Exploration of different QA prompts for answering factoid questions

Camélia Guerraoui
C2IM2015 / GSIS Tohoku University

Abstract

In this project, we explore the performance of a question-answering (QA) system by designing efficient prompts for factoid questions in English and Japanese. The goal is to identify practices for designing prompts that can enhance the accuracy of the system. Four factors are explored, including rephrasing the prompt, few-shots learning, metrics for similarity, and preprocessing instances before computing similarity. The results of this project will provide insights into how to optimize the design of prompts for QA systems.

1 Introduction

A factoid question is a type of question that can be answered with a single, concise verifiable response. These types of questions usually ask for basic facts or information about a specific topic, such as "What is the capital of France?" or "Who is the president of Spain?" Factoid questions are typically objective and do not require complex analysis or interpretation, making them well-suited for automatic answering systems.

In this project, we explore how to build an efficient prompt for a question-answering (QA) system that can answer natural questions in both English and Japanese. The goal is to study the impact of different types of prompts on the performance and effectiveness of the system in providing accurate and relevant answers. The study aims to identify practices for designing prompts that can improve user satisfaction and enhance the overall performance of the question-answer system.

To that aim, we use the following environment available at https://github.com/cl-tohoku/AIO3_GPT_baseline_for_NLP/tree/main and deploy it on a VertexAI platform.

2 Requirements

Our QA system has to follow these requirements:

- No use of other QA systems
- No mention of the correct answer in the prompt
- No modification of the language models' parameters
- No regeneration of answers after the model generates them
- Only able to select the correct answer from the generated text
- Need to answer the question in an off-line setting
- Answer all the questions in the test data set within 30 minutes on the VertexAI platform

3 Data set

For each language (i.e., Japanese and English), We have at our disposal many factoid questions separated into a development and a test set. Table 1 shows how many instances each data sets contain.

	English	Japanese
Development set	86 821	22 335
Test set	500	500

Table 1: Number of instances in each data set

An instance is made of an id, a question, and a list of possible answers, for example:

```
{
    "qid": "squad_v2.0_train-3401",
    "question": "What uncharacteristic attitude did China display?",
    "answers": ["openness"]
}
```

4 Strategical approaches

In this section, we describe different strategies we tried to improve the accuracy of our QA system.

4.1 Rephrasing the prompt

Initially, the current prompt was '**Question: [INPUT]? Answer: [**' in English and '**質問：[INPUT]？回答：「**' in Japanese. However, the model can have some difficulties understanding the task with this kind of prompt. Therefore we tried other formulations described in table 2:

Id	Prompt	Accuracy
1	Question: [INPUT]? Answer: [	0.450
2	The answer of '[INPUT]' is [	0.420
3	Answer the question '[INPUT]'. It is [	0.300
4	Answer the question '[INPUT]'. The answer is [	0.580
5	Answer the question '[INPUT]' by a fact. The answer is [	0.380

Table 2: Accuracy for different prompts tested with a sample of 100 questions in English

Id	Prompt	Accuracy
1	質問：[INPUT]？回答：「	0.270
2	答えてください「[INPUT]」：「	0.000
3	[INPUT]質問を答える：「	0.220
4	[INPUT]の回答は：「	0.280
5	[INPUT]　回答は：「	0.290

Table 3: Accuracy for different prompts tested with a sample of 100 questions in Japanese

We can see that different phrases lead to different results highlighting how important a precise and grammatically correct prompt is to improve the language model's performance. We will continue our analysis with the prompt that gives the best accuracy.

We realized that some prompts made the model repeat itself indefinitely. For example, the question ホッケーは1チーム11人ですが、アイスホッケーは1チーム何人? with the Japanese prompt **3** lead to the prediction アイスホッケーは1チーム11人ですが、ホッケーは1チーム11人ではありません。 again and again, until the maximum length of the output is reached. This behavior might result from an unclear formalization of the expected output.

4.2 Adding examples

Few-shots After looking at the false results obtained with the English prompt 4, we observed that in some cases, although the language model's answer seems correct, its format is not (cf Table 4 and therefore this answer is considered as wrong.

Question	Given answer	Expected answers
What types of programs help to redistribute wealth?	social welfare programs	social welfare
Where does Hamas originate?	the Palestinian territories	Palestine
As of 2008, about what percentage of Swedish students attended private schools?	10.5%	10 or 10%
Where was Friedrich Ratzel born?	Bremen, Germany	Germany

Table 4: Examples of wrong format of results given with the English prompt 4

Even if the answer is correct, the model doesn't know how the answer should be phrased. Therefore it is important to provide examples. A good example is adding at the beginning of the prompt indicates to the model what kind of answers the end-user is expecting. This method is also known as few-shot learning, or N-shot learning, described in Brown et al. (2020) and in Liu et al. (2021).

