
rlscope

Release 1.0.0

James Gleeson

Jan 27, 2021

CONTENTS:

1	RL-Scope Artifact Evaluation	3
1.1	1. Machine configuration	3
1.2	2. Running the Docker development environment	3
1.3	3. Building RL-Scope	4
1.4	4. Installing experiments	4
1.5	5. Running experiments	5
2	Host Configuration	9
2.1	1. Install docker-compose	9
2.2	2. NVIDIA driver	9
2.3	3. Docker default runtime	10
2.4	4. Running the Docker development environment	10
3	Unit tests	13
3.1	Running unit tests	13
3.2	Python unit tests	15
3.3	C++ unit tests	15
4	Source documentation	17
5	Documentation TODOs	19
	Index	21

RL-Scope is a cross-stack profiler for deep reinforcement learning workloads.

For a tutorial on reproducing figures in the RL-Scope paper, see [*RL-Scope Artifact Evaluation*](#).

For information on running the RL-Scope development docker container, see [*Host Configuration*](#).

RL-SCOPE ARTIFACT EVALUATION

This is a tutorial for reproducing figures in the RL-Scope paper. To ease reproducibility, all experiments will run within a Docker development environment.

1.1 1. Machine configuration

Generally speaking, RL-Scope works on multiple GPUs models. The only limitation is that you need to use a GPU that supports the newer “CUPTI Profiling API”. [NVIDIA’s documentation](#) states that Volta and later GPU architectures (i.e., devices with compute capability 7.0 and higher) should support this API. If you attempt to use a GPU that is unsupported, the Docker build will fail, since we check for CUPTI Profiling API compatibility using a sample program (see [dockerfiles/sh/test_cupti_profiler_api.sh](#)).

The machine we used in the RL-Scope paper had a NVIDIA 2080Ti GPU. We have also reproduced results on an AWS [g4dn.xlarge instance](#) which contains a T4 GPU.

1.1.1 AWS

As mentioned above, we have reproduced results on an AWS [g4dn.xlarge instance](#) which contains a single T4 GPU. Please note that the other AWS instances that have more than one GPU are also fine (e.g., [g4dn.12xlarge](#), [p3.8xlarge](#)), and will simply run the experiments faster by using multiple GPUs in parallel.

To make setup as simple as possible, we used [NVIDIA’s Deep Learning AMI](#) to create an VM instance, which comes preinstalled with Ubuntu 18.04, Docker, and an NVIDIA driver. If you wish to use a different OS image, just make sure you install the NVIDIA driver and Docker.

Regardless of the starting OS image you use, there is still some host setup that is required (which will be discussed in the next section).

1.2 2. Running the Docker development environment

In order to run the Docker development environment, you must first perform a one-time configuration of your host system, then use `run_docker.py` to build/run the RL-Scope container. To do this, follow **all** the instructions at [Host Configuration](#). Afterwards, you should be running inside the RL-Scope container, which looks like this:



All remaining instructions will run commands inside this container, which we will emphasize with `[container]$`.

1.3 3. Building RL-Scope

RL-Scope uses a C++ library to collect CUDA profiling information (`librlscope.so`), and offline analysis of collected traces is performed using a C++ binary (`rls-analyze`)

To build the C++ components, run the following:

```
[container]$ build_rlscope
```

1.4 4. Installing experiments

The experiments in RL-Scope consist of taking an existing RL repository and adding RL-Scope annotations to it. In order to clone these repositories and install them using `pip`, run the following:

```
[container]$ install_experiments
```


1.5 5. Running experiments

The RL-Scope paper consists of several case studies. Each case study has its own shell script for reproducing figures from that section. The shell script will collect traces from each relevant algorithm/simulator/framework, then generate a figure seen in the paper in a corresponding subfolder `output/artifacts/*` of the RL-Scope repository.

