

Ocaqe8tpt

December 16, 2024

Question 1:

Pick an existing fine-tuning dataset (or create your own - you can create it by scraping the web or using some of the documents you have on your computer (do not use anything confidential or copyrighted) or by asking ChatGPT to generate the data). Mention which dataset you picked/created and briefly describe the dataset. [10 points]

```
[ ]: import pandas as pd

# Define a new dataset
data = [
    ("What's the best way to start a morning?", "A good way to start is with a_
    ↪glass of water and some light stretching."),
    ("Can you explain photosynthesis?", "Photosynthesis is how plants convert_
    ↪sunlight into energy."),
    ("How do I bake a cake?", "Mix flour, sugar, eggs, and baking powder. Bake_
    ↪at 350°F for 30 minutes."),
    ("What's the capital of Germany?", "The capital of Germany is Berlin."),
    ("Why do stars twinkle?", "Stars twinkle because of turbulence in the_
    ↪Earth's atmosphere."),
    ("Who wrote 'Pride and Prejudice'?", "Jane Austen wrote 'Pride and_
    ↪Prejudice.'"),
    ("What's your favorite hobby?", "I don't have hobbies, but I love helping_
    ↪people!"),
    ("How do airplanes fly?", "Airplanes fly because of lift generated by their_
    ↪wings."),
    ("Can you recommend a good book?", "Sure! '1984' by George Orwell is a_
    ↪thought-provoking read."),
    ("What's the best way to learn programming?", "Start with a_
    ↪beginner-friendly language like Python and practice consistently."),
    ("Who painted 'Starry Night'?", "Vincent van Gogh painted 'Starry Night.'"),
    ("How does gravity work?", "Gravity is the force that attracts objects_
    ↪toward each other."),
    ("What's the fastest animal in the ocean?", "The black marlin is one of the_
    ↪fastest marine animals."),
    ("Can you tell me a motivational quote?", "Sure! 'The only limit to our_
    ↪realization of tomorrow is our doubts of today.'"),
```

```

    ("What is blockchain?", "Blockchain is a decentralized digital ledger used
    ↪for recording transactions."),
    ("What's the longest river in the world?", "The Nile River is often
    ↪considered the longest river."),
    ("Can you teach me how to meditate?", "Sit comfortably, close your eyes,
    ↪and focus on your breath."),
    ("Who is the president of the United States in 2024?", "The current
    ↪president is Joe Biden."),
    ("What is climate change?", "Climate change refers to long-term shifts in
    ↪weather patterns due to human activities."),
    ("How do I write an email?", "Start with a greeting, state your purpose
    ↪clearly, and close politely with your name."),
]

# Convert to a DataFrame
df = pd.DataFrame(data, columns=["User Input", "Model Response"])

# Save to CSV for reproducibility
df.to_csv("new_conversation_dataset.csv", index=False)
print("Dataset saved as 'new_conversation_dataset.csv'.")

```

Dataset saved as 'new_conversation_dataset.csv'.

Dataset Creation:

A list of tuples is defined, where each tuple contains a question and its corresponding answer.

Conversion to DataFrame:

The pandas library is used to convert the list of tuples into a structured DataFrame with two columns: “User Input” and “Model Response”.

Saving as CSV:

The dataset is saved as a CSV file named new_conversation_dataset.csv to ensure reproducibility. This allows the dataset to be used in future workflows or shared with others.

