11011 words | 28 minutes

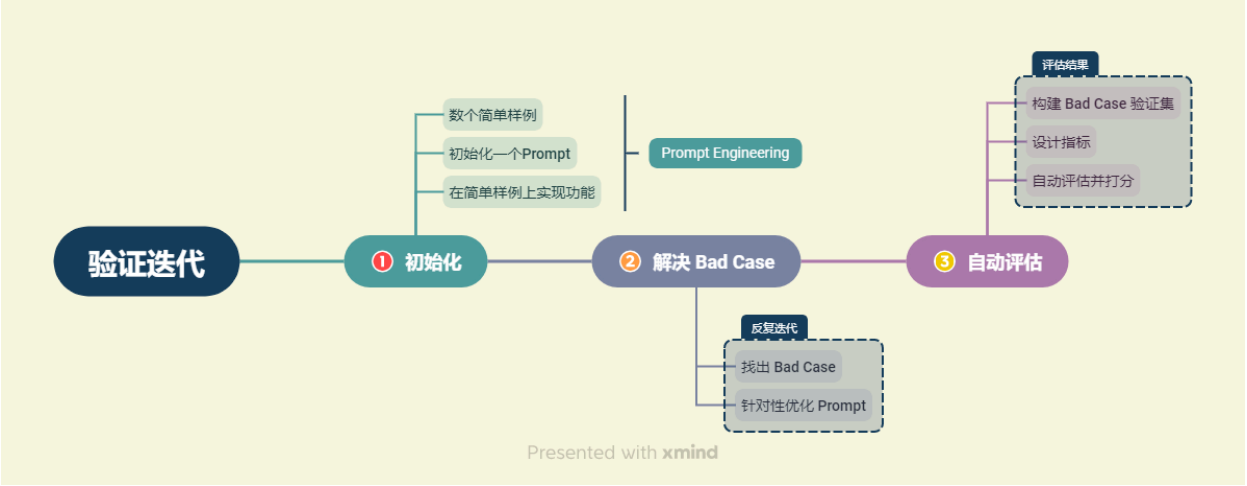
How to evaluate LLM applications

Note: The source code corresponding to this article is in the Github open source project LLM Universe . Readers are welcome to download and run it. You are welcome to give us a Star~

1. General ideas for verification and evaluation

Now, we have built a simple, generalized large model application. Looking back at the entire development process, we can find that large model development, which focuses on calling and using large models, pays more attention to verification and iteration than traditional AI development. Since you can quickly build an LLM-based application, define a prompt in a few minutes, and get feedback results in a few hours, it would be extremely cumbersome to stop and collect a thousand test samples. Because now, you can get results without any training samples.

So, when building an application with LLM, you might go through the following process: First, you tweak Prompt on a small sample of one to three examples, trying to get it to work on those samples. Then, as you test the system further, you might run into some tricky examples that can't be solved by Prompt or the algorithm. That's the challenge facing developers who build applications with LLM. In this case, you can add these extra few examples to the set you are testing, organically adding other difficult examples. Eventually, you will add enough of these examples to your gradually expanding development set that it becomes a bit inconvenient to manually run every example to test Prompt. Then, you start to develop some metrics for measuring the performance of these small sample sets, such as average accuracy. The interesting thing about this process is that if you feel that your system is good enough, you can stop at any time and stop improving it. In fact, many deployed applications stop at step one or two, and they work very well.



In this chapter, we will introduce the general methods of large model application verification and evaluation one by one, and design the verification and iteration process of this project to optimize the application functions. However, please note that since system evaluation and optimization is a topic closely related to business, this chapter mainly introduces the theory, and readers are welcome to actively practice and explore on their own.

We will first introduce several methods for evaluating large model development. For tasks with simple standard answers, evaluation is easy to implement; however, large model development generally requires the implementation of complex generation tasks. How to implement evaluation without simple answers or even standard answers, and accurately reflect the effect of the application, we will briefly introduce several methods.

As we continue to find bad cases and make targeted optimizations, we can gradually add these bad cases to the validation set, thus forming a validation set with a certain number of samples. For such a validation set, it is impractical to evaluate them one by one. We need an automatic evaluation method to achieve an overall evaluation of the performance on the validation set.

