Estimating Energy Cost

GSoC Mid-Term Evaluation

Manas Pratim Biswas

https://github.com/sanam2405/SoftwareEnergyCost

Progress Till Now :

1. Profiled Baler for **CPU computation** time with **cProfile** and Visualised the results with **SnakeViz** and **yelp-gprof2dot** (**Deterministic Profiling**)

2. Profiled Baler for **CPU computation** on wall clock time (**Sampling Profiling**)

3. Profiled Baler for **memory consumption** with **memory-profiler**

4. Profiled Baler for **power consumption** with **powermetrics** and Visualised results using **influxdb**

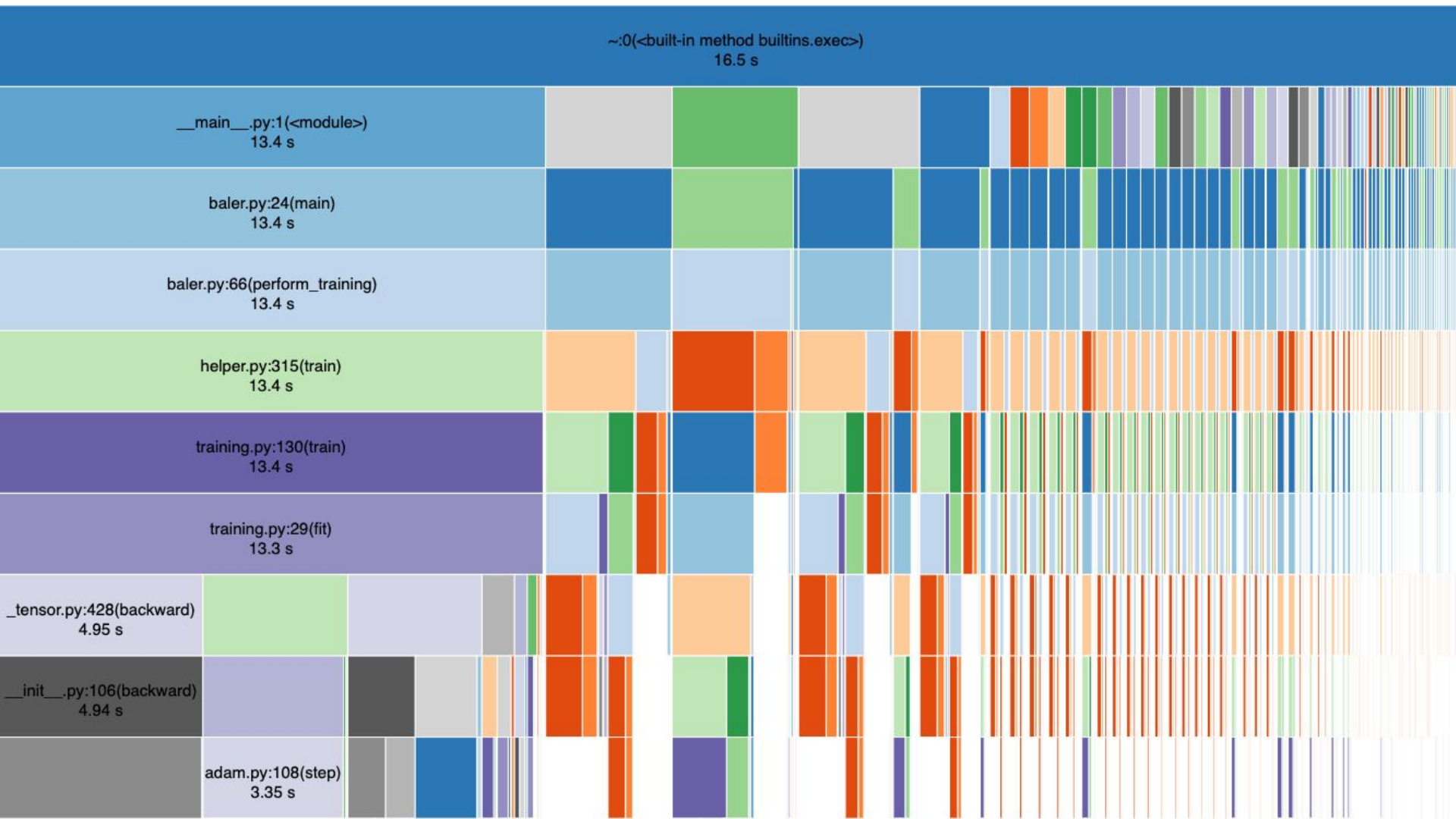
5. Profiled the **CO2 Emissions** and **power consumption** with **codecarbon**

cProfile

Deterministic Profiler. Traces each function call and measures CPU computation.

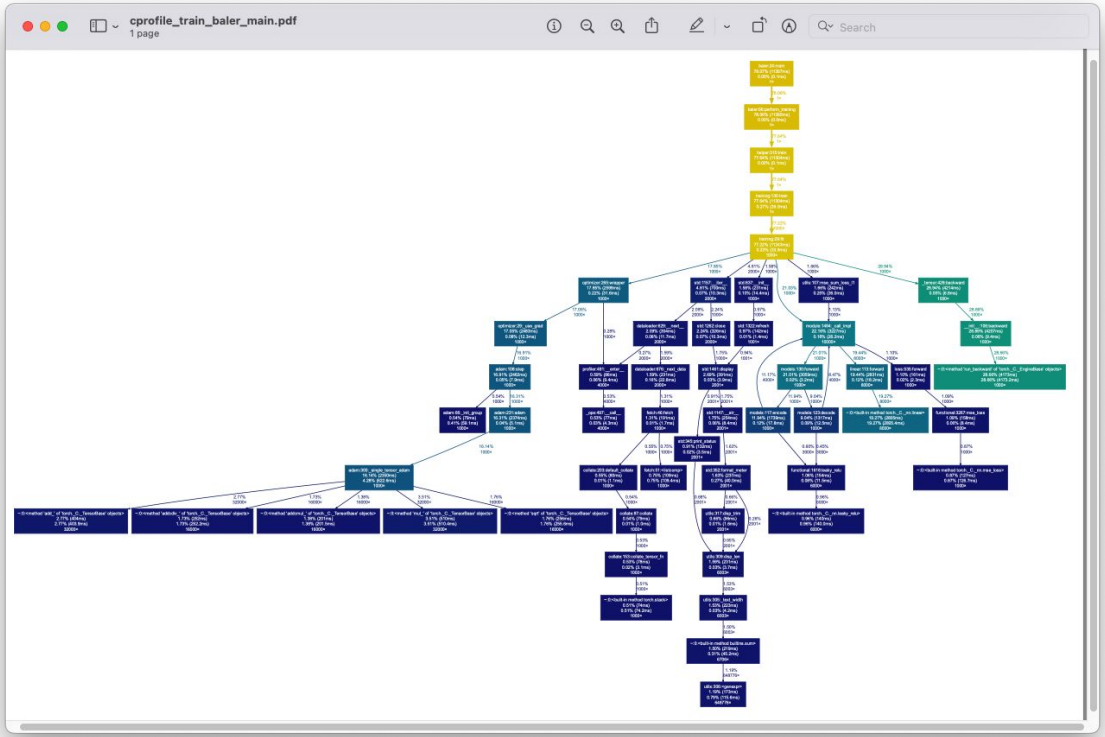
tottime - Total time taken by the method excluding recursive calls
cumtime - Cumulative Time (Time taken by the method including recursive calls)
ncalls - Total number of times the method is invoked (Includes recursive call)
pcall - Total number of primitive calls to the method (Excludes recursive call)

All the results are calculate for 1000 epochs of train, compress and decompress on the CFD dataset. It takes **14.56 seconds** to run baler on this configuration on my Apple M1 silicon.


~:0(<built-in method builtins.exec>)
16.5 s
__main__.py:1(<module>)
13.4 s
baler.py:24(main)
13.4 s
baler.py:66(perform_training)
13.4 s
helper.py:315(train)
13.4 s
training.py:130(train)
13.4 s
training.py:29(fit)
13.3 s
_tensor.py:428(backward)
4.95 s
__init__.py:106(backward)
4.94 s
adam.py:108(step)
3.35 s

cProfile

- DI Graph generated while Training Baler
- Provides the exact call order for the methods
- Shows an exact breakdown of time distribution across various parts of the code



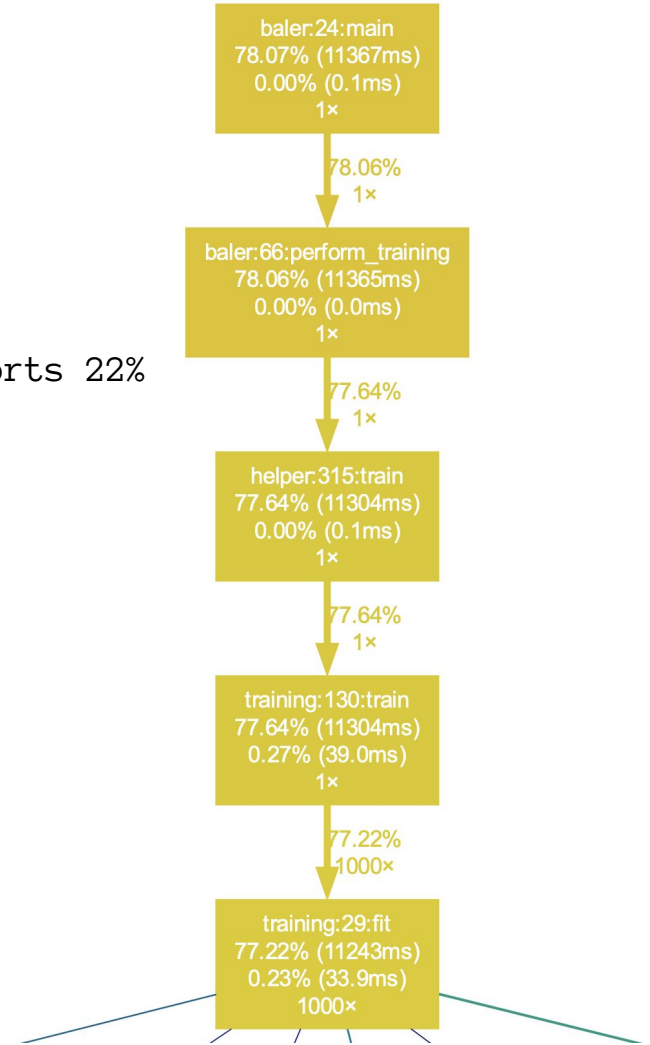
Execution of Baler(train)