1.5.1 RL Framework Comparison

This will reproduce results from the “Case Study: Selecting an RL Framework” section from the RL-Scope paper; In particular, the “RL framework comparison” figures, shown below for reference:

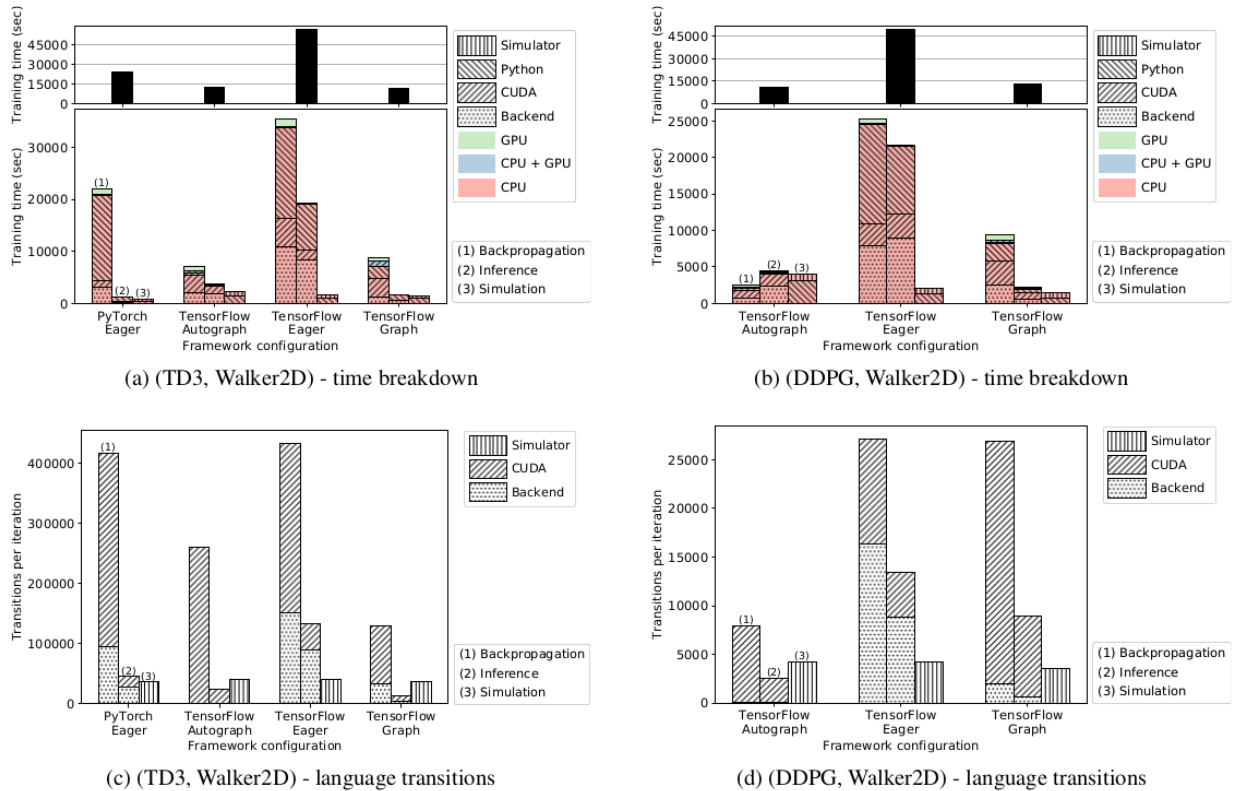


Figure 4. *RL framework comparison*: We used identical RL algorithm (left: TD3, right: DDPG), simulator (Walker2D), and tuned hyperparameters; differences in execution between RL frameworks are strictly due to RL algorithm implementation and ML backend. Differences in *time breakdown* (top) across RL frameworks can be explained by higher number of *language transitions* (bottom) between the Python and the ML backend (*Backend*), and between the ML backend and the accelerator API calls (*CUDA*).

To run the experiment and generate the figures, run:

```
[container]$ experiment_RL_framework_comparison.sh
```

Figures will be output to `output/artifacts/experiment_RL_framework_comparison/*.pdf`.

1.5.2 RL Algorithm Comparison

This will reproduce results from the “Case Study: RL Algorithm and Simulator Survey” section from the RL-Scope paper; In particular, the “Simulator choice” figures, shown below for reference:

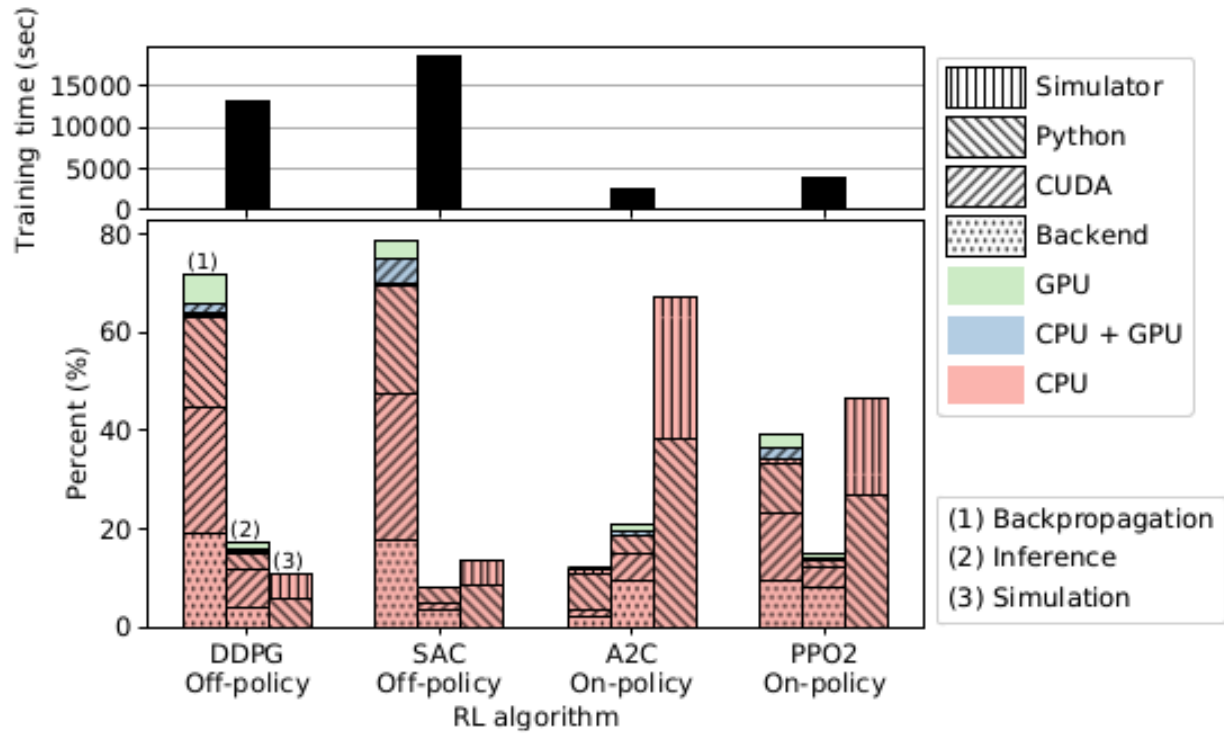


Figure 5. Algorithm choice: We used a popular RL environment (Walker2D: robotics task of a walking humanoid) measured how the stages (backpropagation, inference, simulation) of each measured algorithm change with respect to algorithm choice. All tested RL workloads spend about 90% of their runtime purely in the CPU.

To run the experiment and generate the figures, run:

```
[container]$ experiment_algorithm_choice.sh
```

Figures will be output to `output/artifacts/experiment_algorithm_choice/*.pdf`.

1.5.3 Simulator Comparison

This will reproduce results from the “Case Study: Simulator Survey” section from the RL-Scope paper; In particular, the “Simulator choice” figures, shown below for reference:

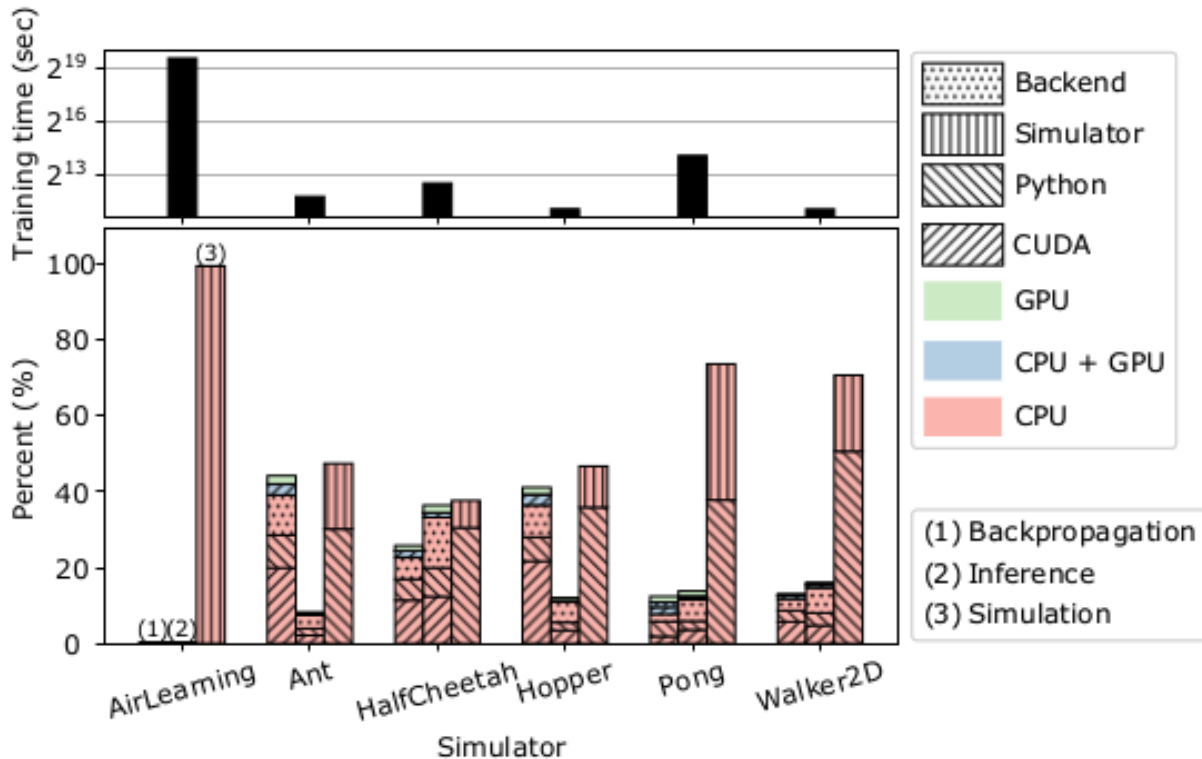


Figure 7. Simulator choice: We used a top-performing RL algorithm (PPO) and measured how each stage of the algorithm changes with respect to environment choice. GPU accounts for 5% or less of the runtime across all simulators.

To run the experiment and generate the figures, run:

```
[container]$ experiment_simulator_choice.sh
```

Figures will be output to `output/artifacts/experiment_simulator_choice/*.pdf`.

NOTE: Your reproduced graph will have a slightly different breakdown for Pong than seen above from the RL-Scope paper; in particular the simulation time will be closer to HalfCheetah. This is likely due to a difference in library version for the `atari-py` backend simulator used by Pong.

HOST CONFIGURATION

In order to run the Docker development environment, you must perform a one-time configuration of your host system. In particular:

1. *Install docker-compose*: install docker and docker-compose.
2. *NVIDIA driver*: allow non-root users to access GPU hardware counters.
3. *Docker default runtime*: make GPUs available to all containers by default.

After you've configured your host system, you can launch the RL-Scope docker container:

4. *Running the Docker development environment*: build and run the container.

2.1 1. Install docker-compose

If your host does not yet have docker installed yet, follow the [instructions on DockerHub for Ubuntu](#).

Make sure you are part of the docker UNIX group:

```
[host]$ sudo usermod -aG docker $USER
```

NOTE: if you weren't already part of the docker group, you will need to logout/login for changes to take effect.

Next, we need to install docker-compose. To install docker-compose into /usr/local/bin/docker-compose, do the following:

```
[host]$ DOCKER_COMPOSE_INSTALL_VERSION=1.27.4
[host]$ sudo curl -L "https://github.com/docker/compose/releases/download/$
↪ {DOCKER_COMPOSE_INSTALL_VERSION}/docker-compose-$(uname -s)-$(uname -m)" -
↪ o /usr/local/bin/docker-compose
[host]$ sudo chmod ugo+rx /usr/local/bin/docker-compose
```

2.2 2. NVIDIA driver

By default, the nvidia kernel module doesn't allow non-root users to access GPU hardware counters. To allow non-root user access, do the following:

1. Paste the following contents into /etc/modprobe.d/nvidia-profiler.conf:

```
options nvidia NVreg_RestrictProfilingToAdminUsers=0
```

2. Reboot the machine for the changes to take effect:

```
[host]$ sudo reboot now
```

2.3 3. Docker default runtime

By default, GPUs are inaccessible during image builds and within containers launched by `docker-compose`. To fix this, we can make `--runtime=nvidia` the default for all containers on the host. To do this, do the following:

1. Stop docker and any running containers:

```
[host]$ sudo service docker stop
```

2. Paste the following contents into `/etc/docker/daemon.json`:

```
{
  "default-runtime": "nvidia",
  "runtimes": {
    "nvidia": {
      "path": "/usr/bin/nvidia-container-runtime",
      "runtimeArgs": []
    }
  }
}
```

3. Restart docker:

```
[host]$ sudo service docker start
```

2.4 4. Running the Docker development environment

The `run_docker.py` python script is used for building and running the docker development environment. In order to run this script on the host, you need to install some minimal “deployment” pip dependencies (`requirements.deploy.txt`).

