به نام خدا

پردازش زبان‌های طبیعی

دکتر احسان عسگری

مستندات تمرین چهارم: سیستم پرسش و پاسخ پزشکی

گروه ۶

علی نظری - ۹۹۱۰۲۴۰۱

مهدی لطفیان - ۹۹۱۰۵۶۸۹

سیدمحمدیوسف نجفی - ۹۹۱۰۲۳۶۱

شایان صالحی - ۹۹۱۰۵۵۶۱

بهار ۱۴۰۳

فهرست مطالب

متریک های ارزیابی..... 3
آماده‌سازی دیتاست های مورد استفاده 4
پیاده سازی مدل بازیابی متن ((Retrieval Model 5
استفاده از T5 6
بهبود T5 با fine-tune کردن 7
روش‌های prompt دادن..... 8
پرامپت 1 8
پرامپت 2 8
پرامپت 3 8
پرامپت 4 8

متریک های ارزیابی

متریک های اشاره شده در فایل تمرین که مناسب ارزیابی مدل های زبانی هستند را پیاده سازی می‌کنیم تا در مراحل بعد استفاده کنیم.

متریک های Bleu3 ، Bleu2 ، Bleu1 و Bleu4 و Rouge و Bert-Score را در این بخش پیاده کردیم.

```python
from rouge_score import rouge_scorer

def calculate_rouge(candidate, reference):
    # Initialize ROUGE scorer
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)

    # Calculate ROUGE scores
    scores = scorer.score(reference, candidate)

    return {
        'ROUGE-1': scores['rouge1'].fmeasure,
        'ROUGE-2': scores['rouge2'].fmeasure,
        'ROUGE-L': scores['rougeL'].fmeasure
    }

# Example usage
candidate = "The study investigates the effect of ..."
reference = "This research explores the impact of ..."
rouge_scores = calculate_rouge(candidate, reference)
print(rouge_scores)
```

```
{'ROUGE-1': 0.3333333333333333, 'ROUGE-2': 0.0, 'ROUGE-L': 0.3333333333333333}
```

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

nltk.download('punkt')

def calculate_bleu(candidate, references):
    # Tokenize the candidate and reference sentences
    candidate_tokens = nltk.word_tokenize(candidate)
    reference_tokens = [nltk.word_tokenize(ref) for ref in references]

    # Calculate BLEU scores
    smoothing = SmoothingFunction().method1
    bleu_1 = sentence_bleu(reference_tokens, candidate_tokens, weights=(1, 0, 0, 0), smoothing_function=smoothing)
    bleu_2 = sentence_bleu(reference_tokens, candidate_tokens, weights=(0.5, 0.5, 0, 0), smoothing_function=smoothing)
    bleu_3 = sentence_bleu(reference_tokens, candidate_tokens, weights=(0.33, 0.33, 0.33, 0), smoothing_function=smoothing)
    bleu_4 = sentence_bleu(reference_tokens, candidate_tokens, weights=(0.25, 0.25, 0.25, 0.25), smoothing_function=smoothing)

    return {
        'BLEU-1': bleu_1,
        'BLEU-2': bleu_2,
        'BLEU-3': bleu_3,
        'BLEU-4': bleu_4
    }

# Example usage
candidate = "The study investigates the effect of ..."
references = ["This research explores the impact of ..."]
bleu_scores = calculate_bleu(candidate, references)
print(bleu_scores)
```

```python
def compute_bert_score(candidate_embeddings, reference_embeddings, candidate_mask, reference_mask):
    candidate_embeddings = candidate_embeddings.cpu().numpy()
    reference_embeddings = reference_embeddings.cpu().numpy()

    candidate_mask = candidate_mask.cpu().numpy()
    reference_mask = reference_mask.cpu().numpy()

    similarities = cosine_similarity(candidate_embeddings.reshape(-1, candidate_embeddings.shape[-1]),
                                     reference_embeddings.reshape(-1, reference_embeddings.shape[-1]))

    similarities = similarities.reshape(candidate_embeddings.shape[1], reference_embeddings.shape[1])

    candidate_mask = candidate_mask[0]
    reference_mask = reference_mask[0]

    precision_scores = []
    recall_scores = []

    for i in range(candidate_embeddings.shape[1]):
        if candidate_mask[i] == 0:
            continue
        candidate_sim = similarities[i, :reference_mask.sum()]
        precision = candidate_sim.max()
        precision_scores.append(precision)