Output Confirmation:

The print statement confirms that the dataset has been saved successfully.

```
[ ]: pip install transformers datasets wandb
```

```

Requirement already satisfied: transformers in
c:\users\charu\anaconda3\lib\site-packages (4.47.0)
Requirement already satisfied: datasets in c:\users\charu\anaconda3\lib\site-
packages (3.2.0)
Requirement already satisfied: wandb in c:\users\charu\anaconda3\lib\site-
packages (0.19.1)
Requirement already satisfied: filelock in c:\users\charu\anaconda3\lib\site-
packages (from transformers) (3.13.1)

```

Requirement already satisfied: huggingface-hub<1.0,>=0.24.0 in
c:\users\charu\anaconda3\lib\site-packages (from transformers) (0.26.5)

Requirement already satisfied: numpy>=1.17 in c:\users\charu\anaconda3\lib\site-
packages (from transformers) (1.26.4)

Requirement already satisfied: packaging>=20.0 in
c:\users\charu\anaconda3\lib\site-packages (from transformers) (24.1)

Requirement already satisfied: pyyaml>=5.1 in c:\users\charu\anaconda3\lib\site-
packages (from transformers) (6.0.1)

Requirement already satisfied: regex!=2019.12.17 in
c:\users\charu\anaconda3\lib\site-packages (from transformers) (2024.9.11)

Requirement already satisfied: requests in c:\users\charu\anaconda3\lib\site-
packages (from transformers) (2.32.3)

Requirement already satisfied: tokenizers<0.22,>=0.21 in
c:\users\charu\anaconda3\lib\site-packages (from transformers) (0.21.0)

Requirement already satisfied: safetensors>=0.4.1 in
c:\users\charu\anaconda3\lib\site-packages (from transformers) (0.4.5)

Requirement already satisfied: tqdm>=4.27 in c:\users\charu\anaconda3\lib\site-
packages (from transformers) (4.66.5)

Requirement already satisfied: pyarrow>=15.0.0 in
c:\users\charu\anaconda3\lib\site-packages (from datasets) (16.1.0)

Requirement already satisfied: dill<0.3.9,>=0.3.0 in
c:\users\charu\anaconda3\lib\site-packages (from datasets) (0.3.8)

Requirement already satisfied: pandas in c:\users\charu\anaconda3\lib\site-
packages (from datasets) (2.2.2)

Requirement already satisfied: xxhash in c:\users\charu\anaconda3\lib\site-
packages (from datasets) (3.5.0)

Requirement already satisfied: multiprocessing<0.70.17 in
c:\users\charu\anaconda3\lib\site-packages (from datasets) (0.70.16)

Requirement already satisfied: fsspec<=2024.9.0,>=2023.1.0 in
c:\users\charu\anaconda3\lib\site-packages (from
fsspec[http]<=2024.9.0,>=2023.1.0->datasets) (2024.6.1)

Requirement already satisfied: aiohttp in c:\users\charu\anaconda3\lib\site-
packages (from datasets) (3.10.5)

Requirement already satisfied: click!=8.0.0,>=7.1 in
c:\users\charu\anaconda3\lib\site-packages (from wandb) (8.1.7)

Requirement already satisfied: docker-pycreds>=0.4.0 in
c:\users\charu\anaconda3\lib\site-packages (from wandb) (0.4.0)

Requirement already satisfied: gitpython!=3.1.29,>=1.0.0 in
c:\users\charu\anaconda3\lib\site-packages (from wandb) (3.1.43)

Requirement already satisfied: platformdirs in
c:\users\charu\anaconda3\lib\site-packages (from wandb) (3.10.0)

Requirement already satisfied: protobuf!=4.21.0,!5.28.0,<6,>=3.19.0 in
c:\users\charu\anaconda3\lib\site-packages (from wandb) (4.25.3)

Requirement already satisfied: psutil>=5.0.0 in
c:\users\charu\anaconda3\lib\site-packages (from wandb) (5.9.0)

Requirement already satisfied: pydantic<3,>=2.6 in
c:\users\charu\anaconda3\lib\site-packages (from wandb) (2.8.2)

