بسمه تعالی

تمرین ۴ پردازش زبان طبیعی

سیستم پرسش و پاسخ پزشکی

گروه شماره ۱۵

اعضای گروه:

- *سید احسان حسن بیگی* - ۴۰۲۲۱۱۷۲۳
- *علی بابابیگ* - ۴۰۲۲۱۱۶۲۶
- *علی یعقوبیان* - ۴۰۲۲۰۴۳۲۸
- *محمد هادی حسینی* - ۴۰۲۲۰۴۷۲۵

فهرست

۱. عنوان پروژه و طرح مسئله

در این تمرین، هدف پیاده‌سازی مدلی برای پرسش و پاسخ در زمینه پزشکی، مبتنی بر بازیابی اطلاعات است. در واقع باید مدلی ارائه شود که با دریافت یک سوال پزشکی و متون بازیابی شده مرتبط با سوال، پاسخ را بر اساس اطلاعات موجود در متن ورودی بازگرداند.

برای پرسش و پاسخ از یک دیتاست شامل سوالات چهار گزینه‌ای استفاده می‌کنیم. همچنین رویکرد حل این مسئله بر اساس retrieval augmented generation بوده و دارای دو بخش زیر است:

1. **بازیابی متون مرتبط**

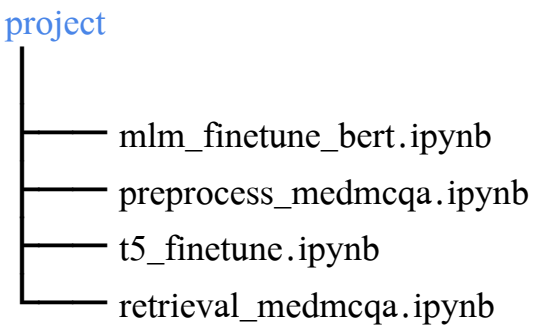
   به ازای سوال ورودی کاربر، سوالات مشابه با آن پیدا شده و توضیحات تشریحی آن‌ها بازگردانده می‌شود.

2. **تولید پاسخ**

   پس از بازیابی متون مرتبط، صورت سوال کاربر به همراه آن متون در قالب یک پرامپت به مدل زبانی T5 داده می‌شود تا گزینه‌ی صحیح را تولید کند.

۲. ساختار پروژه

در شماتیک زیر ساختار مربوط به این پروژه دیده می‌شود که در ادامه به توضیح هر بخش خواهیم پرداخت:



۳. دیتاست

در این تمرین از دیتاست MEDMCQA استفاده شده که شامل ۱۸۰ هزار نمونه است و ساختار نمونه‌های آن به صورت زیر می‌باشد:


```
{
    "id": "68221ed2-3524-48a8-9075-f6cc719f91a6",
    "question": "All of the following are pyrogenic cytokines, except:",
    "opa": "Interleukin 18 (IL-18)",
    "opb": "Interleukin 6 (IL-6)",
    "opc": "Tumor Necrosis Factor (TNF)",
    "opd": "Interferon a (IFN a)",
    "cop": 0,
    "choice_type": "multi",
    "exp": "Interleukin 18 is not a pyrogenic cytokine. IL-18, a member of the Il-1
    family doesnot appear to be a pyrogenic cytokine- harrison. Pyrogenic Cytokines:
    Cytokines are small protein that regulate immune, inflammatory and hematopoietic
    processes Those cytokines that cause fever &;are called as pyrogenic cytokines
    (formerly called Endogenous pyrogens) The pyrogenic cytokines include IL-1, IL-6,
    TNF, CNTF and IFNa (IL-18, a member of the IL-1 family doesnot appear to be a
    pyrogenic cytokine) IL-1 and TNF appear to be the most potent pyrogenic cytokines
    and induce fever at low doses while higher dose is required to induce fever with
    IL-6. Other pyrogenic cytokines probably exist but the above constitute the major
    pyrogenic cytokines Pyrogenic Cytokines: Interleukin-I (IL-1) Interleukin-6
    (IL-6) Tumor necrosis Factor (TNF) Interferon-a (IFN- a) Ref: Ananthanarayana
    10th edition Pgno: 156",
    "subject_name": "Microbiology",
    "topic_name": "Immunology"
}
```


همان طور که مشاهده می‌شود نمونه‌های این دیتاست سوال‌های چهار گزینه‌ای در حوزه پزشکی می‌باشند. ستون question صورت سوال را مشخص می‌کند، ستون های opa, opb, opc, opd به ترتیب گزینه های a, b, c, d را مشخص می‌کنند، ستون cop گزینه‌ی درست را مشخص می‌کند و ستون exp نیز توضیح تشریحی مربوط به آن سوال است.

همان طور که در بخش قبل توضیح داده شد در این تمرین قصد داریم مدلی ایجاد کنیم که بتواند به سوال‌های چهار گزینه‌ای این دیتاست پاسخ صحیح بدهد.

۴. فاین تیون کردن مدل Bert

کدهای این بخش در فایل mlm_finetune_bert.ipynb قرار دارد.

اولین قدم برای پاسخگویی به کوئری کاربر (که یک سوال چهار گزینه‌ای است) آن است که بتوانیم سوالات مشابه آن را بازیابی کنیم. به این منظور از یک مدل انکودر بر پایه‌ی Bert استفاده می‌کنیم تا بتوانیم به ازای هر سوال یک بردار امبدینگ به دست آوریم. بنابراین اگر این امبدینگ معنادار باشد می‌توانیم با یک معیار مشابهت ساده مانند Cosine Similarity سوالات مشابه را پیدا کنیم.

با توجه به اینکه دیتاست مورد استفاده در حوزه پزشکی قرار دارد بهتر است مدل انکودر استفاده شده نیز با ادبیات این حوزه آشنا باشد. به این منظور از مدل BioClinicalBert استفاده می‌کنیم. این مدل بر پایه‌ی Bert-Base است و بر روی حدود ۸۸۰ میلیون توکن از دیتاست MIMIC فاین تیون شده است. این دیتاست شامل رکورد های بیماران است و باعث می‌شود مدل انکودر مان با ادبیات حوزه پزشکی آشنا باشد.

شایان ذکر است که مدل های انکودر متنوعی در حوزه پزشکی وجود دارد. ما در این تمرین چندین مدل مختلف اعم از ClinicalBert، BlueBert, BioClinicalBert, Bert را امتحان کردیم و مشاهده شد که مدل BioClinicalBert بهترین نتیجه را به دست می‌دهد.

برای بازیابی سوالات مشابه باید به ازای هر سوال امبدینگ های معناداری به دست آوریم. به این منظور صورت سوال‌ها را به مدل انکودر ورودی می‌دهیم و بردار خروجی مربوط به CLS token آن را ذخیره می‌کنیم.

در ابتدا فرض کردیم که صرفا استفاده از مدل BioClinicalBert و استخراج CLS token خروجی آن به ازای هر سوال کافی باشد اما مشاهده شد که با این روش بردار های به دست آمده به اندازه کافی خوب نیستند و تنها در حالتی سوال مد نظر را می‌توان بازیابی کرد که عینا همان سوال به عنوان کوئری داده شود. به عبارت دیگر با اینکه مدل pretrained استفاده شده با حوزه پزشکی آشنایی دارد اما به اندازه کافی با دیتاست MEDMCQA و ادبیات مورد استفاده در سوالات آن آشنایی ندارد.