N-shot refers to a type of machine learning evaluation task where the model is required to learn and generalize from a very small number of examples. In N-shot learning, the model is presented with a set of N-labeled examples (or "shots") for a specific task and then evaluated on its ability to perform the same task on previously unseen examples. This type of evaluation is typically used for few-shot or one-shot learning, where the goal is to train a model with a limited amount of labeled data and evaluate its ability to generalize to new examples with a high degree of accuracy.

BLEU To implement a 1-shot setting, we chose to dynamically find the best example, i.e., the ques-

tion and answers which are the most similar to the current question and answers we want to predict. Toward that aim, we use the question available in the development set (cf Appendix C).

As our first step, we use the BLEU score metric to calculate the similarity between the development instances and the current question.

The BLEU score described in Papineni et al. (2002) is a commonly used evaluation metric in the field of machine translation and text generation. It measures the quality of a machine-generated text by comparing it to a set of reference texts, and is used to determine the degree of similarity between the input text and the reference text. A higher BLEU score indicates a higher degree of similarity between the input text and the reference text, and therefore a possible higher quality of the prompt.

1-shot After calculating the similarity scores, we select the instance from the development set with the highest BLEU score and formulate a prompt following the best prompt formats described in tables 2 and 3. Here are the resulting templates for a 1-shot prompt in English and in Japanese:

English:

```
Answer the question '{max_quest}'.
The answer is {max_answers}.
<|endoftext|>
Answer the question '[INPUT]'.
The answer is [
```

Japanese:

```
{max_quest} 回答は{max_answers} です。
<s>
{question} 回答は「
```

We can see that the example and the question are separated with a special token, **<|endoftext|>** in English and **<s>** in Japanese. This token is the output token used by the language model.

The accuracy for the English settings jumps from 0.580 to **0.640** for a sample of 100 test questions, whereas for the Japanese, the accuracy changes from 0.290 to **0.310**. If we take a look at the previous errors formulated in table 4, we obtain the results described in table 5. Specifically, among those four questions, only the answer to the last question has improved. However, the solution to question two is technically correct and more precise than the expected output, which makes us question the relevance of the expected answers.

2-shots We observed that one-shot significantly improved the model's accuracy. What about few-shots? We implement a version which this time takes the first two best examples found in the development set and evaluate it with a sample of 100 test question. Although the accuracy stays the same (i.e., 0.604), it takes twice as long to compute the final result. Indeed, it takes more computational time as we implemented a naive function that calculates, stores and sorts the similarity between each instance from the development set and the current question, and then selects the best two options. The implementation is the following:

Question	Given answer	Previous given answers
What types of programs help to redistribute wealth?	education	**social welfare programs**
Where does Hamas originate?	Gaza	Palestinian territories
As of 2008, about what percentage of Swedish students attended private schools?	1.5%	**10.5%**
Where was Friedrich Ratzel born?	**Germany**	Bremen, Germany

Table 5: Evolution of outputs with a 1-shot learning. The most correct answers are highlighted

```
def _calculate_bleu(dev_set: object,
    tokens: str, n_shot: int= _N) -> int
    :
    scores = []
    chencherry = SmoothingFunction()
    for ind in dev_set.index:
        bleu = sentence_bleu(
            [dev_set["tokens"][ind]],
            tokens,
            smoothing_function=
    chencherry.method3
        )
        scores.append((ind, bleu))
    scores.sort(key=lambda t: t[1],
    reverse=True)
    return scores[:n_shot]

def _design_prompt(dev_set: object,
    n_shots: object, quest: str, markers
    :tuple) -> str:
    prompt = []
    for shot in n_shots:
        max_quest = dev_set['question'][
    shot[0]]
        max_answer =  dev_set['answers'
    ][shot[0]]
        prompt.append(f'{markers[0]} '{
    max_quest}' {markers[1]} {max_answer
```

```
    } {markers[2]} ')
    prompt.append(f'{markers[0]} {quest}
     {markers[1]} [')
    return ' '.join(prompt)
```

Although the accuracy did not change, the correct answers are not exactly the same. In some situations the second example does more harm than good and confuses the model. Indeed the BLEU score is calculated based on the precision of the machine-generated text, with respect to the n-grams (contiguous sequences of n words) in the reference text. The precision is determined by counting the number of matching n-grams between the machine-generated text and the reference text, and normalizing it by the total number of n-grams in the machine-generated text. Therefore, two questions with similar topics will have a high BLEU score, even if their form is very different (e.g., one asks about a nationality, whereas the other asks about a percentage).

