Teacher Model Training

Part 1: Training Setup

Step 1: Neccessary imports, connect to huggingface and mount drive

```
!pip install loralib
```

```
Collecting loralib
  Downloading loralib-0.1.2-py3-none-any.whl.metadata (15 kB)
Downloading loralib-0.1.2-py3-none-any.whl (10 kB)
Installing collected packages: loralib
Successfully installed loralib-0.1.2
```

```
import pandas as pd
import gc
import numpy as np
import os
```

```
from huggingface_hub import notebook_login
notebook_login()
```

```
from peft import LoraConfig, get_peft_model, PeftModel
import loralib as lora
import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
from transformers import AutoTokenizer, pipeline, AutoConfig, AutoModelForCausalLM, AutoModel, get_linear_schedule_w
from tqdm import tqdm
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Step 2: Read csv file prepared for training data from data_preprocessing notebook

```
df_task_1_train=pd.read_csv('/content/drive/MyDrive/df_task_1_train.csv', index_col=False)
```

```
df_task_1_train.head()
```

	Query Item Pair	soft_target
0	Query: !awnmower tires without rims Product Ti...	0.0
1	Query: !awnmower tires without rims Product Ti...	1.0
2	Query: !awnmower tires without rims Product Ti...	1.0
3	Query: !awnmower tires without rims Product Ti...	0.0
4	Query: # 10 self-seal envelopes without window...	1.0

```
print(df_task_1_train['Query Item Pair'].iloc[1])
```

```
Query: !awnmower tires without rims Product Title: (Set of 2) 15x6.00-6 Husqvarna/Poulan Tire Wheel Assy .75" Bearing
6x4.5 Wheel with 3/4" Precision bearings; Hub is 3" Long with .75" precision ball bearings. No grease required. Color
Husqvarna wheel number: 532106732 replaces 106732x645, 106732x643, 106732x417, 532141446, 532138336, 5321383-36 53212
ATW-001
Tire OD: 14.96; Tire SW: 6.3; PSI: 30; Max Load: 570 lbs. Product Brand: Antego Tire & Wheel Product Color: Husqvarna
```

Step 3: Create a LLamaRelevanceScore class that loads the pretrained model and defines a score head (the MLP layer mentioned in the paper) that calculates a logit score (probability that the item is relevant to the query). In the forward pass pass, we pass the last token (complete context) as input to the score head. We use BCEWithLogitsLoss as teacher model should learn a probability of relevance.

```
class LlamaRelevanceScore(nn.Module):
    def __init__(self, model_name, mlp_hidden_dim=1024):
        super().__init__()

        self.llama = AutoModel.from_pretrained(
            model_name,
            torch_dtype=torch.bfloat16
            )

        hidden_size = self.llama.config.hidden_size


        self.score_head = nn.Sequential(
            nn.Linear(hidden_size, mlp_hidden_dim),
            nn.ReLU(),
```

```
            nn.Linear(mlp_hidden_dim, 1)
        )

    def forward(self,
    input_ids=None,
    attention_mask=None,
    inputs_embeds=None,
    position_ids=None,
    past_key_values=None,
    labels=None,
                **kwargs):
        outputs = self.llama(
        input_ids=input_ids,
        attention_mask=attention_mask,
        inputs_embeds=inputs_embeds,
        position_ids=position_ids,
        past_key_values=past_key_values,
        **kwargs
        )

        last_token = outputs.last_hidden_state[:, -1, :]

        logits = self.score_head(last_token).squeeze(-1)

        loss = None
        if labels is not None:
            loss_fn = nn.BCEWithLogitsLoss()
            loss = loss_fn(logits, labels.float())

        return {
            "loss": loss,
            "logits": logits,
        }
```

Step 4: Get tokenizer from pretrained model, and build the model structure