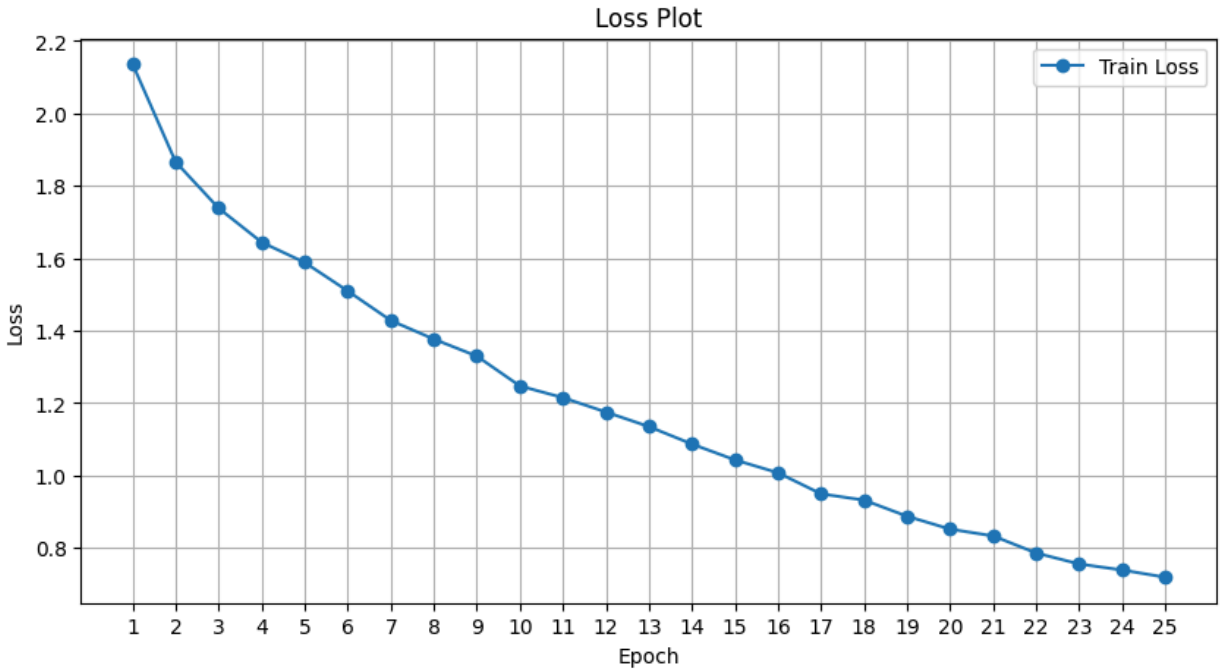
برای رفع این مشکل مدل BioClinicalBert را بر روی زیرمجموعه‌ی کوچکی از دیتاست به اندازه ۴۰ هزار نمونه به صورت MLM فاین تیون کردیم. فرایند فاین تیون به اندازه ۲۵ ایپاک بر روی توضیحات تشریحی و همچنین صورت سوال‌ها انجام شد و مشاهده شد که به این صورت بردار های CLS به دست آمده بسیار معنادار تر شده و حتی اگر صورت سوال کوئری را تا حدی تغییر دهیم، مدل همچنان می‌تواند سوالات مشابه آن را پیدا کند.

در زیر کد مربوط به MLM fine-tuning آمده است:

```python
def filter_none(example):
    return (
        (example["exp"] is not None)
        and (len(example["exp"]) > 20)
        and (example["question"] is not None)
    )


def mlm_map_function(rows):
    input_info = tokenizer(
        rows["exp"],
        max_length=128,
        padding="max_length",
        truncation=True,
        return_tensors="pt",
    )
    return {**input_info, "labels": input_info["input_ids"]}


dataset = load_dataset(medmcqa_dataset_path)
mlm_dataset = dataset["train"].select(trainset_range)
mlm_dataset = mlm_dataset.filter(filter_none).select_columns(["exp"])
mlm_dataset = mlm_dataset.map(
    mlm_map_function,
    batched=True,
    num_proc=2,
)
print(mlm_dataset)

collate_fn = DataCollatorForLanguageModeling(
    tokenizer=tokenizer, mlm=True, mlm_probability=0.15
)
```

```python
training_args = TrainingArguments(
    output_dir="./results",
    learning_rate=lr,
    per_device_train_batch_size=batch_size,
    num_train_epochs=num_epochs,
    save_strategy="no",  # Disable checkpointing
    logging_steps=len(mlm_dataset) // batch_size,  # Log per epoch
    report_to=[],  # Disable wandb logging
)

trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=collate_fn,
    train_dataset=mlm_dataset,
)

results = trainer.train()
```

نمودار تغییرات training loss نیز در زیر آورده شده است:



در نهایت نیز مدل ایجاد شده در مخزن هاگینگ فیس ذخیره می‌شود تا در نوتبوک های بعدی مورد استفاده قرار گیرد.

۵. ایجاد دیتاست بازیابی

کدهای این بخش در فایل preprocess_medmcqa.ipynb قرار دارد.

در این بخش دیتاست مربوط به تسک بازیابی سوالات مشابه را آماده می‌کنیم. این دیتاست شامل ستون‌های question, exp, question_cls می‌باشد.ستون‌های question و exp متن مربوط به صورت سوال و پاسخ تشریحی آن می‌باشند که در دیتاست اصلی وجود داشتند. ستون question_cls نیز توسط مدل انکودری که در بخش قبل توضیح داده شد به دست می‌آید.

ابتدا مدل انکودر بخش قبل را از مخزن هاگینگ فیس لود می‌کنیم:

```
tokenizer = BertTokenizer.from_pretrained(base_bert_path)
model = BertForMaskedLM.from_pretrained(base_bert_path).to(device)

checkpoint_file = hf_hub_download(repo_id=repo_id, filename=finetuned_bert_path)
checkpoint = torch.load(checkpoint_file)
model.load_state_dict(checkpoint["model_state_dict"])
model = model.bert  # dropping MLM head
model.eval()
```

کد مربوط به ایجاد ستون question_cls در زیر مشاهده می‌شود:

```python
max_length = 128
cls_tokens = []
for batch in tqdm(dataloader):
    batch["question"] = [preprocess_text(txt) for txt in batch["question"]]
    tokens = tokenizer(
        batch["question"],
        padding="max_length",
        truncation=True,
        max_length=max_length,
        return_tensors="pt",
    )
    input_ids = tokens["input_ids"].to(device)
    att_mask = tokens["attention_mask"].to(device)

    with torch.no_grad():
        outputs = model(input_ids, att_mask)

    if "pooler_output" in outputs:
        cls_embedding = outputs.pooler_output
    elif "last_hidden_state" in outputs:
        cls_embedding = outputs.last_hidden_state[:, 0, :].squeeze()
    else:
        raise Exception("No CLS token found in the given model")

    cls_embedding = cls_embedding.cpu().numpy().tolist()
    cls_tokens += cls_embedding
```

همان طور که مشاهده می‌شود به ازای هر نمونه ابتدا یک پیش‌پردازش ساده بر روی صورت سوال انجام می‌شود و سپس متن پیش‌پردازش شده به توکنایزر و مدل انکودر داده می‌شود. در نهایت CLS token های به دست آمده به عنوان ستون جدید به دیتاست اضافه می‌شوند که نتیجه آن در زیر مشاهده می‌شود:

	question	exp	question_cls
0	All of the following are pyrogenic cytokines, ...	Interleukin 18 is not a pyrogenic cytokine. IL...	[0.5136734843254089, -0.6453795433044434, 0.18...
1	40-year old female presented with neck swellin...	Ref. Robbins Pathology. 9th edition. Page. 109...	[-0.18327629566192627, -0.23770320415496826, -...
2	Following statement regarding dislocation of t...	Anterior dislocation is more common in which h...	[-0.4960843026638031, -0.30119097232818604, 0....
3	The active search for unrecognized disease or ...	Screening is the search for unrecognized disea...	[0.05901067703962326, 0.06995794922113419, -0....
4	Fir tree pattern lesion is seen in	Fir tree pattern of distribution of lesions is...	[-0.3408776819705963, -0.5097606778144836, -0....
...	...	...	...

