit Chopra, Marc'Aurelio Ranzato, and Jason Weston. Dialogue learning with human-in-the-loop. *arXiv preprint arXiv:1611.09823*, 2016.
7. Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(2):1–49, 2021.
8. Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR, 2016.
9. Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. Holographic embeddings of knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.
10. Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.
11. Wen Zhang, Bibek Paudel, Wei Zhang, Abraham Bernstein, and Huajun Chen. Interaction embeddings for prediction and explanation in knowledge graphs. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 96–104, 2019.
12. Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*, 2019.
13. Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.
14. Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. A novel embedding model for knowledge base completion based on convolutional neural network. *arXiv preprint arXiv:1712.02121*, 2017.
15. Jill Burstein, Christy Doran, and Thamar Solorio. 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). 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)*, 2019.
16. Thanh Vu, Tu Dinh Nguyen, Dat Quoc Nguyen, Dinh Phung, et al. A capsule network-based embedding model for knowledge graph completion and search personalization. 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 2180–2189, 2019.
17. Lingbing Guo, Zequn Sun, and Wei Hu. Learning to exploit long-term relational dependencies in knowledge graphs. In *International Conference on Machine Learning*, pages 2505–2514. PMLR, 2019.
18. A. H. Miller P. Lewis A. Bakhtin Y. Wu F. Petroni, T. Rocktäschel and S. Riedel. Language models as knowledge bases? In *In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2019*, 2019.
19. Hady Elsahar, Pavlos Vougiouklis, Arslen Remaci, Christophe Gravier, Jonathon Hare, Frederique Laforest, and Elena Simperl. T-rex: A large scale alignment of natural language with knowledge base triples. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.
20. Robyn Speer, Catherine Havasi, et al. Representing general relational knowledge in conceptnet 5. In *LREC*, volume 2012, pages 3679–86, 2012.
21. Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.
22. Chenguang Wang, Xiao Liu, and Dawn Song. Language models are open knowledge graphs. *arXiv preprint arXiv:2010.11967*, 2020.
23. Vinitra Swamy, Angelika Romanou, and Martin Jaggi. Interpreting language models through knowledge graph extraction. *arXiv preprint arXiv:2111.08546*, 2021.
24. Shibo Hao, Bowen Tan, Kaiwen Tang, Hengzhe Zhang, Eric P Xing, and Zhiting Hu. Bertnet: Harvesting knowledge graphs from pretrained language models. *arXiv preprint arXiv:2206.14268*, 2022.
25. Tarun Garg, Kaushik Roy, and Amit Sheth. Can language models capture graph semantics? from graphs to language model and vice-versa. *arXiv preprint arXiv:2206.09259*, 2022.
26. Badr AlKhamissi, Millicent Li, Asli Celikyilmaz, Mona Diab, and Marjan Ghazvininejad. A review on language models as knowledge bases. *arXiv preprint arXiv:2204.06031*, 2022.

27. Weikang Wang, Jiajun Zhang, Qian Li, Mei-Yuh Hwang, Chengqing Zong, and Zhifei Li. Incremental learning from scratch for task-oriented dialogue systems. *arXiv preprint arXiv:1906.04991*, 2019.
28. Janarthanan Rajendran, Jatin Ganhotra, and Lazaros C. Polymenakos. Learning End-to-End Goal-Oriented Dialog with Maximal User Task Success and Minimal Human Agent Use. *Transactions of the Association for Computational Linguistics*, 7:375–386, 07 2019.
29. Mausam Mausam. Open information extraction systems and downstream applications. In *Proceedings of the twenty-fifth international joint conference on artificial intelligence*, pages 4074–4077, 2016.
30. Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting of the association for computational linguistics*, pages 423–430, 2003.
31. Swarnadeep Saha et al. Open information extraction from conjunctive sentences. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2288–2299, 2018.
32. Swarnadeep Saha, Harinder Pal, et al. Bootstrapping for numerical open ie. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 317–323, 2017.
33. Harinder Pal et al. Demonyms and compound relational nouns in nominal open ie. In *Proceedings of the 5th Workshop on Automated Knowledge Base Construction*, pages 35–39, 2016.
34. Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.
35. Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd workshop on continuous vector space models and their compositionality*, pages 57–66, 2015.
36. Xu Han, Shulin Cao, Lv Xin, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. Openke: An open toolkit for knowledge embedding. In *Proceedings of EMNLP*, 2018.
37. Denghui Zhang, Zixuan Yuan, Hao Liu, Xiaodong Lin, and Hui Xiong. Learning to walk with dual agents for knowledge graph reasoning. *arXiv preprint arXiv:2112.12876*, 2021.

Appendix

5.1 Using Language Model to Generate KG

In this section, we discuss the experiment of constructing the KG using a language model (LM). The objective of this experiment was to compare and explore other approaches for constructing the KG. Starting with running the experiment on the approach proposed in [23] for which they proposed using masked sentences and passing them to the LM to allow the model to learn the features of the given sentence and accordingly extract the fact triplets. Then they filter those facts using a hybrid layer of SpaCy and Textacy to verify the extracted facts.

In Table 5, we present the results of each phase for this experiment. The *GT* column refers to the Ground Truth facts extracted using the SpaCy and Textacy libraries, the *LM* column refers to the facts predicted by the language model, the *Missed GT* column refers to the facts missed by the LM and captured in the ground truth. The *New LM* refers to the facts captured by the LM but not by the ground truth, and finally, the *Intersection* column is the facts captured by both the LM and the ground truth.

From observing the results of each phase, by manually evaluating samples of each, we noticed that the relevant facts are the intersection between the LM and the ground truth. However, the number of extracted facts was too little compared to the time and resources taken to extract them.

When experimenting with the SpaCy and Textacy layer to extract the facts to see if this filtering step could improve the quality of the facts extracted from OpenIE, we took a

POC	GT	LM	Missed GT	New LM	Intersection
count entities	11,666	6,778	9,593	4,077	4,161
count relations	6,901	5,174	4,907	2,682	3,085
count facts	14,885	12,251	11,330	7,470	5,878

Table 5. Extracting Facts Using Language Model

sample of 10,000 sentences from the question-answer processed pairs. The execution time was 1 hour, 30 minutes, and 27 seconds to generate 3048 facts that contain 3060 entities and 1840 relations. This filtering step could be an extra cleaning and verification step. Still, within our project, we didn't notice any significant improvement after applying this step relative to the time taken to process the whole dataset.

Authors

Mervat Abu-Elkheir Associate Professor and vice-dean at German University in Cairo. She has over 20 years of experience in academic teaching. Her current research interests are in the areas of interpretable machine learning and AI, AI for software engineering, data management, and engineering, and natural language understanding. She has been an IEEE member since 2014.

Mayar Osama received a Bachelor of Science in Media Engineering and Technology from German University in Cairo. Currently, she is pursuing her Master of Computer Science while working as a teaching assistant. Her research interests include Natural Language Processing, and Knowledge Reasoning and Representation