```
model_id = "meta-llama/Llama-3.2-1B"

tokenizer = AutoTokenizer.from_pretrained(model_id)
tokenizer.pad_token = tokenizer.eos_token
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/token
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%  50.5k/50.5k [00:00<00:00, 5.97MB/s]
tokenizer.json: 100%  9.09M/9.09M [00:01<00:00, 8.34MB/s]
special_tokens_map.json: 100%  301/301 [00:00<00:00, 41.9kB/s]
```

```
model = LlamaRelevanceScore(model_name=model_id)
model.score_head = model.score_head.to(torch.bfloat16)
model.eval()
```

```
config.json: 100%  843/843 [00:00<00:00, 52.4kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors: 100%  2.47G/2.47G [00:06<00:00, 480MB/s]
LlamaRelevanceScore(
  (llama): LlamaModel(
    (embed_tokens): Embedding(128256, 2048)
    (layers): ModuleList(
      (0-15): 16 x LlamaDecoderLayer(
        (self_attn): LlamaAttention(
          (q_proj): Linear(in_features=2048, out_features=2048, bias=False)
          (k_proj): Linear(in_features=2048, out_features=512, bias=False)
          (v_proj): Linear(in_features=2048, out_features=512, bias=False)
          (o_proj): Linear(in_features=2048, out_features=2048, bias=False)
        )
        (mlp): LlamaMLP(
          (gate_proj): Linear(in_features=2048, out_features=8192, bias=False)
          (up_proj): Linear(in_features=2048, out_features=8192, bias=False)
          (down_proj): Linear(in_features=8192, out_features=2048, bias=False)
          (act_fn): SiLUActivation()
        )
        (input_layernorm): LlamaRMSNorm((2048,), eps=1e-05)
        (post_attention_layernorm): LlamaRMSNorm((2048,), eps=1e-05)
      )
    )
    (norm): LlamaRMSNorm((2048,), eps=1e-05)
    (rotary_emb): LlamaRotaryEmbedding()
  )
  (score_head): Sequential(
    (0): Linear(in_features=2048, out_features=1024, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1024, out_features=1, bias=True)
  )
)
```

Step 5: We configure the model to use LoRA as mentioned in the paper. We want to train only the q and v layer weights. We set the LoRA dropout to 0.05 as mentioned in the paper, however, we're using a LoRA rank of 128

instead of the 256 as mentioned in the paper.

```
def getLoraModel(model):
    lora_config = LoraConfig(
        r=128,
        target_modules=["q_proj", "v_proj"],
        lora_dropout=0.05,
        task_type="FEATURE_EXTRACTION",
        lora_alpha=64

    )

    model = get_peft_model(model, lora_config)
    lora.mark_only_lora_as_trainable(model)
    model.print_trainable_parameters()

    return model
```

```
model = getLoraModel(model)
```

```
trainable params: 13,631,488 || all params: 1,251,545,089 || trainable%: 1.0892
```

Step 6: We create a QueryItemDataset class, and use it to create an object that will be used to pass training data to our model. We pass our Query Item Pair column data through the model tokeniser. We have set the max length of tokens accepted as training input text to 512, so any additional tokens will be truncated, and padding will be added for shorter token sequences to make them of length 512. We use this encoded data to set input_ids and attention_mask for our model. We use the soft_target column from our training dataframe to define the labels for our model's training data. We use DataLoader to shuffle data and create batches of 43 samples.

```
class QueryItemDataset(Dataset):
  def __init__(self, df, query_col, target_col, tokenizer, max_length=512):
    self.texts=df[query_col].tolist()
    self.labels = df[target_col].tolist()
    self.tokenizer = tokenizer
    self.max_length = max_length

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

  def __getitem__(self, idx):
    text = self.texts[idx]
    label = self.labels[idx]

    encoding = self.tokenizer(
            text,
            max_length=self.max_length,
            padding="max_length",
            truncation=True,
            return_tensors="pt"
        )

    return {
            "input_ids": encoding["input_ids"].squeeze(0),
            "attention_mask": encoding["attention_mask"].squeeze(0),
            "labels": torch.tensor(label, dtype=torch.float),
        }
```

