QnA_Chatbot_BERT

October 21, 2024

0.1 Set up and Dependencies

0.1.1 Install Dependencies

[24]:
```
#!pip install transformers faiss-cpu gradio datasets evaluate seaborn matplotlib
```

0.1.2 Mount Google Drive

[25]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

0.2 Loading the SQuAD Dataset

[37]:
```
import collections
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm.auto import tqdm
from collections import defaultdict
import pandas as pd
import json
import os

import evaluate  # Use the evaluate library instead of load_metric
from datasets import load_dataset
from transformers import (
    AutoTokenizer,
    AutoModelForQuestionAnswering,
    Trainer,
    TrainingArguments,
    default_data_collator,
    EvalPrediction,
)

# Ensure that matplotlib does not try to open any window
```

```python
import matplotlib
matplotlib.use('Agg')  # Use a non-interactive backend


# Directories for saving artifacts
artifact_dir = "/content/drive/MyDrive/AAI-520/QnA_Chatbot/artifacts"
os.makedirs(f'{artifact_dir}/plots', exist_ok=True)
os.makedirs(f'{artifact_dir}/results', exist_ok=True)
os.makedirs(f'{artifact_dir}/logs', exist_ok=True)
os.makedirs(f'{artifact_dir}/models', exist_ok=True)

# Load the SQuAD v2.0 dataset
squad_v2 = load_dataset('squad_v2')

# Limit the dataset for quicker training (adjust as needed)
train_dataset = squad_v2['train'].select(range(10000))
validation_dataset = squad_v2['validation'].select(range(2000))

print("Sample Training Example:")
print(train_dataset[0])
print("\nSample Validation Example:")
print(validation_dataset[0])

def prepare_train_features(examples, tokenizer, max_length=384, doc_stride=128):
    # Tokenize the inputs with truncation and padding, but keep the overflows
    tokenized_examples = tokenizer(
        examples['question'],
        examples['context'],
        truncation='only_second',
        max_length=max_length,
        stride=doc_stride,
        return_overflowing_tokens=True,
        return_offsets_mapping=True,
        padding='max_length',
        # Removed: clean_up_tokenization_spaces=True
    )

    # Since one example might give multiple features, we need a mapping
    sample_mapping = tokenized_examples.pop("overflow_to_sample_mapping")
    offset_mapping = tokenized_examples.pop("offset_mapping")

    # Mapping example indices to their features
    tokenized_examples["start_positions"] = []
    tokenized_examples["end_positions"] = []

    for i, offsets in enumerate(offset_mapping):
        # Map feature to its example
```

```
        input_ids = tokenized_examples['input_ids'][i]
        cls_index = input_ids.index(tokenizer.cls_token_id)

        sequence_ids = tokenized_examples.sequence_ids(i)

        sample_index = sample_mapping[i]
        answers = examples["answers"][sample_index]

        # If no answers are given, set start and end positions to cls_index
        if len(answers["answer_start"]) == 0:
            tokenized_examples["start_positions"].append(cls_index)
            tokenized_examples["end_positions"].append(cls_index)
        else:
            # Start/end character index of the answer in the text
            start_char = answers["answer_start"][0]
            end_char = start_char + len(answers["text"][0])

            # Start token index of the current span in the text
            token_start_index = 0
            while sequence_ids[token_start_index] != 1:
                token_start_index += 1

            # End token index of the current span in the text
            token_end_index = len(input_ids) - 1
            while sequence_ids[token_end_index] != 1:
                token_end_index -= 1

            # Detect if the answer is out of the span
            if not (offsets[token_start_index][0] <= start_char and␣
↪offsets[token_end_index][1] >= end_char):
                tokenized_examples["start_positions"].append(cls_index)
                tokenized_examples["end_positions"].append(cls_index)
            else:
                # Move the token_start_index and token_end_index to the start␣
↪and end of the answer
                while token_start_index < len(offsets) and␣
↪offsets[token_start_index][0] <= start_char:
                    token_start_index += 1
                tokenized_examples["start_positions"].append(token_start_index␣
↪- 1)

                while offsets[token_end_index][1] >= end_char:
                    token_end_index -= 1
                tokenized_examples["end_positions"].append(token_end_index + 1)

    return tokenized_examples
```

```python
def prepare_validation_features(examples, tokenizer, max_length=384,␣
↪doc_stride=128):
    tokenized_examples = tokenizer(
        examples['question'],
        examples['context'],
        truncation='only_second',
        max_length=max_length,
        stride=doc_stride,
        return_overflowing_tokens=True,
        return_offsets_mapping=True,
        padding='max_length',
        # Removed: clean_up_tokenization_spaces=True
    )

    sample_mapping = tokenized_examples.pop("overflow_to_sample_mapping")
    tokenized_examples["example_id"] = []

    for i in range(len(tokenized_examples["input_ids"])):
        # Map feature to its example