دلیل اعمال پیش‌پردازش بر روی صورت سوال قبل از ورودی دادن آن به مدل انکودر آن است که مشاهده شد مدل انکودر به برخی stop word ها حساس است و اثر آن‌ها در CLS token به دست آمده بیشتر از حد مطلوب می‌باشد. حال آن که در صورت مسئله‌ی ما کلمات کلیدی موجود در صورت سوال مهم‌ترین عامل مشابهت است. در زیر کد مربوط به تابع پیش‌پردازش آورده شده است:

```
def preprocess_text(text):
    tokens = word_tokenize(text)
    tokens = [word.lower() for word in tokens]
    tokens = [word for word in tokens if word.isalpha()]
    stop_words = set(stopwords.words("english"))
    tokens = [word for word in tokens if word not in stop_words]
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return " ".join(tokens)
```

این تابع از کتابخانه‌ی NLTK استفاده می‌کند. پایپلاین پیش‌پردازش عملیات های زیر را به ترتیب انجام می‌دهد:

- متن ورودی توکنایز می‌شود
- تمام کلمات به حالت lowercase برده می‌شوند
- حروف غیر از الفبا حذف می‌شوند
- stop word ها حذف می‌شوند
- تمام کلمات lemmatize می‌شوند

در نهایت نیز دیتاست ایجاد شده در مخزن هاگینگ فیس ذخیره می‌شود تا در نوتبوک های بعدی مورد استفاده قرار گیرد.

*شایان ذکر است که طول دیتاست این بخش می‌تواند بسیار بزرگ باشد و هر چقدر که این دیتاست بزرگتر باشد، بانک سوال های بیشتری در اختیار مدل قرار می‌گیرد تا بتواند به سوالات بیشتر و متنوع تری پاسخ دهد. ما اما به دلیل محدودیت منابع زیرمجموعه ای از دیتاست اصلی به اندازه‌ی ۲۰ هزار نمونه را انتخاب کردیم.

۶. فاین تیون کردن مدل T5

کدهای این بخش در فایل t5_finetune.ipynb قرار دارد.

برای تولید متن نیاز به یک مدل زبانی داریم. با توجه به محدودیت منابع پردازشی، مدل T5-small را برای تولید پاسخ نهایی مدل استفاده کردیم.

مدل T5 به طور کلی برای تولید متن استفاده می‌شود و در دامنه های متنوعی آموزش دیده است. در این تسک اما نیاز داریم که مدل زبانی با دریافت یک پرامپت شامل صورت سوال، گزینه ها و متون مرتبط با سوال، گزینه‌ی صحیح را انتخاب کند. بنابراین نیاز است تا این مدل را بر روی تسک مورد نظر خود یعنی پرسش و پاسخ سوالات چهار گزینه‌ای، فاین تیون کنیم. در غیر اینصورت مدل جواب‌هایی خارج از پرامپت های در نظر گرفته شده تولید می‌کند و دقت بسیار پایینی می‌گیرد.

با توجه به اینکه این مدل تعداد پارامتر های زیادی دارد و با انواع تسک ها آموزش دیده است، انجام full fine-tuning هم بسیار کند خواهد بود و هم ممکن است concept های یاد گرفته شده توسط مدل را خراب کند. بنابراین از تکنیک های PEFT و به طور خاص Adapter برای فاین تیون مدل T5 استفاده می‌کنیم.

در این روش، تمام قسمت‌های مدل T5 فریز شده و تنها با اضافه کردن لایه‌های Adapter، این لایه‌های اضافه شده آموزش داده می‌شوند.

ابتدا مدل T5-small که به صورت عمومی آموزش‌داده‌شده است، به همراه tokenizer آن دانلود می‌شود:

```
tokenizer = T5Tokenizer.from_pretrained(base_t5_path)
model = T5ForConditionalGeneration.from_pretrained(base_t5_path)
```

سپس لایه‌های Adapter به مدل اصلی اضافه می‌شوند:

```python
# Adapter layer
class AdapterLayer(nn.Module):
    def __init__(self, emb_dim: int, bottleneck_size: int):

        super().__init__()

        self.sharif_llm_adapter = nn.Sequential(
            nn.Linear(emb_dim, bottleneck_size),
            nn.ReLU(),
            nn.Linear(bottleneck_size, emb_dim),
        )

    def forward(self, x: torch.Tensor):
        adapter_output = self.sharif_llm_adapter(x)
        output = x + adapter_output
        return output


class FeedForwardAdapterWrapper(nn.Module):
    def __init__(self, original_module: T5LayerFF, bottleneck_size: int):

        super().__init__()
        assert isinstance(original_module, T5LayerFF)

        self.original_module = original_module
        emb_dim = original_module.DenseReluDense.wi.in_features
        self.adapter = AdapterLayer(emb_dim, bottleneck_size)

    def forward(self, x: torch.Tensor):
        output = self.original_module(x)
        output = self.adapter(output)
        return output
```

```python
# Add adapter to the model
def mutate_model_recursive(model: nn.Module, bottleneck_size: int):
    for name, module in model.named_children():
        if isinstance(module, T5LayerFF):
            feed_forward_with_adapter = FeedForwardAdapterWrapper(
                module, bottleneck_size
            )
            setattr(model, name, feed_forward_with_adapter)
            print(f"Replaced {name} with FeedForwardAdapterWrapper layer.")
        else:
            mutate_model_recursive(module, bottleneck_size)


def mutate_model(model: nn.Module, bottleneck_size: int):
    if hasattr(model, "_mutated"):
        print("Model already contains adapter layers! \n Try reloading the model.")
        return

    mutate_model_recursive(model, bottleneck_size)

    model._mutated = True


mutate_model(model, bottleneck_size=bottleneck_size)
```

و سپس وزن‌های مدل به غیر از لایه‌های Adapter فریز می‌شوند:

```python
# Freeze non-adapter parameters
def freeze_non_adapter(model, peft_key):
    print("Non freezed weights:")
    total_params = 0
    for param_name, weights in model.named_parameters():
        weights.requires_grad = peft_key in param_name
        if weights.requires_grad:
            print(param_name)
            total_params += weights.numel()
    print(f"Total number of parameters should be update: {total_params}")


freeze_non_adapter(model, peft_key="sharif_llm")
```

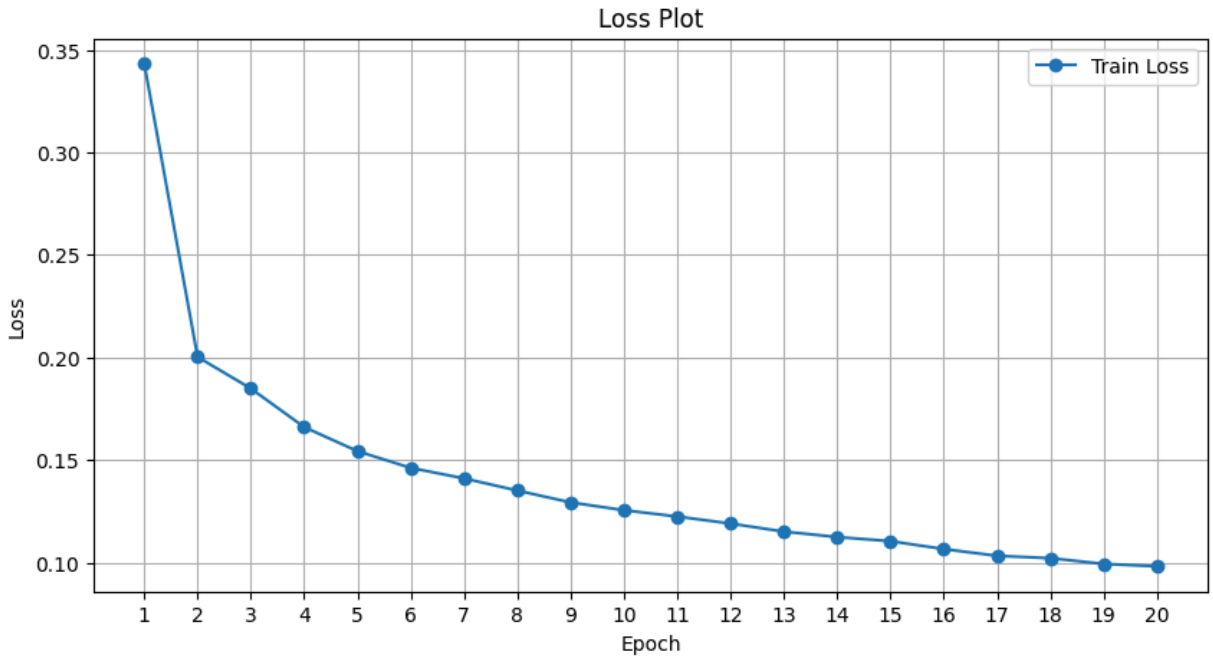
از دیتاست MEDMCQA سه قسمت برای داده‌ی آموزش، تست و ارزیابی جدا کردیم. داده‌ی آموزش را زیرمجموعه‌ی کوچکی از دیتاست به اندازه ۱۵ هزار نمونه، داده‌ی تست را زیرمجموعه‌ی کوچکی از دیتاست به اندازه ۴ هزار نمونه و داده‌ی validation را برابر تمام دیتاست validation قرار دادیم.