```
query_col='Query Item Pair'
target_col='soft_target'
dataset = QueryItemDataset(df_task_1_train, query_col, target_col, tokenizer)
dataloader = DataLoader(dataset, batch_size=43, shuffle=True)
```

Step 7: We define model hyperparameters

```
lr = 1e-5
num_epochs = 3
gradient_accumulation_steps = 1

optimizer = torch.optim.AdamW(
    model.parameters(),
    lr=lr,
    weight_decay=0.01
)

num_training_steps = num_epochs * len(dataloader)
num_warmup_steps = int(0.1 * num_training_steps)  # 10% warmup

scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=num_warmup_steps,
    num_training_steps=num_training_steps
)
```

Step 8: We define a directory to save our model at, as training progresses. We define a save_checkpoint that stores the model's LoRA adapter weights, score head state dictionary, optimiser and scheduler at this directory.

```
BASE_CKPT_DIR = "/content/drive/MyDrive/llama_relevance_checkpoints"
os.makedirs(BASE_CKPT_DIR, exist_ok=True)
```

```
save_every = max(1, int(0.2 * len(dataloader)))

def save_checkpoint(model, optimizer, scheduler, epoch, step):
    ckpt_dir = os.path.join(
        BASE_CKPT_DIR, f"epoch{epoch}_step{step}"
    )
    os.makedirs(ckpt_dir, exist_ok=True)


    model.save_pretrained(ckpt_dir)


    torch.save(
        model.score_head.state_dict(),
        os.path.join(ckpt_dir, "score_head.pt")
    )


    torch.save(
        {
            "optimizer": optimizer.state_dict(),
            "scheduler": scheduler.state_dict(),
            "epoch": epoch,
            "step": step,
        },
        os.path.join(ckpt_dir, "trainer_state.pt")
    )

    print(f"Saved checkpoint at {ckpt_dir}")
```

Part 2: Train Model for 3 epochs

Step 9: We set the device to be used for training as cuda. We train the model over 3 epochs. We checkpoint our model for every 20% of an epoch that is completed. We also compute avg loss for each epoch

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

```
cuda
```

```
torch.cuda.empty_cache()
gc.collect()
```

```
0
```

```
model.train()
model.to(device)
global_step = 0

for epoch in range(num_epochs):
    epoch_loss = 0.0
    progress_bar = tqdm(dataloader, desc=f"Epoch {epoch+1}")

    for step, batch in enumerate(progress_bar):
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["labels"].to(device)

        outputs = model(
            input_ids=input_ids,
            attention_mask=attention_mask,
            labels=labels
        )

        loss = outputs["loss"]
        loss = loss / gradient_accumulation_steps
        loss.backward()

        if (step + 1) % gradient_accumulation_steps == 0:
            optimizer.step()
            scheduler.step()
            optimizer.zero_grad()
            global_step += 1

        epoch_loss += loss.item()
        progress_bar.set_postfix(
            loss=loss.item(),
            lr=scheduler.get_last_lr()[0]
        )

        if (step + 1) % save_every == 0:
```

```
            save_checkpoint(
                model=model,
                optimizer=optimizer,
                scheduler=scheduler,
                epoch=epoch + 1,
                step=step + 1,
            )

    avg_epoch_loss = epoch_loss / len(dataloader)
    print(f"\nEpoch {epoch+1} finished | Avg loss: {avg_epoch_loss:.4f}\n")
```

```
Epoch 1:  20%|██        | 815/4078 [12:52<59:04,  1.09s/it, loss=0.666, lr=6.66e-6]Saved checkpoint at /content/drive
Epoch 1:  40%|████      | 1630/4078 [25:44<43:21,  1.06s/it, loss=0.649, lr=9.63e-6]Saved checkpoint at /content/driv
Epoch 1:  60%|██████    | 2445/4078 [38:35<28:51,  1.06s/it, loss=0.691, lr=8.89e-6]Saved checkpoint at /content/driv
Epoch 1:  80%|████████  | 3260/4078 [51:27<14:29,  1.06s/it, loss=0.644, lr=8.15e-6]Saved checkpoint at /content/driv
Epoch 1: 100%|██████████| 4075/4078 [1:04:18<00:03,  1.06s/it, loss=0.613, lr=7.41e-6]Saved checkpoint at /content/dr
Epoch 1: 100%|██████████| 4078/4078 [1:04:20<00:00,  1.06it/s, loss=0.6, lr=7.41e-6]