First, on the **host** run the following (replacing `[rlscope-root]` with the directory of your RL-Scope repository):

```
# Install python3/virtualenv on host
[host]$ sudo apt install python3-pip python3-virtualenv

# Create python3 virtualenv on host
[host]$ cd [rlscope-root]
[host]$ python3 -m virtualenv -p /usr/bin/python3 ./venv
[host]$ source ./venv/bin/activate
[host (venv)]$ pip install -r requirements.deploy.txt

# Build and run RL-Scope the docker development environment
[host (venv)]$ cd [rlscope-root]
[host (venv)]$ python run_docker.py
```

After the container is built, it will run and you should be greeted with the welcome banner:

A terminal window with a dark background. At the top, 'RL-Scope toolkit' is written in large, red, outlined letters. Below it, 'dev env' is written in a similar red, outlined font. A yellow text block explains the user ID and group mapping. Below that, a yellow text block provides help information and a command. At the bottom, a green prompt shows the user 'jgleeson' at a specific container ID with a tilde and dollar sign.

```
RL-Scope toolkit  
dev env  
  
You are running this container as user with ID 1012 and group 1012,  
which should map to the ID and group for your user on the Docker host.  
  
For help on available RL-Scope commands, run:  
$ rlscope_help  
  
jgleeson@4c802d88ef4f:~$
```

If you wish to restart the container in the future, you can do:

```
[host]$ cd [rlscope-root]  
[host]$ source ./venv/bin/activate  
[host (venv)]$ python run_docker.py
```


UNIT TESTS

3.1 Running unit tests

RL-Scope has both python and C++ unit tests, which can be run either separately or all together.

To run all unit tests (i.e., both python and C++):

```
[container]$ rls-unit-tests
```