After mastering the general idea, we will explore how to evaluate and optimize application performance in large model applications based on the RAG paradigm. Since large model applications developed based on the RAG paradigm generally include two core parts: retrieval and generation, our evaluation and optimization will also focus on these two parts, respectively, to optimize the system retrieval accuracy and the generation quality under the given material.

In each section, we will first introduce some tips on how to find bad cases, as well as general ideas for optimizing retrieval or prompts for bad cases. Note that in this process, you should always keep in mind the series of large model development principles and techniques we described in the previous chapters, and always ensure that the optimized system will not make mistakes on the samples that originally performed well.

Verification iteration is an essential step in building LLM-centric applications. By continuously finding bad cases, adjusting prompts or optimizing retrieval performance, we can drive the

application to achieve the performance and accuracy we want. Next, we will briefly introduce several methods for large model development evaluation, and summarize the general idea from targeted optimization of a few bad cases to overall automated evaluation.

2. Large Model Evaluation Method

In the specific development of large-model applications, we can find Bad Cases and continuously optimize the prompt or retrieval architecture to solve Bad Cases, thereby optimizing the performance of the system. We will add every Bad Case we find to our verification set. After each optimization, we will re-verify all verification cases in the verification set to ensure that the optimized system will not lose its capabilities or performance degradation on the original Good Case. When the verification set is small, we can use the manual evaluation method, that is, manually evaluate the quality of the system output for each verification case in the verification set; however, as the verification set continues to expand with the optimization of the system, its size will continue to increase, so that the time and labor cost of manual evaluation will expand to a level that we cannot accept. Therefore, we need to use an automatic evaluation method to automatically evaluate the system's output quality for each verification case, thereby evaluating the overall performance of the system.

We will first introduce the general ideas of manual evaluation for reference, and then introduce the general methods of large model automatic evaluation in depth, and conduct actual verification on this system to comprehensively evaluate the performance of this system and prepare for further optimization and iteration of the system. Similarly, before officially starting, we first load our vector database and retrieval chain:

python

```
import sys
sys.path.append("../C3 搭建知识库") # 将父目录放入系统路径中

# 使用智谱 Embedding API，注意，需要将上一章实现的封装代码下载到本地
from zhipuai_embedding import ZhipuAIEmbeddings

from langchain.vectorstores.chroma import Chroma
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv, find_dotenv
import os

_ = load_dotenv(find_dotenv())    # read local .env file
zhipuai_api_key = os.environ['ZHIPUAI_API_KEY']
OPENAI_API_KEY = os.environ["OPENAI_API_KEY"]
```

```
# 定义 Embeddings
embedding = ZhipuAIEmbeddings()

# 向量数据库持久化路径
persist_directory = '../../data_base/vector_db/chroma'

# 加载数据库
vectordb = Chroma(
    persist_directory=persist_directory,  # 允许我们将persist_directory目录
    embedding_function=embedding
)

# 使用 OpenAI GPT-3.5 模型
llm = ChatOpenAI(model_name = "gpt-3.5-turbo", temperature = 0)
```

2.1 General Idea of Manual Evaluation

In the early stage of system development, the verification set is small, and the simplest and most intuitive method is to manually evaluate each verification case in the verification set. However, there are some basic principles and ideas for manual evaluation, which are briefly introduced here for learners' reference. But please note that the evaluation of the system is strongly related to the business, and the design of specific evaluation methods and dimensions needs to be considered in depth in combination with specific business.

Criterion 1: Quantitative Assessment

In order to ensure a good comparison of the performance of different versions of the system, quantitative evaluation indicators are very necessary. We should give a score to the answer to each verification case, and finally calculate the average score of all verification cases to get the score of this version of the system. The quantified dimension can be 0~~5, can also be 0~~100, depending on personal style and business circumstances.

The quantified evaluation indicators should have certain evaluation standards. For example, if condition A is met, the score can be y points to ensure relative consistency between evaluations by different evaluators.

For example, we give two verification cases:

① Who is the author of "The Pumpkin Book"?