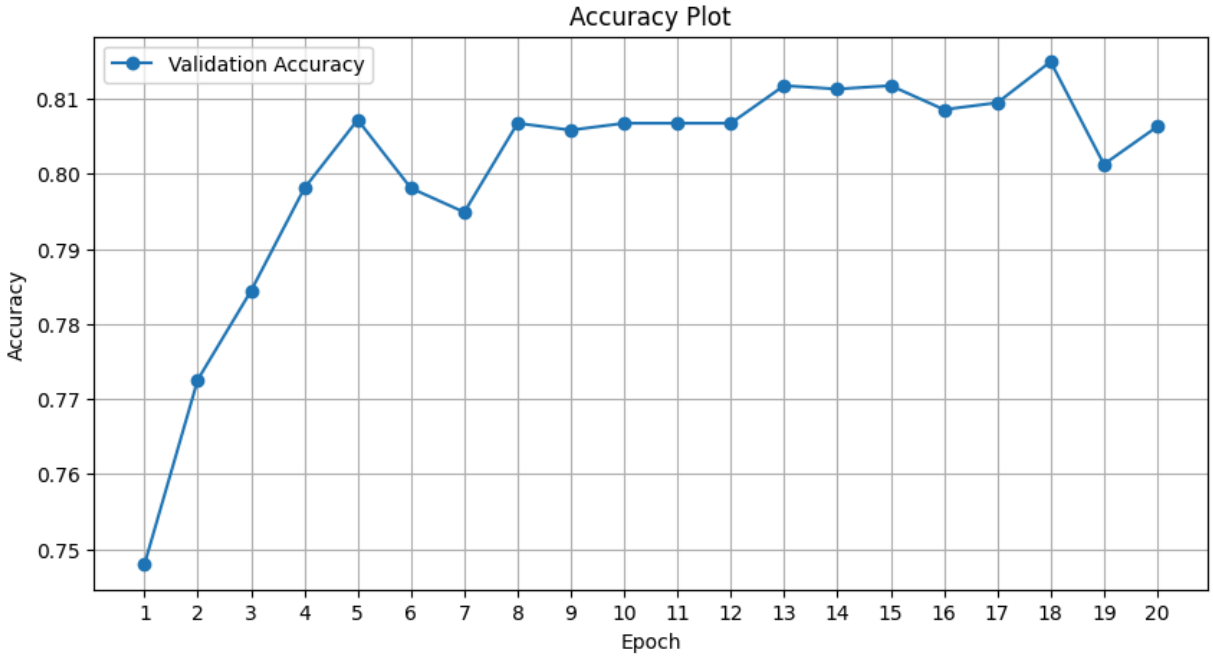
با توجه به آن که مدل تنها یک رشته متن ورودی دریافت می‌کند نیاز است تا یک قالب پرامپت برای آن ایجاد کنیم. در تصویر زیر می‌توان این قالب را برای ورودی مدل و همچنین خروجی آن مشاهده کرد.

```
def format_example_training(row):
    input_text = f"Question: {row['question']}\n\nOptions:\nA: {row['opa']}\nB: {row['opb']}
    \nC: {row['opc']}\nD: {row['opd']}\n\nExplanation: {row['exp']}\n\nAnswer:"
    target_text = f"Answer: {opt_idx2str[row['cop']]}"
    return {"input_text": input_text, "target_text": target_text}


def format_example_validation(row):
    input_text = f"Question: {row['question']}\n\nOptions:\nA: {row['opa']}\nB: {row['opb']}
    \nC: {row['opc']}\nD: {row['opd']}\n\nExplanation: {row['exp']}\n\nAnswer:"
    target_text = f"Answer: {opt_idx2str[row['cop']]}"
    return {"input_text": input_text, "target_text": target_text}


train_dataset = train_dataset.map(
    format_example_training, remove_columns=train_dataset.column_names
)
test_dataset = test_dataset.map(
    format_example_training, remove_columns=test_dataset.column_names
)
dataset["validation"] = dataset["validation"].map(
    format_example_validation, remove_columns=dataset["validation"].column_names
)
```

قالب پرامپت استفاده شده یکی از حالت‌های معروفی است که برای تسک پرسش و پاسخ سوالات چهار گزینه‌ای استفاده می‌شود. فرمت دیگری نیز بررسی شد که ابتدا context قرار می‌گرفت و سپس صورت سوال بعد از آن می‌آمد اما نتایج آن تفاوت زیادی با این حالت نداشت.

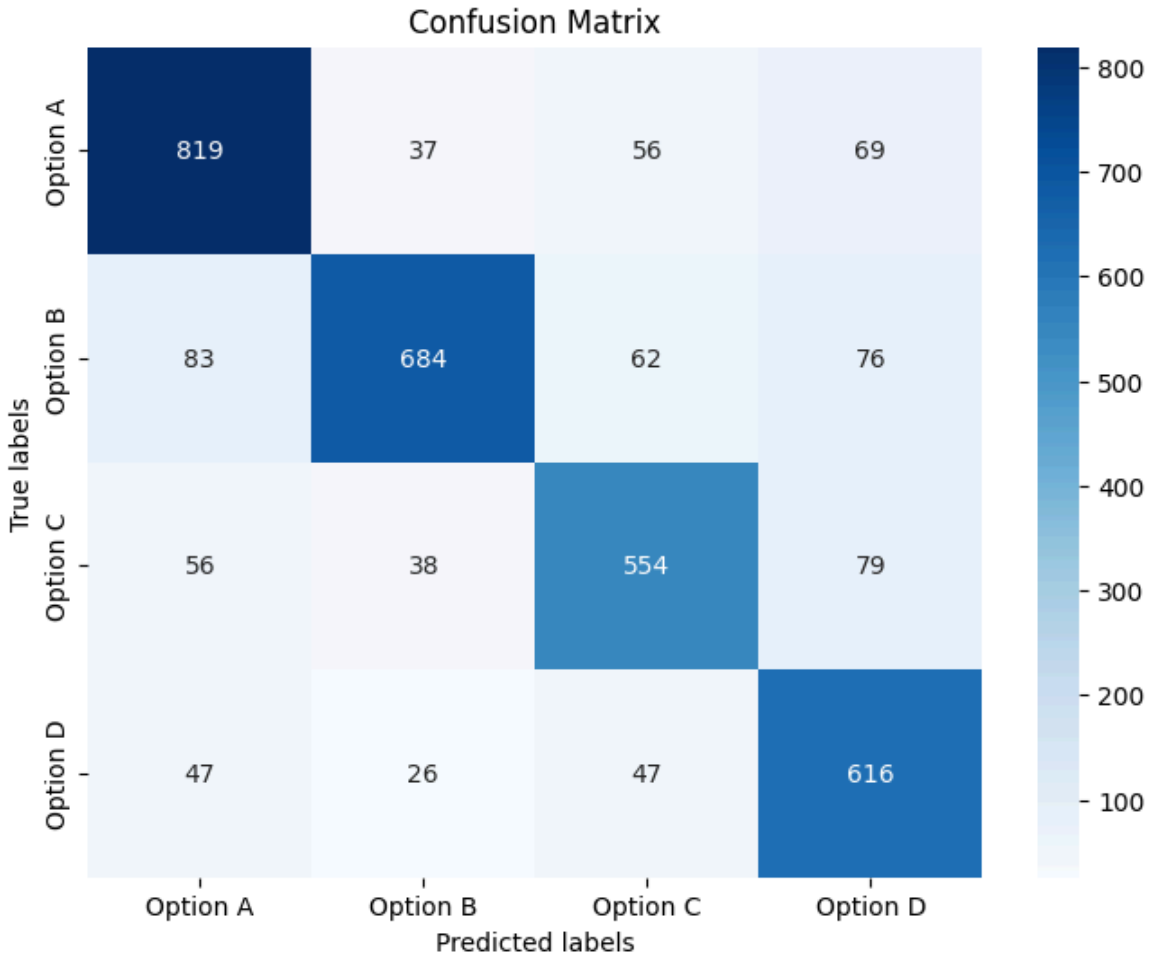
مدل در طی ۲۰ ایپاک آموزش داده شد و نمودار تغییرات training loss و validation accuracy به صورت زیر است:


Loss Plot
Train Loss
Loss
Epoch
0.35
0.30
0.25
0.20
0.15
0.10
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


Accuracy Plot
Validation Accuracy
Accuracy
Epoch
0.81
0.80
0.79
0.78
0.77
0.76
0.75
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

نتایج ارزیابی مدل بر روی داده‌های تست نیز به صورت زیر است:

```
Accuracy        =  79.81%
Macro F1-score  =  79.56%
Micro F1-score  =  79.81%
Macro Precision =  79.75%
Macro Recall    =  79.74%
```



همچنین عملکرد مدل T5 بر روی داده‌های تست قبل از اعمال فاین‌تیونینگ نیز بررسی شد. دقیقا همان مجموعه داده‌های تست به مدل داده شد و در سؤال به طور کاملاً صریح خواسته شده بود که مدل یک گزینه را انتخاب کند. اما مدل در این حالت دقتی معادل ۰.۰۱٪ داشت و توکن‌هایی کاملاً تصادفی و بی‌ربط تولید می‌کرد. در زیر می‌توانید قسمتی از پاسخ درست و پاسخ مدل T5 را برای این مجموعه داده‌های تست مشاهده کنید:

```
['1', '1', '3', '1', '0', '3', '0', '2', '0', '0', '3', '2', '3', '2', '0', '1',
['CO2 rect', 'Porphyri', 'agenesis of', 'intradermally on', 'Cyclopia', 'cisa',
```

در نهایت نیز مدل ایجاد شده در مخزن هاگینگ فیس ذخیره می‌شود تا در نوتبوک های بعدی مورد استفاده قرار گیرد.

۷. تجمیع پایپ‌لاین RAG

کدهای این بخش در فایل retrieval_medmcqa.ipynb قرار دارد.

این بخش را به تعبیری می‌توان ترکیب نتایج بخش‌های قبلی در نظر گرفت. در این بخش، مدل‌های فاین‌تیون‌شده‌ی BioClinicalBERT و T5 و همچنین، دیتاست بازیابی تهیه شده (که در بخش‌های قبلی توضیح داده شد) از HuggingFace بارگذاری می‌شوند تا در ادامه مورد استفاده قرار بگیرند.

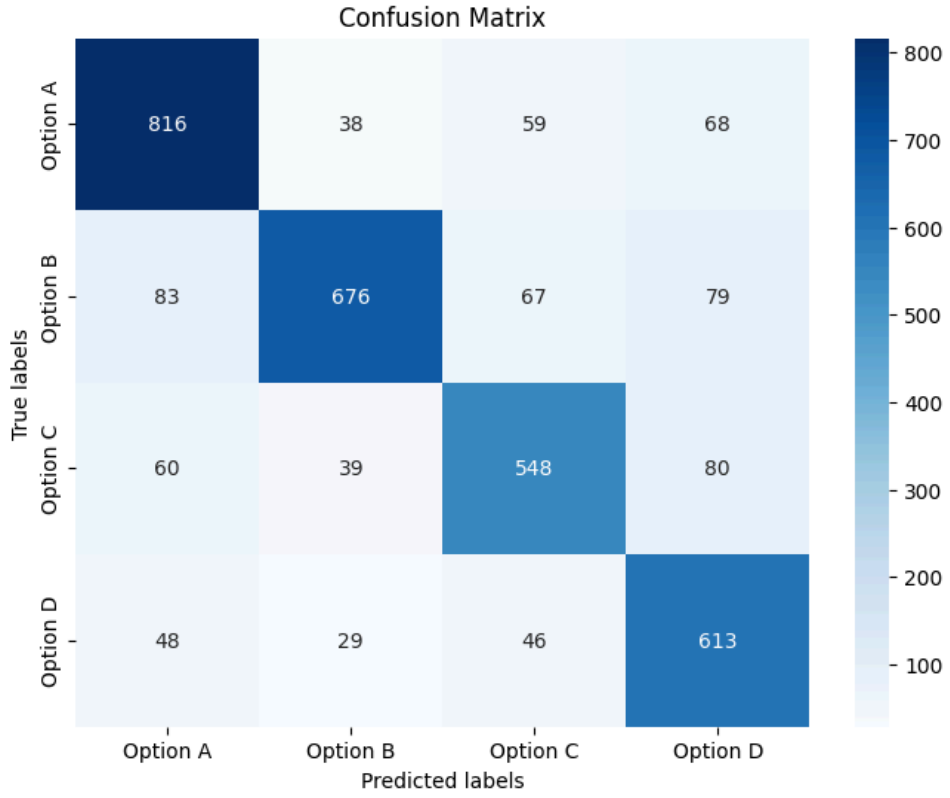
در مهم‌ترین بخش این نوت‌بوک، عملکرد پایپ‌لاین RAG پیاده‌سازی شده بر روی داده‌های تست، سنجیده می‌شود. روال کار در این بخش به این صورت است که:

1. ابتدا، برای هر یک از داده‌های تست (که یک سوال پزشکی به همراه چهار گزینه می‌باشد) مقدار امبدینگ توکن CLS (صرفاً بر روی متن سوال) توسط مدل فاین‌تیون‌شده‌ی BioClinicalBERT محاسبه می‌شود.
2. سپس، با اعمال الگوریتم KNN (به ازای مقادیر k={1, 2, 3}) بر روی مجموعه دادگان بازیابی، k تا از داده‌هایی که از نظر معیار شباهت مورد استفاده (cosine similarity) بیشترین شباهت را با داده‌ی تست مذکور داشته باشند، انتخاب می‌شوند.
3. در ادامه، به ازای هر یک از داده‌های بازیابی شده، پاسخ تشریحی آن سوال را جدا می‌کنیم و با چسباندن پاسخ سوال‌های بازیابی شده، یک context مناسب برای مدل T5 آماده می‌شود.
4. حال، سوال پزشکی اولیه (موجود در مجموعه داده‌ی تست) به همراه پاسخ تشریحی سوالات مرتبط بازیابی شده (موجود در مجموعه داده‌ی بازیابی) در قالب یک پرامپت از پیش تعیین‌شده، به مدل T5 داده می‌شود.
5. در پایان، خروجی مدل T5 به ازای داده‌ی تست مذکور با خروجی صحیح آن داده‌ی تست مطابقت داده شده و توسط معیارهای متعددی، مورد ارزیابی قرار می‌گیرد.

همچنین، همان‌طور که در صورت تمرین اشاره شده بود، تست مذکور یک‌بار با استفاده از مدل T5 خام (بدون فاین‌تیون کردن) نیز تکرار شده است تا اثر فاین‌تیونینگ بر عملکرد نهایی خط‌لوله‌ی RAG مشخص شود. در ادامه، نتایج RAG به ازای پیکربندی‌های مختلف آورده شده است.