Epoch 1 finished | Avg loss: 0.6598

Epoch 2:  20%|██        | 815/4078 [12:51<57:55,  1.07s/it, loss=0.648, lr=6.67e-6]Saved checkpoint at /content/drive
Epoch 2:  40%|████      | 1630/4078 [25:42<43:19,  1.06s/it, loss=0.53, lr=5.93e-6]Saved checkpoint at /content/drive
Epoch 2:  60%|██████    | 2445/4078 [38:33<29:00,  1.07s/it, loss=0.582, lr=5.19e-6]Saved checkpoint at /content/driv
Epoch 2:  80%|████████  | 3260/4078 [51:24<14:35,  1.07s/it, loss=0.64, lr=4.45e-6]Saved checkpoint at /content/drive
Epoch 2: 100%|██████████| 4075/4078 [1:04:16<00:03,  1.06s/it, loss=0.536, lr=3.71e-6]Saved checkpoint at /content/dr
Epoch 2: 100%|██████████| 4078/4078 [1:04:18<00:00,  1.06it/s, loss=0.56, lr=3.7e-6]

Epoch 2 finished | Avg loss: 0.5941

Epoch 3:  20%|██        | 815/4078 [12:51<58:13,  1.07s/it, loss=0.616, lr=2.96e-6]Saved checkpoint at /content/drive
Epoch 3:  40%|████      | 1630/4078 [25:43<44:12,  1.08s/it, loss=0.568, lr=2.22e-6]Saved checkpoint at /content/driv
Epoch 3:  60%|██████    | 2445/4078 [38:35<29:14,  1.07s/it, loss=0.592, lr=1.48e-6]Saved checkpoint at /content/driv
Epoch 3:  80%|████████  | 3260/4078 [51:26<14:40,  1.08s/it, loss=0.524, lr=7.43e-7]Saved checkpoint at /content/driv
Epoch 3: 100%|██████████| 4075/4078 [1:04:18<00:03,  1.08s/it, loss=0.474, lr=2.72e-9]Saved checkpoint at /content/dr
Epoch 3: 100%|██████████| 4078/4078 [1:04:20<00:00,  1.06it/s, loss=0.716, lr=0]
Epoch 3 finished | Avg loss: 0.5760
```

Step 10: We save the epoch 3 model weights

```
save_checkpoint(
                model=model,
                optimizer=optimizer,
                scheduler=scheduler,
                epoch=epoch + 1,
                step=step + 1,
            )
```

```
Saved checkpoint at /content/drive/MyDrive/llama_relevance_checkpoints/epoch3_step4078
```

Part 3: Train Model for additional 4 epochs

Step 11: We load the epoch 3 model using our saved weights and set the LoRA configs to be able to train parameters in q and v layers along with score head layer

```
ckpt_path = f"{BASE_CKPT_DIR}/epoch3_step4078"

torch.cuda.empty_cache()
gc.collect()

model = LlamaRelevanceScore(model_name=model_id)
model.score_head = model.score_head.to(torch.bfloat16)

model = getLoraModel(model)

model = PeftModel.from_pretrained(model, ckpt_path, is_local=True, is_trainable=True)

model.score_head.load_state_dict(
    torch.load(os.path.join(ckpt_path, "score_head.pt"))
)


for p in model.score_head.parameters():
    p.requires_grad = True


model.to(device)
model.train()


optimizer = torch.optim.AdamW(
    model.parameters(),
    lr=lr,
    weight_decay=0.01
)

previous_epochs = 3
new_epochs = 4
total_epochs = previous_epochs + new_epochs

num_training_steps = total_epochs * len(dataloader)
num_warmup_steps = int(0.1 * num_training_steps)
```