② How should the Pumpkin Book be used?

Next, we use version A prompt (brief and to the point) and version B prompt (detailed and specific) to ask the model to answer:

```python
from langchain.prompts import PromptTemplate
from langchain.chains import RetrievalQA


template_v1 = """使用以下上下文来回答最后的问题。如果你不知道答案，就说你不知道，不要
案。最多使用三句话。尽量使答案简明扼要。总是在回答的最后说“谢谢你的提问！”。
{context}
问题: {question}
"""

QA_CHAIN_PROMPT = PromptTemplate(input_variables=["context","question"],
                                 template=template_v1)



qa_chain = RetrievalQA.from_chain_type(llm,
                                       retriever=vectordb.as_retriever(),
                                       return_source_documents=True,
                                       chain_type_kwargs={"prompt":QA_CHA

print("问题一: ")
question = "南瓜书和西瓜书有什么关系? "
result = qa_chain({"query": question})
print(result["result"])

print("问题二: ")
question = "应该如何使用南瓜书? "
result = qa_chain({"query": question})
print(result["result"])
```

```markup
问题一:
南瓜书是以西瓜书为前置知识进行解析和补充的，主要是对西瓜书中比较难理解的公式进行解析和推
问题二:
应该以西瓜书为主线，遇到推导不出来或看不懂的公式时再查阅南瓜书。不建议初学者深究第1章和
```

The above is the answer to the prompt in version A. Let's test version B:

```python
template_v2 = """使用以下上下文来回答最后的问题。如果你不知道答案，就说你不知道，不要
案。你应该使答案尽可能详细具体，但不要偏题。如果答案比较长，请酌情进行分段，以提高答案的
{context}
问题: {question}
有用的回答:"""

QA_CHAIN_PROMPT = PromptTemplate(input_variables=["context","question"],
                                 template=template_v2)

qa_chain = RetrievalQA.from_chain_type(llm,
                                       retriever=vectordb.as_retriever(),
                                       return_source_documents=True,
                                       chain_type_kwargs={"prompt":QA_CHA

print("问题一: ")
question = "南瓜书和西瓜书有什么关系? "
result = qa_chain({"query": question})
print(result["result"])

print("问题二: ")
question = "应该如何使用南瓜书? "
result = qa_chain({"query": question})
print(result["result"])
```

```markup
问题一:
南瓜书和西瓜书之间的关系是南瓜书是以西瓜书的内容为前置知识进行表述的。南瓜书的目的是对西
问题二:
应该将南瓜书作为西瓜书的补充，主要在遇到自己无法推导或理解的公式时进行查阅。对于初学机器
```

It can be seen that the prompt of version A has a better effect in case ①, but the prompt of version B has a better effect in case ②. If we do not quantify the evaluation indicators and only use relative evaluation, we cannot judge which prompt is better between version A and version B. Therefore, we need to find a prompt that performs better in all cases before further iteration; however, this is obviously very difficult and not conducive to our iterative optimization.

We can assign a score of 1 to 5 to each answer. For example, in the above case, we give version A answer ① a score of 4 and answer ② a score of 2, and version B answer ① a score of 3 and answer ② a score of 5; then, the average score of version A is 3 points, and the average score of version B is 4 points, so version B is better than version A.

Criterion 2 Multidimensional Assessment

The big model is a typical generative model, that is, its answer is a sentence generated by the model. Generally speaking, the answer of the big model needs to be evaluated in multiple dimensions. For example, in the personal knowledge base question and answer project of this project, users generally ask questions about the content of the personal knowledge base. The model's answer needs to meet the requirements of making full use of the content of the personal knowledge base, the answer is consistent with the question, the answer is true and effective, and the answer sentence is smooth. An excellent question and answer assistant should be able to answer users' questions well and ensure the correctness of the answers, while also being fully intelligent.

Therefore, we often need to start from multiple dimensions, design evaluation indicators for each dimension, and score each dimension to comprehensively evaluate system performance. At the same time, it should be noted that multi-dimensional evaluation should be effectively combined with quantitative evaluation. For each dimension, the same dimension can be set or different dimensions can be set, which should be fully combined with business reality.

