

به نام خدا



مبانی پردازش زبان و گفتار
پیاده سازی تولید متن با استفاده از بازیابی (RAG)

بهاره کاووسی نژاد ۹۹۴۳۱۲۱۷
سیده شکیبا انارکی فیروز ۹۹۴۴۲۰۴۷

تابستان ۱۴۰۳

عنوان پروژه: پیاده سازی تولید متن با استفاده از بازیابی (RAG)

مقدمه:

هدف اصلی این پروژه بهبود کیفیت و دقت اطلاعات از طریق فناوری پیشرفته Retrieval-Augmented Generation (RAG) می باشد. این فناوری، که به عنوان یکی از نوآورانه ترین روش ها در تولید محتوا شناخته می شود، امکان دسترسی به اطلاعات دقیق و به روز را فراهم کرده و در نتیجه، ارائه پاسخ های با کیفیت تر را ممکن می سازد. در این پروژه، با استفاده از مدل های پیشرفته ای مانند مدل زبانی LaBSE و مدل تولیدی LLaMA، تلاش شده تا الگویی جامع و کارآمد برای افزایش دقت و کیفیت اطلاعات ایجاد شود. این مدل ها با بهره گیری از قدرت پردازشی بالای GPU قابلیت پردازش حجم زیادی از داده ها و ارائه پاسخ های دقیق را فراهم می کنند. با استفاده از فناوری RAG، می توان خدمات اطلاعاتی در حوزه های مختلف را با دقت و کیفیت بیشتری ارائه داد. مثل استفاده در ربات هایی که به سوالات متداول در سایت ها پاسخ می دهند.

پیش زمینه ی تکنولوژیکی:

روند پیاده سازی:

1. **نصب وابستگی ها:** در ابتدا، تمامی کتابخانه ها و وابستگی های لازم برای اجرای پروژه را نصب کردیم تا بتوانیم با ابزارها و مدل های مورد نیاز کار کنیم.
2. **بارگذاری و استخراج دیتاست:** سپس، دیتاست مورد نظر را بارگذاری کرده و فایل های فشرده را استخراج کردیم تا داده ها قابل استفاده شوند.
3. **آپلود داده ها در Google Drive:** برای جلوگیری از نیاز به آپلود داده ها در هر جلسه اجرایی، آن ها را در Google Drive آپلود کرده و لینک داده ها را در نوتبوک وارد کردیم.

Download & unzip dataset from Google Drive

```
[ ] !pip install gdown --q
```

```
[ ] import gdown
```

```
file_id = '10ubJZ0UgGM9wAHHuyfrhgRAuauLpZjAK'  
destination = '/content/Data.zip' # Path where the file will be saved  
gdown.download(f'https://drive.google.com/uc?id={file_id}', destination, quiet=False)
```

```
➡ Downloading...  
From: https://drive.google.com/uc?id=10ubJZ0UgGM9wAHHuyfrhgRAuauLpZjAK  
To: /content/Data.zip  
100%|██████████| 373k/373k [00:00<00:00, 82.9MB/s]  
'/content/Data.zip'
```

```
[ ] import zipfile
```

```
with zipfile.ZipFile(destination, 'r') as zip_ref:  
    zip_ref.extractall('/content/Data')
```

