Retrieval-Augmented Generation (RAG) Mathematical Problem Solver Documentation

Overview

This project implements a Retrieval-Augmented Generation (RAG) pipeline for solving mathematical problems using natural language processing techniques. It integrates fine-tuning, model setup, and evaluation in a modular fashion, leveraging Hugging Face's Transformers library and other tools.

Components

1. prepare_dataset.py

This script prepares the dataset for training and evaluation.

- **Functionality**: Loads and splits the mathematical dataset using Hugging Face's `datasets` library.
- **Usage**: Execute `prepare_dataset()` to retrieve train and evaluation datasets.

2. models_utils.py

This module handles model setup, fine-tuning, and preprocessing.

- **Functions**:
  - `setup_model_and_tokenizer(model_name)`: Sets up the model and tokenizer from Hugging Face's `AutoModelForCausalLM`.
  - `preprocess_function(examples, tokenizer)`: Preprocesses examples for fine-tuning.
  - `fine_tune_model(model, tokenizer, train_dataset, eval_dataset)`: Fine-tunes the model using the `Trainer` from Hugging Face.

3. rag_pipeline.py

Implements the RAG pipeline for generating answers based on retrieved contexts.

- **Functions**:
  - `setup_rag(train_dataset, tokenizer)`: Sets up the RAG pipeline with a generator model and Faiss index.
  - `rag_generate(query, generator, index, get_embedding, train_dataset)`: Generates answers using the RAG pipeline.

4. evaluate.py

Evaluates the performance of the RAG pipeline.

- **Functionality**: Computes accuracy and BLEU scores for generated answers against ground truth.
- **Dependencies**: Uses `nltk` for BLEU score computation.

5. instruct_finetune.py

Performs fine-tuning with instructional prompts.

- **Functionality**: Fine-tunes the model with structured instructional prompts for improved response generation.

6. main.py

Main script orchestrating the entire pipeline.

- **Usage**: Executes dataset preparation, model setup, fine-tuning, RAG pipeline setup, evaluation, and instructive fine-tuning.

Enhancements for Scalability

Parallelization

- Utilizes `multiprocessing` for parallel preprocessing and fine-tuning tasks.

Resource Management

- Monitors memory usage and optimizes batch sizes and concurrency for efficient resource utilization.

Dockerization

- Provides a Dockerfile for containerization:

```
FROM pytorch/pytorch:1.10.0-cuda11.3-cudnn8-devel
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
ENV PYTHONPATH=/app
CMD ["python", "main.py"]
```

Setup Instructions

1. **Clone Repository**: Clone the project repository from GitHub.
2. **Install Dependencies**: Install required dependencies using pip:

   ```
   pip install -r requirements.txt
   ```

3. **Dataset Preparation**: Execute prepare_dataset.py to load and split the dataset.
4. **Model Setup and Fine-Tuning**: Configure models_utils.py for model setup and execute fine_tune_model().
5. **RAG Pipeline Setup**: Set up the RAG pipeline in rag_pipeline.py using setup_rag().
6. **Evaluation**: Run evaluate.py to evaluate RAG pipeline performance.
7. **Instructive Fine-Tuning**: Execute instruct_finetune.py for fine-tuning with instructional prompts.
8. **Main Execution**: Run main.py to orchestrate the entire pipeline.

Usage

- Each script can be executed independently for specific tasks.
- Customize parameters and configurations based on specific requirements.
- Monitor logs and outputs for performance metrics and errors.

Notes

- Ensure compatibility with Python versions (>=3.6) and library versions.
- Adjust configurations (batch_size, num_epochs, etc.) for optimal performance based on hardware and dataset size.