For example, in this project, we can design the following evaluation dimensions:

① Correctness of knowledge search. This dimension requires checking the intermediate results of the system's search for relevant knowledge fragments from the vector database to evaluate whether the knowledge fragments found by the system can answer the question. This dimension is evaluated on a 0-1 scale, that is, a score of 0 means that the knowledge fragments found cannot answer the question, and a score of 1 means that the knowledge fragments found can answer the question.

② Answer consistency. This dimension evaluates whether the system's answers are targeted at user questions, whether there are deviations from the question, or misunderstandings of the question. The dimension is also designed to be 0~1, with 0 being completely off-topic and 1 being completely relevant. Any intermediate result can be selected.

③ Answer hallucination ratio. This dimension needs to integrate the system answer and the found knowledge fragments to evaluate whether the system answer is hallucinated and how high the hallucination ratio is. This dimension is also designed to be 0~1, 0 means all model hallucinations, and 1 means no hallucinations at all.

④ Answer correctness. This dimension evaluates whether the system answers correctly and fully answers the user's question. It is one of the most core evaluation indicators of the system. This dimension can be scored anywhere between 0 and 1.

The above four dimensions are all centered around knowledge and the correctness of answers, which are highly relevant to the question. The following dimensions will focus on the anthropomorphism and grammatical correctness of the results generated by the large model, which are less relevant to the question:

⑤ Logic. This dimension evaluates whether the system answers logically coherently, and whether there are any conflicts or logical confusions. This dimension is evaluated on a 0-1 scale.

⑥ Fluency: This dimension evaluates whether the system answers smoothly and grammatically, and can be scored anywhere between 0 and 1.

⑦ Intelligence. This dimension evaluates whether the system answers are humanized and intelligent, and whether it can fully confuse users with human answers and intelligent answers. This dimension can be scored arbitrarily between 0 and 1.

For example, we evaluate the following responses:

python

```
print("问题: ")
question = "应该如何使用南瓜书? "
print(question)
print("模型回答: ")
result = qa_chain({"query": question})
print(result["result"])
```

markup

```
问题:
应该如何使用南瓜书?
模型回答:
应该将南瓜书作为西瓜书的补充，主要在遇到自己无法推导或理解的公式时进行查阅。对于初学机器
```

The following are the pieces of knowledge found by the system:

python

```
print(result["source_documents"])
```

markup

```
[Document(page_content='为主线，遇到自己推导不出来或者看不懂的公式时再来查阅南瓜书；
```

We make the following assessments:

① Knowledge search accuracy——1

② Response consistency — 0.8 (answered the question, but the topic of “feedback” was off topic)

③ Answer illusion ratio——1

④ Answer accuracy: 0.8 (same reason as above)

⑤ Logic - 0.7 (the subsequent content is not logically consistent with the previous content)

⑥ Fluency — 0.6 (the final summary is wordy and ineffective)

⑦ Intelligence——0.5 (has a distinct AI-style answer)

Combining the above seven dimensions, we can comprehensively and comprehensively evaluate the performance of the system in each case. By comprehensively considering the scores of all cases, we can evaluate the performance of the system in each dimension. If all dimensions are unified, we can also calculate the average score of all dimensions to evaluate the score of the system. We can also assign weights to different dimensions according to their importance, and then calculate the weighted average of all dimensions to represent the system score.

However, we can see that the more comprehensive and specific the evaluation is, the greater the difficulty and cost of the evaluation. Taking the above seven-dimensional evaluation as an example, we need to conduct seven evaluations for each case of each version of the system. If we have two versions of the system and there are 10 verification cases in the verification set, then each evaluation requires$10 \times 2 \times 7 = 140$times; but as our system continues to improve and iterate, the validation set will expand rapidly. Generally speaking, a mature system validation set should be at least several hundred in size, and there should be at least dozens of iterative improvement versions. Then the total number of evaluations will reach tens of thousands, which will bring high manpower and time costs. Therefore, we need a method to automatically evaluate the model's answers.