        sequence_ids = tokenized_examples.sequence_ids(i)
        sample_index = sample_mapping[i]
        tokenized_examples["example_id"].append(examples["id"][sample_index])

        # Set to None the offset_mapping that are not part of the context
        tokenized_examples["offset_mapping"][i] = [
            (o if sequence_ids[k] == 1 else None)
            for k, o in enumerate(tokenized_examples["offset_mapping"][i])
        ]

    return tokenized_examples



def postprocess_qa_predictions(
    examples, features, raw_predictions, tokenizer, n_best_size=20,␣
↪max_answer_length=30
):
    all_start_logits, all_end_logits = raw_predictions
    print(f"Number of start logits: {len(all_start_logits)}")
    print(f"Number of end logits: {len(all_end_logits)}")
    print(f"Number of features: {len(features)}")
    # Proceed only if they match
    if len(all_start_logits) != len(features) or len(all_end_logits) !=␣
↪len(features):
        print("Mismatch between number of predictions and features.")
        return {}
```

```
    example_id_to_index = {k: i for i, k in enumerate(examples["id"])}
    features_per_example = defaultdict(list)
    for i, feature in enumerate(features):
        features_per_example[feature["example_id"]].append(i)

    predictions = collections.OrderedDict()

    print("Post-processing predictions...")
    for example_index, example_id in enumerate(tqdm(examples["id"])):
        example = examples[example_index]
        feature_indices = features_per_example[example_id]

        min_null_score = None  # To track the minimum null score
        valid_answers = []

        context = example["context"]
        for feature_index in feature_indices:
            # Check if feature_index is within the bounds of all_start_logits␣
↪and all_end_logits
            if feature_index >= len(all_start_logits) or feature_index >=␣
↪len(all_end_logits):
                print(f"Feature index {feature_index} out of range for␣
↪predictions.")
                continue

            start_logits = all_start_logits[feature_index]
            end_logits = all_end_logits[feature_index]
            offset_mapping = features[feature_index]["offset_mapping"]
            input_ids = features[feature_index]["input_ids"]
            cls_index = input_ids.index(tokenizer.cls_token_id)

            # The score for the null answer (no answer)
            feature_null_score = start_logits[cls_index] + end_logits[cls_index]
            if min_null_score is None or min_null_score > feature_null_score:
                min_null_score = feature_null_score

            # Go through all possible combinations of start and end logits
            start_indexes = np.argsort(start_logits)[-1 : -n_best_size - 1 :␣
↪-1].tolist()
            end_indexes = np.argsort(end_logits)[-1 : -n_best_size - 1 : -1].
↪tolist()
            for start_index in start_indexes:
                for end_index in end_indexes:
                    # Skip invalid positions
                    if (
                        start_index >= len(offset_mapping)
                        or end_index >= len(offset_mapping)
```

```
                        or offset_mapping[start_index] is None
                        or offset_mapping[end_index] is None
                        or end_index < start_index
                        or end_index - start_index + 1 > max_answer_length
                    ):
                        continue
                    start_char = offset_mapping[start_index][0]
                    end_char = offset_mapping[end_index][1]
                    answer_text = context[start_char:end_char]
                    valid_answers.append(
                        {
                            "score": start_logits[start_index] +␣
↪end_logits[end_index],
                            "text": answer_text,
                        }
                    )
        if valid_answers:
            best_answer = max(valid_answers, key=lambda x: x["score"])
        else:
            # If no valid answers, predict empty string
            best_answer = {"text": "", "score": 0.0}

        predictions[example_id] = best_answer["text"]

    return predictions

def plot_loss(history, model_name):
    # Convert history to a DataFrame for easier manipulation
    df = pd.DataFrame(history)

    # Check if 'epoch' column exists
    if 'epoch' not in df.columns:
        print(f"No epoch information found in the log history for {model_name}.␣
↪Skipping loss plot.")
        return

    # Extract training loss and group by epoch to compute average loss per epoch
    train_loss_df = df[df['loss'].notna()].groupby('epoch')['loss'].mean().
↪reset_index()

    # Extract evaluation loss (will be empty since we aren't computing it)
    eval_loss_df = pd.DataFrame()

    # Ensure there is data to plot
    if train_loss_df.empty and eval_loss_df.empty:
        print(f"No loss data found for {model_name}.")
        return
```

```python
    # Plotting
    plt.figure(figsize=(10, 5))
    if not train_loss_df.empty:
        plt.plot(train_loss_df['epoch'], train_loss_df['loss'], label='Training Loss', marker='o')
    if not eval_loss_df.empty:
        plt.plot(eval_loss_df['epoch'], eval_loss_df['eval_loss'], label='Validation Loss', marker='o')