مدل T5 فاین‌تیون شده / k=1

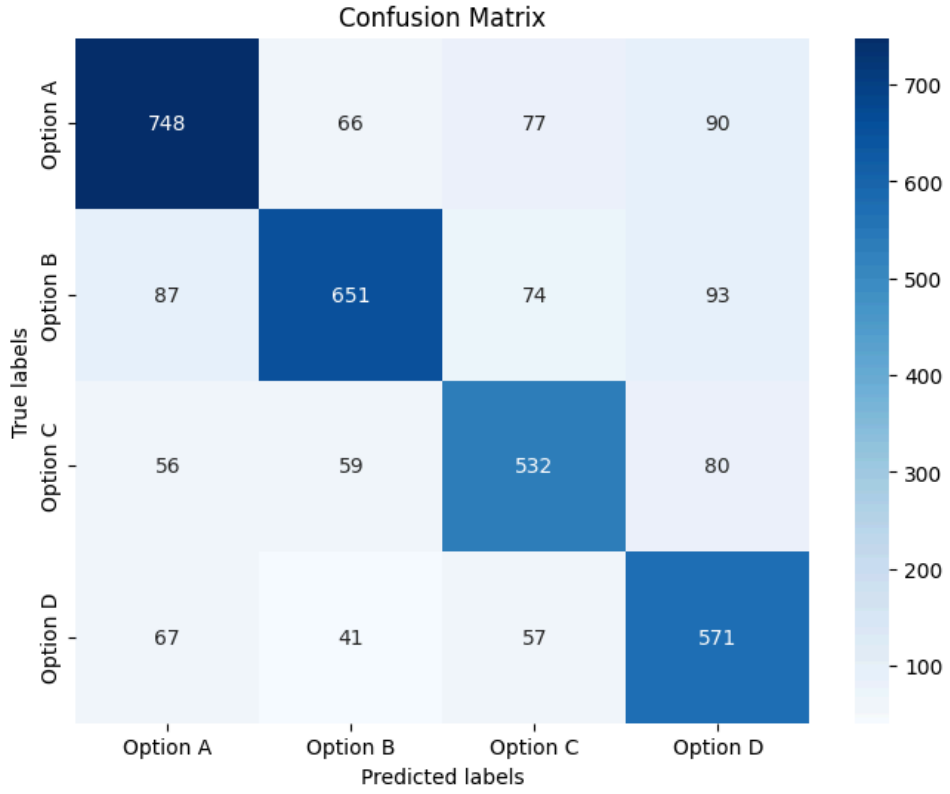
```
Accuracy        =  79.22%
Macro F1-score  =  78.94%
Micro F1-score  =  79.22%
Macro Precision =  79.14%
Macro Recall    =  79.14%
```




```
BLEU-1          =  0.93073
BLEU-2          =  0.91216
BLEU-3          =  0.74526
BLEU-4          =  0.67181

ROUGE-1         =  0.89609
ROUGE-2         =  0.79218
ROUGE-L         =  0.89609

BERT Precision  =  0.94694
BERT Recall     =  0.94694
BERT F1 Score   =  0.94703
```

مدل T5 فاین‌تیون شده / k=2

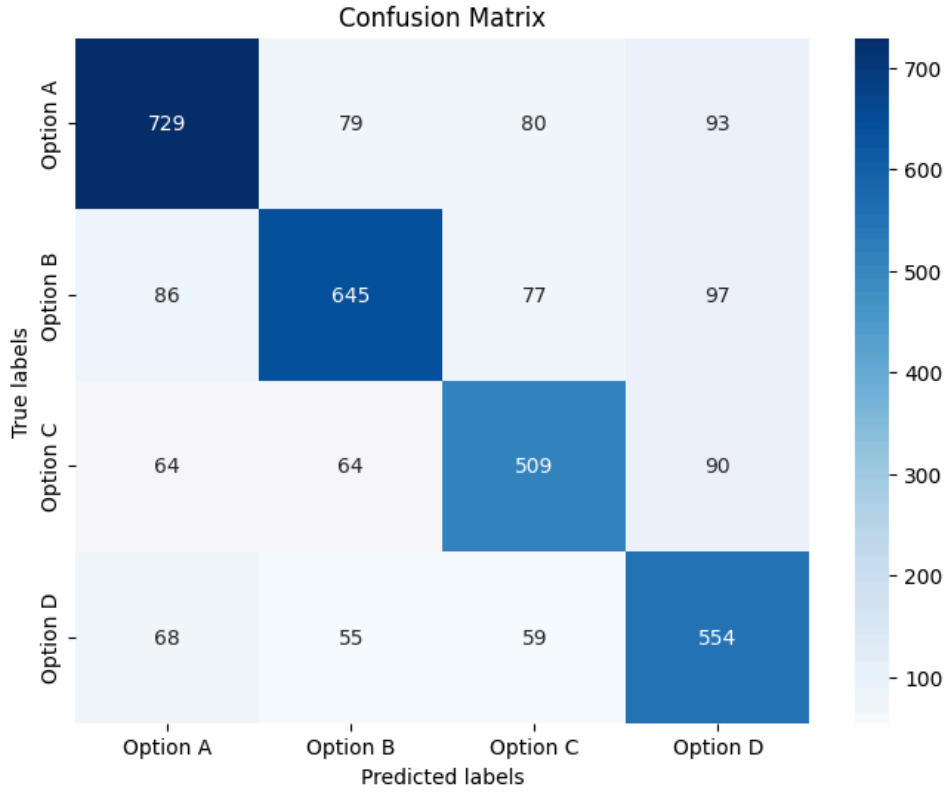
```
Accuracy        =  74.71%
Macro F1-score  =  74.51%
Micro F1-score  =  74.71%
Macro Precision =  74.53%
Macro Recall    =  74.74%
```




```
BLEU-1          =  0.91570
BLEU-2          =  0.89311
BLEU-3          =  0.73435
BLEU-4          =  0.66416

ROUGE-1         =  0.87354
ROUGE-2         =  0.74709
ROUGE-L         =  0.87354

BERT Precision  =  0.93545
BERT Recall     =  0.93546
BERT F1 Score   =  0.93556
```

مدل T5 فاین‌تیون شده / k=3

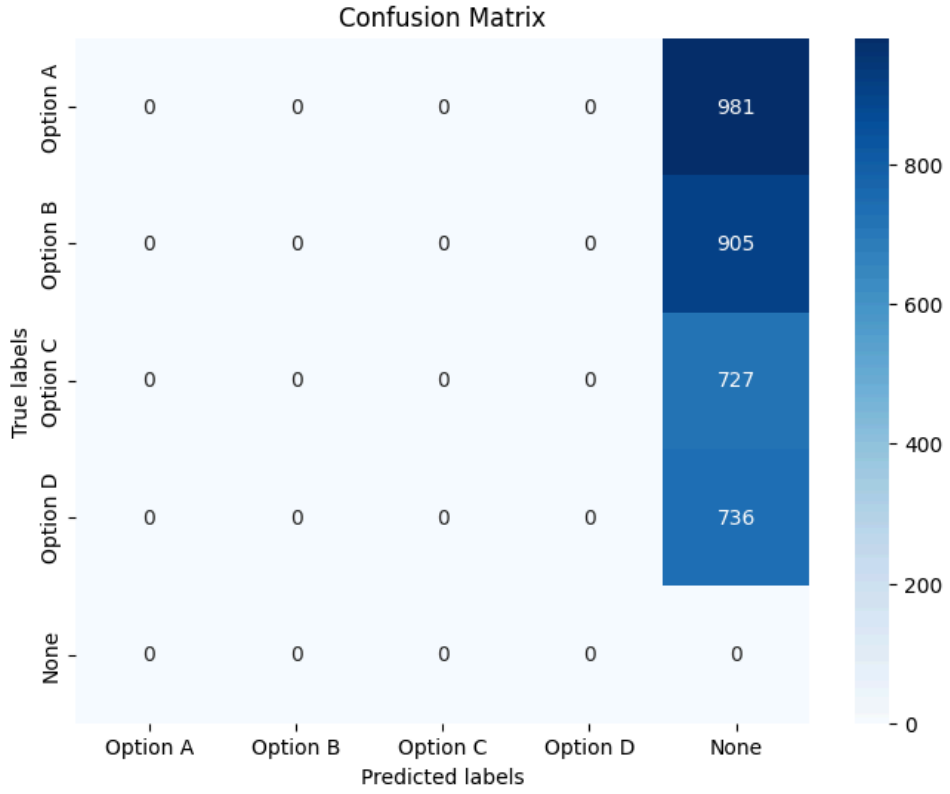
```
Accuracy        =  72.77%
Macro F1-score  =  72.53%
Micro F1-score  =  72.77%
Macro Precision =  72.53%
Macro Recall    =  72.72%
```




```
BLEU-1          =  0.90923
BLEU-2          =  0.88490
BLEU-3          =  0.72966
BLEU-4          =  0.66086

ROUGE-1         =  0.86384
ROUGE-2         =  0.72768
ROUGE-L         =  0.86384

BERT Precision  =  0.93061
BERT Recall     =  0.93061
BERT F1 Score   =  0.93072
```

