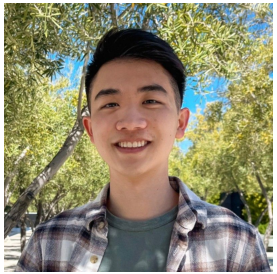
Tutorial: Using the NeuroBench Harness

NICE 2025

Benedetto Leto



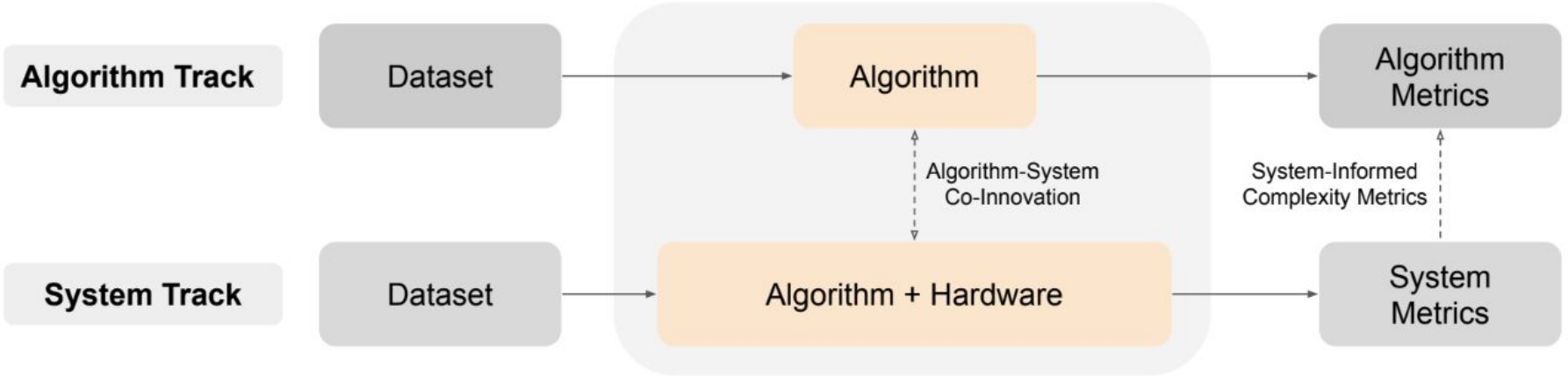
Jason Yik

NeuroBench

Benchmarking under a **common framework** aligns research, identifies best practices, and drives technological progress.

Inclusive **Actionable** **Iterative**

NeuroBench


Algorithm Track
Dataset
Algorithm
Algorithm Metrics
Algorithm-System Co-Innovation
System-Informed Complexity Metrics
System Track
Dataset
Algorithm + Hardware
System Metrics

NeuroBench

Model	R^2	Footprint (bytes)	SynOps		
			Dense	Eff_MACs	Eff_ACs
ANN Baseline	0.5755	27160	6236	4970	0
SNN Baseline	0.5805	29248	7300	0	413
AEGRU	0.6982	45520	54283	25316	0
RSNN-L	0.6978	4833360	1206272	0	42003
RSNN-S	0.6604	27144	13440	0	304
ConvGRU	0.6209	26568	4947	627	247

Tutorial Outline

Presentation

Examples / Documentation Walkthrough

Novel Benchmarking Notebook

- L^2MU, a spiking neuron SSM

Open Discussion

Tutorial Focus: The Algorithm Track Harness

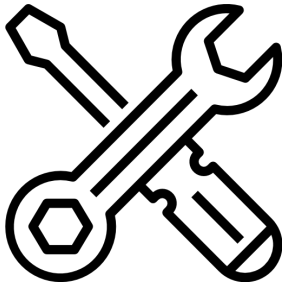
Algorithm Track

- Hardware-agnostic benchmarking of cost metrics
- Open-source research tool (harness) for automatic benchmarking
- Inference-focused, no existing support for training benchmarking

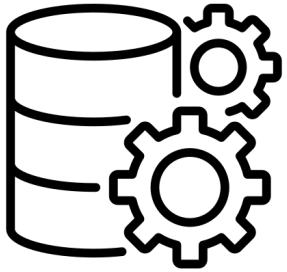
What about hardware system benchmarking?

- Supporting research tooling will need a lot more hardware maturity
- Extensive specifications for application setups, measurement methods are available for ASC and QUBO tasks, as well as official results from Loihi/Xylo
  - Available for you to compare your designs

Algorithm Track Harness



Automatic benchmarking for neural network models



Datasets, Data Processing, Metrics

Currently compatible with **torch**-based libraries

Actively maintained and built for extensibility

```
pip install neurobench
```

Is it really automatic benchmarking?

- Not everyone uses the same research frameworks
  - Norse, SpikingJelly, snnTorch, …
  - PyTorch-based, JAX-based, …
  - Continuous-time, analog, …

- The tooling is as automatic as possible
  - Simplicity and extensibility are the goals
  - Simple enough for you to validate and customize for your method
  - Extensible to your research flow

Places to Find Information

Repository top: https://github.com/NeuroBench/neurobench/tree/main

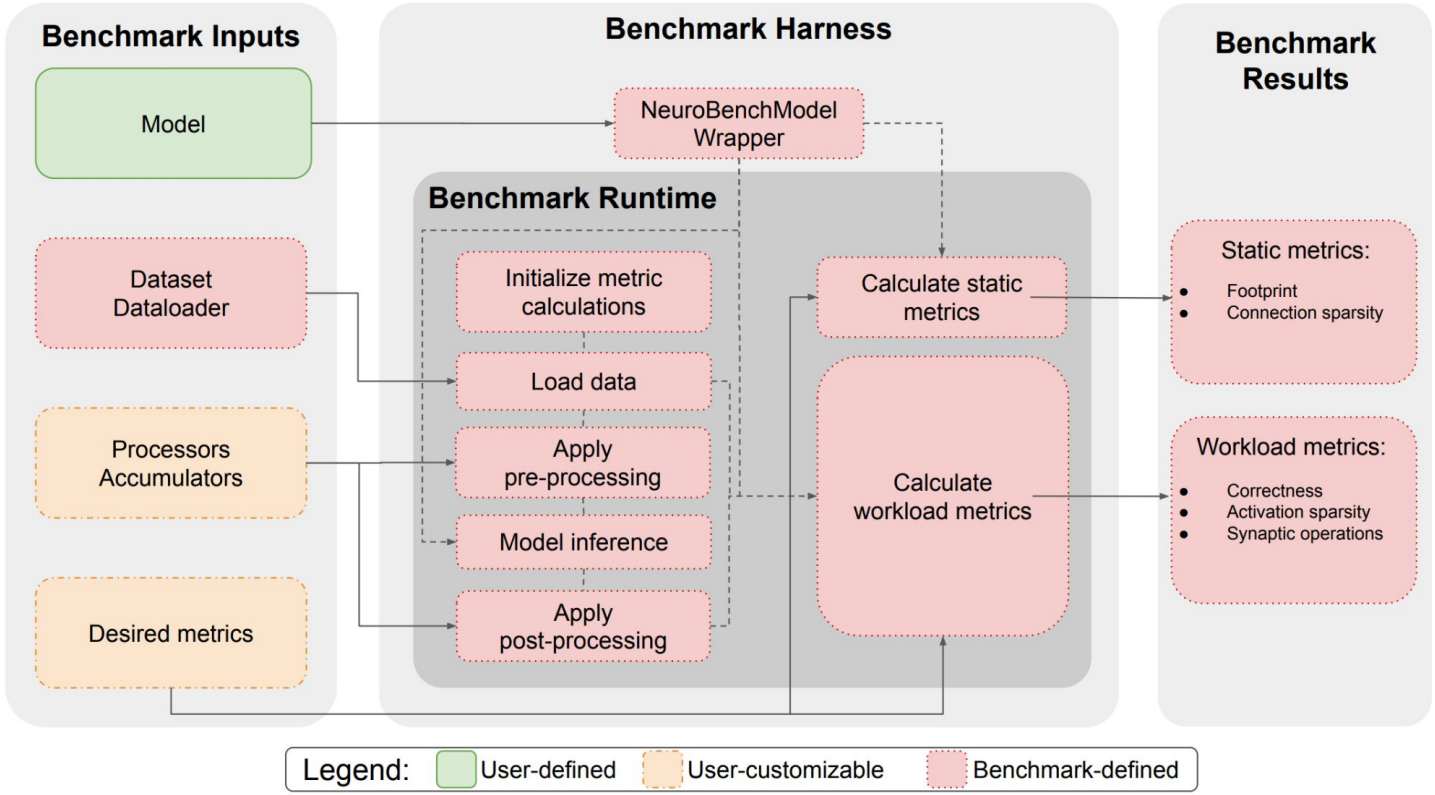
- Everything can be accessed from the README
- Dev branch: https://github.com/NeuroBench/neurobench/tree/dev

Documentation website: https://neurobench.readthedocs.io/en/latest/

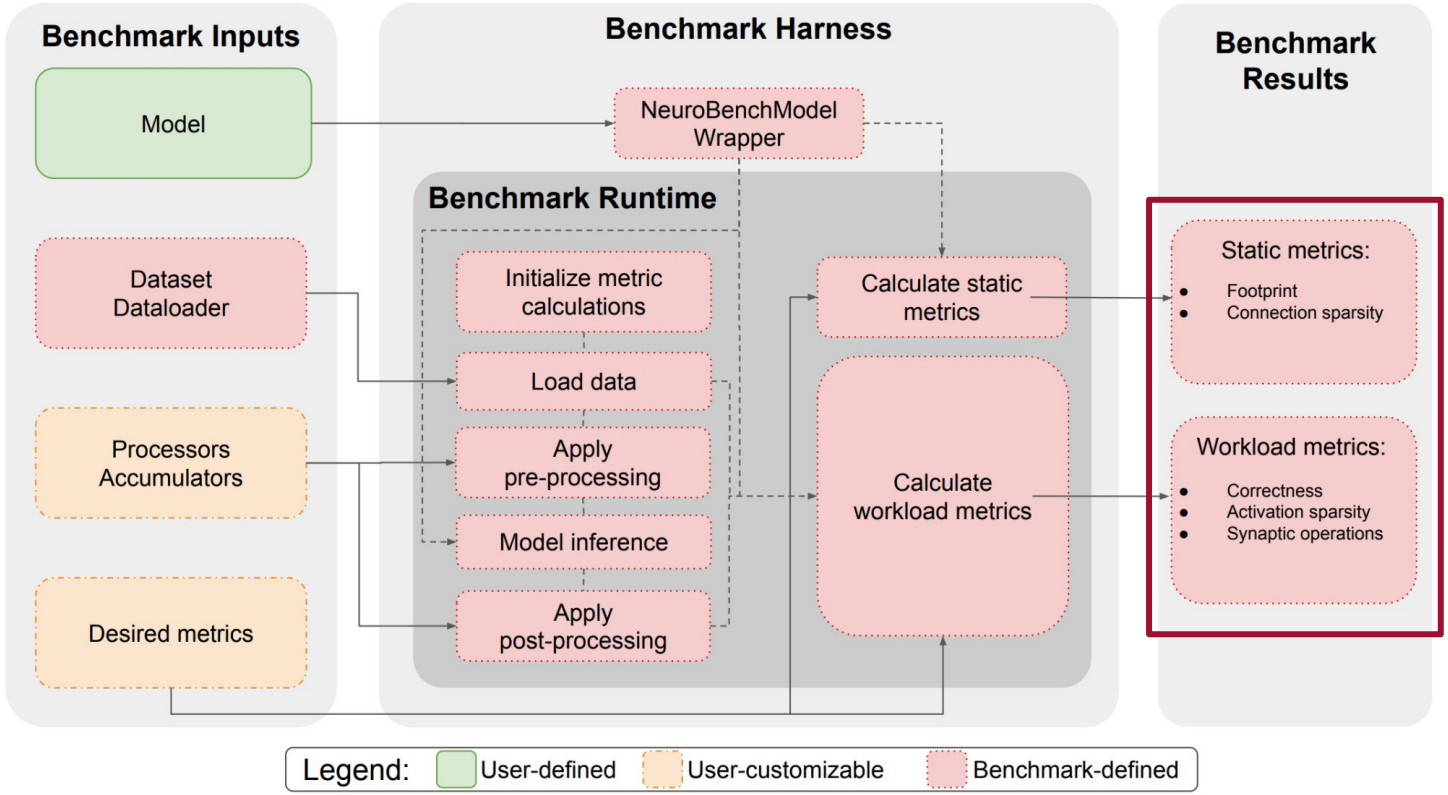
Example scripts (trained models):
https://github.com/NeuroBench/neurobench/tree/dev/examples

Leaderboard of benchmark results:
https://github.com/NeuroBench/neurobench/blob/dev/leaderboard.rst

Algorithm Track Harness



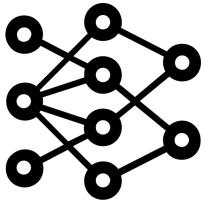
Algorithm Track Harness - Metrics



Algorithm Track Metrics: No influence of runtime platform

Static Metrics:

Footprint

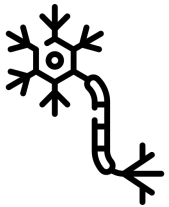


Connection Sparsity

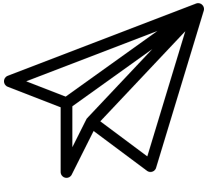
Execution Rate*

Workload Metrics:

Accuracy

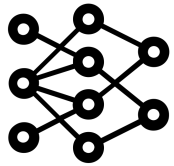


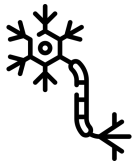
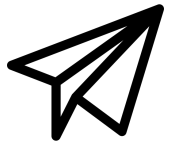
Activation Sparsity



Synaptic Operations

Overview of Included Metrics

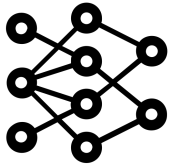



- **Footprint**
  - Static, Memory requirement of parameters and buffers
- **Connection sparsity**
  - Static, Ratio of zeroes in model weights
- **Accuracy**
  - Workload, Task-defined
- **Activation Sparsity**
  - Workload, Ratio of zeroes in neuron activations (LIF, ReLU)
- **Synaptic Operations**
  - Workload, Number of times an activation triggers with a weight (avg over all forward passes)
  - Dense: accounts for all operations, even zero ops
  - Effective MACs: non-zero ops only, for non-binary activations
  - Effective ACs: non-zero ops only, for binary activations
  - !! Expensive to calculate !! Get rid of this metric for faster benchmark run (3x)
- Formal metric specifications are in paper

Static Metrics

Footprint



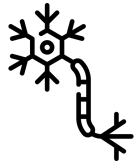
Connection Sparsity

- Evaluate fixed properties of the model, not data-dependent
- Computed once per benchmark run

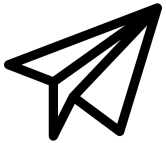
```
class StaticMetric(ABC):
    def __call__(self, model: NeuroBenchModel) -> float
```

Workload Metrics

Accuracy



Activation Sparsity



Synaptic Operations

- Data dependent, computed batch-by-batch
- By default, the metric is averaged over the batch size
  - e.g., classification accuracy
- Also support metrics which use more complex accumulations
  - e.g., mAP (object detection), R^2 (regression)
- Utilizes hooks (callback functions) in order to extract per-layer (per-pytorch module) information
  - e.g., activation sparsity, synaptic operations

Standard Workload Metric: Averaged over Batches

- In addition to the model, the interface uses the predictions and the batch data (processed data and targets)
- Hooks are necessary to extract per-module information from the execution

```
class WorkloadMetric(ABC):
    def __init__(self, requires_hooks: bool = False)
    def __call__(self, model: NeuroBenchModel, preds: Tensor, data:
tuple[Tensor, Tensor]
    ) -> float
```

Accumulated Workload Metric: Define how to calculate

- __init__ initializes state variables
- __call__ updates the state variables for each batch
- compute( ) returns the current metric value
- reset( ) resets the state, useful for successive runs

```
class AccumulatedMetric(WorkloadMetric):
    def compute(self) -> float
    def reset(self) -> None
```

Accumulated Metric Example: R^2

https://github.com/NeuroBench/neurobench/blob/main/neurobench/metrics/workload/r2.py

Developing Custom Metrics

https://neurobench.readthedocs.io/en/latest/custom_metrics.html

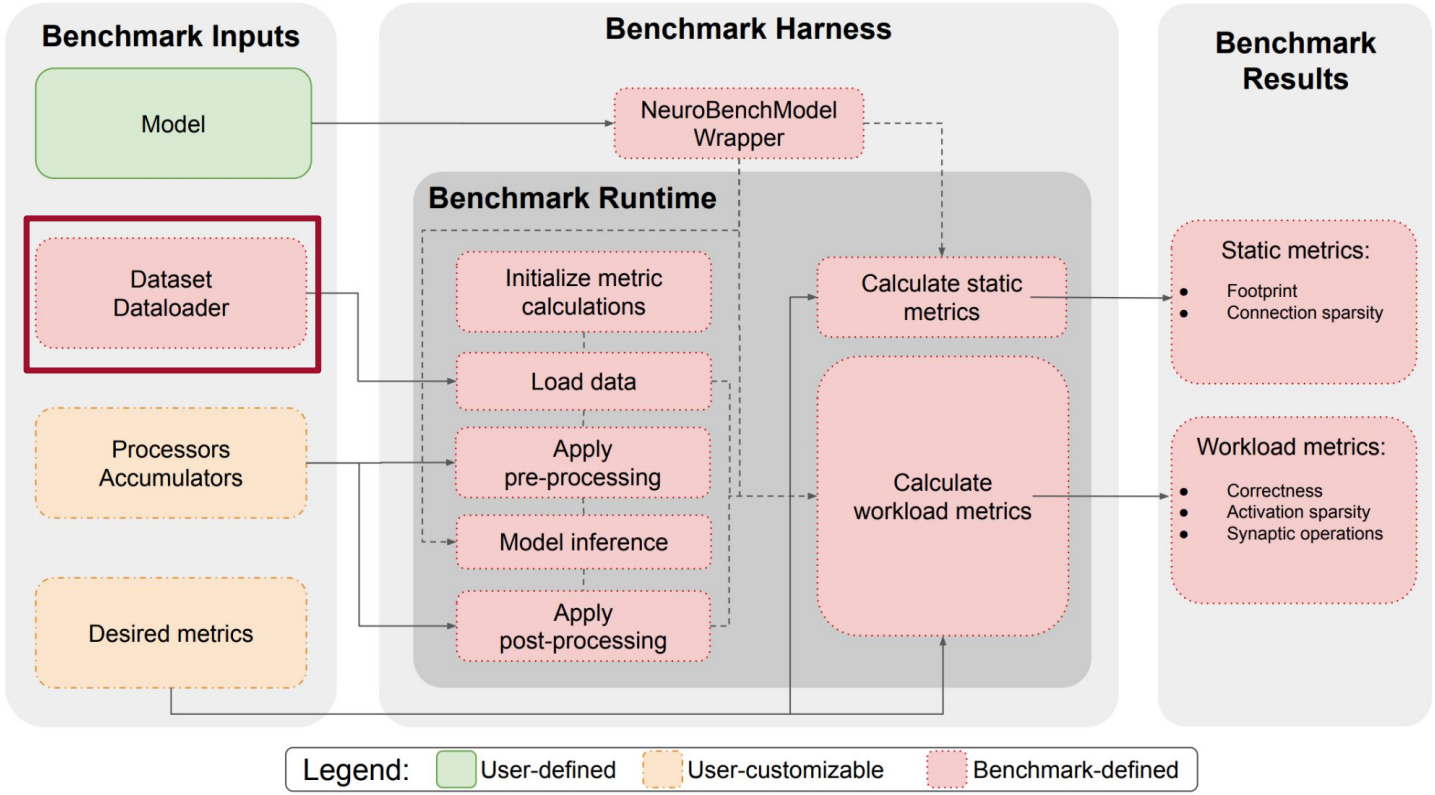
- Extend the abstract classes to define custom metrics
- Tutorial notebook and the docs above have examples of defining custom metrics

Neuron Dynamics

Robustness

Fan-out

Algorithm Track Harness - Datasets



Dataset Interface

- Directly uses pytorch Dataset and DataLoader
- https://pytorch.org/tutorials/beginner/basics/data_tutorial.html
- Native compatibility for dataset libraries like Tonic
  - https://tonic.readthedocs.io/en/latest/
- Benchmark run uses DataLoader to process batch-by-batch

Included Datasets (with Examples)

https://github.com/NeuroBench/neurobench/tree/main/examples

Keyword Few-shot, Continual Learning

Event Camera Object Detection

Primate Motor Prediction

Chaotic Function Prediction

Google Speech Commands

DVS Gesture

IMU Activity Recognition

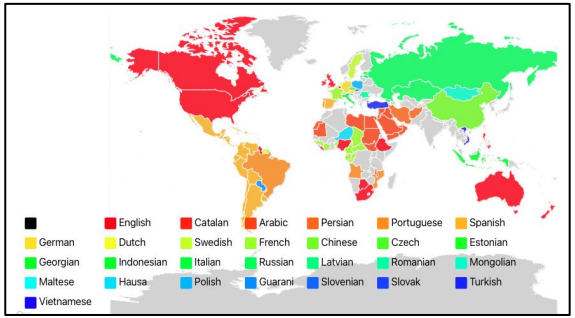
Keyword Few-shot Continual Learning

Application
Continual expansion of multilingual keyword dictionary using few training examples.

Dataset
Multilingual Spoken Word Corpus (MSWC) keyword dataset (50 languages, over 6000 hours).

Task
Model base-trains on 100 keywords across 6 languages. Then, it successively undergoes 10-way, 5-shot learning sessions of 100 total new keywords from 10 new languages.

Correctness
Classification accuracy is measured after each session, on all previously learned classes.



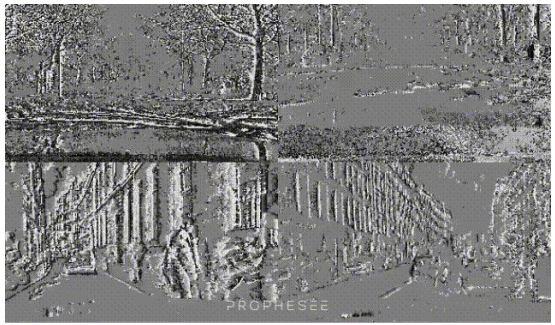
Event Camera Object Detection

Application
Real-time, energy-efficient / always-on automotive object detection, autonomous driving.

Dataset
Prophesee 1MP Gen 4 Automotive Detection dataset (14.65 hours, 3.5TB uncompressed).

Task
Detect car, two-wheeler, pedestrian. [train / val / test] split of [11.2 / 2.2 / 2.2] hours.

Correctness
COCO mean average precision (mAP).



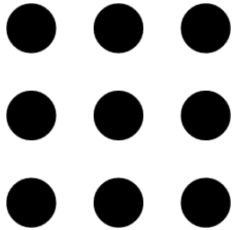
Primate Motor Decoding

Application
Sensorimotor biophysiological emulation, for prosthetics and brain-computer interfaces.

Dataset
Motor cortex recordings of two non-human primates engaged in reaching tasks (touch screen).

Task
Use cortical recording time-series to predict fingertip reach velocity in X and Y dimensions.

Correctness
R^2 of predicted velocities against ground truth.

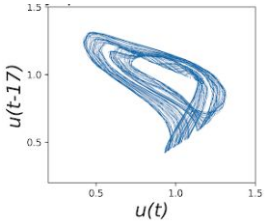


Chaotic Function Prediction

Application
Dynamic time-series forecasting, (markets, climate, signals, etc.). Also a small dimensional problem useful for prototyping emerging resource-constrained hardware (i.e., mixed-signal).

Dataset
Mackey-Glass time series, one-dimensional non-linear time delay differential equation.

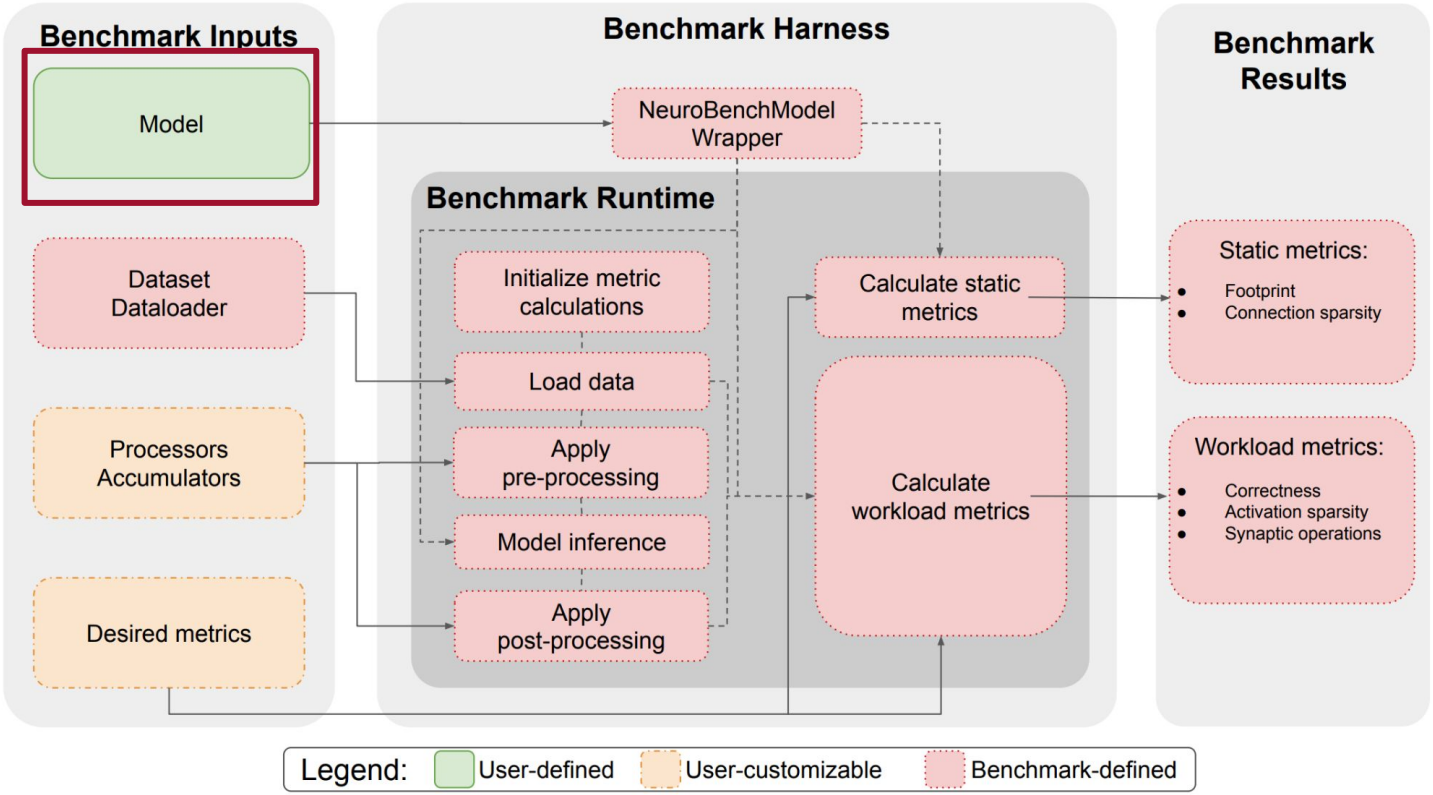
Task
Train using the first half of the generated time series, then autonomously forecast the second half.



Correctness
Symmetric mean absolute percentage error (sMAPE).

$$\frac{dx}{dt} = \beta \frac{x(t-\tau)}{1+x(t-\tau)^n} - \gamma x(t).$$

Algorithm Track Harness - Model



Model Interface

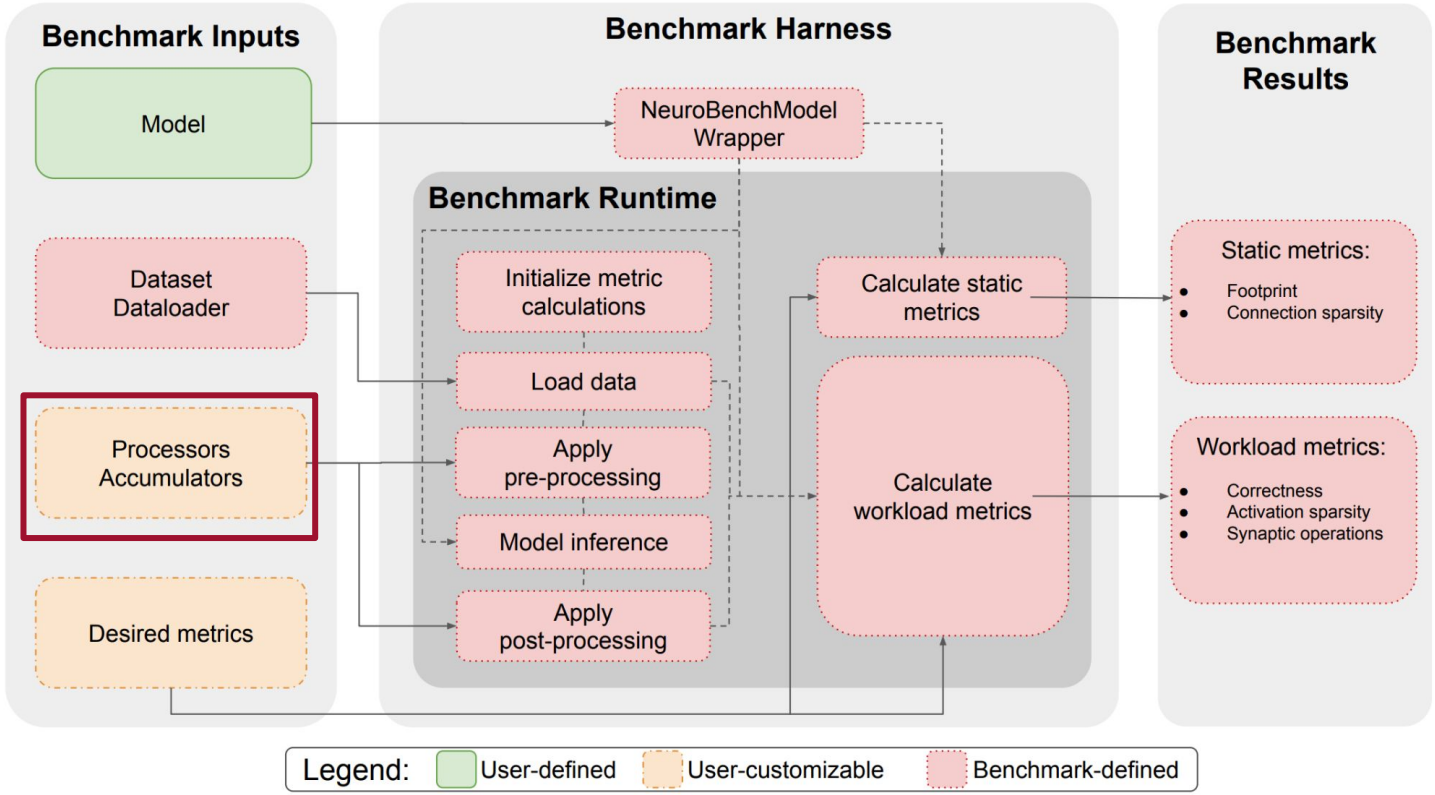
- General interface supports many frameworks
- Support in existing tool for connecting / collecting hooks
- Wrappers for TorchModel, SNNTorchModel
  - Includes boilerplate code for conventional execution
  - Verify that the execution applies correctly to your model

```
class NeuroBenchModel(ABC):
    def __init__(self)
    def __call__(self, batch)
    def __net__(self)
```

Modules as Post-processing Blocks

- Custom NeuroBenchModel can be used to describe sequence of module pipelines
  - e.g., obj detection head/box_coder: https://github.com/NeuroBench/neurobench/blob/dev/examples/obj_detection/benchmark.py#L52
- Currently, only module from __net__ will be hooked/measured

Algorithm Track Harness - Data Processing



Data Processor Interfaces

- Pre-processors applied to data before it is handed to model
- Post-processors take model output and transform to target shape
- Callables (e.g., lambda functions) matching interface can be used
- !! Processors not currently included in metrics evaluation, account for this in the limitations of the benchmarking study !!

```
class NeuroBenchPreProcessor(ABC):
    def __call__(self, dataset: tuple[Tensor, Tensor]) ->
tuple[Tensor, Tensor]

class NeuroBenchPostProcessor(ABC):
    def __call__(self, spikes: Tensor) -> Tensor
```

How much of all this should you know?

- Depends on your algorithm
- Using standard/supported execution flows, should be generally automatic
- Using custom synaptic layers, neurons, you probably need to build custom components
- Your responsibility to ensure the benchmark is correct for your work

Model	R^2	Footprint (bytes)	SynOps		
			Dense	Eff_MACs	Eff_ACs
ANN Baseline	0.5755	27160	6236	4970	0
SNN Baseline	0.5805	29248	7300	0	413
AEGRU	0.6982	45520	54283	25316	0
RSNN-L	0.6978	4833360	1206272	0	42003
RSNN-S	0.6604	27144	13440	0	304
ConvGRU	0.6209	26568	4947	627	247

Benchmark Top - Putting Everything Together

```
B = neurobench.Benchmark(model, dataloader, preprocessors, postprocessors, [static_metrics, workload_metrics])
```

- Define all the pieces and pass them into Benchmark

```
results = B.run(args)
```

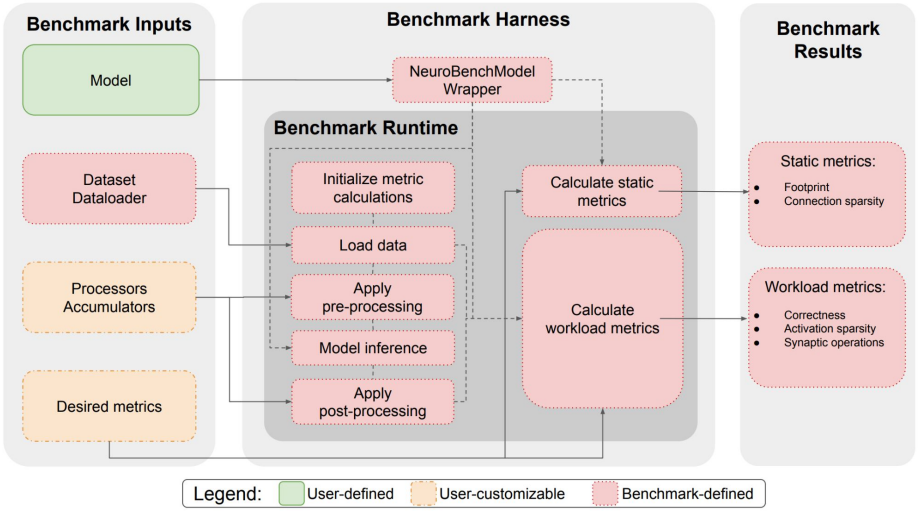
- Call run, returns dict of results

Various QOL Features

- Log results to JSON
- Export model to NIR / ONNX
- Get batch-by-batch results (verbose)
- Run without output (quiet)
- Specify GPU to run on (device)
- Update the processors, dataset for successive runs (i.e., for continual learning)

Harness Recap

- Simple/extensible
- Supports all official NeuroBench algorithmic benchmark tasks, and more
- You are ultimately responsible for correct benchmarking
- We are here to help you!



How to be involved

- **Using tasks for research**
  - Check out examples and open-source code (e.g., RSNN from ZenkeLab)
  - Validate your results
  - Extend using the standard interfaces (e.g., custom metrics)

- **Development**
  - Check out CONTRIBUTING.md
  - We meet every 2 weeks on Tuesdays to cover outstanding issues and next ideas

- **Best way to contact us**: File an issue, reach out via email
  - jyik@g.harvard.edu
  - benedetto.leto@studenti.polito.it

Next directions

Crossing over between the algorithm and hardware tracks

- Metrics that are more representative of hardware performance
  - e.g., Eff. SynOPs currently treats activation sparsity and weight sparsity the same
- Developing virtual-machine backend for network execution
  - Evaluate mapping and routing strategies
  - Metrics like traffic load, core operation counts, memory usage

Connecting to non-Pytorch backends

Closed-loop (gym environment) tasks

Tutorial Outline

Presentation

Examples / Documentation Walkthrough

Novel Benchmarking Notebook

- L^2MU, a spiking neuron SSM

Open Discussion