```
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=num_warmup_steps,
    num_training_steps=num_training_steps
)
```

```
trainable params: 13,631,488 || all params: 1,251,545,089 || trainable%: 1.0892
```

```
for name, param in model.named_parameters():
    if "lora" in name:
        print(name, param.requires_grad)
```

```
base_model.model.base_model.model.llama.layers.1.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.1.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.2.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.2.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.2.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.2.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.3.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.3.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.3.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.3.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.4.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.4.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.4.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.4.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.5.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.5.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.5.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.5.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.6.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.6.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.6.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.6.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.7.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.7.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.7.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.7.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.8.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.8.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.8.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.8.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.9.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.9.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.9.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.9.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.10.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.10.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.10.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.10.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.11.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.11.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.11.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.11.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.12.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.12.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.12.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.12.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.13.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.13.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.13.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.13.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.14.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.14.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.14.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.14.self_attn.v_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.15.self_attn.q_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.15.self_attn.q_proj.lora_B.default.weight True
base_model.model.base_model.model.llama.layers.15.self_attn.v_proj.lora_A.default.weight True
base_model.model.base_model.model.llama.layers.15.self_attn.v_proj.lora_B.default.weight True
```

```
trainer_state = torch.load(
    os.path.join(ckpt_path, "trainer_state.pt"),
    map_location="cpu"
)

optimizer.load_state_dict(trainer_state["optimizer"])
scheduler.load_state_dict(trainer_state["scheduler"])

start_epoch = trainer_state["epoch"]
global_step = trainer_state["step"]
```

Step 12: We train the model for an additional 4 epochs

```
num_epochs = new_epochs
dataloader = DataLoader(dataset, batch_size=43, shuffle=True)

for epoch in range(start_epoch, start_epoch + num_epochs):
    epoch_loss = 0.0
    progress_bar = tqdm(dataloader, desc=f"Epoch {epoch+1}")

    for step, batch in enumerate(progress_bar):
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["labels"].to(device)

        outputs = model(
            input_ids=input_ids,
```

```
            attention_mask=attention_mask,
            labels=labels
        )

        loss = outputs["loss"] / gradient_accumulation_steps
        loss.backward()

        if (step + 1) % gradient_accumulation_steps == 0:
            optimizer.step()
            scheduler.step()
            optimizer.zero_grad()
            global_step += 1

        epoch_loss += loss.item()
        progress_bar.set_postfix(
            loss=loss.item(),
            lr=scheduler.get_last_lr()[0]
        )

        if (step + 1) % save_every == 0:
            save_checkpoint(
                model=model,
                optimizer=optimizer,
                scheduler=scheduler,
                epoch=epoch + 1,
                step=step + 1,
            )

    print(f"\nEpoch {epoch+1} finished | Avg loss: {epoch_loss / len(dataloader):.4f}\n")
```

```
Epoch 4:  20%|██         | 815/4078 [12:55<58:19,  1.07s/it, loss=0.647, lr=6.03e-6]Saved checkpoint at /content/drive
Epoch 4:  40%|████       | 1630/4078 [25:49<43:36,  1.07s/it, loss=0.558, lr=5.71e-6]Saved checkpoint at /content/driv
Epoch 4:  60%|██████     | 2445/4078 [38:43<29:03,  1.07s/it, loss=0.67, lr=5.4e-6]Saved checkpoint at /content/drive/
Epoch 4:  80%|████████   | 3260/4078 [51:37<14:37,  1.07s/it, loss=0.644, lr=5.08e-6]Saved checkpoint at /content/driv
Epoch 4: 100%|██████████| 4075/4078 [1:04:32<00:03,  1.07s/it, loss=0.67, lr=4.76e-6]Saved checkpoint at /content/dri
Epoch 4: 100%|██████████| 4078/4078 [1:04:34<00:00,  1.05it/s, loss=0.62, lr=4.76e-6]