    plt.title(f'Training Loss for {model_name}')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.savefig(f'{artifact_dir}/plots/{model_name}_loss.png')
    plt.close()

def plot_metrics(metrics, model_name):
    em = metrics.get('exact', 0)
    f1 = metrics.get('f1', 0)

    plt.figure(figsize=(6, 4))
    # Removed palette to avoid FutureWarning
    sns.barplot(x=['Exact Match', 'F1 Score'], y=[em, f1])
    plt.ylim(0, 100)
    plt.title(f'Performance Metrics for {model_name}')
    for index, value in enumerate([em, f1]):
        plt.text(index, value + 1, f"{value:.2f}", ha='center', fontsize=12)
    plt.savefig(f'{artifact_dir}/plots/{model_name}_metrics.png')
    plt.close()

def plot_comparative_metrics(all_metrics):
    models = list(all_metrics.keys())
    em_scores = [all_metrics[model].get('exact', 0) for model in models]
    f1_scores = [all_metrics[model].get('f1', 0) for model in models]

    x = np.arange(len(models))  # label locations
    width = 0.35  # width of the bars

    plt.figure(figsize=(14, 7))
    plt.bar(x - width/2, em_scores, width, label='Exact Match', color='skyblue')
    plt.bar(x + width/2, f1_scores, width, label='F1 Score', color='salmon')

    plt.ylabel('Scores')
    plt.title('Comparison of EM and F1 Scores Across Models')
    plt.xticks(x, models, rotation=45, ha='right')
    plt.ylim(0, 100)
```

```python
    plt.legend()
    plt.tight_layout()
    plt.savefig(f'{artifact_dir}/plots/comparative_metrics.png')
    plt.close()

# Load the metric once globally to avoid reloading
metric = evaluate.load('squad_v2')

def train_and_evaluate(model_name):
    print(f"\nTraining and evaluating model: {model_name}")

    # Load tokenizer and model
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForQuestionAnswering.from_pretrained(model_name)

    # Prepare training features
    tokenized_train = train_dataset.map(
        lambda x: prepare_train_features(x, tokenizer),
        batched=True,
        remove_columns=train_dataset.column_names,
    )

    # Prepare validation features
    tokenized_validation = validation_dataset.map(
        lambda x: prepare_validation_features(x, tokenizer),
        batched=True,
        remove_columns=validation_dataset.column_names,
    )

    def compute_metrics_fn(p: EvalPrediction):
      # Postprocess the predictions
      predictions = postprocess_qa_predictions(
          validation_dataset,
          tokenized_validation,
          (p.predictions[0], p.predictions[1]),
          tokenizer
      )

      # Format predictions for metric computation
      formatted_predictions = [
          {"id": k, "prediction_text": v, "no_answer_probability": 0.0}
          for k, v in predictions.items()
      ]
      references = [{"id": ex["id"], "answers": ex["answers"]} for ex in␣
 ↪validation_dataset]

      # Compute metrics using the evaluate library
```

```python
        metrics = metric.compute(predictions=formatted_predictions,␣
↪references=references)


        # Return the metrics with correct keys
        return {
            "exact_match": metrics["exact_match"],
            "f1": metrics["f1"],
            "eval_loss": metrics["eval_loss"] if metrics["eval_loss"] is not None␣
↪else None,
        }

    # Define training arguments without 'compute_metrics_fn'
    training_args = TrainingArguments(
        output_dir=f'{artifact_dir}/results/{model_name}',
        eval_strategy='epoch',
        save_strategy='epoch',
        learning_rate=2e-5,
        per_device_train_batch_size=2,
        per_device_eval_batch_size=2,
        num_train_epochs=3,
        weight_decay=0.01,
        logging_dir=f'./logs/{model_name}',
        logging_steps=10,
        save_total_limit=1,
        push_to_hub=False,
        report_to="none",
        load_best_model_at_end=False,    # Disabled automatic best model loading
        # Removed: metric_for_best_model
    )

    # Initialize the Trainer without 'compute_metrics_fn'
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_train,
        eval_dataset=tokenized_validation,
        tokenizer=tokenizer,
        data_collator=default_data_collator,
        compute_metrics=compute_metrics_fn
    )

    # Train the model
    trainer.train()

    eval_results = trainer.evaluate()
    if 'eval_loss' in eval_results:
```

```
        print(f"Validation Loss: {eval_results['eval_loss']}")
        trainer.log({'eval_loss': eval_results['eval_loss']})
    else:
        print("Validation Loss not captured.")


    # Plot training loss
    plot_loss(trainer.state.log_history, model_name)

    # Print Trainer's Log History for debugging
    print("Trainer's Log History:")
    print(pd.DataFrame(trainer.state.log_history))

    # Obtain raw predictions using trainer.predict()
    prediction_output = trainer.predict(tokenized_validation)
    raw_predictions = prediction_output.predictions
    # Note: 'prediction_output.metrics' will not contain 'exact_match' and 'f1'␣
↪since 'compute_metrics_fn' is not set

    print("Processing predictions and computing metrics...")

    # Postprocess predictions
    predictions_text = postprocess_qa_predictions(
        validation_dataset,
