GRPO Web Feedback Trainer — Documentation

# Project Structure

.  
├── app/  
│ ├── \_\_init\_\_.py # App package initializer  
│ ├── web.py # Flask UI and endpoints for prompt input and feedback  
│ ├── trainer.py # Core training logic using GRPOTrainer  
│ ├── model\_loader.py # Handles model loading, saving, and reloading  
│ ├── state.py # Manages shared state across threads and sessions  
├── run.py # Entry point to launch web server and background trainer  
├── config.py # Config file for environment settings (HOST, PORT, etc.)

# Purpose

This project is a human-in-the-loop reward trainer for fine-tuning LLMs using the GRPO (Generative Reward Policy Optimization) algorithm. It enables:  
- Prompt submission via a web interface.  
- Collection of multiple model completions.  
- Rating and feedback from humans per completion.  
- Fine-tuning of the model based on real-time feedback.

# How It Works

## 1. run.py

Initializes the Flask app and starts the background training loop in a separate daemon thread. Run it with:  
python run.py

## 2. app/web.py

Defines two main endpoints:  
- `/` (GET/POST): Submit prompt and redirect to /rate/<session\_id>.  
- `/rate/<session\_id>` (GET/POST): Display completions and collect ratings.

## 3. app/state.py

Maintains shared in-memory data structures:  
- session\_store: Stores prompts, completions, ratings, etc.  
- prompt\_queue: FIFO queue of prompts  
- state\_lock: Ensures thread-safe access

## 4. app/trainer.py

Handles training logic using GRPOTrainer. Uses a reward function to:  
- Store completions in session\_store  
- Wait for ratings  
- Train model based on human feedback

## 5. app/model\_loader.py

Handles model loading, saving, and reloading of LoRA weights.

## 6. config.py

Stores configuration variables like HOST, PORT, DEBUG\_MODE, and LORA\_PATH.

# 🌐 Features

- 🧾 Prompt submission via Bootstrap form  
- 🔄 Asynchronous feedback loop via background thread  
- ✍️ Human rating UI (0.0–1.0 scale)  
- 🔁 Supports multiple feedback rounds  
- 🔒 Thread-safe shared state  
- 💾 LoRA weight saving/loading

# Limitations

- Single shared trainer/model  
- Not production-scale ready out of the box  
- GPU contention possible with concurrent threads

# Future Improvements

- Dynamic trainer sessions  
- Thread pool/job queuing  
- Persistent DB for sessions  
- Auth for user-level feedback tracking  
- Analytics dashboard

# Setup & Installation

1. Install dependencies:  
 pip install flask transformers datasets unsloth accelerate  
  
2. Set environment variables:  
 export DEBUG=True  
 export PORT=8000  
  
3. Run the app:  
 python run.py

# Example Usage

1. Visit http://localhost:8000/  
2. Submit a prompt (e.g., "Write a poem about the moon.")  
3. Wait for completions to appear  
4. Rate each response and optionally provide feedback  
5. Submit and trigger fine-tuning