4. خواندن داده‌ها: در مرحله بعد، داده‌ها را خواندیم. برخی از چالش‌ها مربوط به فرمت‌های مختلف داده‌ها مانند فایل‌های doc و docx بود که برخی سوالات و پاسخ‌ها داخل جداول این فایل‌ها قرار داشتند و تنها برخی از سطرها داده باید استخراج می‌شدند. با استفاده از تابع `extract_txt_list` هر خط یک فایل تکست را به یک لیست اضافه می‌کنیم

```
def extract_txt_list(path):
    l = []
    with open(path, 'r') as file:
        l = file.read().split('\n')
        # print(lines)
        # l = lines
    return l

[9] import os

# Specify the directory containing the .txt files
directory = '/content/'

# Get a list of all .txt files in the directory
test_questions_1 = extract_txt_list('/content/test_questions_1.txt')
test_answers_1 = extract_txt_list('/content/test_answers_1.txt')
# print(test_questions_1)

test_questions_2 = extract_txt_list('/content/test_questions_2.txt')
test_answers_2 = extract_txt_list('/content/test_answers_2.txt')

train_questions_1 = extract_txt_list('/content/train_questions_1.txt')
train_answers_1 = extract_txt_list('/content/train_answers_1.txt')

train_questions_2 = extract_txt_list('/content/train_questions_2.txt')
train_answers_2 = extract_txt_list('/content/train_answers_2.txt')

[10] all_questions = test_questions_1 + test_questions_2 + train_questions_1 + train_questions_2
all_answers = test_answers_1 + test_answers_2 + train_answers_1 + train_answers_2
```

این کار را برای تمامی فایل‌های تکست انجام داده و در انتها آنها را `concat` می‌کنیم تا تمام سوالات و جواب‌ها در کنار هم باشند

```
# Assuming your data is stored in the variable questions_2_content

# Initialize empty lists to store the extracted data
questions_2 = []
answers_2 = []

# Iterate over the content and extract the desired columns
for row in questions_2_content[1:]: # Skip the first row as it contains column names
    question = row[3] # Extract the question from the 4th column ('عنوان پرسش')
    answer = row[4] # Extract the answer from the 5th column ('پاسخ تفصیلی و تحلیلی')

    questions_2.append(question)
    answers_2.append(answer)

# Print the extracted data
for q, a in zip(questions_2, answers_2):
    print(f"Question: {q}")
    print(f"Answer: {a}")
    print("\n")
```

Question: عنوان پرسش
Answer: پاسخ تفصیلی و تحلیلی

Question: انواع پروانه های ساختمانی را نام ببرید؟
Answer: 1- پروانه شهرسازی
2- پروانه شروع عملیات ساختمانی-
3- تخریب و نوسازی
4- اضافه اشکوب-
5- توسعه بنا-
6- تغییرات
7- تبدیل
8- تغییر نقشه پروانه

5. تقسیم داده‌ها به مجموعه‌های آموزشی و آزمایشی: داده‌ها را به نسبت 20-80 به دو مجموعه آموزشی و آزمایشی تقسیم کردیم تا بتوانیم مدل‌ها را به درستی ارزیابی کنیم.

6. پاکسازی و نرمال‌سازی داده‌ها: برای نرمال‌سازی مجموعه داده‌ها، از چندین ابزار مختلف برای زبان فارسی استفاده کردیم که در نهایت، ابزار Dadmatools بهترین عملکرد را داشت.

7. ایجاد بردارهای تعبیه‌سازی (Embeddings) برای داده‌ها: برای آمادسازی داده‌ها جهت استفاده در مدل‌های یادگیری عمیق، بردارهای تعبیه‌سازی را برای داده‌ها ایجاد کردیم.

8. بارگذاری یک مدل از Hugging Face: سپس، یکی از مدل‌های موجود در Hugging Face را بارگذاری کردیم تا از آن در پروژه استفاده کنیم.

```
✓ [24] from transformers import AutoModelForCausalLM, AutoTokenizer, pipeline
```

```
✓ [25] model = AutoModelForCausalLM.from_pretrained(  
39s     "microsoft/Phi-3-mini-4k-instruct",  
     device_map="cuda",  
     torch_dtype="auto",  
     trust_remote_code=True,  
 )  
     tokenizer = AutoTokenizer.from_pretrained("microsoft/Phi-3-mini-4k-instruct")
```

9. استفاده از مدل به عنوان بازیابی‌کننده برای پاسخ به سوالات: از مدل بارگذاری شده به عنوان یک بازیابی‌کننده برای پاسخ به سوالات استفاده کردیم و مدل را به نحوی تنظیم کردیم که بتواند به سوالات به بهترین شکل پاسخ دهد.