Requirement already satisfied: sentry-sdk>=2.0.0 in

c:\users\charu\anaconda3\lib\site-packages (from wandb) (2.19.2)
 Requirement already satisfied: setproctitle in
 c:\users\charu\anaconda3\lib\site-packages (from wandb) (1.3.4)
 Requirement already satisfied: setuptools in c:\users\charu\anaconda3\lib\site-
 packages (from wandb) (75.1.0)
 Requirement already satisfied: colorama in c:\users\charu\anaconda3\lib\site-
 packages (from click!=8.0.0,>=7.1->wandb) (0.4.6)
 Requirement already satisfied: six>=1.4.0 in c:\users\charu\anaconda3\lib\site-
 packages (from docker-pycreds>=0.4.0->wandb) (1.16.0)
 Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
 c:\users\charu\anaconda3\lib\site-packages (from aiohttp->datasets) (2.4.0)
 Requirement already satisfied: aiosignal>=1.1.2 in
 c:\users\charu\anaconda3\lib\site-packages (from aiohttp->datasets) (1.2.0)
 Requirement already satisfied: attrs>=17.3.0 in
 c:\users\charu\anaconda3\lib\site-packages (from aiohttp->datasets) (23.1.0)
 Requirement already satisfied: frozenlist>=1.1.1 in
 c:\users\charu\anaconda3\lib\site-packages (from aiohttp->datasets) (1.4.0)
 Requirement already satisfied: multidict<7.0,>=4.5 in
 c:\users\charu\anaconda3\lib\site-packages (from aiohttp->datasets) (6.0.4)
 Requirement already satisfied: yarl<2.0,>=1.0 in
 c:\users\charu\anaconda3\lib\site-packages (from aiohttp->datasets) (1.11.0)
 Requirement already satisfied: gitdb<5,>=4.0.1 in
 c:\users\charu\anaconda3\lib\site-packages (from
 gitpython!=3.1.29,>=1.0.0->wandb) (4.0.7)
 Requirement already satisfied: typing-extensions>=3.7.4.3 in
 c:\users\charu\anaconda3\lib\site-packages (from huggingface-
 hub<1.0,>=0.24.0->transformers) (4.11.0)
 Requirement already satisfied: annotated-types>=0.4.0 in
 c:\users\charu\anaconda3\lib\site-packages (from pydantic<3,>=2.6->wandb)
 (0.6.0)
 Requirement already satisfied: pydantic-core==2.20.1 in
 c:\users\charu\anaconda3\lib\site-packages (from pydantic<3,>=2.6->wandb)
 (2.20.1)
 Requirement already satisfied: charset-normalizer<4,>=2 in
 c:\users\charu\anaconda3\lib\site-packages (from requests->transformers) (3.3.2)
 Requirement already satisfied: idna<4,>=2.5 in
 c:\users\charu\anaconda3\lib\site-packages (from requests->transformers) (3.7)
 Requirement already satisfied: urllib3<3,>=1.21.1 in
 c:\users\charu\anaconda3\lib\site-packages (from requests->transformers) (2.2.3)
 Requirement already satisfied: certifi>=2017.4.17 in
 c:\users\charu\anaconda3\lib\site-packages (from requests->transformers)
 (2024.8.30)
 Requirement already satisfied: python-dateutil>=2.8.2 in
 c:\users\charu\anaconda3\lib\site-packages (from pandas->datasets) (2.9.0.post0)
 Requirement already satisfied: pytz>=2020.1 in
 c:\users\charu\anaconda3\lib\site-packages (from pandas->datasets) (2024.1)
 Requirement already satisfied: tzdata>=2022.7 in
 c:\users\charu\anaconda3\lib\site-packages (from pandas->datasets) (2023.3)

Requirement already satisfied: smmap<5,>=3.0.1 in
c:\users\charu\anaconda3\lib\site-packages (from
gitdb<5,>=4.0.1->gitpython!=3.1.29,>=1.0.0->wandb) (4.0.0)
Note: you may need to restart the kernel to use updated packages.