Total time taken 14.56 second

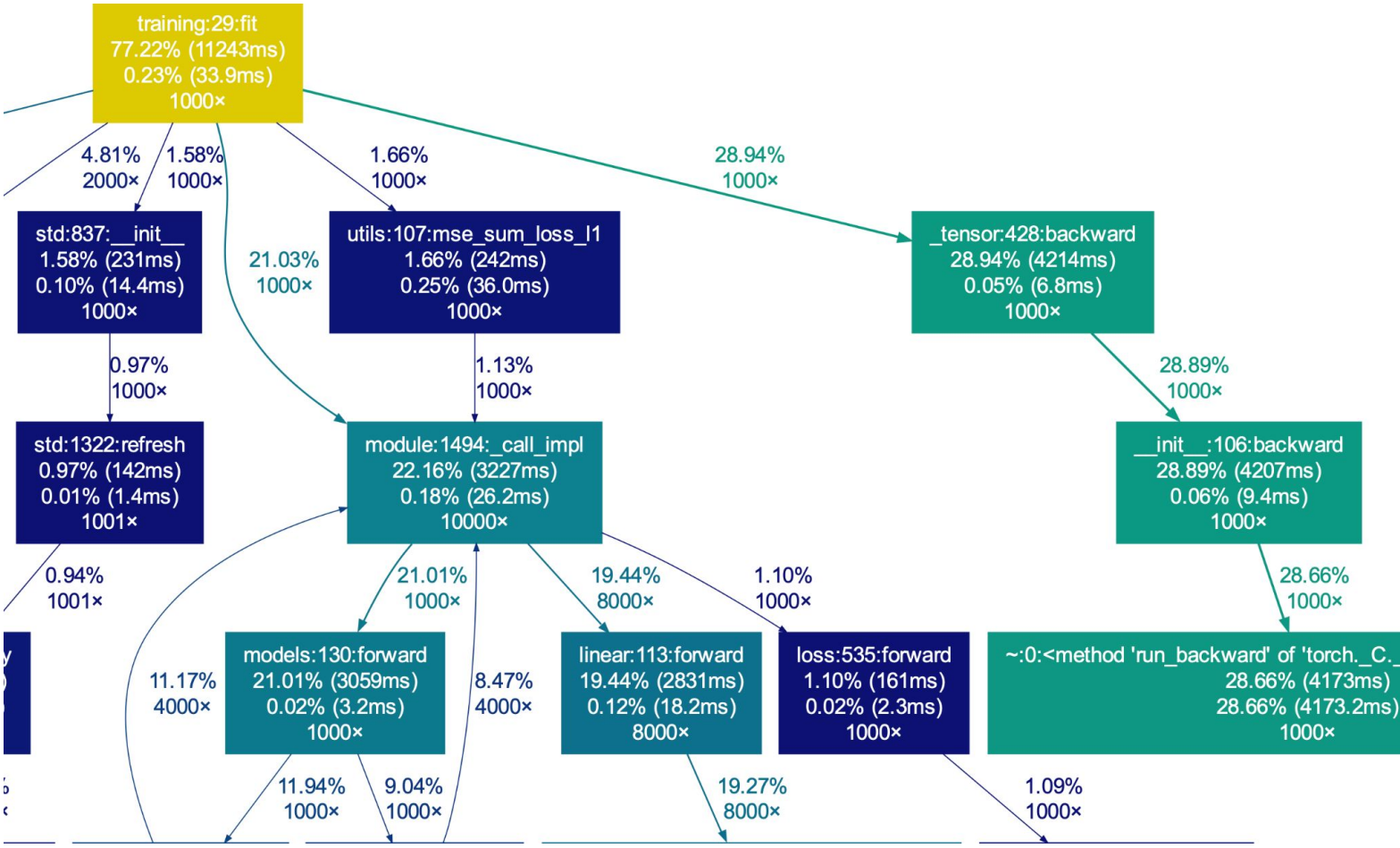
- baler main 78%
- overheads and imports 22%

Our subject of interests are

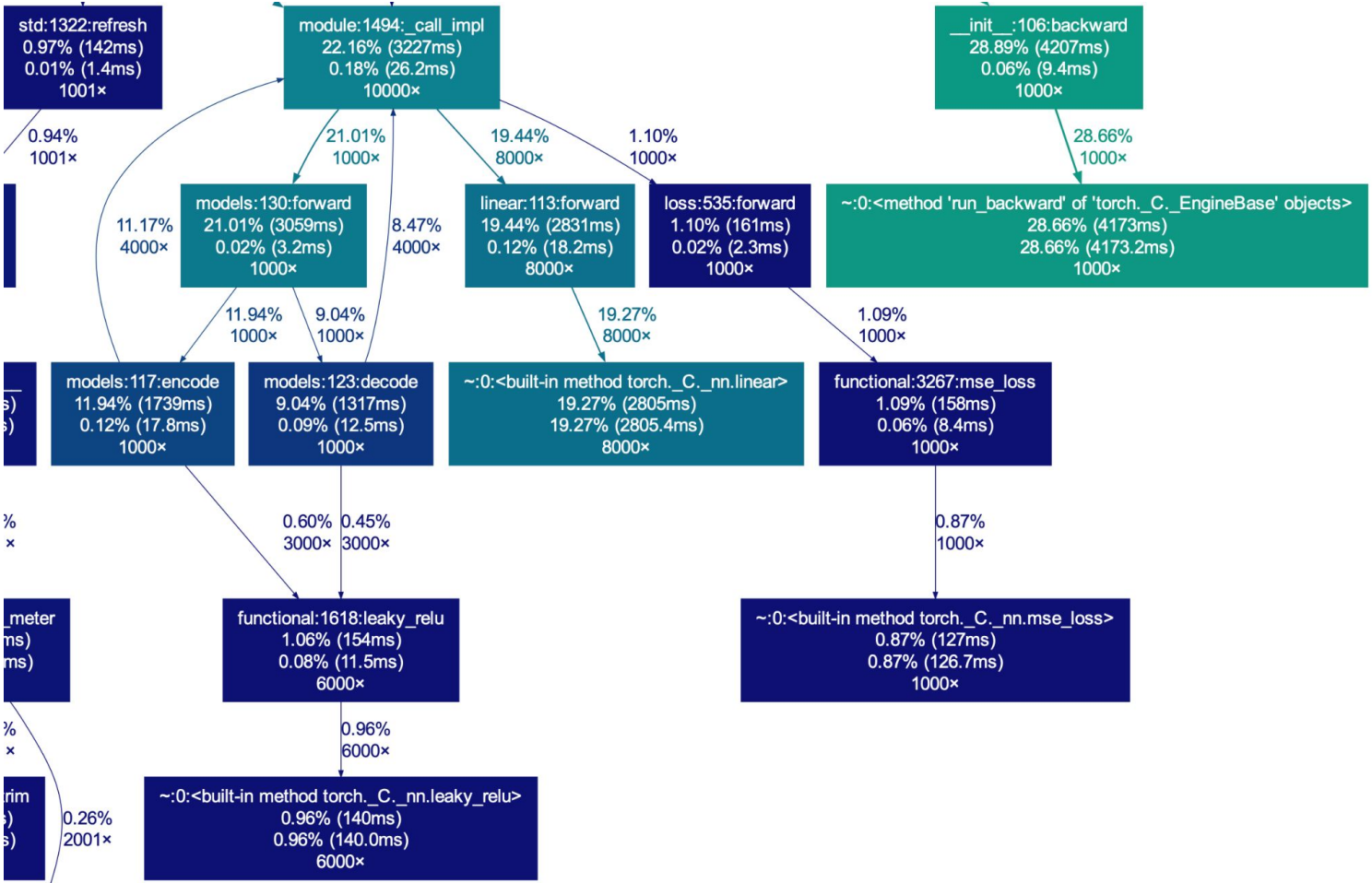
- train(...) in training.py
- fit(...) in training.py



Train fit(...) | T(Backpropagation) › T(Forward)



Train fit(...) | T(encode) > T(decode)



cProfile (Deterministic)

Disadvantages

1. High overhead and distorted results
2. Too much information is given as tracing all the method invokes
3. Suitable to run and test on local environment. Doesn’t consider production env
4. Profiles on the basis of method execution time and not on input or logic

Pyinstrument (Sampling)

1. Solves the issue of high overhead and distorted results
2. Concise information is given
3. Measures wall clock time contrary to cProfile which measured CPU time

Pyinstrument (Sampling Profiler)

Methodology

- Draws samples at some defined intervals of time
- Observe and make estimations based on the sampled data

Benefit

- A more statistically suited sampling method is used
- This uniforms the biases and outliers

Pyinstrument: How it works, cat edition

- 12:00: Sleeping 💤
- 12:05: Sleeping 💤
- 12:10: Eating 🍜
- 12:15: Using the litterbox 💩
- 12:20: Sleeping 💤
- 12:25: Sleeping 💤
- 12:30: Sleeping 💤

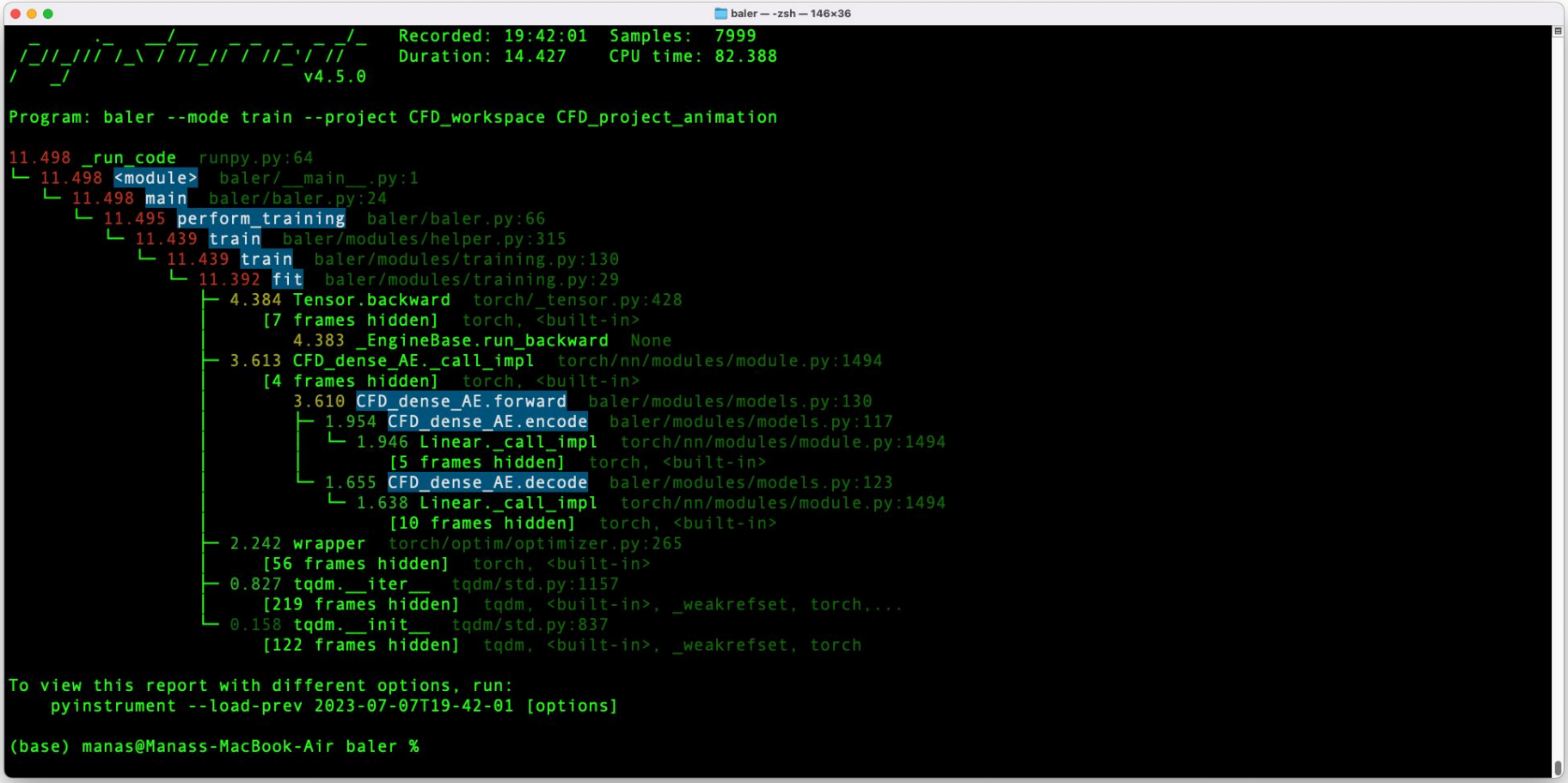
Pyinstrument: How it works, cat edition

- Breakdown of observed cat activities
  - 90%: Sleeping 💤
  - 10%: Eating 🍲
  - 9%: Using the litterbox 💩
  - 1%: Staring longingly through the window at birds 🐦

