KITLM: Domain-Specific Knowledge InTegration into Language Models for Question Answering

Ankush Agarwal[1*], **Sakharam Gawade**[1*],
Amar Prakash Azad[2], **Pushpak Bhattacharyya**[1]
[1]IIT Bombay,
[2]IBM Research India
{ankushagrawal, sakharamg, pb}@cse.iitb.ac.in,
amarazad@gmail.com

Abstract

Large language models (LLMs) have demonstrated remarkable performance in a wide range of natural language tasks. However, as these models continue to grow in size, they face significant challenges in terms of computational costs. Additionally, LLMs often lack efficient domain-specific understanding, which is particularly crucial in specialized fields such as aviation and healthcare. To boost the domain-specific understanding, we propose, *KITLM*[1], a novel knowledge base integration approach into language model through relevant information infusion. By integrating pertinent knowledge, not only the performance of the language model is greatly enhanced, but the model size requirement is also significantly reduced while achieving comparable performance. Our proposed knowledge-infused model surpasses the performance of both GPT-3.5-turbo and the state-of-the-art knowledge infusion method, SKILL, achieving over 1.5 times improvement in exact match scores on the MetaQA. KITLM showed a similar performance boost in the aviation domain with AeroQA. The drastic performance improvement of KITLM over the existing methods can be attributed to the infusion of relevant knowledge while mitigating noise. In addition, we release two curated datasets to accelerate knowledge infusion research in specialized fields: a) AeroQA, a new benchmark dataset designed for multi-hop question-answering within the aviation domain, and b) Aviation Corpus, a dataset constructed from unstructured text extracted from the National Transportation Safety Board reports. Our research contributes to advancing the field of domain-specific language understanding and showcases the potential of knowledge infusion techniques in improving the performance of language models on question-answering.

[*]Equal contribution
[1]The URL for our dataset and source codes is: https://github.com/sakharamg/KITLM

1 Introduction

Large pre-trained language models (PLMs) (Raffel et al., 2020b) have succeeded remarkably in various NLP downstream tasks. Their achievements can be attributed to two key factors: extensive pre-training on diverse text sources and the ability to fine-tune domain-specific data. PLMs undergo extensive pre-training on vast amounts of text data from various sources such as books, articles, and websites. This process allows them to develop a profound understanding of language and capture a comprehensive range of linguistic patterns and contextual information. Furthermore, PLMs can be fine-tuned on domain-specific datasets, enabling them to specialize and adapt to a particular domain. This fine-tuning process refines the models' knowledge and performance, allowing them to excel in tasks specific to those domains. However, recent research has highlighted the efficacy of incorporating knowledge graphs into language models using diverse techniques (Saxena et al., 2022; Moiseev et al., 2022; Zhang et al., 2022b; Yasunaga et al., 2021a). Our paper shows that incorporating relevant structured knowledge from knowledge graphs can further enhance language model performance and domain-specific understanding.

Various studies have explored different methods for infusing knowledge into language models. One popular approach involves verbalizing triples in the knowledge base and continually pretrain the LLM using a training criteria such as masked language modeling. However, this approach can be computationally demanding. Other methods like QA-GNN (Yasunaga et al., 2021a) and GreaseLM (Zhang et al., 2022b) rely on knowledge graph embeddings (Dai et al., 2020) to obtain the domain knowledge which requires additional training. The two critical factors in a knowledge infusion method are: i) the quality of infused knowledge, which allows for achieving strong empirical performance, and ii) the

simplicity of the architecture. These underscore the need for a knowledge infusion technique that is computationally efficient while maintaining high quality and simplicity.

Our paper presents an innovative framework for integrating knowledge into language models like T5 (Raffel et al., 2020a) through fine-tuning. The experimental results demonstrate that the checkpoints trained using the proposed approach on AviationKG (Agarwal et al., 2022b) and WikiMovies (Miller et al., 2016) outperforms the T5 baselines, state-of-the-art SKILL (Moiseev et al., 2022) and GPT-3.5-turbo on MetaQA (Zhang et al., 2018) and our curated multihop QA dataset, AeroQA. Instead of introducing additional parameters to pre-trained language models (PLMs) or modifying their architectures, the proposed framework employs a novel knowledge integration objective. This objective entails verbalizing the KG triples, extracting pertinent triples for each question-answer pair using ColBERTv2 (Santhanam et al., 2022), and incorporating them during both the training and testing phases of the language model.

We conducted a comprehensive study to enhance our proposed framework, KITLM, by exploring the impact of different formats of external knowledge on a language model. We incorporated unstructured general corpora, domain-specific corpora, and structured knowledge (triples) into the T5 model for question-answering. To evaluate the effectiveness of these settings, we employed the SKILL approach (Moiseev et al., 2022) and compared T5, T5 + unstructured text, T5 + KG triples, and T5 + unstructured text + KG triples on the AeroQA and MetaQA datasets. Our proposed approach, KITLM, outperformed all other settings, underscoring the importance of integrating relevant knowledge alongside LLMs while mitigating noise for enhancing question-answering capabilities.

Our contributions are:

1. Introduce two datasets to accelerate knowledge infusion research in specialized fields: (a) AeroQA, a closed-book question-answering dataset with multi-hop reasoning. It contains 34k QA pairs, with 21k 1-hop pairs and the rest being 2-hop pairs. (b) Aviation Corpus, comprising 665,000 lines of clean English text from 4,000 NTSB [2] reports. It is specifically curated for continual pre-training to facilitate knowledge infusion tasks in language models.

2. KITLM, a novel framework introduces a seamless integration of relevant verbalized triples from a knowledge base into the language model without modifying its architecture. Leveraging ColBERTv2, KITLM extracts the most pertinent triples associated with each instance in the question-answering dataset. Our approach surpasses the state-of-the-art knowledge infusion method, SKILL, by more than 20% on both AeroQA and MetaQA datasets.

3. KITLM $>$ GPT-3.5-turbo; Our knowledge-infused model surpasses GPT-3.5-turbo by over 1.5 times in AeroQA and MetaQA, highlighting the significant reduction in the requirement of language model size through relevant knowledge infusion.

2 Motivation

Aviation-related datasets are scarce and highly sought after, posing challenges for building question-answering (QA) systems capable of reasoning over knowledge graphs like AviationKG (Agarwal et al., 2022b). To address this, we have developed a valuable multi-hop reasoning QA dataset derived from the National Transportation Safety Board reports in the aviation domain. This dataset is valuable for the aviation industry and researchers, facilitating information retrieval and QA tasks. Its creation aims to provide deeper insights into aircraft accidents and contribute to developing preventive measures to enhance aviation safety.

Large Language Models (Brown et al., 2020; Scao et al., 2022) have demonstrated efficient performance across various downstream NLP tasks. However, the high computational requirements associated with LLMs have raised concerns. Furthermore, LLMs are typically trained on generic datasets, so their suitability for domain-specific tasks is limited. Our study provides evidence that computational resources can be conserved by employing smaller language models for specific tasks. Additionally, we highlight the importance of integrating relevant knowledge in the LM to address the needs of domain-specific tasks.

The background and related work for our paper "KITLM" can be found in the Appendix.

[2] https://www.ntsb.gov/Pages/AviationQuery.aspx

3 Background and Related Work

Prevalent state-of-the-art models like BERT (Devlin et al., 2019a), GPT-3 (Brown et al., 2020), and T5 (Raffel et al., 2020b) have emerged as powerful tools for various tasks. These models are typically pre-trained on unstructured text data, allowing them to comprehend language within a contextual framework. However, knowledge about the real world is crucial to gain a comprehensive understanding of a statement. This world knowledge is frequently represented as triples within a knowledge graph.

Knowledge Graph Question Answering. A Knowledge Graph (KG) is a collection of entities and their relationships, represented as triples (subject, relation, object). KGs are commonly stored in a triple format, ranging from large-scale KGs like Wikidata (Vrandečić and Krötzsch, 2014) to small-scale KGs such as those in (Miller et al., 2016) and (Agarwal et al., 2022b). KGs are particularly valuable when accurate information can be extracted from them. Initially, querying KGs in Natural Language (NL) involved rule-based (Guo et al., 2020) and pattern-based systems (Affolter et al., 2019). Semantic parsing (Bast and Haussmann, 2015) was also utilized for solving these queries by converting NL questions into symbolic queries over the KG. However, recent advancements have shifted towards the adoption of sequence-to-sequence (seq2seq) architectures (Zhong et al., 2017) and pre-trained models, harnessing the power of neural networks.

Knowledge infusion. Extensive research on querying knowledge graphs in natural language has driven the development of diverse methods for knowledge retrieval, addressing the challenge of converting natural language into graph query language. A particularly successful approach involves combining knowledge graphs with deep learning (DL), which has generated considerable interest among researchers due to the increasing significance of knowledge globally. One commonly used approach for incorporating structured knowledge into models is to convert the knowledge into natural language text. ERNIE 3.0 (Sun et al., 2021) adopts this approach by training a knowledge-enhanced model on a corpus that combines triples and their corresponding sentences. During training, random masking is applied to either the relation in a triple or words in a sentence. Methods like QA-GNN (Yasunaga et al., 2021a) and GreaseLM (Zhang et al., 2022b) employ knowledge infusion techniques that involve propagating information through a graph to capture the dependencies and relationships among entities.

The synergy of KG and DL can be categorized into two groups: a) Utilizing KGs during inference, as demonstrated in studies like PullNet (Sun et al., 2019). b) Infusing knowledge into model weights during pre-training, as explored in approaches such as K-BERT (Liu et al., 2020), KGT5 (Saxena et al., 2022) and SKILL (Moiseev et al., 2022). This paper examines the SKILL technique for infusing knowledge into language models (LMs) during pre-training, and a novel framework called KITLM is introduced for knowledge infusion during inference in LMs. KITLM uniquely incorporates relevant knowledge into language models while effectively mitigating noise, a feature lacking in previous infusion methods.

4 Methodology

This section details the following methodologies for knowledge integration:

1. Our novel framework *KITLM*, designed for multi-hop question answering, depicted in Figure 1.

2. The T5 pre-trained model and its continual pre-training using structured knowledge, unstructured corpora, including C4 and Aviation Corpus, inspired by the state-of-the-art infusion method, SKILL (Moiseev et al., 2022).

4.1 Knowledge Integration for Multi-hop QA

An overview of the KITLM framework is depicted in Figure 1. Knowledge infusion method, especially question booster(explained later), in KITLM is designed in a way to be more effective for multi-hop question answering. KITLM can be incorporated with most of the language models seamlessly independent of the LM architecture. Designed specifically for multi-hop question-answering, KITLM exhibits adaptability across various domains without requiring modifications to the language model's architecture, as long as a knowledge graph is available. The architecture of the language model does not need to be altered as KITLM seamlessly incorporates knowledge during the inference phase.

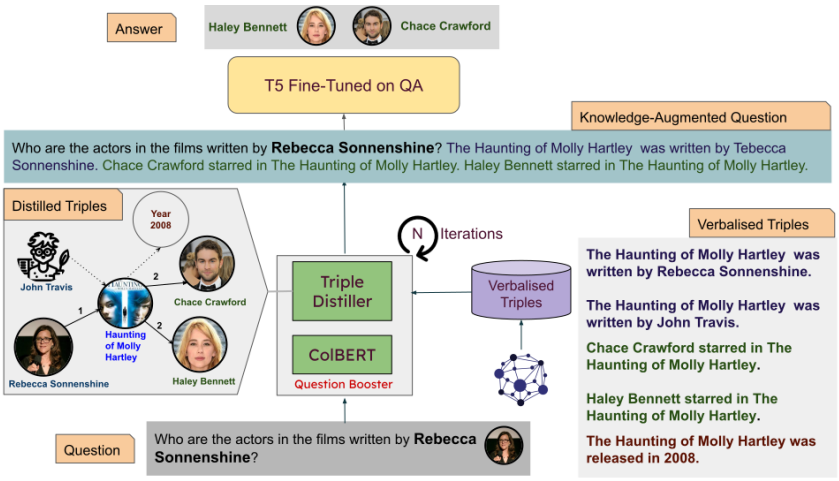


Figure 1: The proposed framework, KITLM (Section 4.1), is illustrated in the flow diagram. Initially, triples are extracted from knowledge bases like the WikiMovies dataset (Miller et al., 2016) and transformed into verbalized form. Subsequently, ColBERTv2 (Santhanam et al., 2022) is employed to retrieve the top-K relevant triples related to the given question from the set of verbalized triples. The triples are distilled N times for the N-hop question-answering. The distilled triples are then concatenated with the question and provided as input to the fine-tuned T5 to generate an answer.

4.1.1 Task Formulation

KITLM obtains the answer to the question using two stages, namely retrieval and prediction stages. Let A be the set of potential answers and a^* be the predicted answer where $a^* \in A$, Q be the set of questions and $\mathbf{q}$ be the input question, and θ represents the weights of models used in KITLM. Then the predicted answer

$$a^* = argmax_{a \in A} P(a|\mathbf{q}; \theta). \quad (1)$$

where $P(a|\mathbf{q}; \theta)$ represents the probability of answer given a question. $P(a|\mathbf{q}; \theta)$ can be decomposed into the retrieval stage and prediction stage.

Retrieval stage: Given a question q, we retrieve a set of triples $t \in T$ using an iterative retrieval mechanism, where T is the set of triples used as context for the prediction stage. We can decompose $P(a|q; \theta)$ as:

$$P(a|q) = \sum_{t \in T} P(a|t, q; \theta_p) P(t|q; \theta_r). \quad (2)$$

Here, $P(a|t, q; \theta_p)$ is the probability of the answer a given question q and triples t to the predictor and $P(t|q; \theta_r)$ is the probability of a triple t given a question q to the retriever. Further, the set of triple t can be rewritten as the set of relevant triples retrieved at each retrieval step helpful for answering a K hop question. Let t^k represent the set of triples retrieved at the k^{th} hop where $t^k \in T$. It can be written as:

$$P(t|q) = P(t^K, t^{K-1}, ..., t^k, ..., t^2, t^1|q). \quad (3)$$

For retrieving t at the k^{th} step, we need t^{k-1} ... t^1 along with question q. Therefore using chain rule, we can express $P(t|q)$ as:

$$P(t|q) = \prod_{k=2}^{K} P(t^k|t^{k-1}, ..., t^1, q) P(t^1|q). \quad (4)$$

$P(t|q)$ can be interpreted using the score used by the retriever for ranking the triples i.e. $P(t|q) \propto S_{q,t}$. We use ColBERTv2 (Santhanam et al., 2022) as the retriever to score the triples as $S_{q,t} = \sum_{i=1}^{N} max_{j=1}^{M} Q_i T_j$, where each query token's representation Q_i is aligned with the most relevant triple token representation T_j.

Prediction stage: To predict the answer a^*, we input a question $\mathbf{q}$ and its relevant triples $\mathbf{t}$ to a T5 fine-tuned model for QA task. $\mathbf{q} < /s > \mathbf{t}$ is the input to the model and a^* is the output obtained through greedy decoding. Here, $< /s >$ is the separator token used by T5.

4.1.2 KITLM Algorithm

In this section, we describe the detailed methodology of the retrieval and prediction stages implemented in KITLM which is shown in figure 1. KITLM recognizes the relevance of integrating triples with input questions as contextual information to improve question-answering accuracy. The integration process begins with the verbalization of extracted triples from a knowledge graph. To identify the most pertinent triples for a given question, ColBERTv2 is employed to index the verbalized

triples. The highest-ranked triples are selected as the context for the question during the fine-tuning process.

Since the integration of knowledge relies heavily on the retriever, we use ColBERTv2 because of its high performance in both in-domain and out-of-domain information retrieval. However, in case of multi-hop questions, the retrieval is likely to be highly noisy even with ColBERTv2. To alleviate this problem, we propose an iterative approach where after every retrieval iteration, we filter out the noise using the *triple distiller*.

We repeat the distilling process N times for a N-hop question. Additionally, after each iteration, the distilled triples are appended to the question and used as additional input for querying ColBERTv2 in the subsequent iteration. This augments the query with additional knowledge after each iteration. The method of question augmentation through triple distiller is called *Question Booster* as shown in figure 1. The iterative process is outlined in Algorithm 1.

Algorithm 1 Retrieving context for N-hop Question Answering with the KITLM approach. The context in this case comprises the pertinent triples extracted from the knowledge base.

Require:

$Q_0 \rightarrow$ Input Question
$T \rightarrow$ Triples in a Knowledge Graph
$N \rightarrow$ Number of hops in N-hop Question
$E \rightarrow$ Set of Entities
$ColBERT(Q_i|T) \rightarrow$ ColBERTv2 indexed on T

Ensure:

$Q_N = Q_0 + RelevantTriples$
procedure N-HOPQA(Q_0, T, E, N)
    $E' \leftarrow entities\ in\ Q_0$
    for $i = 0$ to $N - 1$ **do**
        $Ret \leftarrow ColBERT(Q_i|T)$ ▷ Retrieve top triples, $Ret \subseteq T$
        $Fil \leftarrow Triples \in Ret\ having\ Entities \in E'$ ▷ Clean the retrieved triples
        $Q_{i+1} \leftarrow Q_i + Fil$ ▷ Append the filtered triples to Q_i
        $E' \leftarrow entitiesin\ Fil \setminus E'$ ▷ New entities in the filtered triples
    end for
    return Q_N
end procedure

Following Algorithm 1, the multi-hop question-answering system gradually gathers relevant triples from the knowledge base, avoids repetition, and maintains an updated entity set to guide the retrieval process. The iterative loop is repeated for N iterations to achieve optimal N-hop question answering (QA) results. The retrieval process enhances the model's ability to generate accurate answers by continuously refining available information. As a result, the accuracy of multi-hop question answering improves.

The integration of relevant triples with the questions in KITLM encompasses the entire QA dataset, which includes the train, validation, and test sets. During this process, the input question is denoted as Q_0, while the retrieved sequence of triples is represented as Fil. The input provided to the language model is constructed as "question: Q_0</s>context: Fil" where </s> is a separator token. After the integration of triples, the language model undergoes fine-tuning using the training data. Following the fine-tuning process, the model is utilized on the test set to generate question-answering results.

4.2 Structured Knowledge Infusion for Language Models

In this section, we investigate incorporating knowledge into language models using knowledge triples and textual information. We delve into the details of infusing knowledge into LMs by training the T5 model on unstructured corpora, and factual triples extracted from knowledge graphs. We compare the performance of different models, namely: a) T5-large as the baseline model, b) T5-large + textual information, c) T5-large + KG triples, and d) T5 + textual information + KG triples. The results of these models are presented in the Table 1.

The knowledge infusion method during pre-training is inspired from SKILL. In this approach, triples are extracted from the knowledge graph and combined with the text to prevent any degradation in the model's performance on natural language understanding tasks. For the MetaQA dataset, the C4 text is utilized, while the curated Aviation corpus (Section 5.2) is employed for the AeroQA dataset (Section 5.1). A subset of the C4 corpus and Aviation corpus equal to the number of triples in the KG is used for continual pre-training of the language model. After combining the triples and text, the T5 model is continually pre-trained using a salient masked language modeling technique.

T5, a text-to-text transfer transformer model, was initially trained on the C4 corpus using a masked-language modeling technique. In this approach, certain spans of tokens in a sequence are randomly masked, and the model predicts the missing tokens. The approach described in (Roberts et al., 2020) is followed where instead of masking random tokens, salient terms are masked to improve performance on downstream tasks that require a deeper understanding of the sequence, such as question answering (Guu et al., 2020a). The salient terms are the entities found within the corpora and knowledge graphs. The entities in C4 with the highest predicted probability is masked by a BERT (Devlin et al., 2019a) model finetuned on the CoNLL 2003 NER dataset [3] (Tjong Kim Sang and De Meulder, 2003). For aviation corpus, we additionally masked NERs and nouns detected by Spacy [4] since entities in AviationKG and NTSB reports can also be compound nouns. E.g. Visual Conditions is an entity in the AviationKG's triple: AccidentNumber_LAX05LA060 | hasConditionsAtAccidentSite | Visual Conditions. For AviationKG and Wikimovies, we randomly masked either the head entity or tail entities.

By following the described continual-training process, a T5 model is transformed into a knowledge-infused model. Subsequently, the trained model is fine-tuned for the specific task, which in our case is question answering, leading to the creation of the fine-tuned model. The fine-tuned model is employed to generate answers for the test-set.

5 Dataset

This section introduces AeroQA, a benchmark dataset specifically designed for question-answering tasks in the aviation domain. Additionally, an aviation-related text dataset called Aviation Corpus, similar to C4, is created. The rest of the datasets used for experimentation is explained in Appendix.

5.1 AeroQA: A Benchmark Dataset for Aviation Domain

To address the limitations of the AviationQA (Agarwal et al., 2022b) dataset and evaluate the reasoning ability over the AviationKG knowledge graph, we have created AeroQA, a multi-hop question-answering dataset in the aviation domain. While AviationQA is a large dataset in the aviation domain, it is limited in two key aspects. Firstly, all the questions in AviationQA are single-hop, which does not allow for evaluating the model's ability to reason over knowledge graphs like AviationKG. Secondly, only a fraction of AviationQA pairs contain questions that can be answered using the triples from AviationKG, limiting the utilization of the full reasoning potential of the QA pairs. AeroQA is specifically curated to overcome these limitations and provide a dataset that facilitates reasoning over KGs in the aviation domain.

AeroQA is a multi-hop closedbook QA dataset for the aeronautics domain. This dataset complements the pre-processed AviationKG knowledge graph and enables reasoning tasks. The AviationKG is constructed from the National Transportation Safety Board (NTSB) reports which contain information about aircraft accidents and their investigation. AeroQA consists of a comprehensive collection of 34k questions specifically designed to assess both single-hop and multi-hop reasoning abilities. Out of these QA pairs, 21k are 1-hop QA pairs, while the remaining are for 2-hop reasoning. The dataset is divided into three parts: training, validation, and testing, with an 80:10:10 split ratio. The AeroQA dataset contains multiple answers for each question, which are separated by the 'I' symbol. The entities mentioned in the questions are enclosed within square brackets '[]'. These entities are present in the AviationKG knowledge base. The dataset consists of 87 relations for the 1-hop question-answer pairs and 35 relations for the 2-hop question-answer pairs. These relations serve as templates for constructing the question-answer pairs. The template generation process involved using the prompt-based approach with ChatGPT (OpenAI, 2023), where different relations along with their head and tail entities were used as prompt text. The model was then prompted to generate the template for the question-answer pairs. The generated output was subsequently filtered and manually checked to form the final question-answer templates.