As the overall accuracy does not improve with few-shots learning, we keep the one-shot setting.

4.3 Changing the similarity metrics

BLEU is widely used in the natural language processing community as a standard evaluation metric for machine translation and text generation systems, and has been shown to be a reliable and effective method for comparing the performance of different systems. However, it might not be the best-suited metric in our setting. Indeed as we have previously seen, we need to find similar questions content and form-wise. In this section, we explore the use of other metrics.

METEOR The Metric for Evaluation of Translation with Explicit ORdering (METEOR) made by Banerjee and Lavie (2005) is a precision-based metric for evaluating machine-translation output. It overcomes some of the pitfalls of the BLEU score. While BLEU focuses mainly on n-gram overlap between the reference and candidate translations, METEOR considers a more comprehensive set of factors, including word order, synonymy, and stemming. The metric also correlates well with human judgment at the sentence or segment level. This differs from the BLEU metric because BLEU seeks correlation at the corpus level. Herefore, METEOR might be a better metric for the English corpus.

Although, with the METEOR metric, we obtain a slightly better accuracy (0.660), it takes 34 minutes to compute the result instead of one minute, which is an important increase. Hence, we will continue to use the BLEU metric.

RIBES The RIBES metric made by Isozaki et al. (2010) doesn't rely on languages having the same qualities as English. It was designed to be more informative for Asian languages like Japanese and Chinese. We try this metric for the Japanese corpus, and obtain an accuracy of **0.250** in 18 minutes. Even if this metric is more suited for Asian languages, it is not designed for a question-answering task, but more language translation. RIBES works well for distant language pairs such as Japanese and English, which have different word orders.

4.4 Optimizing BLEU Score

After using different metrics, we realized that BLEU gives the best accuracy. However, BLEU relies on different parameters, that can be optimized.

Smoothing BLEU works at a corpus level and compares n-grams. When the corpus is small, it is better to use a smoothing method. In our case, the references and the hypotheses are only one question, implying that we should use a smoothing method to have a non-zero similarity.

We use the implementation of the BLEU score from the **nltk** library. We have seven different smoothing methods[1].

We compute the accuracy for 100 test questions in the English set and always obtain the same accuracy. Therefore, the smoothing method does not impact the overall accuracy of the model.

4.5 Input Preprocessing

BLEU metric needs two lists of words as input. Therefore we need to tokenize the questions.

In English We tokenize the questions by splitting each words by space and removing the empty words.

We do some standard text preprocessing, such as changing all characters to lowercase, removing the punctuation, the stop words, and the pronouns, and using the lemmas. However, each step individually made the accuracy worst. Indeed, we want to find an instance from the development set with the same structure as our current question.

In Japanese We used the tagger Fugashi from McCann (2020) to tokenize our sentence. To have

[1]More information available at https://www.nltk.org/api/nltk.translate.bleu_score.html

a non-zero BLEU score, we transform each word into its lemma.

4.6 Balance between time and resources

The development sets are big corpora. Comparing each instance of the development set to the current question might be too time-consuming, even though we have a higher chance of finding an adequate example. We randomly sample the development set into different sizes and compute the accuracy for each sample (cf table 6, appendix C). As expected, the more data we use, the higher the accuracy is and the more time is needed to compute the results. Therefore, finding a good balance between computational time and accuracy is important to get the best performance.

Sample size	Accuracy	Time
25%	0.286	9 min 44
50%	0.310	18 min 16
75%	0.346	26 min 51
100%	0.354	34 min 40

Table 6: Accuracy and computational time depending on the number of instances from the development set for the whole Japanese test set

5 Future Directions

We realized that in some situation the example is similar content-wise but not form-wise. For example, although the two questions:

```
Answer the question "What is the rate of interreligious marriage in the United States?". The answer is ['just under 50\%']. <|endoftext|>
Answer the question "What happened with the rate of flow in the Rhine with the straightening program?". The answer is [
```

are a bit similar because they both talk about rate, but they are completely different questions. It may have been a better approach to distinguish questions by categories and types and then compute the similarity between questions from the same category. This approach will allow the model to select more similar questions and perform better.

Moreover, it will also improve the time complexity, as the model will compute the similarity between the current question and a smaller set of examples.

To improve the overall time complexity, we could also save the output of function *preprocess* (cf Appendix C) in an external file and load this output when initializing the model.

Using a different metric for computing the similarity might help to get better performances. BLEU and METEOR rely mainly on n-grams, meaning they do not consider the structure of a sentence. On the other side, the subtree metric (STM), described in Liu and Gildea (2005), compares the parses for the reference and output translations and penalizes outputs with different syntactic structures. As we want to find a similar question with a similar structure, we think this metric might be more accurate.