Wall Clock Time vs CPU Time

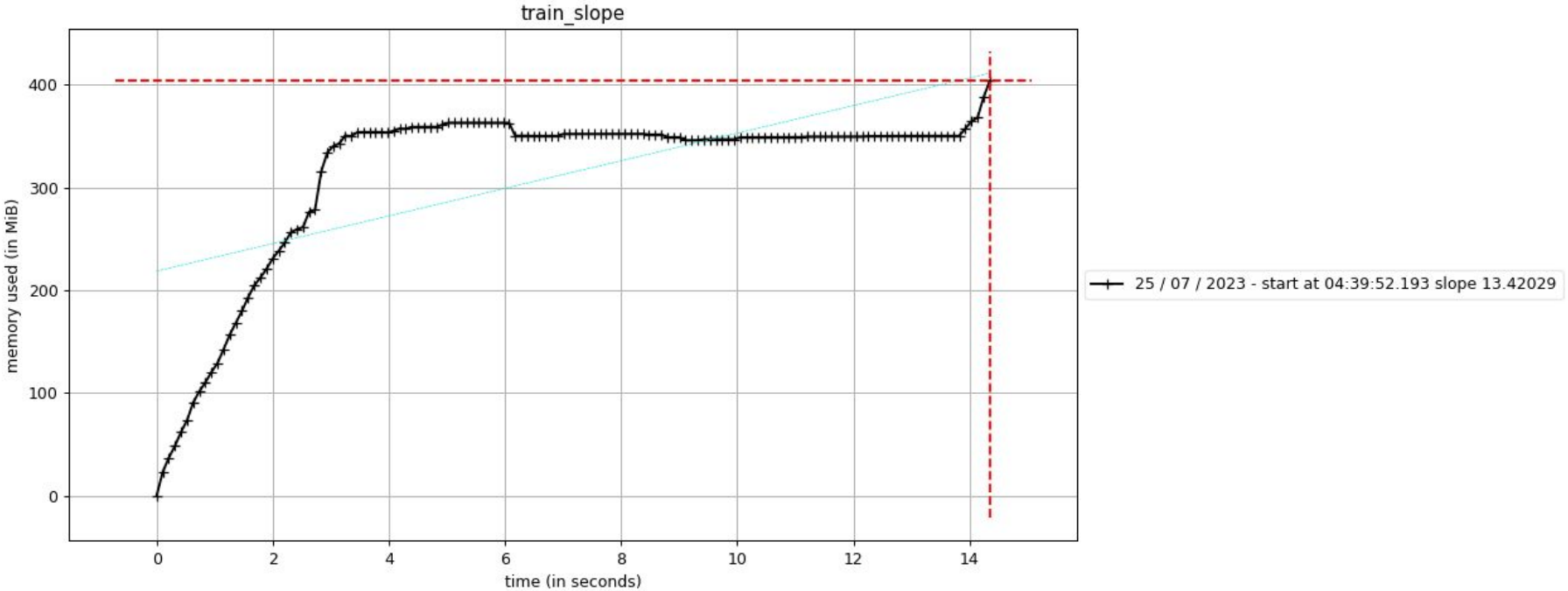
- Wall-clock time is the time that a clock on the wall (or a stopwatch in hand) would measure as having elapsed between the start of the process and 'now'
- The CPU time is the amount of time spent by the user code in the CPU
- The wall-clock time is hence not the number of seconds that the process has spent on the CPU rather it is the elapsed time, including time spent waiting for its turn on the CPU (while other processes get to run)
- Multicore processors, which incorporate multiple processing units, can run several threads at once to use more CPU time than wall clock time

Execution of baler(train)



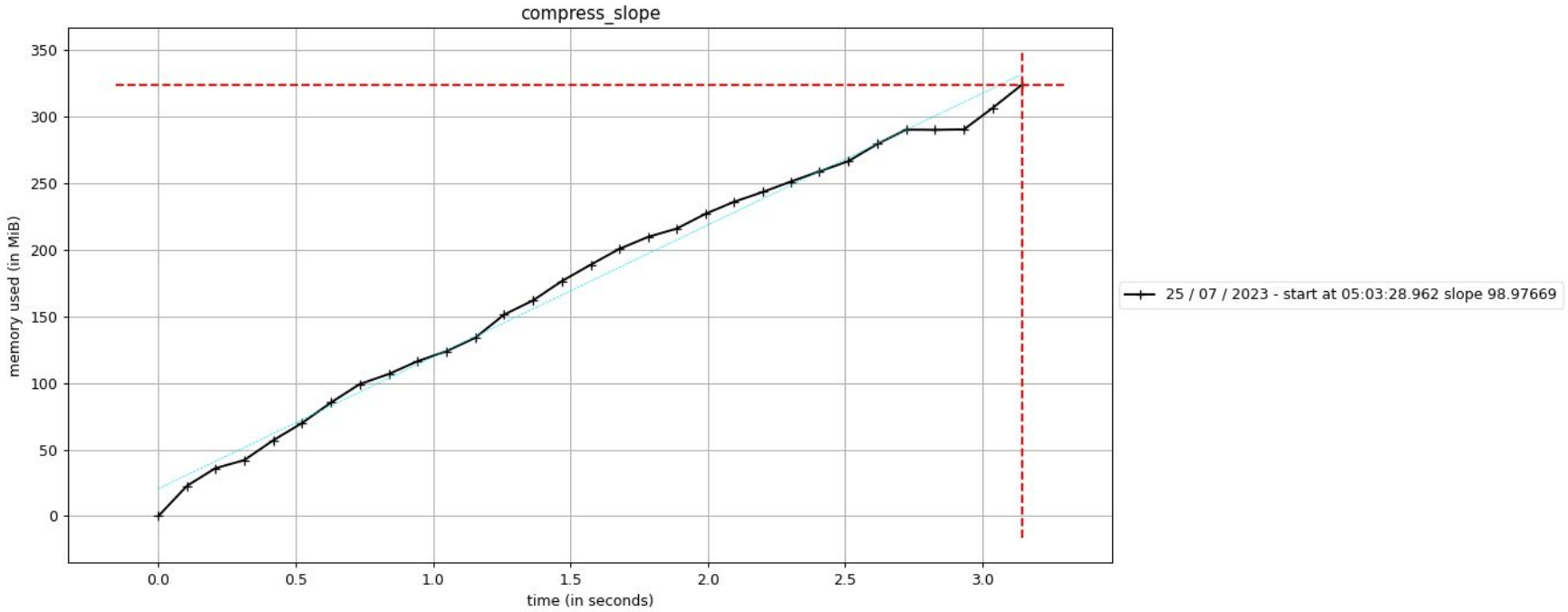
Memory-profiler (MiB - mebibyte)

Train 1000 epochs



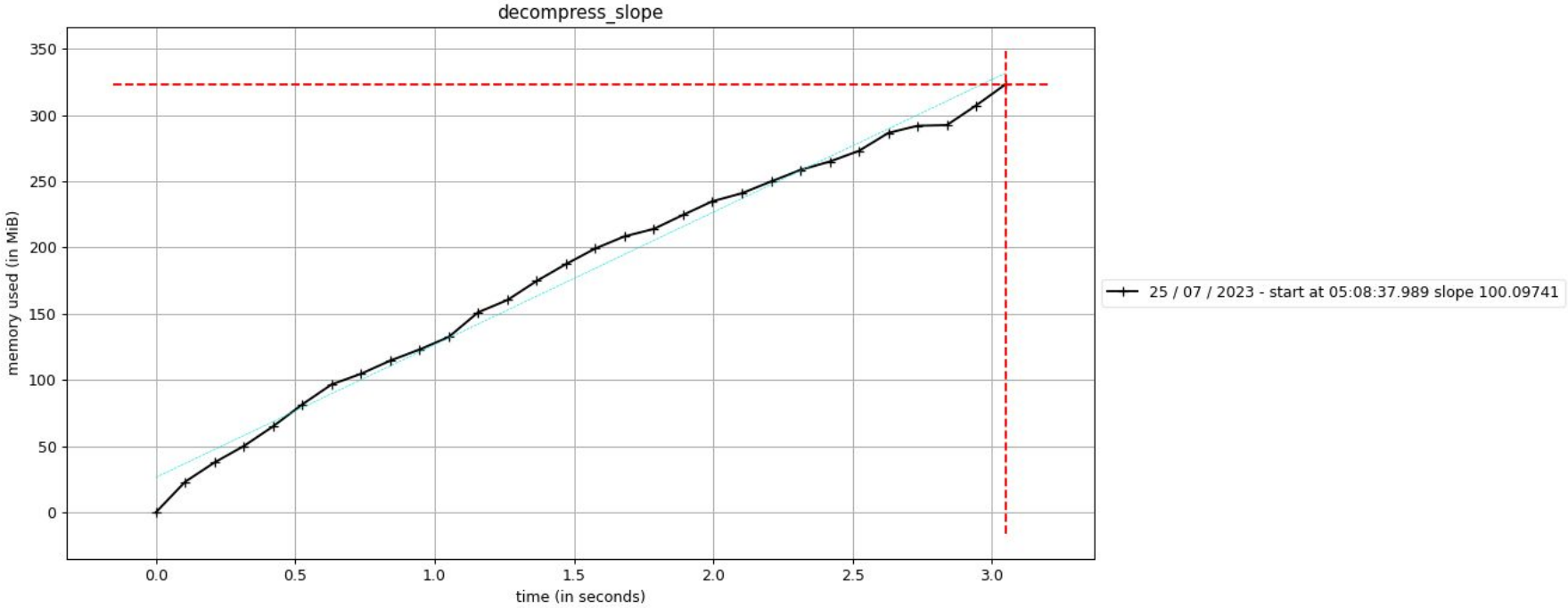
Memory-profiler (MiB - mebibyte)

Compress 1000 epochs



Memory-profiler (MiB - mebibyte)

Decompress 1000 epochs



codecarbon

Methodology - https://mlco2.github.io/codecarbon/methodology.html

- C = Carbon Intensity of the electricity consumed for computation: quantified as g of CO_2 emitted per kilowatt-hour of electricity.
- P = Power Consumed by the computational infrastructure: quantified as kilowatt-hours.
- Carbon dioxide emissions (CO_2eq) can then be calculated as C * P

Carbon Intensity of the consumed electricity is calculated as a weighted average of the emissions from the different energy sources that are used to generate electricity, including fossil fuels and renewables.