5.2 Aviation Corpus: A dataset consisting of Aviation text

The MetaQA dataset requires C4 (Raffel et al., 2020b) corpus for the MLM training with the

[3] https://huggingface.co/dslim/bert-base-NER

[4] https://spacy.io/

Models	AeroQA 1-hop	AeroQA 2-hop	MetaQA 1-hop	MetaQA 2-hop	MetaQA 3-hop
T5-large (Baseline)	52.88	41.57	24.5	32.65	42.31
T5-large + C4	51.38	40.65	23.53	32.78	39.66
T5-large + Aviation_Corpus	52.04	41.73	-	-	-
T5-large + KG	52.64	41.19	23.89	15.82	31.30
T5-large + C4 + KG (SKILL (Moiseev et al., 2022))	54.66	41.34	71.47	33.57	43.41
T5-large + Aviation_Corpus + KG	56.78	42.11	-	-	-
GPT-3 (Fine-tuned)	24.30	20.99	18.73	16.71	54.77
GPT-3.5-turbo (one-shot)	0.37	2.22	53.90	21.07	23.06
KITLM (Our method: T5-large + $relevant\ KG - Triples$)	**86.06**	**43.52**	**91.26**	**71.19**	**71.62**

Table 1: The table displays the exact match scores obtained on the test set for three models, T5-large, GPT-3, and GPT-3.5, utilized for the QA tasks. In our proposed KITLM approach (Section 4.1), the term $relevant\ KG - Triples$ represents distilled triples sourced from the knowledge base, which were utilized to provide contextual information for the questions. The table includes models labeled as T5-large + Z, which underwent continual pre-training with additional inputs (Z) such as text, KG triples, or a combination of text and triples. These pre-trained models were further fine-tuned for the QA task to generate exact match scores. The performance of the KITLM approach is also compared with the state-of-the-art language models, namely GPT-3 and GPT-3.5-turbo. GPT-3 was fine-tuned on the corresponding dataset for the QA task. The "-" symbol refers to non-applicability as the aviation corpus is a distinct domain and cannot be applied to the MetaQA dataset. In contrast, the C4 dataset, being a generic domain dataset, applies to both MetaQA and AeroQA.

SKILL (Moiseev et al., 2022) approach. To conduct experiments using our AeroQA dataset, we compiled the Aviation corpus, comprising 665k lines of English text related to the aviation domain. This corpus was obtained by scraping 4,000 National Transportation Safety Board reports from the NTSB website, covering the period between 1981 and 2018. The reports, initially in PDF format, were converted to JSON format for easier processing. The paragraphs that contain clean text were extracted from selected sections of the reports which are Analysis, Probable Cause and Findings, and Factual Information. The selected paragraphs were then curated and included in the Aviation corpus, which served as a valuable resource for our research and experimentation.

6 Results and Analysis

In table 1, we depict the results for the question-answering task and compare the performance of our proposed framework, KITLM, and different knowledge integration settings using the T5-large model. The evaluation is conducted on the AviationQA and MetaQA datasets. Additionally, the table includes the performance of GPT-3 and GPT-3.5-turbo (ChatGPT). The experimental setup is explained in the Appendix.

The T5-large model is used as the baseline, while the state-of-the-art knowledge infusion method, described in (Moiseev et al., 2022), is referred to as the SKILL pre-trained combined model. This model combines T5 with additional input, which is unstructured corpora denoted as X, and also incorporates triples as part of the knowledge infusion process. The T5 + X + KG demonstrates improved performance compared to T5, T5 + X, and T5 + KG for both MetaQA and AeroQA datasets. In MetaQA, the performance is adversely affected when using only non-verbalized KG-Triples, leading to a decrease in scores compared to the C4 text experiment. Although the T5 + KG-Triples model exhibits a slight improvement in the AeroQA, it fails to surpass the T5 baseline due to catastrophic interference, despite its inherent domain-specific benefits. The T5 + KG-Triples result underscores the importance of incorporating unstructured corpora for language models, as relying solely on triples can result in catastrophic forgetting. But this doesn't mean that the integration of triples is irrelevant, and infusing only text into the language model will help. We observed a decline in the performance for the tasks after utilization of only C4 text compared to the baseline. The rationale behind the ongoing use of text-only pre-training is

Dataset	Question	Gold Answer	ChatGPT (using prompt)	KITLM
AeroQA 1-hop	What environmental issue caused [AccidentNumber_IAD05LA071]?	Tailwheel	N/A (The given Accident Number does not provide any information related to an environmental issue)	tailwheel
AeroQA 2-hop	What is the aircraft category of the registered aircraft involved in [AccidentNumber_LAX04LA084]?	Airplane	turbo-jet-turbofan-turboprop-turboshaft	airplane
MetaQA 1-hop	What movies were [Jessica Simpson] an actor in?	Employee of the Month|Blonde Ambition	Employee of the Month| Blonde Ambition|Private Valentine: Blonde & Dangerous|The Love Guru	employee of the month
MetaQA 2-hop	What are the primary languages in the movies directed by [David Mandel]?	German	N/A	german
MetaQA 3-hop	The movies that share actors with the movie [Waxworks] were in which languages?	Swedish|German| French|English	N/A	german

Table 2: The table compares examples from the AeroQA and MetaQA datasets, showcasing the differences between ChatGPT and our proposed framework, KITLM.

explained in (Raffel et al., 2020b). It is proposed that repeatedly training on C4 text could potentially lead to a decrease in performance for a T5 model. Although incorporating the aviation corpus in the AeroQA task significantly improves the score compared to the C4 text, this improvement can be primarily attributed to the domain similarity between the corpus and the task itself. To conclude, considering the individual inclusion of unstructured text and triples, the outcomes vary, with some cases showing a decline in performance while others exhibit slight improvements. However, when both text and triples are combined using the SKILL, the results demonstrate enhancements. It is important to acknowledge that explaining the performance of these approaches can be complex, and their effectiveness relies on the particular corpora employed, such as C4 or the Aviation Corpus. Furthermore, the continual pre-training demands higher computational resources.

To address the aforementioned limitations, we developed the KITLM approach, demonstrating the best performance across all tasks. The reason why KITLM performs better is as it selects the most relevant triples from the knowledge base after mitigating noise. The triples serve as the context for answering the question. KITLM is a selective approach that helps to remove confusing elements compared to existing methods, such as SKILL. This resulted in KITLM to achieve a significant improvement of 30% and 20% in the exact match score for the AeroQA 1-hop and MetaQA 1-hop tasks, respectively. Attributing to similar reasons, KITLM demonstrates better performance for the 2-hop and 3-hop QA. The reason KITLM surpasses other methods is due to its adeptness in handling multi-hop question-answering tasks, which demand advanced reasoning abilities for accurate answers. Unlike previous infusion methods that relied on continual pre-training, which can result in catastrophic forgetting, KITLM addresses this challenge effectively by removing the pre-training procedure to infuse knowledge. Another advantage of KITLM is its ability to mitigate the computational power requirements associated with knowledge infusion methods, which typically arise from pre-training. KITLM addresses this issue by selecting triples N times for N-hop QA, eliminating the need to train the model repeatedly. This approach not only reduces computational costs but also improves performance. The significance of the KITLM is amplified by its capability to adapt to diverse domains and tasks, eliminating the necessity for domain-specific corpora in the process of knowledge infusion.

Performance on GPT-3 and ChatGPT. In our experiments on MetaQA and AeroQA datasets, we observed that ChatGPT, despite being a

powerful model, encountered challenges in producing accurate results. In table 2, we depict some examples from both datasets. ChatGPT is able to provide some answers for 1-hop MetaQA questions. This can be attributed to the fact that ChatGPT has movie domain knowledge. However, ChatGPT faces challenges in 2-hop and 3-hop questions. In the case of AeroQA, ChatGPT faces challenges even in the 1-hop question. This can be because chatGPT might lack knowledge of a specialized domain, aviation. On the other hand, KITLM is able to answer all the questions in table 2. These observations highlight the limitations of ChatGPT in domain-specific question-answering tasks and the effectiveness of the KITLM in achieving higher accuracy with relevant knowledge infusion. It becomes evident that smaller language models, when combined with knowledge infusion techniques, can achieve better accuracy than LLMs for the QA task.

Following the evaluation of ChatGPT's performance with one-shot learning, we fine-tune the GPT-3 model for the QA tasks. However, the results obtained for AeroQA 1-hop and MetaQA 1-hop were relatively poor, with EM scores of 24.3% and 18.73% respectively. Also, the 2-hop and 3-hop tasks performed worse than KITLM.

Our results using GPT-3 and ChatGPT highlight the significant advantage of knowledge infusion for smaller models compared to large language models such as the GPT series. Through the incorporation of knowledge infusion techniques, we achieved superior performance across various tasks, demonstrating the effectiveness of leveraging domain-specific knowledge to enhance the capabilities of smaller models.

7 Conclusion and Future Work

Our framework, KITLM, addresses the challenge of providing context for multi-hop question answering by leveraging a knowledge base to filter and select relevant information. Our study underscores the ongoing importance of incorporating domain-specific knowledge and context, even with the availability of large language models like GPT-3 and ChatGPT, to enhance their performance in specialized domains. We have also introduced the AeroQA dataset, designed specifically for multi-hop question-answering in the aviation domain, and the Aviation corpus, which serves as a valuable resource for knowledge infusion tasks in language models.

Our future work aims to enhance NLP tasks like sentiment analysis and summarization by leveraging knowledge bases' reasoning capabilities. We'll address existing knowledge source limitations and explore ways to integrate domain-specific knowledge directly into language models, potentially replacing traditional knowledge bases.

References

Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. 2019. A comparative survey of recent natural language interfaces for databases. *The VLDB Journal*, 28(5):793–819.

Ankush Agarwal, Sakharam Gawade, Sachin Channabasavarajendra, and Pushpak Bhattacharya. 2022a. There is no big brother or small brother:knowledge infusion in language models for link prediction and question answering. In *Proceedings of the 19th International Conference on Natural Language Processing (ICON)*, pages 204–211, New Delhi, India. Association for Computational Linguistics.

Ankush Agarwal, Raj Gite, Shreya Laddha, Pushpak Bhattacharyya, Satyanarayan Kar, Asif Ekbal, Prabhjit Thind, Rajesh Zele, and Ravi Shankar. 2022b. Knowledge graph - deep learning: A case study in question answering in aviation safety domain. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 6260–6270, Marseille, France. European Language Resources Association.

Oshin Agarwal, Heming Ge, Siamak Shakeri, and Rami Al-Rfou. 2021. Knowledge graph based synthetic corpus generation for knowledge-enhanced language model pre-training. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3554–3565, Online. Association for Computational Linguistics.

Hannah Bast and Elmar Haussmann. 2015. More accurate question answering on freebase. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 1431–1440.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Yuanfei Dai, Shiping Wang, Neal N. Xiong, and Wenzhong Guo. 2020. A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9(5).

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019a. 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. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019b. 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. Association for Computational Linguistics.

J. Guo, Z. Wang, Y. He, J. Su, and Y. Yu. 2020. A survey on rule-based reasoning for knowledge graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(4):1–32.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020a. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3929–3938. PMLR.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020b. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. 2020. K-bert: Enabling language representation with knowledge graph. In *AAAI*.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1400–1409, Austin, Texas. Association for Computational Linguistics.

Fedor Moiseev, Zhe Dong, Enrique Alfonseca, and Martin Jaggi. 2022. SKILL: Structured knowledge infusion for large language models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1581–1588, Seattle, United States. Association for Computational Linguistics.

OpenAI. 2023. Chatgpt. Accessed on May 31, 2023.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020a. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020b. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5418–5426, Online. Association for Computational Linguistics.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3715–3734, Seattle, United States. Association for Computational Linguistics.

Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. 2022. Sequence-to-sequence knowledge graph completion and question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2814–2828, Dublin, Ireland. Association for Computational Linguistics.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Noam M. Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. *ArXiv*, abs/1804.04235.

Haitian Sun, Tania Bedrax-Weiss, and William Cohen. 2019. PullNet: Open domain question answering with iterative retrieval on knowledge bases and text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2380–2390, Hong Kong, China. Association for Computational Linguistics.

Yu Sun, Shuohuan Wang, Shikun Feng, Siyu Ding, Chao Pang, Junyuan Shang, Jiaxiang Liu, Xuyi Chen, Yanbin Zhao, Yuxiang Lu, et al. 2021. Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation. *arXiv preprint arXiv:2107.02137*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In

Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003, pages 142–147.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85.

Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021a. QA-GNN: Reasoning with language models and knowledge graphs for question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 535–546, Online. Association for Computational Linguistics.

Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021b. Qa-gnn: Reasoning with language models and knowledge graphs for question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 535–546.

X Zhang, A Bosselut, M Yasunaga, H Ren, P Liang, C Manning, and J Leskovec. 2022a. Greaselm: Graph reasoning enhanced language models for question answering. In *International Conference on Representation Learning (ICLR)*.

Xikun Zhang, Antoine Bosselut, Michihiro Yasunaga, Hongyu Ren, Percy Liang, Christopher D. Manning, and Jure Leskovec. 2022b. Greaselm: Graph reasoning enhanced language models for question answering.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Thirty-second AAAI conference on artificial intelligence*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Appendix

A Experiment Data

Our research utilizes the following datasets: a) Aviation Knowledge Graph (AviationKG) (Agarwal et al., 2022b) and AeroQA (Section 5.1): AviationKG is a knowledge graph designed explicitly for the aviation domain, while AeroQA is a question-answering dataset curated for multi-hop reasoning over AviationKG, b) MetaQA (Zhang et al., 2018): MetaQA consists of a knowledge base constructed from the WikiMovies dataset (Miller et al., 2016) and a set of question-answer pairs. It serves as a benchmark dataset for multi-hop reasoning. WikiMovies represents the movie domain. The statistics of these datasets are presented in Table 3 and 4.

The QA datasets chosen for the experiments undergo preprocessing to make them suitable for the experimental procedures. During the preprocessing stage, if a single question has multiple answers separated by the 'l' symbol, each answer is treated as a distinct instance of a question-answer pair. Rather than considering them as a single combined instance, they are split into individual question-answer pairs, with each answer associated with the same question. This separation facilitates improved handling and analysis of the data throughout the experiments.

We chose these datasets deliberately because they cover diverse domains and exhibit variations in terms of size and characteristics, allowing us to evaluate the performance and generalizability of our proposed method across different contexts. This choice allows us to demonstrate the versatility of our approach across diverse datasets.

Additionally, Table 4 provides the statistics for the AviationQA dataset, which was utilized in the experimentation conducted by Agarwal et al. 2022a. However, we did not employ this dataset in our experiments due to its limitation in lacking multi-hop reasoning capabilities for QA tasks, as discussed in detail in Section 5.1.

Dataset	# of triples
AviationKG	193,372
WikiMovies	269,482

Table 3: The statistics of triples (subject, relation, object) for two knowledge bases: AviationKG (Agarwal et al., 2022b) and WikiMovies (Zhang et al., 2018).

Dataset	Train	Validation	Test
MetaQA 1-hop	96,106	9,992	9,947
MetaQA 2-hop	118,980	14,872	14,872
MetaQA 3-hop	114,196	14,274	14,274
AeroQA 1-hop	17,038	2,130	2,131
AeroQA 2-hop	10,433	1,305	1,305
AviationQA	367,304	10,000	10,000

Table 4: The statistics of question-answer pairs from the aviation and movie domains. The dataset MetaQA (Zhang et al., 2018) includes 1-hop, 2-hop, and 3-hop questions from the movies domain. AviationQA (Agarwal et al., 2022a) specifically contains 1-hop questions from aviation domain. Our curated dataset, AeroQA, comprises both 1-hop and 2-hop questions. These statistics provide an overview of the question-answer distribution across different datasets used in our research.

To provide an overview of the dataset, Table 4 presents the distribution of 1-hop and 2-hop questions. Below, we present a selection of examples from the AeroQA dataset to provide a glimpse into its content.

Examples of One-hop Questions in AeroQA:

- Q: What certificate does [Pilot_ATL03LA101] have?
  A: Private
- Q: What is the engine manufacturer associated with [Registration_N127RB]?
  A: Lycoming
- Q: What caused [AccidentNumber_FTW93LA202]?
  A: Pre-Flight Planning | Fluid Fuel | Terrain Condition

Examples of Two-hop Questions in AeroQA:

- Q: What is the aircraft category of the registered aircraft involved in [AccidentNumber_CHI03LA242]?
  A: Airplane | Gyroplane
- What could have contributed to the cause of the accident [AccidentNumber_SEA96TA046]?
  A: Pilot in Command | Pilot of other Aircraft | Check Pilot

The structure of the templates in the AeroQA dataset is exemplified in Table 5 for 1-hop questions and Table 6 for 2-hop questions.

B Experimental Setup

We applied the SKILL (Moiseev et al., 2022) approach and our proposed KITLM approach on the T5-large model, which has 770M parameters. Additionally, we included the GPT-3 and GPT-3.5-turbo models for comparison with the knowledge-infused T5.

SKILL. The approach consists of two parts: continual pre-training and fine-tuning. In the process of continual pre-training, a balanced distribution is maintained by integrating both text and triples, ensuring an equal ratio of 50:50 between the two. For T5-large, we conducted SKILL training for 20 epochs with a batch size of 32, followed by fine-tuning for 20 epochs with a batch size of 128. We used seeds 0 and 42 for continual pre-training and fine-tuning, respectively. During the training process, we utilized AdaFactor (Shazeer and Stern, 2018) as the optimizer with specific settings: a learning rate of 1e-3, scale_parameter as False, relative_step as False, and warmup_init as False [5]. The maximum sequence length for both training and fine-tuning was set to 128, and a doc stride of 128 was applied during fine-tuning.

Baseline. For baseline comparisons, we utilize pre-trained T5 checkpoints of the same size. In order to isolate the effect of knowledge infusion from the influence of additional text sources such as C4 and Aviation corpus used for pre-training, we follow a similar approach as the SKILL (Moiseev et al., 2022). However, since the code repository for the SKILL is not available, we implemented our own code for the method. To differentiate the effects of knowledge infusion, we create a second baseline by training the T5 checkpoints on the text for half of the previously mentioned steps. This adjustment ensures that the amount of text pre-training aligns with the SKILL model, allowing us to attribute any observed improvements to knowledge infusion.

[5] https://discuss.huggingface.co/t/t5-finetuning-tips/684/3

Relation	Template
hasAircraftManufacturer	What is the aircraft manufacturer associated with [HEAD]
hasFederalAviationRegulation	What is the Federal Aviation Regulation associated with [HEAD]
OccurredAtCountry	In which country did [HEAD] occur

Table 5: The table displays the templates employed in constructing the AeroQA 1-hop dataset. These templates utilize the placeholder [HEAD], which corresponds to the head entity of the KG triples, *i.e.*, accident number, and registration number present in the NTSB report.

Relation1	Relation2	Template
hasRegistrationNumber	hasAirworthinessCertificate	What is the airworthiness certificate of the registered aircraft involved in [HEAD]
IsCausedBy	IsCausedDueTo	What could have contributed to the cause of the accident [HEAD]
hasPilot	hasInstructorRating	What was the instructor rating of the pilot in the aircraft involved in [HEAD]

Table 6: The table showcases the templates used for constructing the AeroQA 2-hop dataset. In these templates, the placeholder [HEAD] represents the head entity of the KG triples, *i.e.*, accident number, and registration number of the NTSB report, which is utilized to generate the 2-hop AeroQA pairs.

Furthermore, to assess the significance of text in conjunction with structured data, we create an additional T5 baseline that only utilizes triples. All other settings remain consistent with the SKILL.

KITLM. The approach comprises two main modules: (a) the retrieval module, which extracts triples from the knowledge base to provide contextual information, and (b) the fine-tuning module, which involves fine-tuning the T5-large model using the question+context combination. For single-hop QA, we retrieved the top 5 triples. In the first iteration of multi-hop QA, the top-k triples are retrieved with a value of k=3. In the subsequent iteration, k^2 triples are retrieved due to the reduction in triples after filtration (explained in Section 4.1). This iteration process continues for N-hop QA, with N*k triples retrieved in the final iteration. For the fine-tuning of the model, a batch size of 128 is used, while the other settings are the same as the experimental setup of SKILL.

GPTs. The GPT-3 and GPT-3.5-turbo models were accessed via the OpenAI API, and specifically, the GPT-3 model was fine-tuned on the QA tasks for AeroQA and MetaQA. During the fine-tuning of the GPT-3 model, we employed a batch size of 32 for AeroQA and 128 for MetaQA. To control the randomness of the model in training, a temperature of 0 was employed, and the model was trained for two epochs. This decision was made based on the observation that the loss started converging by the second epoch for the Curie model. To accommodate the multi-word factual answers present in the dataset, we have set the maximum token size to 50 for both GPT-3 and GPT-3.5-turbo. For GPT-3.5-turbo, we utilized a prompt-based approach along with one-shot learning. The prompt instructed the model to predict the answer to a given question and to output "N/A" if the answer was not available. To construct the prompt, we included a random example from the development set of the corresponding datasets. However, specifically for GPT-3.5-turbo, we pre-processed the dataset by removing the square brackets from both the test set and the example included in the prompt. This adjustment was implemented because it was observed that removing the brackets slightly improved the performance compared to when the brackets were not removed. These training and fine-tuning parameters were utilized to ensure consistency and enable meaningful comparisons between the models in our experiments.