        tokenized_validation,
        raw_predictions,
        tokenizer
    )

    # Compute metrics manually
    formatted_predictions = [
        {"id": k, "prediction_text": v, "no_answer_probability": 0.0}
        for k, v in predictions_text.items()
    ]
    references = [{"id": ex["id"], "answers": ex["answers"]} for ex in␣
↪validation_dataset]

    metrics = metric.compute(predictions=formatted_predictions,␣
↪references=references)

    print("Evaluation Metrics:")
    for key, value in metrics.items():
        print(f"{key}: {value:.2f}")

    # Plot metrics (EM and F1)
    plot_metrics(metrics, model_name)
```

```
    # Interpretability: show a few predictions
    for i in range(3):
        example = validation_dataset[i]
        prediction = predictions_text.get(example['id'], "")
        print(f"\nQuestion: {example['question']}")
        true_answers = example['answers']['text']
        print(f"True Answer: {', '.join(true_answers) if true_answers else 'No␣
↪Answer'}")
        print(f"Predicted Answer: {prediction if prediction else 'No Answer'}")

    # Manually save the model if desired
    trainer.save_model(f'{artifact_dir}/models/{model_name}_final')

    return metrics

# List of models to train and evaluate

models = [
    'bert-base-uncased',
    'roberta-base',
    'albert-base-v2'
]

results = {}

for model_name in models:
    metrics = train_and_evaluate(model_name)
    results[model_name] = metrics

# Plot comparative metrics across all models
plot_comparative_metrics(results)

# Save the results to a file for future reference
with open(f'{artifact_dir}/results/model_performance.json', 'w') as f:
    json.dump(results, f, indent=4)

print("\nAll plots have been saved in the 'artifacts/plots' directory.")
print("Comparative metrics plot saved as 'artifacts/comparative_metrics.png'.")
print("Model performance metrics saved in 'artifacts/results/model_performance.
↪json'.")
```

```
Sample Training Example:
{'id': '56be85543aeaaa14008c9063', 'title': 'Beyoncé', 'context': 'Beyoncé
Giselle Knowles-Carter (/bi j nse / bee-YON-say) (born September 4, 1981) is an
American singer, songwriter, record producer and actress. Born and raised in
Houston, Texas, she performed in various singing and dancing competitions as a
child, and rose to fame in the late 1990s as lead singer of R&B girl-group
```

```
Destiny\'s Child. Managed by her father, Mathew Knowles, the group became one of
the world\'s best-selling girl groups of all time. Their hiatus saw the release
of Beyoncé\'s debut album, Dangerously in Love (2003), which established her as
a solo artist worldwide, earned five Grammy Awards and featured the Billboard
Hot 100 number-one singles "Crazy in Love" and "Baby Boy".', 'question': 'When
did Beyonce start becoming popular?', 'answers': {'text': ['in the late 1990s'],
'answer_start': [269]}}

Sample Validation Example:
{'id': '56ddde6b9a695914005b9628', 'title': 'Normans', 'context': 'The Normans
(Norman: Nourmands; French: Normands; Latin: Normanni) were the people who in
the 10th and 11th centuries gave their name to Normandy, a region in France.
They were descended from Norse ("Norman" comes from "Norseman") raiders and
pirates from Denmark, Iceland and Norway who, under their leader Rollo, agreed
to swear fealty to King Charles III of West Francia. Through generations of
assimilation and mixing with the native Frankish and Roman-Gaulish populations,
their descendants would gradually merge with the Carolingian-based cultures of
West Francia. The distinct cultural and ethnic identity of the Normans emerged
initially in the first half of the 10th century, and it continued to evolve over
the succeeding centuries.', 'question': 'In what country is Normandy located?',
'answers': {'text': ['France', 'France', 'France', 'France'], 'answer_start':
[159, 159, 159, 159]}}

Training and evaluating model: bert-base-uncased

Some weights of BertForQuestionAnswering were not initialized from the model
checkpoint at bert-base-uncased and are newly initialized: ['qa_outputs.bias',
'qa_outputs.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Map:   0%|          | 0/10000 [00:00<?, ? examples/s]

Map:   0%|          | 0/2000 [00:00<?, ? examples/s]

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Validation Loss not captured.
Trainer's Log History:
        loss   grad_norm  learning_rate     epoch   step  eval_runtime  \
0     5.8651   11.766853   1.998685e-05  0.001973     10           NaN
1     5.4760   14.465985   1.997370e-05  0.003946     20           NaN
2     5.1531   14.777692   1.996054e-05  0.005918     30           NaN
3     5.2820   43.271248   1.994739e-05  0.007891     40           NaN
4     4.8353   16.344652   1.993424e-05  0.009864     50           NaN
…        …           …              …         …      …             …
1520  1.2271  263.654297   2.235812e-08  2.996646  15190           NaN
1521  0.5903    6.742366   9.206287e-09  2.998619  15200           NaN
1522     NaN         NaN            NaN  3.000000  15207       47.9352
```

```
1523     NaN         NaN           NaN  3.000000  15207           NaN
1524     NaN         NaN           NaN  3.000000  15207       47.5978