با استفاده از تابع `get_models_response` های مدل به هر سوال را ذخیره می کنیم.

```
✓ [11] def get_models_response(model, tokenizer, prompt, generated_texts):  
0s     prompt = f"Answer {prompt}"  
     # Tokenize the input prompt  
     inputs = tokenizer(prompt, return_tensors="pt").to("cuda")  
  
     # Generate the output  
     output = model.generate(**inputs, max_new_tokens=50) # Generate up to 50 new  
  
     # Decode the output tokens to get the generated text  
     generated_text = tokenizer.decode(output[0], skip_special_tokens=True)  
  
     # Append the generated text to the list  
     generated_texts.append(generated_text)
```

```
✓ [27] # Initialize the list to store generated texts  
7m     generated_texts = []  
  
     # Iterate over all questions and get the model's response  
     for question in all_questions:  
         get_models_response(model, tokenizer, question, generated_texts)
```

⚠ The `seen_tokens` attribute is deprecated and will be removed in v4.41. Use the `WARNING:transformers_modules.microsoft.Phi-3-mini-4k-instruct.c1358f8a35e6d2af818`

```
✓ [13] # Print the generated texts  
0s     for i, text in enumerate(generated_texts):  
         print(f"Response to Question {i+1}: {text}\n")
```

10. ارزیابی نتایج: نتایج بدست‌آمده از مدل را ارزیابی کردیم تا دقت و کیفیت پاسخ‌ها را بررسی کنیم. برای محاسبه دقت از یک مدل sentence transformer به نام sentence-transformers/all-MiniLM-L6-v2 استفاده کردیم و جملات پاسخ مدل و پاسخ‌های اصلی رو بهش دادیم تا قابل مقایسه با هم باشند سپس شباهت کسینوسی دو پاسخ را بدست آورده و از شباهت‌های کسینوسی تمامی جفت‌های پاسخ، میانگین گرفتیم

```
[3] from sentence_transformers import SentenceTransformer
import torch

# Load model
model_SentenceTransformer = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
```

```
[30] import torch
from torch.nn.functional import cosine_similarity
from sentence_transformers import SentenceTransformer
```

```
[31] # Initialize lists to store cosine similarities
cosine_similarities_list = []

for all_answer, generated_text in zip(all_answers, generated_texts):
    embeddings_all_answer = model_SentenceTransformer.encode(all_answer, convert_to_tensor=True)
    embeddings_generated_text = model_SentenceTransformer.encode(generated_text, convert_to_tensor=True)

    # Ensure embeddings have shape (1, embedding_dim) if they are single sentences
    embeddings_all_answer = embeddings_all_answer.unsqueeze(0)
    embeddings_generated_text = embeddings_generated_text.unsqueeze(0)

    cosine_sim = cosine_similarity(embeddings_all_answer, embeddings_generated_text)
    cosine_similarities_list.append(cosine_sim.item()) # Append cosine similarity to the list
```

```
[32] # Calculate average cosine similarity
avg_cosine_similarity = sum(cosine_similarities_list) / len(cosine_similarities_list)

# Print the average cosine similarity
print(f'Average Cosine Similarity: {avg_cosine_similarity:.4f}')
```

→ Average Cosine Similarity: 0.6197

11. بازنویسی متون بازیابی‌شده (Paraphrasing): همچنین، بازنویسی متون بازیابی‌شده می‌تواند به عنوان یک مرحله دیگر در بهبود کیفیت پاسخ‌ها مورد استفاده قرار گیرد.