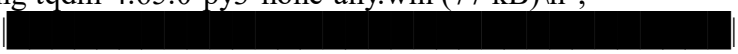
Evaluation. During the evaluation, as part of pre-processing, both the correct answers and predicted answers of the test set for all models are converted to lowercase. The exact match score is utilized as the evaluation metric on the test set for all experiments In the case of a QA dataset where a question can have multiple correct answers,

the scoring is determined based on the following criteria: If the predicted answer matches any of the correct answers associated with a question, a score of 1 is assigned. Conversely, if the predicted answer does not match any of the correct answers, a score of 0 is assigned. Unlike the T5 model, which predicts only one answer and matches it with the gold answers, GPTs are generative models that generate answers. We determine if any of the gold answers are present in the generated answer and assign a value of 1 if there is a match and 0 otherwise.

```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 4,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Requirement already satisfied: sentencepiece in /opt/conda/lib/python3.8/site-packages
(0.1.94)\n",
      "Collecting datasets\n",
      "  Downloading datasets-2.12.0-py3-none-any.whl (474 kB)\n",
      "\u001b[K     |
```


```
| 474 kB 17.7 MB/s eta
0:00:01\n",
      "\u001b[?25hCollecting pyarrow>=8.0.0\n",
      "  Downloading pyarrow-11.0.0-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(35.0 MB)\n",
      "\u001b[K     |
```


```
| 35.0 MB 54.7 MB/s eta
0:00:01\n",
      "\u001b[?25hCollecting responses<0.19\n",
      "  Downloading responses-0.18.0-py3-none-any.whl (38 kB)\n",
      "Requirement already satisfied: numpy>=1.17 in /opt/conda/lib/python3.8/site-packages (from
datasets) (1.19.2)\n",
      "Collecting xxhash\n",
      "  Downloading xxhash-3.2.0-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(213 kB)\n",
      "\u001b[K     |
```


```
| 213 kB 86.0 MB/s eta
0:00:01\n",
      "\u001b[?25hRequirement already satisfied: pandas in /opt/conda/lib/python3.8/site-packages (from
datasets) (1.1.4)\n",
      "Collecting tqdm>=4.62.1\n",
      "  Downloading tqdm-4.65.0-py3-none-any.whl (77 kB)\n",
      "\u001b[K     |
```


```
| 77 kB 26.1 MB/s eta
0:00:01\n",
      "\u001b[?25hCollecting fsspec[http]>=2021.11.1\n",
      "  Downloading fsspec-2023.4.0-py3-none-any.whl (153 kB)\n",
      "\u001b[K     |
```


```
| 153 kB 94.6 MB/s eta
0:00:01\n",
      "\u001b[?25hCollecting multiprocess\n",
      "  Downloading multiprocess-0.70.14-py38-none-any.whl (132 kB)\n",
      "\u001b[K     |
```


```
| 132 kB 99.9 MB/s eta
0:00:01\n",
      "\u001b[?25hRequirement already satisfied: requests>=2.19.0 in /opt/conda/lib/python3.8/site-
packages (from datasets) (2.24.0)\n",
      "Collecting dill<0.3.7,>=0.3.0\n",
```

```
"  Downloading dill-0.3.6-py3-none-any.whl (110 kB)\n",
"\u001b[K     |████████████████████████████████| 110 kB 103.1 MB/s eta 0:00:01\n",
"\u001b[?25hRequirement already satisfied: pyyaml>=5.1 in /opt/conda/lib/python3.8/site-packages (from datasets) (5.3.1)\n",
"Requirement already satisfied: huggingface-hub<1.0.0,>=0.11.0 in /opt/conda/lib/python3.8/site-packages (from datasets) (0.13.4)\n",
"Collecting aiohttp\n",
"  Downloading aiohttp-3.8.4-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.0 MB)\n",
"\u001b[K     |████████████████████████████████| 1.0 MB 71.0 MB/s eta 0:00:01\n",
"\u001b[?25hRequirement already satisfied: packaging in /opt/conda/lib/python3.8/site-packages (from datasets) (20.4)\n",
"Requirement already satisfied: urllib3>=1.25.10 in /opt/conda/lib/python3.8/site-packages (from responses<0.19->datasets) (1.25.11)\n",
"Requirement already satisfied: pytz>=2017.2 in /opt/conda/lib/python3.8/site-packages (from pandas->datasets) (2020.1)\n",
"Requirement already satisfied: python-dateutil>=2.7.3 in /opt/conda/lib/python3.8/site-packages (from pandas->datasets) (2.8.1)\n",
"Requirement already satisfied: chardet<4,>=3.0.2 in /opt/conda/lib/python3.8/site-packages (from requests>=2.19.0->datasets) (3.0.4)\n",
"Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.8/site-packages (from requests>=2.19.0->datasets) (2020.11.8)\n",
"Requirement already satisfied: idna<3,>=2.5 in /opt/conda/lib/python3.8/site-packages (from requests>=2.19.0->datasets) (2.10)\n",
"Requirement already satisfied: typing-extensions>=3.7.4.3 in /opt/conda/lib/python3.8/site-packages (from huggingface-hub<1.0.0,>=0.11.0->datasets) (3.7.4.3)\n",
"Requirement already satisfied: filelock in /opt/conda/lib/python3.8/site-packages (from huggingface-hub<1.0.0,>=0.11.0->datasets) (3.0.12)\n",
"Collecting async-timeout<5.0,>=4.0.0a3\n",
"  Downloading async_timeout-4.0.2-py3-none-any.whl (5.8 kB)\n",
"Collecting charset-normalizer<4.0,>=2.0\n",
"  Downloading charset_normalizer-3.1.0-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (195 kB)\n",
"\u001b[K     |████████████████████████████████| 195 kB 84.8 MB/s eta 0:00:01\n",
"\u001b[?25hCollecting aiosignal>=1.1.2\n",
"  Downloading aiosignal-1.3.1-py3-none-any.whl (7.6 kB)\n",
"Collecting frozenlist>=1.1.1\n",
"  Downloading frozenlist-1.3.3-cp38-cp38-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (161 kB)\n",
"\u001b[K     |████████████████████████████████| 161 kB 78.3 MB/s eta 0:00:01\n",
"\u001b[?25hCollecting yarl<2.0,>=1.0\n",
"  Downloading yarl-1.9.2-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (266 kB)\n",
```

```
    "\u001b[K     |████████████████████████████████| 266 kB 85.6 MB/s eta 0:00:01\n",
    "\u001b[?25hRequirement already satisfied: attrs>=17.3.0 in /opt/conda/lib/python3.8/site-packages (from aiohttp->datasets) (20.3.0)\n",
    "Collecting multidict<7.0,>=4.5\n",
    "  Downloading multidict-6.0.4-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (121 kB)\n",
    "\u001b[K     |████████████████████████████████| 121 kB 91.9 MB/s eta 0:00:01\n",
    "\u001b[?25hRequirement already satisfied: pyparsing>=2.0.2 in /opt/conda/lib/python3.8/site-packages (from packaging->datasets) (2.4.7)\n",
    "Requirement already satisfied: six in /opt/conda/lib/python3.8/site-packages (from packaging->datasets) (1.15.0)\n",
    "Installing collected packages: pyarrow, responses, xxhash, tqdm, async-timeout, charset-normalizer, frozenlist, aiosignal, multidict, yarl, aiohttp, fsspec, dill, multiprocess, datasets\n",
    "  Attempting uninstall: tqdm\n",
    "    Found existing installation: tqdm 4.53.0\n",
    "    Uninstalling tqdm-4.53.0:\n",
    "      Successfully uninstalled tqdm-4.53.0\n",
    "\u001b[31mERROR: After October 2020 you may experience errors when installing or updating packages. This is because pip will change the way that it resolves dependency conflicts.\n",
    "\n",
    "We recommend you use --use-feature=2020-resolver to test your packages with the new resolver before it becomes the default.\n",
    "\n",
    "huggingface-hub 0.13.4 requires packaging>=20.9, but you'll have packaging 20.4 which is incompatible.\u001b[0m\n",
    "Successfully installed aiohttp-3.8.4 aiosignal-1.3.1 async-timeout-4.0.2 charset-normalizer-3.1.0 datasets-2.12.0 dill-0.3.6 frozenlist-1.3.3 fsspec-2023.4.0 multidict-6.0.4 multiprocess-0.70.14 pyarrow-11.0.0 responses-0.18.0 tqdm-4.65.0 xxhash-3.2.0 yarl-1.9.2\n"
   ]
  }
 ],
 "source": [
  "!pip install sentencepiece datasets"
 ]
},
{
 "cell_type": "code",
 "execution_count": 60,
 "metadata": {},
 "outputs": [
  {
   "ename": "NameError",
   "evalue": "name 'transformers' is not defined",
   "output_type": "error",
   "traceback": [
    "\u001b[0;31m---------------------------------------------------------------------------\u001b[0m",
    "\u001b[0;31mNameError\u001b[0m                                 Traceback (most recent call last)",
```

```
      "\u001b[0;32m<ipython-input-60-de6040bb8a15>\u001b[0m in
\u001b[0;36m<module>\u001b[0;34m\u001b[0m\n\u001b[0;32m----> 1\u001b[0;31m
\u001b[0mprint\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0mtransformers\u001b[0m\u001b[0;34m.\
u001b[0m\u001b[0m__version__\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m\u001b[0m\u001b[
0;34m\u001b[0m\u001b[0m\n\u001b[0m",
      "\u001b[0;31mNameError\u001b[0m: name 'transformers' is not defined"
     ]
    }
   ],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Requirement already satisfied: seaborn in /opt/conda/lib/python3.8/site-packages (0.12.2)\n",
      "Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /opt/conda/lib/python3.8/site-packages
(from seaborn) (3.3.3)\n",
      "Requirement already satisfied: numpy!=1.24.0,>=1.17 in /opt/conda/lib/python3.8/site-packages
(from seaborn) (1.19.2)\n",
      "Requirement already satisfied: pandas>=0.25 in /opt/conda/lib/python3.8/site-packages (from
seaborn) (1.1.4)\n",
      "Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.8/site-packages (from
matplotlib!=3.6.1,>=3.1->seaborn) (0.10.0)\n",
      "Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.8/site-packages (from
matplotlib!=3.6.1,>=3.1->seaborn) (9.5.0)\n",
      "Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
/opt/conda/lib/python3.8/site-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2.4.7)\n",
      "Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.8/site-packages (from
matplotlib!=3.6.1,>=3.1->seaborn) (1.3.1)\n",
      "Requirement already satisfied: python-dateutil>=2.1 in /opt/conda/lib/python3.8/site-packages
(from matplotlib!=3.6.1,>=3.1->seaborn) (2.8.1)\n",
      "Requirement already satisfied: pytz>=2017.2 in /opt/conda/lib/python3.8/site-packages (from
pandas>=0.25->seaborn) (2020.1)\n",
      "Requirement already satisfied: six in /opt/conda/lib/python3.8/site-packages (from
cycler>=0.10->matplotlib!=3.6.1,>=3.1->seaborn) (1.15.0)\n"
     ]
    }
   ],
   "source": [
    "!pip install seaborn"
   ]
  },
  {
```

```
"cell_type": "code",
"execution_count": 62,
"metadata": {},
"outputs": [],
"source": [
 "import pandas as pd\n",
 "import random\n",
 "import csv\n",
 "import numpy as np\n",
 "from datasets import load_dataset\n",
 "from transformers import T5Tokenizer, T5ForConditionalGeneration, T5Config\n",
 "from transformers import  DataCollatorWithPadding\n",
 "from transformers import get_scheduler\n",
 "from torch.utils.data import DataLoader\n",
 "from tqdm.auto import tqdm\n",
 "from transformers import Adafactor\n",
 "import torch"
]
},
{
"cell_type": "code",
"execution_count": 63,
"metadata": {},
"outputs": [
 {
  "name": "stdout",
  "output_type": "stream",
  "text": [
   "4.28.1\n"
  ]
 }
],
"source": [
 "import transformers\n",
 "print(transformers.__version__)"
]
},
{
"cell_type": "code",
"execution_count": 37,
"metadata": {},
"outputs": [],
"source": [
 "triples = []\n",
 "total = 0\n",
 "files = [\"test.txt\", \"train.txt\", \"valid.txt\"]\n",
 "for f in files:\n",
 "    with open(f'aviationKG/AviationKG_{f}', newline='') as file:\n",
 "        lines = file.readlines()\n",
```

```
 "        total += len(lines)\n",
 "        for line in lines:\n",
 "            tokens = line.split(\"\\t\")\n",
 "            if len(tokens) != 3:\n",
 "                print(tokens)\n",
 "                continue\n",
 "                \n",
 "            tokens =  [token.strip() for token in tokens]\n",
 "            tokens[0] = tokens[0][13:].strip()\n",
 "            triples.append(tokens)\n",
 "        \n",
 "        "
]
},
{
"cell_type": "code",
"execution_count": 38,
"metadata": {},
"outputs": [
 {
  "data": {
   "text/plain": [
    "96686"
   ]
  },
  "execution_count": 38,
  "metadata": {},
  "output_type": "execute_result"
 }
],
"source": [
 "len(triples)"
]
},
{
"cell_type": "code",
"execution_count": 39,
"metadata": {},
"outputs": [
 {
  "data": {
   "text/plain": [
    "['r_LAX00LA285', 'type', 'NamedIndividual']"
   ]
  },
  "execution_count": 39,
  "metadata": {},
  "output_type": "execute_result"
 }
```

```
],
"source": [
 "triples[1]"
]
},
{
"cell_type": "code",
"execution_count": 40,
"metadata": {},
"outputs": [],
"source": [
 "random.seed(0)\n",
 "random.shuffle(triples)"
]
},
{
"cell_type": "code",
"execution_count": 41,
"metadata": {},
"outputs": [],
"source": [
 "df = pd.DataFrame(triples, columns =['subject', 'relation', 'object'])\n"
]
},
{
"cell_type": "code",
"execution_count": 42,
"metadata": {},
"outputs": [
 {
  "data": {
   "text/plain": [
    "96686"
   ]
  },
  "execution_count": 42,
  "metadata": {},
  "output_type": "execute_result"
 }
],
"source": [
 "len(df)"
]
},
{
"cell_type": "code",
"execution_count": 43,
"metadata": {},
"outputs": [
```

```
{
 "data": {
  "text/html": [
  "<div>\n",
  "<style scoped>\n",
  "    .dataframe tbody tr th:only-of-type {\n",
  "        vertical-align: middle;\n",
  "    }\n",
  "\n",
  "    .dataframe tbody tr th {\n",
  "        vertical-align: top;\n",
  "    }\n",
  "\n",
  "    .dataframe thead th {\n",
  "        text-align: right;\n",
  "    }\n",
  "</style>\n",
  "<table border=\"1\" class=\"dataframe\">\n",
  "  <thead>\n",
  "    <tr style=\"text-align: right;\">\n",
  "      <th></th>\n",
  "      <th>subject</th>\n",
  "      <th>relation</th>\n",
  "      <th>object</th>\n",
  "    </tr>\n",
  "  </thead>\n",
  "  <tbody>\n",
  "    <tr>\n",
  "      <th>0</th>\n",
  "      <td>r_FTW03LA001</td>\n",
  "      <td>type</td>\n",
  "      <td>Accident_Number</td>\n",
  "    </tr>\n",
  "    <tr>\n",
  "      <th>1</th>\n",
  "      <td>N366KR</td>\n",
  "      <td>hasEmergencyLocatorTransmitterInstalled</td>\n",
  "      <td>false</td>\n",
  "    </tr>\n",
  "    <tr>\n",
  "      <th>2</th>\n",
  "      <td>r_CHI04LA052</td>\n",
  "      <td>unitOfTemperature</td>\n",
  "      <td>degreeCelsius</td>\n",
  "    </tr>\n",
  "    <tr>\n",
  "      <th>3</th>\n",
  "      <td>r_LAX02LA036</td>\n",
  "      <td>hasWindSpeed</td>\n",
```

```
    "      <td>4</td>\n",
    "    </tr>\n",
    "    <tr>\n",
    "      <th>4</th>\n",
    "      <td>r_LAX02LA074</td>\n",
    "      <td>unitOfDewPoint</td>\n",
    "      <td>degreeCelsius</td>\n",
    "    </tr>\n",
    "  </tbody>\n",
    "</table>\n",
    "</div>"
   ],
   "text/plain": [
    "       subject                          relation          object\n",
    "0  r_FTW03LA001                              type  Accident_Number\n",
    "1        N366KR  hasEmergencyLocatorTransmitterInstalled           false\n",
    "2  r_CHI04LA052                   unitOfTemperature    degreeCelsius\n",
    "3  r_LAX02LA036                       hasWindSpeed                4\n",
    "4  r_LAX02LA074                    unitOfDewPoint    degreeCelsius"
   ]
  },
  "execution_count": 43,
  "metadata": {},
  "output_type": "execute_result"
 }
],
"source": [
 "df.head()"
]
},
{
"cell_type": "code",
"execution_count": 44,
"metadata": {},
"outputs": [],
"source": [
 "def masked_subject(row):\n",
 "    return \"<extra_id_0> {} {}\".format(row[\"relation\"], row[\"object\"])\n",
 "def masked_object(row):\n",
 "    return \"{} {} <extra_id_0>\".format(row[\"subject\"], row[\"relation\"])"
]
},
{
"cell_type": "code",
"execution_count": 45,
"metadata": {},
"outputs": [],
"source": [
 "def form_label(row):\n",
```

```
    "    return \"<extra_id_0> {}.\".format(row)\n",
    "    "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 46,
   "metadata": {},
   "outputs": [],
   "source": [
    "df[\"label\"] = np.where(df.index % 2, df[\"subject\"].map(form_label),
df[\"object\"].map(form_label))\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 47,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/html": [
       "<div>\n",
       "<style scoped>\n",
       "    .dataframe tbody tr th:only-of-type {\n",
       "        vertical-align: middle;\n",
       "    }\n",
       "\n",
       "    .dataframe tbody tr th {\n",
       "        vertical-align: top;\n",
       "    }\n",
       "\n",
       "    .dataframe thead th {\n",
       "        text-align: right;\n",
       "    }\n",
       "</style>\n",
       "<table border=\"1\" class=\"dataframe\">\n",
       "  <thead>\n",
       "    <tr style=\"text-align: right;\">\n",
       "      <th></th>\n",
       "      <th>subject</th>\n",
       "      <th>relation</th>\n",
       "      <th>object</th>\n",
       "      <th>label</th>\n",
       "    </tr>\n",
       "  </thead>\n",
       "  <tbody>\n",
       "    <tr>\n",
       "      <th>0</th>\n",
```

```
"     <td>r_FTW03LA001</td>\n",
"     <td>type</td>\n",
"     <td>Accident_Number</td>\n",
"     <td>&lt;extra_id_0&gt; Accident_Number.</td>\n",
"    </tr>\n",
"    <tr>\n",
"     <th>1</th>\n",
"     <td>N366KR</td>\n",
"     <td>hasEmergencyLocatorTransmitterInstalled</td>\n",
"     <td>false</td>\n",
"     <td>&lt;extra_id_0&gt; N366KR.</td>\n",
"    </tr>\n",
"    <tr>\n",
"     <th>2</th>\n",
"     <td>r_CHI04LA052</td>\n",
"     <td>unitOfTemperature</td>\n",
"     <td>degreeCelsius</td>\n",
"     <td>&lt;extra_id_0&gt; degreeCelsius.</td>\n",
"    </tr>\n",
"    <tr>\n",
"     <th>3</th>\n",
"     <td>r_LAX02LA036</td>\n",
"     <td>hasWindSpeed</td>\n",
"     <td>4</td>\n",
"     <td>&lt;extra_id_0&gt; r_LAX02LA036.</td>\n",
"    </tr>\n",
"    <tr>\n",
"     <th>4</th>\n",
"     <td>r_LAX02LA074</td>\n",
"     <td>unitOfDewPoint</td>\n",
"     <td>degreeCelsius</td>\n",
"     <td>&lt;extra_id_0&gt; degreeCelsius.</td>\n",
"    </tr>\n",
"  </tbody>\n",
"</table>\n",
"</div>"
],
"text/plain": [
"       subject                          relation          object  \\\n",
"0  r_FTW03LA001                                type  Accident_Number   \n",
"1       N366KR  hasEmergencyLocatorTransmitterInstalled           false   \n",
"2  r_CHI04LA052                 unitOfTemperature    degreeCelsius   \n",
"3  r_LAX02LA036                     hasWindSpeed                4   \n",
"4  r_LAX02LA074                    unitOfDewPoint    degreeCelsius   \n",
"\n",
"                      label  \n",
"0  <extra_id_0> Accident_Number.  \n",
"1          <extra_id_0> N366KR.  \n",
"2    <extra_id_0> degreeCelsius.  \n",
```

```
      "3     <extra_id_0> r_LAX02LA036.  \n",
      "4    <extra_id_0> degreeCelsius.  "
     ]
    },
    "execution_count": 47,
    "metadata": {},
    "output_type": "execute_result"
   }
  ],
  "source": [
   "df.head()"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 48,
  "metadata": {},
  "outputs": [],
  "source": [
   "df[\"input\"] = np.where(df.index % 2, df.apply(masked_subject, axis = 1),df.apply(masked_object, axis = 1))\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 49,
  "metadata": {},
  "outputs": [
   {
    "data": {
     "text/html": [
      "<div>\n",
      "<style scoped>\n",
      "    .dataframe tbody tr th:only-of-type {\n",
      "        vertical-align: middle;\n",
      "    }\n",
      "\n",
      "    .dataframe tbody tr th {\n",
      "        vertical-align: top;\n",
      "    }\n",
      "\n",
      "    .dataframe thead th {\n",
      "        text-align: right;\n",
      "    }\n",
      "</style>\n",
      "<table border=\"1\" class=\"dataframe\">\n",
      "  <thead>\n",
      "    <tr style=\"text-align: right;\">\n",
      "      <th></th>\n",
```

```
"     <th>subject</th>\n",
"     <th>relation</th>\n",
"     <th>object</th>\n",
"     <th>label</th>\n",
"     <th>input</th>\n",
"    </tr>\n",
"  </thead>\n",
"  <tbody>\n",
"    <tr>\n",
"     <th>0</th>\n",
"     <td>r_FTW03LA001</td>\n",
"     <td>type</td>\n",
"     <td>Accident_Number</td>\n",
"     <td>&amp;lt;extra_id_0&amp;gt; Accident_Number.</td>\n",
"     <td>r_FTW03LA001 type &amp;lt;extra_id_0&amp;gt;</td>\n",
"    </tr>\n",
"    <tr>\n",
"     <th>1</th>\n",
"     <td>N366KR</td>\n",
"     <td>hasEmergencyLocatorTransmitterInstalled</td>\n",
"     <td>false</td>\n",
"     <td>&amp;lt;extra_id_0&amp;gt; N366KR.</td>\n",
"     <td>&amp;lt;extra_id_0&amp;gt; hasEmergencyLocatorTransmitterIns...</td>\n",
"    </tr>\n",
"    <tr>\n",
"     <th>2</th>\n",
"     <td>r_CHI04LA052</td>\n",
"     <td>unitOfTemperature</td>\n",
"     <td>degreeCelsius</td>\n",
"     <td>&amp;lt;extra_id_0&amp;gt; degreeCelsius.</td>\n",
"     <td>r_CHI04LA052 unitOfTemperature &amp;lt;extra_id_0&amp;gt;</td>\n",
"    </tr>\n",
"    <tr>\n",
"     <th>3</th>\n",
"     <td>r_LAX02LA036</td>\n",
"     <td>hasWindSpeed</td>\n",
"     <td>4</td>\n",
"     <td>&amp;lt;extra_id_0&amp;gt; r_LAX02LA036.</td>\n",
"     <td>&amp;lt;extra_id_0&amp;gt; hasWindSpeed 4</td>\n",
"    </tr>\n",
"    <tr>\n",
"     <th>4</th>\n",
"     <td>r_LAX02LA074</td>\n",
"     <td>unitOfDewPoint</td>\n",
"     <td>degreeCelsius</td>\n",
"     <td>&amp;lt;extra_id_0&amp;gt; degreeCelsius.</td>\n",
"     <td>r_LAX02LA074 unitOfDewPoint &amp;lt;extra_id_0&amp;gt;</td>\n",
"    </tr>\n",
"  </tbody>\n",
```

```
   "</table>\n",
   "</div>"
  ],
  "text/plain": [
   "       subject                          relation          object  \\\n",
   "0  r_FTW03LA001                                 type  Accident_Number   \n",
   "1       N366KR  hasEmergencyLocatorTransmitterInstalled           false   \n",
   "2  r_CHI04LA052                  unitOfTemperature    degreeCelsius   \n",
   "3  r_LAX02LA036                      hasWindSpeed                4   \n",
   "4  r_LAX02LA074                     unitOfDewPoint    degreeCelsius   \n",
   "\n",
   "                       label  \\\n",
   "0  <extra_id_0> Accident_Number.   \n",
   "1          <extra_id_0> N366KR.   \n",
   "2    <extra_id_0> degreeCelsius.   \n",
   "3     <extra_id_0> r_LAX02LA036.   \n",
   "4    <extra_id_0> degreeCelsius.   \n",
   "\n",
   "                                    input  \n",
   "0                 r_FTW03LA001 type <extra_id_0>  \n",
   "1  <extra_id_0> hasEmergencyLocatorTransmitterIns...  \n",
   "2       r_CHI04LA052 unitOfTemperature <extra_id_0>  \n",
   "3                    <extra_id_0> hasWindSpeed 4  \n",
   "4          r_LAX02LA074 unitOfDewPoint <extra_id_0>  "
  ]
 },
 "execution_count": 49,
 "metadata": {},
 "output_type": "execute_result"
}
],
"source": [
 "df.head()"
]
},
{
"cell_type": "code",
"execution_count": 50,
"metadata": {},
"outputs": [
 {
  "data": {
   "text/plain": [
   "array(['type', 'hasEmergencyLocatorTransmitterInstalled',\n",
   "       'unitOfTemperature', 'hasWindSpeed', 'unitOfDewPoint',\n",
   "       'IsCausedBy', 'isCausedByEnvironmentIssue', 'hasPrecipitation',\n",
   "       'hasObservationFacility', 'hasRatedPower', 'unitOfVisibility',\n",
   "       'hasConditionsAtAccidentSite', 'hasOperator', 'hasPilot',\n",
   "       'unitOfDirectionFromAccidentSite', 'unitOfRatedPower',\n",
```

```
"     'hasSecondPilotPresent', 'hasAircraftFire', 'unitOfElevation',\n",
"     'hasAirworthinessCertificate', 'IsCausedBecause',\n",
"     'hasAircraftExplosion', 'hasSerialNumber', 'hasTemperature',\n",
"     'wasToxicologyPerformed', 'unitOfGusts', 'occurredAtCountry',\n",
"     'hasSecondEvent', 'hasMedicalCertification', 'hasRegisteredOwner',\n",
"     'hasLowestCloudCondition', 'unitOfDistanceFromAccidentSite',\n",
"     'isCausedByPersonnelIssue', 'isCauseddueto-PersonnelIssue',\n",
"     'hasTurbulenceSeverityForecast', 'hasFederalAviationRegulation',\n",
"     'hasCausalAgent_FlightCrew', 'hasLowestCeiling', 'isAmateurBuilt',\n",
"     'hasRunwayLength', 'subPropertyOf', 'hasFirstEvent',\n",
"     'hasAircraftManufacturer', 'hasYearOfManufacture',\n",
"     'hadLastFAAMedicalExam', 'hasEngine', 'hasDewPoint',\n",
"     'unitOfCertifiedMaxGrossWeight', 'hasSeats', 'hasAirport',\n",
"     'hasRunwayUsed', 'hasCrewInjury', 'hasAirspace',\n",
"     'hasTurbulenceSeverityActual', 'IsCausedDueTo',\n",
"     'unitOfWindDirection', 'hasRunwaySurfaceCondition',\n",
"     'hasSeatOccupied', 'hasAirframeTotalTime', 'occurredAtCity',\n",
"     'hasEmergencyLocatorTransmitter', 'occurredAtLongitude',\n",
"     'unitOfRunwayWidth', 'hasAltimeterSetting', 'hasThirdEvent',\n",
"     'hasTurbulenceActual', 'hasObscuration', 'hasRunwaySurfaceType',\n",
"     'hasTurbulenceForecast', 'hasAirportName', 'hasRestraintUsed',\n",
"     'unitOfRunwayLength', 'hasPassangerInjury', 'hasAge',\n",
"     'hasOccupationalPilot', 'hasCertifiedMaxGrossWeight',\n",
"     'hasAircraftDamage', 'hasRunwayWidth', 'hadLastFlightReview',\n",
"     'hasDepartureTime', 'isCausedByAircraftIssue',\n",
"     'hasAircraftCategory', 'hasDistanceFromAccidentSite',\n",
"     'hasAirplaneRating', 'hasDirectionFromAccidentSite',\n",
"     'hasTotalInjury', 'hasClearance', 'hasOtherAircraftRating',\n",
"     'hasInstructorRating', 'hasLastInspection', 'hasLandingGear',\n",
"     'hasVisibility', 'hasTimeSinceLastInspection',\n",
"     'occurredAtLatitude', 'hasInstrumentRating', 'subClassOf',\n",
"     'hasDefiningEvent', 'label', 'unitOfAirportElevation',\n",
"     'hasCertificate', 'isCauseddueto-AircraftIssue',\n",
"     'hasAircraftModel', 'unitOfAltimeterSetting',\n",
"     'http://purl.org/dc/elements/1.1/description', 'unitOfWindSpeed',\n",
"     'unitOfAge', 'hasVisibilityRVR', 'hasDateOfLastInspection',\n",
"     'hasObservationTime', 'unitOfLongitude', 'hasEngineModel',\n",
"     'hasElevation', 'hasAirportElevation', 'hasFlightPlanFiled',\n",
"     'hasEmergencyLocatorTransmitterActivated', 'hasEngineManufacturer',\n",
"     'hasInstrumentFlightRulesApproach', 'hasVisualFlightRulesApproach',\n",
"     'unitOfLatitude', 'hasFourthEvent', 'hasDateTime',\n",
"     'hasRegistrationNumber', 'comment', 'hasWindDirection',\n",
"     'hasConditionOfLight', 'unitOfTimeSinceLastInspection',\n",
"     'hasCausalAgent_GroundPersonnel', 'hasOperatingCertificatesHeld',\n",
"     'domain', 'hasGusts', 'hasCausalAgent_OtherPersonsOnBoard',\n",
"     'range', 'hasCausalAgent_Maintenance/Repair',\n",
"     'hasCausalAgent_Manufacturer', 'isCauseddueto-EnvironmentalIssue',\n",
"     'hasFifthEvent', 'http://purl.org/dc/elements/1.1/source',\n",
"     'hasCausalAgent_AerodromePersonnel', 'hasCausalAgent_Other',\n",
```

```
    "       'first', 'hasCausalAgent_ATSPersonnel', 'defintion',\n",
    "       'hasSixthEvent', 'rest', 'hasCausalAgent_Operator',\n",
    "       'hasCausalAgent_Government_-_FAA', 'hasCausalAgent_Unknown',\n",
    "       'members', 'imports', 'seeAlso', 'disjointWith'], dtype=object)"
   ]
  },
  "execution_count": 50,
  "metadata": {},
  "output_type": "execute_result"
 }
],
"source": [
 "df.relation.unique()"
]
},
{
"cell_type": "code",
"execution_count": 51,
"metadata": {},
"outputs": [
 {
  "data": {
   "text/plain": [
    "0                                  type\n",
    "1       hasEmergencyLocatorTransmitterInstalled\n",
    "2                          unitOfTemperature\n",
    "3                             hasWindSpeed\n",
    "4                           unitOfDewPoint\n",
    "                  ...                  \n",
    "96681                       hasRestraintUsed\n",
    "96682                              type\n",
    "96683                              type\n",
    "96684                 hasLowestCloudCondition\n",
    "96685                              type\n",
    "Name: relation, Length: 96686, dtype: object"
   ]
  },
  "execution_count": 51,
  "metadata": {},
  "output_type": "execute_result"
 }
],
"source": [
 "df[\"relation\"]"
]
},
{
"cell_type": "code",
"execution_count": 52,
```

```
  "metadata": {},
  "outputs": [],
  "source": [
   "to_rem = ['http://purl.org/dc/elements/1.1/source', 'http://purl.org/dc/elements/1.1/description',
'imports']\n",
   "df = df[~df.relation.isin(to_rem)]\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 53,
  "metadata": {},
  "outputs": [
   {
    "data": {
     "text/plain": [
      "(96625,\n",
      "         subject                          relation          object  \\\n",
      " 0  r_FTW03LA001                                type  Accident_Number   \n",
      " 1       N366KR  hasEmergencyLocatorTransmitterInstalled           false   \n",
      " 2  r_CHI04LA052                 unitOfTemperature    degreeCelsius   \n",
      " 3  r_LAX02LA036                     hasWindSpeed                4   \n",
      " 4  r_LAX02LA074                   unitOfDewPoint    degreeCelsius   \n",
      " \n",
      "                      label  \\\n",
      " 0  <extra_id_0> Accident_Number.   \n",
      " 1         <extra_id_0> N366KR.   \n",
      " 2    <extra_id_0> degreeCelsius.   \n",
      " 3     <extra_id_0> r_LAX02LA036.   \n",
      " 4    <extra_id_0> degreeCelsius.   \n",
      " \n",
      "                                    input  \n",
      " 0                 r_FTW03LA001 type <extra_id_0>  \n",
      " 1  <extra_id_0> hasEmergencyLocatorTransmitterIns...  \n",
      " 2       r_CHI04LA052 unitOfTemperature <extra_id_0>  \n",
      " 3                  <extra_id_0> hasWindSpeed 4  \n",
      " 4         r_LAX02LA074 unitOfDewPoint <extra_id_0>  ,\n",
      "       index       subject                          relation  \\\n",
      " 0         0  r_FTW03LA001                                type   \n",
      " 1         1       N366KR  hasEmergencyLocatorTransmitterInstalled   \n",
      " 2         2  r_CHI04LA052                 unitOfTemperature   \n",
      " 3         3  r_LAX02LA036                     hasWindSpeed   \n",
      " 4         4  r_LAX02LA074                   unitOfDewPoint   \n",
      " ...     ...          ...                              ...   \n",
      " 96620  96681          002                 hasRestraintUsed   \n",
      " 96621  96682          154                             type   \n",
      " 96622  96683                                          type   \n",
      " 96623  96684  r_DEN02LA067                hasLowestCloudCondition   \n",
      " 96624  96685           L)                             type   \n",
```

```
    " \n",
    "               object                        label  \\\n",
    " 0     Accident_Number  <extra_id_0> Accident_Number.   \n",
    " 1              false          <extra_id_0> N366KR.   \n",
    " 2      degreeCelsius    <extra_id_0> degreeCelsius.   \n",
    " 3                  4     <extra_id_0> r_LAX02LA036.   \n",
    " 4      degreeCelsius    <extra_id_0> degreeCelsius.   \n",
    " ...              ...                           ...   \n",
    " 96620        Seatbelt             <extra_id_0> 002.   \n",
    " 96621  NamedIndividual  <extra_id_0> NamedIndividual.   \n",
    " 96622  NamedIndividual               <extra_id_0> .   \n",
    " 96623           Clear           <extra_id_0> Clear.   \n",
    " 96624  NamedIndividual              <extra_id_0> L).   \n",
    " \n",
    "                                          input  \n",
    " 0                    r_FTW03LA001 type <extra_id_0>  \n",
    " 1      <extra_id_0> hasEmergencyLocatorTransmitterIns...  \n",
    " 2           r_CHI04LA052 unitOfTemperature <extra_id_0>  \n",
    " 3                       <extra_id_0> hasWindSpeed 4  \n",
    " 4             r_LAX02LA074 unitOfDewPoint <extra_id_0>  \n",
    " ...                                          ...  \n",
    " 96620           <extra_id_0> hasRestraintUsed Seatbelt  \n",
    " 96621                          154 type <extra_id_0>  \n",
    " 96622                <extra_id_0> type NamedIndividual  \n",
    " 96623  r_DEN02LA067 hasLowestCloudCondition <extra_id_0>  \n",
    " 96624                <extra_id_0> type NamedIndividual  \n",
    " \n",
    " [96625 rows x 6 columns])"
   ]
  },
  "execution_count": 53,
  "metadata": {},
  "output_type": "execute_result"
 }
],
"source": [
 "len(df), df.head(), df.reset_index()\n"
]
},
{
"cell_type": "code",
"execution_count": 54,
"metadata": {},
"outputs": [
 {
  "data": {
   "text/html": [
    "<div>\n",
    "<style scoped>\n",
```

```
"    .dataframe tbody tr th:only-of-type {\n",
"        vertical-align: middle;\n",
"    }\n",
"\n",
"    .dataframe tbody tr th {\n",
"        vertical-align: top;\n",
"    }\n",
"\n",
"    .dataframe thead th {\n",
"        text-align: right;\n",
"    }\n",
"</style>\n",
"<table border=\"1\" class=\"dataframe\">\n",
"  <thead>\n",
"    <tr style=\"text-align: right;\">\n",
"      <th></th>\n",
"      <th>subject</th>\n",
"      <th>relation</th>\n",
"      <th>object</th>\n",
"      <th>label</th>\n",
"      <th>input</th>\n",
"    </tr>\n",
"  </thead>\n",
"  <tbody>\n",
"    <tr>\n",
"      <th>0</th>\n",
"      <td>r_FTW03LA001</td>\n",
"      <td>type</td>\n",
"      <td>Accident_Number</td>\n",
"      <td>&amp;lt;extra_id_0&amp;gt; Accident_Number.</td>\n",
"      <td>r_FTW03LA001 type &amp;lt;extra_id_0&amp;gt;</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>1</th>\n",
"      <td>N366KR</td>\n",
"      <td>hasEmergencyLocatorTransmitterInstalled</td>\n",
"      <td>false</td>\n",
"      <td>&amp;lt;extra_id_0&amp;gt; N366KR.</td>\n",
"      <td>&amp;lt;extra_id_0&amp;gt; hasEmergencyLocatorTransmitterIns...</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>2</th>\n",
"      <td>r_CHI04LA052</td>\n",
"      <td>unitOfTemperature</td>\n",
"      <td>degreeCelsius</td>\n",
"      <td>&amp;lt;extra_id_0&amp;gt; degreeCelsius.</td>\n",
"      <td>r_CHI04LA052 unitOfTemperature &amp;lt;extra_id_0&amp;gt;</td>\n",
"    </tr>\n",
"    <tr>\n",
```

```
"      <th>3</th>\n",
"      <td>r_LAX02LA036</td>\n",
"      <td>hasWindSpeed</td>\n",
"      <td>4</td>\n",
"      <td>&lt;extra_id_0&gt; r_LAX02LA036.</td>\n",
"      <td>&lt;extra_id_0&gt; hasWindSpeed 4</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>4</th>\n",
"      <td>r_LAX02LA074</td>\n",
"      <td>unitOfDewPoint</td>\n",
"      <td>degreeCelsius</td>\n",
"      <td>&lt;extra_id_0&gt; degreeCelsius.</td>\n",
"      <td>r_LAX02LA074 unitOfDewPoint &lt;extra_id_0&gt;</td>\n",
"    </tr>\n",
"  </tbody>\n",
"</table>\n",
"</div>"
],
"text/plain": [
"       subject                          relation          object  \\\n",
"0  r_FTW03LA001                                type  Accident_Number   \n",
"1       N366KR  hasEmergencyLocatorTransmitterInstalled           false   \n",
"2  r_CHI04LA052                 unitOfTemperature    degreeCelsius   \n",
"3  r_LAX02LA036                      hasWindSpeed                4   \n",
"4  r_LAX02LA074                     unitOfDewPoint    degreeCelsius   \n",
"\n",
"                      label  \\\n",
"0  <extra_id_0> Accident_Number.   \n",
"1         <extra_id_0> N366KR.   \n",
"2    <extra_id_0> degreeCelsius.   \n",
"3     <extra_id_0> r_LAX02LA036.   \n",
"4    <extra_id_0> degreeCelsius.   \n",
"\n",
"                                    input  \n",
"0                 r_FTW03LA001 type <extra_id_0>  \n",
"1  <extra_id_0> hasEmergencyLocatorTransmitterIns...  \n",
"2       r_CHI04LA052 unitOfTemperature <extra_id_0>  \n",
"3                    <extra_id_0> hasWindSpeed 4  \n",
"4          r_LAX02LA074 unitOfDewPoint <extra_id_0>  "
]
},
"execution_count": 54,
"metadata": {},
"output_type": "execute_result"
}
],
"source": [
"df.head()"
```

```
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 55,
  "metadata": {},
  "outputs": [
   {
    "data": {
     "text/plain": [
      "96625"
     ]
    },
    "execution_count": 55,
    "metadata": {},
    "output_type": "execute_result"
   }
  ],
  "source": [
   "len(df)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 56,
  "metadata": {},
  "outputs": [],
  "source": [
   "df =  df.sort_values(by=['subject', 'object'], ascending=True).reset_index(drop = True)\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 59,
  "metadata": {},
  "outputs": [
   {
    "ename": "AttributeError",
    "evalue": "Can't get attribute '_unpickle_block' on <module 'pandas._libs.internals' from
'/opt/conda/lib/python3.8/site-packages/pandas/_libs/internals.cpython-38-x86_64-linux-gnu.so'>",
    "output_type": "error",
    "traceback": [
     "\u001b[0;31m---------------------------------------------------------------------------\u001b[0m",
     "\u001b[0;31mAttributeError\u001b[0m                            Traceback (most recent call last)",
     "\u001b[0;32m<ipython-input-59-65d6248550cd>\u001b[0m in
\u001b[0;36m<module>\u001b[0;34m\u001b[0m\n\u001b[1;32m      1\u001b[0m
\u001b[0;32mimport\u001b[0m
\u001b[0mpickle\u001b[0m\u001b[0;34m\u001b[0m\u001b[0;34m\u001b[0m\n\u001b[1;32
m      2\u001b[0m \u001b[0mfile\u001b[0m \u001b[0;34m=\u001b[0m
```