A scaling factor of **1e6** was been used to generate the plots for **50 baler runs with 1000 epochs each**

Data Fields Logged for Each Experiment

Field	Description
timestamp	Time of the experiment in `%Y-%m-%dT%H:%M:%S` format
project_name	Name of the project, defaults to `codecarbon`
run-id	id of the run
duration	Duration of the compute, in seconds
emissions	Emissions as CO_2-equivalents [CO_2eq], in kg
emissions_rate	emissions divided per duration, in Kg/s
cpu_power	CPU power (W)
gpu_power	GPU power (W)
ram_power	RAM power (W)
cpu_energy	Energy used per CPU (kW)
gpu_energy	Energy used per GPU (kW)
ram_energy	Energy used per RAM (kW)
energy_consumed	sum of cpu_energy, gpu_energy and ram_energy (kW)
country_name	Name of the country where the infrastructure is hosted
country_iso_code	3-letter alphabet ISO Code of the respective country
region	Province/State/City where the compute infrastructure is hosted

codecarbon

Synopsis

- CO2 Emission
- 0.58 * 1e-6 kg CO2 eqv while train
- 0.44 * 1e-6 kg CO2 eqv while compress
- 0.59 * 1e-6 kg CO2 eqv while decompress
- Energy
- 15.35 * 1e-6 kWh energy per consumed while train
- 0.078 * 1e-6 kWh energy per consumed while compress
- 0.064 * 1e-6 kWh energy per consumed while decompress

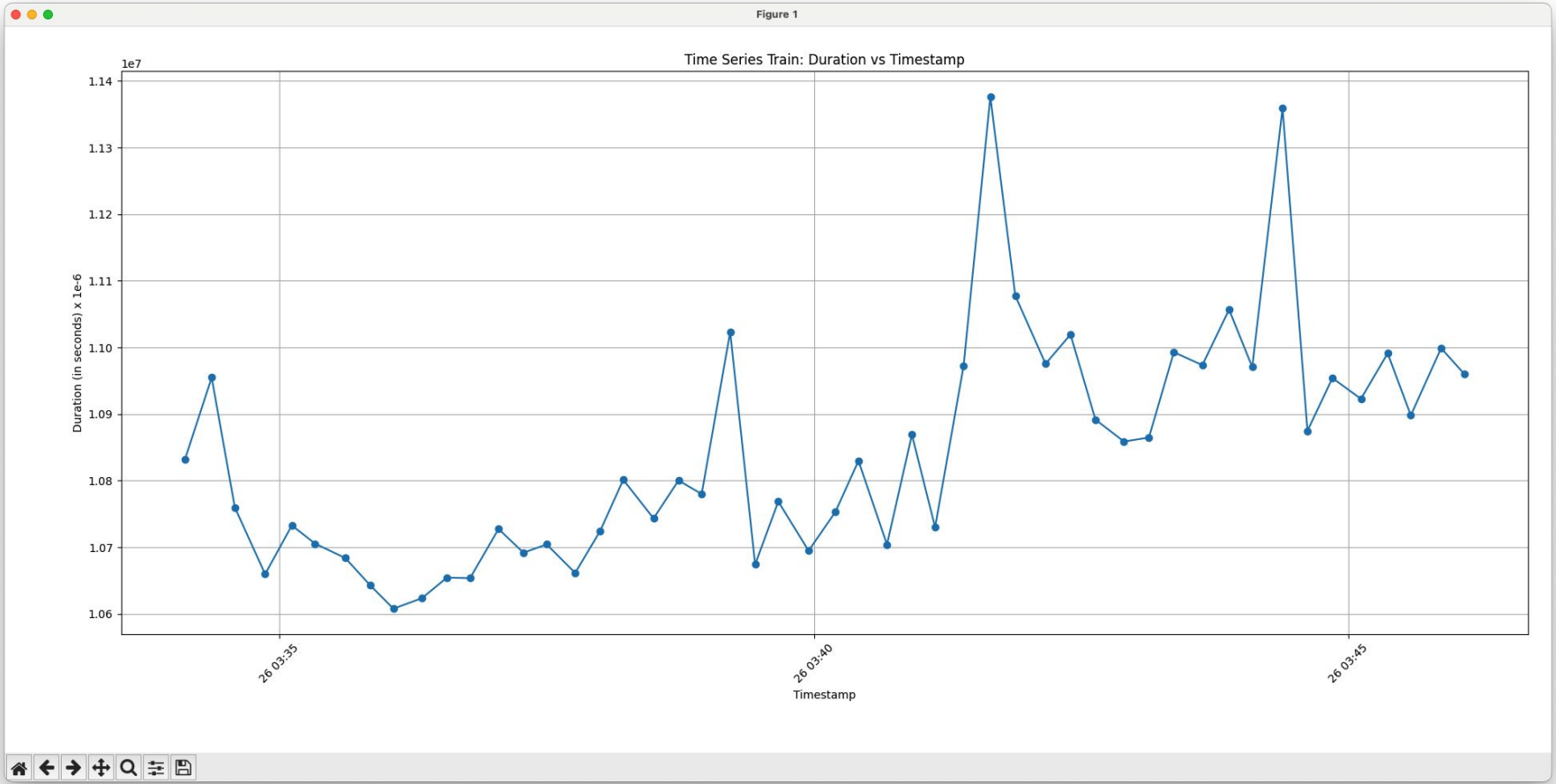


Figure 1
Time Series Train: Duration vs Timestamp
1e7
1.14
1.13
1.12
1.11
1.10
1.09
1.08
1.07
1.06
Duration (in seconds) x 1e-6
26 03:35
26 03:40
26 03:45
Timestamp

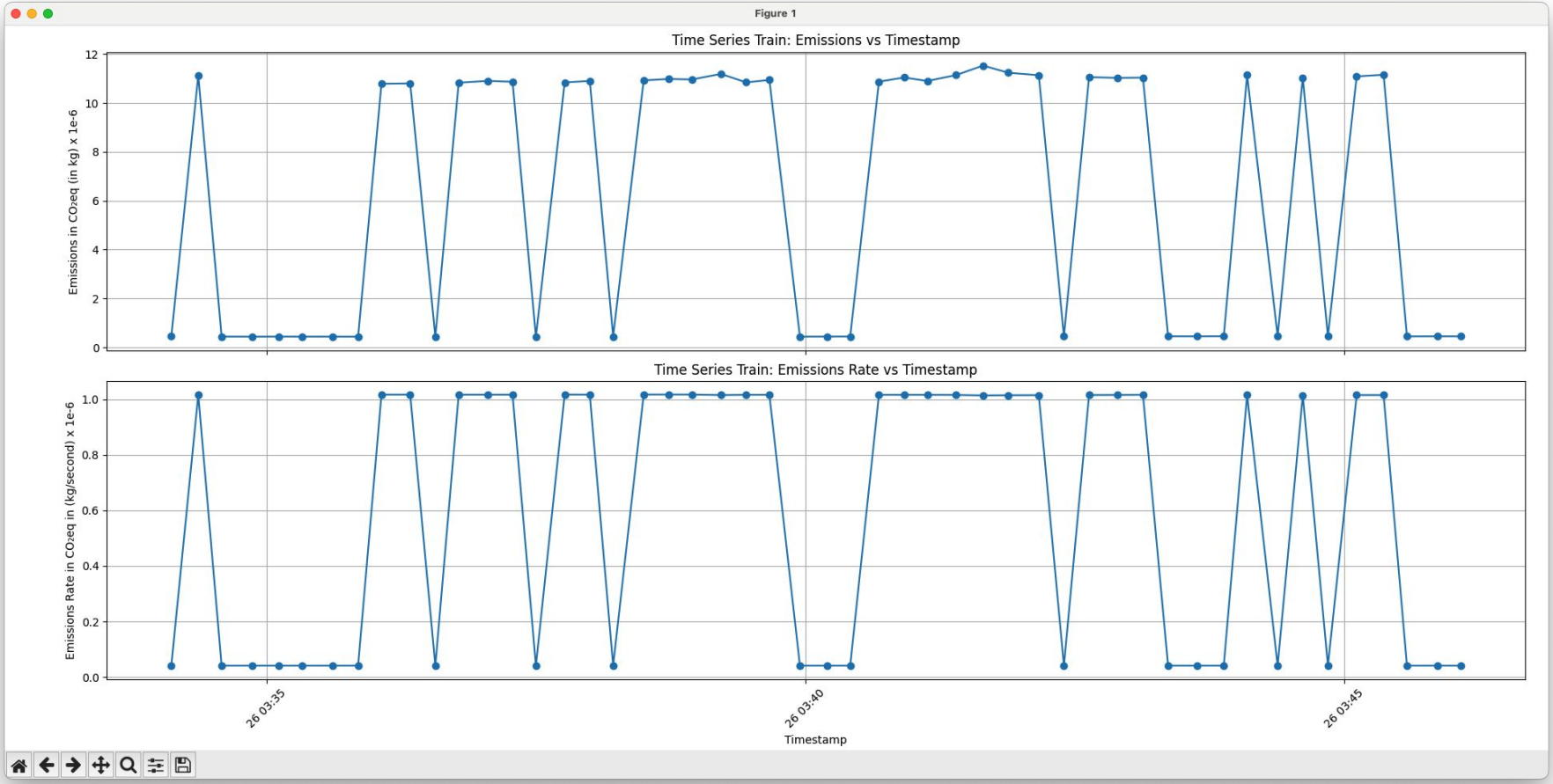


Figure 1
Time Series Train: Emissions vs Timestamp
Emissions in CO2eq (in kg) x 1e-6
0
2
4
6
8
10
12
Time Series Train: Emissions Rate vs Timestamp
Emissions Rate in CO2eq in (kg/second) x 1e-6
0.0
0.2
0.4
0.6
0.8
1.0
26 03:35
26 03:40
26 03:45
Timestamp