▼ paraphrase the answer by the generator model.

```
[ ] # Store for model3 outputs
model3_outputs = []

# Process each item in the dataset for model3
for result in results:
    query, retrieved_text, score = result

    prompt = f"""You are a bot that makes recommendations for travellers. Try to be a helpful recommender system.
    This is the recommended answer: {retrieved_text}
    The user question is: {query}
    Compile a recommendation to the user based on the recommended answer and the user question."""

    messages = [{"role": "user", "content": prompt}]
    generated_text = pipe3(messages, **generation_args3)

    # Append the generated text to model3_outputs
    model3_outputs.append(generated_text)

# Store the model3_outputs to a file if needed
with open("model3_outputs.txt", "w") as f:
    for item in model3_outputs:
        f.write("%s\n" % item)

# Clear the cache for model3
model3_dir = f"../root/.cache/huggingface/hub/models--openchat-openchat-3.6-8b-20240522"
shutil.rmtree(model3_dir, ignore_errors=True)

print("Model 3 cache cleared successfully.")
```

```
[ ] # from transformers import AutoModelForCausalLM, AutoTokenizer, pipeline
import shutil

# Load the model and tokenizer for model3
model3 = AutoModelForCausalLM.from_pretrained(
    "openchat/openchat-3.6-8b-20240522",
    quantization_config = q_config,
    device_map="cuda",
    torch_dtype="auto",
    trust_remote_code=True,
)

tokenizer3 = AutoTokenizer.from_pretrained("openchat/openchat-3.6-8b-20240522")

# Create a text generation pipeline for model3
pipe3 = pipeline(
    "text-generation",
    model=model3,
    tokenizer=tokenizer3,
)

# Define generation arguments for model3
generation_args3 = {
    "max_new_tokens": 200,
    "return_full_text": False,
    "temperature": 0.0,
    "do_sample": False,
}
```

نتایج:

بنا به نتایج به دست آمده از پروژه و تجربیات جمع‌آوری شده، استفاده از فناوری Retrieval-Augmented Generation (RAG) به طور مشخصی بهبود در کیفیت و دقت پاسخ‌ها ایجاد می‌کند. این فناوری امکان دسترسی به اطلاعات دقیق و به‌روز را فراهم می‌کند و به واسطه استفاده از مدل‌های پیشرفته زبانی مانند LLaMA و LaBSE، قادر است پاسخ‌هایی با کیفیت و درست‌تر ارائه دهد. علاوه بر این، بازنویسی متون بازیابی‌شده نیز می‌تواند به عنوان یک مرحله اضافی برای بهبود کیفیت پاسخ‌ها مورد استفاده قرار گیرد.

به طور کلی، استفاده از RAG در پروژه‌هایی که نیازمند ارائه پاسخ‌های دقیق و با کیفیت است، به عنوان یک روش نوآورانه و مؤثر شناخته می‌شود که می‌تواند به بهبود عملکرد سیستم‌های مشابه و افزایش دقت در ارائه خدمات اطلاعاتی کمک کند.

موانع و چالش‌ها:

- در طول اجرای این پروژه، با چندین چالش مواجه شدیم. یکی از آنها، مواجهه با تضادها و ناسازگاری‌ها در هنگام import کردن کتابخانه‌ها و مدل‌ها بود. به ویژه یکی از مشکلات مهم، مربوط به نسخه کتابخانه Torch بود که چندین بار باعث ایجاد مشکل در اجرای کدها شد و در نهایت به نسخه ۲.۱.۲ رسیدیم.
- چالش دیگری که با آن روبرو شدیم، یافتن یک نرم‌الایزر مناسب و کارآمد برای متن‌های فارسی بود. ما بسیاری از ابزارها و کتابخانه‌ها را امتحان کردیم تا به یک مدل با دقت مناسب برسیم. در نهایت، دو ابزار Dadmatools و Hazm بهترین عملکرد را ارائه دادند و توانستیم با استفاده از آن‌ها متن‌های فارسی را به خوبی نرمال‌سازی کنیم.
- محدودیت استفاده از GPU در نسخه‌ی رایگان Google Colab باعث شد که زمان اجرای مدل‌ها به طور چشمگیری افزایش یابد و کار روی پروژه را به تعویق بیندازد.
- یکی دیگر از موانع، بارگذاری داده‌ها به صورت دستی بود. ما قبلاً از دیتاست‌های موجود در Hugging Face استفاده کرده بودیم، اما بارگذاری داده‌های سفارشی نیازمند تنظیمات خاص و پیچیده‌تری بود.
- ساختار داده‌ها نیز چالشی برای خواندن و پردازش آن‌ها بود. برخی از داده‌ها به صورت لیست بودند و نیاز به استخراج string داشتند که این امر باعث افزایش پیچیدگی پردازش داده‌ها شد.

