# Fine-Tuning TinyLLama Model for Question-Answering

## Overview

This document explains the process of fine-tuning the `Maykeye/TinyLLama-v0` model for a Question-Answering (Q&A) task using a custom dataset. It covers the steps from loading the model and tokenizer to fine-tuning the model and generating predictions.

### Required Libraries

Ensure you have the following Python libraries installed:  
- `transformers`  
- `datasets`  
- `torch`

## Step 1: Load Model and Tokenizer

The model and tokenizer are loaded using the `transformers` library.

```python  
from transformers import AutoTokenizer, AutoModelForCausalLM  
  
# Load the tokenizer and model  
tokenizer = AutoTokenizer.from\_pretrained("Maykeye/TinyLLama-v0")  
model = AutoModelForCausalLM.from\_pretrained("Maykeye/TinyLLama-v0")  
```

## Step 2: Load the Dataset

Load the Q&A dataset in JSON format.

```python  
from datasets import load\_dataset  
  
# Load your Q&A dataset  
dataset = load\_dataset("json", data\_files="/content/new.json")  
  
# Ensure dataset keys are correctly renamed  
def rename\_keys(example):  
 return {"prompt": example["question"], "response": example["answer"]}  
  
dataset = dataset.map(rename\_keys)  
```

## Step 3: Training Arguments

Define the training parameters using the `TrainingArguments` class.

```python  
from transformers import TrainingArguments  
  
training\_args = TrainingArguments(  
 output\_dir="./fine\_tuned\_tinyllama",  
 num\_train\_epochs=5,  
 per\_device\_train\_batch\_size=8,  
 gradient\_accumulation\_steps=4,  
 warmup\_steps=500,  
 learning\_rate=3e-5,  
 logging\_dir="./logs",  
 logging\_steps=100,  
 save\_steps=500,  
 save\_total\_limit=3,  
 fp16=True,  
 evaluation\_strategy="steps",  
 save\_strategy="steps",  
 load\_best\_model\_at\_end=True,  
 metric\_for\_best\_model="eval\_loss",  
 greater\_is\_better=False,  
)  
```

## Step 4: Adjust Tokenizer and Model

Add a padding token and resize the model's token embeddings to match the tokenizer.

```python  
# Add a padding token  
tokenizer.add\_special\_tokens({'pad\_token': '[PAD]'})  
tokenizer.pad\_token\_id = tokenizer.eos\_token\_id # Set pad token id to eos token id  
model.resize\_token\_embeddings(len(tokenizer))  
```

## Step 5: Tokenize the Dataset

Define a function to tokenize and preprocess the dataset for training.

```python  
import torch  
  
def tokenize\_function(example):  
 # Tokenize the prompt and response  
 prompt = tokenizer(example["prompt"], padding="max\_length", truncation=True, max\_length=128)  
 response = tokenizer(example["response"], padding="max\_length", truncation=True, max\_length=128)  
  
 # Create labels by shifting the response input\_ids  
 labels = response['input\_ids']  
 labels = [tokenizer.pad\_token\_id] + labels[:-1] # Shift labels to the right  
  
 return {  
 "input\_ids": torch.tensor(prompt["input\_ids"]),  
 "attention\_mask": torch.tensor(prompt["attention\_mask"]),  
 "labels": torch.tensor(labels),  
 "label\_attention\_mask": torch.tensor(response["attention\_mask"]),  
 }  
  
# Split dataset into train and eval sets  
train\_test\_split = dataset["train"].train\_test\_split(test\_size=0.2)  
train\_dataset = train\_test\_split["train"]  
eval\_dataset = train\_test\_split["test"]  
  
# Tokenize datasets  
tokenized\_train = train\_dataset.map(tokenize\_function, batched=False)  
tokenized\_eval = eval\_dataset.map(tokenize\_function, batched=False)  
```

## Step 6: Initialize Trainer

Set up the Trainer class for fine-tuning the model.

```python  
from transformers import Trainer  
  
trainer = Trainer(  
 model=model,  
 args=training\_args,  
 train\_dataset=tokenized\_train,  
 eval\_dataset=tokenized\_eval  
)  
```

## Step 7: Train the Model

Train the model using the `.train()` method.

```python  
# Train the model  
trainer.train()  
```

## Step 8: Save the Fine-Tuned Model

Save the fine-tuned model and tokenizer for future use.

```python  
# Save the fine-tuned model and tokenizer  
model.save\_pretrained("./fine\_tuned\_tinyllama")  
tokenizer.save\_pretrained("./fine\_tuned\_tinyllama")  
```

## Step 9: Generate Predictions

Use the fine-tuned model to generate predictions for a given prompt.

```python  
prompt = "1. What is Data Science?"  
input\_ids = tokenizer(prompt, return\_tensors="pt").input\_ids  
output\_ids = model.generate(input\_ids, max\_length=128, eos\_token\_id=tokenizer.eos\_token\_id)  
response = tokenizer.decode(output\_ids[0], skip\_special\_tokens=True)  
print(response)  
```

## Notes

- Ensure the dataset is formatted correctly with `question` and `answer` keys.  
- Adjust hyperparameters in `TrainingArguments` for optimal performance.  
- Use a GPU for training to significantly reduce time.