To run only C++ unit tests:

```
[container]$ rls-unit-tests --tests cpp
```

Output should look like:

```

jgleeson@B946558367:~/clone$ ./rls-unit-tests --tests cpp
> CMD:
$ rls-test
PWD=/home/jgleeson/clone/rls
Running 24 tests from 11 test cases.
-----
Global test environment set-up.
-----
2 tests from TestCommonUtil
RUN      TestCommonUtil.Test_StringsSplit_01_One
OK      TestCommonUtil.Test_StringsSplit_01_One (0 ms)
RUN      TestCommonUtil.Test_StringsSplit_02_Empty
OK      TestCommonUtil.Test_StringsSplit_02_Empty (0 ms)
-----
2 tests from TestCommonUtil (0 ms total)
-----
2 tests from TestIdMap
RUN      TestIdMap.Test_BitsetAdd
OK      TestIdMap.Test_BitsetAdd (0 ms)
RUN      TestIdMap.Test_OverlapsData
[2021-01-15 15:57:59.612] [info] [trace file_parser.cc:496] pid=10956 @ operator(): Preallocate(eo.times) category=CategoryKey(pr
oscs="process"), ops={"C2"}, non_ops={}, n_events=2
[2021-01-15 15:57:59.612] [info] [trace file_parser.cc:496] pid=10956 @ operator(): Preallocate(eo.times) category=CategoryKey(pr
oscs="process"), ops={"C2"}, non_ops={}, n_events=2
[2021-01-15 15:57:59.612] [info] [trace file_parser.cc:496] pid=10956 @ operator(): Preallocate(eo.times) category=CategoryKey(pr
oscs="process"), ops={"C2"}, non_ops={}, n_events=2
[2021-01-15 15:57:59.612] [info] [trace file_parser.cc:496] pid=10956 @ operator(): Preallocate(eo.times) category=CategoryKey(pr
oscs="process"), ops={"C2"}, non_ops={}, n_events=2
[2021-01-15 15:57:59.612] [info] [trace file_parser.cc:1901] pid=10956 @ ComputeOverlap: ctimes.size at start = 3, ctimes.size at
end = 3
OK      TestIdMap.Test_OverlapsData (0 ms)
-----
2 tests from TestIdMap (1 ms total)
-----
1 test from TestIdMap
RUN      TestIdMap.Test_01
OK      TestIdMap.Test_01 (0 ms)
-----
1 test from TestIdMap (0 ms total)
-----
1 test from TestComputeOverlap
RUN      TestComputeOverlap.Test_01_Complete
[2021-01-15 15:57:59.612] [info] [trace file_parser.cc:496] pid=10956 @ operator(): Preallocate(eo.times) category=CategoryKey(pr
oscs="process"), ops={"A"}, non_ops={}, n_events=2
[2021-01-15 15:57:59.612] [info] [trace file_parser.cc:496] pid=10956 @ operator(): Preallocate(eo.times) category=CategoryKey(pr
oscs="process"), ops={"A"}, non_ops={}, n_events=2
[2021-01-15 15:57:59.612] [info] [trace file_parser.cc:496] pid=10956 @ operator(): Preallocate(eo.times) category=CategoryKey(pr
oscs="process"), ops={"A"}, non_ops={}, n_events=2
[2021-01-15 15:57:59.612] [info] [trace file_parser.cc:496] pid=10956 @ operator(): Preallocate(eo.times) category=CategoryKey(pr
oscs="process"), ops={"A"}, non_ops={}, n_events=2
[2021-01-15 15:57:59.612] [info] [trace file_parser.cc:1901] pid=10956 @ ComputeOverlap: ctimes.size at start = 3, ctimes.size at
end = 3
OK      TestComputeOverlap.Test_01_Complete (0 ms)
-----
1 test from TestComputeOverlap (0 ms total)
-----
1 test from TestEachMerged
RUN      TestEachMerged.Test_01_Merge
OK      TestEachMerged.Test_01_Merge (1 ms)
-----
1 test from TestEachMerged (1 ms total)
-----
4 tests from TestBitset
RUN      TestBitset.Test_01_30_bits
OK      TestBitset.Test_01_30_bits (0 ms)
-----
4 tests from TestBitset (0 ms total)
-----
Test sizes:
sizeoff(int) = 4
sizeof(unsigned int) = 4
sizeof(unsigned long int) = 8
sizeof(unsigned long long int) = 8
sizeof(int64_t) = 8
-----
OK      TestBitset.Test_01_30_bits (0 ms)
-----
TestBitset.Test_01_31_bits
RUN      TestBitset.Test_01_31_bits (0 ms)
-----
TestBitset.Test_01_31_bit_from_string
RUN      TestBitset.Test_01_31_bit_from_string (0 ms)
-----
TestBitset.Test_01_32_bits
RUN      TestBitset.Test_01_32_bits (0 ms)
-----
4 tests from TestBitset (0 ms total)
-----
2 tests from TestCategoryKey
RUN      TestCategoryKey_MapContainsKey
OK      TestCategoryKey_MapContainsKey (0 ms)
RUN      TestCategoryKey_MapContainsKeyCopy
OK      TestCategoryKey_MapContainsKeyCopy (0 ms)
-----
2 tests from TestCategoryKey (0 ms total)
-----
2 tests from TestEigen
RUN      TestEigen.ElementWiseLessThan
OK      TestEigen.ElementWiseLessThan (0 ms)
RUN      TestEigen.ArrayModifyOne
OK      TestEigen.ArrayModifyOne (0 ms)
-----
2 tests from TestEigen (0 ms total)
-----
1 test from TestPath
RUN      TestPath.TestTraceID
OK      TestPath.TestTraceID (0 ms)
-----
1 test from TestPath (0 ms total)
-----
6 tests from TestProcessing
RUN      TestProcessing.TestEachOpEvent_01
OK      TestProcessing.TestEachOpEvent_01 (0 ms)
RUN      TestProcessing.TestEachOpEvent_02
OK      TestProcessing.TestEachOpEvent_02 (0 ms)
RUN      TestProcessing.TestEachOpEvent_03
OK      TestProcessing.TestEachOpEvent_03 (0 ms)
RUN      TestProcessing.TestEachOpEvent_04
OK      TestProcessing.TestEachOpEvent_04 (0 ms)
RUN      TestProcessing.TestEachOpEvent_05
OK      TestProcessing.TestEachOpEvent_05 (0 ms)
RUN      TestProcessing.TestEventFlattener_01
OK      TestProcessing.TestEventFlattener_01 (0 ms)
-----
6 tests from TestProcessing (0 ms total)
-----
2 tests from TestUtil
RUN      TestUtil.TestKeepExtreme_01
OK      TestUtil.TestKeepExtreme_01 (0 ms)
RUN      TestUtil.TestKeepExtreme_02
OK      TestUtil.TestKeepExtreme_02 (0 ms)
-----
2 tests from TestUtil (0 ms total)
-----
Global test environment tear-down
-----
24 tests from 11 test cases ran. (3 ms total)
PASSED: 24 tests
PID=10952/MainProcess @ run.cpp, rls unit tests.py:91 2021-01-15 15:57:59.615 INFO | RL-Scope cpp unit tests PASSED