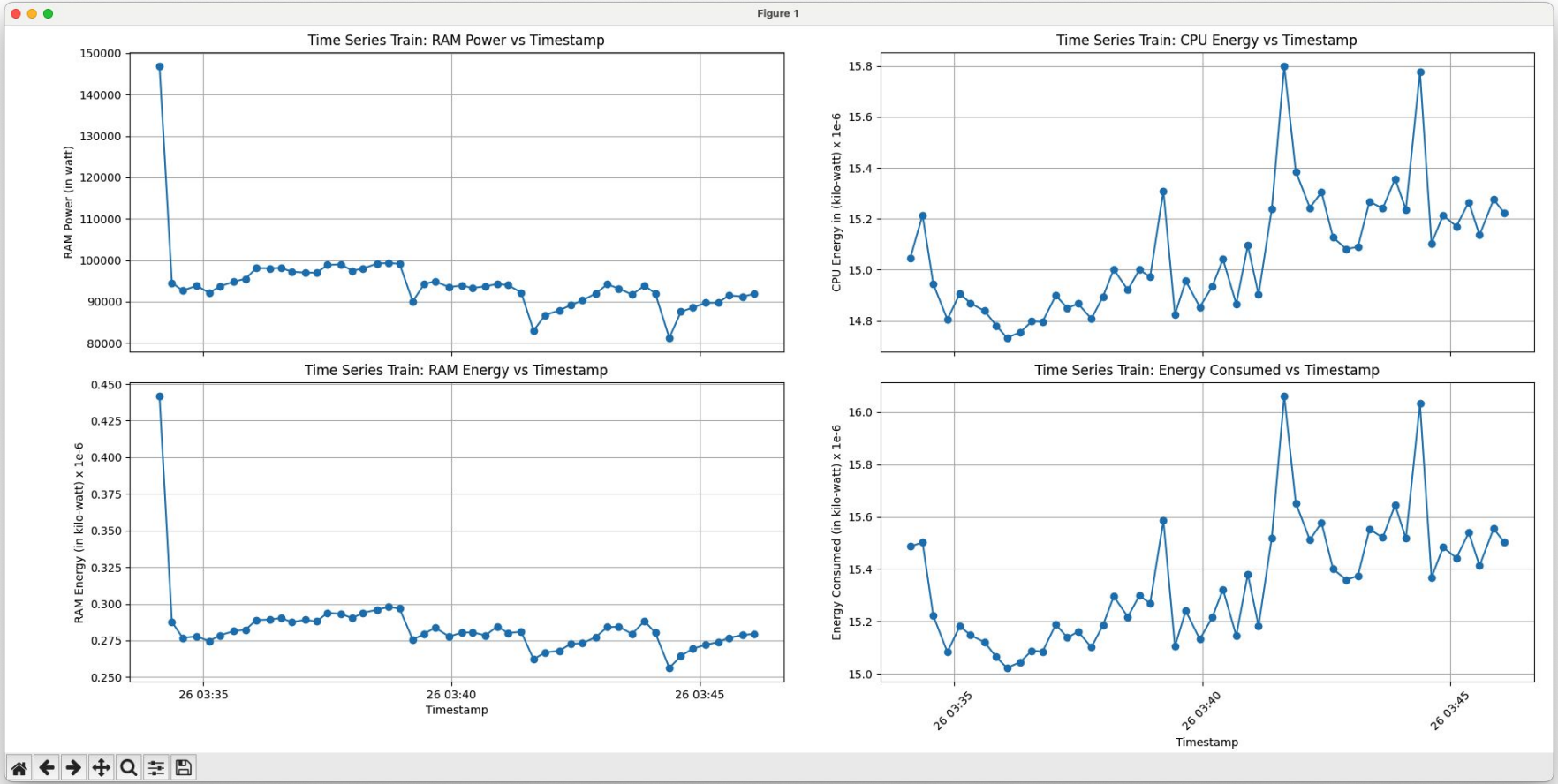


Figure 1
Time Series Train: RAM Power vs Timestamp
RAM Power (in watt)
150000
140000
130000
120000
110000
100000
90000
80000
Time Series Train: CPU Energy vs Timestamp
CPU Energy in (kilo-watt) x 1e-6
15.8
15.6
15.4
15.2
15.0
14.8
Time Series Train: RAM Energy vs Timestamp
RAM Energy (in kilo-watt) x 1e-6
0.450
0.425
0.400
0.375
0.350
0.325
0.300
0.275
0.250
26 03:35
26 03:40
26 03:45
Timestamp
Time Series Train: Energy Consumed vs Timestamp
Energy Consumed (in kilo-watt) x 1e-6
16.0
15.8
15.6
15.4
15.2
15.0
26 03:35
26 03:40
26 03:45
Timestamp

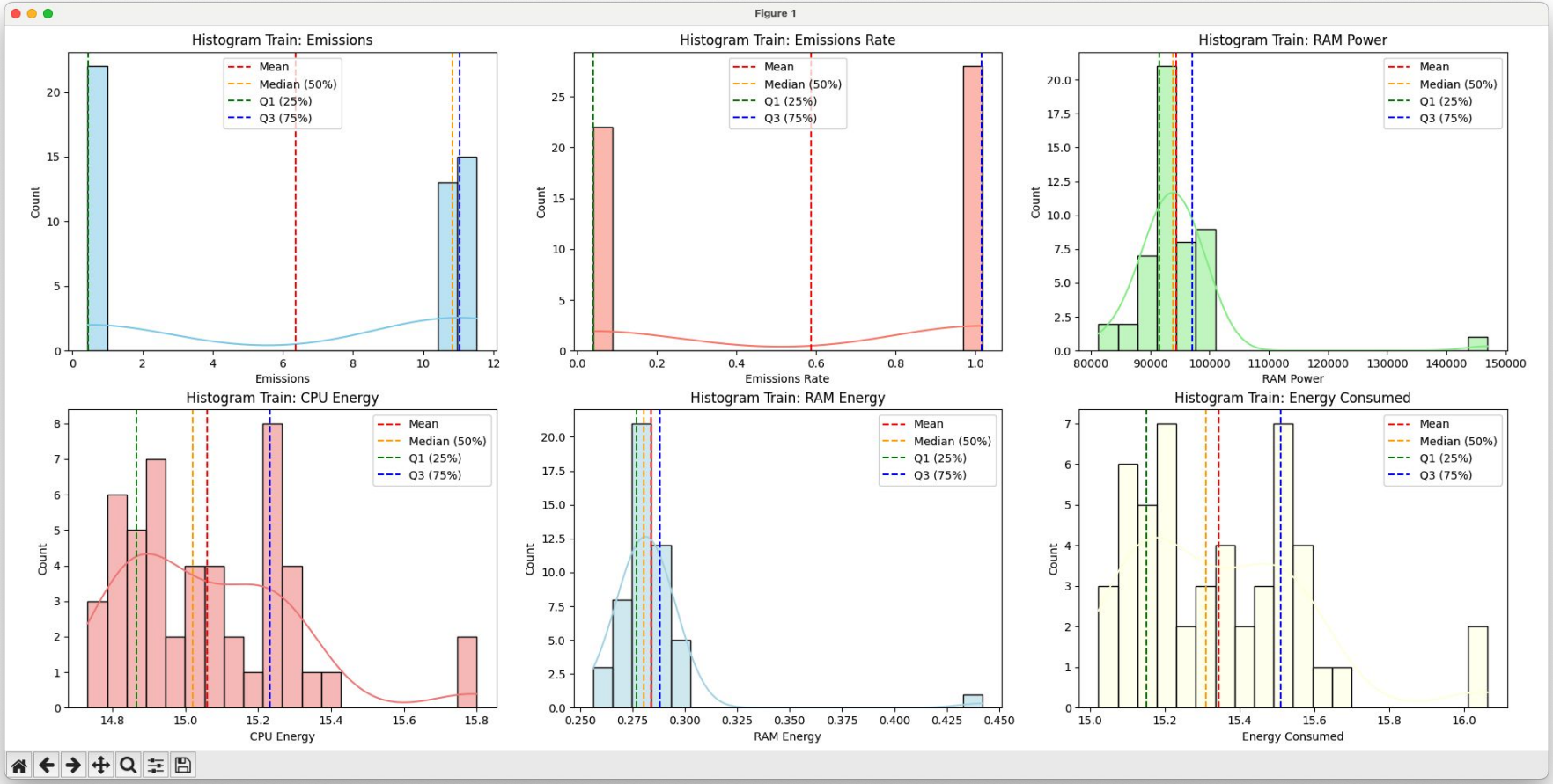


Figure 1
Histogram Train: Emissions
Mean
Median (50%)
Q1 (25%)
Q3 (75%)
Count
Emissions
Histogram Train: Emissions Rate
Emissions Rate
Histogram Train: RAM Power
RAM Power
Histogram Train: CPU Energy
CPU Energy
Histogram Train: RAM Energy
RAM Energy
Histogram Train: Energy Consumed
Energy Consumed

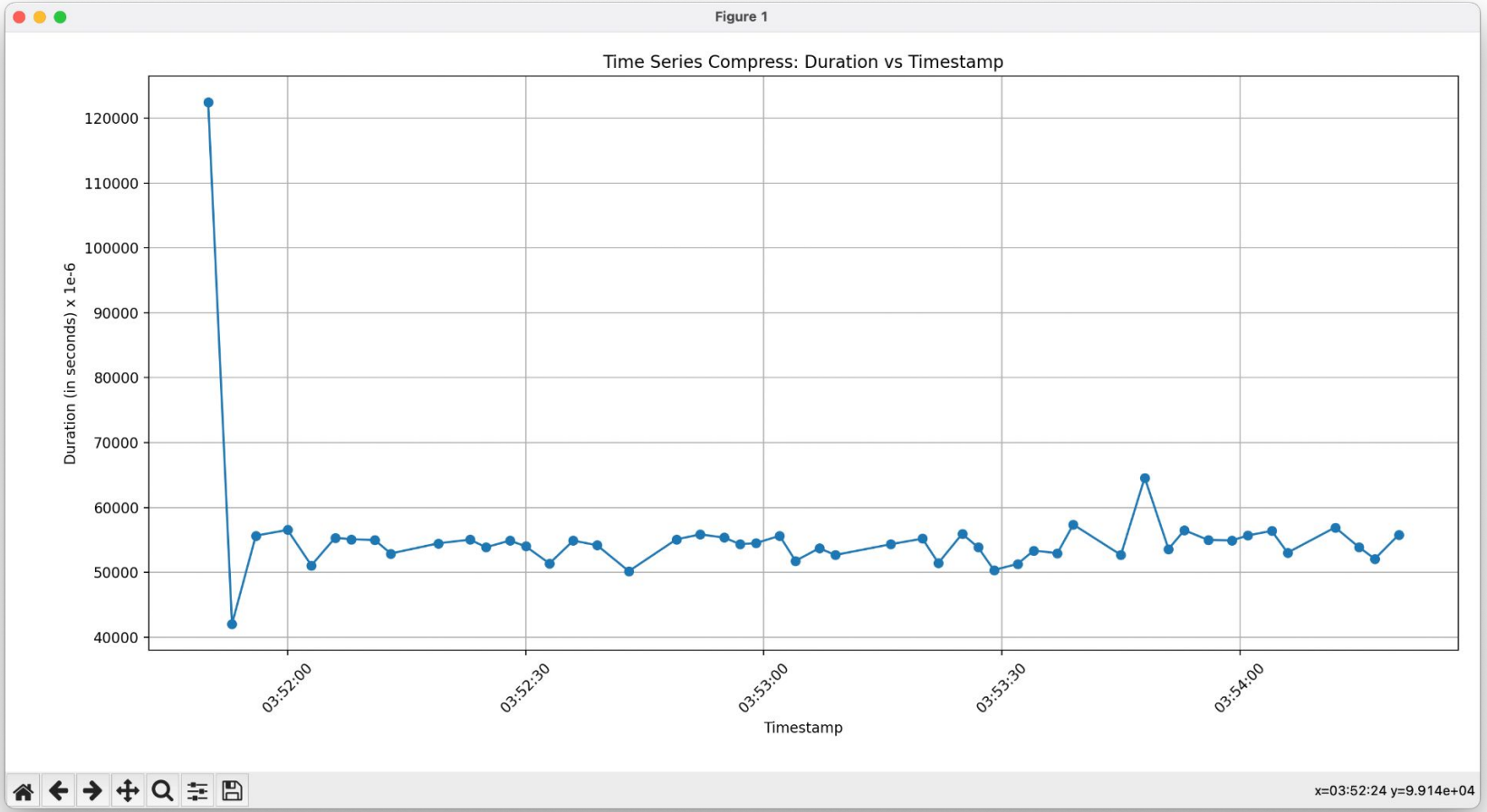


Figure 1
Time Series Compress: Duration vs Timestamp
Duration (in seconds) x 1e-6
120000
110000
100000
90000
80000
70000
60000
50000
40000
03:52:00
03:52:30
03:53:00
03:53:30
03:54:00
Timestamp
x=03:52:24 y=9.914e+04

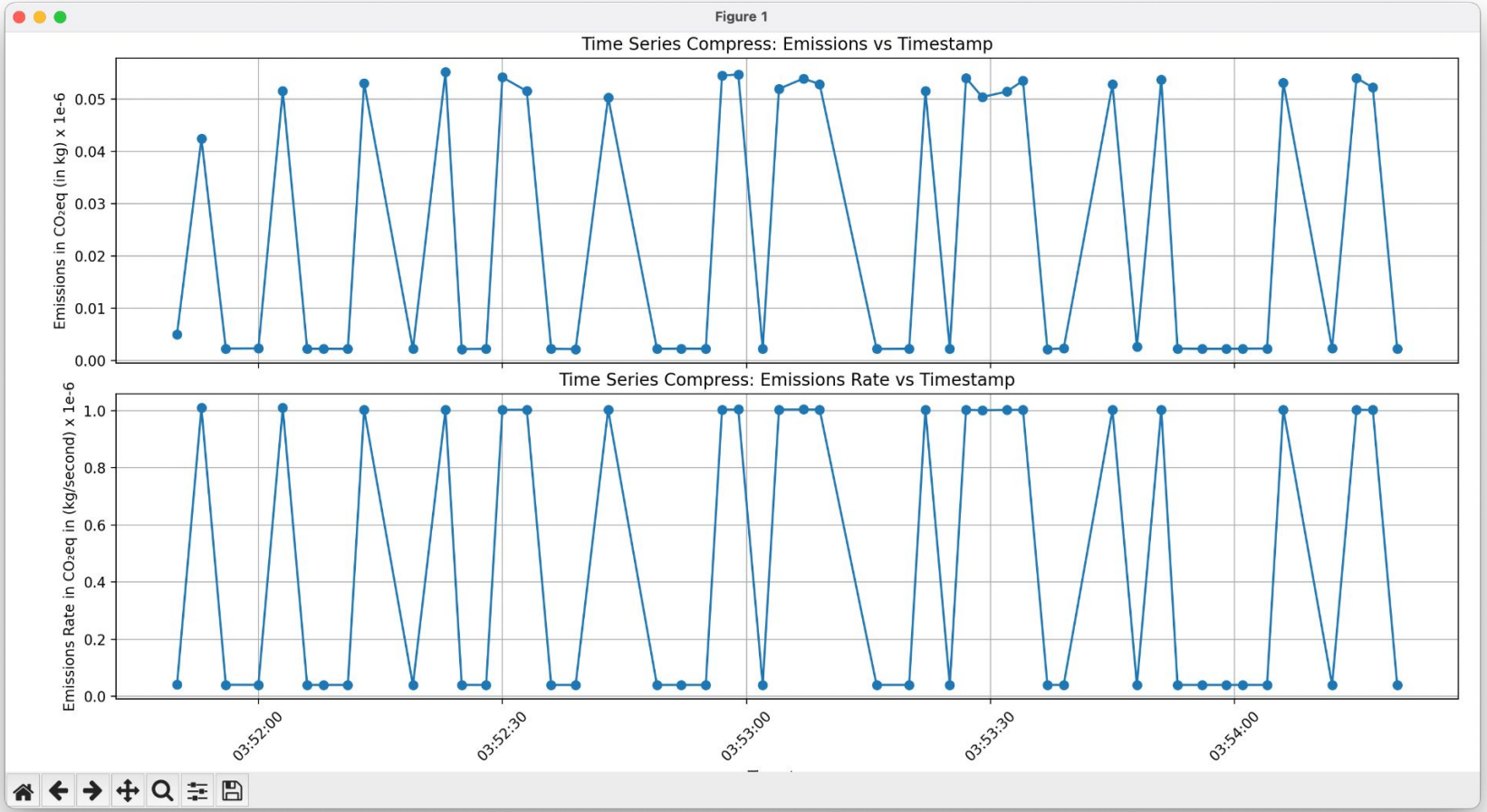


Figure 1
Time Series Compress: Emissions vs Timestamp
Emissions in CO₂eq (in kg) x 1e-6
0.00
0.01
0.02
0.03
0.04
0.05
Time Series Compress: Emissions Rate vs Timestamp
Emissions Rate in CO₂eq in (kg/second) x 1e-6
0.0
0.2
0.4
0.6
0.8
1.0
03:52:00
03:52:30
03:53:00
03:53:30
03:54:00

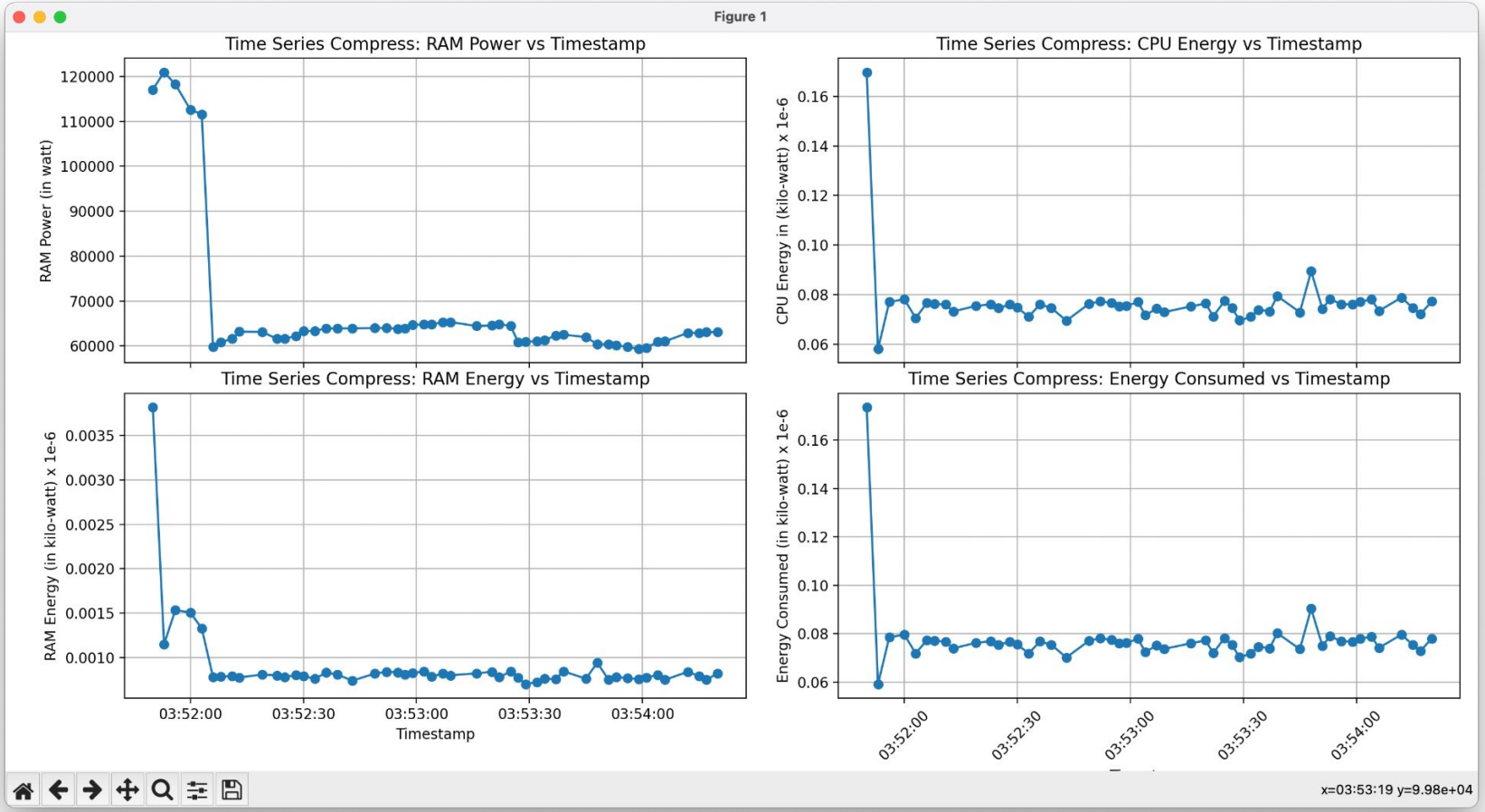


Figure 1
Time Series Compress: RAM Power vs Timestamp
RAM Power (in watt)
120000
110000
100000
90000
80000
70000
60000
Time Series Compress: CPU Energy vs Timestamp
CPU Energy in (kilo-watt) x 1e-6
0.16
0.14
0.12
0.10
0.08
0.06
Time Series Compress: RAM Energy vs Timestamp
RAM Energy (in kilo-watt) x 1e-6
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
03:52:00
03:52:30
03:53:00
03:53:30
03:54:00
Timestamp
Time Series Compress: Energy Consumed vs Timestamp
Energy Consumed (in kilo-watt) x 1e-6
0.16
0.14
0.12
0.10
0.08
0.06
03:52:00
03:52:30
03:53:00
03:53:30
03:54:00
x=03:53:19 y=9.98e+04

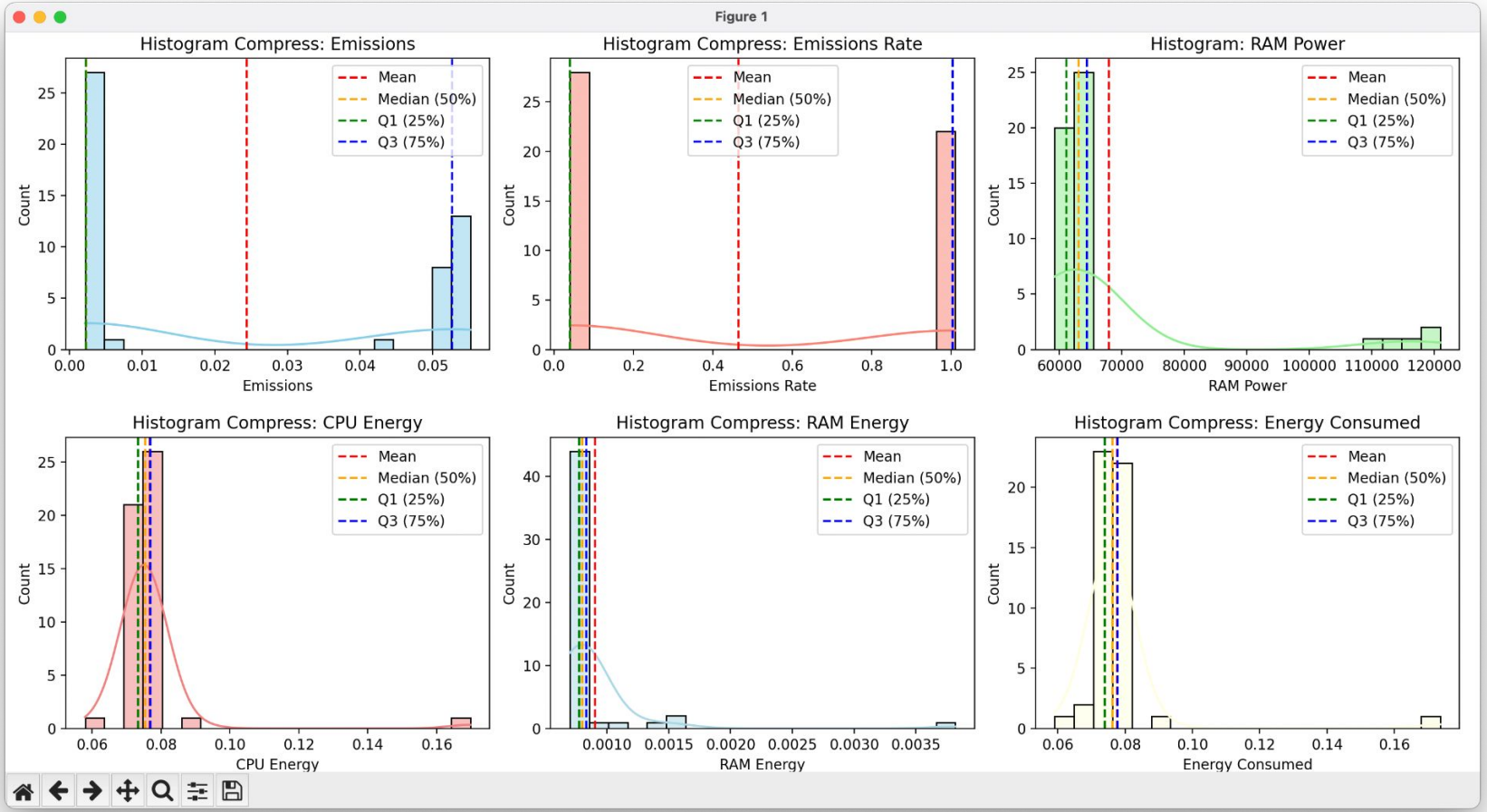


Figure 1
Histogram Compress: Emissions
Mean
Median (50%)
Q1 (25%)
Q3 (75%)
Count
Emissions
Histogram Compress: Emissions Rate
Emissions Rate
Histogram: RAM Power
RAM Power
Histogram Compress: CPU Energy
CPU Energy
Histogram Compress: RAM Energy
RAM Energy
Histogram Compress: Energy Consumed
Energy Consumed

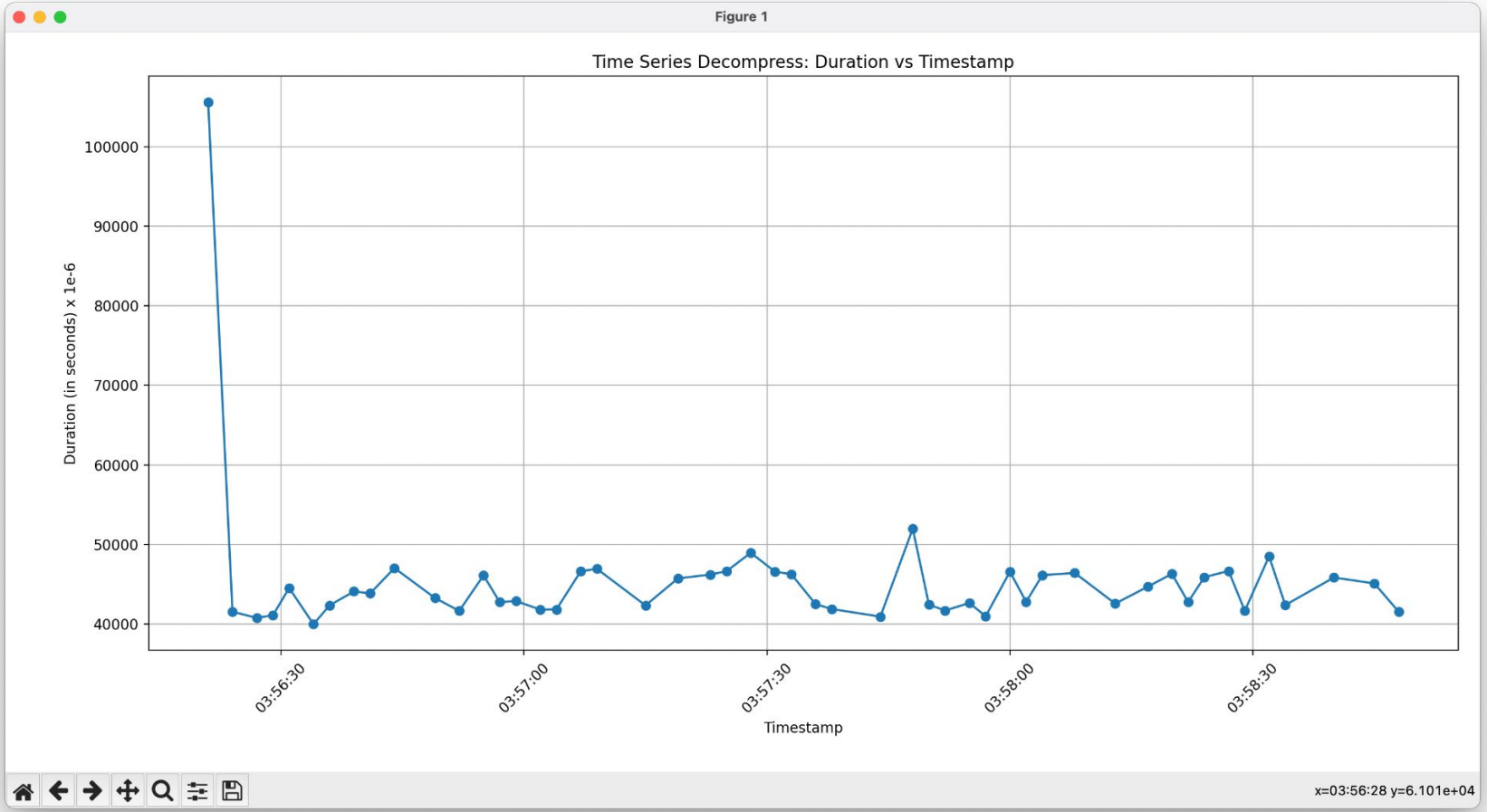


Figure 1
Time Series Decompress: Duration vs Timestamp
Duration (in seconds) x 1e-6
100000
90000
80000
70000
60000
50000
40000
03:56:30
03:57:00
03:57:30
03:58:00
03:58:30
Timestamp
x=03:56:28 y=6.101e+04

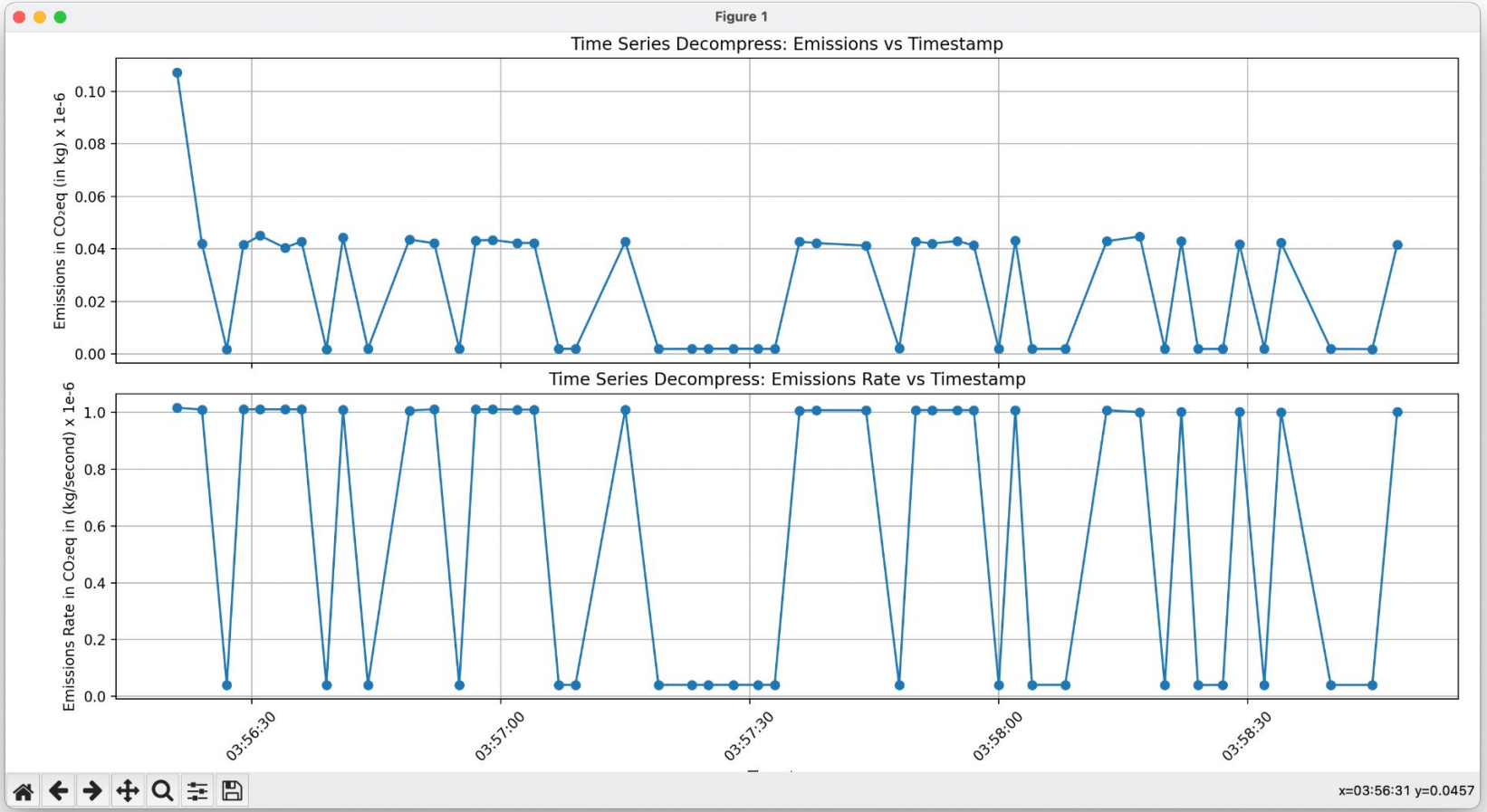


Figure 1
Time Series Decompress: Emissions vs Timestamp
Emissions in CO₂eq (in kg) x 1e-6
0.10
0.08
0.06
0.04
0.02
0.00
Time Series Decompress: Emissions Rate vs Timestamp
Emissions Rate in CO₂eq in (kg/second) x 1e-6
1.0
0.8
0.6
0.4
0.2
0.0
03:56:30
03:57:00
03:57:30
03:58:00
03:58:30
x=03:56:31 y=0.0457

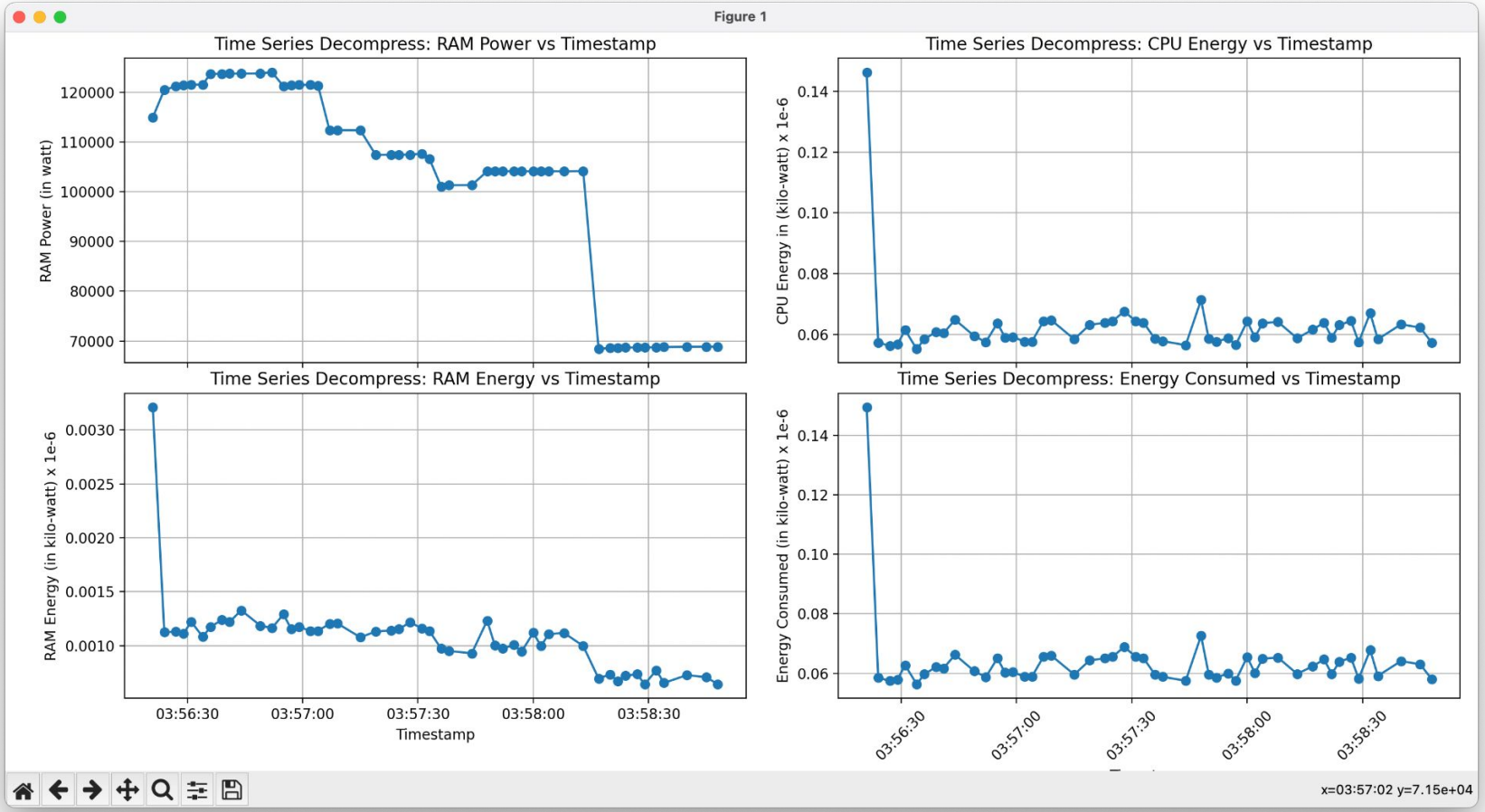


Figure 1
Time Series Decompress: RAM Power vs Timestamp
RAM Power (in watt)
120000
110000
100000
90000
80000
70000
Time Series Decompress: CPU Energy vs Timestamp
CPU Energy in (kilo-watt) x 1e-6
0.14
0.12
0.10
0.08
0.06
Time Series Decompress: RAM Energy vs Timestamp
RAM Energy (in kilo-watt) x 1e-6
0.0030
0.0025
0.0020
0.0015
0.0010
03:56:30
03:57:00
03:57:30
03:58:00
03:58:30
Timestamp
Time Series Decompress: Energy Consumed vs Timestamp
Energy Consumed (in kilo-watt) x 1e-6
0.14
0.12
0.10
0.08
0.06
03:56:30
03:57:00
03:57:30
03:58:00
03:58:30
x=03:57:02 y=7.15e+04

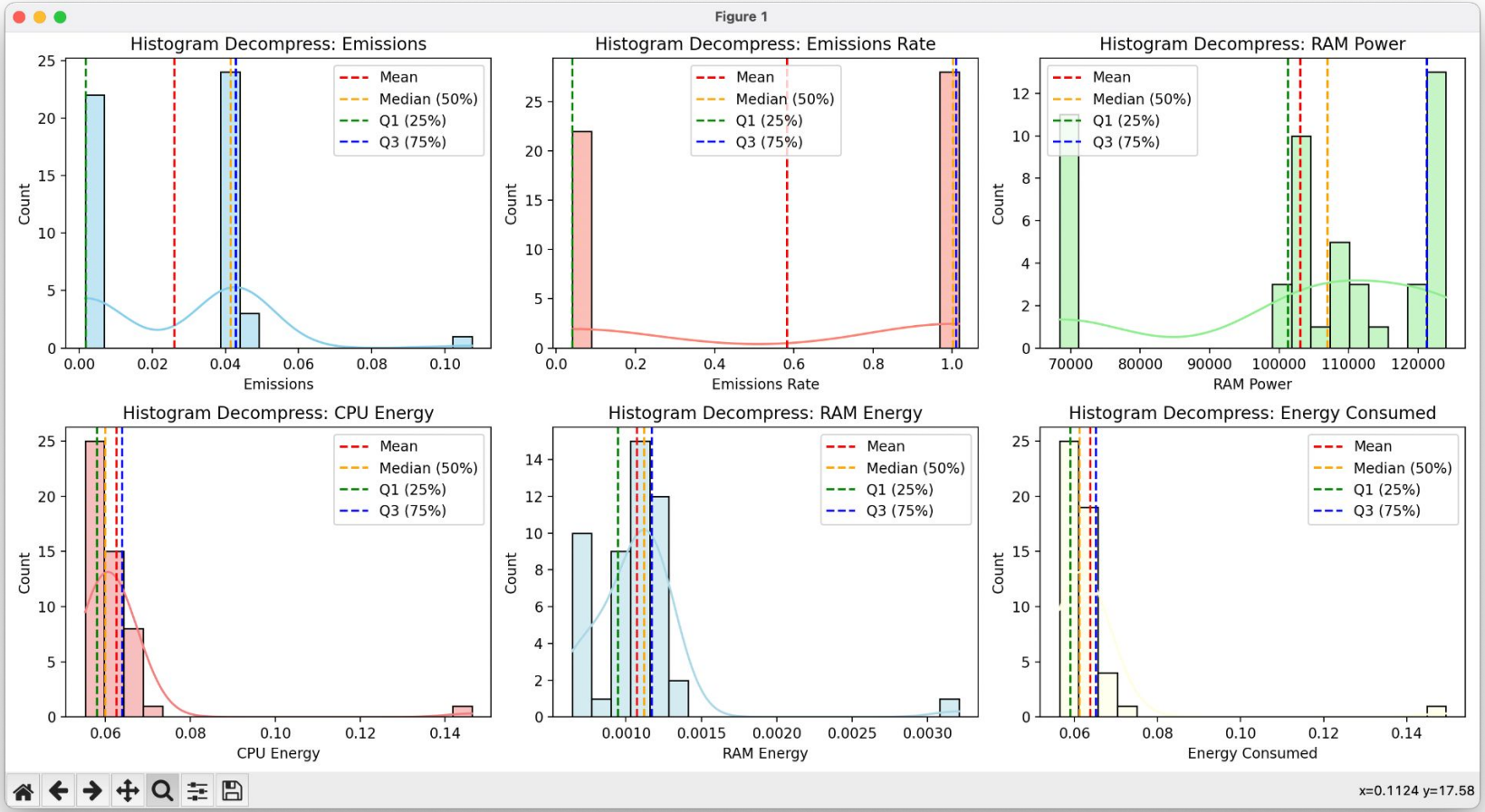


Figure 1
Histogram Decompress: Emissions
Mean
Median (50%)
Q1 (25%)
Q3 (75%)
Count
Emissions
Histogram Decompress: Emissions Rate
Emissions Rate
Histogram Decompress: RAM Power
RAM Power
Histogram Decompress: CPU Energy
CPU Energy
Histogram Decompress: RAM Energy
RAM Energy
Histogram Decompress: Energy Consumed
Energy Consumed
x=0.1124 y=17.58

powermetrics and influxdb

Powermetrics is a built-in tool for macOS which monitors CPU usage and determines how much CPU time and CPU power is being allocated to different quality of service classes (QoS). By assigning a QoS to work, you indicate its importance, and the system prioritizes it and schedules it accordingly

Link to the MAN page - https://www.unix.com/man-page/osx/1/powermetrics/

powermetrics and influxdb

Live Demonstration!

Next Plans

- Test the results with another memory profiler memray
- Try out scaphandre
- Read more about refactoring python code and try implementing some of them
- https://aman.ai/tnt/docs/Chapter5/speed_up_code.html
- https://pythonspeed.com/pygotham19
- Write blogs for Mid-Term Evaluation

Link to the GSoC Blogs - https://manaspratimbiswas.com/gsoc

Blockers

- Some dependencies and tools are not running on my current Ubuntu 22.04.2 LTS

- Need to downgrade and install Ubuntu 18.04.6

- Need intel machines with Nvidia to test out some tools