\u001b[0mopen\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0;34m\"c4_masked.pickle\"\u001b[0m\u001b[0;34m,\u001b[0m\u001b[0;34m'rb'\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m\u001b[0m\n\u001b[0;32m----> 3\u001b[0;31m \u001b[0mc4df\u001b[0m \u001b[0;34m=\u001b[0m \u001b[0mpickle\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0mload\u001b[0;34m(\u001b[0mfile\u001b[0;34m)\u001b[0m\u001b[0;34m\u001b[0m\n\u001b[0;31m\u001b[1;32m   4\u001b[0m \u001b[0mfile\u001b[0;34m.\u001b[0m\u001b[0mclose\u001b[0;34m(\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m\u001b[0m\u001b[0;34m\u001b[0m\n",
      "\u001b[0;31mAttributeError\u001b[0m: Can't get attribute '_unpickle_block' on <module 'pandas._libs.internals' from '/opt/conda/lib/python3.8/site-packages/pandas/_libs/internals.cpython-38-x86_64-linux-gnu.so'>"
     ]
    }
   ],
   "source": [
    "import pickle\n",
    "file = open(\"c4_masked.pickle\",'rb')\n",
    "c4df = pickle.load(file)\n",
    "file.close()\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 29,
   "metadata": {},
   "outputs": [
    {
     "ename": "NameError",
     "evalue": "name 'c4df' is not defined",
     "output_type": "error",
     "traceback": [
      "\u001b[0;31m---\u001b[0m",
      "\u001b[0;31mNameError\u001b[0m   Traceback (most recent call last)",
      "\u001b[0;32m<ipython-input-29-eee9442f7f00>\u001b[0m in \u001b[0;36m<module>\u001b[0;34m\u001b[0m\n\u001b[0;32m----> 1\u001b[0;31m \u001b[0md1\u001b[0m \u001b[0;34m=\u001b[0m \u001b[0mc4df\u001b[0m\u001b[0;34m[\u001b[0m\u001b[0;34m[\u001b[0m\u001b[0;34m'input'\u001b[0m\u001b[0;34m,\u001b[0m\u001b[0;34m'label'\u001b[0m\u001b[0;34m]\u001b[0m\u001b[0;34m]\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0msample\u001b[0;34m(\u001b[0mfrac\u001b[0;34m=\u001b[0m\u001b[0;36m1\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0mreset_index\u001b[0;34m(\u001b[0mdrop\u001b[0m \u001b[0;34m=\u001b[0m \u001b[0;32mTrue\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0miloc\u001b[0;34m[\u001b[0m\u001b[0;34m:\u001b[0m\u001b[0mlen\u001b[0;34m(\u001b[0mdf\u001b[0;34m)\u001b[0m\u001b[0;34m]\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0mcopy\u001b[0;34m(\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m\u001b[0m\n\u001b[0;31m\u001b[1;32m   2\u001b[0m \u001b[0md2\u001b[0m \u001b[0;34m=\u001b[0m