Question 2:

Fine-tune an open-source pre-trained conversational language model of your choice (that you can take, e.g., from the Hugging Face Transformers library) with the dataset you picked or created. Make sure the model you pick has at least 500M parameters. [20 points]

Connect to wandb and to track the progress of your fine-tuning (e.g. your training loss). Share the link to your wandb project with us in the report you submit (see here for how to do it: <https://wandb.ai/ivangoncharov/wandb-teams-for-students/reports/How-to-Use-W-B-Teams-For-Your-University-Machine-Learning-Projects-For-Free—VmllldzoxMjk1MjkxLinks> to an external site.) [10 points]

Test your model on a few prompts before and after fine-tuning and report any interesting differences. If you didn't observe any interesting differences, comment on why not. [10 points]

```
[ ]: import wandb

# Initialize wandb
wandb.login()
```

```
wandb: Using wandb-core as the SDK backend. Please refer to
https://wandb.me/wandb-core for more information.
wandb: Currently logged in as: subashreevs (subashreevs-indiana-university). Use
`wandb login --relogin` to force relogin
```

```
[ ]: True
```

```
[ ]: # Set project name for tracking
wandb.init(project="dialogPT-fine-tuning", name="subashree-dialogPT")
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
[ ]: <wandb.sdk.wandb_run.Run at 0x21bd556ac30>
```

Tracking Experiment with Weights and Biases (wandb)

Weights and Biases (wandb) is a tool used for tracking machine learning experiments. It helps monitor metrics, visualize training progress, and log results for reproducibility. In this section, we set up wandb for logging the fine-tuning process of the DialogPT model.

wandb.login():

Authenticates the user with their Weights and Biases account. It will prompt for an API key or use an already authenticated session.

wandb.init():

Initializes a new run in a specified project.

project: The project name under which this run will be logged.

name: A unique name for this specific run, making it easier to identify among multiple runs.

```
[ ]: from datasets import Dataset

# Load the dataset
df = pd.read_csv("new_conversation_dataset.csv")

# Convert to Hugging Face Dataset format
dataset = Dataset.from_pandas(df)
```

Loading the Dataset:

The dataset is loaded from the new_conversation_dataset.csv file. It is converted into a Hugging Face Dataset for efficient processing.

```
[ ]: from transformers import AutoTokenizer
from transformers import AutoTokenizer, AutoModelForCausalLM, \
    TrainingArguments, Trainer

# Load the tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2-large")

# Add padding token
tokenizer.pad_token = tokenizer.eos_token

# Tokenize the dataset
def tokenize_function(examples):
    inputs = tokenizer(examples["User Input"], truncation=True, \
        padding="max_length", max_length=512)
    targets = tokenizer(examples["Model Response"], truncation=True, \
        padding="max_length", max_length=512)
    inputs["labels"] = targets["input_ids"] # Add labels for supervised \
        fine-tuning
    return inputs

# Apply tokenization
tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

```
# Split dataset into train and validation
train_dataset = tokenized_datasets.train_test_split(test_size=0.1)["train"]
val_dataset = tokenized_datasets.train_test_split(test_size=0.1)["test"]
```

WARNING:tensorflow:From C:\Users\charu\anaconda3\Lib\site-packages\tf_keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

Map: 0%| | 0/20 [00:00<?, ? examples/s]

Tokenization:

The AutoTokenizer is used to convert text into numerical input IDs. Padding ensures all sequences have the same length, while truncation ensures no sequence exceeds the maximum token length supported by the model.

Supervised Fine-Tuning:

Input tokens (input_ids) are paired with their corresponding labels, allowing the model to learn the mapping between inputs and responses.

Dataset Splitting:

The data is split into training and validation sets to monitor model performance during training and prevent overfitting.