      eval_samples_per_second  eval_steps_per_second  train_runtime  \
0                         NaN                    NaN            NaN
1                         NaN                    NaN            NaN
2                         NaN                    NaN            NaN
3                         NaN                    NaN            NaN
4                         NaN                    NaN            NaN
…                       …                    …            …
1520                      NaN                    NaN            NaN
1521                      NaN                    NaN            NaN
1522                   41.869                 20.945            NaN
1523                      NaN                    NaN       2933.506
1524                   42.166                 21.093            NaN

      train_samples_per_second  train_steps_per_second    total_flos  \
0                          NaN                     NaN           NaN
1                          NaN                     NaN           NaN
2                          NaN                     NaN           NaN
3                          NaN                     NaN           NaN
4                          NaN                     NaN           NaN
…                        …                     …           …
1520                       NaN                     NaN           NaN
1521                       NaN                     NaN           NaN
1522                       NaN                     NaN           NaN
1523                    10.367                   5.184  5.959722e+15
1524                       NaN                     NaN           NaN

      train_loss
0            NaN
1            NaN
2            NaN
3            NaN
4            NaN
…          …
1520         NaN
1521         NaN
1522         NaN
1523    1.135215
1524         NaN

[1525 rows x 13 columns]
Processing predictions and computing metrics…
Number of start logits: 2007
Number of end logits: 2007
Number of features: 2007
Post-processing predictions…
```

```
  0%|          | 0/2000 [00:00<?, ?it/s]

Evaluation Metrics:
exact: 32.80
f1: 37.83
total: 2000.00
HasAns_exact: 65.40
HasAns_f1: 75.43
HasAns_total: 1003.00
NoAns_exact: 0.00
NoAns_f1: 0.00
NoAns_total: 997.00
best_exact: 50.10
best_exact_thresh: 0.00
best_f1: 50.10
best_f1_thresh: 0.00

Question: In what country is Normandy located?
True Answer: France, France, France, France
Predicted Answer: France

Question: When were the Normans in Normandy?
True Answer: 10th and 11th centuries, in the 10th and 11th centuries, 10th and
11th centuries, 10th and 11th centuries
Predicted Answer: 10th and 11th centuries

Question: From which countries did the Norse originate?
True Answer: Denmark, Iceland and Norway, Denmark, Iceland and Norway, Denmark,
Iceland and Norway, Denmark, Iceland and Norway
Predicted Answer: Denmark, Iceland and Norway

Training and evaluating model: roberta-base

Some weights of RobertaForQuestionAnswering were not initialized from the model
checkpoint at roberta-base and are newly initialized: ['qa_outputs.bias',
'qa_outputs.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Map:   0%|          | 0/10000 [00:00<?, ? examples/s]

Map:   0%|          | 0/2000 [00:00<?, ? examples/s]

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Validation Loss not captured.
Trainer's Log History:
        loss  grad_norm  learning_rate     epoch   step  eval_runtime  \
0     5.8863  10.273878   1.998683e-05  0.001976     10           NaN
```

```
1     5.5009  12.485327   1.997365e-05  0.003952     20           NaN
2     4.6940  15.171370   1.996048e-05  0.005928     30           NaN
3     4.3942  41.202702   1.994731e-05  0.007904     40           NaN
4     4.2809  48.811317   1.993414e-05  0.009879     50           NaN
…      …        …             …          …     …        …
1518  0.4846  18.280777   1.712442e-08  2.997431  15170           NaN
1519  0.8913   0.034381   3.951788e-09  2.999407  15180           NaN
1520     NaN        NaN            NaN  3.000000  15183       44.7196
1521     NaN        NaN            NaN  3.000000  15183           NaN
1522     NaN        NaN            NaN  3.000000  15183       44.6226

      eval_samples_per_second  eval_steps_per_second  train_runtime  \
0                         NaN                    NaN            NaN
1                         NaN                    NaN            NaN
2                         NaN                    NaN            NaN
3                         NaN                    NaN            NaN
4                         NaN                    NaN            NaN
…                       …                      …             …
1518                      NaN                    NaN            NaN
1519                      NaN                    NaN            NaN
1520                   44.880                 22.451            NaN
1521                      NaN                    NaN      3011.0374
1522                   44.977                 22.500            NaN

      train_samples_per_second  train_steps_per_second    total_flos  \
0                          NaN                     NaN           NaN
1                          NaN                     NaN           NaN