\u001b[0mdf\u001b[0m\u001b[0;34m[\u001b[0m\u001b[0;34m[\u001b[0m\u001b[0;34m'input'\u001b[0m\u001b[0;34m,\u001b[0m
\u001b[0;34m'label'\u001b[0m\u001b[0;34m]\u001b[0m\u001b[0;34m]\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0mcopy\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m\u001b[0m\u001b[0;34m\u001b[0m\n",
      "\u001b[0;31mNameError\u001b[0m: name 'c4df' is not defined"
     ]
    }
   ],
   "source": [
    "d1 = c4df[['input', 'label']].sample(frac=1).reset_index(drop = True).iloc[:len(df)].copy()\n",
    "d2 = df[['input', 'label']].copy()\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 30,
   "metadata": {},
   "outputs": [
    {
     "ename": "NameError",
     "evalue": "name 'd1' is not defined",
     "output_type": "error",
     "traceback": [
      "\u001b[0;31m---\u001b[0m",
      "\u001b[0;31mNameError\u001b[0m   Traceback (most recent call last)",
      "\u001b[0;32m<ipython-input-30-2742e05f697c>\u001b[0m in
\u001b[0;36m<module>\u001b[0;34m\u001b[0m\n\u001b[0;32m----> 1\u001b[0;31m
\u001b[0md3\u001b[0m \u001b[0;34m=\u001b[0m
\u001b[0mpd\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0mconcat\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0;34m[\u001b[0m\u001b[0md1\u001b[0m\u001b[0;34m,\u001b[0m\u001b[0md2\u001b[0m\u001b[0;34m]\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m\u001b[0m\u001b[0;34m\u001b[0m\n\u001b[0m\u001b[1;32m   2\u001b[0m \u001b[0md3\u001b[0m
\u001b[0;34m=\u001b[0m
\u001b[0md3\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0msample\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0mfrac\u001b[0m\u001b[0;34m=\u001b[0m\u001b[0;36m1\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m.\u001b[0m\u001b[0mreset_index\u001b[0m\u001b[0;34m(\u001b[0m\u001b[0mdrop \u001b[0;34m=\u001b[0m
\u001b[0;32mTrue\u001b[0m\u001b[0;34m)\u001b[0m\u001b[0;34m\u001b[0m\u001b[0;34m\u001b[0m\n",
      "\u001b[0;31mNameError\u001b[0m: name 'd1' is not defined"
     ]
    }
   ],
   "source": [
    "d3 = pd.concat([d1,d2])\n",
    "d3 = d3.sample(frac=1).reset_index(drop = True)\n"
   ]
  },

```
{
 "cell_type": "code",
 "execution_count": 27,
 "metadata": {},
 "outputs": [
  {
   "data": {
    "text/plain": [
     "(96625, 96625, 193250)"
    ]
   },
   "execution_count": 27,
   "metadata": {},
   "output_type": "execute_result"
  }
 ],
 "source": [
  "len(d1), len(d2), len(d3)"
 ]
},
{
 "cell_type": "code",
 "execution_count": 28,
 "metadata": {},
 "outputs": [
  {
   "data": {
    "text/html": [
     "<div>\n",
     "<style scoped>\n",
     "    .dataframe tbody tr th:only-of-type {\n",
     "        vertical-align: middle;\n",
     "    }\n",
     "\n",
     "    .dataframe tbody tr th {\n",
     "        vertical-align: top;\n",
     "    }\n",
     "\n",
     "    .dataframe thead th {\n",
     "        text-align: right;\n",
     "    }\n",
     "</style>\n",
     "<table border=\"1\" class=\"dataframe\">\n",
     "  <thead>\n",
     "    <tr style=\"text-align: right;\">\n",
     "      <th></th>\n",
     "      <th>input</th>\n",
     "      <th>label</th>\n",
     "    </tr>\n",
```

```
 "  </thead>\n",
 "  <tbody>\n",
 "    <tr>\n",
 "      <th>0</th>\n",
 "      <td>Back by popular demand, see \"The Closest thing...</td>\n",
 "      <td>&lt;extra_id_0&gt; Milton Theatre.</td>\n",
 "    </tr>\n",
 "    <tr>\n",
 "      <th>1</th>\n",
 "      <td>More information on hiking in and around &lt;extr...</td>\n",
 "      <td>&lt;extra_id_0&gt; Vienna.</td>\n",
 "    </tr>\n",
 "    <tr>\n",
 "      <th>2</th>\n",
 "      <td>In the past, &lt;extra_id_0&gt; tourism focused on t...</td>\n",
 "      <td>&lt;extra_id_0&gt; Beijing.</td>\n",
 "    </tr>\n",
 "    <tr>\n",
 "      <th>3</th>\n",
 "      <td>This site is for managing the &lt;extra_id_0&gt; Res...</td>\n",
 "      <td>&lt;extra_id_0&gt; “.</td>\n",
 "    </tr>\n",
 "    <tr>\n",
 "      <th>4</th>\n",
 "      <td>The district court declined to grant a motion ...</td>\n",
 "      <td>&lt;extra_id_0&gt; Oppenheimer.</td>\n",
 "    </tr>\n",
 "  </tbody>\n",
 "</table>\n",
 "</div>"
],
"text/plain": [
 "                                      input  \\\n",
 "0  Back by popular demand, see \"The Closest thing...   \n",
 "1  More information on hiking in and around <extr...   \n",
 "2  In the past, <extra_id_0> tourism focused on t...   \n",
 "3  This site is for managing the <extra_id_0> Res...   \n",
 "4  The district court declined to grant a motion ...   \n",
 "\n",
 "                        label  \n",
 "0  <extra_id_0> Milton Theatre.  \n",
 "1         <extra_id_0> Vienna.  \n",
 "2        <extra_id_0> Beijing.  \n",
 "3             <extra_id_0> “.  \n",
 "4    <extra_id_0> Oppenheimer.  "
]
},
"execution_count": 28,
"metadata": {},
```

```
  "output_type": "execute_result"
 }
],
"source": [
 "d1.head()"
]
},
{
"cell_type": "code",
"execution_count": 29,
"metadata": {},
"outputs": [
 {
  "data": {
   "text/html": [
    "<div>\n",
    "<style scoped>\n",
    "    .dataframe tbody tr th:only-of-type {\n",
    "        vertical-align: middle;\n",
    "    }\n",
    "\n",
    "    .dataframe tbody tr th {\n",
    "        vertical-align: top;\n",
    "    }\n",
    "\n",
    "    .dataframe thead th {\n",
    "        text-align: right;\n",
    "    }\n",
    "</style>\n",
    "<table border=\"1\" class=\"dataframe\">\n",
    "  <thead>\n",
    "    <tr style=\"text-align: right;\">\n",
    "      <th></th>\n",
    "      <th>input</th>\n",
    "      <th>label</th>\n",
    "    </tr>\n",
    "  </thead>\n",
    "  <tbody>\n",
    "    <tr>\n",
    "      <th>0</th>\n",
    "      <td>&amp;lt;extra_id_0&amp;gt; comment</td>\n",
    "      <td>&amp;lt;extra_id_0&amp;gt; .</td>\n",
    "    </tr>\n",
    "    <tr>\n",
    "      <th>1</th>\n",
    "      <td>&amp;lt;extra_id_0&amp;gt; defintion A gust or wind gust is ...</td>\n",
    "      <td>&amp;lt;extra_id_0&amp;gt; .</td>\n",
    "    </tr>\n",
    "    <tr>\n",
```

```
    "      <th>2</th>\n",
    "      <td>&lt;extra_id_0&gt; IsCausedBecause ABRUPT</td>\n",
    "      <td>&lt;extra_id_0&gt; .</td>\n",
    "    </tr>\n",
    "    <tr>\n",
    "      <th>3</th>\n",
    "      <td>IsCausedBecause &lt;extra_id_0&gt;</td>\n",
    "      <td>&lt;extra_id_0&gt; ABRUPT.</td>\n",
    "    </tr>\n",
    "    <tr>\n",
    "      <th>4</th>\n",
    "      <td>&lt;extra_id_0&gt; IsCausedBecause ABRUPT</td>\n",
    "      <td>&lt;extra_id_0&gt; .</td>\n",
    "    </tr>\n",
    "  </tbody>\n",
    "</table>\n",
    "</div>"
   ],
   "text/plain": [
    "                                        input                label\n",
    "0                           <extra_id_0> comment         <extra_id_0> .\n",
    "1  <extra_id_0> defintion A gust or wind gust is ...        <extra_id_0> .\n",
    "2               <extra_id_0> IsCausedBecause ABRUPT        <extra_id_0> .\n",
    "3                     IsCausedBecause <extra_id_0>  <extra_id_0> ABRUPT.\n",
    "4               <extra_id_0> IsCausedBecause ABRUPT        <extra_id_0> ."
   ]
  },
  "execution_count": 29,
  "metadata": {},
  "output_type": "execute_result"
 }
],
"source": [
 "d2.head()"
]
},
{
"cell_type": "code",
"execution_count": 30,
"metadata": {},
"outputs": [
 {
  "data": {
   "text/html": [
    "<div>\n",
    "<style scoped>\n",
    "    .dataframe tbody tr th:only-of-type {\n",
    "        vertical-align: middle;\n",
    "    }\n",
```

```
"\n",
"    .dataframe tbody tr th {\n",
"        vertical-align: top;\n",
"    }\n",
"\n",
"    .dataframe thead th {\n",
"        text-align: right;\n",
"    }\n",
"</style>\n",
"<table border=\"1\" class=\"dataframe\">\n",
"  <thead>\n",
"    <tr style=\"text-align: right;\">\n",
"      <th></th>\n",
"      <th>input</th>\n",
"      <th>label</th>\n",
"    </tr>\n",
"  </thead>\n",
"  <tbody>\n",
"    <tr>\n",
"      <th>0</th>\n",
"      <td>N515KH hasEmergencyLocatorTransmitterInstalled...</td>\n",
"      <td>&lt;extra_id_0&gt; false.</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>1</th>\n",
"      <td>r_MIA05LA036 occurredAtLatitude &lt;extra_id_0&gt;</td>\n",
"      <td>&lt;extra_id_0&gt; 30.6375.</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>2</th>\n",
"      <td>Last minute prep for &lt;extra_id_0&gt;.</td>\n",
"      <td>&lt;extra_id_0&gt; Winter Storm Jonas.</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>3</th>\n",
"      <td>&lt;extra_id_0&gt; was born August 7, 1958, to paren...</td>\n",
"      <td>&lt;extra_id_0&gt; Danny Scott Easterling.</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>4</th>\n",
"      <td>r_DEN06MA119 IsCausedBy &lt;extra_id_0&gt;</td>\n",
"      <td>&lt;extra_id_0&gt; WINGSPAR.</td>\n",
"    </tr>\n",
"  </tbody>\n",
"</table>\n",
"</div>"
],
"text/plain": [
"                                      input  \\\n",
```

```
    "0  N515KH hasEmergencyLocatorTransmitterInstalled...   \n",
    "1       r_MIA05LA036 occurredAtLatitude <extra_id_0>   \n",
    "2               Last minute prep for <extra_id_0>.   \n",
    "3  <extra_id_0> was born August 7, 1958, to paren...   \n",
    "4             r_DEN06MA119 IsCausedBy <extra_id_0>   \n",
    "\n",
    "                              label  \n",
    "0               <extra_id_0> false.  \n",
    "1              <extra_id_0> 30.6375.  \n",
    "2     <extra_id_0> Winter Storm Jonas.  \n",
    "3  <extra_id_0> Danny Scott Easterling.  \n",
    "4              <extra_id_0> WINGSPAR.  "
   ]
   },
   "execution_count": 30,
   "metadata": {},
   "output_type": "execute_result"
  }
 ],
 "source": [
  "d3.head()"
 ]
},
{
 "cell_type": "code",
 "execution_count": 31,
 "metadata": {},
 "outputs": [],
 "source": [
  "train_data = d3.copy()"
 ]
},
{
 "cell_type": "code",
 "execution_count": 32,
 "metadata": {},
 "outputs": [
  {
   "name": "stderr",
   "output_type": "stream",
   "text": [
    "/opt/conda/lib/python3.8/site-packages/transformers/models/t5/tokenization_t5.py:164: FutureWarning: This tokenizer was incorrectly instantiated with a model max length of 512 which will be corrected in Transformers v5.\n",
    "For now, this behavior is kept to avoid breaking backwards compatibility when padding/encoding with `truncation is True`.\n",
    "- Be aware that you SHOULD NOT rely on t5-large automatically truncating your input to 512 when padding/encoding.\n",
```

```
    "- If you want to encode/pad to sequences longer than 512 you can either instantiate this tokenizer with `model_max_length` or pass `max_length` when encoding/padding.\n",
    "- To avoid this warning, please instantiate this tokenizer with `model_max_length` set to your preferred value.\n",
    "  warnings.warn(\n"
   ]
  }
 ],
 "source": [
  "tokenizer = T5Tokenizer.from_pretrained(\"t5-large\")\n",
  "model = T5ForConditionalGeneration.from_pretrained(\"t5-large\")\n",
  "# model = T5ForConditionalGeneration.from_pretrained(\"/workspace/tanu/BTP-2/exp/knowledge infusion/trained models/KGinfusedLM/6\")\n",
  "# config = T5Config(dropout_rate  = 0.1)  # default value only, do we need to set it explicitly\n",
  "# model = T5ForConditionalGeneration.config(config).from_pretrained(\"t5-large\")\n",
  "\n"
 ]
},
{
 "cell_type": "code",
 "execution_count": 33,
 "metadata": {},
 "outputs": [],
 "source": [
  "def tokenize(text):\n",
  "    return tokenizer(text, return_tensors=\"pt\",  padding='longest', truncation=True )\n",
  "def tokenize_target(text):\n",
  "    return tokenizer(text, return_tensors=\"pt\",   padding=\"longest\",  truncation=True).input_ids"
 ]
},
{
 "cell_type": "code",
 "execution_count": 34,
 "metadata": {},
 "outputs": [],
 "source": [
  "optimizer = Adafactor(model.parameters(), lr=1e-3, relative_step = False)\n",
  "# parameters as specified in the paper\n",
  "# num_epochs = 380\n",
  "# batch_size = 1024\n",
  "num_epochs = 25\n",
  "batch_size = 16\n",
  "num_training_steps = num_epochs * (train_data.shape[0] // batch_size )\n"
 ]
},
{
 "cell_type": "code",
 "execution_count": 35,
 "metadata": {},
```

```
"outputs": [],
"source": [
 "import os\n",
 "os.environ[\"CUDA_DEVICE_ORDER\"]=\"PCI_BUS_ID\"\n",
 "os.environ[\"CUDA_VISIBLE_DEVICES\"]=\"1\"\n"
]
},
{
"cell_type": "code",
"execution_count": 36,
"metadata": {},
"outputs": [],
"source": [
 "# model = torch.nn.DataParallel(model, device_ids=[0, 1])\n",
 "# device = torch.device(\"cuda:1\") if torch.cuda.is_available() else torch.device(\"cpu\")\n",
 "# device"
]
},
{
"cell_type": "code",
"execution_count": 37,
"metadata": {},
"outputs": [
 {
  "data": {
   "text/plain": [
    "device(type='cuda')"
   ]
  },
  "execution_count": 37,
  "metadata": {},
  "output_type": "execute_result"
 }
],
"source": [
 "device = torch.device(\"cuda\") if torch.cuda.is_available() else torch.device(\"cpu\")\n",
 "model.to(device)\n",
 "device"
]
},
{
"cell_type": "code",
"execution_count": 38,
"metadata": {},
"outputs": [
 {
  "data": {
   "text/html": [
    "<div>\n",
```

```
"<style scoped>\n",
"    .dataframe tbody tr th:only-of-type {\n",
"        vertical-align: middle;\n",
"    }\n",
"\n",
"    .dataframe tbody tr th {\n",
"        vertical-align: top;\n",
"    }\n",
"\n",
"    .dataframe thead th {\n",
"        text-align: right;\n",
"    }\n",
"</style>\n",
"<table border=\"1\" class=\"dataframe\">\n",
"  <thead>\n",
"    <tr style=\"text-align: right;\">\n",
"      <th></th>\n",
"      <th>input</th>\n",
"      <th>label</th>\n",
"    </tr>\n",
"  </thead>\n",
"  <tbody>\n",
"    <tr>\n",
"      <th>0</th>\n",
"      <td>N515KH hasEmergencyLocatorTransmitterInstalled...</td>\n",
"      <td>&lt;extra_id_0&gt; false.</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>1</th>\n",
"      <td>r_MIA05LA036 occurredAtLatitude &lt;extra_id_0&gt;</td>\n",
"      <td>&lt;extra_id_0&gt; 30.6375.</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>2</th>\n",
"      <td>Last minute prep for &lt;extra_id_0&gt;.</td>\n",
"      <td>&lt;extra_id_0&gt; Winter Storm Jonas.</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>3</th>\n",
"      <td>&lt;extra_id_0&gt; was born August 7, 1958, to paren...</td>\n",
"      <td>&lt;extra_id_0&gt; Danny Scott Easterling.</td>\n",
"    </tr>\n",
"    <tr>\n",
"      <th>4</th>\n",
"      <td>r_DEN06MA119 IsCausedBy &lt;extra_id_0&gt;</td>\n",
"      <td>&lt;extra_id_0&gt; WINGSPAR.</td>\n",
"    </tr>\n",
"  </tbody>\n",
"</table>\n",
```

```
    "</div>"
   ],
   "text/plain": [
    "                                        input  \\\n",
    "0  N515KH hasEmergencyLocatorTransmitterInstalled...   \n",
    "1       r_MIA05LA036 occurredAtLatitude <extra_id_0>   \n",
    "2                 Last minute prep for <extra_id_0>.   \n",
    "3  <extra_id_0> was born August 7, 1958, to paren...   \n",
    "4              r_DEN06MA119 IsCausedBy <extra_id_0>   \n",
    "\n",
    "                                label  \n",
    "0                 <extra_id_0> false.  \n",
    "1                <extra_id_0> 30.6375.  \n",
    "2      <extra_id_0> Winter Storm Jonas.  \n",
    "3  <extra_id_0> Danny Scott Easterling.  \n",
    "4               <extra_id_0> WINGSPAR.  "
   ]
  },
  "execution_count": 38,
  "metadata": {},
  "output_type": "execute_result"
 }
],
"source": [
 "train_data.head()"
]
},
{
"cell_type": "code",
"execution_count": 39,
"metadata": {},
"outputs": [
 {
  "data": {
   "text/plain": [
    "193250"
   ]
  },
  "execution_count": 39,
  "metadata": {},
  "output_type": "execute_result"
 }
],
"source": [
 "len(train_data)"
]
},
{
"cell_type": "code",
```

```
"execution_count": 40,
"metadata": {},
"outputs": [],
"source": [
 "def run_data(model, data, batch_size, optimizer, tokenizer, device, eval_mode):\n",
 "    iters = int(np.ceil(data.shape[0] / batch_size))\n",
 "    avg_loss = 0\n",
 "    step = 0\n",
 "    p_bar = tqdm(total=iters, position=0, leave=True, desc='Running through data')\n",
 "    for row_idx in range(0, data.shape[0], batch_size):\n",
 "        upper_idx = min(row_idx + batch_size, data.shape[0]) -1\n",
 "        \n",
 "        labels = data.loc[row_idx : upper_idx]['label'].tolist()\n",
 "        inputs = data.loc[row_idx : upper_idx]['input'].tolist()\n",
 "        tokenized_labels = tokenize_target(labels)\n",
 "        tokenized_input = tokenize(inputs)\n",
 "\n",
 "        input_ids = tokenized_input[\"input_ids\"].to(device)\n",
 "        attention_mask = tokenized_input[\"attention_mask\"].to(device)\n",
 "        labels = tokenized_labels.to(device)\n",
 "        labels[labels == tokenizer.pad_token_id] = -100\n",
 "        loss = model(input_ids= input_ids, attention_mask= attention_mask,labels= labels).loss\n",
 "        loss.backward()\n",
 "        optimizer.step()\n",
 "        optimizer.zero_grad()\n",
 "        loss_item = loss.detach().clone().item()\n",
 "        avg_loss = (avg_loss * step + loss_item) / (step + 1)\n",
 "\n",
 "        p_bar.set_postfix(avg_loss=avg_loss)\n",
 "        p_bar.update(1)\n",
 "        step += 1\n",
 "\n",
 "    p_bar.close()\n",
 "    return model, optimizer, avg_loss"
]
},
{
"cell_type": "code",
"execution_count": 41,
"metadata": {},
"outputs": [],
"source": [
 "torch.cuda.empty_cache()\n"
]
},
{
"cell_type": "code",
"execution_count": 42,
"metadata": {},
```

```
  "outputs": [],
  "source": [
   "to_save_epochs = [1,2,5,10,15,20,25]\n",
   "# to_save_epochs = [0,1,4,9,14,19,24]"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [
   {
    "name": "stderr",
    "output_type": "stream",
    "text": [
     "Running through data:  26%|□| 3082/12079 [14:35<43:44,  3.43it/s, avg_loss=1.45]"
    ]
   }
  ],
  "source": [
   "train_losses = []     \n",
   "for epoch in range(0,num_epochs):\n",
   "        shuffled_train_data = train_data.sample(frac=1).reset_index() \n",
   "        model.train()\n",
   "        optimizer.zero_grad() \n",
   "        model, optimizer, avg_train_loss = run_data(model, shuffled_train_data, batch_size, \\\n",
   "                                optimizer, tokenizer, device, eval_mode=False)\n",
   "        train_losses.append(avg_train_loss)\n",
   "        if epoch in to_save_epochs:\n",
   "            model.save_pretrained(f\"aviation/trained models/KGinfusedLM/{epoch}\", from_pt=True)
\n",
   "        print(f'Epoch {epoch}:\\tTrain loss: {avg_train_loss}')\n",
   "\n"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "train_losses"
  ]
 }
],
"metadata": {
 "kernelspec": {
  "display_name": "Python 3",
  "language": "python",
```

```
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.8.5"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

```
import regex as re
from colbert.data import Queries
from colbert.infra import Run, RunConfig, ColBERTConfig
from colbert import Searcher
import pandas as pd
from tqdm import tqdm
import itertools
hops = ['2hop', '3hop']
splits = ["dev", "test", "train"]
# HOP = 1
TOP_K = 3
all_entities = [word.lower() for word in pd.read_csv("../data/wikimovies/entities.txt",
                                            sep="\t", header=0, names=["id", "entity"])["entity"].to_list()]

with Run().context(RunConfig(experiment='metaqa')):
    searcher = Searcher(index='wikimovies.nbits=2')


def query_colbert_fancier(query, searcher=searcher, k=5, hops=1, limit=20):
    visited_entities = [[re.search('\[(.*)\]', query).group(1).lower()]]
    all_visited_entities = []
    all_visited_entities.append(visited_entities[0][0])

    visited_pids = []
    og_query = query
    # print(visited_entities)
    for hop in range(hops):
        curr_passages = []
        results = searcher.search(query, k=k)
```

```
        curr = []
        for id, (passage_id, passage_rank, passage_score) in enumerate(zip(*results)):
            if passage_id not in visited_pids:
                visited_pids.append(passage_id)
                curr_passage = " "+searcher.collection[passage_id][:-1]+" ."
                # print(hop+1, curr_passage)
                if any(" "+entity+" " in curr_passage for entity in visited_entities[-1]):
                    # print("out",hop+1, curr_passage)
                    curr_passages.append(curr_passage)
                    # query+=curr_passage
                    for entity in all_entities:
                        if " "+entity+" " in curr_passage and entity not in visited_entities[-1] and not any(entity
in enTT for enTT in all_visited_entities):
                            curr.append(entity)
        if len(curr) == 0:
            results = searcher.search(query, k=limit)
            for id, (passage_id, passage_rank, passage_score) in enumerate(zip(*results)):
                if passage_id not in visited_pids:
                    visited_pids.append(passage_id)
                    curr_passage = " " + \
                        searcher.collection[passage_id][:-1]+" ."
                    # print(hop+1, curr_passage)
                    if any(" "+entity+" " in curr_passage for entity in visited_entities[-1]):
                        # print("in",hop+1, curr_passage)
                        curr_passages.append(curr_passage)
                        for entity in all_entities:
                            if " "+entity+" " in curr_passage and entity not in visited_entities[-1] and not
any(entity in enTT for enTT in all_visited_entities):
                                curr.append(entity)
        k *= 2
        # print(visited_entities)
        # print(query)
        visited_entities.append(list(set(curr)))
        all_visited_entities.extend(curr)
        query += ".join(curr_passages)
    return str.strip(query.replace(og_query, ""))
    # return [searcher.collection[passage_id] for passage_id in visited_pids]


for hop in hops:
    for split in splits:
        questions = []
        answers = []
        contexts = []
        # qas = open("../data/metaqa/"+hop+"/qa_"+split +
        #            ".txt", 'r').read().splitlines()
        qas = pd.read_csv("../data/metaqa/"+hop+"/qa_"+split +
                          ".txt", sep='\t', header=0, names=["query", 'answer'])
        # queries = qas["query"].to_list()
```