```
[ ]: from transformers import GPT2LMHeadModel

# Load the pre-trained GPT2 Large model
model = GPT2LMHeadModel.from_pretrained("gpt2-large")

# Resize token embeddings for the added padding token
model.resize_token_embeddings(len(tokenizer))
```

```
[ ]: Embedding(50257, 1280)
```

Model Selection:

GPT-2 Large (812M params) is selected for this task, which is well-suited for causal language modeling tasks like dialogue generation and question answering. The GPT2LMHeadModel class is specifically designed for generating text by predicting the next token in a sequence.

Padding Token Addition:

Since GPT-2 Large was not pretrained with a padding token, the code manually adds a padding token to the tokenizer (pad_token) and ensures the model can handle it by resizing the token embedding layer.

Embedding Resizing:

The embedding layer is responsible for converting input tokens (numerical IDs) into dense vectors for processing by the transformer layers. By resizing the embeddings with

`model.resize_token_embeddings(len(tokenizer))`, the code ensures that the new padding token is properly incorporated into the model's vocabulary and embedding space.

Integration with Tokenizer:

The synchronization between the tokenizer and the model is critical. This step guarantees that all tokens, including the new padding token, are recognized by the model during training and inference.

```
[ ]: pip install transformers[torch] --upgrade
```

```
Requirement already satisfied: transformers[torch] in
c:\users\charu\anaconda3\lib\site-packages (4.47.0)
Requirement already satisfied: filelock in c:\users\charu\anaconda3\lib\site-
packages (from transformers[torch]) (3.13.1)Note: you may need to restart the
kernel to use updated packages.
Requirement already satisfied: huggingface-hub<1.0,>=0.24.0 in
c:\users\charu\anaconda3\lib\site-packages (from transformers[torch]) (0.26.5)

Requirement already satisfied: numpy>=1.17 in c:\users\charu\anaconda3\lib\site-
packages (from transformers[torch]) (1.26.4)
Requirement already satisfied: packaging>=20.0 in
c:\users\charu\anaconda3\lib\site-packages (from transformers[torch]) (24.1)
Requirement already satisfied: pyyaml>=5.1 in c:\users\charu\anaconda3\lib\site-
packages (from transformers[torch]) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in
c:\users\charu\anaconda3\lib\site-packages (from transformers[torch])
(2024.9.11)
Requirement already satisfied: requests in c:\users\charu\anaconda3\lib\site-
packages (from transformers[torch]) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in
c:\users\charu\anaconda3\lib\site-packages (from transformers[torch]) (0.21.0)
Requirement already satisfied: safetensors>=0.4.1 in
c:\users\charu\anaconda3\lib\site-packages (from transformers[torch]) (0.4.5)
Requirement already satisfied: tqdm>=4.27 in c:\users\charu\anaconda3\lib\site-
packages (from transformers[torch]) (4.66.5)
Requirement already satisfied: torch in c:\users\charu\anaconda3\lib\site-
packages (from transformers[torch]) (2.5.1)
Requirement already satisfied: accelerate>=0.26.0 in
c:\users\charu\anaconda3\lib\site-packages (from transformers[torch]) (1.2.1)
```

```
[ ]: from transformers import TrainingArguments
```

```
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    logging_dir="./logs",
    logging_steps=10,
    per_device_train_batch_size=4,
```



```

    per_device_eval_batch_size=4,
    num_train_epochs=3,
    save_total_limit=2,
    load_best_model_at_end=True,
    learning_rate=5e-5,
    lr_scheduler_type="linear",
    warmup_steps=500,
    weight_decay=0.01,
)

```