مدل T5 خام / k=1

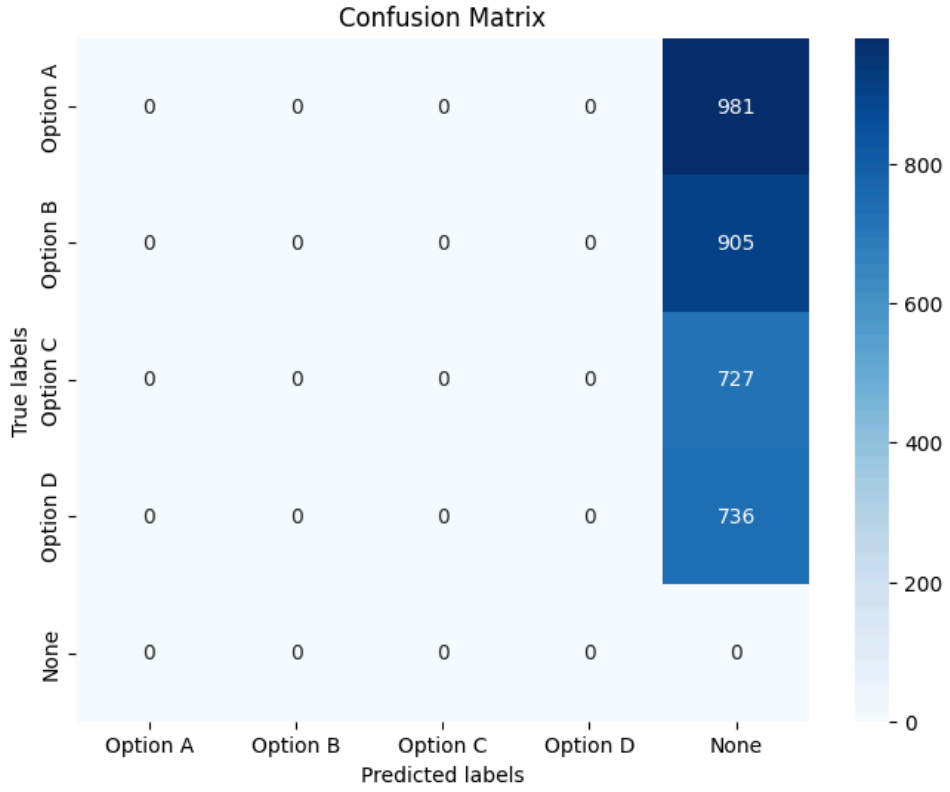
```
Accuracy        =  0.00%
Macro F1-score  =  0.00%
Micro F1-score  =  0.00%
Macro Precision =  0.00%
Macro Recall    =  0.00%
```




```
BLEU-1          =  0.03878
BLEU-2          =  0.02854
BLEU-3          =  0.02679
BLEU-4          =  0.02575

ROUGE-1         =  0.06066
ROUGE-2         =  0.00058
ROUGE-L         =  0.05800

BERT Precision =  -0.12086
BERT Recall    =  -0.02083
BERT F1 Score  =  -0.07951
```

مدل T5 خام / k=2

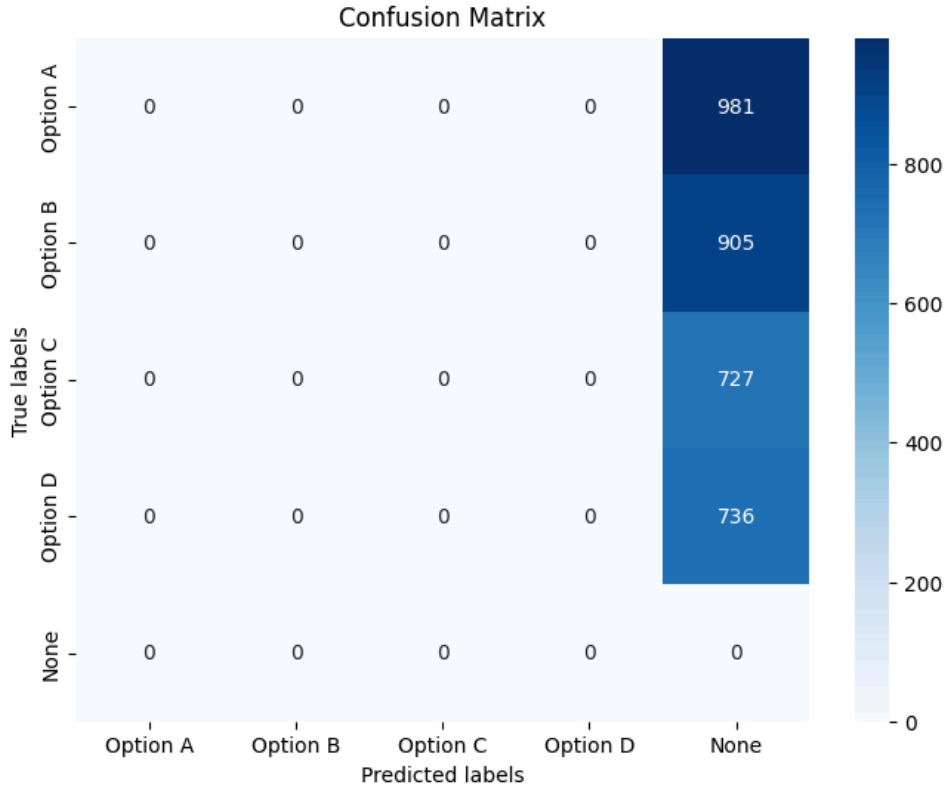
```
Accuracy        =  0.00%
Macro F1-score  =  0.00%
Micro F1-score  =  0.00%
Macro Precision =  0.00%
Macro Recall    =  0.00%
```




```
BLEU-1          =  0.02187
BLEU-2          =  0.01612
BLEU-3          =  0.01528
BLEU-4          =  0.01478

ROUGE-1         =  0.03885
ROUGE-2         =  0.00033
ROUGE-L         =  0.03717

BERT Precision =  -0.31922
BERT Recall    =  -0.07895
BERT F1 Score  =  -0.20612
```

مدل T5 خام / k=3

```
Accuracy        =  0.00%
Macro F1-score  =  0.00%
Micro F1-score  =  0.00%
Macro Precision =  0.00%
Macro Recall    =  0.00%
```




```
BLEU-1          =  0.01285
BLEU-2          =  0.00953
BLEU-3          =  0.00917
BLEU-4          =  0.00893

ROUGE-1         =  0.02759
ROUGE-2         =  0.00033
ROUGE-L         =  0.02645

BERT Precision =  -0.40838
BERT Recall    =  -0.11437
BERT F1 Score  =  -0.26751
```

۸. ارزیابی و مقایسه‌ی مدل‌ها روی چند ورودی دلخواه

علاوه بر ارزیابی انجام شده بر روی داده‌های تست که در بخش قبلی مورد بررسی قرار گرفت، در این بخش قصد داریم عملکرد پایپ‌لاین RAG را با داده‌هایی خارج از مجموعه‌ی داده مورد بررسی قرار دهیم.

