Appendix B

User Guide

B.1 Dependencies

To get started with using ‘Athena‘, ensure that you have either Conda or Miniconda installed and have downloaded the repository.

Next, we will create an environment:

B.1.1 For Macs with Apple Silicon

To ensure that the accelerated `tensorflow-metal` library is used, run the following commands inside the repository:

```
CONDA_SUBDIR=osx-arm64 conda create -n athena python=3.9
conda activate athena
pip install -r requirements.macos.txt
```

B.1.2 For other computers

To create an environment on other computers, run the following commands inside the repository:

```
conda create -n athena python=3.9
conda activate athena
pip install -r requirements.txt
```

Athena can be used once pip notifies you that all packages have been successfully installed. Note, due to different architectures and systems, the requirements may have to be modified. However, Athena was developed using Python 3.9 and Tensorflow 2.11.0, so any other versions may produce unexpected outputs.

B.2 Commands

Any time you intend to interact with Athena in a new terminal instance, please remember to activate your environment with `conda activate athena`.

B.2.1 Generate

The generate command generates mutants for a given subejct, accessed by running `python main.py generate`. Access the following help menu by running `python main.py generate --help`:

```
Usage: main.py generate [OPTIONS] SUBJECT_NAME

  Generates mutant for subject.

Options:
  -t, --trained-models-dir TEXT  Directory to load/save trained models.
  -m, --mutants_dir TEXT         Directory to save mutated models.
  -p, --specific-output TEXT     Specific output to generate mutants for.
  -o, --additional-config TEXT   Path to additional configuration json file or
                                 json string.
  -v, --verbose                  Enable verbose output
  --help                         Show this message and exit.
```

B.2.2 Run

The run command executes mutation testing on a test set for a given subject, accessed by running `python main.py run`. Access the following help menu by running `python main.py run --help`:

```
Usage: main.py run [OPTIONS] SUBJECT_NAME TEST_SET

  Runs example test set on subject.

Options:
  -t, --trained-models-dir TEXT  Directory to load/save trained models.
  -m, --mutants_dir TEXT         Directory to load/save mutated models.
  -p, --specific-output TEXT     Specific output to generate mutants for.
  -o, --additional-config TEXT   Path to additional configuration json file or
                                 json string.
  -v, --verbose                  Enable verbose output
  --help                         Show this message and exit.
  ↪  Show this message and exit.
```

B.2.3 Evaluate

The evaluate command is used for accessing the killability of mutants generated by an operator on a given model. It can be accessed by running `python main.py evaluate` and displays the following help menu when `python main.py run --help` is run:

```
Usage: main.py evaluate [OPTIONS] SUBJECT_NAME

  Evaluates a given operator by retraining the model, generating a mutant and
  measuring the effect size of the mutation.

Options:
  -t, --trained-models-dir TEXT  Directory to load/save trained models.
  -m, --mutants_dir TEXT         Directory to save mutated models.
  -p, --specific-output TEXT     Specific output to generate mutants for.
```

```
  -o, --additional-config TEXT   Path to additional configuration json file or
                                 json string.
  -v, --verbose                  Enable verbose output
  --help                         Show this message and exit.
```

B.3 Additional Configuration

To access more specific configuration options, you can use the `--additional-config` flag to specify either a JSON file or string. The specification of this configuration is given by the JSON schema found at the root of the project. The `config/` directory contains a set of pre-made configuration files for the following uses:

- `athena.json`: default configuration options for Athena operator with DE plot shown
- `athena_multiprocessing.json`: default configuration options for Athena operator with max CPU workers in DE algorithm
- `athena_multiprocessing_fast.json`: configuration options for Athena operator with max CPU workers and reduced tolerance in DE algorithm
- `athena_multiprocessing_no_generic.json`: configuration options for Athena operator with max CPU workers where generic inputs are provided no weighting

B.4 Extensibility

Athena was developed with extensibility in mind, hence, as a software developer you are able to add your own custom models, test cases and further mutation operators.

B.4.1 Creating new models

To create mutants for a model which is not already defined in `models/`, you can simply create a new file `models/<model_name>.py` which defines a class which extends the `Model` class (defined in `models/model.py`). The new model can then be referenced by `<model_name>` where a 'subject name' is required. Athena should then dynamically import your model.

The existing example classes and base class `model` provide a good indication as to how one would define a new model class. A new model class must implement methods with the following signatures:

- `train(self) -> Sequential`: trains the model.
- `generate_inputs_outputs(self, model: Sequential, n: int = 20, specific_output: int = None, generic: bool = False )`: generates inputs and outputs for Athena operator.
- `generate_evaluation_data(self, specific_output: int = None, generic: bool = False)`: generates evaluation data for testing the model.

B.4.2 Creating new test cases

To create test sets to assess a new file can be created `test_sets/<test_set_name>.py` which defines a single `TestSet` class, containing one or more `TestCase` classes. A new `TestCase` class must implement methods with the following signatures:

- `run(self, model: Sequential, X: ndarray, Y: ndarray) -> float`: runs a test upon the model.
- `test_passed(self, test_result: float) -> bool`: outputs true if the test in `run` passed.

A new `TestSet` class must contain an attribute `test_cases`, a list of TestCases to run. The new test set can then be referenced by `<test_set_name>` where a 'test set' is required.

B.4.3 Creating new operators

To create a new operator, a new file can be created `operators/<operator_name>.py` which defines a single `Operator` class. This class takes in `model` and `additional_config` and implements the `__call__` method, which generates a mutant from `model`. The new operator can then be referenced in the configuration property `operator.name`