```
    # anss = qas["answer"].to_list()
    for ind in tqdm(qas.index, desc=hop+" and "+split):
        # for index, row in tqdm(qas.iterrows(), desc=hop+" and "+split):
        query, ans = qas['query'][ind], qas['answer'][ind]
        top_k_passages = query_colbert_fancier(
            query, hops=int(hop[0]), searcher=searcher, k=TOP_K)
        context = top_k_passages.replace("\t", " ")
        questions.append(query)
        answers.append(ans)
        contexts.append(context)
        # if ind == 20:
        #     break
    details = {
        'qid': [a for a in range(len(questions))],
        'question': questions,
        'answer': answers,
        'context': contexts,
    }

    # creating a Dataframe object with skipping
    # one column i.e skipping age column.
    df = pd.DataFrame(details, columns=[
                    'qid', 'question', 'answer', 'context'])
    df.to_csv("../data/metaqa/"+hop+"/qa_"+split+"_triples_multitop"+str(TOP_K)+".tsv",
              index=False, sep='\t', header=False)
```

continual_pretraining.ipynb: Code to continually pretrain using salient span masking
query_metaqamultiT2.py: Code for Question Booster
run_seq2seq_cbqa.py: To finetune on closed book QA i.e. without context
run_seq2seq_qa.py: To finetune with context
trainer_seq2seq_cbqa.py: Allows to change functions of trainer for run_seq2seq_cbqa
trainer_seq2seq_qa.py: Allows to change functions of trainer for run_seq2seq_cbqa
Fine tune code is from:
https://github.com/huggingface/transformers/tree/main/examples/pytorch/question-answering and instructions to run are available there.
Example Commands:
CUDA_VISIBLE_DEVICES=3,5 python run_seq2seq_cbqa.py --model_name_or_path ../SKILL/aviation/trained_models/T5_large_with_MetaTriples/20 --train_file data/metaqa/3hop/qa_train_triples.csv --validation_file data/metaqa/3hop/qa_dev_triples.csv --test_file data/metaqa/3hop/qa_test_triples.csv --preprocessing_num_workers 10 --question_column question --answer_column answer --do_train --per_device_train_batch_size 128 --per_device_eval_batch_size 32 --num_train_epochs 20 --max_seq_length 128 --doc_stride 128 --output_dir models/metaqa_3hop_kg_unverb_20/ --save_steps 1000 --seed 42 --overwrite_output_dir --report_to wandb --logging_steps 100 --learning_rate 1e-3 --tokenizer_name t5-large

CUDA_VISIBLE_DEVICES=0 python run_seq2seq_qa.py --model_name_or_path t5-large --train_file data/metaQA/2hop/qa_train_triples_multitop3.csv --validation_file data/metaQA/2hop/qa_dev_triples_multitop3.csv --test_file

data/metaQA/2hop/qa_test_triples_multitop3.csv --overwrite_cache --preprocessing_num_workers 10 --question_column question --context_column context --answer_column answer --do_train --per_device_train_batch_size 128 --per_device_eval_batch_size 32 --num_train_epochs 20 --max_seq_length 128 --doc_stride 128 --output_dir models/metaqa_2hop_top3_COLBERT_multihop/ --save_steps 200 --seed 42 --overwrite_output_dir --report_to wandb --logging_steps 100 --learning_rate 1e-3 --tokenizer_name t5-large

```
#!/usr/bin/env python
# coding=utf-8
# Copyright 2021 The HuggingFace Team All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
# s
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
"""
Fine-tuning the library's seq2seq models for question answering using the □ Seq2SeqTrainer.
"""
# You can also adapt this script on your own question answering task. Pointers for this are left as comments.

import logging
import os
import sys
from dataclasses import dataclass, field
from typing import List, Optional, Tuple
from transformers.optimization import Adafactor  # , AdafactorSchedule
import datasets
import evaluate
from datasets import load_dataset
from trainer_seq2seq_qa import QuestionAnsweringSeq2SeqTrainer

import transformers
from transformers import (
    AutoConfig,
    AutoModelForSeq2SeqLM,
    AutoTokenizer,
    DataCollatorForSeq2Seq,
    HfArgumentParser,
    Seq2SeqTrainingArguments,
    set_seed,
```

```
)
from transformers.trainer_utils import EvalLoopOutput, EvalPrediction, get_last_checkpoint
from transformers.utils import check_min_version, send_example_telemetry
from transformers.utils.versions import require_version


# Will error if the minimal version of Transformers is not installed. Remove at your own risks.
check_min_version("4.28.0.dev0")

require_version("datasets>=1.8.0",
                "To fix: pip install -r examples/pytorch/question-answering/requirements.txt")

logger = logging.getLogger(__name__)


@dataclass
class ModelArguments:
    """
    Arguments pertaining to which model/config/tokenizer we are going to fine-tune from.
    """

    model_name_or_path: str = field(
        metadata={
            "help": "Path to pretrained model or model identifier from huggingface.co/models"}
    )
    config_name: Optional[str] = field(
        default=None, metadata={"help": "Pretrained config name or path if not the same as
model_name"}
    )
    tokenizer_name: Optional[str] = field(
        default=None, metadata={"help": "Pretrained tokenizer name or path if not the same as
model_name"}
    )
    cache_dir: Optional[str] = field(
        default=None,
        metadata={
            "help": "Path to directory to store the pretrained models downloaded from huggingface.co"},
    )
    use_fast_tokenizer: bool = field(
        default=True,
        metadata={
            "help": "Whether to use one of the fast tokenizer (backed by the tokenizers library) or not."},
    )
    model_revision: str = field(
        default="main",
        metadata={
            "help": "The specific model version to use (can be a branch name, tag name or commit id)."},
    )
    use_auth_token: bool = field(
```

```
        default=False,
        metadata={
            "help": (
                "Will use the token generated when running `huggingface-cli login` (necessary to use this
script "
                "with private models)."
            )
        },
    )


@dataclass
class DataTrainingArguments:
    """
    Arguments pertaining to what data we are going to input our model for training and eval.
    """

    dataset_name: Optional[str] = field(
        default=None, metadata={"help": "The name of the dataset to use (via the datasets library)."}
    )
    dataset_config_name: Optional[str] = field(
        default=None, metadata={"help": "The configuration name of the dataset to use (via the datasets
library)."}
    )
    question_column: Optional[str] = field(
        default="question",
        metadata={
            "help": "The name of the column in the datasets containing the questions (for question
answering)."},
    )
    answer_column: Optional[str] = field(
        default="answers",
        metadata={
            "help": "The name of the column in the datasets containing the answers (for question
answering)."},
    )
    train_file: Optional[str] = field(
        default=None, metadata={"help": "The input training data file (a text file)."})
    validation_file: Optional[str] = field(
        default=None,
        metadata={
            "help": "An optional input evaluation data file to evaluate the perplexity on (a text file)."},
    )
    test_file: Optional[str] = field(
        default=None,
        metadata={
            "help": "An optional input test data file to evaluate the perplexity on (a text file)."},
    )
    overwrite_cache: bool = field(
```

```
        default=False, metadata={"help": "Overwrite the cached training and evaluation sets"}
    )
    preprocessing_num_workers: Optional[int] = field(
        default=None,
        metadata={"help": "The number of processes to use for the preprocessing."},
    )
    max_seq_length: int = field(
        default=384,
        metadata={
            "help": (
                "The maximum total input sequence length after tokenization. Sequences longer "
                "than this will be truncated, sequences shorter will be padded."
            )
        },
    )
    max_answer_length: int = field(
        default=30,
        metadata={
            "help": (
                "The maximum length of an answer that can be generated. This is needed because the start "
                "and end predictions are not conditioned on one another."
            )
        },
    )
    val_max_answer_length: Optional[int] = field(
        default=None,
        metadata={
            "help": (
                "The maximum total sequence length for validation target text after tokenization. Sequences
longer "
                "than this will be truncated, sequences shorter will be padded. Will default to
`max_answer_length`."
                "This argument is also used to override the ``max_length`` param of ``model.generate``,
which is used "
                "during ``evaluate`` and ``predict``."
            )
        },
    )
    pad_to_max_length: bool = field(
        default=True,
        metadata={
            "help": (
                "Whether to pad all samples to `max_seq_length`. If False, will pad the samples dynamically
when"
                " batching to the maximum length in the batch (which can be faster on GPU but will be
slower on TPU)."
            )
        },
    )
```

```
    max_train_samples: Optional[int] = field(
        default=None,
        metadata={
            "help": (
                "For debugging purposes or quicker training, truncate the number of training examples to this
"
                "value if set."
            )
        },
    )
    max_eval_samples: Optional[int] = field(
        default=None,
        metadata={
            "help": (
                "For debugging purposes or quicker training, truncate the number of evaluation examples to
this "
                "value if set."
            )
        },
    )
    max_predict_samples: Optional[int] = field(
        default=None,
        metadata={
            "help": (
                "For debugging purposes or quicker training, truncate the number of prediction examples to
this "
                "value if set."
            )
        },
    )
    version_2_with_negative: bool = field(
        default=False, metadata={"help": "If true, some of the examples do not have an answer."}
    )
    null_score_diff_threshold: float = field(
        default=0.0,
        metadata={
            "help": (
                "The threshold used to select the null answer: if the best answer has a score that is less than "
                "the score of the null answer minus this threshold, the null answer is selected for this
example. "
                "Only useful when `version_2_with_negative=True`."
            )
        },
    )
    doc_stride: int = field(
        default=128,
        metadata={
            "help": "When splitting up a long document into chunks, how much stride to take between
chunks."},
```

```
    )
    n_best_size: int = field(
        default=20,
        metadata={
            "help": "The total number of n-best predictions to generate when looking for an answer."},
    )
    num_beams: Optional[int] = field(
        default=None,
        metadata={
            "help": (
                "Number of beams to use for evaluation. This argument will be passed to ``model.generate``,
"
                "which is used during ``evaluate`` and ``predict``."
            )
        },
    )
    ignore_pad_token_for_loss: bool = field(
        default=True,
        metadata={
            "help": "Whether to ignore the tokens corresponding to padded labels in the loss computation or
not."
        },
    )

    def __post_init__(self):
        if (
            self.dataset_name is None
            and self.train_file is None
            and self.validation_file is None
            and self.test_file is None
        ):
            raise ValueError(
                "Need either a dataset name or a training/validation file/test_file.")
        else:
            if self.train_file is not None:
                extension = self.train_file.split(".")[-1]
                assert extension in [
                    "csv", "json"], "`train_file` should be a csv or a json file."
            if self.validation_file is not None:
                extension = self.validation_file.split(".")[-1]
                assert extension in [
                    "csv", "json"], "`validation_file` should be a csv or a json file."
            if self.test_file is not None:
                extension = self.test_file.split(".")[-1]
                assert extension in [
                    "csv", "json"], "`test_file` should be a csv or a json file."
        if self.val_max_answer_length is None:
            self.val_max_answer_length = self.max_answer_length
```

```
question_answering_column_name_mapping = {
    "squad_v2": ("question", "context", "answer"),
    "sakharamg/AviationQA": ("Question", "", "Answer"),
    "sakharamg/metaQA": ("question", "", "answer"),
    "sakharamg/AeroQA": ("question", "", "answer"),
}


def main():
    # See all possible arguments in src/transformers/training_args.py
    # or by passing the --help flag to this script.
    # We now keep distinct sets of args, for a cleaner separation of concerns.

    parser = HfArgumentParser(
        (ModelArguments, DataTrainingArguments, Seq2SeqTrainingArguments))
    if len(sys.argv) == 2 and sys.argv[1].endswith(".json"):
        # If we pass only one argument to the script and it's the path to a json file,
        # let's parse it to get our arguments.
        model_args, data_args, training_args = parser.parse_json_file(
            json_file=os.path.abspath(sys.argv[1]))
    else:
        model_args, data_args, training_args = parser.parse_args_into_dataclasses()

    # Sending telemetry. Tracking the example usage helps us better allocate resources to maintain them.
The
    # information sent is the one passed as arguments along with your Python/PyTorch versions.
    send_example_telemetry("run_seq2seq_qa", model_args, data_args)

    # Setup logging
    logging.basicConfig(
        format="%(asctime)s - %(levelname)s - %(name)s - %(message)s",
        datefmt="%m/%d/%Y %H:%M:%S",
        handlers=[logging.StreamHandler(sys.stdout)],
    )

    if training_args.should_log:
        # The default of training_args.log_level is passive, so we set log level at info here to have that
default.
        transformers.utils.logging.set_verbosity_info()

    log_level = training_args.get_process_log_level()
    logger.setLevel(log_level)
    datasets.utils.logging.set_verbosity(log_level)
    transformers.utils.logging.set_verbosity(log_level)
    transformers.utils.logging.enable_default_handler()
    transformers.utils.logging.enable_explicit_format()

    # Log on each process the small summary:
```

```
    logger.warning(
        f"Process rank: {training_args.local_rank}, device: {training_args.device}, n_gpu:
{training_args.n_gpu}"
        + f"distributed training: {bool(training_args.local_rank != -1)}, 16-bits training:
{training_args.fp16}"
    )
    logger.info(f"Training/evaluation parameters {training_args}")

    # Detecting last checkpoint.
    last_checkpoint = None
    if os.path.isdir(training_args.output_dir) and training_args.do_train and not
training_args.overwrite_output_dir:
        last_checkpoint = get_last_checkpoint(training_args.output_dir)
        if last_checkpoint is None and len(os.listdir(training_args.output_dir)) > 0:
            raise ValueError(
                f"Output directory ({training_args.output_dir}) already exists and is not empty. "
                "Use --overwrite_output_dir to overcome."
            )
        elif last_checkpoint is not None and training_args.resume_from_checkpoint is None:
            logger.info(
                f"Checkpoint detected, resuming training at {last_checkpoint}. To avoid this behavior,
change "
                "the `--output_dir` or add `--overwrite_output_dir` to train from scratch."
            )

    # Set seed before initializing model.
    set_seed(training_args.seed)

    # Get the datasets: you can either provide your own CSV/JSON/TXT training and evaluation files
(see below)
    # or just provide the name of one of the public datasets available on the hub at
https://huggingface.co/datasets/
    # (the dataset will be downloaded automatically from the datasets Hub).
    #
    # For CSV/JSON files, this script will use the column called 'text' or the first column if no column
called
    # 'text' is found. You can easily tweak this behavior (see below).
    #
    # In distributed training, the load_dataset function guarantee that only one local process can
concurrently
    # download the dataset.
    if data_args.dataset_name is not None:
        # Downloading and loading a dataset from the hub.
        raw_datasets = load_dataset(
            data_args.dataset_name,
            data_args.dataset_config_name,
            cache_dir=model_args.cache_dir,
            use_auth_token=True if model_args.use_auth_token else None,
        )
```

```
    else:
        data_files = {}
        if data_args.train_file is not None:
            data_files["train"] = data_args.train_file
            extension = data_args.train_file.split(".")[-1]
        if data_args.validation_file is not None:
            data_files["validation"] = data_args.validation_file
            extension = data_args.validation_file.split(".")[-1]
        if data_args.test_file is not None:
            data_files["test"] = data_args.test_file
            extension = data_args.test_file.split(".")[-1]
        raw_datasets = load_dataset(
            extension,
            data_files=data_files,
            cache_dir=model_args.cache_dir,
            use_auth_token=True if model_args.use_auth_token else None,
        )
    # See more about loading any type of standard or custom dataset (from files, python dict, pandas
DataFrame, etc) at
    # https://huggingface.co/docs/datasets/loading_datasets.html.

    # Load pretrained model and tokenizer
    #
    # Distributed training:
    # The .from_pretrained methods guarantee that only one local process can concurrently
    # download model & vocab.
    config = AutoConfig.from_pretrained(
        model_args.config_name if model_args.config_name else model_args.model_name_or_path,
        cache_dir=model_args.cache_dir,
        revision=model_args.model_revision,
        use_auth_token=True if model_args.use_auth_token else None,
    )
    tokenizer = AutoTokenizer.from_pretrained(
        model_args.tokenizer_name if model_args.tokenizer_name else
model_args.model_name_or_path,
        cache_dir=model_args.cache_dir,
        use_fast=model_args.use_fast_tokenizer,
        revision=model_args.model_revision,
        use_auth_token=True if model_args.use_auth_token else None,
    )
    model = AutoModelForSeq2SeqLM.from_pretrained(
        model_args.model_name_or_path,
        from_tf=bool(".ckpt" in model_args.model_name_or_path),
        config=config,
        cache_dir=model_args.cache_dir,
        revision=model_args.model_revision,
        use_auth_token=True if model_args.use_auth_token else None,
    )
```

```
    # We resize the embeddings only when necessary to avoid index errors. If you are creating a model
from scratch
    # on a small vocab and want a smaller embedding size, remove this test.
    embedding_size = model.get_input_embeddings().weight.shape[0]
    if len(tokenizer) > embedding_size:
        model.resize_token_embeddings(len(tokenizer))

    if model.config.decoder_start_token_id is None:
        raise ValueError(
            "Make sure that `config.decoder_start_token_id` is correctly defined")

    # Preprocessing the datasets.
    # We need to generate and tokenize inputs and targets.
    if training_args.do_train:
        column_names = raw_datasets["train"].column_names
    elif training_args.do_eval:
        column_names = raw_datasets["validation"].column_names
    elif training_args.do_predict:
        column_names = raw_datasets["test"].column_names
    else:
        logger.info(
            "There is nothing to do. Please pass `do_train`, `do_eval` and/or `do_predict`.")
        return

    # Get the column names for input/target.
    dataset_columns = question_answering_column_name_mapping.get(
        "sakharamg/AeroQA", None)
    if data_args.question_column is None:
        question_column = dataset_columns[0] if dataset_columns is not None else column_names[0]
    else:
        question_column = data_args.question_column
        if question_column not in column_names:
            raise ValueError(
                f"--question_column' value '{data_args.question_column}' needs to be one of: {',
'.join(column_names)}"
            )

    if data_args.answer_column is None:
        answer_column = dataset_columns[2] if dataset_columns is not None else column_names[2]
    else:
        answer_column = data_args.answer_column
        if answer_column not in column_names:
            raise ValueError(
                f"--answer_column' value '{data_args.answer_column}' needs to be one of: {',
'.join(column_names)}"
            )

    # Temporarily set max_answer_length for training.
    max_answer_length = data_args.max_answer_length
```

```
    padding = "max_length" if data_args.pad_to_max_length else False

    if training_args.label_smoothing_factor > 0 and not hasattr(model,
"prepare_decoder_input_ids_from_labels"):
        logger.warning(
            "label_smoothing is enabled but the `prepare_decoder_input_ids_from_labels` method is not
defined for"
            f"`{model.__class__.__name__}`. This will lead to loss being calculated twice and will take up
more memory"
        )

    if data_args.max_seq_length > tokenizer.model_max_length:
        logger.warning(
            f"The max_seq_length passed ({data_args.max_seq_length}) is larger than the maximum length
for the"
            f"model ({tokenizer.model_max_length}). Using
max_seq_length={tokenizer.model_max_length}."
        )
    max_seq_length = min(data_args.max_seq_length, tokenizer.model_max_length)

    def preprocess_squad_batch(
        examples,
        question_column: str,
        answer_column: str,
    ) -> Tuple[List[str], List[str]]:
        questions = examples[question_column]
        answers = examples[answer_column]

        def generate_input(_question):
            return " ".join(["question:", _question.lstrip()])

        inputs = [generate_input(question) for question in questions]
        targets = [answer if answer != None else "None" for answer in answers]
        return inputs, targets

    def preprocess_function(examples):
        inputs, targets = preprocess_squad_batch(
            examples, question_column, answer_column)

        model_inputs = tokenizer(
            inputs, max_length=max_seq_length, padding=padding, truncation=True)
        # Tokenize targets with text_target=...
        labels = tokenizer(
            text_target=targets, max_length=max_answer_length, padding=padding, truncation=True)

        # If we are padding here, replace all tokenizer.pad_token_id in the labels by -100 when we want to
ignore
        # padding in the loss.
        if padding == "max_length" and data_args.ignore_pad_token_for_loss:
```

```
            labels["input_ids"] = [
                [(l if l != tokenizer.pad_token_id else -100) for l in label] for label in labels["input_ids"]
            ]

        model_inputs["labels"] = labels["input_ids"]
        return model_inputs

    # Validation preprocessing
    def preprocess_validation_function(examples):
        inputs, targets = preprocess_squad_batch(
            examples, question_column, answer_column)

        model_inputs = tokenizer(
            inputs,
            max_length=max_seq_length,
            padding=padding,
            truncation=True,
            return_overflowing_tokens=True,
            return_offsets_mapping=True,
        )
        # Tokenize targets with the `text_target` keyword argument
        labels = tokenizer(
            text_target=targets, max_length=max_answer_length, padding=padding, truncation=True)

        # If we are padding here, replace all tokenizer.pad_token_id in the labels by -100 when we want to
ignore
        # padding in the loss.
        if padding == "max_length" and data_args.ignore_pad_token_for_loss:
            labels["input_ids"] = [
                [(l if l != tokenizer.pad_token_id else -100) for l in label] for label in labels["input_ids"]
            ]

        # Since one example might give us several features if it has a long context, we need a map from a
feature to
        # its corresponding example. This key gives us just that.
        sample_mapping = model_inputs.pop("overflow_to_sample_mapping")

        # For evaluation, we will need to convert our predictions to substrings of the context, so we keep
the
        # corresponding example_id and we will store the offset mappings.
        model_inputs["example_id"] = []
        # Augment the overflowing tokens to the labels
        labels_out = []

        for i in range(len(model_inputs["input_ids"])):
            # One example can give several spans, this is the index of the example containing this span of
text.
            sample_index = sample_mapping[i]
            model_inputs["example_id"].append(examples["id"][sample_index])
```

```
            labels_out.append(labels["input_ids"][sample_index])

        model_inputs["labels"] = labels_out
        return model_inputs

    if training_args.do_train:
        if "train" not in raw_datasets:
            raise ValueError("--do_train requires a train dataset")
        train_dataset = raw_datasets["train"]
        if data_args.max_train_samples is not None:
            # We will select sample from whole data if agument is specified
            max_train_samples = min(
                len(train_dataset), data_args.max_train_samples)
            train_dataset = train_dataset.select(range(max_train_samples))
        # Create train feature from dataset
        with training_args.main_process_first(desc="train dataset map pre-processing"):
            train_dataset = train_dataset.map(
                preprocess_function,
                batched=True,
                num_proc=data_args.preprocessing_num_workers,
                remove_columns=column_names,
                load_from_cache_file=not data_args.overwrite_cache,
                desc="Running tokenizer on train dataset",
            )
        if data_args.max_train_samples is not None:
            # Number of samples might increase during Feature Creation, We select only specified max
samples
            max_train_samples = min(
                len(train_dataset), data_args.max_train_samples)
            train_dataset = train_dataset.select(range(max_train_samples))

    if training_args.do_eval:
        if "validation" not in raw_datasets:
            raise ValueError("--do_eval requires a validation dataset")
        eval_examples = raw_datasets["validation"]
        if data_args.max_eval_samples is not None:
            # We will select sample from whole data
            max_eval_samples = min(
                len(eval_examples), data_args.max_eval_samples)
            eval_examples = eval_examples.select(range(max_eval_samples))
        # Validation Feature Creation
        with training_args.main_process_first(desc="validation dataset map pre-processing"):
            eval_dataset = eval_examples.map(
                preprocess_validation_function,
                batched=True,
                num_proc=data_args.preprocessing_num_workers,
                remove_columns=column_names,
                load_from_cache_file=not data_args.overwrite_cache,
                desc="Running tokenizer on validation dataset",
```