برای این کار، با در نظر گرفتن اطلاعات موجود در مجموعه‌داده‌ی بازیابی، تعدادی نمونه‌ی سوال را به صورت دستی طراحی کرده‌ایم. ویژگی مهم این نمونه‌ها آن است که اطلاعات مورد نیاز برای پاسخ‌گویی به آن‌ها در مجموعه‌داده‌ی بازیابی وجود دارد، اما صورت و گزینه‌های سوال، نسبت به سوالات مشابه موجود در داده‌های بازیابی کاملاً متفاوت است. بنابراین، مدلی که بتواند به این سوال‌ها به درستی پاسخ دهد، احتمالاً تعمیم‌پذیری مناسبی خواهد داشت.

همچنین، برای اطمینان از درستی سوالات طراحی شده، سوالات را از GPT-4o پرسیده و اطمینان حاصل کرده‌ایم که این مدل، بدون هیچ مشکلی و به درستی می‌تواند سوالات را پاسخ دهد.

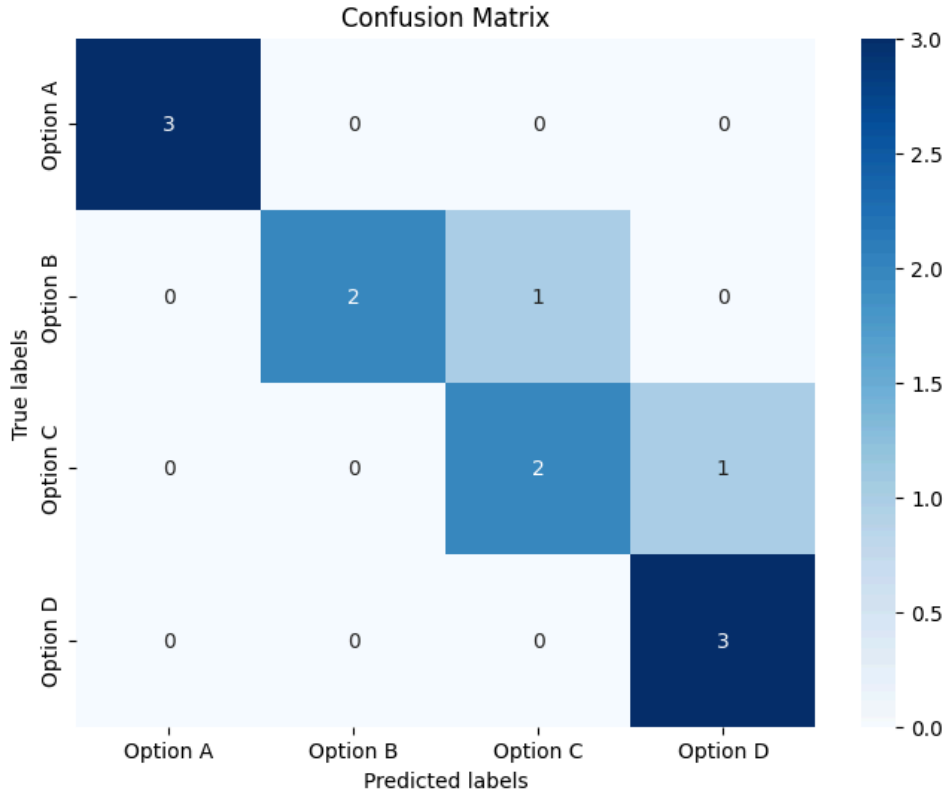
در زیر، چند نمونه از این سوالات طراحی شده را مشاهده می‌کنیم:


```
{
    "question": "Which of the following causes a decrease in ESR?",
    "opa": "Sickle cell anaemia",
    "opb": "Inflammation",
    "opc": "COVID-19",
    "opd": "Pregnancy",
    "cop": 0,
},
{
    "question": "Which option could be identified using the cephalic index?",
    "opa": "Blood type",
    "opb": "Sex",
    "opc": "Hair color",
    "opd": "Race",
    "cop": 3,
},
{
    "question": "In which case is a magistrate inquest NOT required?",
    "opa": "Death in police custody",
    "opb": "Death in police firing",
    "opc": "Death by suicide",
    "opd": "Death in psychiatry hospital",
    "cop": 2,
},
{
    "question": "Which test is most related to Addison's disease (i.e. adrenal insufficiency)?",
    "opa": "ACTH (Cosyntropin) test",
    "opb": "Blood glucose test",
    "opc": "MRI of the adrenal glands",
    "opd": "CT scan of abdomen",
    "cop": 0,
},
```


در ادامه، نتیجه‌ی ارزیابی عملکرد مدل بر روی این نمونه‌داده‌ها آورده شده است.

مدل T5 فاین‌تیون شده / k=3

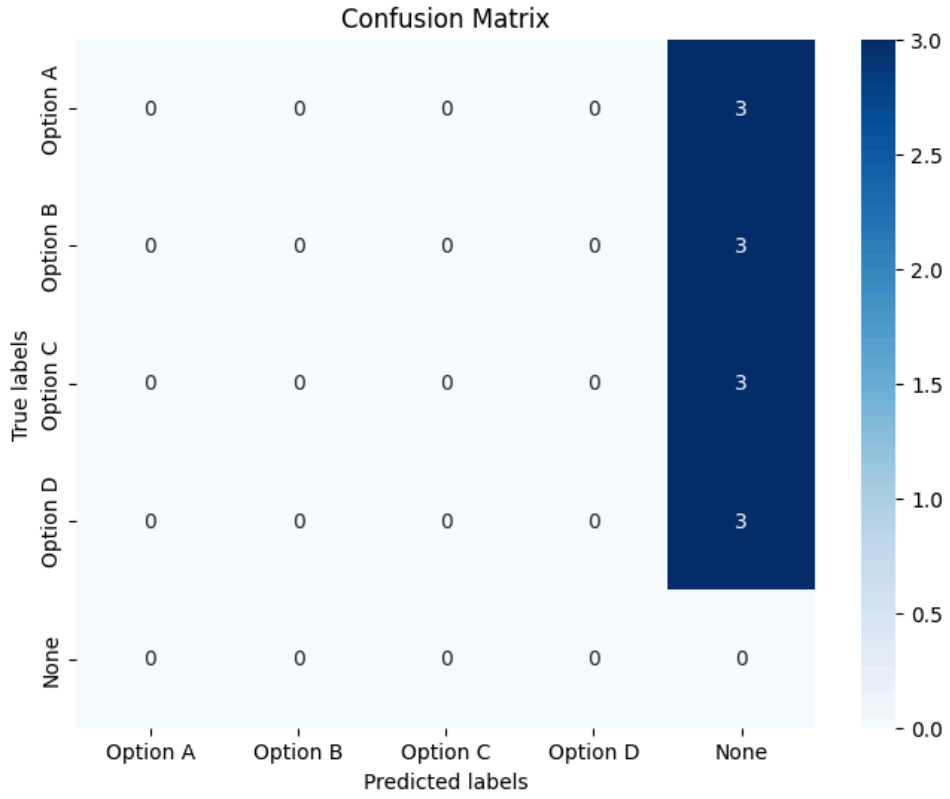
```
Accuracy        =  83.33%
Macro F1-score  =  83.10%
Micro F1-score  =  83.33%
Macro Precision =  85.42%
Macro Recall    =  83.33%
```




```
BLEU-1          =  0.94444
BLEU-2          =  0.92956
BLEU-3          =  0.75522

BLEU-4          =  0.67880
ROUGE-1         =  0.91667
ROUGE-2         =  0.83333
ROUGE-L         =  0.91667

BERT Precision  =  0.95879
BERT Recall     =  0.95880
BERT F1 Score   =  0.95886
```

مدل T5 خام / k=3

```
Accuracy        =  0.00%
Macro F1-score  =  0.00%
Micro F1-score  =  0.00%
Macro Precision =  0.00%
Macro Recall    =  0.00%
```




```
BLEU-1          =  0.01389
BLEU-2          =  0.01026
BLEU-3          =  0.00978
BLEU-4          =  0.00949

ROUGE-1         =  0.02177
ROUGE-2         =  0.00000
ROUGE-L         =  0.02177

BERT Precision =  -0.37401
BERT Recall    =  -0.18195
BERT F1 Score  =  -0.27928
```