```

C:\Users\charu\anaconda3\Lib\site-packages\transformers\training_args.py:1575:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of Transformers. Use `eval_strategy` instead
  warnings.warn(

```

The code defines the training configuration for fine-tuning a model using the `TrainingArguments` class from the `transformers` library. These arguments control key aspects of the training process, such as saving checkpoints, batch size, logging, and learning rate scheduling.

Output Directory (`output_dir`):

The trained model and its checkpoints will be saved in the specified directory `./results`, ensuring that the progress of training is stored for later use or analysis.

Evaluation and Saving Strategies:

`evaluation_strategy="epoch"`: The model is evaluated on the validation dataset at the end of each epoch. `save_strategy="epoch"`: The model checkpoints are saved at the end of every epoch.

Logging:

Logs will be stored in the directory `./logs`. Metrics such as loss and evaluation scores are logged every 10 training steps, providing visibility into the training progress.

Batch Size:

`per_device_train_batch_size=4` and `per_device_eval_batch_size=4`: These control the number of examples processed per device (e.g., GPU) during training and evaluation. This setup balances performance and memory usage.

Number of Training Epochs:

`num_train_epochs=3`: The model will go through the entire training dataset 3 times, providing multiple opportunities for optimization.

Checkpoint Management:

`save_total_limit=2`: Only the 2 most recent checkpoints are kept, ensuring efficient use of storage by deleting older checkpoints.

Model Optimization:

`load_best_model_at_end=True`: The best-performing model on the validation dataset will be loaded at the end of training. `learning_rate=5e-5`: A relatively small learning rate is used, which is typical for fine-tuning pretrained models to ensure stable convergence. `lr_scheduler_type="linear"`:

A linear learning rate schedule is applied, starting from the initial learning rate and gradually decreasing over time. `warmup_steps=500`: Gradual warmup of the learning rate over the first 500 steps prevents instability during the initial phase of training. `weight_decay=0.01`: A small weight decay helps reduce overfitting by penalizing large weights in the model.

```
[ ]: from transformers import Trainer, DataCollatorForSeq2Seq

# Use a data collator for sequence-to-sequence tasks
data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)

# Initialize the Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=tokenizer,
    data_collator=data_collator,
)

# Fine-tune the model
trainer.train()

# Save the fine-tuned model and tokenizer
model.save_pretrained("./fine_tuned_gpt2")
tokenizer.save_pretrained("./fine_tuned_gpt2")
```

```
C:\Users\charu\AppData\Local\Temp\ipykernel_22988\3082441552.py:7:
FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0
for `Trainer.__init__`. Use `processing_class` instead.
  trainer = Trainer(
wandb: WARNING The `run_name` is currently set to the same value as
`TrainingArguments.output_dir`. If this was not intended, please specify a
different run name by setting the `TrainingArguments.run_name` parameter.
```

<IPython.core.display.HTML object>

There were missing keys in the checkpoint model loaded: ['lm_head.weight'].

```
[ ]: ('./fine_tuned_gpt2\\tokenizer_config.json',
      './fine_tuned_gpt2\\special_tokens_map.json',
      './fine_tuned_gpt2\\vocab.json',
      './fine_tuned_gpt2\\merges.txt',
      './fine_tuned_gpt2\\added_tokens.json',
      './fine_tuned_gpt2\\tokenizer.json')
```

This section of the code fine-tunes the pretrained model (GPT-2 Large) on the dataset using the `Trainer` class from Hugging Face's `transformers` library. The `Trainer` simplifies the training process by handling data batching, evaluation, and model checkpointing automatically.

DataCollatorForSeq2Seq:

A DataCollator is used to dynamically pad sequences in the dataset to the same length within each batch. DataCollatorForSeq2Seq is ideal for sequence-to-sequence tasks like text generation, where both input and output sequences need padding.

Trainer Initialization:

Combines the model, dataset, training arguments, tokenizer, and data collator. Automates training, evaluation, and checkpoint saving.

Parameters: * model: The pretrained model to fine-tune (GPT2 Large). * args: Training configuration, such as batch size, learning rate, and epochs. * train_dataset and eval_dataset: The datasets used for training and validation. * tokenizer: Converts text to token IDs for model input. * data_collator: Handles padding and formatting for batches during training.

Fine-Tuning:

The trainer.train() function initiates the training process based on the configuration specified in training_args.

Saving the Fine-Tuned Model:

The trained model and tokenizer are saved to the ./fine_tuned_gpt2 directory for later use.