2                          NaN                     NaN           NaN
3                          NaN                     NaN           NaN
4                          NaN                     NaN           NaN
…                        …                       …           …
1518                       NaN                     NaN           NaN
1519                       NaN                     NaN           NaN
1520                       NaN                     NaN           NaN
1521                    10.084                   5.042  5.950315e+15
1522                       NaN                     NaN           NaN

      train_loss
0            NaN
1            NaN
2            NaN
3            NaN
4            NaN
…          …
1518         NaN
1519         NaN
1520         NaN
1521    1.087086
```

```
1522         NaN

[1523 rows x 13 columns]
Processing predictions and computing metrics…
Number of start logits: 2007
Number of end logits: 2007
Number of features: 2007
Post-processing predictions…

  0%|          | 0/2000 [00:00<?, ?it/s]

Evaluation Metrics:
exact: 37.95
f1: 42.30
total: 2000.00
HasAns_exact: 75.47
HasAns_f1: 84.14
HasAns_total: 1003.00
NoAns_exact: 0.20
NoAns_f1: 0.20
NoAns_total: 997.00
best_exact: 50.10
best_exact_thresh: 0.00
best_f1: 50.10
best_f1_thresh: 0.00

Question: In what country is Normandy located?
True Answer: France, France, France, France
Predicted Answer: France

Question: When were the Normans in Normandy?
True Answer: 10th and 11th centuries, in the 10th and 11th centuries, 10th and
11th centuries, 10th and 11th centuries
Predicted Answer: 10th and 11th centuries

Question: From which countries did the Norse originate?
True Answer: Denmark, Iceland and Norway, Denmark, Iceland and Norway, Denmark,
Iceland and Norway, Denmark, Iceland and Norway
Predicted Answer: Denmark, Iceland and Norway

Training and evaluating model: albert-base-v2

Some weights of AlbertForQuestionAnswering were not initialized from the model
checkpoint at albert-base-v2 and are newly initialized: ['qa_outputs.bias',
'qa_outputs.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Map:   0%|          | 0/10000 [00:00<?, ? examples/s]
```

```
Map:   0%|          | 0/2000 [00:00<?, ? examples/s]

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Validation Loss not captured.
Trainer's Log History:
        loss   grad_norm  learning_rate     epoch   step  eval_runtime  \
0     5.4562  110.185028   1.998687e-05  0.001969     10           NaN
1     3.8403   48.186203   1.997375e-05  0.003938     20           NaN
2     3.5685   62.518715   1.996062e-05  0.005907     30           NaN
3     4.3222   84.525391   1.994750e-05  0.007876     40           NaN
4     3.4583   85.292900   1.993437e-05  0.009844     50           NaN
…        …          …             …         …      …           …
1523  1.4080    1.340448   2.231410e-08  2.996653  15220           NaN
1524  0.0041   73.672035   9.188160e-09  2.998622  15230           NaN
1525     NaN         NaN            NaN  3.000000  15237       53.2887
1526     NaN         NaN            NaN  3.000000  15237           NaN
1527     NaN         NaN            NaN  3.000000  15237       53.5613

      eval_samples_per_second  eval_steps_per_second  train_runtime  \
0                         NaN                    NaN            NaN
1                         NaN                    NaN            NaN
2                         NaN                    NaN            NaN
3                         NaN                    NaN            NaN
4                         NaN                    NaN            NaN
…                           …                      …              …
1523                      NaN                    NaN            NaN
1524                      NaN                    NaN            NaN
1525                   37.644                 18.822            NaN
1526                      NaN                    NaN       2633.262
1527                   37.452                 18.726            NaN

      train_samples_per_second  train_steps_per_second    total_flos  \
0                          NaN                     NaN           NaN
1                          NaN                     NaN           NaN
2                          NaN                     NaN           NaN
3                          NaN                     NaN           NaN
4                          NaN                     NaN           NaN
…                            …                       …             …
1523                       NaN                     NaN           NaN
1524                       NaN                     NaN           NaN
1525                       NaN                     NaN           NaN
1526                    11.572                   5.786  5.046867e+14
1527                       NaN                     NaN           NaN

      train_loss
0            NaN
```

```
1            NaN
2            NaN
3            NaN
4            NaN
…          …
1523         NaN
1524         NaN
1525         NaN
1526     1.01706
1527         NaN

[1528 rows x 13 columns]
Processing predictions and computing metrics…
Number of start logits: 2006
Number of end logits: 2006
Number of features: 2006
Post-processing predictions…
```