آموخته‌ها و پیشنهادات:

در طول اجرای این پروژه، ما با اجزای مختلف سیستم RAG آشنا شدیم و نحوه پیاده‌سازی هر یک از آن‌ها را یاد گرفتیم. همچنین، برخی نکات و ترفندهای کاربردی در استفاده از Google Colab را نیز فراگرفتیم که به ما کمک کرد تا فرآیندهای پردازش داده و آموزش مدل‌ها را بهینه‌سازی کنیم. یکی از مهمترین دستاوردهای این پروژه برای ما، توانایی پیاده‌سازی مدل‌های پیشرفته زبانی و تولیدی است که امکان دسترسی به اطلاعات دقیق‌تر و به‌روزتر را فراهم می‌کند و با بهره‌گیری از این مدل‌ها، می‌توان به دقت و کیفیت بالاتری در ارائه پاسخ‌ها دست یافت.

در آینده‌ای نه چندان دور، ممکن است مدل‌های دقیق‌تر و بهتری به خصوص در حوزه زبان فارسی ارائه شوند که بتوانند بهبودهای چشمگیری در عملکرد سیستم‌های مشابه ایجاد کنند و سرویس‌های مرتبط با اطلاعات را با کیفیت و دقت بیشتری ارائه دهند. به طور کلی، این پروژه نه تنها به ما کمک کرد تا با تکنولوژی‌های پیشرفته در زمینه هوش مصنوعی و پردازش زبان طبیعی آشنا شویم، بلکه توانستیم مهارت‌های خود را در استفاده از ابزارهای پردازشی مانند Google Colab و kaggle بهبود بخشیم. این تجربیات ارزشمند می‌توانند در پروژه‌های آینده ما نیز بسیار مفید واقع شوند و مقدمه‌ای برای توسعه سیستم‌های پیشرفته‌تری باشند.

https://huggingface.co/learn/cookbook/en/rag_evaluation
<https://medium.com/aimonks/text-similarity-building-a-text-similarity-checker-with-hugging-face-and-streamlit-db0eaf66048c>
<https://huggingface.co/sentence-transformers/LaBSE>
<https://github.com/roshan-research/hazm>
<https://github.com/Dadmatech/DadmaTools>
https://python.langchain.com/v0.2/docs/integrations/llms/huggingface_pipelines/
<https://github.com/bojone/labse>
<https://vectorize.io/picking-the-best-embedding-model-for-rag/>
<https://github.com/flin3500/Cuda-Google-Colab>
<https://huggingface.co/HuggingFaceH4/zephyr-7b-beta>
<https://huggingface.co/microsoft/Phi-3-mini-4k-instruct>
<https://learn.deeplearning.ai/courses/quantization-fundamentals/lesson/4/loading-models-by-data-type>
<https://www.deeplearning.ai/short-courses/quantization-fundamentals-with-hugging-face/>
https://colab.research.google.com/github/patrickvonplaten/notebooks/blob/master/Getting_the_most_out_of_LLMs.ipynb#scrollTo=yhwHj948GdQy

Some chats GPT-3.5-turbo:

<https://shareg.pt/e06b5q4>
<https://shareg.pt/JTLb1c5>
<https://shareg.pt/Q5aRxac>
<https://sharegpt.com/c/tkK6kWK>
<https://shareg.pt/RUNWZPq>
<https://shareg.pt/TvC2XSK>