Contribution Guidelines

Table of Contents

- [Project Overview](#project-overview)
- [Getting Started](#getting-started)
  - [Installation](#installation)
  - [Project Structure](#project-structure)
- [How to Contribute](#how-to-contribute)
  - [Reporting Issues](#reporting-issues)
  - [Submitting Pull Requests](#submitting-pull-requests)
- [Coding Standards](#coding-standards)
  - [Type Annotations](#type-annotations)
  - [Docstrings and Documentation](#docstrings-and-documentation)
  - [Testing](#testing)
  - [Code Style](#code-style)
- [Areas Needing Contributions](#areas-needing-contributions)
  - [Writing Tests](#writing-tests)
  - [Improving Documentation](#improving-documentation)
  - [Creating Training Scripts](#creating-training-scripts)
- [Community and Support](#community-and-support)
- [License](#license)

Project Overview

swarms is a library focused on making it simple to orchestrate agents to automate real-world activities. The goal is to automate the world economy with these swarms of agents.

We need your help to:

- **Write Tests**: Ensure the reliability and correctness of the codebase.
- **Improve Documentation**: Maintain clear and comprehensive documentation.
- **Add New Orchestration Methods**: Add multi-agent orchestration methods
- **Removing Defunct Code**: Removing bad code

Your contributions will help us push the boundaries of AI and make this library a valuable resource for the community.

Getting Started

Installation

You can install swarms using `pip`:

```bash
pip3 install swarms
```

Alternatively, you can clone the repository:

```bash
git clone https://github.com/kyegomez/swarms
```

Project Structure

- **`swarms/`**: Contains all the source code for the library.
- **`examples/`**: Includes example scripts and notebooks demonstrating how to use the library.
- **`tests/`**: (To be created) Will contain unit tests for the library.
- **`docs/`**: (To be maintained) Contains documentation files.

How to Contribute

Reporting Issues

If you find any bugs, inconsistencies, or have suggestions for enhancements, please open an issue on GitHub:

1. **Search Existing Issues**: Before opening a new issue, check if it has already been reported.
2. **Open a New Issue**: If it hasn't been reported, create a new issue and provide detailed information.
   - **Title**: A concise summary of the issue.
   - **Description**: Detailed description, steps to reproduce, expected behavior, and any relevant logs or screenshots.
3. **Label Appropriately**: Use labels to categorize the issue (e.g., bug, enhancement, documentation).

Submitting Pull Requests

We welcome pull requests (PRs) for bug fixes, improvements, and new features. Please follow these guidelines:

1. **Fork the Repository**: Create a personal fork of the repository on GitHub.
2. **Clone Your Fork**: Clone your forked repository to your local machine.

   ```bash
   git clone https://github.com/kyegomez/swarms.git
   ```

3. **Create a New Branch**: Use a descriptive branch name.

   ```bash
   git checkout -b feature/your-feature-name
   ```

4. **Make Your Changes**: Implement your code, ensuring it adheres to the coding standards.

5. **Add Tests**: Write tests to cover your changes.

6. **Commit Your Changes**: Write clear and concise commit messages.

   ```bash
   git commit -am "Add feature X"
   ```

7. **Push to Your Fork**:

   ```bash
   git push origin feature/your-feature-name
   ```

8. **Create a Pull Request**:

   - Go to the original repository on GitHub.
   - Click on "New Pull Request".
   - Select your branch and create the PR.
   - Provide a clear description of your changes and reference any related issues.

9. **Respond to Feedback**: Be prepared to make changes based on code reviews.

Note: It's recommended to create small and focused PRs for easier review and faster integration.

Coding Standards

To maintain code quality and consistency, please adhere to the following standards.

Type Annotations

- **Mandatory**: All functions and methods must have type annotations.
- **Example**:

  ```python
  def add_numbers(a: int, b: int) -> int:
      return a + b
  ```

- **Benefits**:
  - Improves code readability.
  - Helps with static type checking tools.

Docstrings and Documentation

- **Docstrings**: Every public class, function, and method must have a docstring following the [Google Python Style Guide](http://google.github.io/styleguide/pyguide.html#38-comments-and-docstrings) or [NumPy Docstring Standard](https://numpydoc.readthedocs.io/en/latest/format.html).

- **Content**:
  - **Description**: Briefly describe what the function or class does.
  - **Args**: List and describe each parameter.
  - **Returns**: Describe the return value(s).
  - **Raises**: List any exceptions that are raised.

- **Example**:

  ```python
  def calculate_mean(values: List[float]) -> float:
      """
      Calculates the mean of a list of numbers.

      Args:
          values (List[float]): A list of numerical values.

      Returns:
          float: The mean of the input values.

      Raises:
          ValueError: If the input list is empty.
      """
      if not values:
          raise ValueError("The input list is empty.")
      return sum(values) / len(values)
  ```

- **Documentation**: Update or create documentation pages if your changes affect the public API.

Testing

- **Required**: All new features and bug fixes must include appropriate unit tests.
- **Framework**: Use `unittest`, `pytest`, or a similar testing framework.
- **Test Location**: Place tests in the `tests/` directory, mirroring the structure of `swarms/`.
- **Test Coverage**: Aim for high test coverage to ensure code reliability.
- **Running Tests**: Provide instructions for running tests.

  ```bash
  pytest tests/
  ```

Code Style

- **PEP 8 Compliance**: Follow [PEP 8](https://www.python.org/dev/peps/pep-0008/) style guidelines.
- **Linting Tools**: Use `flake8`, `black`, or `pylint` to check code style.
- **Consistency**: Maintain consistency with the existing codebase.

Areas Needing Contributions

We have several areas where contributions are particularly welcome.

Writing Tests

- **Goal**: Increase test coverage to ensure the library's robustness.
- **Tasks**:
  - Write unit tests for existing code in `swarms/`.
  - Identify edge cases and potential failure points.
  - Ensure tests are repeatable and independent.

Improving Documentation

- **Goal**: Maintain clear and comprehensive documentation for users and developers.
- **Tasks**:
  - Update docstrings to reflect any changes.
  - Add examples and tutorials in the `examples/` directory.
  - Improve or expand the content in the `docs/` directory.

Creating Multi-Agent Orchestration Methods

- **Goal**: Provide new multi-agent orchestration methods

Community and Support

- **Communication**: Engage with the community by participating in discussions on issues and pull requests.
- **Respect**: Maintain a respectful and inclusive environment.
- **Feedback**: Be open to receiving and providing constructive feedback.

License

By contributing to swarms, you agree that your contributions will be licensed under the [MIT License](LICENSE).

Thank you for contributing to swarms! Your efforts help make this project better for everyone.

If you have any questions or need assistance, please feel free to open an issue or reach out to the maintainers.