O-MAPL: Offline Multi-Agent Preference Learning

This repository contains the implementation of **O-MAPL** (Offline Multi-Agent Preference Learning), an end-to-end framework for preference-based learning in cooperative multi-agent reinforcement learning (MARL). O-MAPL eliminates the need for explicit reward modeling by directly learning Q-functions from human preferences. The framework is designed to handle both rule-based and LLM-based preference datasets, with experiments conducted on **SMACv1**, **SMACv2**, and **MaMujoco** benchmarks.

Table of Contents

Overview

O-MAPL introduces a unified, end-to-end learning process for multi-agent preference-based reinforcement learning. Instead of relying on a two-phase approach (reward modeling followed by policy optimization), O-MAPL directly learns Q-functions from preference data. Key highlights include:

- **Value Factorization**: A novel value decomposition strategy ensures scalability and stability in multi-agent systems.
- **LLM Integration**: Support for preference datasets generated using large language models (LLMs) like GPT-4.
- **Benchmarks**: Comprehensive evaluation on **SMACv1**, **SMACv2**, and **MaMujoco** benchmarks.

Project Structure

```
.
├── algos                 # Implementation of O-MAPL and baseline algorithms
│   ├── bc.py             # Behavioral Cloning (BC)
│   ├── iipl.py           # Independent Inverse Preference Learning (IIPL)
│   ├── iplvdn.py         # IPL with VDN aggregation
│   ├── omarl.py          # O-MAPL implementation
│   ├── slmarl.py         # Supervised Learning for MARL (2-phase approach)
│   └── utils.py          # Utilities for algorithms
├── envs                  # Environment wrappers for SMACv1, SMACv2, and MaMujoco
│   ├── mamujoco          # Multi-agent MuJoCo environments
│   ├── smacv1            # SMACv1 environments
│   └── smacv2            # SMACv2 environments
├── trainers              # Training scripts for each algorithm
├── rollouts              # Rollout utilities for continuous and discrete environments
├── saved_results_final   # Saved experimental results (rule-based and LLM-based)
├── graphs                # Pre-generated graphs for analysis
├── analyze.ipynb         # Jupyter notebook for analysis and visualization
├── llm_generate.py       # Script to generate prompts for LLM-based annotations
├── llm_upload.py         # Script to upload prompts to OpenAI APIs
├── llm_retrieve.py       # Script to retrieve LLM-based annotation results
├── main.py               # Main training script
```

```
├── evaluate.py           # Evaluation script
├── config.py             # Configuration file
├── utils.py              # General utilities
└── readme.md             # Readme file
```

Installation Instructions

Follow these steps to set up and run the project:

1. **Clone the repository**:

   ```
   git clone https://github.com/your-repo/omapl.git
   cd omapl
   ```

2. **Install dependencies**: Ensure you have Python 3.8+ installed. Then, install the required packages:

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

3. **Set up OpenAI API access**: For generating LLM-based datasets, ensure you have access to OpenAI APIs. Set your API key as an environment variable:

   ```
   export OPENAI_API_KEY="your-api-key"
   ```

Usage

Step 1: Generate LLM Dataset

To generate preference datasets using LLM-based annotations, follow these steps:

1. **Generate prompts**: This script generates prompts based on trajectory data:

   ```
   python -u llm_generate.py
   ```

2. **Upload prompts to OpenAI**: Use the `llm_upload.py` script to submit prompts to the OpenAI API in batches:

   ```
   python -u llm_upload.py
   ```

   This script will handle token usage and batch processing.

3. **Retrieve LLM responses**: Once the prompts are processed, retrieve the results:

   ```
   python -u llm_retrieve.py
   ```

The generated preference data will be saved for use in training.

Step 2: Training

Train the O-MAPL algorithm or baselines on different environments:

- **For MaMujoco**:

  ```
  python -u main.py --algo OMAPL --env_name Ant-v2 --seed 0 --lr 1e-4 --action_scale 0.7
  ```

- **For SMACv1**:

  ```
  python -u main.py --algo OMAPL --env_name 5m_vs_6m --seed 0 --batch_size 8 --lr 1e-4
  ```

- **For SMACv2**:

  ```
  python -u main.py --algo OMAPL --env_name protoss_5_vs_5 --seed 0 --batch_size 8 --lr 1e-4
  ```

To train with LLM-based datasets, add the `--use-llm` flag:

```
python -u main.py --algo OMAPL --env_name protoss_5_vs_5 --use-llm
```

Step 3: Evaluation

Evaluate trained models using the `evaluate.py` script:

```
python -u evaluate.py --algo {algo} --env_name {env_name} --seed {seed} --eval_step {eval_step}
```

Replace `{algo}`, `{env_name}`, `{seed}`, and `{eval_step}` with the appropriate algorithm, environment, seed number, and training step.

Step 4: Analysis and Visualization

Use the `analyze.ipynb` notebook for detailed analysis:

- Load experimental results.
- Visualize performance metrics (returns, win rates).
- Generate tables and graphs for evaluation.

Example:

- Rule-based results: `saved_results_final/results.pkl`
- LLM-based results: `saved_results_final/results_llm.pkl`

Datasets

Rule-Based Preference Data

- Generated by sampling trajectories from offline datasets and assigning binary preferences based on quality (e.g., poor vs. expert).
- Covers **SMACv1**, **SMACv2**, and **MaMujoco** environments.

LLM-Based Preference Data

- Generated using GPT-4 with detailed prompts describing trajectory outcomes.
- Provides richer annotations for tasks like **SMACv1** and **SMACv2**.
- Example prompt:

  ```
  Scenario: 5m_vs_6m
  Allied Agents Health: [0.5, 0.4, 0.3, 0.2, 0.1]
  Enemy Agents Health: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
  Total Steps: 28
  Which trajectory is better? #1 or #2
  ```

Key Features

- **Algorithms**:
  - O-MAPL (proposed method)
  - Baselines: BC, IIPL, IPL-VDN, SL-MARL
- **Environments**:
  - SMACv1: Cooperative tasks in StarCraft II.
  - SMACv2: Enhanced StarCraft II tasks with randomized start positions and unit types.
  - MaMujoco: Continuous control tasks for multi-agent robotics.
- **Datasets**:
  - Rule-based and LLM-generated preference datasets.
- **Visualization**:
  - Pre-generated graphs for returns and win rates (see `graphs/` folder).

Results

- **SMACv1**: O-MAPL achieves the highest win rates in tasks like `2c_vs_64zg` and `corridor`.
- **SMACv2**: Significant improvements in complex scenarios like `protoss_10_vs_11` and `terran_20_vs_20`.
- **MaMujoco**: O-MAPL outperforms baselines in continuous control tasks like `Ant-v2` and `HalfCheetah-v2`.

For detailed results, see the `saved_results_final` folder and the `graphs` directory.

License