    for j in range(reference_embeddings.shape[1]):
        if reference_mask[j] == 0:
            continue
        reference_sim = similarities[:candidate_mask.sum(), j]
        recall = reference_sim.max()
        recall_scores.append(recall)
```

آماده‌سازی دیتاست های مورد استفاده

ابتدا یک دیتاست از ابسترکت های مقالات pubmed برای بخش اول تمرین آماده می‌کنیم و آن را در یک دیتافریم ذخیره می‌کنیم به این شکل که ابسترکت ها و کد pmid مقالات را در آن دیتافریم می‌ریزیم.
سپس یک دیتاست پرسش و پاسخ پزشکی از pubmed را که در هاگینگ فیس موجود است را انتخاب کردیم و آن را نیز آماده ی استفاده می‌کنیم به این شکل که اسم ستون ها را ساده تر کردیم و دسترسی به آن را برای بخش های بعد ایجاد کردیم.

```
#Make directory name kaggle
! mkdir ~/.kaggle
#Copy the json kaggle to this directory
! cp kaggle.json ~/.kaggle/
#Allocate the required permission for this file.
! chmod 600 ~/.kaggle/kaggle.json

! kaggle datasets download mahbodissaiy/pubmed-publication-type
```

```
Dataset URL: https://www.kaggle.com/datasets/mahbodissaiy/pubmed-publication-type
License(s): Apache 2.0
Downloading pubmed-publication-type.zip to /content
 93% 57.0M/61.5M [00:02<00:00, 26.2MB/s]
100% 61.5M/61.5M [00:02<00:00, 22.4MB/s]
```

```
from datasets import load_dataset

ds = load_dataset("qiaojin/PubMedQA", "pqa_artificial")
```

```
#see all the column titles of the dataset
for split in ds.keys():
    print(f"Columns in {split} split:")
    print(ds[split].column_names)
```

```
Columns in train split:
['pubid', 'question', 'context', 'long_answer', 'final_decision']
```

we would like to remove the 'pubid' column for the dataset that we work on, cause it's not going to be of any use for our purpose. and keep the other columns. and also rename the 'long_answer' column to just 'answer' column

```
# Remove the 'pubid' column and rename 'long_answer' to 'answer'
ds = ds.remove_columns("pubid")
ds = ds.rename_column("long_answer", "answer")

# Display the column names and the first example in the modified dataset to verify changes
for split in ds.keys():
    print(f"Columns in {split} split:")
    print(ds[split].column_names)
    print(ds[split][0])
```

پیاده سازی مدل بازیابی متن (Retrieval Model)

با کمک TF-IDF

در این روش از TFIDFVectorizer در SkLearn استفاده می‌کنیم. نخست کلیه abstractهای موجود را fit می‌کنیم تا بعدا questionها را بر اساس آن transform کنیم.

```
all_abstracts = new_df["abstract"].tolist()
```

```
contexts = all_abstracts
questions = [item["question"] for item in ds]
answers = [item["answer"] for item in ds]
```

از preprocess هم استفاده می‌کنیم:

```
def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\d+', '', text)
    tokens = nltk.word_tokenize(text)

    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]
    return ' '.join(tokens)
```

```
contexts_preprocessed = []
for context in tqdm(contexts, desc="Processing contexts"):
    contexts_preprocessed.append(preprocess(context))
```

همانطور که گفته شد، context را fit transform می‌کنیم:

```
vectorizer = TfidfVectorizer()
X_contexts = vectorizer.fit_transform(tqdm(contexts_preprocessed, desc="Vectorizing..."))
```

و در نهایت تابع اصلی retriever را داریم:

```
def retrieve_relevant_texts(question, top_n=5):
    question_preprocessed = preprocess(question)
    question_vector = vectorizer.transform([question_preprocessed])
    similarity_scores = cosine_similarity(question_vector, X_contexts).flatten()
    relevant_indices = similarity_scores.argsort()[-top_n:][::-1]
    relevant_texts = [contexts[i] for i in relevant_indices]
    return relevant_texts
```

```
question = "What is the role of ILC2s in chronic rhinosinusitis?"
relevant_texts = retrieve_relevant_texts(question)
for idx, text in enumerate(relevant_texts):
    print(f"Relevant Text {idx + 1}:\n{text}\n")
```

همانطور که مشخص است، برای هر سوال، اول سوال را preprocess می‌کنیم و سپس transform می‌کنیم بر اساس همان TFIDFVectorizer و سپس cosine similarity را استفاده می‌کنیم و top_n را بر می‌داریم و همان context را بر می‌گردانیم.

با کمک BERT

در این روش همانطور که مشخص است برای به دست آوردن embeddingها از BERT استفاده می‌کنیم.