3.2 Simple Automatic Evaluation

One of the important reasons why large model evaluation is complicated is that the answers generated by the model are difficult to judge. In other words, it is easy to judge objective

questions, but difficult to judge subjective questions. Especially for some questions that do not have standard answers, it is particularly difficult to achieve automatic evaluation. However, at the expense of a certain degree of evaluation accuracy, we can transform complex subjective questions without standard answers into questions with standard answers, and then achieve it through simple automatic evaluation. Here we introduce two methods: constructing objective questions and calculating the similarity of standard answers.

Method 1: Constructing objective questions

The evaluation of subjective questions is very difficult, but for objective questions, we can directly compare whether the system answer is consistent with the standard answer, so as to achieve simple evaluation. We can construct some subjective questions into multiple or single choice objective questions to achieve simple evaluation. For example, for the question:

```markup
【问答题】南瓜书的作者是谁?
```

We can construct this subjective question into the following objective question:

```markup
【多项选择题】南瓜书的作者是谁?  A 周志明 B 谢文睿 C 秦州 D 贾彬彬
```

The model is required to answer this objective question. We give the standard answer as BCD. The model's answer can be compared with the standard answer to achieve evaluation and scoring. Based on the above ideas, we can construct a Prompt question template:

```python
prompt_template = '''
请你做如下选择题:
题目:南瓜书的作者是谁?
选项:A 周志明 B 谢文睿 C 秦州 D 贾彬彬
你可以参考的知识片段:
~~~
{}
~~~
请仅返回选择的选项
如果你无法做出选择,请返回空
'''
```

Of course, due to the instability of large models, even if we ask it to only give the selected options, the system may return a lot of text, which explains in detail why the following options are selected. Therefore, we need to extract the options from the model's answers. At the same time, we need to design a scoring strategy. In general, we can use the general scoring strategy for multiple-choice questions: 1 point for selecting all, 0.5 points for missing an option, and no points for not selecting the wrong option:

python

```
def multi_select_score_v1(true_answer : str, generate_answer : str) -> fl
    # true_anser : 正确答案，str 类型，例如 'BCD'
    # generate_answer : 模型生成答案，str 类型
    true_answers = list(true_answer)
    '''为便于计算，我们假设每道题都只有 A B C D 四个选项'''
    # 先找出错误答案集合
    false_answers = [item for item in ['A', 'B', 'C', 'D'] if item not in
    # 如果生成答案出现了错误答案
    for one_answer in false_answers:
        if one_answer in generate_answer:
            return 0
    # 再判断是否全选了正确答案
    if_correct = 0
    for one_answer in true_answers:
        if one_answer in generate_answer:
            if_correct += 1
            continue
    if if_correct == 0:
        # 不选
        return 0
    elif if_correct == len(true_answers):
        # 全选
        return 1
    else:
        # 漏选
        return 0.5
```

Based on the above scoring function, we can test four answers:

① BC

② Except A Zhou Zhihua, all others are authors of Pumpkin Books

③ You should choose BCD

④ I don't know

```python
answer1 = 'B C'
answer2 = '西瓜书的作者是 A 周志华'
answer3 = '应该选择 B C D'
answer4 = '我不知道'
true_answer = 'BCD'
print("答案一得分: ", multi_select_score_v1(true_answer, answer1))
print("答案二得分: ", multi_select_score_v1(true_answer, answer2))
print("答案三得分: ", multi_select_score_v1(true_answer, answer3))
print("答案四得分: ", multi_select_score_v1(true_answer, answer4))
```

```markup
答案一得分:  0.5
答案二得分:  0
答案三得分:  1
答案四得分:  0
```

But we can see that we require the model not to make a choice when it cannot answer, rather than randomly choosing. However, in our scoring strategy, both wrong choices and no choices are scored 0 points, which actually encourages the model to give hallucinatory answers. Therefore, we can adjust the scoring strategy according to the situation and deduct one point for wrong choices:

```python
def multi_select_score_v2(true_answer : str, generate_answer : str) -> fl
    # true_anser : 正确答案，str 类型，例如 'BCD'
    # generate_answer : 模型生成答案，str 类型
    true_answers = list(true_answer)
    '''为便于计算，我们假设每道题都只有 A B C D 四个选项'''
    # 先找出错误答案集合
    false_answers = [item for item in ['A', 'B', 'C', 'D'] if item not in
    # 如果生成答案出现了错误答案
    for one_answer in false_answers:
        if one_answer in generate_answer:
            return -1
    # 再判断是否全选了正确答案
    if_correct = 0
```

```
        for one_answer in true_answers:
            if one_answer in generate_answer:
                if_correct += 1
                continue
        if if_correct == 0:
            # 不选
            return 0
        elif if_correct == len(true_answers):
            # 全选
            return 1
        else:
            # 漏选
            return 0.5
```