6 Conclusion

We see in this project that there are four aspects of prompting that impact the overall performance of a QA system:

- *Prompt's phrasing*: The model needs to clearly understand the task.
- *Few-shots*: If the model has examples, it can use them as templates and generate a better answer.
- *Metrics for similarity*: the way we compute the similarity between two questions to select the best example plays a crucial role. It is important to select a metric that considers the sentence's structure.
- *Preprocessing*: not only the similarity metric is important, but also the input to calculate the similarity. Questions such as "should upper case and the lower case be considered the same?" or "is the punctuation important?" need to be thought about when designing the prompt

References

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Hideki Isozaki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada. 2010. Automatic evaluation of translation quality for distant language pairs. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 944–952, Cambridge, MA. Association for Computational Linguistics.

Ding Liu and Daniel Gildea. 2005. Syntactic features for evaluation of machine translation. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 25–32, Ann Arbor, Michigan. Association for Computational Linguistics.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing.

Paul McCann. 2020. fugashi, a tool for tokenizing japanese in python.

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 of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

A Utils.py - Librairies

```
from __future__ import annotations
import re
from typing import Any
import json
import pandas as pd

import nltk
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
import fugashi
```

B Utils.py - Constant

```
_TAGGER = fugashi.Tagger('-Owakati')
_EN_MARK = "en"
_JA_MARK = "ja"
```

C Utils.py - Preprocess

```
def _split_question(row: str, lang: str) -> list:
    tokens = []
    if lang == _EN_MARK:
        for word in row.split(" "):
            if len(word) > 0:
                tokens.append(word)
    elif lang == _JA_MARK:
        _TAGGER.parse(row)
        for word in _TAGGER(row):
            tokens.append(word.feature.lemma)
    else:
        assert 0, lang
    return tokens

def preprocess(lang: str) -> dict:
    df = pd.read_json(f'data/LecNLP_dev_{lang}.jsonl', lines=True)
    if lang==_EN_MARK:
        df = df.sample(n=5000, random_state=1)
    elif lang==_JA_MARK:
        df = df.sample(frac=0.50, random_state=1)
    else:
        assert 0, lang
    df['tokens'] = df['question'].apply(lambda x: _split_question(x, lang))
    return df
```

D Utils.py - Add prompt

```
    def _calculate_bleu(dev_set: object, tokens: str) -> int:
    max_bleu = -1
    max_quest = dev_set["question"].iloc[0]
    max_answers = dev_set["answers"].iloc[0]

    chencherry = SmoothingFunction()

    for ind in dev_set.index:
        bleu = sentence_bleu(
            [dev_set["tokens"][ind]],
            tokens,
            smoothing_function=chencherry.method3
        )
        if max_bleu < bleu:
            max_quest = dev_set["question"][ind]
            max_answers =  dev_set["answers"][ind]
            max_bleu = bleu

    return max_quest, max_answers

def add_prompt(question: str, lang: str, dev_set: object, **kwargs: dict[str, Any]) -> str:
    tokens = _split_question(question, lang)
    if lang == _EN_MARK:
        max_quest, max_answers = _calculate_bleu(dev_set, tokens)
        prompt = f'Answer the question "{max_quest}". The answer is {max_answers}. <|endoftext|> Answer the question "{question}". The answer is ['
    elif lang == _JA_MARK:
        max_quest, max_answers = _calculate_bleu(dev_set, tokens)
        prompt = f'{max_quest }　回答は{max_answers} です。<s>question　回答は「'
    else:
        assert 0, lang
    return prompt
```

E Utils.py - Extract answer

```
def extract_answer(output: str, lang: str) -> str:
    if lang == _EN_MARK:
        return re.findall("\[(.*?)\]", output)[-1]
    elif lang == _JA_MARK:
        return re.findall("「(.*?)」", output)[-1]
    else:
        assert 0, lang
```

F Output.ipynb - Save output

```
import pandas as pd

def _read_result(lang: str) -> object:
    predictions = pd.read_json(f'outputs/LecNLP_test_{lang}_prediction.jsonl', lines=True)
    predictions.drop("qid", axis=1, inplace=True)
    return predictions

def _save_predictions_to_csv(data_type: bool, lang: str, df: object) -> None:
    pred = df[df.correct == data_type].copy()
```

```
    pred.drop("correct", axis=1, inplace=True)
    pred.to_csv(f"outputs/pred_{lang}_{data_type}.tsv", sep = "\t", index=False)

_LANGS = ["en", "ja"]

for lang in _LANGS:
    predictions = _read_result(lang)
    _save_predictions_to_csv(0, lang, predictions)
    _save_predictions_to_csv(1, lang, predictions)
```