```
        )
    if data_args.max_eval_samples is not None:
        # During Feature creation dataset samples might increase, we will select required samples again
        max_eval_samples = min(
            len(eval_dataset), data_args.max_eval_samples)
        eval_dataset = eval_dataset.select(range(max_eval_samples))

if training_args.do_predict:
    if "test" not in raw_datasets:
        raise ValueError("--do_predict requires a test dataset")
    predict_examples = raw_datasets["test"]
    if data_args.max_predict_samples is not None:
        # We will select sample from whole data
        predict_examples = predict_examples.select(
            range(data_args.max_predict_samples))
    # Predict Feature Creation
    with training_args.main_process_first(desc="prediction dataset map pre-processing"):
        predict_dataset = predict_examples.map(
            preprocess_validation_function,
            batched=True,
            num_proc=data_args.preprocessing_num_workers,
            remove_columns=column_names,
            load_from_cache_file=not data_args.overwrite_cache,
            desc="Running tokenizer on prediction dataset",
        )
    if data_args.max_predict_samples is not None:
        # During Feature creation dataset samples might increase, we will select required samples again
        max_predict_samples = min(
            len(predict_dataset), data_args.max_predict_samples)
        predict_dataset = predict_dataset.select(
            range(max_predict_samples))

# Data collator
label_pad_token_id = - \
    100 if data_args.ignore_pad_token_for_loss else tokenizer.pad_token_id
data_collator = DataCollatorForSeq2Seq(
    tokenizer,
    model=model,
    label_pad_token_id=label_pad_token_id,
    pad_to_multiple_of=8 if training_args.fp16 else None,
)

metric = evaluate.load(
    "squad_v2" if data_args.version_2_with_negative else "exact_match")

def compute_metrics(p: EvalPrediction):
    return metric.compute(predictions=p.predictions, references=p.label_ids)

# Post-processing:
```

```
def post_processing_function(
    examples: datasets.Dataset, features: datasets.Dataset, outputs: EvalLoopOutput, stage="eval"
):
    # Decode the predicted tokens.
    preds = outputs.predictions
    if isinstance(preds, tuple):
        preds = preds[0]
    decoded_preds = tokenizer.batch_decode(preds, skip_special_tokens=True)

    # Build a map example to its corresponding features.
    example_id_to_index = {k: i for i, k in enumerate(examples["id"])}
    feature_per_example = {
        example_id_to_index[feature["example_id"]]: i for i, feature in enumerate(features)}
    predictions = {}
    # Let's loop over all the examples!
    for example_index, example in enumerate(examples):
        # This is the index of the feature associated to the current example.
        feature_index = feature_per_example[example_index]
        predictions[example["id"]] = decoded_preds[feature_index]

    # Format the result to the format the metric expects.
    if data_args.version_2_with_negative:
        formatted_predictions = [
            {"id": k, "prediction_text": v, "no_answer_probability": 0.0} for k, v in predictions.items()
        ]
    else:
        formatted_predictions = [
            {"id": k, "prediction_text": v} for k, v in predictions.items()]

    references = [{"id": ex["id"], "answers": ex[answer_column]}
                  for ex in examples]
    return EvalPrediction(predictions=formatted_predictions, label_ids=references)

# Initialize our Trainer
 # Initialize our Trainer
optimizer = Adafactor(model.parameters(), scale_parameter=False,
                      relative_step=False, warmup_init=False, lr=training_args.learning_rate)
# lr_scheduler = AdafactorSchedule(optimizer)
# training_args.optim_args={'scale_parameter':False, 'relative_step':False, 'warmup_init': False}
trainer = QuestionAnsweringSeq2SeqTrainer(
    model=model,
    args=training_args,
    # optimizers=(optimizer,lr_scheduler),
    train_dataset=train_dataset if training_args.do_train else None,
    eval_dataset=eval_dataset if training_args.do_eval else None,
    eval_examples=eval_examples if training_args.do_eval else None,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics if training_args.predict_with_generate else None,
```

```
        post_process_function=post_processing_function,
    )
    trainer.optimizer = optimizer
    # Training
    if training_args.do_train:
        checkpoint = None
        if training_args.resume_from_checkpoint is not None:
            checkpoint = training_args.resume_from_checkpoint
        elif last_checkpoint is not None:
            checkpoint = last_checkpoint
        train_result = trainer.train(resume_from_checkpoint=checkpoint)
        trainer.save_model()  # Saves the tokenizer too for easy upload

        metrics = train_result.metrics
        max_train_samples = (
            data_args.max_train_samples if data_args.max_train_samples is not None else len(
                train_dataset)
        )
        metrics["train_samples"] = min(max_train_samples, len(train_dataset))

        trainer.log_metrics("train", metrics)
        trainer.save_metrics("train", metrics)
        trainer.save_state()

    # Evaluation
    results = {}
    max_length = (
        training_args.generation_max_length
        if training_args.generation_max_length is not None
        else data_args.val_max_answer_length
    )
    num_beams = data_args.num_beams if data_args.num_beams is not None else
training_args.generation_num_beams
    if training_args.do_eval:
        logger.info("*** Evaluate ***")
        metrics = trainer.evaluate(
            max_length=max_length, num_beams=num_beams, metric_key_prefix="eval")

        max_eval_samples = data_args.max_eval_samples if data_args.max_eval_samples is not None
else len(
            eval_dataset)
        metrics["eval_samples"] = min(max_eval_samples, len(eval_dataset))

        trainer.log_metrics("eval", metrics)
        trainer.save_metrics("eval", metrics)

    # Prediction
    if training_args.do_predict:
        logger.info("*** Predict ***")
```

```
        results = trainer.predict(predict_dataset, predict_examples)
        metrics = results.metrics

        max_predict_samples = (
            data_args.max_predict_samples if data_args.max_predict_samples is not None else len(
                predict_dataset)
        )
        metrics["predict_samples"] = min(
            max_predict_samples, len(predict_dataset))

        trainer.log_metrics("predict", metrics)
        trainer.save_metrics("predict", metrics)

    if training_args.push_to_hub:
        kwargs = {"finetuned_from": model_args.model_name_or_path,
                  "tasks": "question-answering"}
        if data_args.dataset_name is not None:
            kwargs["dataset_tags"] = data_args.dataset_name
            if data_args.dataset_config_name is not None:
                kwargs["dataset_args"] = data_args.dataset_config_name
                kwargs["dataset"] = f"{data_args.dataset_name} {data_args.dataset_config_name}"
            else:
                kwargs["dataset"] = data_args.dataset_name

        trainer.push_to_hub(**kwargs)


def _mp_fn(index):
    # For xla_spawn (TPUs)
    main()


if __name__ == "__main__":
    main()