```

To run only python unit tests:

```
[container]$ rls-unit-tests --tests py
```

Output should look like:

```

jgleeson@B946558367:~/clone$ ./rls-unit-tests --tests py
> CMD:
$ /home/jgleeson/venv/bin/python -Wignore::: pytest.assertion.rewrite -m pytest
PWD=/home/jgleeson/clone/rls/rlscope
platform linux -- Python 3.6.9, pytest=6.2.1, py-1.10.0, pluggy=0.13.1 -- /home/jgleeson/venv/bin/python
cachedir: .pytest_cache
rootdir: /home/jgleeson/clone/rls/rlscope, configfile: pytest.ini
collecting 2 items
-----
Live log collection
-----
PID=10951/MainProcess @ instance_base_parser.py:30 : 15:58:08 INFO: Loaded ['luigi.cfg']
collected 21 items

experiment/util.py::TestTee::test_tee_01 PASSED                                     (4%)
experiment/util.py::TestTee::test_tee_02 PASSED                                     (9%)
parser/dataframe.py::TestVennOverlapDict::test_venn.as_overlap.dict_01 PASSED      (14%)
parser/dataframe.py::TestVennOverlapDict::test_venn.as_overlap.dict_02 PASSED      (19%)
parser/dataframe.py::TestVennOverlapDict::test_venn.as_overlap.dict_03 PASSED      (24%)
parser/dataframe.py::TestVennOverlapDict::test_venn.as_overlap.dict_04 PASSED      (29%)
parser/dataframe.py::TestVennOverlapDict::test_venn.as_overlap.dict_05 PASSED      (34%)
parser/db.py::test_merge_sorted PASSED                                             (40%)
parser/db.py::TestProcessQueue::test_01_1_stack PASSED                             (45%)
parser/db.py::TestProcessQueue::test_02_2_stacks PASSED                             (50%)
parser/db.py::TestProcessQueue::test_03_multiple_sub_events PASSED                 (55%)
parser/db.py::TestProcessQueue::test_04_wild_data_01 PASSED                         (60%)
parser/db.py::TestProcessQueue::test_05 PASSED                                     (65%)
parser/stacked_bar_plots.py::test_join_plus_01 PASSED                             (70%)
parser/stacked_bar_plots.py::test_join_plus_02 PASSED                             (75%)
parser/tfprof.py::test_merge_adjacent_events PASSED                               (80%)
parser/tfprof.py::test_split PASSED                                                 (85%)
profiler/concurrent.py::TestForkedProcessPool::test_01_sys_exit_1 PASSED           (90%)
profiler/concurrent.py::TestForkedProcessPool::test_02_exception PASSED           (95%)
profiler/concurrent.py::TestForkedProcessPool::test_03_endir PASSED                (100%)

-----
21 passed in 12.75s
-----
PID=10957/MainProcess @ run.py, rls unit tests.py:79 2021-01-15 15:58:10.121 INFO | RL-Scope python unit tests PASSED

```

3.2 Python unit tests

Python unit tests made are using the `pytest` testing framework. Unit tests are written in the same module as the function they are testing. To locate unit tests, search for `def test_` in a file.

3.3 C++ unit tests

C++ unit tests are made using the `gtest` testing framework. Unit tests are any/all files rooted under `test` whose filename matches `test_*. [cc|cpp]`. All unit tests are compiled into the `rls-test` binary.

SOURCE DOCUMENTATION

- [genindex](#)
- [modindex](#)
- [search](#)
- [modules](#)

DOCUMENTATION TODOS

INDEX

M

module
 rlscope, 1

R

rlscope
 module, 1