Epoch 4 finished | Avg loss: 0.6644

Epoch 5:  20%|██         | 815/4078 [12:54<58:36,  1.08s/it, loss=0.603, lr=4.44e-6]Saved checkpoint at /content/drive
Epoch 5:  40%|████       | 1630/4078 [25:48<43:41,  1.07s/it, loss=0.639, lr=4.13e-6]Saved checkpoint at /content/driv
Epoch 5:  60%|██████     | 2445/4078 [38:42<29:10,  1.07s/it, loss=0.654, lr=3.81e-6]Saved checkpoint at /content/driv
Epoch 5:  80%|████████   | 3260/4078 [51:36<14:36,  1.07s/it, loss=0.66, lr=3.49e-6]Saved checkpoint at /content/drive
Epoch 5: 100%|██████████| 4075/4078 [1:04:31<00:03,  1.07s/it, loss=0.645, lr=3.18e-6]Saved checkpoint at /content/dr
Epoch 5: 100%|██████████| 4078/4078 [1:04:33<00:00,  1.05it/s, loss=0.734, lr=3.17e-6]

Epoch 5 finished | Avg loss: 0.6597

Epoch 6:  20%|██         | 815/4078 [12:54<58:12,  1.07s/it, loss=0.676, lr=2.86e-6]Saved checkpoint at /content/drive
Epoch 6:  40%|████       | 1630/4078 [25:48<44:20,  1.09s/it, loss=0.699, lr=2.54e-6]Saved checkpoint at /content/driv
Epoch 6:  60%|██████     | 2445/4078 [38:43<29:11,  1.07s/it, loss=0.649, lr=2.22e-6]Saved checkpoint at /content/driv
Epoch 6:  80%|████████   | 3260/4078 [51:37<14:42,  1.08s/it, loss=0.592, lr=1.91e-6]Saved checkpoint at /content/driv
Epoch 6: 100%|██████████| 4075/4078 [1:04:31<00:03,  1.08s/it, loss=0.623, lr=1.59e-6]Saved checkpoint at /content/dr
Epoch 6: 100%|██████████| 4078/4078 [1:04:34<00:00,  1.05it/s, loss=0.538, lr=1.59e-6]

Epoch 6 finished | Avg loss: 0.6423

Epoch 7:  20%|██         | 815/4078 [12:54<59:57,  1.10s/it, loss=0.529, lr=1.27e-6]Saved checkpoint at /content/drive
Epoch 7:  40%|████       | 1630/4078 [25:48<43:46,  1.07s/it, loss=0.608, lr=9.53e-7]Saved checkpoint at /content/driv
Epoch 7:  60%|██████     | 2445/4078 [38:42<29:14,  1.07s/it, loss=0.678, lr=6.36e-7]Saved checkpoint at /content/driv
Epoch 7:  80%|████████   | 3260/4078 [51:37<14:41,  1.08s/it, loss=0.615, lr=3.18e-7]Saved checkpoint at /content/driv
Epoch 7: 100%|██████████| 4075/4078 [1:04:31<00:03,  1.07s/it, loss=0.667, lr=1.17e-9]Saved checkpoint at /content/dr
Epoch 7: 100%|██████████| 4078/4078 [1:04:33<00:00,  1.05it/s, loss=0.622, lr=0]
Epoch 7 finished | Avg loss: 0.6076
```

Step 13: We save the weights for epoch 7 model

```
save_checkpoint(
                model=model,
                optimizer=optimizer,
                scheduler=scheduler,
                epoch=epoch + 1,
                step=step + 1,
            )
```

```
Saved checkpoint at /content/drive/MyDrive/llama_relevance_checkpoints/epoch7_step4078
```

Part 3: Calculate MSE for Models

Step 14: We define a function to load the checkpointed model