#!/usr/bin/env python
# 0oding=utf-8
# Copyright 2021 The HuggingFace Team All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

```
# See the License for the specific language governing permissions and
# limitations under the License.
"""
Fine-tuning the library's seq2seq models for question answering using the  Seq2SeqTrainer.
"""
# You can also adapt this script on your own question answering task. Pointers for this are left as
comments.

import logging
import os
import sys
from dataclasses import dataclass, field
from typing import List, Optional, Tuple

import datasets
import evaluate
from datasets import load_dataset
from trainer_seq2seq_qa import QuestionAnsweringSeq2SeqTrainer

import transformers
from transformers import (
    AutoConfig,
    AutoModelForSeq2SeqLM,
    AutoTokenizer,
    DataCollatorForSeq2Seq,
    HfArgumentParser,
    Seq2SeqTrainingArguments,
    set_seed,
)
from transformers.trainer_utils import EvalLoopOutput, EvalPrediction, get_last_checkpoint
from transformers.utils import check_min_version  # , send_example_telemetry
from transformers.utils.versions import require_version
from transformers.optimization import Adafactor, AdafactorSchedule

# Will error if the minimal version of Transformers is not installed. Remove at your own risks.
check_min_version("4.28.0.dev0")

require_version("datasets>=1.8.0",
                "To fix: pip install -r examples/pytorch/question-answering/requirements.txt")

logger = logging.getLogger(__name__)


@dataclass
class ModelArguments:
    """
    Arguments pertaining to which model/config/tokenizer we are going to fine-tune from.
    """
```

```
    model_name_or_path: str = field(
        metadata={
            "help": "Path to pretrained model or model identifier from huggingface.co/models"}
    )
    config_name: Optional[str] = field(
        default=None, metadata={"help": "Pretrained config name or path if not the same as
model_name"}
    )
    tokenizer_name: Optional[str] = field(
        default=None, metadata={"help": "Pretrained tokenizer name or path if not the same as
model_name"}
    )
    cache_dir: Optional[str] = field(
        default=None,
        metadata={
            "help": "Path to directory to store the pretrained models downloaded from huggingface.co"},
    )
    use_fast_tokenizer: bool = field(
        default=True,
        metadata={
            "help": "Whether to use one of the fast tokenizer (backed by the tokenizers library) or not."},
    )
    model_revision: str = field(
        default="main",
        metadata={
            "help": "The specific model version to use (can be a branch name, tag name or commit id)."},
    )
    use_auth_token: bool = field(
        default=False,
        metadata={
            "help": (
                "Will use the token generated when running `huggingface-cli login` (necessary to use this
script "
                "with private models)."
            )
        },
    )


@dataclass
class DataTrainingArguments:
    """
    Arguments pertaining to what data we are going to input our model for training and eval.
    """

    dataset_name: Optional[str] = field(
        default=None, metadata={"help": "The name of the dataset to use (via the datasets library)."}
    )
    dataset_config_name: Optional[str] = field(
```

```
        default=None, metadata={"help": "The configuration name of the dataset to use (via the datasets library)."}
    )
    context_column: Optional[str] = field(
        default="context",
        metadata={
            "help": "The name of the column in the datasets containing the contexts (for question answering)."},
    )
    question_column: Optional[str] = field(
        default="question",
        metadata={
            "help": "The name of the column in the datasets containing the questions (for question answering)."},
    )
    answer_column: Optional[str] = field(
        default="answers",
        metadata={
            "help": "The name of the column in the datasets containing the answers (for question answering)."},
    )
    train_file: Optional[str] = field(
        default=None, metadata={"help": "The input training data file (a text file)."})
    validation_file: Optional[str] = field(
        default=None,
        metadata={
            "help": "An optional input evaluation data file to evaluate the perplexity on (a text file)."},
    )
    test_file: Optional[str] = field(
        default=None,
        metadata={
            "help": "An optional input test data file to evaluate the perplexity on (a text file)."},
    )
    overwrite_cache: bool = field(
        default=False, metadata={"help": "Overwrite the cached training and evaluation sets"}
    )
    preprocessing_num_workers: Optional[int] = field(
        default=None,
        metadata={"help": "The number of processes to use for the preprocessing."},
    )
    max_seq_length: int = field(
        default=384,
        metadata={
            "help": (
                "The maximum total input sequence length after tokenization. Sequences longer "
                "than this will be truncated, sequences shorter will be padded."
            )
        },
    )
```

```
    max_answer_length: int = field(
        default=30,
        metadata={
            "help": (
                "The maximum length of an answer that can be generated. This is needed because the start "
                "and end predictions are not conditioned on one another."
            )
        },
    )
    val_max_answer_length: Optional[int] = field(
        default=None,
        metadata={
            "help": (
                "The maximum total sequence length for validation target text after tokenization. Sequences
longer "
                "than this will be truncated, sequences shorter will be padded. Will default to
`max_answer_length`."
                "This argument is also used to override the ``max_length`` param of ``model.generate``,
which is used "
                "during ``evaluate`` and ``predict``."
            )
        },
    )
    pad_to_max_length: bool = field(
        default=True,
        metadata={
            "help": (
                "Whether to pad all samples to `max_seq_length`. If False, will pad the samples dynamically
when"
                " batching to the maximum length in the batch (which can be faster on GPU but will be
slower on TPU)."
            )
        },
    )
    max_train_samples: Optional[int] = field(
        default=None,
        metadata={
            "help": (
                "For debugging purposes or quicker training, truncate the number of training examples to this
"
                "value if set."
            )
        },
    )
    max_eval_samples: Optional[int] = field(
        default=None,
        metadata={
            "help": (
```

```
            "For debugging purposes or quicker training, truncate the number of evaluation examples to
this "
            "value if set."
        )
    },
  )
  max_predict_samples: Optional[int] = field(
    default=None,
    metadata={
      "help": (
        "For debugging purposes or quicker training, truncate the number of prediction examples to
this "
        "value if set."
      )
    },
  )
  version_2_with_negative: bool = field(
    default=False, metadata={"help": "If true, some of the examples do not have an answer."}
  )
  null_score_diff_threshold: float = field(
    default=0.0,
    metadata={
      "help": (
        "The threshold used to select the null answer: if the best answer has a score that is less than "
        "the score of the null answer minus this threshold, the null answer is selected for this
example. "
        "Only useful when `version_2_with_negative=True`."
      )
    },
  )
  doc_stride: int = field(
    default=128,
    metadata={
      "help": "When splitting up a long document into chunks, how much stride to take between
chunks."},
  )
  n_best_size: int = field(
    default=20,
    metadata={
      "help": "The total number of n-best predictions to generate when looking for an answer."},
  )
  num_beams: Optional[int] = field(
    default=None,
    metadata={
      "help": (
        "Number of beams to use for evaluation. This argument will be passed to ``model.generate``,
"
        "which is used during ``evaluate`` and ``predict``."
      )
```

```
        },
    )
    ignore_pad_token_for_loss: bool = field(
        default=True,
        metadata={
            "help": "Whether to ignore the tokens corresponding to padded labels in the loss computation or
not."
        },
    )

    def __post_init__(self):
        if (
            self.dataset_name is None
            and self.train_file is None
            and self.validation_file is None
            and self.test_file is None
        ):
            raise ValueError(
                "Need either a dataset name or a training/validation file/test_file.")
        else:
            if self.train_file is not None:
                extension = self.train_file.split(".")[-1]
                assert extension in [
                    "csv", "json"], "`train_file` should be a csv or a json file."
            if self.validation_file is not None:
                extension = self.validation_file.split(".")[-1]
                assert extension in [
                    "csv", "json"], "`validation_file` should be a csv or a json file."
            if self.test_file is not None:
                extension = self.test_file.split(".")[-1]
                assert extension in [
                    "csv", "json"], "`test_file` should be a csv or a json file."
        if self.val_max_answer_length is None:
            self.val_max_answer_length = self.max_answer_length


question_answering_column_name_mapping = {
    "squad_v2": ("question", "context", "answer"),
}


def main():
    # See all possible arguments in src/transformers/training_args.py
    # or by passing the --help flag to this script.
    # We now keep distinct sets of args, for a cleaner separation of concerns.

    parser = HfArgumentParser(
        (ModelArguments, DataTrainingArguments, Seq2SeqTrainingArguments))
    if len(sys.argv) == 2 and sys.argv[1].endswith(".json"):
```

```
        # If we pass only one argument to the script and it's the path to a json file,
        # let's parse it to get our arguments.
        model_args, data_args, training_args = parser.parse_json_file(
            json_file=os.path.abspath(sys.argv[1]))
    else:
        model_args, data_args, training_args = parser.parse_args_into_dataclasses()

    # Sending telemetry. Tracking the example usage helps us better allocate resources to maintain them.
The
    # information sent is the one passed as arguments along with your Python/PyTorch versions.
    # send_example_telemetry("run_seq2seq_qa", model_args, data_args)

    # Setup logging
    logging.basicConfig(
        format="%(asctime)s - %(levelname)s - %(name)s - %(message)s",
        datefmt="%m/%d/%Y %H:%M:%S",
        handlers=[logging.StreamHandler(sys.stdout)],
    )

    if training_args.should_log:
        # The default of training_args.log_level is passive, so we set log level at info here to have that
default.
        transformers.utils.logging.set_verbosity_info()

    log_level = training_args.get_process_log_level()
    logger.setLevel(log_level)
    datasets.utils.logging.set_verbosity(log_level)
    transformers.utils.logging.set_verbosity(log_level)
    transformers.utils.logging.enable_default_handler()
    transformers.utils.logging.enable_explicit_format()

    # Log on each process the small summary:
    logger.warning(
        f"Process rank: {training_args.local_rank}, device: {training_args.device}, n_gpu:
{training_args.n_gpu}"
        + f"distributed training: {bool(training_args.local_rank != -1)}, 16-bits training:
{training_args.fp16}"
    )
    logger.info(f"Training/evaluation parameters {training_args}")

    # Detecting last checkpoint.
    last_checkpoint = None
    if os.path.isdir(training_args.output_dir) and training_args.do_train and not
training_args.overwrite_output_dir:
        last_checkpoint = get_last_checkpoint(training_args.output_dir)
        if last_checkpoint is None and len(os.listdir(training_args.output_dir)) > 0:
            raise ValueError(
                f"Output directory ({training_args.output_dir}) already exists and is not empty. "
                "Use --overwrite_output_dir to overcome."
```

```
            )
        elif last_checkpoint is not None and training_args.resume_from_checkpoint is None:
            logger.info(
                f"Checkpoint detected, resuming training at {last_checkpoint}. To avoid this behavior,
change "
                "the `--output_dir` or add `--overwrite_output_dir` to train from scratch."
            )

    # Set seed before initializing model.
    set_seed(training_args.seed)

    # Get the datasets: you can either provide your own CSV/JSON/TXT training and evaluation files
(see below)
    # or just provide the name of one of the public datasets available on the hub at
https://huggingface.co/datasets/
    # (the dataset will be downloaded automatically from the datasets Hub).
    #
    # For CSV/JSON files, this script will use the column called 'text' or the first column if no column
called
    # 'text' is found. You can easily tweak this behavior (see below).
    #
    # In distributed training, the load_dataset function guarantee that only one local process can
concurrently
    # download the dataset.
    if data_args.dataset_name is not None:
        # Downloading and loading a dataset from the hub.
        raw_datasets = load_dataset(
            data_args.dataset_name,
            data_args.dataset_config_name,
            cache_dir=model_args.cache_dir,
            use_auth_token=True if model_args.use_auth_token else None,
        )
    else:
        data_files = {}
        if data_args.train_file is not None:
            data_files["train"] = data_args.train_file
            extension = data_args.train_file.split(".")[-1]
        if data_args.validation_file is not None:
            data_files["validation"] = data_args.validation_file
            extension = data_args.validation_file.split(".")[-1]
        if data_args.test_file is not None:
            data_files["test"] = data_args.test_file
            extension = data_args.test_file.split(".")[-1]
        raw_datasets = load_dataset(
            extension,
            data_files=data_files,
            cache_dir=model_args.cache_dir,
            use_auth_token=True if model_args.use_auth_token else None,
        )
```

```
    # See more about loading any type of standard or custom dataset (from files, python dict, pandas DataFrame, etc) at
    # https://huggingface.co/docs/datasets/loading_datasets.html.

    # Load pretrained model and tokenizer
    #
    # Distributed training:
    # The .from_pretrained methods guarantee that only one local process can concurrently
    # download model & vocab.
    config = AutoConfig.from_pretrained(
        model_args.config_name if model_args.config_name else model_args.model_name_or_path,
        cache_dir=model_args.cache_dir,
        revision=model_args.model_revision,
        use_auth_token=True if model_args.use_auth_token else None,
    )
    tokenizer = AutoTokenizer.from_pretrained(
        model_args.tokenizer_name if model_args.tokenizer_name else model_args.model_name_or_path,
        cache_dir=model_args.cache_dir,
        use_fast=model_args.use_fast_tokenizer,
        revision=model_args.model_revision,
        use_auth_token=True if model_args.use_auth_token else None,
    )
    model = AutoModelForSeq2SeqLM.from_pretrained(
        model_args.model_name_or_path,
        from_tf=bool(".ckpt" in model_args.model_name_or_path),
        config=config,
        cache_dir=model_args.cache_dir,
        revision=model_args.model_revision,
        use_auth_token=True if model_args.use_auth_token else None,
    )

    # We resize the embeddings only when necessary to avoid index errors. If you are creating a model from scratch
    # on a small vocab and want a smaller embedding size, remove this test.
    embedding_size = model.get_input_embeddings().weight.shape[0]
    if len(tokenizer) > embedding_size:
        model.resize_token_embeddings(len(tokenizer))

    if model.config.decoder_start_token_id is None:
        raise ValueError(
            "Make sure that `config.decoder_start_token_id` is correctly defined")

    # Preprocessing the datasets.
    # We need to generate and tokenize inputs and targets.
    if training_args.do_train:
        column_names = raw_datasets["train"].column_names
    elif training_args.do_eval:
        column_names = raw_datasets["validation"].column_names
```

```
    elif training_args.do_predict:
        column_names = raw_datasets["test"].column_names
    else:
        logger.info(
            "There is nothing to do. Please pass `do_train`, `do_eval` and/or `do_predict`.")
        return

    # Get the column names for input/target.
    dataset_columns = question_answering_column_name_mapping.get(
        'squad_v2', None)
    if data_args.question_column is None:
        question_column = dataset_columns[0] if dataset_columns is not None else column_names[0]
    else:
        question_column = data_args.question_column
        if question_column not in column_names:
            raise ValueError(
                f"--question_column' value '{data_args.question_column}' needs to be one of: {',
'.join(column_names)}"
            )
    if data_args.context_column is None:
        context_column = dataset_columns[1] if dataset_columns is not None else column_names[1]
    else:
        context_column = data_args.context_column
        if context_column not in column_names:
            raise ValueError(
                f"--context_column' value '{data_args.context_column}' needs to be one of: {',
'.join(column_names)}"
            )
    if data_args.answer_column is None:
        answer_column = dataset_columns[2] if dataset_columns is not None else column_names[2]
    else:
        answer_column = data_args.answer_column
        if answer_column not in column_names:
            raise ValueError(
                f"--answer_column' value '{data_args.answer_column}' needs to be one of: {',
'.join(column_names)}"
            )

    # Temporarily set max_answer_length for training.
    max_answer_length = data_args.max_answer_length
    padding = "max_length" if data_args.pad_to_max_length else False

    if training_args.label_smoothing_factor > 0 and not hasattr(model,
"prepare_decoder_input_ids_from_labels"):
        logger.warning(
            "label_smoothing is enabled but the `prepare_decoder_input_ids_from_labels` method is not
defined for"
            f"`{model.__class__.__name__}`. This will lead to loss being calculated twice and will take up
more memory"
```

```
        )

    if data_args.max_seq_length > tokenizer.model_max_length:
        logger.warning(
            f"The max_seq_length passed ({data_args.max_seq_length}) is larger than the maximum length
for the"
            f"model ({tokenizer.model_max_length}). Using
max_seq_length={tokenizer.model_max_length}."
        )
    max_seq_length = min(data_args.max_seq_length, tokenizer.model_max_length)

    def preprocess_squad_batch(
        examples,
        question_column: str,
        context_column: str,
        answer_column: str,
    ) -> Tuple[List[str], List[str]]:
        questions = examples[question_column]
        contexts = examples[context_column]
        answers = examples[answer_column]

        def generate_input(_question, _context):
            return " ".join(["question:", _question.lstrip() if _question != None else "", "context:",
_context.lstrip() if _context != None else ""])

        inputs = [generate_input(question, context)
                  for question, context in zip(questions, contexts)]
        targets = [answer if answer != None else "None" for answer in answers]
        return inputs, targets

    def preprocess_function(examples):
        inputs, targets = preprocess_squad_batch(
            examples, question_column, context_column, answer_column)

        model_inputs = tokenizer(
            inputs, max_length=max_seq_length, padding=padding, truncation=True)
        # Tokenize targets with text_target=...
        labels = tokenizer(
            text_target=targets, max_length=max_answer_length, padding=padding, truncation=True)

        # If we are padding here, replace all tokenizer.pad_token_id in the labels by -100 when we want to
ignore
        # padding in the loss.
        if padding == "max_length" and data_args.ignore_pad_token_for_loss:
            labels["input_ids"] = [
                [(l if l != tokenizer.pad_token_id else -100) for l in label] for label in labels["input_ids"]
            ]

        model_inputs["labels"] = labels["input_ids"]
```

```
        return model_inputs

    # Validation preprocessing
    def preprocess_validation_function(examples):
        inputs, targets = preprocess_squad_batch(
            examples, question_column, context_column, answer_column)

        model_inputs = tokenizer(
            inputs,
            max_length=max_seq_length,
            padding=padding,
            truncation=True,
            return_overflowing_tokens=True,
            return_offsets_mapping=True,
        )
        # Tokenize targets with the `text_target` keyword argument
        labels = tokenizer(
            text_target=targets, max_length=max_answer_length, padding=padding, truncation=True)

        # If we are padding here, replace all tokenizer.pad_token_id in the labels by -100 when we want to
ignore
        # padding in the loss.
        if padding == "max_length" and data_args.ignore_pad_token_for_loss:
            labels["input_ids"] = [
                [(l if l != tokenizer.pad_token_id else -100) for l in label] for label in labels["input_ids"]
            ]

        # Since one example might give us several features if it has a long context, we need a map from a
feature to
        # its corresponding example. This key gives us just that.
        sample_mapping = model_inputs.pop("overflow_to_sample_mapping")

        # For evaluation, we will need to convert our predictions to substrings of the context, so we keep
the
        # corresponding example_id and we will store the offset mappings.
        model_inputs["example_id"] = []
        # Augment the overflowing tokens to the labels
        labels_out = []

        for i in range(len(model_inputs["input_ids"])):
            # One example can give several spans, this is the index of the example containing this span of
text.
            sample_index = sample_mapping[i]
            model_inputs["example_id"].append(examples["id"][sample_index])
            labels_out.append(labels["input_ids"][sample_index])

        model_inputs["labels"] = labels_out
        return model_inputs
```

```
    if training_args.do_train:
        if "train" not in raw_datasets:
            raise ValueError("--do_train requires a train dataset")
        train_dataset = raw_datasets["train"]
        if data_args.max_train_samples is not None:
            # We will select sample from whole data if agument is specified
            max_train_samples = min(
                len(train_dataset), data_args.max_train_samples)
            train_dataset = train_dataset.select(range(max_train_samples))
        # Create train feature from dataset
        with training_args.main_process_first(desc="train dataset map pre-processing"):
            train_dataset = train_dataset.map(
                preprocess_function,
                batched=True,
                num_proc=data_args.preprocessing_num_workers,
                remove_columns=column_names,
                load_from_cache_file=not data_args.overwrite_cache,
                desc="Running tokenizer on train dataset",
            )
        if data_args.max_train_samples is not None:
            # Number of samples might increase during Feature Creation, We select only specified max
samples
            max_train_samples = min(
                len(train_dataset), data_args.max_train_samples)
            train_dataset = train_dataset.select(range(max_train_samples))

    if training_args.do_eval:
        if "validation" not in raw_datasets:
            raise ValueError("--do_eval requires a validation dataset")
        eval_examples = raw_datasets["validation"]
        if data_args.max_eval_samples is not None:
            # We will select sample from whole data
            max_eval_samples = min(
                len(eval_examples), data_args.max_eval_samples)
            eval_examples = eval_examples.select(range(max_eval_samples))
        # Validation Feature Creation
        with training_args.main_process_first(desc="validation dataset map pre-processing"):
            eval_dataset = eval_examples.map(
                preprocess_validation_function,
                batched=True,
                num_proc=data_args.preprocessing_num_workers,
                remove_columns=column_names,
                load_from_cache_file=not data_args.overwrite_cache,
                desc="Running tokenizer on validation dataset",
            )
        if data_args.max_eval_samples is not None:
            # During Feature creation dataset samples might increase, we will select required samples again
            max_eval_samples = min(
                len(eval_dataset), data_args.max_eval_samples)
```

```
        eval_dataset = eval_dataset.select(range(max_eval_samples))

if training_args.do_predict:
    if "test" not in raw_datasets:
        raise ValueError("--do_predict requires a test dataset")
    predict_examples = raw_datasets["test"]
    if data_args.max_predict_samples is not None:
        # We will select sample from whole data
        predict_examples = predict_examples.select(
            range(data_args.max_predict_samples))
    # Predict Feature Creation
    with training_args.main_process_first(desc="prediction dataset map pre-processing"):
        predict_dataset = predict_examples.map(
            preprocess_validation_function,
            batched=True,
            num_proc=data_args.preprocessing_num_workers,
            remove_columns=column_names,
            load_from_cache_file=not data_args.overwrite_cache,
            desc="Running tokenizer on prediction dataset",
        )
    if data_args.max_predict_samples is not None:
        # During Feature creation dataset samples might increase, we will select required samples again
        max_predict_samples = min(
            len(predict_dataset), data_args.max_predict_samples)
        predict_dataset = predict_dataset.select(
            range(max_predict_samples))

# Data collator
label_pad_token_id = - \
    100 if data_args.ignore_pad_token_for_loss else tokenizer.pad_token_id
data_collator = DataCollatorForSeq2Seq(
    tokenizer,
    model=model,
    label_pad_token_id=label_pad_token_id,
    pad_to_multiple_of=8 if training_args.fp16 else None,
)

metric = evaluate.load(
    "squad_v2" if data_args.version_2_with_negative else "squad")

def compute_metrics(p: EvalPrediction):
    return metric.compute(predictions=p.predictions, references=p.label_ids)

# Post-processing:
def post_processing_function(
    examples: datasets.Dataset, features: datasets.Dataset, outputs: EvalLoopOutput, stage="eval"
):
    # Decode the predicted tokens.
    preds = outputs.predictions
```

```
    if isinstance(preds, tuple):
        preds = preds[0]
    decoded_preds = tokenizer.batch_decode(preds, skip_special_tokens=True)

    # Build a map example to its corresponding features.
    example_id_to_index = {k: i for i, k in enumerate(examples["id"])}
    feature_per_example = {
        example_id_to_index[feature["example_id"]]: i for i, feature in enumerate(features)}
    predictions = {}
    # Let's loop over all the examples!
    for example_index, example in enumerate(examples):
        # This is the index of the feature associated to the current example.
        feature_index = feature_per_example[example_index]
        predictions[example["id"]] = decoded_preds[feature_index]

    # Format the result to the format the metric expects.
    if data_args.version_2_with_negative:
        formatted_predictions = [
            {"id": k, "prediction_text": v, "no_answer_probability": 0.0} for k, v in predictions.items()
        ]
    else:
        formatted_predictions = [
            {"id": k, "prediction_text": v} for k, v in predictions.items()]

    references = [{"id": ex["id"], "answers": ex[answer_column]}
                  for ex in examples]
    return EvalPrediction(predictions=formatted_predictions, label_ids=references)

# Initialize our Trainer
optimizer = Adafactor(model.parameters(), scale_parameter=False,
                      relative_step=False, warmup_init=False, lr=training_args.learning_rate)
# lr_scheduler = AdafactorSchedule(optimizer)
# training_args.optim_args={'scale_parameter':False, 'relative_step':False, 'warmup_init': False}
trainer = QuestionAnsweringSeq2SeqTrainer(
    model=model,
    args=training_args,
    # optimizers=(optimizer,lr_scheduler),
    train_dataset=train_dataset if training_args.do_train else None,
    eval_dataset=eval_dataset if training_args.do_eval else None,
    eval_examples=eval_examples if training_args.do_eval else None,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics if training_args.predict_with_generate else None,
    post_process_function=post_processing_function,
)
trainer.optimizer = optimizer
# Training
if training_args.do_train:
    checkpoint = None
```

```
        if training_args.resume_from_checkpoint is not None:
            checkpoint = training_args.resume_from_checkpoint
        elif last_checkpoint is not None:
            checkpoint = last_checkpoint
        train_result = trainer.train(resume_from_checkpoint=checkpoint)
        trainer.save_model()  # Saves the tokenizer too for easy upload

        metrics = train_result.metrics
        max_train_samples = (
            data_args.max_train_samples if data_args.max_train_samples is not None else len(
                train_dataset)
        )
        metrics["train_samples"] = min(max_train_samples, len(train_dataset))

        trainer.log_metrics("train", metrics)
        trainer.save_metrics("train", metrics)
        trainer.save_state()

    # Evaluation
    results = {}
    max_length = (
        training_args.generation_max_length
        if training_args.generation_max_length is not None
        else data_args.val_max_answer_length
    )
    num_beams = data_args.num_beams if data_args.num_beams is not None else
training_args.generation_num_beams
    if training_args.do_eval:
        logger.info("*** Evaluate ***")
        metrics = trainer.evaluate(
            max_length=max_length, num_beams=num_beams, metric_key_prefix="eval")

        max_eval_samples = data_args.max_eval_samples if data_args.max_eval_samples is not None
else len(
            eval_dataset)
        metrics["eval_samples"] = min(max_eval_samples, len(eval_dataset))

        trainer.log_metrics("eval", metrics)
        trainer.save_metrics("eval", metrics)

    # Prediction
    if training_args.do_predict:
        logger.info("*** Predict ***")
        results = trainer.predict(predict_dataset, predict_examples)
        metrics = results.metrics

        max_predict_samples = (
            data_args.max_predict_samples if data_args.max_predict_samples is not None else len(
                predict_dataset)
```

```
        )
        metrics["predict_samples"] = min(
            max_predict_samples, len(predict_dataset))

        trainer.log_metrics("predict", metrics)
        trainer.save_metrics("predict", metrics)

    if training_args.push_to_hub:
        kwargs = {"finetuned_from": model_args.model_name_or_path,
                  "tasks": "question-answering"}
        if data_args.dataset_name is not None:
            kwargs["dataset_tags"] = data_args.dataset_name
            if data_args.dataset_config_name is not None:
                kwargs["dataset_args"] = data_args.dataset_config_name
                kwargs["dataset"] = f"{data_args.dataset_name} {data_args.dataset_config_name}"
            else:
                kwargs["dataset"] = data_args.dataset_name

        trainer.push_to_hub(**kwargs)


def _mp_fn(index):
    # For xla_spawn (TPUs)
    main()


if __name__ == "__main__":
    main()


# coding=utf-8
# Copyright 2021 The HuggingFace Team All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
"""
A subclass of `Trainer` specific to Question-Answering tasks
"""
import math
import time
```

```
from typing import Dict, List, Optional

from torch.utils.data import Dataset

from transformers import Seq2SeqTrainer, is_torch_tpu_available
from transformers.trainer_utils import PredictionOutput, speed_metrics


if is_torch_tpu_available(check_device=False):
    import torch_xla.core.xla_model as xm
    import torch_xla.debug.metrics as met


class QuestionAnsweringSeq2SeqTrainer(Seq2SeqTrainer):
    def __init__(self, *args, eval_examples=None, post_process_function=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.eval_examples = eval_examples
        self.post_process_function = post_process_function

    # def evaluate(self, eval_dataset=None, eval_examples=None, ignore_keys=None,
metric_key_prefix: str = "eval"):
    def evaluate(
        self,
        eval_dataset: Optional[Dataset] = None,
        eval_examples=None,
        ignore_keys: Optional[List[str]] = None,
        metric_key_prefix: str = "eval",
        **gen_kwargs,
    ) -> Dict[str, float]:
        gen_kwargs = gen_kwargs.copy()
        gen_kwargs["max_length"] = (
            gen_kwargs["max_length"] if gen_kwargs.get("max_length") is not None else
self.args.generation_max_length
        )
        gen_kwargs["num_beams"] = (
            gen_kwargs["num_beams"] if gen_kwargs.get("num_beams") is not None else
self.args.generation_num_beams
        )
        self._gen_kwargs = gen_kwargs

        eval_dataset = self.eval_dataset if eval_dataset is None else eval_dataset
        eval_dataloader = self.get_eval_dataloader(eval_dataset)
        eval_examples = self.eval_examples if eval_examples is None else eval_examples

        # Temporarily disable metric computation, we will do it in the loop here.
        compute_metrics = self.compute_metrics
        self.compute_metrics = None
        start_time = time.time()
        eval_loop = self.prediction_loop if self.args.use_legacy_prediction_loop else self.evaluation_loop
```

```
        try:
            output = eval_loop(
                eval_dataloader,
                description="Evaluation",
                # No point gathering the predictions if there are no metrics, otherwise we defer to
                # self.args.prediction_loss_only
                prediction_loss_only=True if compute_metrics is None else None,
                ignore_keys=ignore_keys,
                metric_key_prefix=metric_key_prefix,
            )
        finally:
            self.compute_metrics = compute_metrics
        total_batch_size = self.args.eval_batch_size * self.args.world_size
        if f"{metric_key_prefix}_jit_compilation_time" in output.metrics:
            start_time += output.metrics[f"{metric_key_prefix}_jit_compilation_time"]
        output.metrics.update(
            speed_metrics(
                metric_key_prefix,
                start_time,
                num_samples=output.num_samples,
                num_steps=math.ceil(output.num_samples / total_batch_size),
            )
        )

        if self.post_process_function is not None and self.compute_metrics is not None and
    self.args.should_save:
            # Only the main node write the results by default
            eval_preds = self.post_process_function(eval_examples, eval_dataset, output)
            metrics = self.compute_metrics(eval_preds)

            # Prefix all keys with metric_key_prefix + '_'
            for key in list(metrics.keys()):
                if not key.startswith(f"{metric_key_prefix}_"):
                    metrics[f"{metric_key_prefix}_{key}"] = metrics.pop(key)

            metrics.update(output.metrics)
        else:
            metrics = output.metrics

        if self.args.should_log:
            # Only the main node log the results by default
            self.log(metrics)

        if self.args.tpu_metrics_debug or self.args.debug:
            # tpu-comment: Logging debug metrics for PyTorch/XLA (compile, execute times, ops, etc.)
            xm.master_print(met.metrics_report())

        self.control = self.callback_handler.on_evaluate(self.args, self.state, self.control, metrics)
        return metrics
```

```
    def predict(
        self, predict_dataset, predict_examples, ignore_keys=None, metric_key_prefix: str = "test",
**gen_kwargs
    ):
        self._gen_kwargs = gen_kwargs.copy()

        predict_dataloader = self.get_test_dataloader(predict_dataset)

        # Temporarily disable metric computation, we will do it in the loop here.
        compute_metrics = self.compute_metrics
        self.compute_metrics = None
        start_time = time.time()
        eval_loop = self.prediction_loop if self.args.use_legacy_prediction_loop else self.evaluation_loop
        try:
            output = eval_loop(
                predict_dataloader,
                description="Prediction",
                # No point gathering the predictions if there are no metrics, otherwise we defer to
                # self.args.prediction_loss_only
                prediction_loss_only=True if compute_metrics is None else None,
                ignore_keys=ignore_keys,
                metric_key_prefix=metric_key_prefix,
            )
        finally:
            self.compute_metrics = compute_metrics

        total_batch_size = self.args.eval_batch_size * self.args.world_size
        if f"{metric_key_prefix}_jit_compilation_time" in output.metrics:
            start_time += output.metrics[f"{metric_key_prefix}_jit_compilation_time"]
        output.metrics.update(
            speed_metrics(
                metric_key_prefix,
                start_time,
                num_samples=output.num_samples,
                num_steps=math.ceil(output.num_samples / total_batch_size),
            )
        )
        if self.post_process_function is None or self.compute_metrics is None:
            return output

        predictions = self.post_process_function(predict_examples, predict_dataset, output, "predict")
        metrics = self.compute_metrics(predictions)

        # Prefix all keys with metric_key_prefix + '_'
        for key in list(metrics.keys()):
            if not key.startswith(f"{metric_key_prefix}_"):
                metrics[f"{metric_key_prefix}_{key}"] = metrics.pop(key)
        metrics.update(output.metrics)
```

```
        return PredictionOutput(predictions=predictions.predictions, label_ids=predictions.label_ids,
metrics=metrics)


# coding=utf-8
# Copyright 2021 The HuggingFace Team All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
"""
A subclass of `Trainer` specific to Question-Answering tasks
"""
import math
import time
from typing import Dict, List, Optional

from torch.utils.data import Dataset

from transformers import Seq2SeqTrainer, is_torch_tpu_available
from transformers.trainer_utils import PredictionOutput, speed_metrics


if is_torch_tpu_available():
    import torch_xla.core.xla_model as xm
    import torch_xla.debug.metrics as met


class QuestionAnsweringSeq2SeqTrainer(Seq2SeqTrainer):
    def __init__(self, *args, eval_examples=None, post_process_function=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.eval_examples = eval_examples
        self.post_process_function = post_process_function

    # def evaluate(self, eval_dataset=None, eval_examples=None, ignore_keys=None,
metric_key_prefix: str = "eval"):
    def evaluate(
        self,
        eval_dataset: Optional[Dataset] = None,
        eval_examples=None,
        ignore_keys: Optional[List[str]] = None,
```

```
        metric_key_prefix: str = "eval",
        **gen_kwargs,
    ) -> Dict[str, float]:
        gen_kwargs = gen_kwargs.copy()
        gen_kwargs["max_length"] = (
            gen_kwargs["max_length"] if gen_kwargs.get("max_length") is not None else
self.args.generation_max_length
        )
        gen_kwargs["num_beams"] = (
            gen_kwargs["num_beams"] if gen_kwargs.get("num_beams") is not None else
self.args.generation_num_beams
        )
        self._gen_kwargs = gen_kwargs

        eval_dataset = self.eval_dataset if eval_dataset is None else eval_dataset
        eval_dataloader = self.get_eval_dataloader(eval_dataset)
        eval_examples = self.eval_examples if eval_examples is None else eval_examples

        # Temporarily disable metric computation, we will do it in the loop here.
        compute_metrics = self.compute_metrics
        self.compute_metrics = None
        start_time = time.time()
        eval_loop = self.prediction_loop if self.args.use_legacy_prediction_loop else self.evaluation_loop
        try:
            output = eval_loop(
                eval_dataloader,
                description="Evaluation",
                # No point gathering the predictions if there are no metrics, otherwise we defer to
                # self.args.prediction_loss_only
                prediction_loss_only=True if compute_metrics is None else None,
                ignore_keys=ignore_keys,
                metric_key_prefix=metric_key_prefix,
            )
        finally:
            self.compute_metrics = compute_metrics
        total_batch_size = self.args.eval_batch_size * self.args.world_size
        if f"{metric_key_prefix}_jit_compilation_time" in output.metrics:
            start_time += output.metrics[f"{metric_key_prefix}_jit_compilation_time"]
        output.metrics.update(
            speed_metrics(
                metric_key_prefix,
                start_time,
                num_samples=output.num_samples,
                num_steps=math.ceil(output.num_samples / total_batch_size),
            )
        )

        if self.post_process_function is not None and self.compute_metrics is not None and
self.args.should_save:
```

```
            # Only the main node write the results by default
            eval_preds = self.post_process_function(eval_examples, eval_dataset, output)
            metrics = self.compute_metrics(eval_preds)

            # Prefix all keys with metric_key_prefix + '_'
            for key in list(metrics.keys()):
                if not key.startswith(f"{metric_key_prefix}_"):
                    metrics[f"{metric_key_prefix}_{key}"] = metrics.pop(key)

            metrics.update(output.metrics)
        else:
            metrics = output.metrics

        if self.args.should_log:
            # Only the main node log the results by default
            self.log(metrics)

        if self.args.tpu_metrics_debug or self.args.debug:
            # tpu-comment: Logging debug metrics for PyTorch/XLA (compile, execute times, ops, etc.)
            xm.master_print(met.metrics_report())

        self.control = self.callback_handler.on_evaluate(self.args, self.state, self.control, metrics)
        return metrics

    def predict(
        self, predict_dataset, predict_examples, ignore_keys=None, metric_key_prefix: str = "test",
**gen_kwargs
    ):
        self._gen_kwargs = gen_kwargs.copy()

        predict_dataloader = self.get_test_dataloader(predict_dataset)

        # Temporarily disable metric computation, we will do it in the loop here.
        compute_metrics = self.compute_metrics
        self.compute_metrics = None
        start_time = time.time()
        eval_loop = self.prediction_loop if self.args.use_legacy_prediction_loop else self.evaluation_loop
        try:
            output = eval_loop(
                predict_dataloader,
                description="Prediction",
                # No point gathering the predictions if there are no metrics, otherwise we defer to
                # self.args.prediction_loss_only
                prediction_loss_only=True if compute_metrics is None else None,
                ignore_keys=ignore_keys,
                metric_key_prefix=metric_key_prefix,
            )
        finally:
            self.compute_metrics = compute_metrics
```

```
        total_batch_size = self.args.eval_batch_size * self.args.world_size
        if f"{metric_key_prefix}_jit_compilation_time" in output.metrics:
            start_time += output.metrics[f"{metric_key_prefix}_jit_compilation_time"]
        output.metrics.update(
            speed_metrics(
                metric_key_prefix,
                start_time,
                num_samples=output.num_samples,
                num_steps=math.ceil(output.num_samples / total_batch_size),
            )
        )
        if self.post_process_function is None or self.compute_metrics is None:
            return output

        predictions = self.post_process_function(predict_examples, predict_dataset, output, "predict")
        metrics = self.compute_metrics(predictions)

        # Prefix all keys with metric_key_prefix + '_'
        for key in list(metrics.keys()):
            if not key.startswith(f"{metric_key_prefix}_"):
                metrics[f"{metric_key_prefix}_{key}"] = metrics.pop(key)
        metrics.update(output.metrics)
        return PredictionOutput(predictions=predictions.predictions, label_ids=predictions.label_ids,
metrics=metrics)
```