```
[ ]: # Test prompts
test_prompts = [
    "What's the capital of Germany?",
    "Can you recommend a good book?",
    "What's the longest river in the world?",
]

# Function to generate responses
def generate_responses(model, prompts, tokenizer):
    responses = []
    for prompt in prompts:
        inputs = tokenizer.encode(prompt, return_tensors="pt")
        outputs = model.generate(inputs, max_length=50, pad_token_id=tokenizer.
        ↪pad_token_id)
        responses.append(tokenizer.decode(outputs[0], skip_special_tokens=True))
    return responses

# Load the fine-tuned model
fine_tuned_model = GPT2LMHeadModel.from_pretrained("./fine_tuned_gpt2")

# Generate responses
responses = generate_responses(fine_tuned_model, test_prompts, tokenizer)

# Print results
for i, prompt in enumerate(test_prompts):
    print(f"Prompt: {prompt}")
```

```
print(f"Response: {responses[i]}")
```

The attention mask is not set and cannot be inferred from input because pad token is same as eos token. As a consequence, you may observe unexpected behavior. Please pass your input's `attention_mask` to obtain reliable results.

Prompt: What's the capital of Germany?

Response: What's the capital of Germany?

The capital of Germany is Berlin.

The capital of Germany is Berlin.

The capital of Germany is Berlin.

The capital of Germany is Berlin.

The capital of

Prompt: Can you recommend a good book?

Response: Can you recommend a good book?

I'm not sure I can. I've read a lot of books, but I don't know if I can recommend a good book. I think it's a good book, but I don't know

Prompt: What's the longest river in the world?

Response: What's the longest river in the world?

The longest river in the world is the Nile, which is about 1,000 miles long. The longest river in the world is the Nile, which is about 1,000 miles long

Test Prompts:

A list of test prompts is provided to evaluate the fine-tuned model's ability to generate relevant and accurate responses

Response Generation:

The `generate_responses` function takes the model, test prompts, and tokenizer as input. Each prompt is tokenized using `tokenizer.encode()`, and the model generates a response using `model.generate()`. The output tokens are decoded into human-readable text using `tokenizer.decode()`.

Model Loading:

The fine-tuned model is loaded from the `./fine_tuned_gpt2` directory, where it was saved after training.

Printing Results:

The prompts and their corresponding responses are printed for evaluation.

Before Fine-Tuning:

The pretrained GPT-2 model is likely to produce general, less specific, or irrelevant responses since it hasn't been fine-tuned for the test prompts.

After Fine-Tuning:

The fine-tuned GPT-2 model should produce more accurate, specific, and contextually relevant responses, reflecting the content of the fine-tuning dataset.

Observations:

1. Repetition in Responses:

The generated responses exhibit excessive repetition, such as: "The capital of Germany is Berlin." being repeated multiple times for the first prompt. This suggests that the model might have overfitted to patterns in the training data or lacks variability in its output.

2. Inaccurate Responses:

For "What's the longest river in the world?", the model generated: "The longest river in the world is the Nile, which is about 1,000 miles long." While the Nile is the correct answer, its length is factually incorrect (~4,000 miles).

3. Vague Responses:

For "Can you recommend a good book?", the model provided: "I'm not sure I can. I've read a lot of books, but I don't know if I can recommend a good book." This response is vague and lacks the specific answer "1984 by George Orwell" provided in the training dataset.

Conclusion:

1. What Worked:

The framework for generating and testing responses using the fine-tuned GPT-2 Large model is functional. The model generates relevant responses that align with the prompts for simple and straightforward queries.

2. What Needs Improvement:

- Repetition: The model often repeats phrases, which can be mitigated by adding diverse examples to the dataset and penalizing repetition during text generation using parameters like `repetition_penalty` in `model.generate()`.
- Precision: The responses need to be more specific and factually correct. Expanding the dataset with more curated and accurate examples can improve precision.
- Garbled Output: Debug tokenization issues and ensure training data adequately covers diverse phrasing and detailed examples to address vague or garbled outputs.