GraphNeT 2.0

A Deep Learning Library for Neutrino Telescopes

Münster Collaboration Meeting / March 2024

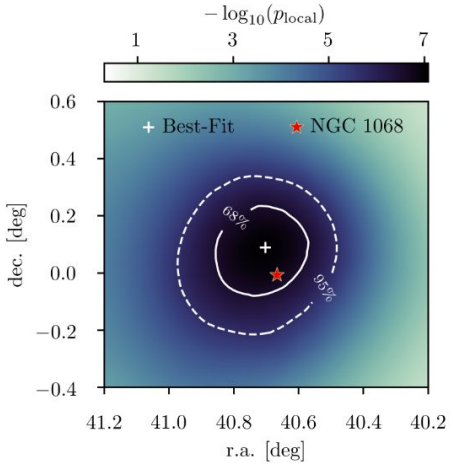
Rasmus Ørsøe

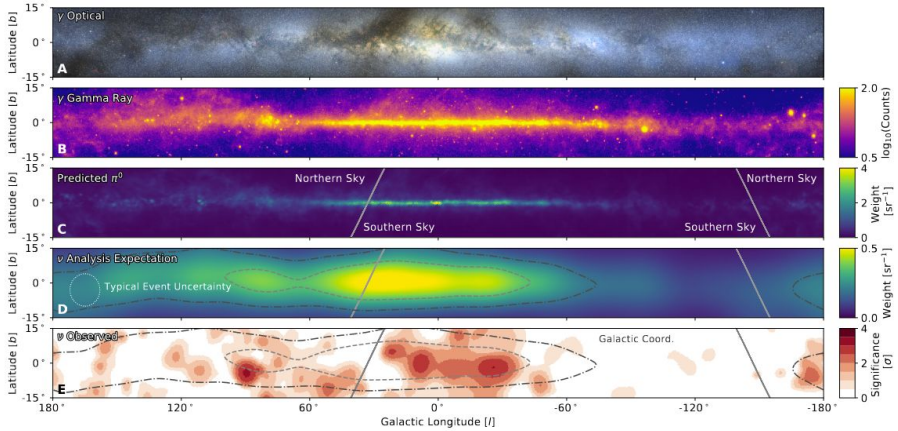
Technical University of Munich

Deep Learning is yet another thing to master ..

It is difficult to find physics analyses that does not depend on ML/DL in some capacity.

However, it is rare that one person masters both the physics and the deep learning





Examples of recent analyses in IceCube that relies on deep learning techniques

GraphNeT

A Deep Learning Library for Neutrino Telescopes

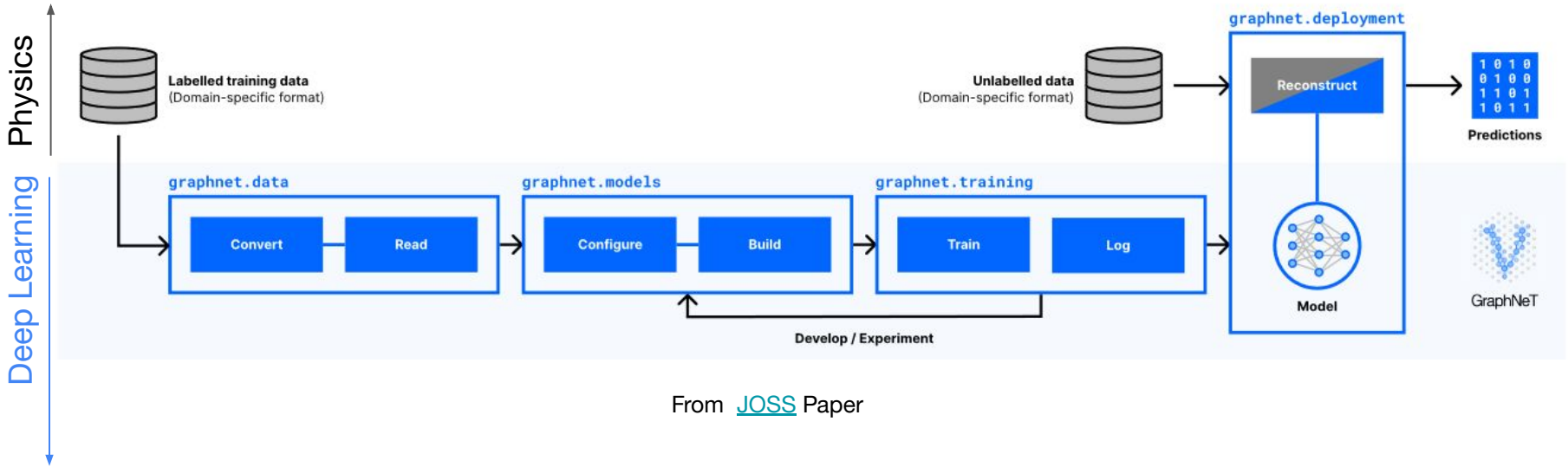
- **A framework for developing DL-based tools for neutrino telescopes**
- **One stop shop: from model development to deployment**
  - **Developers**
    - Everything needed to build, train and validate models from scratch
  - **Physics Domain Experts**
    - Can choose from a library of pre-trained models and apply them

- **Actively maintained with modern industry-standard code practices**
  - code conventions
  - proper documentation
  - Unit tests

graphnet-team/graphnet

GraphNeT

A Deep Learning Library for Neutrino Event Reconstruction



From JOSS Paper

graphnet-team/graphnet

GraphNeT

A Deep Learning Library for Neutrino Event Reconstruction

GraphNeT

Note

We recommend installing GraphNeT in a separate environment, e.g. using a Python virtual environment or Anaconda (see details on installation here).

Quick Start

PyTorch	PyTorch 2.2.*		w/o PyTorch
Your OS	Linux		Mac
CUDA	11.8	12.1	CPU

Run:

```
git clone https://github.com/graphnet-team/graphnet.git
cd graphnet

pip install -r requirements/torch_cpu.txt -e .[torch,develop]
```

When installation is completed, you should be able to run the examples.

graphnet.data

code for reading, writing and processing data

GraphNeT comes with modularized data conversion code that is parallelized.

It converts experiment-specific files to a deep learning friendly file format.

It follows a Reader/Writer structure.

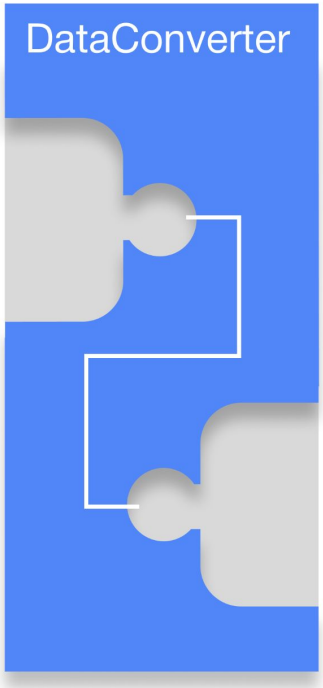


Illustration of an empty DataConverter

graphnet.data

code for reading, writing and processing data

The FileReader opens and extracts data from experiment-specific files, and converts it into a interim data format that the DataConverter can handles.

An I3Reader is already implemented in GraphNeT, making data conversion available for IceCube.

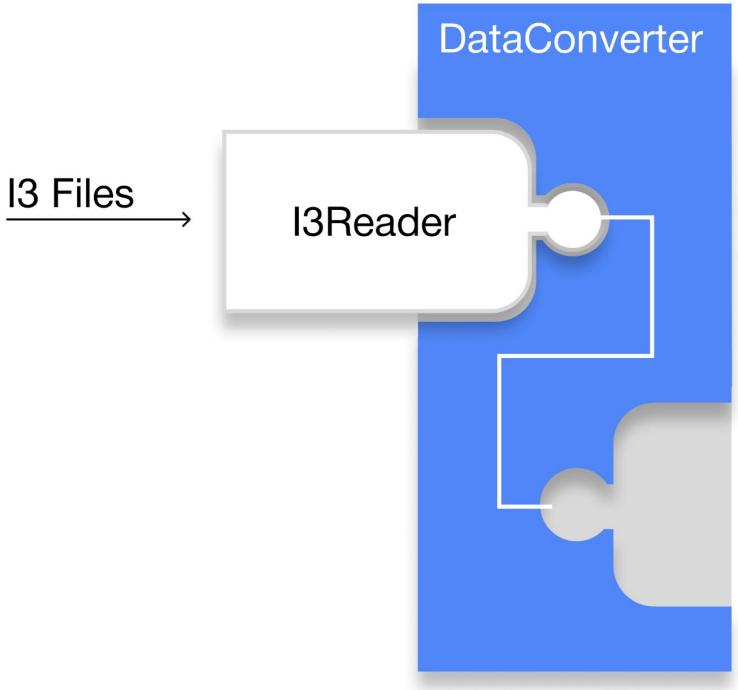


Illustration of a DataConverter configured to read I3 files.

graphnet.data

code for reading, writing and processing data

This interim data format is passed to the Writer, which saves it to disk in a deep learning friendly file format.

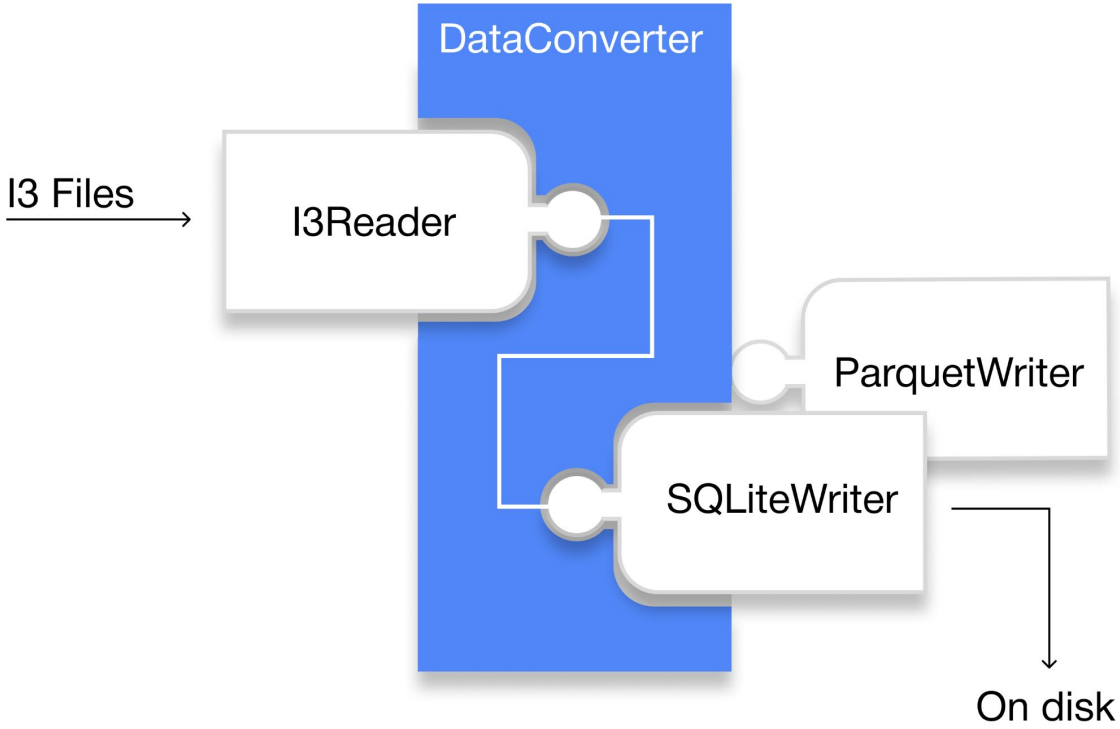


Illustration of a DataConverter configured to read I3 files and save as SQLite.

graphnet.data

code for reading, writing and processing data

With very few lines of code, this DataConverter is configured to extract data from I3 files and save the output as SQLite databases.

The conversion will run in parallelized using 10 CPUs.

```python
from graphnet.data import DataConverter
from graphnet.data.readers import I3Reader
from graphnet.data.writers import SQLiteWriter

# Instantiate DataConverter
converter = DataConverter(file_reader = I3Reader(..),
                          save_method = SQLiteWriter(..),
                          num_workers = 10,
                          outdir = '/home/outdir')

# Convert i3 files to supported backend in parallel
converter(input_dir = '/home/14000')

# Merge files (Optional)
converter.merge_files(outdir = '/home/outdir')
```

Code example for a DataConverter that converts i3 files to SQLite Databases.

graphnet.data

code for reading, writing and processing data

Experiment-specific datasets (also behind credential walls) and open-source simulation from Prometheus can be made available as CuratedDatasets through GraphNeT, making comparisons easier.

```python
from graphnet.datasets import PONESmall
from graphnet.models.graphs import KNNGraph
from graphnet.models.detector.prometheus import PONETriangle

# Download & instantiate dataset
graph_definition = KNNGraph(detector = PONETriangle())

dataset =  PONESmall(graph_definition = graph_definition,
                     download_dir = '/content/datasets',
                     train_dataloader_kwargs = {'batch_size': 500,
                                                'num_workers': 2,
                                                },
                     backend = 'sqlite')

# Pre-defined training/validation/test sets
train_dataloader = dataset.train_dataloader
val_dataloader = dataset.val_dataloader
test_dataloader = dataset.test_dataloader

# Ready to train
```

Code example of instantiating a public dataset in GraphNeT

graphnet.data

code for reading, writing and processing data

Different data representations can be built in graphnet, making it compatible with the established deep learning paradigms like

CNNs, **GNNs**, **RNNs**, **Transformers**, etc.

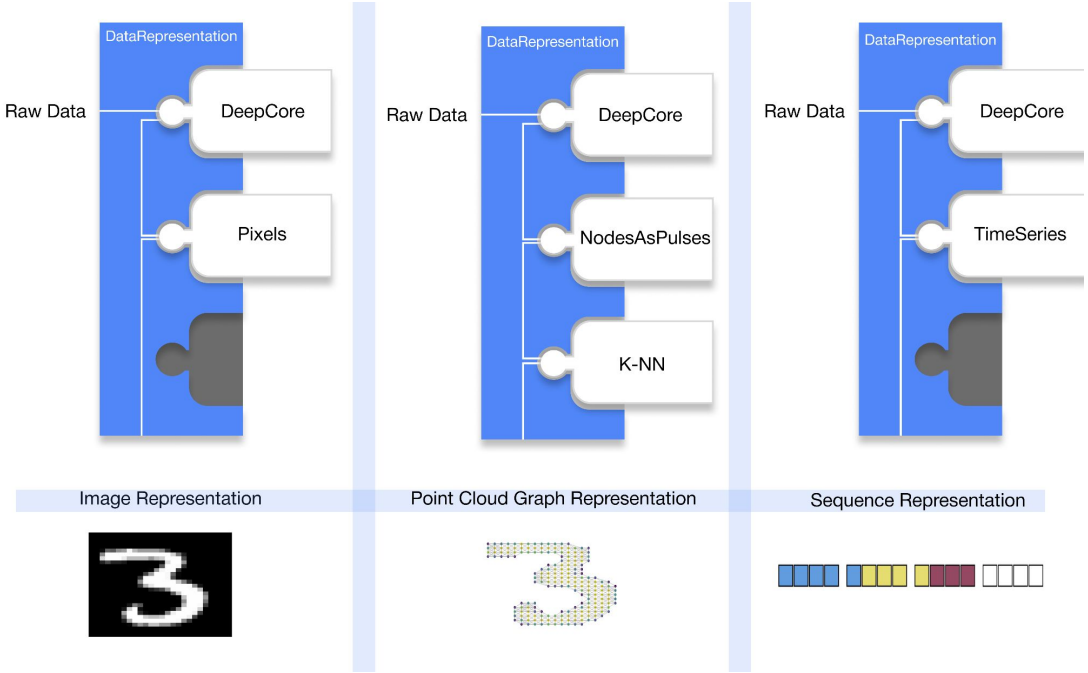


Illustration of how fundamentally different data representations can be generated in GraphNeT 2.0, making the framework compatible with established deep learning paradigms.

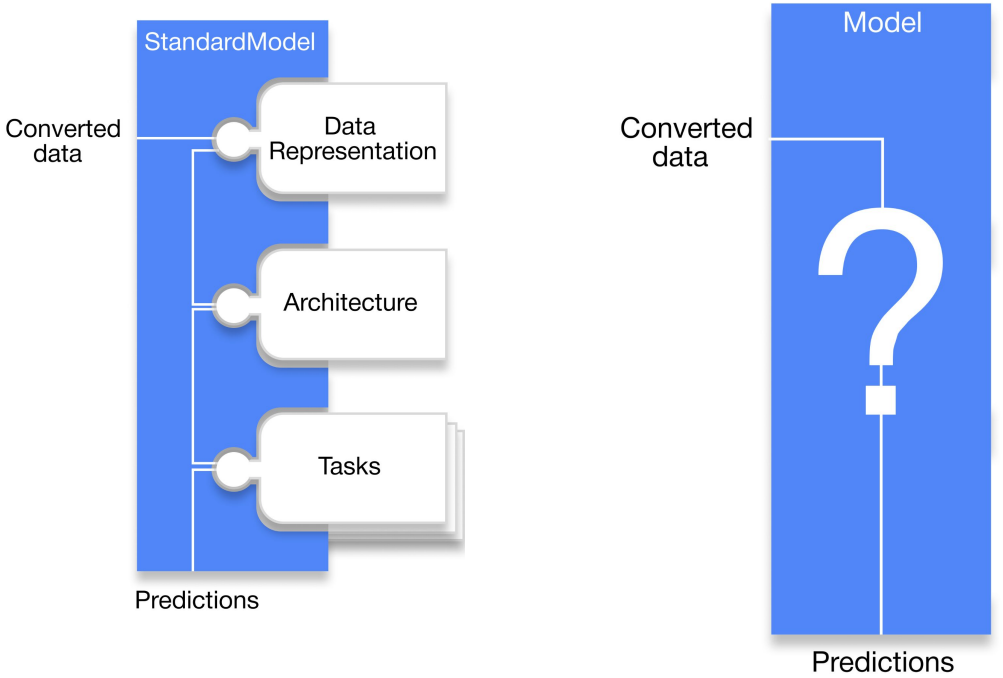
graphnet.models

code for building, configuring, training and saving models

StandardModel is a modularized model class where data representation, model architecture and physics task are interchangeable modules, allowing methods to be repurposed easily.

This class supports the vast majority of supervised learning tasks.

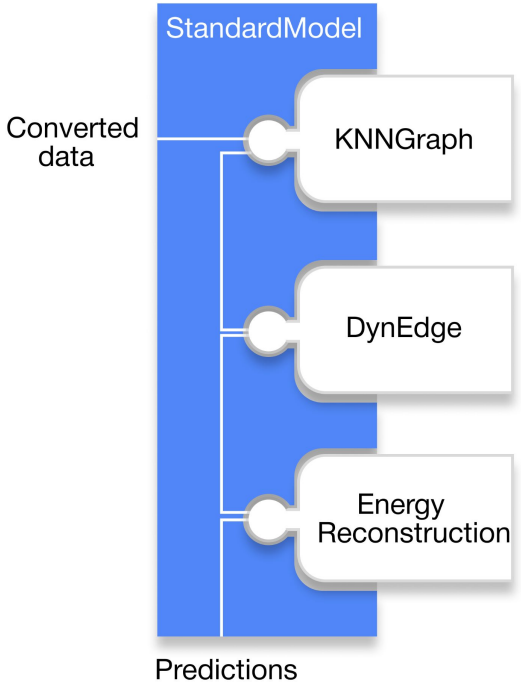
Model is a freeform model class with no rails, attended for niche methods and advanced users.



Left) StandardModel, a modularized DL model where components are interchangeable. **Right)** Freeform Model class with no rails.

graphnet.models

code for building, configuring, training and saving models



Conceptual illustration of the code to the right

```python
from graphnet.models.detector.icecube import IceCubeDeepCore
from graphnet.models.task.reconstruction import EnergyReconstruction
from graphnet.training.loss_functions import LogCoshLoss
from graphnet.models.graphs import KNNGraph
from graphnet.models.gnn import DynEdge

from graphnet.models import StandardModel

# Instantiate Modules
graph_definition = KNNGraph(detector = IceCubeDeepCore())
backbone = DynEdge(nb_inputs = graph_definition.nb_outputs)
task = EnergyReconstruction(loss_function = LogCoshLoss(),
                            hidden_size = backbone.nb_outputs)

# Build GraphNeT Model
model = StandardModel(
        graph_definition=graph_definition,
        backbone=backbone,
        tasks=[task]
        )
```

Example of defining a StandardModel in GraphNeT

graphnet.models

code for building, configuring and saving models

The StandardModel provides a very simple training syntax, significantly lowering the technical threshold for training complex models, without compromising on functionality.

```
# Train Model w. EarlyStopping On GPU 0
model.fit(train_dataloader = train_dataloader,
          val_dataloader = val_dataloader,
          max_epochs = max_epochs,
          gpus = [0])

 # Predict on Test Set Using GPUs
 results = model.predict_as_dataframe(dataloader = test_dataloader,
                                      gpus = [0])

 # Save model and predictions to disk
 model.save_state_dict('state_dict.pth')
 model.save_config('model_config.yml')

 results.to_csv('results.csv)
```

Example of training, predicting and saving

graphnet.models

code for building, configuring and saving models

Models in GraphNeT can be fully summarized using config files and their weights.

```python
from graphnet.models import StandardModel

# Re-Create Model From Config File
model = StandardModel.from_config('model_config.yml')

# Load Weights from Training Session
model.load_state_dict('state_dict.pth')


# Re-use model!
```

Example of re-instantiating a pre-trained model from configuration files.

graphnet.deployment

code for applying models to i3 files

Using the config files, models can be applied in parallel to i3 files without any expertise in either icetray or deep learning.

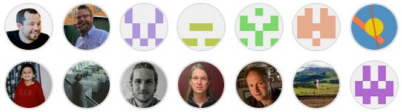
Models can also be used directly in native icetray scripts.

```python
from graphnet.deployment.i3modules import I3InferenceModule, GraphNeTI3Deployer

# Instantiate inference module with pre-trained model
deployment_module = I3InferenceModule(..
                                       state_dict = 'state_dict.pth',
                                       model_config = 'model_config.yml',
                                       model_name = 'DynEdge_Energy_Reco',
                                       ..)


# Instantiate deployer with inference module(s)
deployer = GraphNeTI3Deployer(graphnet_modules = [deployment_module],
                              num_workers = 10)

# Apply module(s) to input files in parallel
deployer.run(input_files = [..],
             output_folder = 'i3_files_with_predictions')
```

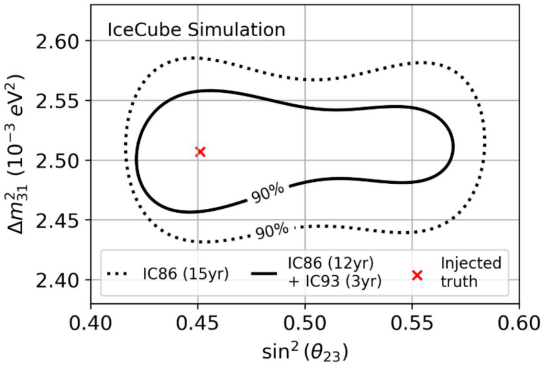
Example of using GraphNetI3Deployer to write model predictions into i3 files.

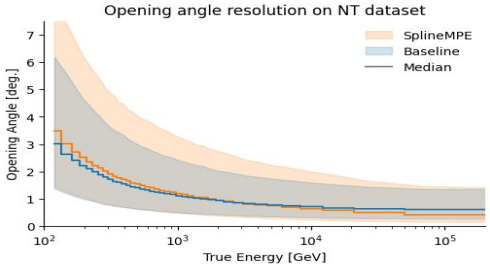
So..

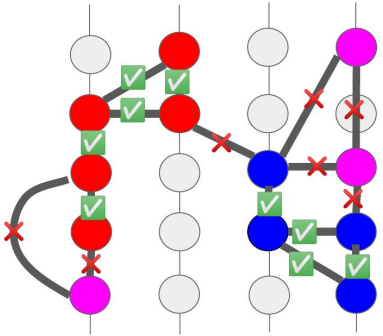
Contributors 20

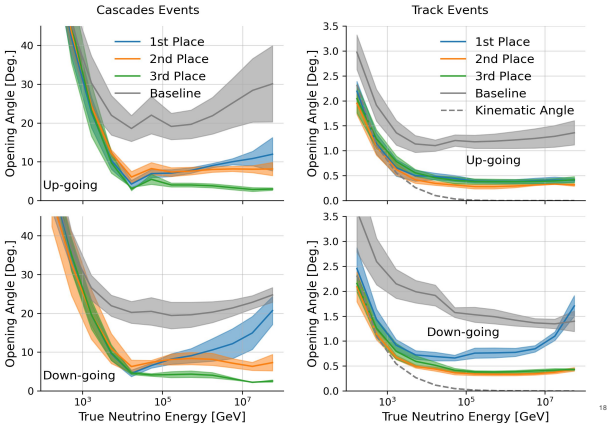


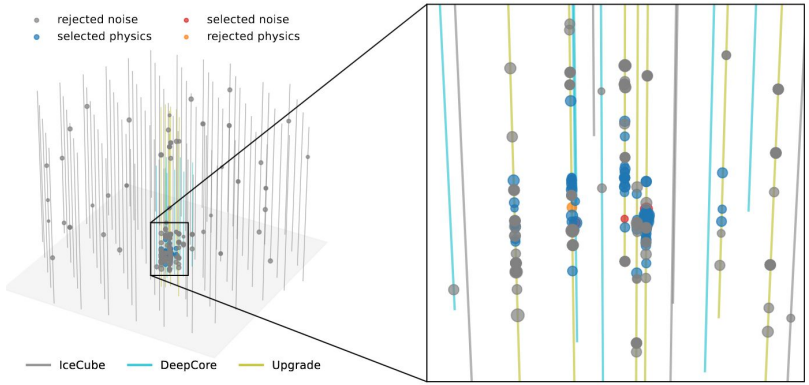
+ 6 contributors











You can use GraphNeT 2.0 to:

Experiment with different deep learning techniques **not just GNNs**

Compare with existing techniques, **e.g. the kaggle solutions**

Distribute your models to other people - **and to other experiments**

Apply these complex methods to your analyses - **without being an expert in them**

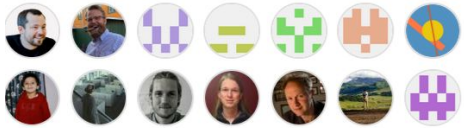
- Pulse
- Contributors
- Community
- Community Standards
- Traffic
- Commits
- Code frequency
- Dependency graph
- Network
- Forks

Forks

graphnet-team / graphnet

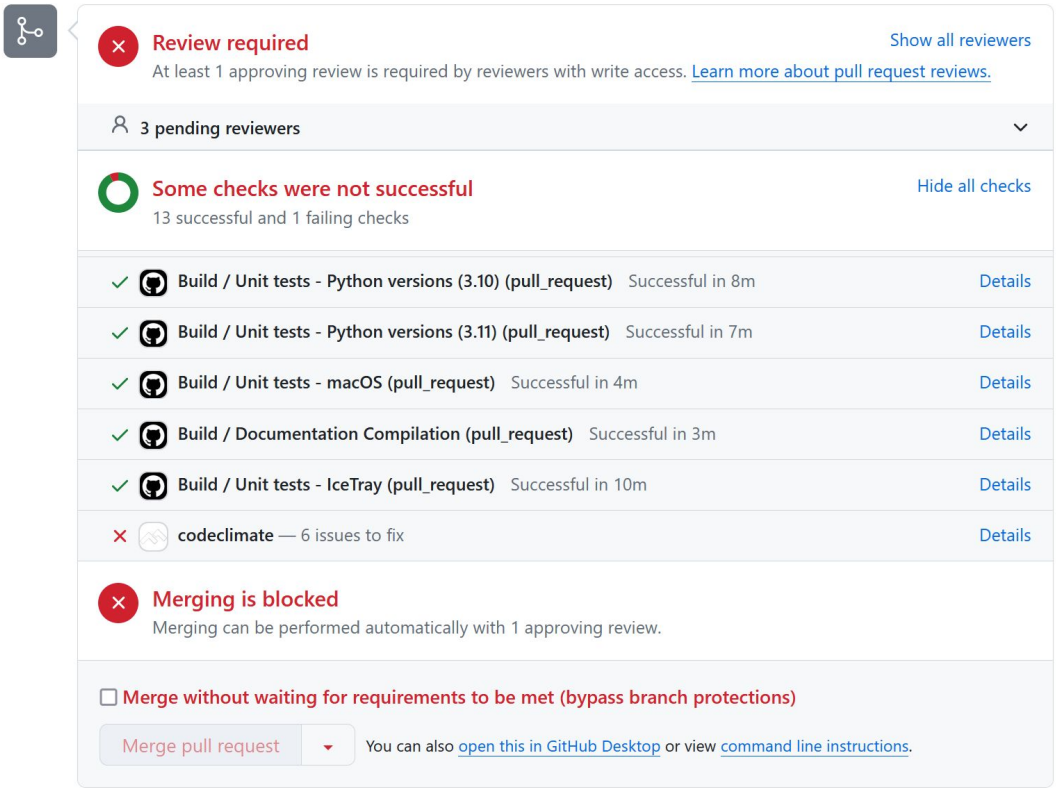
- akkarimi / graphnet
- AMHermansen / graphnet
- AnandMK-cloud / graphnet
- Andreas-MJ / graphnet
- ArturoLlorente / graphnet
- Aske-Rosted / graphnet
- asogaard / graphnet
  - gtc1030 / graphnet
- atamazian / graphnet
- axel-ponten / graphnet
- BozianuLeon / graphnet
- branchialspace / graphnet
- Camhock101 / graphnet
- ChenLi2049 / graphnet
- dghuman / graphnet
- edenni / graphnet
- edrakopo / graphnet
- FNoaGut / graphnet
- FredeVHansen / graphnet
- giogiopg / graphnet
- grasseau / graphnet
- grburgess / graphnet
- gurusarath1 / graphnet
- HiroakiHibi / graphnet

Contributors 20



+ 6 contributors

Code Quality



Contributing to GraphNeT

Contributing To GraphNeT

To make sure that the process of contributing is as smooth and effective as possible, we provide a few guidelines in this contributing guide that we encourage contributors to follow.

GitHub issues

Use GitHub issues for tracking and discussing requests and bugs. If there is anything you'd wish to contribute, the best place to start is to create a new issues and describe what you would like to work on. Alternatively you can assign open issues to yourself, to indicate that you would like to take ownership of a particular task. Using issues actively in this way ensures transparency and agreement on priorities. This helps avoid situations with a lot of development effort going into a feature that e.g. turns out to be outside of scope for the project; or a specific solution to a problem that could have been better solved differently.

Pull requests

Develop code in a fork of the main repo. Make contributions in dedicated development/feature branches on your forked repositories, e.g. if you are implementing a specific `GraphDefinition` class you could create a branch named `add-euclidean-graph-definition` on your own fork.

Create pull requests from your development branch into `graphnet-team/graphnet:main` to contribute to the project. **To be accepted**, pull requests must: