Question Generation System and Question Answering System

Juhyeon Nam **Chanwoo Kim** **Donggeun Kim** **Minjeong Kim**
Carnegie Mellon University / 5000 Forbes Ave, Pittsburgh, PA 15213
{juhyeonn, chanwoo2, donggeuk, minjeong}@andrew.cmu.edu

Abstract

This project aims to create a question generation (QG) and answering (QA) system. The QG system consists of three pipelines, each dealing with 'Wh-Questions', 'Yes/No-Questions', and 'Multi-hop Yes/No-Questions'. The QA system can produce answers in an extractive and generative way by selecting a suitable model based on the question-type classifier. We compared the perplexity of generated questions from our system and human-generated questions to evaluate the proficiency of question sentences. Also, we measure the BLEU score of generated answers from our system and those from baseline models to assess the accuracy of our QA system. The evaluation was performed on the human-generated question-answer pairs from Wikipedia-like documents with three levels of inference difficulty. In conclusion, our QG and QA systems outperform baselines.

1 Introduction

Question generation (QG) and question answering (QA) are two of the most important tasks in natural language processing. QG is used for many applications, including data augmentation for QA and machine reading comprehension. Automatic conversation for the human-computer dialogue process and an intelligent tutor are popular applications. QA also has lots of prominent applications, including assisting tools for fact-checking, customer service, technical service, market research, and so on.

Basic QG systems and QA systems usually get one paragraph of input. However, in this project, the input of each system is one whole Wikipedia-like document. Hence, systems are designed to properly deal with it by using suitable summarization. Furthermore, the QG system is composed of three main pipelines so that each can produce 'Wh-Questions', 'Yes/No-Questions', and 'Multi-hop Yes/No-Questions. Hence, the variety of generated questions increased. Moreover, the QA system comprises two main pipelines. Each of the pipelines deals with extractive questions and generative questions. Note that we narrowed the concept of generative questions to only the closed ones since the open ones are too vague to consider.

2 System Architecture

Our system consists of QG and QA systems.

2.1 Question Generation

To generate various types of questions, the overall structure of the QG system is defined by three major pipelines, as shown in Figure 1. The first pipeline generates 'Wh-Questions', which are preceded by a wh-word ('what', 'when', 'where', 'who', 'whom', 'which', 'whose', 'why', and 'how') and require more information than a simple yes or no in reply. And the second pipeline generates 'Yes/No questions', which can be answered with yes or no. Lastly, the third pipeline generates 'Multi-hop Yes/No questions', which also can be answered with yes or no, but require integration of information across multiple lines in an input document.

For the pipelines, there are three different types of input preprocessing: 'Keyword Extraction', 'Key Sentence Extraction' and 'Document Summarization'. And there are two different types of processing, which include transformer-based model and rule-based algorithm. Lastly, there is a postprocessing filter for question selection.

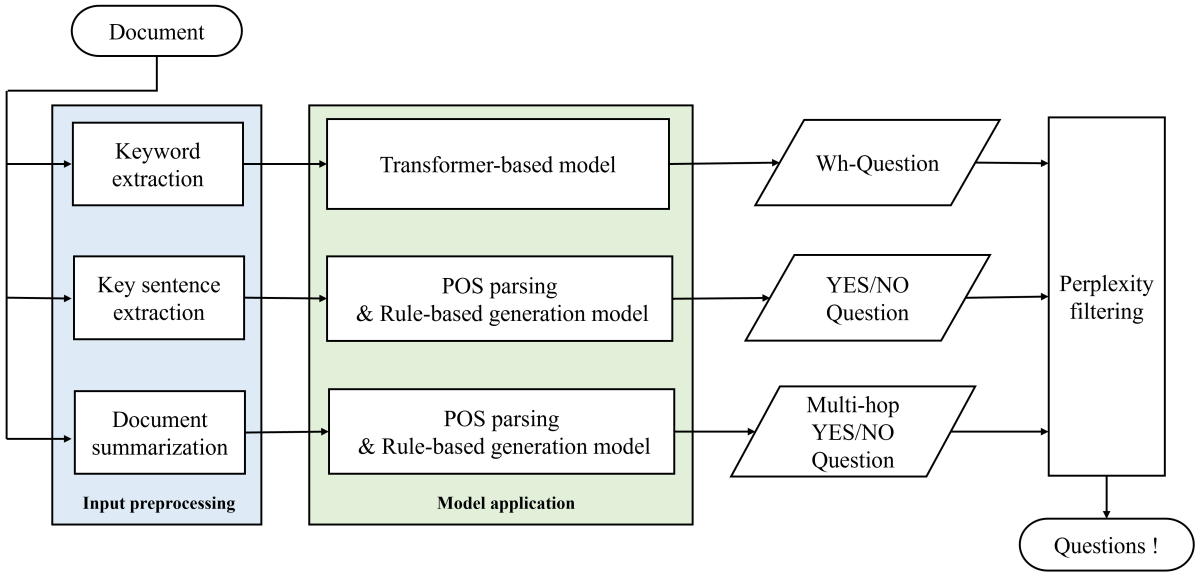

Figure 1: QG system Diagram

2.1.1 Wh-Question

For the pre-processing, a keyword is extracted from the input document. Keyword extraction utilizes model KeyBERT[1] which leverages BERT(Devlin et al., 2018) embeddings to create keywords in the document. The keyword is used for setting the primary context of an answer to 'Wh-Questions'. Hence, the keyword is processed by T5[2](Raffel et al., 2019), an encoder-decoder model pre-trained on both supervised and unsupervised tasks, to output a question and answer pair.

2.1.2 Yes/No Question

At first, input document is preprocessed to output a key sentence of the document. For this, a sentence ranking algorithm[3] is used. It calculates cosine-similarities between sentences in the document, and then chooses the most significant sentences. The key sentence is then transformed into 'Yes/No question' by our own sentence-to-question algorithm.

In the beginning, the sentence is parsed as POS (Part of Speech) using Spacy[4]. Then, based on the POS token, the algorithm first classifies the sentence by considering whether it has a pronoun or auxiliary verb. If there is a pronoun in the sentence, it omits the sentence. It is because if the sentence is transformed into a question with a pronoun, readers can't tell what the pronoun indicates. Otherwise, if there is an auxiliary verb in the sentence, the algorithm inverts the positions of the subject and the auxiliary verb. In addition, if there is no auxiliary verb in the sentence, the algorithm finds the verbs' tenses. The algorithm then adds 'Did' if past, 'Does' if 3rd person singular present, 'Do' if non-3rd person singular present in front of the sentence. It then lemmatizes the verbs. Lastly, the algorithm changes the last punctuation into a question mark.

Because the tasks above only generate questions with 'yes' answers, the algorithm also attempts to find an antonym to the verb using WordNet[5](Fellbaum, 1998). WordNet functions as a taxonomy exploring tool. Hence, if there is an antonym for the verb in the question, we replace the verb with the antonym so that the answer to the question becomes 'no'.

2.1.3 Multi-hop Yes/No Question

The input document is preprocessed using transformer-based 'PegasusTokenizer' and 'PegasusForConditionalGeneration'. It outputs a summary sentence of the input, so if the algorithm generates a question with the sentence, the question can be answered only if the reader knows information across the whole input. Then, the sentence is processed similarly to the 'Yes/No Ques-

[1] https://github.com/MaartenGr/KeyBERT

[2] https://github.com/google-research/text-to-text-transfer-transformer

[3] https://gist.github.com/akashp1712/

[4] https://spacy.io/models

[5] https://github.com/goodmami/wn

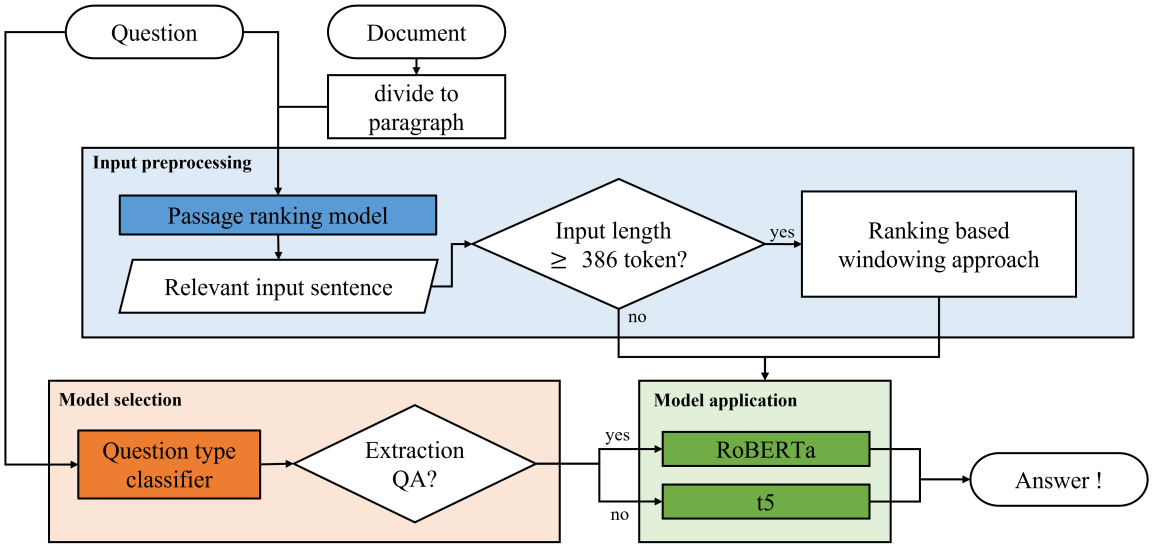

Figure 2: QA System Diagram

tion' above.

2.1.4 Perplexity Filtering

Since our purpose is to output a specific number of questions given an input document, it needs to select that number of questions if there are more than that number of questions. To choose more plausible questions, the algorithm calculates the perplexity of each question using HuggingFace module[6]. Since the perplexity is exponentiated average negative log-likelihood of a sequence calculated with exponent base, the algorithm chooses questions with the lowest perplexities as the output questions.

2.2 Question Answering

QA system is shown in Figure 2. It receives a question and the corresponding document as inputs.

2.2.1 Ranking-based Windowing Approach

Additionally, our QA model includes a preprocessing step of input documents for generating effective answers. This is because pre-trained models have token limitations in input which is usually shorter than the total length of the document. For example, RoBERTa accepts a token length of 386 at maximum, which is problematic in cases where the answer lies in the later part of a text. Our data visualization in Figure 3 supports our process showing that all the documents in the dataset have more than 386 tokens, and out of 5203 paragraphs in the whole dataset, 264 (5%) of them have more than 386 tokens. Therefore, truncating the input document is essential. Usually, sentences that are likely to contain the answer to the question are just a part of the document, not the whole, which makes it reasonable for us to use only part of the document as a model input. For this, we implement a ranking-based windowing approach. We use the module Sentence Transformer[7] to calculate similarities between the question and all the sentences in the document and then find the sentence with the maximum similarity. We extract sentences before and after the sentence with predefined window size to deal with 'HARD' questions requiring inference across multiple sentences. Finally, those sentences are concatenated and made as input for the model.

2.2.2 Question Type Classifier

Questions we deal with are classified into three levels of inference difficulty. 'EASY' represents questions that can be answered simply by extracting an answer from a sentence similar to the question, including 'Yes/No questions'. 'MEDIUM' requires additional inference or more processing from the question. 'HARD' requires inference

[6] https://huggingface.co/spaces/evaluate-metric/perplexity

[7] https://github.com/UKPLab/sentence-transformers/blob/master/index.rst

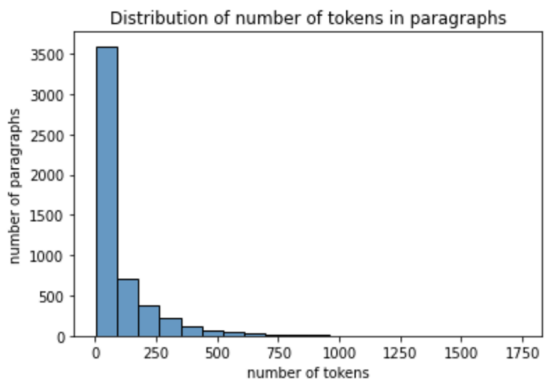

Figure 3: Token visualization

across multiple sentences with 'hops'.

To deal these three questions, we think of two types of QA variants. One is the extractive QA, in which the model extracts the answer from the input document. This includes answers that can be simply found in sentences. This variant can answer 'EASY' questions. The other is the generative QA in which the model generates the answer directly based on the input document context. This includes answers that require inference across sentences and true/false, yes/no answers. This variant can answer 'MEDIUM' or 'HARD' questions.

We implement a hybrid of two pre-trained QA models to deal with each QA variant. For the extractive QA case, we use RoBERTa-base-SQuAD[8] model.(Liu et al., 2019) However, this model finds it hard to answer yes/no questions that are not directly revealed in the input document but should be inferred from the context. Therefore, we add the T5 model to deal with generative QA cases. We expect the two models to complement each other in dealing with two types of QA variants.

3 Experiments

For the evaluation of the total system, we plan to use the question-answer pairs dataset from hw1. The dataset consists of human-generated question-answer pairs from Wikipedia-like documents with three levels of inference difficulty. We use all 7,100 question-answer pairs for evaluation.

3.1 Evaluating Question Generation System

We compare the perplexity of generated questions from our system with the perplexity of human-generated questions to evaluate the proficiency of question sentences. The results are shown in Table 1. The table shows the average perplexity of generated questions in each document class and all generated questions. Questions from our QG system have lower perplexity in every document class compared to human-generated questions. The individual perplexity of every document can be found in Figure 4.

	Ours	Human
chinese_dynasties	73.64	440.18
constellations	77.89	217.57
indian_states	60.01	147.27
languages	75.41	204.77
pokemon_characters	71.24	157.31
total avg	**71.64**	**233.42**

Table 1: Average perplexity of generated questions by our system and human annotators in each document class

3.2 Evaluating Question Answering System

We measure the BLEU(Papineni et al., 2002) score of generated answers from our system and those from baseline models to evaluate the accuracy of our QA system. The baselines are RoBERTa(Liu et al., 2019) and T5(Raffel et al., 2019). Both models were pre-trained on the SQuAD(Rajpurkar et al., 2016) dataset, which was posed by crowdworkers on a set of Wikipedia articles. Therefore, consistent performance can also be expected for our evaluation dataset, since human annotators posed it on a subset of Wikipedia articles. Evaluation results are shown in Table 2. We also evaluate the effectiveness of our ranking-based paragraph windowing approach by adding it to the baseline models. The table shows that our system consistently outperforms any of the baselines regardless of the difficulty of the questions. We also confirm that our ranking-based paragraph windowing approach improves the BLEU scores for the T5 baseline. However, it degrades the performance of the RoBERTa baseline.

4 Discussion

There are some difficult aspects of implementing QA and QG.

[8] https://huggingface.co/deepset/roberta-base-squad2

	Pre-trained	Easy	Medium	Hard	Overall
Baseline	RoBERTa	0.6084	0.2792	0.2862	0.3801
	T5	0.3965	0.3046	0.2481	0.3152
Windowing approach	RoBERTa	0.5254	0.2391	0.2353	0.3238
	T5	0.5899	0.4298	0.3557	0.4556
Ours (full pipeline)		**0.6180**	**0.5232**	**0.37931**	**0.4800**

Table 2: BLEU scores evaluation for question answering system

4.1 QG: Issues of pronouns

Most QG methods are about mapping a sentence to a corresponding question. However, almost all of the sentences in a document use pronouns because only limited objects appear in a document, and they are mentioned multiple times. If a sentence containing pronouns is directly mapped into a corresponding question, the question will not be plausible since readers can not tell what the pronouns indicate. Therefore, a proper preprocessing method should be devised, which substitutes pronouns in a sentence into the corresponding original nouns.

4.2 QA: Difficulty of answering generative questions

Answering an extractive question is mostly about finding the object that can substitute the 'Wh' part of the question and creating an answer based on it. Therefore, it is easier than answering a generative question. However, in a generative question, an answer can not be formed by adequately choosing the words in the sentence because there is no answer explicitly in the sentence. Hence, one has to infer the answer by understanding the context of the sentence. As the difficulty in answering generative questions is higher, QA in AI models also has poorer outputs in this case. We need to explore methods to deal with these problems.

5 Conclusions

This paper proposes a QA and QG system consisting of a pipeline of pre-trained models. The former is designed to make various questions, including 'Wh-Questions', 'Yes/No-Questions', and 'Multi-hop Yes/No-Questions'. The latter is created to provide answers for both extractive and generative questions. We evaluated both systems using a dataset of 7,100 question-answer pairs in hw1. Finally, we demonstrated that our systems outperformed baselines.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. cite arxiv:1810.04805Comment: 13 pages.

Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. Bradford Books.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250.

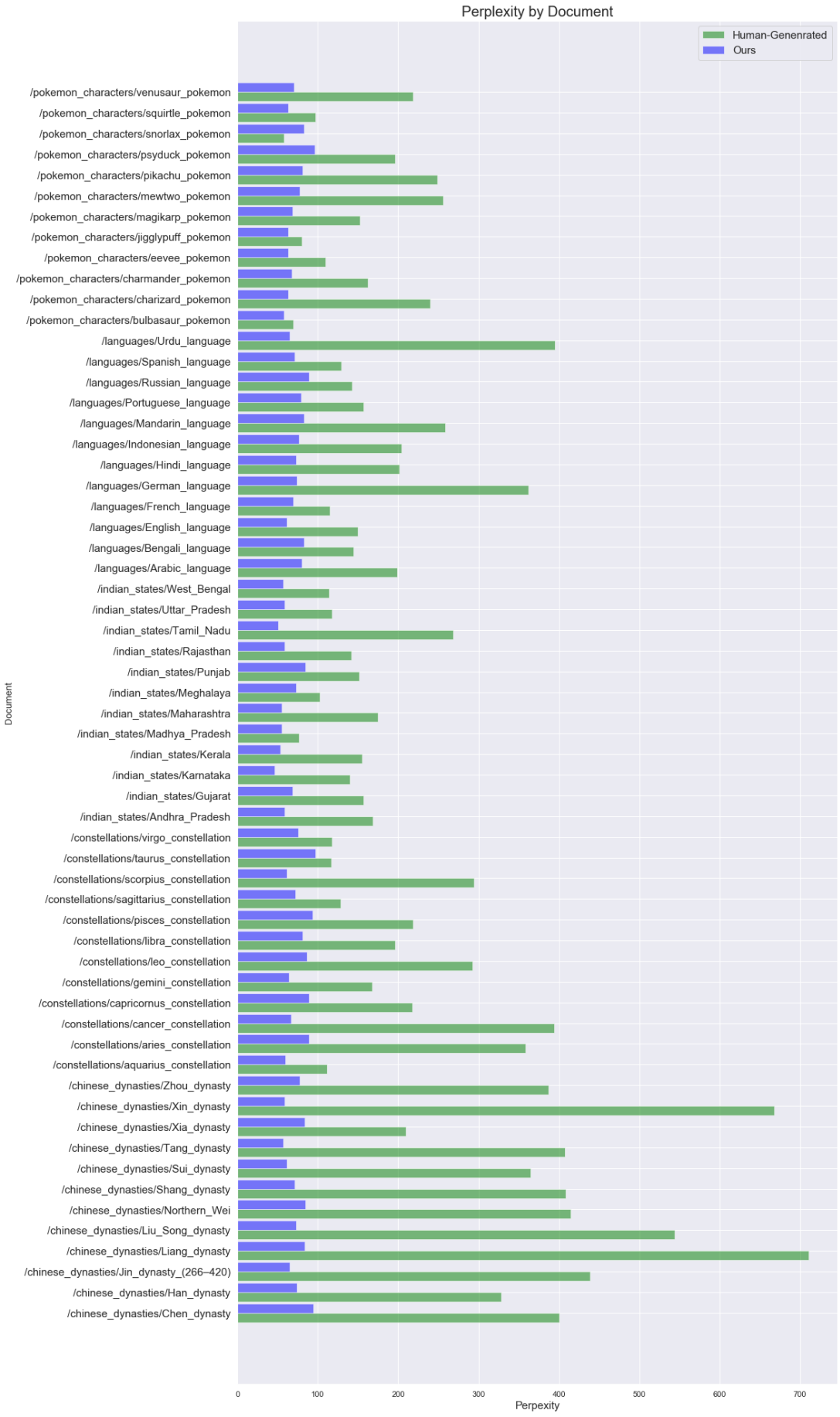

Figure 4: Perplexity of questions generated from every document