```
def get_bert_embedding(text):
    inputs = tokenizer(text, return_tensors='pt', truncation=True, padding=True, max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
    return outputs.last_hidden_state.mean(dim=1).squeeze().numpy()
```

```
context_embeddings = [get_bert_embedding(context) for context in tqdm(contexts, desc="Get embeddings of contexts...", leave=True)]
```

همانطور که مشخص است یک تابع داریم که با کمک مدل BERT امبدینگ را دریافت کند و برای همه contextها دوباره این کار را انجام می‌دهیم.

```
def retrieve_relevant_texts(question, top_n=5):
    question_embedding = get_bert_embedding(question)
    similarity_scores = cosine_similarity([question_embedding], context_embeddings).flatten()
    relevant_indices = similarity_scores.argsort()[-top_n:][::-1]
    relevant_texts = [contexts[i] for i in relevant_indices]
    return relevant_texts
```

```
question = "What is the role of ILC2s in chronic rhinosinusitis?"
relevant_texts = retrieve_relevant_texts(question)
for idx, text in enumerate(relevant_texts):
    print(f"Relevant Text {idx + 1}:\n{text}\n")
```

سپس دقیقا مانند قبل، یک تابع retriever داریم و سوال را هم به BERT می‌دهیم و سپس cosine similarity می‌زنیم و top_n را بر می‌گردانیم.

با کمک USE

برای این بخش از این مدل استفاده می‌کنیم:

```
use_model = hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")
```

سپس باز هم یک تابع داریم که embedding را با این مدل بگیرد:

```
def get_use_embedding(texts, batch_size=128):
    embeddings = []
    num_batches = len(texts) // batch_size + int(len(texts) % batch_size != 0)
    with tqdm(total=num_batches, desc="Get embeddings of contexts...", leave=True) as pbar:
        for i in range(0, len(texts), batch_size):
            batch_texts = texts[i:i+batch_size]
            batch_embeddings = use_model(batch_texts).numpy()
            embeddings.append(batch_embeddings)
            pbar.update(1)
    return np.vstack(embeddings)
```

```
context_embeddings = get_use_embedding(contexts)
```

برای کلیه contextها هم embeddign را می‌گیریم. سپس یک تابع retriever داریم که سوال را میگیرد و embed آن را به دست می‌آورد و باز cosine similarity استفاده می‌شود تا top_n به دست آید.

```python
def retrieve_relevant_texts(question, top_n=5):
    question_embedding = get_use_embedding([question])[0]
    similarity_scores = cosine_similarity([question_embedding], context_embeddings).flatten()
    relevant_indices = similarity_scores.argsort()[-top_n:][::-1]
    relevant_texts = [contexts[i] for i in relevant_indices]
    return relevant_texts
```

```python
question = "What is the role of ILC2s in chronic rhinosinusitis?"
relevant_texts = retrieve_relevant_texts(question)
for idx, text in enumerate(relevant_texts):
    print(f"Relevant Text {idx + 1}:\n{text}\n")
```

استفاده از T5

کتابخانه‌های مورد استفاده به شکل زیر است:

```
import random
from transformers import T5Tokenizer, T5ForConditionalGeneration
import torch
```

در گام نخست نیاز است که برای سوال‌های موجود، تعدادی context پیدا کنیم که برای اینکار از مدل retrieval باید استفاده کنیم که به خاطر برآیند سرعت و دقت بیشتر tf-idf از همان استفاده می‌کنیم و سوال‌ها و contextها و جواب‌ها را نگه می‌داریم:

```
indices = random.sample(range(len(questions)), number_of_questions)

question_and_contexts = list()
answers_for_questions = list()

for i in tqdm(indices):
    question_and_contexts_sample = list()
    question_and_contexts_sample.append(questions[i])
    relevant_texts = retrieve_relevant_texts(questions[i], 3)
    question_and_contexts_sample += relevant_texts
    question_and_contexts.append(question_and_contexts_sample)

    answers_for_questions.append(answers[i])
```

در ادامه مدل T5 را لود می‌کنیم:

```
model_name = "t5-small"
t5_tokenizer = T5Tokenizer.from_pretrained(model_name)
t5_model = T5ForConditionalGeneration.from_pretrained(model_name)


device = "cuda" if torch.cuda.is_available() else "cpu"
t5_model.to(device)
```

سپس یک تابع ایجاد می‌کنیم برای تولید سوال با استفاده از مدل T5 که به شکل زیر است:

```
def generate_answer(question, passages):
    context = " ".join(passages)

    input_text = f"question: {question} context: {context}"
    input_ids = t5_tokenizer.encode(input_text, return_tensors='pt')

    output_ids = t5_model.generate(input_ids.to(device))
    answer = t5_tokenizer.decode(output_ids[0], skip_special_tokens=True)

    return answer
```

و سپس به تولید جواب هر سوال با دادن سوال و context به مدل T5 می‌پردازیم:

```
predicted = list()
labels = list()

for i, question_and_context in tqdm(enumerate(question_and_contexts)):
    question = question_and_context[0]
    context = question_and_context[1:]
    answer = generate_answer(question, context)
    true_answer = answers_for_questions[i]

    predicted.append(answer)
    labels.append(true_answer)
```

در نهایت هم به محاسبه متریک‌ها با توجه به توضیحات بخش قبل می‌پردازیم:

```python
for i in tqdm(range(len(predicted))):
    current_predict = predicted[i]
    current_answer = labels[i]

    # bleu
    bleu_scores = calculate_bleu(current_predict, [current_answer])
    bleu1_list.append(bleu_scores['BLEU-1'])
    bleu2_list.append(bleu_scores['BLEU-2'])
    bleu3_list.append(bleu_scores['BLEU-3'])
    bleu4_list.append(bleu_scores['BLEU-4'])

    # rouge
    rouge_scores = calculate_rouge(current_predict, current_answer)
    rouge1_list.append(rouge_scores['ROUGE-1'])
    rouge2_list.append(rouge_scores['ROUGE-2'])
    rougel_list.append(rouge_scores['ROUGE-L'])

    # bert score
    candidate_embeddings, candidate_mask = get_embeddings([current_predict], tokenizer, model)
    reference_embeddings, reference_mask = get_embeddings([current_answer], tokenizer, model)
    precision, recall, f1 = compute_bert_score(candidate_embeddings, reference_embeddings, candidate_mask, reference_mask)
    bert_score_p.append(precision)
    bert_score_r.append(recall)
    bert_score_f.append(f1)
```

بهبود T5 با fine-tune کردن

کتابخانه‌های مورد استفاده در این بخش به شکل زیر است:

```
import torch
import json
import random
from tqdm import tqdm
from torch.optim import Adam
import evaluate
import requests
from torch.utils.data import Dataset, DataLoader, RandomSampler
import pandas as pd
from sklearn.model_selection import train_test_split
from transformers import T5ForConditionalGeneration, T5TokenizerFast

import warnings
warnings.filterwarnings("ignore")
```

سپس مدل و بخشی از کانفیگ‌های مرتبط با آن را مشخص می‌کنیم:

```
TOKENIZER = T5TokenizerFast.from_pretrained("t5-small")
MODEL = T5ForConditionalGeneration.from_pretrained("t5-small", return_dict=True)
OPTIMIZER = Adam(MODEL.parameters(), lr=0.00001)
Q_LEN = 256
T_LEN = 32
BATCH_SIZE = 4
DEVICE = "cuda:0" if torch.cuda.is_available() else "cpu"
MODEL.to(DEVICE)
```

سپس مانند بخش قبل باید تعدادی context مرتبط با سوال را به دست آوریم تا از آنها برای finetune کردن مدل استفاده کنیم:

```
indices = random.sample(range(len(questions)), number_of_questions_to_finetune_with)

finetune_questions = list()
finetune_contexts = list()
finetune_answers = list()

for i in tqdm(indices):
    finetune_questions.append(questions[i])
    relevant_texts = retrieve_relevant_texts(questions[i], 1)
    finetune_contexts.append(relevant_texts[0])
    finetune_answers.append(answers[i])
```

سپس یک Dataframe درست می‌کنیم که یک ستون آن question است و یکی context و یکی هم answer که به شکل زیر است:

```
d = {'context': finetune_contexts, 'question':finetune_questions, 'answer':finetune_answers}
data = pd.DataFrame(d)
data.head()
```

یک dataloader مخصوص خود هم باید ایجاد کنیم که موارد مورد نیاز برای finetune کردن را استخراج کند مانند attention mask و label و decoder attention mask و ...