As above, we use the second version of the scoring function to score the four answers again:

python

```
answer1 = 'B C'
answer2 = '西瓜书的作者是 A 周志华'
answer3 = '应该选择 B C D'
answer4 = '我不知道'
true_answer = 'BCD'
print("答案一得分: ", multi_select_score_v2(true_answer, answer1))
print("答案二得分: ", multi_select_score_v2(true_answer, answer2))
print("答案三得分: ", multi_select_score_v2(true_answer, answer3))
print("答案四得分: ", multi_select_score_v2(true_answer, answer4))
```

markup

```
答案一得分:  0.5
答案二得分:  -1
答案三得分:  1
答案四得分:  0
```

As you can see, we have achieved fast, automatic and discriminative automatic evaluation. In this way, we only need to construct each verification case, and then each verification and iteration can be fully automated, thus achieving efficient verification.

However, not all cases can be constructed as objective questions. For some cases that cannot be constructed as objective questions or where constructing them as objective questions will cause

the difficulty of the questions to drop sharply, we need to use the second method: calculating the similarity of answers.

Method 2: Calculate answer similarity

Evaluating the answers to generated questions is not a new problem in NLP. Whether it is machine translation or automatic summarization, it is necessary to evaluate the quality of generated answers. NLP generally uses the method of manually constructing standard answers to generated questions and calculating the similarity between the answer and the standard answer to achieve automatic evaluation.

For example, for the question:

```markup
南瓜书的目标是什么?
```

We can first artificially construct a standard answer:

```markup
周志华老师的《机器学习》（西瓜书）是机器学习领域的经典入门教材之一，周老师为了使尽可能多
```

Then we calculate the similarity between the model answer and the standard answer. The more similar they are, the more correct we think the answer is.

There are many ways to calculate similarity. We can generally use BLEU to calculate similarity. The principle is detailed in: **Zhihu | BLEU Detailed Explanation** . For students who do not want to delve into the principles of the algorithm, it can be simply understood as topic similarity.

We can call the bleu scoring function in the nltk library to calculate:

```python
from nltk.translate.bleu_score import sentence_bleu
import jieba

def bleu_score(true_answer : str, generate_answer : str) -> float:
    # true_anser : 标准答案，str 类型
    # generate_answer : 模型生成答案，str 类型
    true_answers = list(jieba.cut(true_answer))
    # print(true_answers)
    generate_answers = list(jieba.cut(generate_answer))
```

```
    # print(generate_answers)
    bleu_score = sentence_bleu(true_answers, generate_answers)
    return bleu_score
```

have a test:

python

```
true_answer = '周志华老师的《机器学习》（西瓜书）是机器学习领域的经典入门教材之一，周

print("答案一：")
answer1 = '周志华老师的《机器学习》（西瓜书）是机器学习领域的经典入门教材之一，周老师
print(answer1)
score = bleu_score(true_answer, answer1)
print("得分：", score)
print("答案二：")
answer2 = '本南瓜书只能算是我等数学渣渣在自学的时候记下来的笔记，希望能够帮助大家都成
print(answer2)
score = bleu_score(true_answer, answer2)
print("得分：", score)
```

markup

```
答案一：
周志华老师的《机器学习》（西瓜书）是机器学习领域的经典入门教材之一，周老师为了使尽可能多
得分： 1.2705543769116016e-231
答案二：
本南瓜书只能算是我等数学渣渣在自学的时候记下来的笔记，希望能够帮助大家都成为一名合格的“
得分： 1.1935398790363042e-231
```

It can be seen that the higher the consistency between the answer and the standard answer, the higher the evaluation score. With this method, we only need to construct a standard answer for each question in the validation set, and then we can achieve automatic and efficient evaluation.