0%| | 0/2000 [00:00<?, ?it/s]

```
Evaluation Metrics:
exact: 37.30
f1: 41.25
total: 2000.00
HasAns_exact: 74.38
HasAns_f1: 82.25
HasAns_total: 1003.00
NoAns_exact: 0.00
NoAns_f1: 0.00
NoAns_total: 997.00
best_exact: 50.10
best_exact_thresh: 0.00
best_f1: 50.10
best_f1_thresh: 0.00

Question: In what country is Normandy located?
True Answer: France, France, France, France
Predicted Answer: France

Question: When were the Normans in Normandy?
True Answer: 10th and 11th centuries, in the 10th and 11th centuries, 10th and
11th centuries, 10th and 11th centuries
Predicted Answer: 10th and 11th centuries

Question: From which countries did the Norse originate?
True Answer: Denmark, Iceland and Norway, Denmark, Iceland and Norway, Denmark,
Iceland and Norway, Denmark, Iceland and Norway
Predicted Answer: Denmark, Iceland and Norway
```

```
All plots have been saved in the 'artifacts/plots' directory.
Comparative metrics plot saved as 'artifacts/comparative_metrics.png'.
Model performance metrics saved in 'artifacts/results/model_performance.json'.
```

0.3 5. Retrieval-Augmented Generation (RAG)