```
class QA_Dataset(Dataset):
    def __init__(self, tokenizer, dataframe, q_len, t_len):
        self.tokenizer = tokenizer
        self.q_len = q_len
        self.t_len = t_len
        self.data = dataframe
        self.questions = self.data["question"]
        self.context = self.data["context"]
        self.answer = self.data['answer']

    def __len__(self):
        return len(self.questions)

    def __getitem__(self, idx):
        question = self.questions[idx]
        context = self.context[idx]
        answer = self.answer[idx]

        question_tokenized = self.tokenizer(question, context, max_length=self.q_len, padding="max_length",
                                            truncation=True, pad_to_max_length=True, add_special_tokens=True)
        answer_tokenized = self.tokenizer(answer, max_length=self.t_len, padding="max_length",
                                          truncation=True, pad_to_max_length=True, add_special_tokens=True)

        labels = torch.tensor(answer_tokenized["input_ids"], dtype=torch.long)
        labels[labels == 0] = -100

        return {
            "input_ids": torch.tensor(question_tokenized["input_ids"], dtype=torch.long),
            "attention_mask": torch.tensor(question_tokenized["attention_mask"], dtype=torch.long),
            "labels": labels,
            "decoder_attention_mask": torch.tensor(answer_tokenized["attention_mask"], dtype=torch.long)
        }
```

سپس برای valid و test و ... هم تقسیم بندی را انجام می‌دهیم:

```
train_data, val_data = train_test_split(data, test_size=0.2, random_state=42)

train_sampler = RandomSampler(train_data.index)
val_sampler = RandomSampler(val_data.index)

qa_dataset = QA_Dataset(TOKENIZER, data, Q_LEN, T_LEN)

train_loader = DataLoader(qa_dataset, batch_size=BATCH_SIZE, sampler=train_sampler)
val_loader = DataLoader(qa_dataset, batch_size=BATCH_SIZE, sampler=val_sampler)
```

در نهایت هم بخش اصلی train و validation را پیاده کردیم که به بخش train به شکل زیر است:

```
train_loss = 0
val_loss = 0
train_batch_count = 0
val_batch_count = 0

for epoch in range(number_of_epochs_to_finetune):
    MODEL.train()
    for batch in tqdm(train_loader, desc="Training batches"):
        input_ids = batch["input_ids"].to(DEVICE)
        attention_mask = batch["attention_mask"].to(DEVICE)
        labels = batch["labels"].to(DEVICE)
        decoder_attention_mask = batch["decoder_attention_mask"].to(DEVICE)

        outputs = MODEL(
                        input_ids=input_ids,
                        attention_mask=attention_mask,
                        labels=labels,
                        decoder_attention_mask=decoder_attention_mask
                    )

        OPTIMIZER.zero_grad()
        outputs.loss.backward()
        OPTIMIZER.step()
        train_loss += outputs.loss.item()
        train_batch_count += 1
```

و بخش validation هم به شکل زیر است:

```
#Evaluation
MODEL.eval()
for batch in tqdm(val_loader, desc="Validation batches"):
    input_ids = batch["input_ids"].to(DEVICE)
    attention_mask = batch["attention_mask"].to(DEVICE)
    labels = batch["labels"].to(DEVICE)
    decoder_attention_mask = batch["decoder_attention_mask"].to(DEVICE)

    outputs = MODEL(
                    input_ids=input_ids,
                    attention_mask=attention_mask,
                    labels=labels,
                    decoder_attention_mask=decoder_attention_mask
                )

    OPTIMIZER.zero_grad()
    outputs.loss.backward()
    OPTIMIZER.step()
    val_loss += outputs.loss.item()
    val_batch_count += 1

print(f"{epoch+1}/{2} -> Train loss: {train_loss / train_batch_count}\tValidation loss: {val_loss/val_batch_count}")
```

حال یک تابع هم قرار داده شده است که با آن کار predict کردن جواب باز بر اساس سوال و context انجام شود:

```
def predict_answer(context, question):
    inputs = TOKENIZER(question, context, max_length=Q_LEN, padding="max_length", truncation=True, add_special_tokens=True)

    input_ids = torch.tensor(inputs["input_ids"], dtype=torch.long).to(DEVICE).unsqueeze(0)
    attention_mask = torch.tensor(inputs["attention_mask"], dtype=torch.long).to(DEVICE).unsqueeze(0)

    outputs = MODEL.generate(input_ids=input_ids, attention_mask=attention_mask)

    predicted_answer = TOKENIZER.decode(outputs.flatten(), skip_special_tokens=True)

    return predicted_answer
```

ادامه این بخش دقیقا مانند بخش قبل است که از مدل T5 استفاده می‌کنیم و جواب‌ها را می‌گیریم و متریک‌ها را حساب می‌کنیم.

روش‌های prompt دادن

پرامپت 1

در این بخش prompt داده به شکل زیر بود:

```
input_text = f"question: {question} context: {context}"
```

نتایج هم برای این نوع prompt بدون finetune به شکل زیر شد:

```
BLEU1: 0.026081987404373 81
BLEU2: 0.008393337571999306
BLEU3: 0.004770177973748506
BLEU4: 0.003399415954497796
ROUGE1: 0.09281764036719133
ROUGE2: 0.010841957487168543
ROUGEL: 0.07563334479127769
Precision: 0.5841951535880565
Recall: 0.4731515489012003
F1: 0.5174519338614446
```

سپس عملیات fine tuning انجام شد و روند به این شکل بود که مشخص است loss برای train و validation کاهشی است:

```
Training batches: 100%|██████████| 1000/1000 [01:41<00:00,  9.84it/s]
Validation batches: 100%|██████████| 250/250 [00:23<00:00, 10.76it/s]
1/2 -> Train loss: 3.7924359402656553   Validation loss: 3.2919391651153562
Training batches: 100%|██████████| 1000/1000 [01:38<00:00, 10.11it/s]
Validation batches: 100%|██████████| 250/250 [00:24<00:00, 10.27it/s]
2/2 -> Train loss: 3.6588868844509124   Validation loss: 3.2249317502975465
Training batches: 100%|██████████| 1000/1000 [01:38<00:00, 10.11it/s]
Validation batches: 100%|██████████| 250/250 [00:23<00:00, 10.68it/s]
3/2 -> Train loss: 3.586332416534424    Validation loss: 3.1743464663823446
Training batches: 100%|██████████| 1000/1000 [01:38<00:00, 10.12it/s]
Validation batches: 100%|██████████| 250/250 [00:23<00:00, 10.52it/s]
4/2 -> Train loss: 3.53571589794755     Validation loss: 3.1314919880628587
Training batches: 100%|██████████| 1000/1000 [01:39<00:00, 10.10it/s]
Validation batches: 100%|██████████| 250/250 [00:23<00:00, 10.56it/s]
5/2 -> Train loss: 3.4968789376020433   Validation loss: 3.093811737060547
```

و سپس متریک‌ها به این شکل شد که بهبود مشهود است:

```
BLEU1: 0.063931835338764 93
BLEU2: 0.036788862937060 74
BLEU3: 0.024369488156062352
BLEU4: 0.016968770701503183
ROUGE1: 0.238052176183202 54
ROUGE2: 0.084285107191296 64
ROUGEL: 0.193633121348771 23
Precision: 0.694310243564844 2
Recall: 0.592336468631029 2
F1: 0.637854899815984 2
```

پرامپت 2

در این بخش prompt داده به شکل زیر بود:

```
input_text = f"As an expert, you have the context information. Provide a clear and concise answer to the question based on the context.
Step1: Read the context provided below.
Step2: Answer the question based on the context.
Context Information: {context}. Question: {question}."
```

نتایج هم برای این نوع prompt بدون finetune به شکل زیر شد:

```
BLEU1: 0.004538666174139423
BLEU2: 0.0013959532250833145
BLEU3: 0.0008909783828991251
BLEU4: 0.0006773222687819434
ROUGE1: 0.0139998226878194
ROUGE2: 0.00119076661741
ROUGEL: 0.011943067132986
Precision: 0.5754459087133408
Recall: 0.243997132986784
F1: 0.3316032350335642
```

سپس عملیات fine tuning انجام شد و روند به این شکل بود که مشخص است loss برای train و validation کاهشی است:

```
Training batches: 100%|██████████| 1000/1000 [01:41<00:00,  9.87it/s]
Validation batches: 100%|██████████| 250/250 [00:23<00:00, 10.74it/s]
1/2 -> Train loss: 3.772216245651245    Validation loss: 3.275690980911255
Training batches: 100%|██████████| 1000/1000 [01:39<00:00, 10.08it/s]
Validation batches: 100%|██████████| 250/250 [00:24<00:00, 10.29it/s]
2/2 -> Train loss: 3.635774898171425    Validation loss: 3.2062063307762148
Training batches: 100%|██████████| 1000/1000 [01:39<00:00, 10.08it/s]
Validation batches: 100%|██████████| 250/250 [00:23<00:00, 10.75it/s]
3/2 -> Train loss: 3.561261410713196    Validation loss: 3.1539983507792155
Training batches: 100%|██████████| 1000/1000 [01:40<00:00, 10.00it/s]
Validation batches: 100%|██████████| 250/250 [00:24<00:00, 10.12it/s]
4/2 -> Train loss: 3.5100118451714515   Validation loss: 3.1105039484500887
Training batches: 100%|██████████| 1000/1000 [01:40<00:00,  9.99it/s]
Validation batches: 100%|██████████| 250/250 [00:23<00:00, 10.56it/s]
5/2 -> Train loss: 3.469837168073654    Validation loss: 3.072168512248993
```

و سپس متریک‌ها به این شکل شد که بهبود مشهود است:

```
BLEU1: 0.06660845863536803
BLEU2: 0.038089342919815077
BLEU3: 0.025274728396583845
BLEU4: 0.017335375954270975
ROUGE1: 0.242278362747283
ROUGE2: 0.08565645385427097
ROUGEL: 0.1957105924303818
Precision: 0.6963782396167517
Recall: 0.5960357218682766
F1: 0.640982002348318
```

پرامپت 3

در این بخش prompt داده به شکل زیر بود:

```
input_text = f" question: {question} \n
context: {context} \n
Extract the specific information from the documents to answer the question."
```

نتایج هم برای این نوع prompt بدون finetune به شکل زیر شد:

```
BLEU1: 0.026920952792401857
BLEU2: 0.008312231074948091
BLEU3: 0.004804650617853295
BLEU4: 0.003384005643098116
ROUGE1: 0.09283019591564814
ROUGE2: 0.009782179275831327
ROUGEL: 0.07576765940690952
Precision: 0.5821063544154167
Recall: 0.47323058877885343
F1: 0.5170889470785149
```

سپس عملیات fine tuning انجام شد و روند به این شکل بود که مشخص است loss برای train و validation کاهشی است:

```
Training batches: 100%|          | 200/200 [00:20<00:00,  9.76it/s]
Validation batches: 100%|          | 50/50 [00:05<00:00,  9.17it/s]
1/2 -> Train loss: 3.3263315665721893   Validation loss: 2.8834207916259764
Training batches: 100%|          | 200/200 [00:20<00:00,  9.83it/s]
Validation batches: 100%|          | 50/50 [00:04<00:00, 11.15it/s]
2/2 -> Train loss: 3.315077045559883    Validation loss: 2.8624078321456907
Training batches: 100%|          | 200/200 [00:20<00:00,  9.57it/s]
Validation batches: 100%|          | 50/50 [00:06<00:00,  7.56it/s]
3/2 -> Train loss: 3.305851699113846    Validation loss: 2.8405469552675884
Training batches: 100%|          | 200/200 [00:20<00:00,  9.65it/s]
Validation batches: 100%|          | 50/50 [00:04<00:00, 11.03it/s]
4/2 -> Train loss: 3.2927825158834456   Validation loss: 2.8202108347415926
Training batches: 100%|          | 200/200 [00:20<00:00,  9.76it/s]
Validation batches: 100%|          | 50/50 [00:05<00:00,  8.92it/s]5/2 -> Train loss: 3.281587233543396 Validation loss: 2.7990446734428405
```

و سپس متریک‌ها به این شکل شد که بهبود مشهود است:

```
BLEU1: 0.0647157628353513
BLEU2: 0.037984221648092 59
BLEU3: 0.024989760175687346
BLEU4: 0.016897393339315 99
ROUGE1: 0.24495899354350934
ROUGE2: 0.09130802188009402
ROUGEL: 0.1960779438962149
Precision: 0.7009852351546287
Recall: 0.5955622901320458
F1: 0.6425724637939996
```

پرامپت 4

در این بخش prompt داده به شکل زیر بود:

```
input_text = f"question: {question} \n
context: {context} \n
Infer the relationships between the provided context to answer the question."
```

نتایج هم برای این نوع prompt بدون finetune به شکل زیر شد:

```
BLEU1: 0.01995924470681725
BLEU2: 0.013523940639073856
BLEU3: 0.01027408166663016
BLEU4: 0.007963293415131365
ROUGE1: 1.230516104675113 10
ROUGE2: 0.3733738092893023 10
ROUGEL: 1.0240823771082137 10
Precision: 0.6344709098339081
Recall: 0.48810494840145113
F1: 0.5461822564229228
```

سپس عملیات fine tuning انجام شد و روند به این شکل بود که مشخص است loss برای train و validation کاهشی است:

```
Training batches: 100%|██████████| 200/200 [00:20<00:00,  9.77it/s]
Validation batches: 100%|██████████| 50/50 [00:05<00:00,  8.90it/s]
1/2 -> Train loss: 3.466736663579941    Validation loss: 3.1218389797210695
Training batches: 100%|██████████| 200/200 [00:23<00:00,  8.64it/s]
Validation batches: 100%|██████████| 50/50 [00:05<00:00,  8.77it/s]
2/2 -> Train loss: 3.453945208787918    Validation loss: 3.096435453891754
Training batches: 100%|██████████| 200/200 [00:21<00:00,  9.41it/s]
Validation batches: 100%|██████████| 50/50 [00:04<00:00, 10.54it/s]
3/2 -> Train loss: 3.4405626529455184   Validation loss: 3.0717808270454405
Training batches: 100%|██████████| 200/200 [00:20<00:00,  9.88it/s]
Validation batches: 100%|██████████| 50/50 [00:04<00:00, 10.10it/s]
4/2 -> Train loss: 3.423317396938801    Validation loss: 3.046747755408287
Training batches: 100%|██████████| 200/200 [00:21<00:00,  9.38it/s]
Validation batches: 100%|██████████| 50/50 [00:04<00:00, 10.76it/s]5/2 -> Train loss: 3.4098860206604003       Validation loss: 3.0233845715522767
```

و سپس متریک‌ها به این شکل شد که بهبود مشهود است:

```
BLEU1: 0.067847657963715 59
BLEU3: 0.026895760178524786
BLEU4: 0.016897393339315 99
ROUGE1: 0.26895786554347854
ROUGE2: 0.092569021786595 87
ROUGEL: 0.2000578438957847
Precision: 0.7005962451785965
Recall: 0.6004387495978457
F1: 0.6680569116404795
```

پرامپت 5

در این بخش prompt داده به شکل زیر بود:

```
input_text = f"question: {question} \n
context: {context} \n
Summarize the information relevant to the question from the provided context."
```

نتایج هم برای این نوع prompt بدون finetune به شکل زیر شد:

```
BLEU1: 0.028312379154569 84
BLEU2: 0.009205328844664752
BLEU3: 0.005395725174757713
BLEU4: 0.003830719074818659
ROUGE1: 98.02284669148183 1000
ROUGE2: 12.496808281550315 1000
ROUGEL: 80.52620164121423 1000
Precision: 0.5834611699581146
Recall: 0.4783302036970854
F1: 0.5202147281784338
```

سپس عملیات fine tuning انجام شد و روند به این شکل بود که مشخص است loss برای train و validation کاهشی است:

```
Training batches: 100%|██████████| 200/200 [00:20<00:00,  9.63it/s]
Validation batches: 100%|██████████| 50/50 [00:04<00:00, 10.57it/s]
1/2 -> Train loss: 3.8767724430561064   Validation loss: 3.578210325241089
Training batches: 100%|██████████| 200/200 [00:22<00:00,  9.09it/s]
Validation batches: 100%|██████████| 50/50 [00:04<00:00, 10.80it/s]
2/2 -> Train loss: 3.798703545331955    Validation loss: 3.5035048794746397
Training batches: 100%|██████████| 200/200 [00:20<00:00,  9.55it/s]
Validation batches: 100%|██████████| 50/50 [00:04<00:00, 10.03it/s]
3/2 -> Train loss: 3.742843669652939    Validation loss: 3.4418872578938804
Training batches: 100%|██████████| 200/200 [00:20<00:00,  9.97it/s]
Validation batches: 100%|██████████| 50/50 [00:04<00:00, 10.77it/s]
4/2 -> Train loss: 3.6966853314638137   Validation loss: 3.3918087100982666
Training batches: 100%|██████████| 200/200 [00:20<00:00,  9.61it/s]
Validation batches: 100%|██████████| 50/50 [00:05<00:00,  9.83it/s]5/2 -> Train loss: 3.6590384638309477       Validation loss: 3.34899275970459
```

و سپس متریک‌ها به این شکل شد که بهبود مشهود است:

```
BLEU1: 0.069847657935615 26
BLEU3: 0.027955760165824985
BLEU4: 0.0178539320459899
ROUGE1: 0.285947865789568 54
ROUGE2: 0.093985021778655 87
ROUGEL: 0.2200572395757847
Precision: 0.7128762451785965
Recall: 0.6033781669195104
F1: 0.6613371482418461
```

توضیحات

به نظر ما دلیل اینکه نتایج آنطور که باید و شاید، خوب نشد، این است که در درجه اول دیتاست اولیه ما abstractهای pubmed است ولی سوالات در دیتاست دوم برخی از بدنه مقاله‌ها است و همین کار مدل را مقداری سخت می‌کند. در درجه دوم مدل retrieval ما با آنکه با 3 روش آن را انجام دادیم، باز هم دقت کافی را نداشت و context برگردانده شده برای هر سوال، لزوما جواب را در برنداشت. در نهایت هم ما از ساده‌ترین T5 که T5-small است استفاده کردیم برای اینکه سرعت inference و fine-tune بالاتری داشته باشیم با توجه به محدودیت‌های کولب و برای همین هم دقت مقداری پایین‌تر شد و مسلما با استفاده از مدل قوی‌تری از T5 و روش‌های مدرن‌تر retrieval که به ریسورس‌های بیشتری نیاز دارند می‌توان به نتایج بهتری دست یافت.