However, this method also has several problems: ① Standard answers need to be constructed manually. For some vertical fields, constructing standard answers may be a difficult task; ② Evaluation by similarity may have problems. For example, if the generated answer is highly consistent with the standard answer but is exactly the opposite in several core places, resulting in a completely wrong answer, the bleu score will still be very high; ③ The flexibility of calculating the consistency with the standard answer is very poor. If the model generates a better

answer than the standard answer, the evaluation score will be reduced; ④ The intelligence and fluency of the answer cannot be evaluated. If the answer is spliced from the keywords in each standard answer, we believe that such an answer is unusable and incomprehensible, but the bleu score will be higher.

Therefore, depending on business situations, we sometimes also need some advanced evaluation methods that do not require constructing standard answers.

2.3 Evaluation with Large Models

Manual evaluation is highly accurate and comprehensive, but has high labor and time costs; automatic evaluation is low-cost and fast, but has problems of insufficient accuracy and incomplete evaluation. So, do we have a way to combine the advantages of both to achieve fast and comprehensive generation problem evaluation?

Large models such as GPT-4 provide us with a new method: using large models for evaluation. We can construct Prompt Engineering to let the large model act as an evaluator, thereby replacing the evaluator of manual evaluation; at the same time, the large model can give results similar to manual evaluation, so we can adopt the multi-dimensional quantitative evaluation method in manual evaluation to achieve fast and comprehensive evaluation.

For example, we can construct the following Prompt Engineering to let the big model do the scoring:

python

```
prompt = '''
你是一个模型回答评估员。
接下来，我将给你一个问题、对应的知识片段以及模型根据知识片段对问题的回答。
请你依次评估以下维度模型回答的表现，分别给出打分：

① 知识查找正确性。评估系统给定的知识片段是否能够对问题做出回答。如果知识片段不能做出回答

② 回答一致性。评估系统的回答是否针对用户问题展开，是否有偏题、错误理解题意的情况，打分分

③ 回答幻觉比例。该维度需要综合系统回答与查找到的知识片段，评估系统的回答是否出现幻觉，打

④ 回答正确性。该维度评估系统回答是否正确，是否充分解答了用户问题，打分分值在0~1之间，0为

⑤ 逻辑性。该维度评估系统回答是否逻辑连贯，是否出现前后冲突、逻辑混乱的情况。打分分值在0~

⑥ 通顺性。该维度评估系统回答是否通顺、合乎语法。打分分值在0~1之间，0为语句完全不通顺，1
```

⑦ 智能性。该维度评估系统回答是否拟人化、智能化，是否能充分让用户混淆人工回答与智能回答。

你应该是比较严苛的评估员，很少给出满分的高评估。
用户问题：
~~~
{}
~~~
待评估的回答：
~~~
{}
~~~
给定的知识片段：
~~~
{}
~~~
你应该返回给我一个可直接解析的 Python 字典，字典的键是如上维度，值是每一个维度对应的评
不要输出任何其他内容。
'''

We can actually test its effect:

python

```python
# 使用第二章讲过的 OpenAI 原生接口

from openai import OpenAI

client = OpenAI(
    # This is the default and can be omitted
    api_key=os.environ.get("OPENAI_API_KEY"),
)


def gen_gpt_messages(prompt):
    '''
    构造 GPT 模型请求参数 messages

    请求参数：
        prompt: 对应的用户提示词
    '''
    messages = [{"role": "user", "content": prompt}]
    return messages
```

```
def get_completion(prompt, model="gpt-3.5-turbo", temperature = 0):
    '''
    获取 GPT 模型调用结果

    请求参数:
        prompt: 对应的提示词
        model: 调用的模型，默认为 gpt-3.5-turbo，也可以按需选择 gpt-4 等其他模型
        temperature: 模型输出的温度系数，控制输出的随机程度，取值范围是 0~2。温度系
    '''
    response = client.chat.completions.create(
        model=model,
        messages=gen_gpt_messages(prompt),
        temperature=temperature,
    )
    if len(response.choices) > 0:
        return response.choices[0].message.content
    return "generate answer error"

question = "应该如何使用南瓜书？"
result = qa_chain({"query": question})
answer = result["result"]
knowledge = result["source_documents"]

response = get_completion(prompt.format(question, answer, knowledge))
response
```

markup

```
'{\n    "知识查找正确性": 1,\n    "回答一致性": 0.9,\n    "回答幻觉比例": 0.9,\
```

Note, however, that there are still problems with evaluating large models:

① Our goal is to iteratively improve Prompt to improve the performance of large models. Therefore, the large model we choose for evaluation needs to have better performance than the large model base we use. For example, the most powerful large model at present is still GPT-4. It is recommended to use GPT-4 for evaluation, which has the best effect.

② Large models have powerful capabilities, but they also have their limits. If the questions and answers are too complex, the knowledge fragments are too long, or too many evaluation dimensions are required, even GPT-4 will have incorrect evaluations, incorrect formats, and

inability to understand instructions. For these situations, we recommend considering the following solutions to improve the performance of large models:

1. Improve Prompt Engineering. In a similar way to the Prompt Engineering improvements of the system itself, iteratively optimize and evaluate Prompt Engineering, especially paying attention to whether the basic principles and core recommendations of Prompt Engineering are followed;

2. Split the evaluation dimensions. If there are too many evaluation dimensions, the model may have an incorrect format and the returned data cannot be parsed. You can consider splitting the multiple dimensions to be evaluated, calling the large model once for each dimension to evaluate, and finally get a unified result;

3. Merge evaluation dimensions. If the evaluation dimensions are too detailed, the model may not be able to understand them correctly and the evaluation may be incorrect. You can consider merging multiple dimensions to be evaluated. For example, merge logic, fluency, and intelligence into intelligence.

4. Provide detailed evaluation specifications. Without evaluation specifications, it is difficult for the model to give ideal evaluation results. Consider providing detailed and specific evaluation specifications to improve the model's evaluation capabilities;

5. Provide a small number of examples. The model may have difficulty understanding the evaluation specification, in which case you can give a small number of evaluation examples for the model to refer to in order to achieve the correct evaluation.

2.4 Hybrid Evaluation

In fact, the above evaluation methods are not isolated or contradictory. Compared with using a certain evaluation method independently, we recommend mixing multiple evaluation methods and selecting the appropriate evaluation method for each dimension, taking into account the comprehensiveness, accuracy and efficiency of the evaluation.

For example, for the personal knowledge base assistant in this project, we can design the following hybrid evaluation method:

1. Objective correctness. Objective correctness means that the model can give correct answers to questions with fixed correct answers. We can select some cases and use the method of constructing objective questions to evaluate the model and its objective correctness.

2. Subjective correctness. Subjective correctness means that the model can give correct and comprehensive answers to subjective questions that do not have fixed correct answers. We

can select some cases and use the large model evaluation method to evaluate whether the model's answers are correct.

3. Intelligence. Intelligence refers to whether the model's answer is sufficiently humanized. Since intelligence is weakly correlated with the question itself, but strongly correlated with the model and prompt, and the model's ability to judge intelligence is relatively weak, we can manually evaluate its intelligence by sampling a small amount.

4. Knowledge search correctness. Knowledge search correctness refers to whether the knowledge fragments retrieved from the knowledge base are correct and sufficient to answer the question for a specific question. It is recommended to use a large model to evaluate knowledge search correctness, that is, to require the model to determine whether a given knowledge fragment is sufficient to answer the question. At the same time, the evaluation results of this dimension can be combined with subjective correctness to calculate the hallucination situation, that is, if the subjective answer is correct but the knowledge search is incorrect, it means that a model hallucination has occurred.

Using the above evaluation method, we can make a reasonable evaluation of the project based on the obtained validation set examples. Due to time and manpower limitations, we will not show it in detail here.

Note: The source code corresponding to this article is in the Github open source project LLM Universe . Readers are welcome to download and run it. You are welcome to give us a Star~

Next Chapter >
2. Evaluate and optimize the generation part