Green AI

Release 0.0.2

Rover van der Noort, Martijn Smits, Remy Duijsens, Dajt Mullaj

Apr 01, 2023

USER MANUAL:

Date: Apr 01, 2023 **Version**: 0.0.2

GATorch is a tool seamlessly integrated with PyTorch that enables ML developers to generate an energy consumption report. By attaching your model, the tool automatically tracks the energy consumption of your model's training and generates graphs and plots to gain in-depth insights into the energy consumption of your model.

This guide provides the user with the explenention of GATorch concepts and functionality.

CHAPTER

ONE

INSTALLATION

1.1 Installing an official release

You can install GATorch using pip. To install GATorch, at the command line, run:

```
~$ pip install GATorch
```

1.2 Verify Installation

You can verify that GATorch was installed on your local computer by running:

```
~$ pip show GATorch
Name: GATorch
...
```

If instead of what is shown above your output is:

```
WARNING: Package(s) not found: GATorch
```

GATorch was not installed correctly or your system cannot find the path to it.

CHAPTER

TWO

GETTING STARTED WITH GATORCH

To use GATorch simply create a `GA` object. You can then track the measurements of a pytorch model by simply using `attach_model()`. With the latter function you can pass a model to your `GA` object, which will then measure the energy consumption of the model during each forward and backward pass.

```
from GATorch import GA

# Create the profiler object and attach a model to it
ga_measure = GA()
ga_measure.attach_model(model)
```

Note: Due to Platypus attack Intel RAPL requires root permission for energy readings. In order to run this program with the correct permissions, do NOT make Intel RAPL readable for any user as this introduces vulernability. Instead use Python with sudo instead:

```
~$ sudo ./.venv/bin/python <script_name>.py
```

Once the model is attached to the `GA` object you can simply follow you normal training loop routine. GATorch will take care to create the measurements in the background. You can then retrieve the measurements at any point.

```
# Let's try to do a single forward pass and a backward pass
x = torch.zeros([1, 1, 28, 28]).to(device)
y = torch.zeros([1, 10]).to(device)
pred = model(x) # forward

loss = loss_fn(pred, y)
optimizer.zero_grad()
loss.backward() # backward
optimizer.step()

# Now lets print the mean measurements
print(ga_measure.get_mean_measurements())
```

2.1 Retrieving the measurements

To retrieve the different energy consumption measurements you can use either of the following functions:

```
ga_measure.get_mean_measurements()
ga_measure.get_sum_measurements()
ga_measure.get_full_measurements()
```

Respectively they will show the mean energy measurement of a forward or backward pass of the model, the sum of all the measurements for each pass and the full list of measurements. The measurements are displayed in Joules.

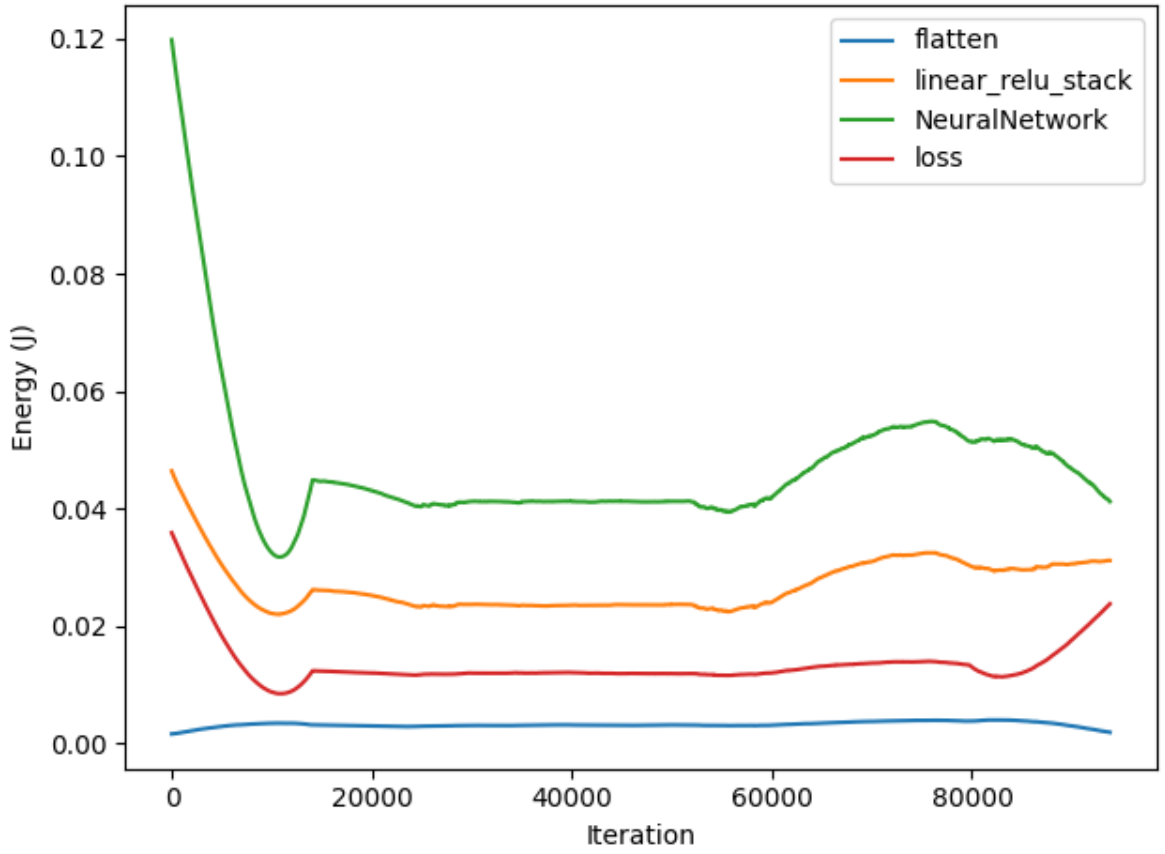
You can also convert the readings into a `pandas.DataFrame`.

```
ga_measure.to_pandas()
```

NeuralNetwork_forward	NeuralNetwork_backward
0.031860	0.042359
0.042236	0.034667

To now visualize the results use `visualize_data()`.

```
ga_measure.visualize_data()
```



2.2 Tracking a model

To start tracking a model you need to attach it to a `GA` object. By default the `GA` profiler will also track the energy consumption that each named layer of the model generates. If you are not intrested in this data you can specify it when attaching the model to the profiler.

```
ga_measure = GA()

ga_measure.attach_model(model_1, named_layer=False)
```

If you want to track the energy consumption of a new model you must first detach the previous one. Finally you can also attach a loss function to track the forward and backward passes of the torch loss criterion.

```
ga_measure.detach_model()
ga_measure.attach_model(model_2, loss=loss_fn)
```

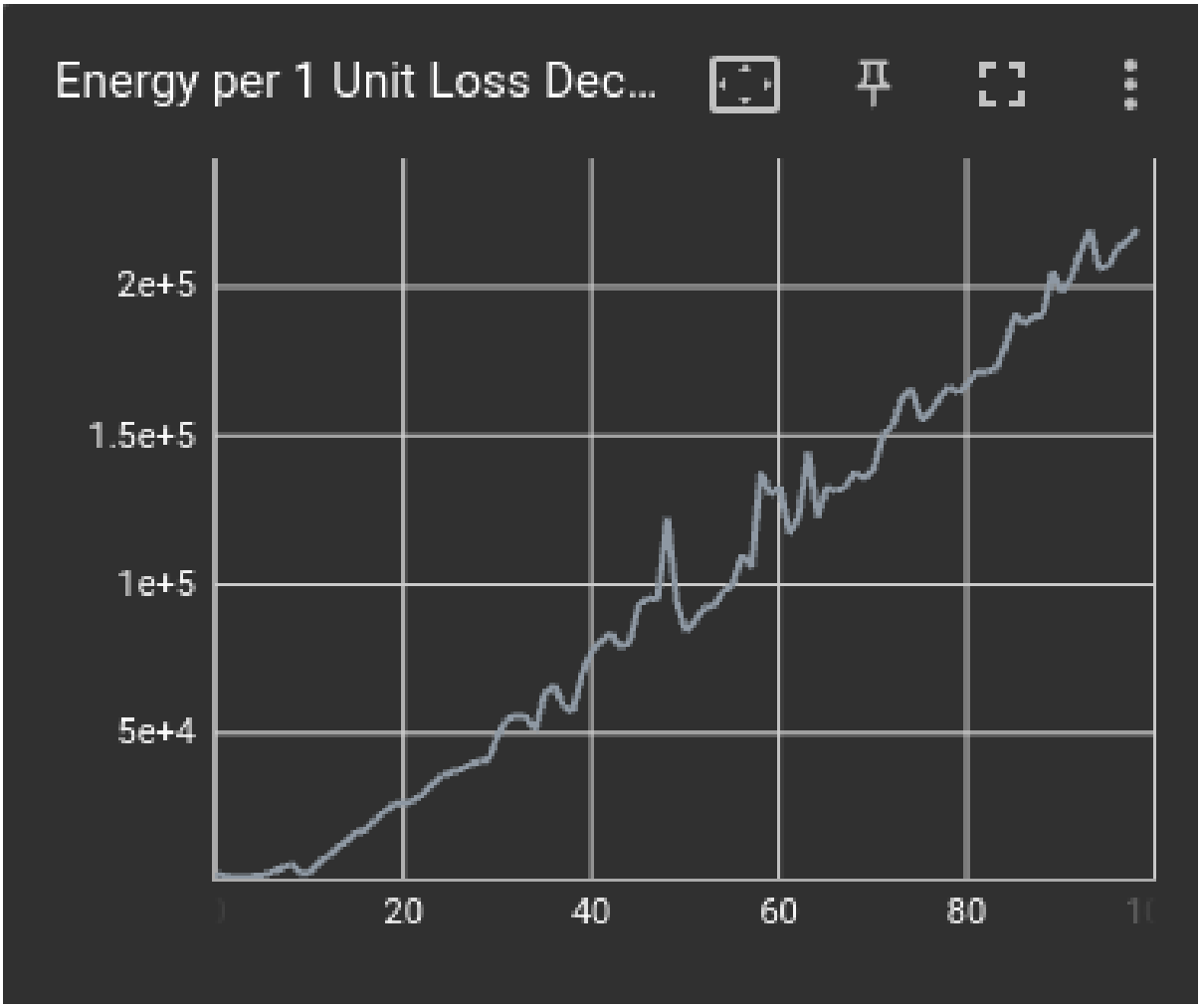
To reset the current energy measurements use `reset()`.

```
ga_measure.reset()
```

2.3 Loss vs Energy Consumption

An important feature of `GATorch` is its ability to compute and display how much energy is needed to improve the loss of a model at each step of the training loop. This way it is possible to see how improving the loss becomes more expensive as the loss becomes smaller, giving machine learning scientist and enginners a new criterion to judge, for example, when to stop training. We show this data using tensorboard.

```
ga_measure.set_tensorboard_stats()
```



For more details about `tensorboard` integration see this *section*.

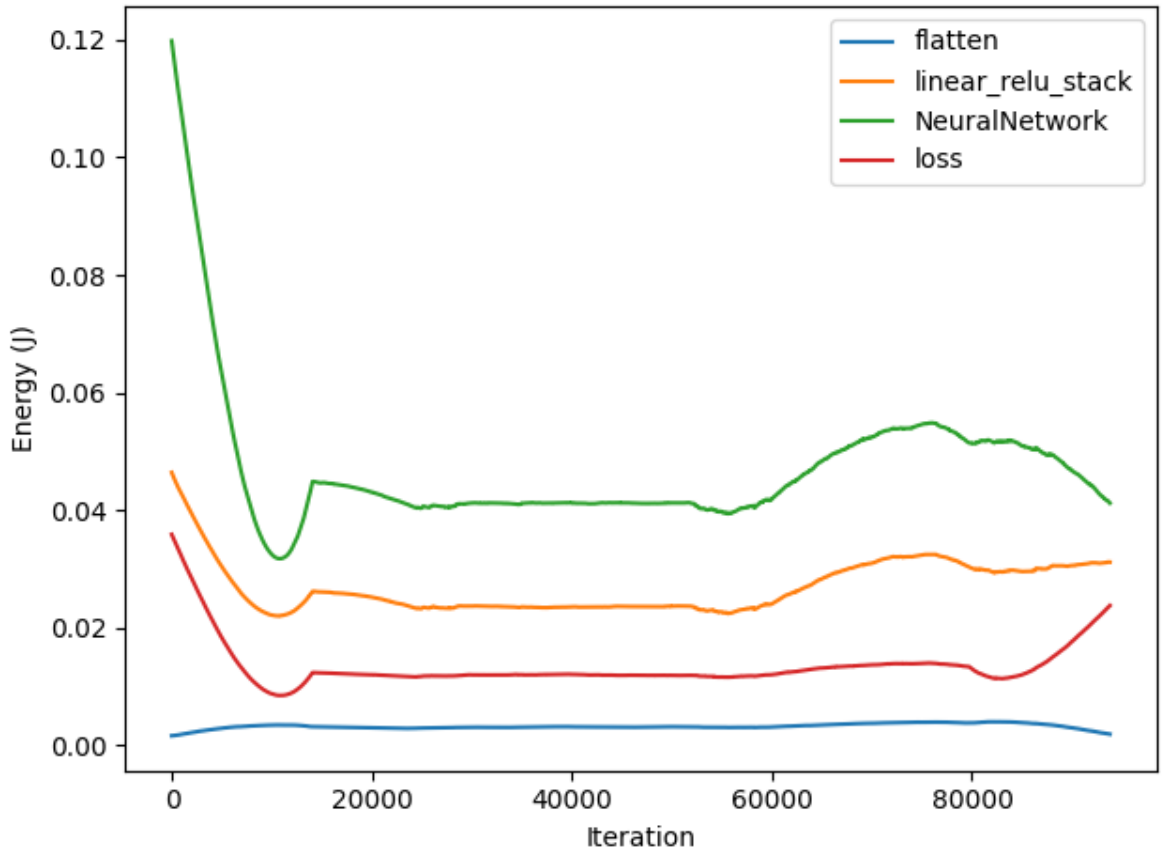
CHAPTER

THREE

VISUALIZE MEASUREMENTS

Using GATorch you can track the energy consumption of your models. To visualize the results you can use the `visualize_data()` function.

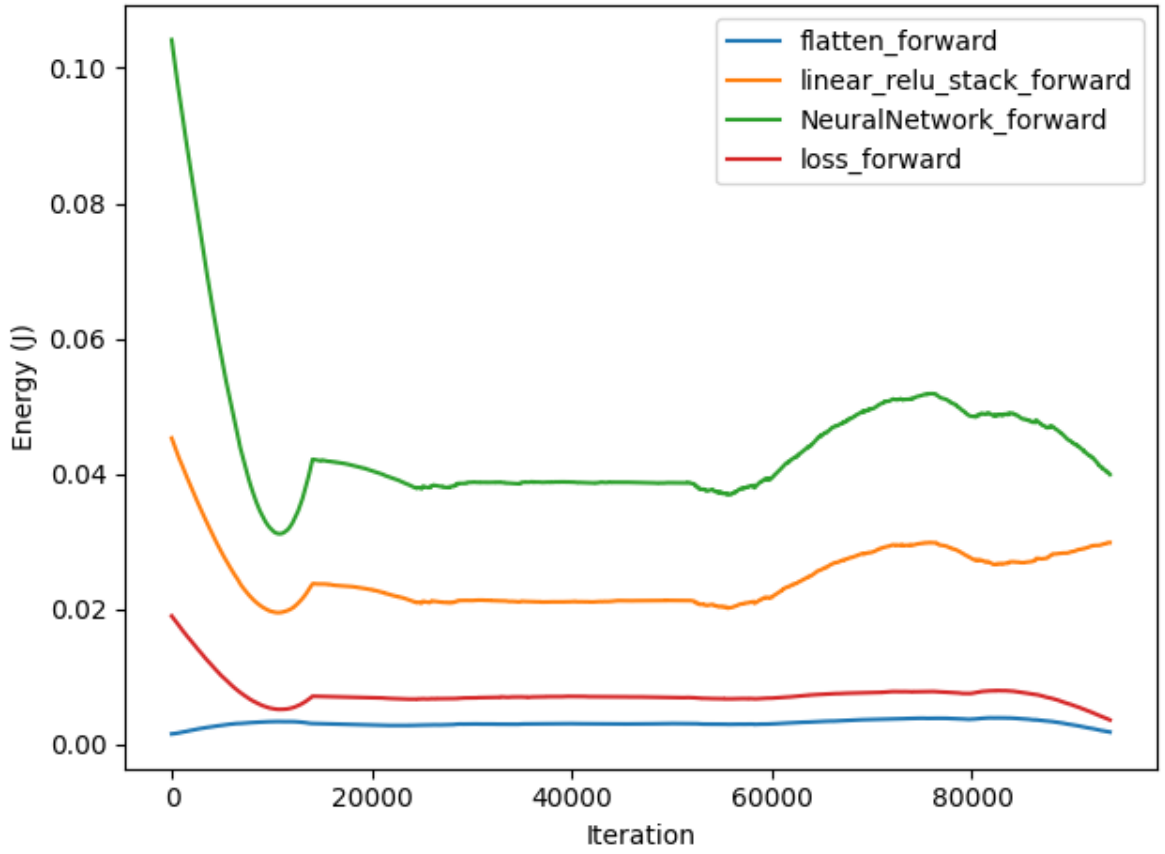
```
from GATorch import GA

ga_measure = GA()
ga_measure.attach_model(model)
train(model)

ga_measure.visualize_data()
```



You can use the function to specify if you want to see the energy consumption of the model during a forward pass or

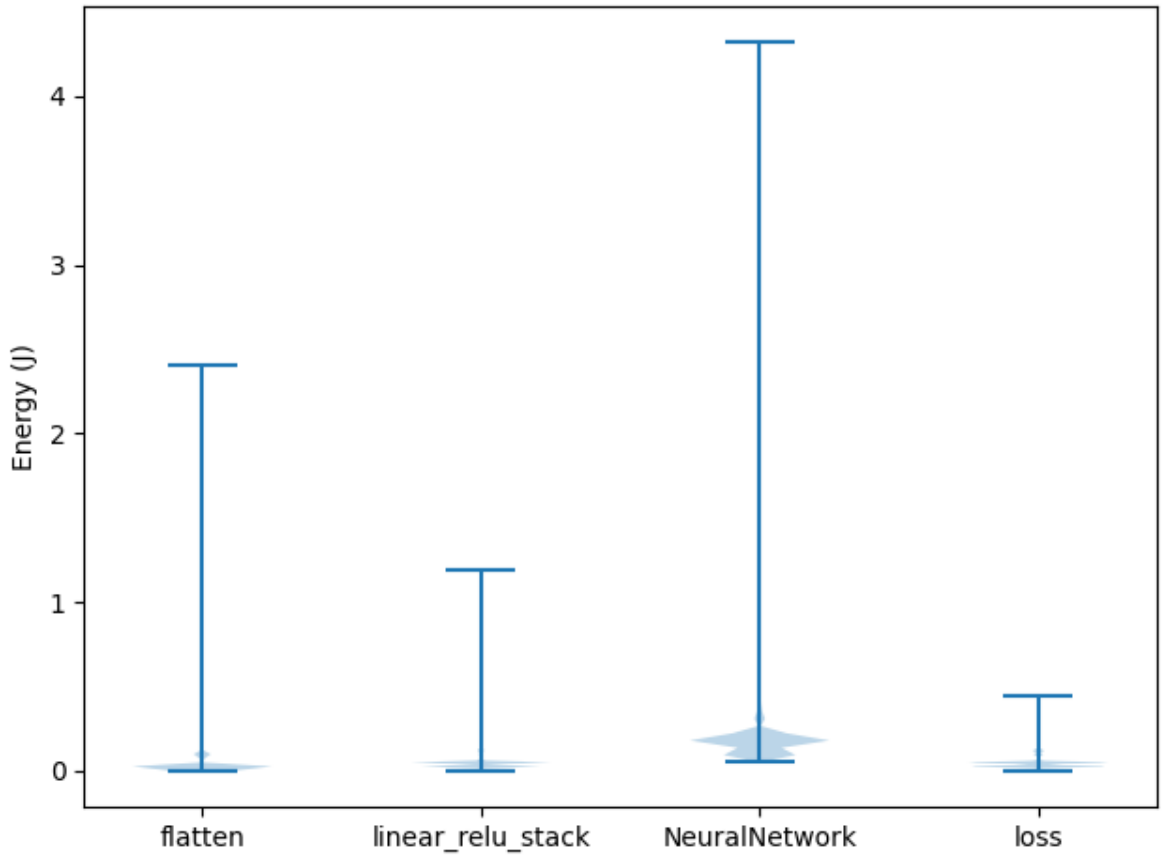
a backward pass. Furthermore if you attached also a loss function to the profiler you can also see the consumption of the loss computation. Finally you can specify which named layers you are intrested in seeing.

```
ga_measure.visualize_data(layers=[], loss=True, phase='forward')
```



If you are intrested in seeing the distribution of energy consumption the model consumes during a batch pass you can also visualize a violinplot or a boxplot.

```
ga_measure.visualize_data(kind='violin', loss=True)
```


Energy (J)
0
1
2
3
4
flatten
linear_relu_stack
NeuralNetwork
loss

CHAPTER

FOUR

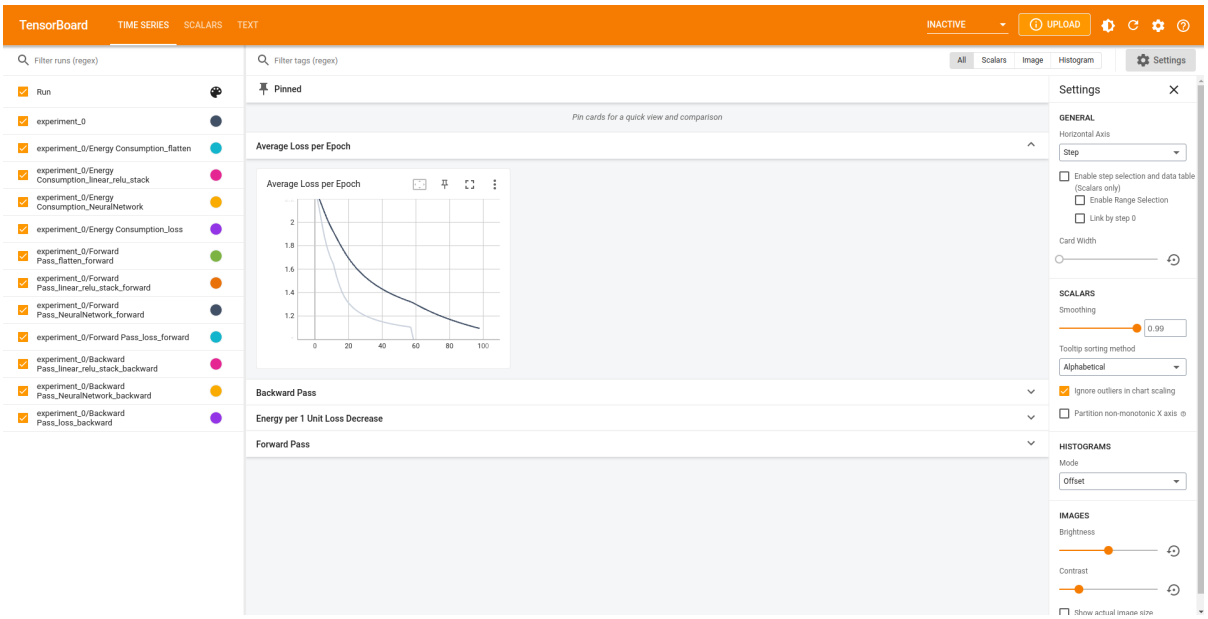
TENSORBOARD INTEGRATION

We set up a tensorboard dashboard to visualize the energy consumption data tracked by the profiler. After you trained your model tell the profiler to set up the tensorboard data using `set_tensorboard_stats()`.

```
from GATorch import GA

ga_measure = GA()
ga_measure.attach_model(model)
train(model)

ga_measure.set_tensorboard_stats()
```

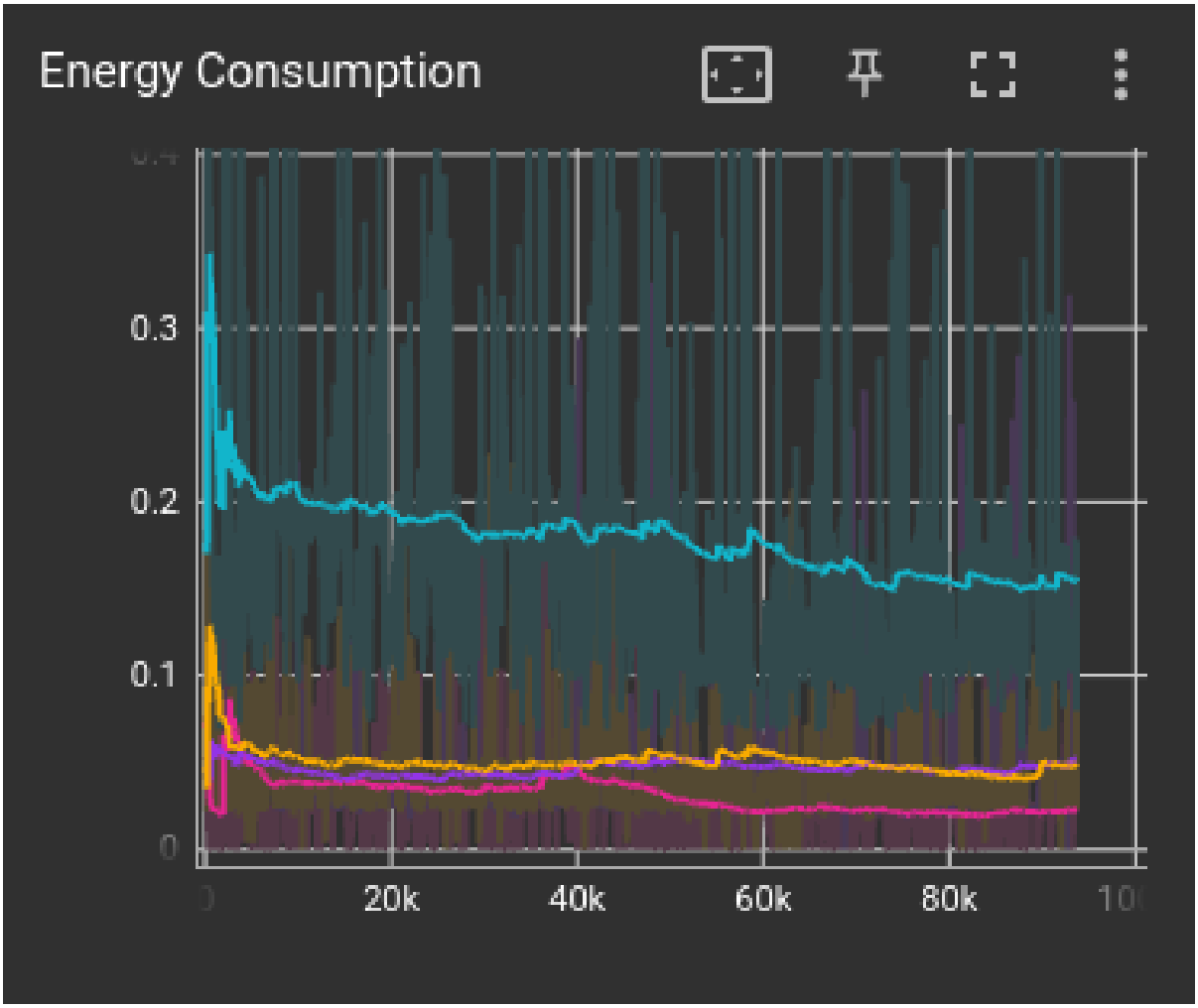
You can then access the tensorboard dashboard by running the following command from the terminal.

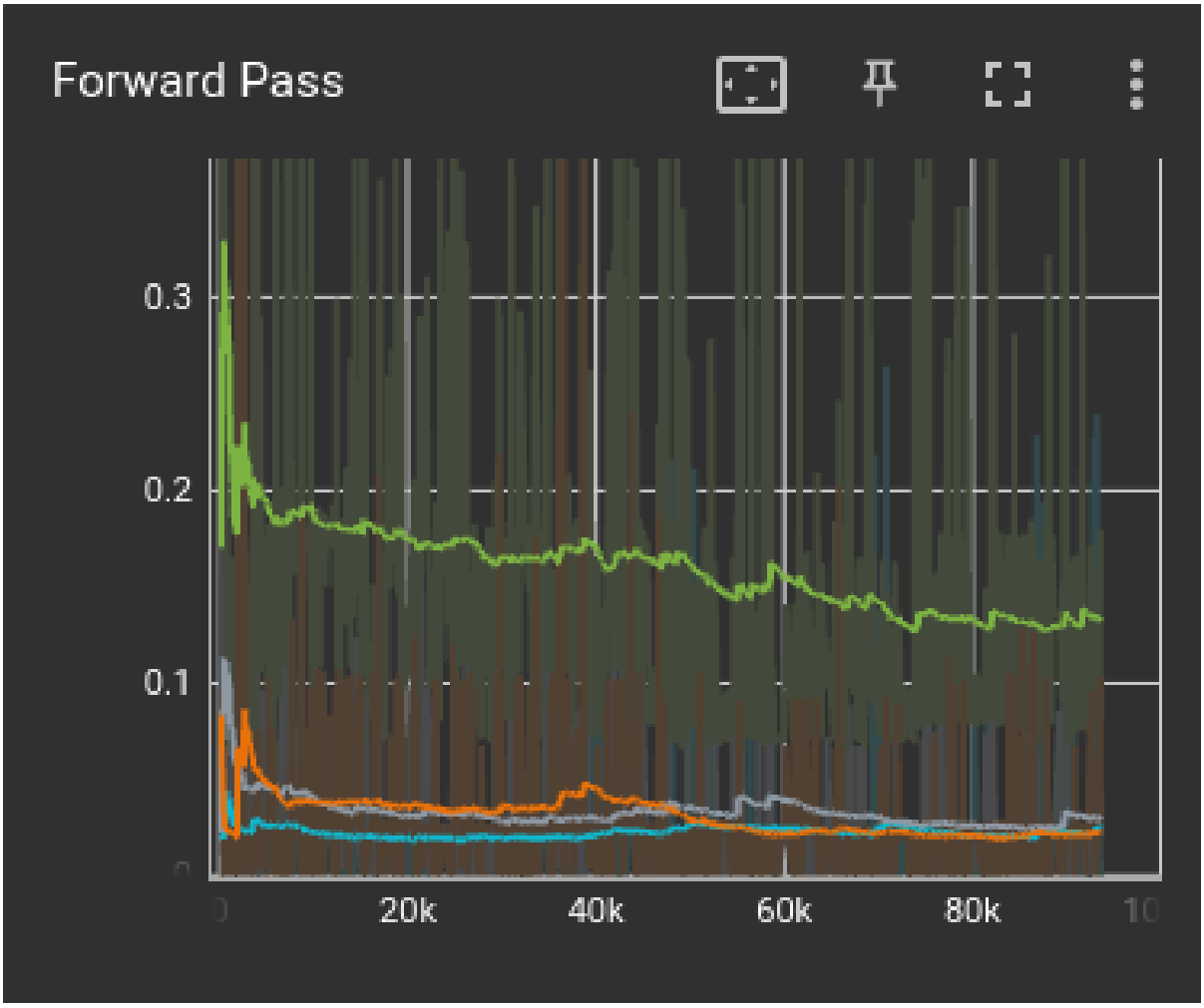
```
~$ tensorboard --logdir=runs
```

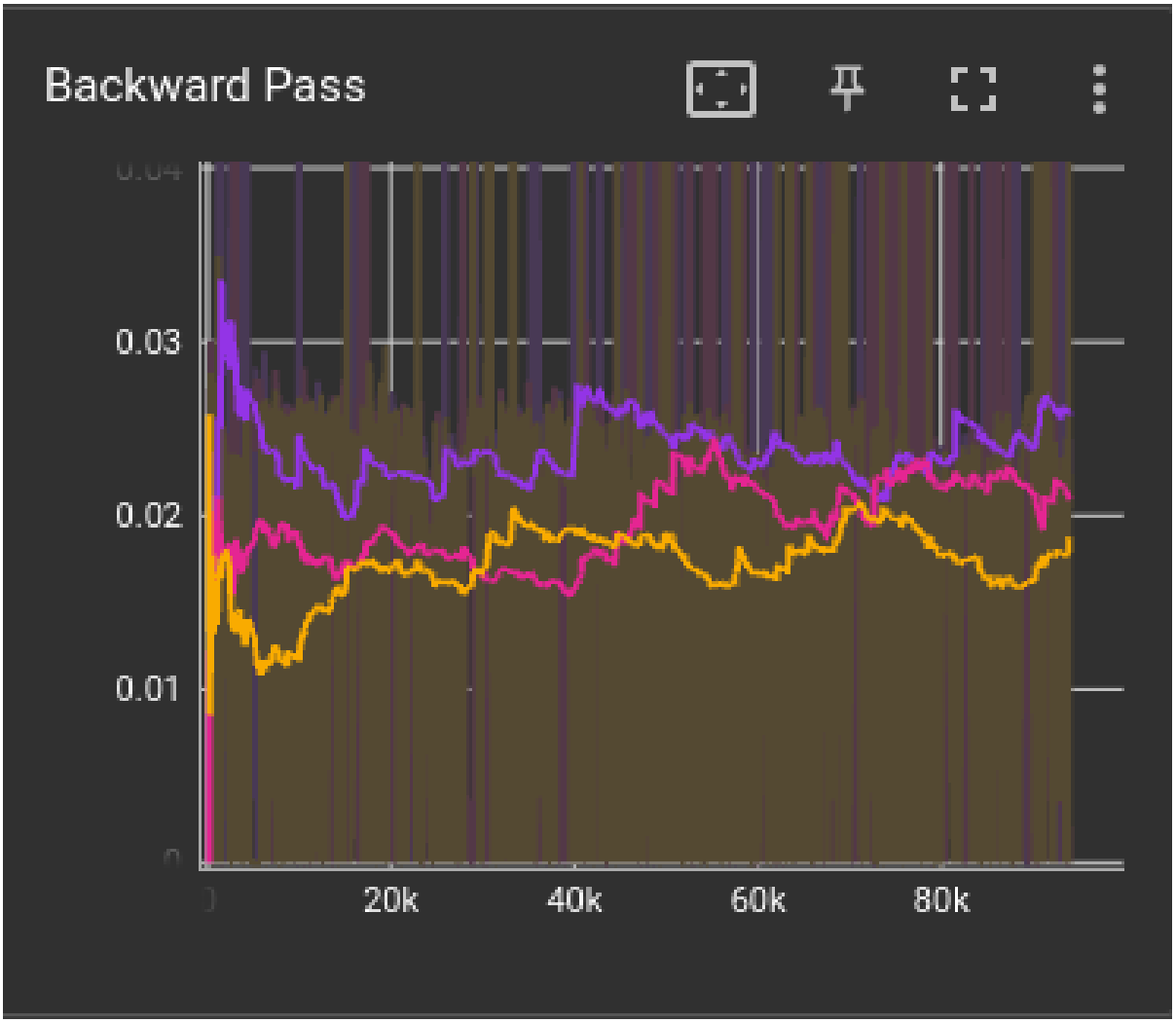


4.1 Graphs

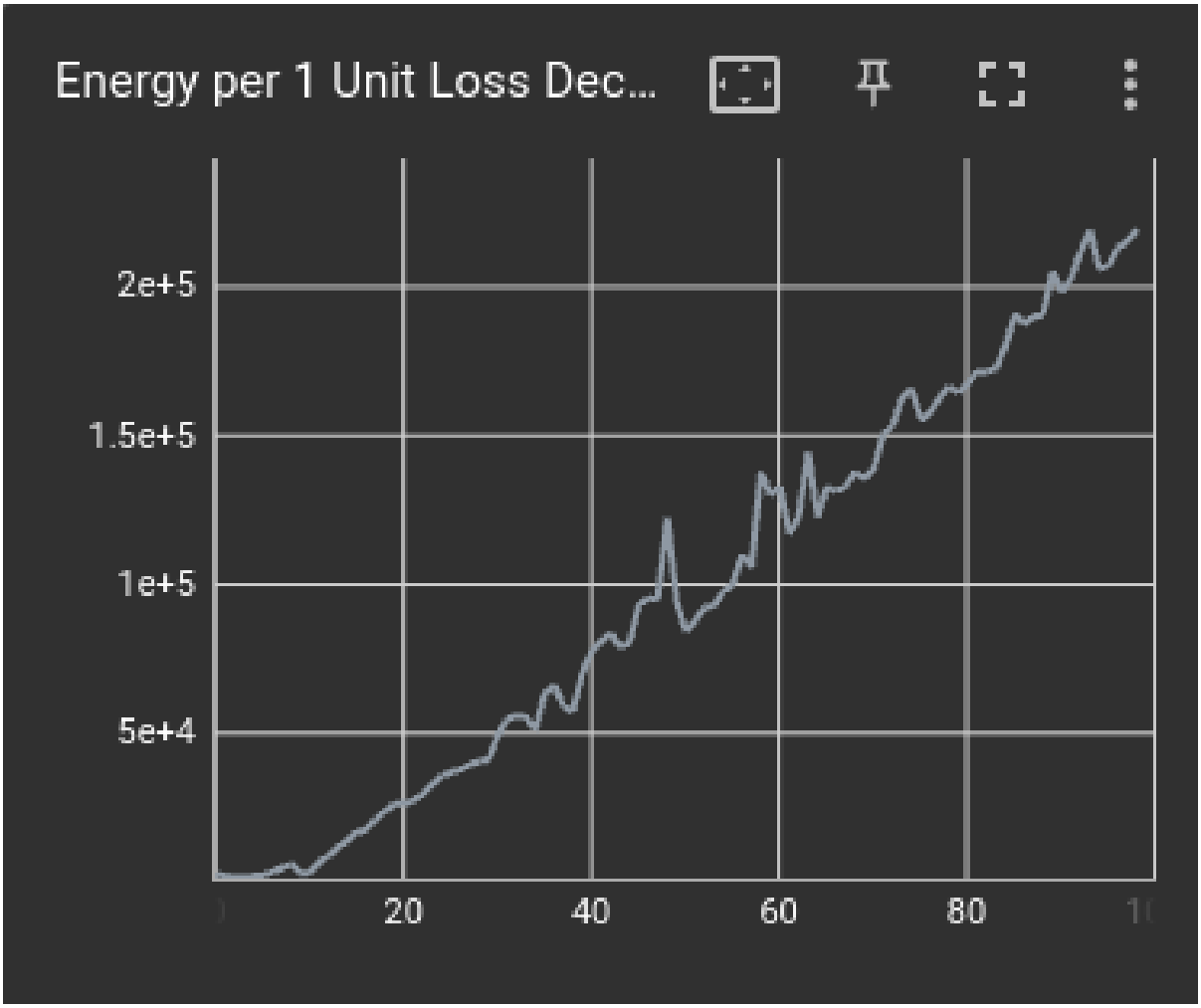
Under the `Scalar` tab of the dashboard you can see graphs indicating the energy consumption of the different components of your model through training. The graphs indicate the consumption during each full pass of the model or during the forward or backward pass.

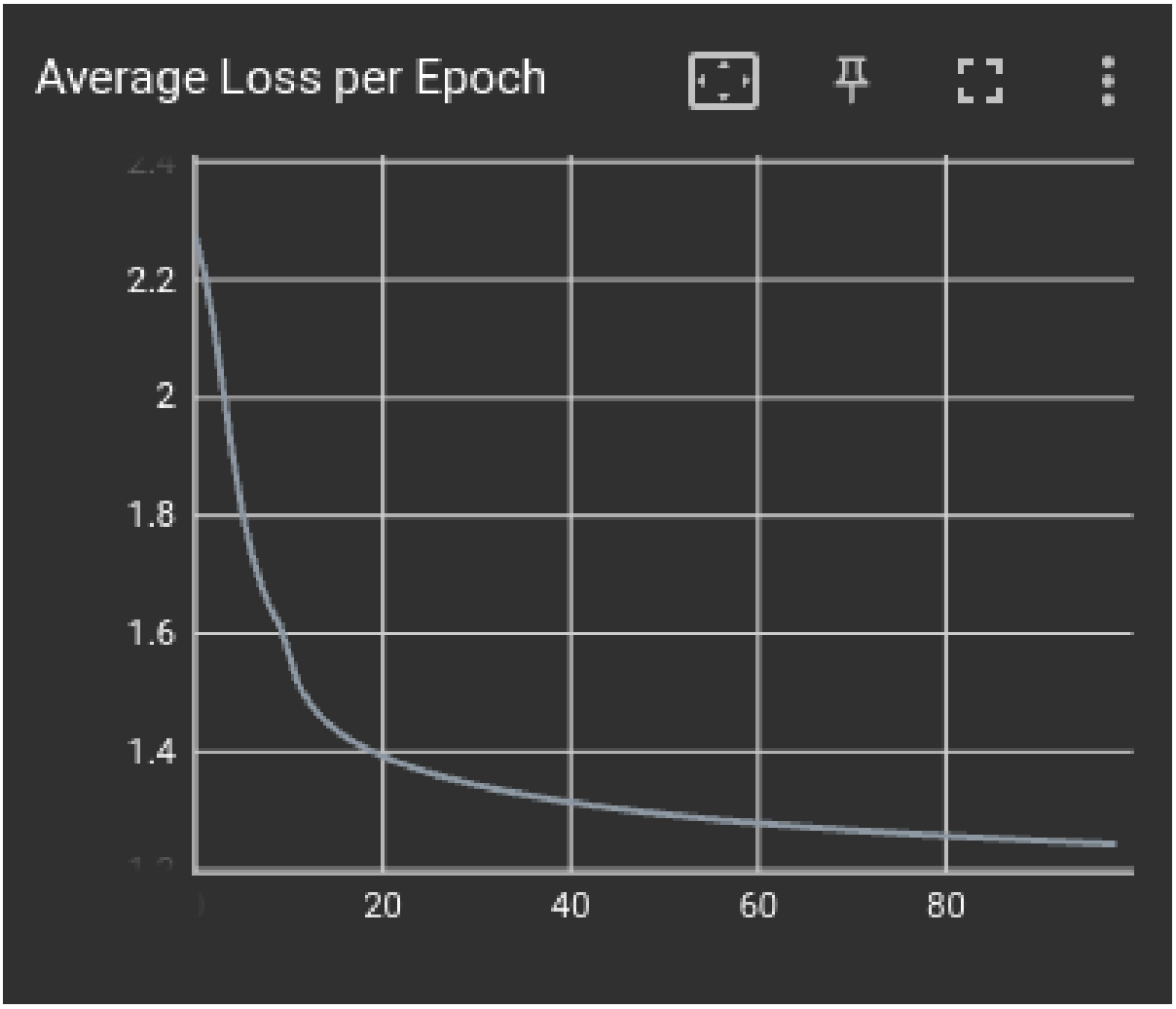



Forward Pass
0.3
0.2
0.1
20k
40k
60k
80k



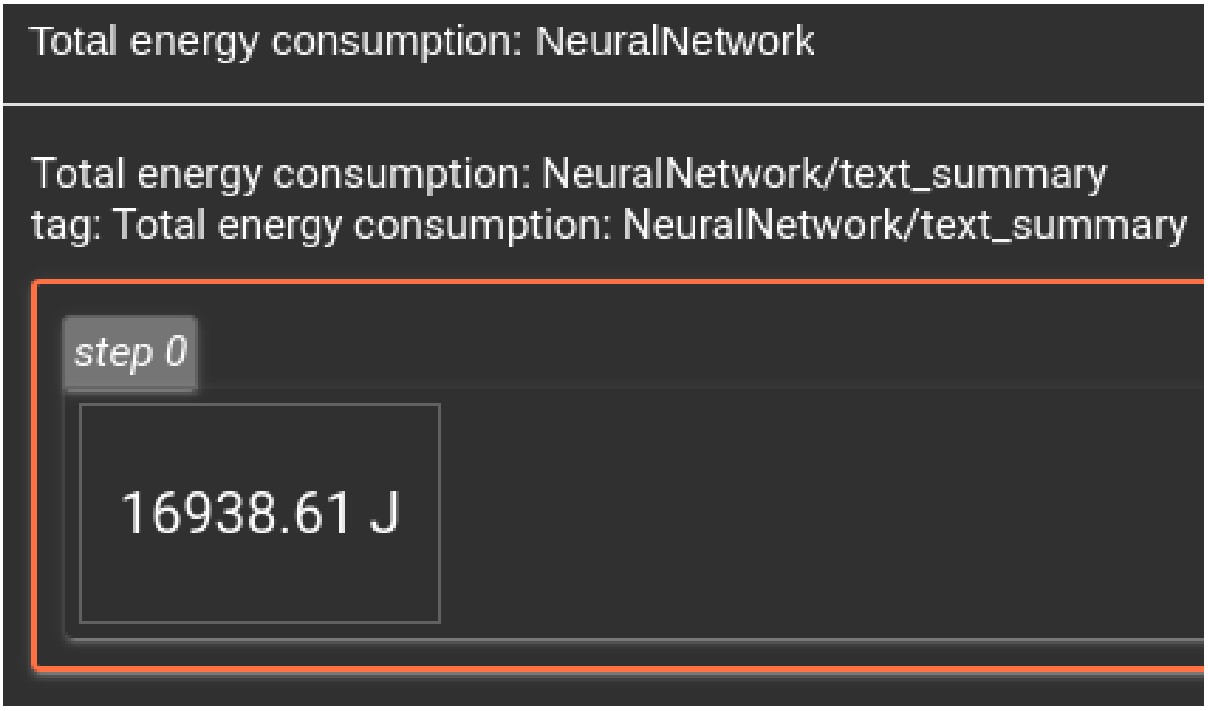
Furthermore if you attached also your torch `loss` function to the profiler, than the dashboard will also display how much energy is needed to improve the loss by a constant amount. When compared to the loss graph the data can help engineers understand when the energy consumption is getting too high compared to the relative improvement in loss.


Energy per 1 Unit Loss Dec...
2e+5
1.5e+5
1e+5
5e+4
20
40
60
80



4.2 Overall Consumption

Under the `Text` tab of the dashboard you can see the value of the overall energy consumption of the profiled model throughout the whole training loop.


Total energy consumption: NeuralNetwork
Total energy consumption: NeuralNetwork/text_summary
tag: Total energy consumption: NeuralNetwork/text_summary
step 0
16938.61 J

CHAPTER

FIVE

API DOCUMENTATION

GATorch principal class is `GA`.

5.1 GATorch.GA

class GATorch.**GA**

attach_model(*model*, *loss=None*, *named_layer=True*, *profiler='pyjoules'*, *disable_measurements=[]*)
Attach a model to the GA profiler. You can add also a loss function. The profiler will then track the energy consumption of the attached model.

Parameters

- **model** (*torch.nn.module*) – PyTorch model you want to track.
- **loss** (*torch.nn.module*) – Loss function to be passed for tracking the consumption of the loss computation.
- **named_layers** (*boolean*) – Set to False to not track individual named layers.
- **profiler** (*string*) – Typed of profiler used to track the energy consumption. Currently only 'pyjoules' is implemented.
- **disable_measurements** (*list*) – Set which hardware components you are not intrested in tracking between cpu, ram and gpu.

detach_model()
Detach the current model from the GA profiler.

get_full_measurements()
Get the full measurements collected by the profiler.

Returns The energy consumptions for each pass and model component.

Return type dict

get_mean_measurements()
Get the mean of the measurements collected by the profiler.

Returns The mean energy consumption for each model component.

Return type dict

get_sum_measurements()
Get the sum of the measurements collected by the profiler.

Returns The full energy consumption for each model component.

Return type dict

`get_losses()`

`to_pandas()`
Convert the energy measurements into a pandas.DataFrame object.

Returns The dataframe of the energy consumption for each model component.

Return type pandas.DataFrame

`to_csv(`*filename*`)`
Save the energy measurements into a csv file.

Parameters **filename** (*string*) – Name of the csv file.

`visualize_data(`*layers='all'*, *complete_model=True*, *loss=False*, *phase='total'*, *kind='line'*, *smoothing=0.3*, *figsize=None*, *filename=None*`)`
Generate a matplotlib plot for the energy measurements.

Parameters

- **layers** (*list or string*) – Pass in a list which named layers you want to display. Pass 'all' if you want to see all of them.
- **complete_model** (*boolean*) – Set to False if you don't want to see the data for the complete model.
- **loss** (*boolean*) – Set to True to display also the loss function data.
- **phase** (*string*) – Select which phase to display between 'total', 'forward' or 'backward'.
- **kind** (*string*) – Select which type of plot to generate between 'line', 'violin' or 'box'.
- **smoothing** (*float*) – Only used for lineplots. Value between 0 and 1 used to add smoothing to the displayed data.
- **figsize** (*Tuple*) – Size of the figure generated.
- **filename** (*string*) – Pass a filename if you want to save the image created.

Returns The matplotlib axes containing the plot.

Return type Axes

`start_tracker_emissions(`*save_to_file=False*`)`
Start a tracker for carbon emission. Implemented using codecarbon.

Parameters **save_to_file** (*boolean*) – Save the tracking results in a file called 'emissions'.

`stop_tracker_emissions()`
Stop the tracker for carbon emission.

Returns The total emissions produced by the model since the tracker was started.

Return type float

`set_tensorboard_stats(`*writer=None*, *experiment=0*, *named_layer=True*, *sample_size=500*`)`
Sets the tensorboard data that can be viewed through the tensorboard dashboard.

Parameters

- **writer** (*torch.utils.tensorboard.SummaryWriter*) – Optionally pass an already existing tensorboard writer.
- **experiment** (*int*) – Set an identifier for the displayed data.

- **named_layer** (*boolean*) – Indicate if the named layers should also be displayed in tensorboard.
- **sample_size** (*int*) – Size of the sample size used to define a loss step. Used to compute the grap for energy per loss step.

reset(*emission_tracker=True*)

Reset all the measurements.

Parameters **emission_tracker** (*boolean*) – Indicates if the emissions tracker needs to be reset as well.

INDEX