```
def getModelFromCheckpoint(modelDirPath):
  ckpt_path = f"{BASE_CKPT_DIR}/"+modelDirPath

  torch.cuda.empty_cache()
  gc.collect()

  cpmodel = LlamaRelevanceScore(model_name=model_id)
  cpmodel.score_head = model.score_head.to(torch.bfloat16)

  cpmodel = getLoraModel(model)

  cpmodel = PeftModel.from_pretrained(model, ckpt_path, is_local=True)
```

```
  cpmodel.score_head.load_state_dict(
      torch.load(os.path.join(ckpt_path, "score_head.pt"))
  )


  for p in cpmodel.score_head.parameters():
      p.requires_grad = True


  cpmodel.to(device)
  cpmodel.eval()


  return cpmodel
```

Step 15: We define a Compute MSE function for the teacher model. A sigmoid is applied to the model's logit output to get values between 0 and 1 and they are compared against the actual soft label.

```
def compute_teacher_mse(teacher_model, eval_dataloader, eval_device, max_batches):
    teacher_model.eval()
    total_sq_error = 0.0
    total_count = 0

    with torch.no_grad():
        for i, batch in enumerate(eval_dataloader):
            if i >= max_batches:
                break

            input_ids = batch["input_ids"].to(eval_device)
            attention_mask = batch["attention_mask"].to(eval_device)
            labels = batch["labels"].to(eval_device).float()

            outputs = teacher_model(
                input_ids=input_ids,
                attention_mask=attention_mask
            )

            probs = torch.sigmoid(outputs["logits"])

            total_sq_error += ((probs - labels) ** 2).sum().item()
            total_count += labels.numel()

    return total_sq_error / total_count
```

Step 16: Calculate MSE for the epoch 3 model over training data (1720 samples) and test data (31003 samples)

```
model = getModelFromCheckpoint("epoch3_step4078")
```

```
/usr/local/lib/python3.12/dist-packages/peft/mapping_func.py:72: UserWarning: You are trying to modify a model with P
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/peft/tuners/tuners_utils.py:282: UserWarning: Already found a `peft_config` a
  warnings.warn(
trainable params: 13,631,488 || all params: 1,251,545,089 || trainable%: 1.0892
/usr/local/lib/python3.12/dist-packages/peft/peft_model.py:598: UserWarning: Found missing adapter keys while loading
  warnings.warn(warn_message)
```

```
mse = compute_teacher_mse(model, dataloader, device, max_batches=40)
print(f"Teacher MSE: {mse:.6f}")
```

```
Teacher MSE: 0.179435
```

```
df_task_1_test=pd.read_csv('/content/drive/MyDrive/df_task_1_test.csv', index_col=False)
```

```
print(len(df_task_1_test))
```

```
31020
```

```
query_col='Query Item Pair'
target_col='soft_target'
test_dataset = QueryItemDataset(df_task_1_test, query_col, target_col, tokenizer)
test_dataloader = DataLoader(test_dataset, batch_size=43, shuffle=True)
```

```
test_mse = compute_teacher_mse(model, test_dataloader, device, max_batches=721)
print(f"Teacher MSE (test): {test_mse:.6f}")
```

```
Teacher MSE (test): 0.174628
```

Step 17: Calculate MSE for the epoch 7 model over training data (1720 samples) and test data (31003 samples)

```
model = getModelFromCheckpoint("epoch7_step4078")
```

```
/usr/local/lib/python3.12/dist-packages/peft/mapping_func.py:72: UserWarning: You are trying to modify a model with P
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/peft/tuners/tuners_utils.py:282: UserWarning: Already found a `peft_config` a
  warnings.warn(
trainable params: 13,631,488 || all params: 1,251,545,089 || trainable%: 1.0892
/usr/local/lib/python3.12/dist-packages/peft/peft_model.py:598: UserWarning: Found missing adapter keys while loading
  warnings.warn(warn_message)
```

```
```

```
mse = compute_teacher_mse(model, dataloader, device, max_batches=40)
print(f"Teacher MSE: {mse:.6f}")
```

```
Teacher MSE: 0.151584
```

```
test_mse = compute_teacher_mse(model, test_dataloader, device, max_batches=721)
print(f"Teacher MSE (test): {test_mse:.6f}")
```

```
Teacher MSE (test): 0.155861
```