A simple Retrieval-Augmented Generation system that retrieves relevant passages from the corpus and generates answers using a generative model.

0.3.1 Document Chunking

[38]:
```
# Simulate a corpus of documents
corpus = list(set(train_dataset['context'] + validation_dataset['context']))

# Chunk documents into passages (e.g., every 100 words)
passages = []
for doc in corpus:
    words = doc.split()
    for i in range(0, len(words), 100):
        chunk = ' '.join(words[i:i+100])
        passages.append(chunk)
```

0.3.2 Build Passage Embeddings

[39]:
```
import torch
from transformers import AutoTokenizer, AutoModel
from torch.utils.data import DataLoader

# Use a pre-trained model for embedding (e.g., MiniLM)
retriever_model_name = 'sentence-transformers/all-MiniLM-L6-v2'
retriever_tokenizer = AutoTokenizer.from_pretrained(retriever_model_name)
retriever_model = AutoModel.from_pretrained(retriever_model_name).to(device)

def get_embeddings(texts):
    inputs = retriever_tokenizer(
        texts, padding=True, truncation=True, return_tensors='pt'
    ).to(device)
    with torch.no_grad():
        outputs = retriever_model(**inputs)
    embeddings = outputs.last_hidden_state.mean(dim=1).cpu().numpy()
    return embeddings

# Process passages in batches to reduce memory usage
batch_size = 32  # Adjust as needed based on your GPU memory
passage_embeddings = []

# Create a DataLoader to handle batching
```

```
dataloader = DataLoader(passages, batch_size=batch_size)

for batch in dataloader:
    # Get embeddings for the current batch
    batch_embeddings = get_embeddings(batch)
    passage_embeddings.extend(batch_embeddings) # Extend the list with new␣
 ↪embeddings

# Convert the list of embeddings to a NumPy array
passage_embeddings = np.array(passage_embeddings)
```

0.3.3 Define Retrieval Function

[40]:
```
def retrieve(query, k=5):
    query_embedding = get_embeddings([query])[0]
    similarities = cosine_similarity([query_embedding], passage_embeddings)[0]
    top_k = np.argsort(similarities)[-k:][::-1]
    retrieved_passages = [passages[i] for i in top_k]
    return retrieved_passages
```

0.3.4 Initialize Generative Model

[41]:
```
generator_model_name = 't5-small'
generator_tokenizer = T5Tokenizer.from_pretrained(generator_model_name)
generator_model = T5ForConditionalGeneration.
 ↪from_pretrained(generator_model_name).to(device)
```

0.3.5 Define RAG QA Function

[42]:
```
def rag_qa(question):
    # Retrieve passages
    retrieved_passages = retrieve(question, k=3)
    # Combine passages into context
    context = ' '.join(retrieved_passages)
    # Prepare input for generator
    input_text = f"question: {question}  context: {context}"
    inputs = generator_tokenizer.encode(
        input_text, return_tensors='pt', truncation=True, max_length=512
    ).to(device)
    # Generate answer
    outputs = generator_model.generate(inputs, max_length=50)
    answer = generator_tokenizer.decode(outputs[0], skip_special_tokens=True)
    return answer
```

0.3.6 Test RAG QA

[43]:
```python
# Test the RAG QA system with a question from the validation set
test_question = validation_dataset[190]['question']
print(f"Question: {test_question}")
rag_answer = rag_qa(test_question)
print(f"Generated Answer: {rag_answer}")
```

```
Question: What kind of needlework was used in the creation of the Bayeux
Tapestry?
Generated Answer: embroidery
```

0.4 Comparison of Models

[44]:
```python
print("\nModel Performance Comparison:")
for model_name in models:
    metrics = results[model_name]
    print(f"\nModel: {model_name}")
    print(f"Exact Match: {metrics['exact']:.2f}")
    print(f"F1 Score: {metrics['f1']:.2f}")
```

```
Model Performance Comparison:

Model: bert-base-uncased
Exact Match: 32.80
F1 Score: 37.83

Model: roberta-base
Exact Match: 37.95
F1 Score: 42.30

Model: albert-base-v2
Exact Match: 37.30
F1 Score: 41.25
```

[ ]: