User Developer Guide

Edit New page

Jump to bottom

Aditi edited this page 2 days ago · 13 revisions

User and Developer Guide

Welcome to the User and Developer Guide for the AMOS SS 2025 AI-Driven Testing project! This guide provides information on how to use the application and how to contribute to its development.

Table of Contents

1. For Users

This section is for anyone who wants to use the AI-Driven Testing application to generate test code.

1.1 What is this Project?

This project aims to leverage Large Language Models (LLMs) to automatically generate test code for existing software. You can provide a piece of code, and the system will attempt to create relevant unit tests for it using an AI model of your choice (from a list of supported models). It aims to simplify the process of creating initial test suites or extending existing ones by leveraging the code understanding and generation capabilities of various Large Language Models.

The primary way to interact with the system is through a web interface, which communicates with a backend service that manages the LLMs.

1.2 Getting Started as a User

There are a couple of ways to get the application running locally. The recommended method for most users is using Docker Compose as it simplifies the setup of all components. An alternative method involves setting up the backend with Conda and running it directly, which might be preferred by users with more technical experience or specific needs.

Common Prerequisites for All Users

- **Git:** You'll need Git to download (clone) the project files from GitHub.
  - Install Git (if you don't have it already).
- **Docker: Docker is essential for this project** as it's used to run the AI models (via Ollama). Please ensure Docker is installed and running on your machine.
  - Install Docker Desktop (for Windows or macOS) or Docker Engine (for Linux).
- **Web Browser:** A modern web browser (like Chrome, Firefox, Edge, Safari) to access the web interface.

Option 1: Running the Full Application with Docker Compose (Recommended)

This is the easiest way to run the complete application (frontend and backend).

1. **Clone the Repository (if not already done):** Open your terminal or command prompt and run:

```
git clone https://github.com/amosproj/amos2025ss04-ai-driven-testing.git
cd amos2025ss04-ai-driven-testing
```

2. **Start the Application:** From the **root directory** of the cloned project (where `docker-compose.yml` is located), run:

```
docker-compose up -d
```

- This command builds the Docker images for frontend and backend (if needed) and starts them.
- The first run might take time to download images and build. Subsequent starts are faster.
- Wait a few minutes for services to initialize, especially the backend which might download AI models on its first run.

3. **Accessing the Web Interface:** Open your web browser and navigate to: http://localhost:3000 The backend API will be at `http://localhost:8000` (Swagger docs: http://localhost:8000/docs).

4. **Stopping the Application:** From the project root directory, run:

```
docker-compose down
```

Option 2: Running Backend with Conda and Frontend Separately (Advanced User / Specific Cases)

This option allows you to run the backend Python application directly using a Conda environment, while still relying on Docker for Ollama. You would typically run the frontend development server separately. This is more involved and generally recommended if you have a reason not to use Docker Compose for the backend service itself.

1. **Prerequisites for this option (in addition to common prerequisites):**
   - **Conda:** For managing the backend's Python environment.
     - Install Anaconda/Miniconda.
   - **Node.js & npm:** For running the frontend development server.
     - Install Node.js.

2. **Clone the Repository (if not already done):**

```
git clone https://github.com/amosproj/amos2025ss04-ai-driven-testing.git
cd amos2025ss04-ai-driven-testing
```

3. **Set up and Run the Backend:**
   - **Create/Update Conda Environment:**

```
# From the project root directory
conda env create -f backend/environment.yml
# If the environment 'backend' already exists and you want to update it:
# conda env update --name backend --file backend/environment.yml --prune
```

- **Activate Conda Environment:**

```
conda activate backend
```

- **Run the Backend API Server:** (Ensure your Docker daemon is running for Ollama management by `LLMManager`)

```
# From the project root directory, while the (backend) Conda environment is v
cd backend
uvicorn api:app --reload --port 8000
```

The backend API will now be running on `http://localhost:8000`.

4. **Set up and Run the Frontend:**

- Open a **new terminal window/tab**.
- Navigate to the frontend directory:

```
# From the project root directory
cd frontend
```

- **Install Frontend Dependencies:**

```
npm install
```

- **Start Frontend Development Server:**

```
npm start
```

This will usually open the web interface in your browser at `http://localhost:3000`. The frontend development server is typically configured to proxy API requests to `http://localhost:8000` (where your backend is running).

5. **Stopping the Services (Manual):**

- To stop the backend Uvicorn server, go to its terminal and press `Ctrl+C`.
- To stop the frontend development server, go to its terminal and press `Ctrl+C`.
- Remember that any Ollama Docker containers started by the `LLMManager` might still be running if not explicitly shut down (e.g., via API calls to `/shutdown` or if the backend script handles cleanup on exit). The `docker-compose down` command (from Option 1) is more comprehensive for stopping everything defined in the `docker-compose.yml`.

1.3 Using the Web Interface

(This section provides a general guide. For detailed UI screenshots and specific operational steps, please refer to the dedicated How to start the Webinterface for dummies: Wiki page if available, or explore the UI once running.)

1. **Overview:** The web interface (accessible at http://localhost:3000 when run locally using either Option 1 or Option 2 above) allows you to interact with the AI to generate tests. *(...rest of section 1.3 as before...)*

1.4 Using the Backend API Directly (For Advanced Users / Integration)

If the backend is running (either via Docker Compose or directly with Conda/Uvicorn), you can interact with its API.

- The backend API, built with FastAPI, automatically generates interactive documentation (Swagger UI), typically accessible at `http://localhost:8000/docs`. *(...rest of section 1.4 as before...)*

---

2. For Developers

This section provides guidance for developers who want to contribute to the project, understand its internals, or set up a development environment.

2.1 Getting Started as a Developer

Prerequisites

- **Git:** For version control.
- **Docker:** Essential for running and managing Ollama containers and for the Dockerized application setup. Install Docker.
- **Conda:** For managing Python environments for the backend. Install Anaconda/Miniconda.
- **Node.js:** For frontend development (includes npm). Install Node.js.
- **Python:** Version specified in `backend/environment.yml` (e.g., 3.13.2).

Cloning the Repository

```
git clone https://github.com/amosproj/amos2025ss04-ai-driven-testing.git
cd amos2025ss04-ai-driven-testing
```

Project Setup (Automated)

The easiest way to set up your development environment is to use the provided `setup.sh` script (run from the project root). Before running, ensure the script is executable:

```
chmod +x setup.sh
./setup.sh
```

This script will:

1. Create or update the Conda environment for the backend (named `backend`, as defined in `backend/environment.yml`).
2. Install Node.js dependencies for the frontend (by running `npm install` in the `frontend/` directory).

After running `setup.sh`, remember to activate the Conda environment for backend work:

```
conda activate backend
```

Pre-commit Hooks

This step is crucial for maintaining code consistency across contributions. This project uses pre-commit hooks to maintain code quality (formatting with Black, linting with Flake8, and running Pytests). It's highly recommended to install and use them:

1. Ensure pre-commit is installed: `pip install pre-commit` (preferably in your global Python or a shared tools environment).
2. Install the hooks (run from the project root): `pre-commit install`

Now, the hooks will run automatically before each commit.

2.2 Understanding the Project Structure

(For a detailed visual and component interaction overview, please see the Architecture Wiki page.)

The project is a monorepo containing several key parts:

- **Root Directory:** Contains overall project configurations (e.g., `.gitignore`, `.dockerignore`), service orchestration (`docker-compose.yml`), developer setup (`setup.sh`), code quality tools configurations (`pyproject.toml`, `.flake8`, `.pre-commit-config.yaml`), main test runner configuration (`pytest.ini`), and the main `README.md`.
- **`frontend/`**: The React/TypeScript frontend application. Uses React, TypeScript, Material-UI, and Emotion. It has its own `Dockerfile` for building a production image that serves static assets. Key files: `package.json` (dependencies, scripts), `src/` (React components), `public/` (static assets like `index.html`).
- **`backend/`**: The Python/FastAPI backend service that manages LLMs via Ollama. Key class `LLMManager` in `llm_manager.py` handles Dockerized Ollama instances. Uses `ollama-models/` (mounted as a volume, ignored by Git) for persistent Ollama model storage.
  - `Dockerfile`: For building the backend API image.

  - `api.py`: FastAPI application definition and API endpoints.
  - `schemas.py`: Pydantic data models for API requests/responses.
  - `environment.yml`: Conda environment definition.
  - `allowed_models.json`: Configuration for supported LLMs.
  - `module_manager.py`: Enables a plugin system for pre/post-processing prompts and responses via modules in a `backend/modules/` subdirectory.
  - `cli.py`, `execution.py`, `main.py`: For CLI interaction and orchestrating the LLM pipeline.
- **`ai-driven-testing/`**: This directory contains the primary Python application (built with Flask, NumPy, Pandas) that this project aims to generate tests for using the AI/LLM backend.
- **`ExampleTests/`**: This directory is specifically for the research and development aspect of **generating unit tests using various LLMs for a predefined set of Python programs.**
  - `pythonTestPrograms/`: Contains `correct_python_programs/` (targets for test generation), `faulty_python_programs/`, and `python_testcases/` (which includes reference `unittest` scripts using `load_testdata.py` from JSON test cases).
  - `Scripts/`: Python scripts to automate these test generation experiments (e.g., `run_test_creation_for_all_models.py`).
  - `generatedTests/`: Stores the output (generated tests) from various LLMs.
  - `llm_list.txt`, `README_tests`: Configuration and methodology for these experiments.
- **`python-test-cases/` (Root level):** Contains simple, standalone Python scripts (`test_case_one.py` to `test_case_five.py`) likely used for basic demonstrations or initial testing of AI capabilities.

2.3 Development Workflows

Running the Full Stack (Frontend + Backend)

The recommended way for integrated development is using Docker Compose:

```
docker-compose up
```

(Remove `-d` to see logs from both frontend and backend services).

- Frontend will be at `http://localhost:3000`.
- Backend API will be at `http://localhost:8000` (Swagger docs at `http://localhost:8000/docs`). Changes to frontend or backend code might require rebuilding the respective image (`docker-compose build <service_name>`) or restarting the service, depending on how live-reloading is configured within the containers (Uvicorn's `--reload` for backend and React's Fast Refresh for frontend are typically used).

Backend Development

1. Activate the Conda environment: `conda activate backend`.
2. Navigate to the `backend/` directory.
3. Run the FastAPI server directly using Uvicorn:
   ```
   uvicorn api:app --reload --port 8000
   ```
   This provides hot-reloading for backend code changes.
4. When running `uvicorn api:app --reload` directly, ensure your local Docker daemon is running, as `LLMManager` will attempt to control Ollama Docker containers. You might need to manually pull Ollama models if not using the full Docker Compose setup which automates this via `LLMManager` calls triggered by API usage.

Frontend Development

1. Navigate to the `frontend/` directory.
2. Install dependencies if you haven't already: `npm install`.
3. Start the React development server:
   ```
   npm start
   ```
   This usually opens the application in your browser at `http://localhost:3000` and provides hot-reloading for frontend code changes.
4. Ensure the backend API is running (either via Docker Compose or directly on port 8000) for the frontend to function fully. The frontend development server will typically proxy API requests to the backend.

2.4 Code Quality & Testing

- **Formatting:** Use `black` for Python code formatting. It's integrated into pre-commit hooks. Configuration is in `pyproject.toml`. To run manually:
  ```
  black .
  ```
- **Linting:** Use `flake8` for Python linting. Also integrated into pre-commit. Configuration is in `.flake8`. To run manually:
  ```
  flake8 .
  ```
- **Running Tests:**
  - **For the `ai-driven-testing/` application:** Pytest is used. Configuration is in `pytest.ini`.
    ```
    pytest
    ```

(Run from the root directory, or as configured in `pytest.ini` to target `ai-driven-testing/tests`).

- **For `ExampleTests/` and `python-test-cases/`:**
  - The test generation scripts in `ExampleTests/Scripts/` are run directly, like `python ExampleTests/Scripts/main.py [args...]`.
  - The reference tests for the algorithms in `ExampleTests/pythonTestPrograms/python_testcases/` (e.g., `test_bitcount.py`) are `unittest`-based and can be run as `python ExampleTests/pythonTestPrograms/python_testcases/test_bitcount.py`.
  - Standalone examples in `python-test-cases/` are run directly, e.g., `python python-test-cases/test_case_one.py`.
- Pre-commit hooks will also attempt to run `pytest -q` on every commit (targeting tests defined in `pytest.ini`).

2.5 Contributing

- Follow the established code style (enforced by Black and Flake8).
- Ensure all tests pass before committing/pushing (pre-commit hooks help with this).
- For new features or bug fixes, consider creating an issue first to discuss the changes.
- Submit changes via Pull Requests.
- Refer to specific contribution guidelines if they exist on other Wiki pages (e.g., "How to be an AMOS Release Manager").

2.6 Key Technologies & Tools Summary

- **Backend:** Python, FastAPI, Uvicorn, Conda, Docker, Ollama, `docker-py` (Python Docker SDK), Pydantic, (potentially LangChain for advanced LLM workflows).
- **Frontend:** React, TypeScript, Node.js/npm, Material-UI, Emotion.
- **LLMs:** Various open-source models managed via Ollama (e.g., Mistral, Gemma, DeepSeek, Qwen, Phi4 - see `backend/allowed_models.json`).
- **Testing Frameworks:** `unittest` (for LLM-generated tests and reference tests in `ExampleTests`), `pytest` (for the `ai-driven-testing/` application).
- **Code Quality:** Black (formatter), Flake8 (linter), Pre-commit (git hooks).
- **Version Control:** Git.
- **Orchestration:** Docker Compose.

---

3. Troubleshooting

- **`docker-compose up` fails:**
  - Ensure Docker Desktop (or Docker Engine) is running.

    - Check for port conflicts (e.g., if port 3000 or 8000 is already in use on your host).
    - Look at the error messages from Docker Compose for specific issues (e.g., Dockerfile errors, network problems).
- **Backend API (`localhost:8000`) not reachable:**
    - Check the logs of the `backend` service: `docker-compose logs backend`.
    - Ensure the Conda environment inside the backend container was set up correctly.
    - Verify Ollama containers (if managed by `LLMManager`) are starting correctly. Check `LLMManager` output if running the backend directly.
- **Frontend (`localhost:3000`) shows errors or doesn't connect to backend:**
    - Check browser developer console for errors.
    - Check logs of the `frontend` service: `docker-compose logs frontend`.
    - Ensure the backend API is running and accessible from the frontend container (Docker Compose network `backend` should handle this via service name `http://backend:8000`).
- **Conda environment issues:**
    - Ensure `backend/environment.yml` is correctly formatted.
    - Try removing and recreating the environment: `conda env remove -n backend` then re-run `setup.sh` or `conda env create -f backend/environment.yml`.
- **LLM Issues:**
    - If `LLMManager` has trouble pulling models or starting Ollama:
        - Check your internet connection.
        - Ensure you have enough disk space for Ollama models.
        - Check Docker daemon logs and `LLMManager` output for errors.
        - Consult Ollama's own documentation for issues with specific models. Try pulling the model manually via the Ollama CLI first (`ollama pull <model_id>`) to isolate issues.

---

4. Further Information

For more detailed information on specific components or aspects of the project, please refer to the following Wiki pages and project README files:

- **Project Overview & Goals:** Home (Wiki Main Page), Main Project `README.md`
- **System Architecture:** Architecture, `docker-compose.yml` (for service orchestration)
- **Backend Details:** `backend/README.md`, Possible Ways to run Ollama
    - API Endpoints: `http://localhost:8000/docs` (when backend is running)
- **Frontend Details:** How to start the Webinterface for dummies:
- **AI Test Generation Experiments:** `ExampleTests/README_tests`
- **LLM Information:** Language Models Considered, LLMs incompatibility with our project, LLM Research
- **Continuous Integration:** CLI Pipeline

- **Code Quality & Conventions:** `pyproject.toml` (Black), `.flake8` (Flake8), `.pre-commit-config.yaml`

```
https://github.com/amosproj/amos2025ss04-ai-driven-testing.wiki.git
```