Contents

[Qwen LLM Fine-tuning Project Documentation 1](#_Toc198279072)

[1. Introduction 1](#_Toc198279073)

[1.1. Project Objectives 1](#_Toc198279074)

[1.2. Team Members 1](#_Toc198279075)

[2. Project Architecture 1](#_Toc198279076)

[2.1. Key Components and Files 2](#_Toc198279077)

[3. Implementation Details 2](#_Toc198279078)

[3.1. Environment Setup 2](#_Toc198279079)

[3.2. Data Preparation and Loading 3](#_Toc198279080)

[3.3. Model Loading and Configuration 3](#_Toc198279081)

[3.4. Fine-Tuning Process 4](#_Toc198279082)

[3.5. Model Saving and Inference 4](#_Toc198279083)

[3.6. Role of data\_model.py and functions.py 5](#_Toc198279084)

[4. How to Run the Project 5](#_Toc198279085)

[4.1. Prerequisites 5](#_Toc198279086)

[4.2. Setup and Installation 5](#_Toc198279087)

[4.3. Running the Fine-Tuning 6](#_Toc198279088)

[5. Project Files Overview 6](#_Toc198279089)

[6. Potential Challenges and Considerations 6](#_Toc198279090)

[7. Conclusion 7](#_Toc198279091)

# Qwen LLM Fine-tuning Project Documentation

## 1. Introduction

This document provides a comprehensive overview of the Qwen Large Language Model (LLM) Fine-tuning project. The project focuses on adapting a pre-trained Qwen LLM to a specific downstream task or dataset, enhancing its capabilities for specialized applications. Fine-tuning allows the model to learn domain-specific nuances, terminology, and response styles not extensively covered in its general pre-training.

### 1.1. Project Objectives

* To understand and implement the process of fine-tuning a Qwen LLM.
* To prepare a custom dataset suitable for fine-tuning.
* To configure and run the fine-tuning process using appropriate tools and libraries (e.g., Transformers, PEFT/LoRA, bitsandbytes for quantization if used).
* To evaluate the performance of the fine-tuned model (qualitatively or quantitatively based on the task).
* To document the fine-tuning pipeline, including data preparation, model configuration, training parameters, and potential challenges.

### 1.2. Team Members

* Abdallah Samir
* Youssef Samy
* Shaaban Mosaad
* Nada Amr
* Mostafa Ahmed Elesely
* Mohammed Ahmed Mohammed Badrawy

## 2. Project Architecture

The Qwen LLM fine-tuning project generally involves the following stages:

1. **Dataset Preparation**: Creating or sourcing a dataset tailored to the target task. This involves formatting the data into a prompt-response or instruction-following structure.
2. **Environment Setup**: Installing necessary libraries, including Hugging Face Transformers, Accelerate, PEFT (Parameter-Efficient Fine-Tuning), datasets, etc.
3. **Model Loading**: Loading the pre-trained Qwen LLM (e.g., Qwen-7B-Chat) and its tokenizer.
4. **Fine-tuning Configuration**: Setting up training arguments, LoRA/PEFT configurations (if used for efficiency), quantization (like 4-bit if using QLoRA), and data collation.
5. **Training**: Running the fine-tuning script or notebook cells to train the model on the custom dataset.
6. **Model Saving and Inference**: Saving the fine-tuned model adapters (if using PEFT) or the full model, and then loading it for inference to test its performance on new prompts.

### 2.1. Key Components and Files

From the GitHub repository (https://github.com/YosefSamy019/DEPI\_graduation\_project/tree/main/llm\_fine\_tuning\_(qwen\_code)):

* llm\_finetuning(all\_project).ipynb: The primary Jupyter Notebook containing the entire fine-tuning pipeline, from environment setup and data loading to training and inference. This is the central piece of the project.
* project.ipynb: Another Jupyter Notebook, potentially an earlier version or a focused part of the fine-tuning process. Its exact role relative to llm\_finetuning(all\_project).ipynb would be clarified by its content.
* data\_model.py: Defines Pydantic models for structuring data, possibly for data generation, validation, or formatting inputs/outputs for the LLM. It includes models like Entity, NewsDetails, StoryContext, GeneratedStory, etc., suggesting the fine-tuning task might be related to news/story generation or analysis.
* functions.py: Contains utility functions, such as parse\_json, which might be used for processing model outputs or preparing data.
* requirements.txt (expected, though not explicitly listed in the immediate directory view, it’s a common practice for Python projects, especially those involving LLMs, and was mentioned in the context of the other project): Would list all necessary Python packages.

## 3. Implementation Details

This section describes the fine-tuning process based on the likely contents of llm\_finetuning(all\_project).ipynb and supporting files.

### 3.1. Environment Setup

The notebook typically starts by installing required libraries. Key libraries for Qwen fine-tuning often include:

* transformers: For loading Qwen models and tokenizers, and for the Trainer API.
* datasets: For loading and processing custom datasets.
* peft: For Parameter-Efficient Fine-Tuning techniques like LoRA.
* accelerate: To enable distributed training and mixed-precision training.
* bitsandbytes: For 4-bit quantization (QLoRA) to reduce memory footprint.
* torch: The underlying deep learning framework.
* tensorboard or wandb: For logging and monitoring training progress (wandb is imported in the notebook).

**Conceptual Setup Snippet (from llm\_finetuning(all\_project).ipynb):**

# Conceptual pip installs from the notebook  
# !pip install -q transformers==4.40.3 datasets==2.18.0 optimum==1.24.0  
# !pip install -q openai==1.61.0 wandb  
# !pip install -q json-repair==0.29.1  
# !pip install -qU faker==35.2.0  
# !pip install -qU vllm==0.7.2 # vLLM might be for optimized inference, not training  
  
# Cloning LLaMA-Factory (if used, as seen in a cell)  
# !git clone --depth 1 https://github.com/hiyouga/LLaMA-Factory.git  
# !cd LLaMA-Factory && pip install -e .

### 3.2. Data Preparation and Loading

* **Dataset Format**: The custom dataset needs to be formatted in a way the model can learn from. For chat models like Qwen-Chat, this often involves a list of conversations, where each conversation has turns with “role” (e.g., “system”, “user”, “assistant”) and “content”. The data\_model.py suggests a structured data approach, possibly for generating or processing data that will then be formatted into prompts.
* **Loading**: The dataset might be loaded from a CSV, JSON, or Hugging Face datasets library format.
* **Tokenization**: The Qwen tokenizer is used to convert the text data into input IDs, attention masks, etc.

### 3.3. Model Loading and Configuration

* **Base Model**: Loading a pre-trained Qwen model (e.g., Qwen/Qwen-7B-Chat or a specific version like Qwen/Qwen1.5-7B-Chat).
* **Quantization (Optional but common for large models)**: If using QLoRA, the model is loaded in 4-bit precision using bitsandbytes configuration (BitsAndBytesConfig).
* **PEFT Configuration (LoRA)**: If LoRA is used, a LoraConfig is defined, specifying target modules (e.g., q\_proj, k\_proj, v\_proj, o\_proj, gate\_proj, up\_proj, down\_proj for Qwen), rank (r), alpha (lora\_alpha), and dropout (lora\_dropout). The base model is then wrapped with get\_peft\_model.

**Conceptual Model Loading Snippet (from llm\_finetuning(all\_project).ipynb):**

from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig  
from peft import LoraConfig, get\_peft\_model, prepare\_model\_for\_kbit\_training  
  
model\_name = "Qwen/Qwen1.5-7B-Chat" # Example model  
  
# Quantization Config (if used)  
# bnb\_config = BitsAndBytesConfig(  
# load\_in\_4bit=True,  
# bnb\_4bit\_quant\_type="nf4",  
# bnb\_4bit\_compute\_dtype=torch.bfloat16,  
# bnb\_4bit\_use\_double\_quant=True,  
# )  
  
# model = AutoModelForCausalLM.from\_pretrained(  
# model\_name,  
# # quantization\_config=bnb\_config, # if using QLoRA  
# trust\_remote\_code=True,  
# device\_map="auto"  
# )  
# tokenizer = AutoTokenizer.from\_pretrained(model\_name, trust\_remote\_code=True)  
# tokenizer.pad\_token = tokenizer.eos\_token # Common practice  
  
# LoRA Config (if used)  
# lora\_config = LoraConfig(  
# r=16, # Example rank  
# lora\_alpha=32,  
# target\_modules=["q\_proj", "k\_proj", "v\_proj", "o\_proj", "gate\_proj", "up\_proj", "down\_proj"], # Specific to Qwen architecture  
# lora\_dropout=0.05,  
# bias="none",  
# task\_type="CAUSAL\_LM"  
# )  
# model = prepare\_model\_for\_kbit\_training(model) # if using k-bit training  
# model = get\_peft\_model(model, lora\_config)

### 3.4. Fine-Tuning Process

* **Training Arguments**: TrainingArguments from the Transformers library are defined, specifying output directory, number of epochs, batch size, learning rate, weight decay, logging steps, evaluation strategy, save strategy, etc.
* **Trainer**: The Trainer (or SFTTrainer from trl if using supervised fine-tuning utilities) is initialized with the model, tokenizer, training arguments, and the dataset.
* **Starting Training**: trainer.train() is called to start the fine-tuning process.
* **Monitoring**: Tools like Weights & Biases (wandb) are often used to monitor training metrics (loss, accuracy, etc.). The notebook shows wandb.login() and report\_to="wandb" in training arguments.

### 3.5. Model Saving and Inference

* **Saving**: After training, the fine-tuned model (or LoRA adapters) are saved. trainer.save\_model("output\_dir") or model.save\_pretrained("output\_dir\_lora").
* **Inference**: To test the fine-tuned model, it (and its adapters, if any) are loaded, and prompts are passed through its generation pipeline.

**Conceptual Inference Snippet:**

# Load fine-tuned model (if LoRA, merge adapters or load with PeftModel)  
# from peft import PeftModel  
# base\_model = AutoModelForCausalLM.from\_pretrained(model\_name, trust\_remote\_code=True, device\_map="auto")  
# model = PeftModel.from\_pretrained(base\_model, "path\_to\_lora\_adapters")  
# model = model.merge\_and\_unload() # To get a standalone fine-tuned model  
  
# prompt = "Your test prompt here"  
# inputs = tokenizer(prompt, return\_tensors="pt").to(model.device)  
# outputs = model.generate(\*\*inputs, max\_new\_tokens=100)  
# response = tokenizer.decode(outputs[0], skip\_special\_tokens=True)  
# print(response)

### 3.6. Role of data\_model.py and functions.py

* data\_model.py: The Pydantic models (e.g., NewsDetails, StoryContext, GeneratedStory) suggest that the fine-tuning task might involve generating structured news articles or stories, or extracting information into these structures. The LLM could be fine-tuned to output JSON conforming to these Pydantic models or to take inputs structured by them.
* functions.py: The parse\_json function (using json\_repair) is crucial if the LLM is expected to generate JSON outputs, as LLM-generated JSON can sometimes be malformed. This function would help in reliably parsing the model’s output.

## 4. How to Run the Project

### 4.1. Prerequisites

* Python 3.x
* pip
* GPU with sufficient VRAM (fine-tuning LLMs is computationally intensive). Google Colab (as indicated by from google.colab import drive) is often used.
* Libraries listed in requirements.txt (or installed in the notebook).

### 4.2. Setup and Installation

1. **Clone the Repository** (if not already done): bash git clone https://github.com/YosefSamy019/DEPI\_graduation\_project.git cd DEPI\_graduation\_project/llm\_fine\_tuning\_(qwen\_code)
2. **Install Dependencies**: Run pip install -r requirements.txt (if provided) or execute the installation cells at the beginning of the llm\_finetuning(all\_project).ipynb notebook.
3. **Data**: Ensure the custom dataset is available and accessible by the notebook (e.g., uploaded to Google Drive if using Colab, or placed in the project directory).
4. **W&B Login (Optional)**: If using Weights & Biases for logging, log in using wandb.login() as prompted in the notebook.

### 4.3. Running the Fine-Tuning

* Open and run the cells in the llm\_finetuning(all\_project).ipynb Jupyter Notebook sequentially. This will typically cover:
  + Mounting Google Drive (if applicable).
  + Installing packages.
  + Loading data.
  + Loading the model and tokenizer.
  + Configuring PEFT/LoRA (if used).
  + Setting training arguments.
  + Executing the training loop.
  + Saving the model.
  + Running inference examples.

## 5. Project Files Overview

Located in DEPI\_graduation\_project/llm\_fine\_tuning\_(qwen\_code)/:

* llm\_finetuning(all\_project).ipynb: The main Jupyter Notebook for the fine-tuning process.
* project.ipynb: Another Jupyter Notebook, possibly for experimentation or a subset of the main process.
* data\_model.py: Python script defining Pydantic data models for structured data handling.
* functions.py: Python script with utility functions, like JSON parsing.
* \_\_pycache\_\_/: Directory containing Python bytecode cache.

## 6. Potential Challenges and Considerations

* **Computational Resources**: Fine-tuning LLMs requires significant GPU memory and processing power. QLoRA helps, but resources are still a major factor.
* **Dataset Quality and Size**: The performance of the fine-tuned model heavily depends on the quality and quantity of the fine-tuning dataset.
* **Hyperparameter Tuning**: Finding optimal training arguments (learning rate, batch size, LoRA parameters, etc.) can be iterative and time-consuming.
* **Catastrophic Forgetting**: The model might forget some of its general capabilities while learning the specific task. Techniques exist to mitigate this, but it’s a consideration.
* **Evaluation**: Defining appropriate metrics and a robust evaluation strategy for the specific downstream task is crucial.

## 7. Conclusion

The Qwen LLM Fine-tuning project provides a practical guide to adapting a powerful pre-trained language model for specialized tasks. By leveraging tools like Hugging Face Transformers, PEFT, and potentially quantization, it demonstrates how to make LLM fine-tuning more accessible and efficient. The structured approach to data (indicated by data\_model.py) and utility functions (functions.py) further enhances the robustness of the pipeline, particularly if the task involves generating or processing structured JSON outputs.