How to use GraphNeT in KM3NeT

“3rd Meeting on Common Neutrino Dataformats”
30th August 2024

Jorge Prado González
Instituto de Física Corpuscular, Valencia, España.
jorge.prado@ific.uv.es
08/05/2024

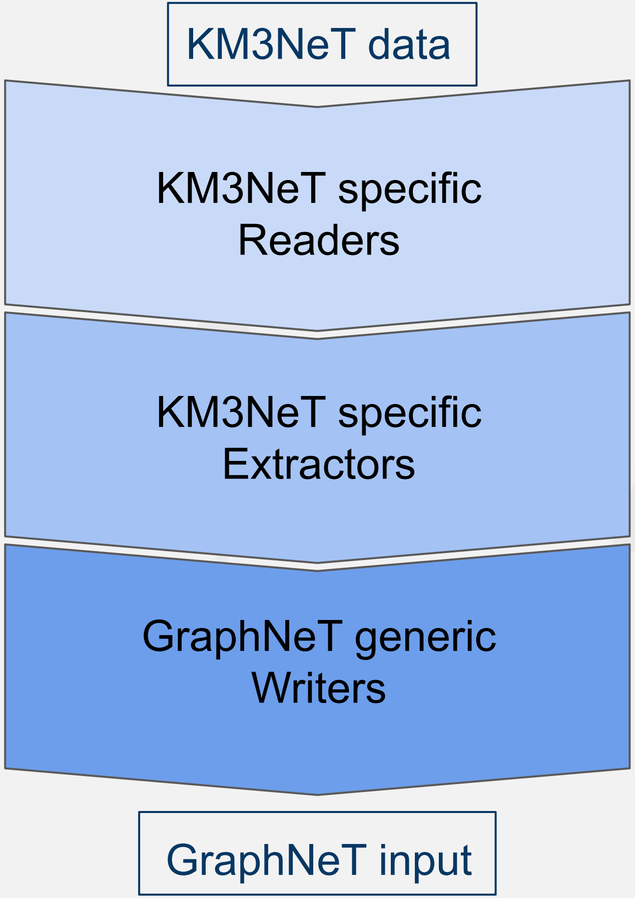
Ice-breaking slide

- Follow up of the presentation given by Rasmus in this same call.

- Presenting the work some members of KM3NeT have been making the last months to **include KM3NeT detector into GraphNeT**.

Iván Mozun, **Max Eff**, **Lukas Hennig** and **Jorge Prado** (all KM3NeT members) in the **4th GraphNeT Workshop** that took place in Munich.

Convert KM3NeT data



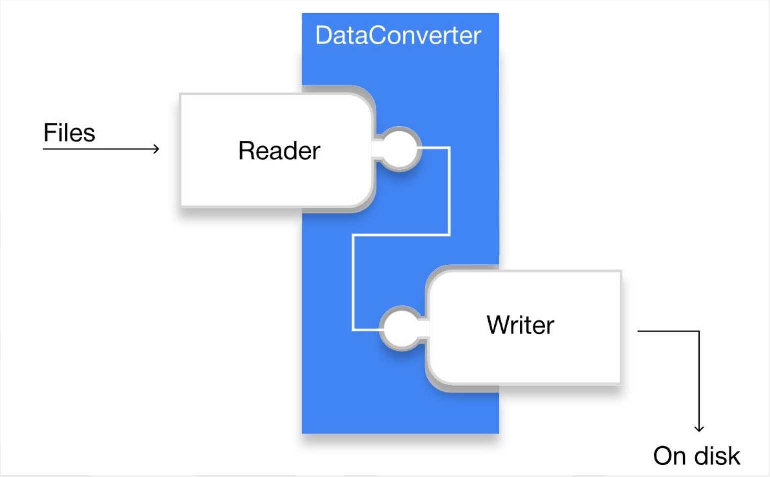
KM3NeT data

KM3NeT specific Readers

KM3NeT specific Extractors

GraphNeT generic Writers

GraphNeT input



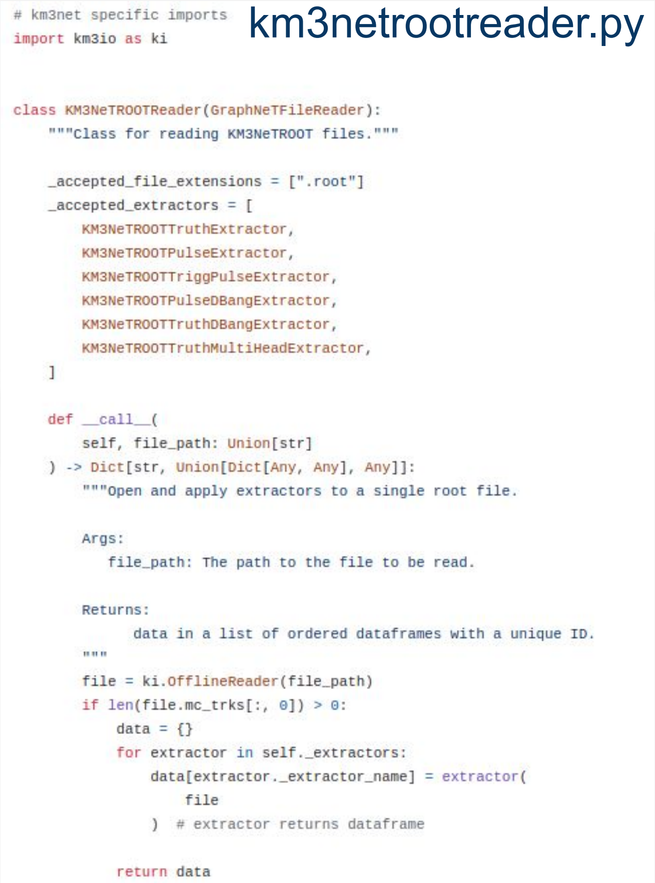
GraphNeT documentation

Check it for more general but technical information about how to include your experiment in GraphNeT.

KM3NeT Reader

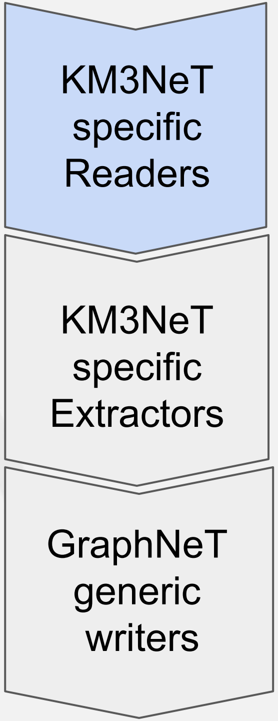
Class **KM3NeTROOTReader** that:

- Gathers all the accepted extension of the files we want to read and all the accepted extractors.
- Opens the .root file with KM3NeT data using **km3io** (can be installed with using pip).
- Applies the selected extractor to the file getting in **return a dictionary of dataframes** (truth variables, pulses etc…)



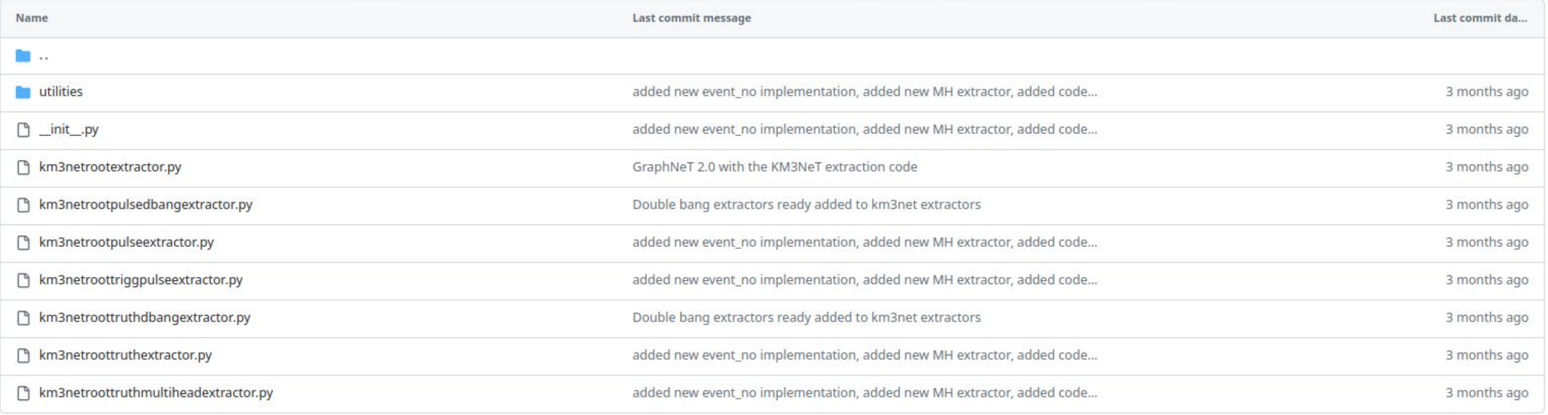
km3netrootreader.py

```python
# km3net specific imports
import km3io as ki


class KM3NeTROOTReader(GraphNeTFileReader):
    """Class for reading KM3NeTROOT files."""

    _accepted_file_extensions = [".root"]
    _accepted_extractors = [
        KM3NeTROOTTruthExtractor,
        KM3NeTROOTPulseExtractor,
        KM3NeTROOTTriggPulseExtractor,
        KM3NeTROOTPulseDBangExtractor,
        KM3NeTROOTTruthDBangExtractor,
        KM3NeTROOTTruthMultiHeadExtractor,
    ]

    def __call__(
        self, file_path: Union[str]
    ) -> Dict[str, Union[Dict[Any, Any], Any]]:
        """Open and apply extractors to a single root file.

        Args:
           file_path: The path to the file to be read.

        Returns:
              data in a list of ordered dataframes with a unique ID.
        """
        file = ki.OfflineReader(file_path)
        if len(file.mc_trks[:, 0]) > 0:
            data = {}
            for extractor in self._extractors:
                data[extractor._extractor_name] = extractor(
                    file
                )  # extractor returns dataframe

            return data
```

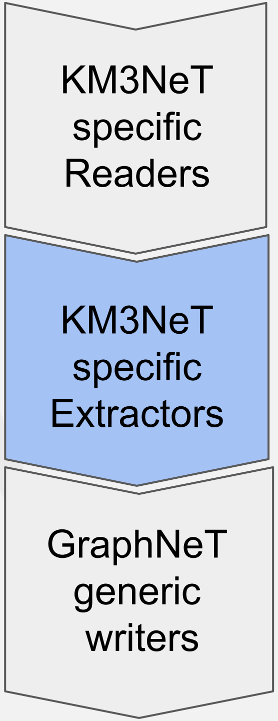


KM3NeT Extractors

- **Extractors** are called by the reader and must **return a dataframe** with the desired information.
- **Different extractors for different data formats**: ie. productions v9.0 and v7.2 have different header structure with different labels -> Using *KM3NeTROOTTruthMultiHeadExtractor* class instead of *KM3NeTROOTTruthExtractor*.
- Also this **extractors create a unique id** to identify an event and tag all hits coming from that event.



Name	Last commit message	Last commit da...
..		
utilities	added new event_no implementation, added new MH extractor, added code...	3 months ago
__init__.py	added new event_no implementation, added new MH extractor, added code...	3 months ago
km3netrootextractor.py	GraphNeT 2.0 with the KM3NeT extraction code	3 months ago
km3netrootpulsedbangextractor.py	Double bang extractors ready added to km3net extractors	3 months ago
km3netrootpulseextractor.py	added new event_no implementation, added new MH extractor, added code...	3 months ago
km3netroottriggpulseextractor.py	added new event_no implementation, added new MH extractor, added code...	3 months ago
km3netroottruthdbangextractor.py	Double bang extractors ready added to km3net extractors	3 months ago
km3netroottruthextractor.py	added new event_no implementation, added new MH extractor, added code...	3 months ago
km3netroottruthmultiheadextractor.py	added new event_no implementation, added new MH extractor, added code...	3 months ago



KM3NeT specific Readers

KM3NeT specific Extractors

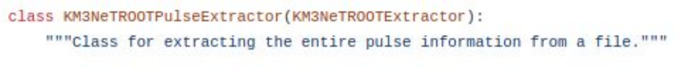
GraphNeT generic writers

KM3NeT Extractors

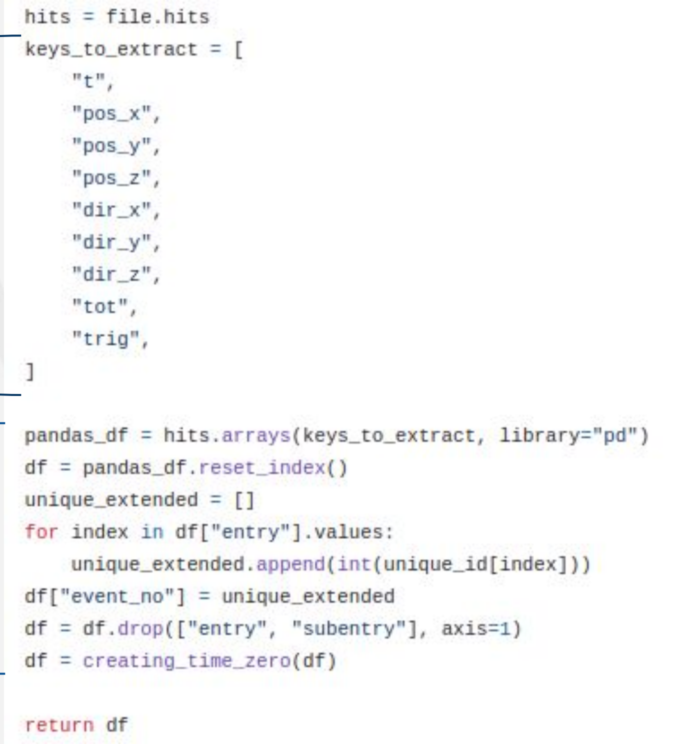
Part of the code extracting relevant features to define hits in KM3NeT such as the time and ToT of the hit or the orientation and position of the PMT recording the hit.

Create a dataframe with those features (or slightly modified features) and add a unique id (“event_no”).

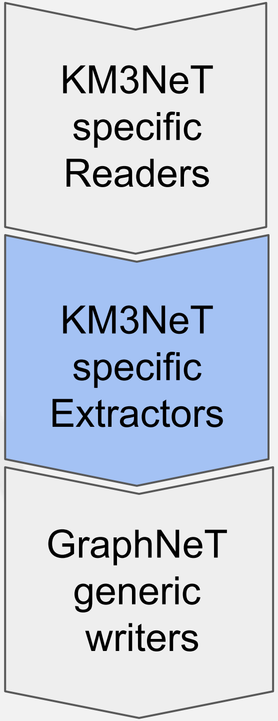
Dataframe containing all the pulse information of all the events in the file opened in the reader.



```
class KM3NeTROOTPulseExtractor(KM3NeTROOTExtractor):
    """Class for extracting the entire pulse information from a file."""
```



```
hits = file.hits
keys_to_extract = [
    "t",
    "pos_x",
    "pos_y",
    "pos_z",
    "dir_x",
    "dir_y",
    "dir_z",
    "tot",
    "trig",
]

pandas_df = hits.arrays(keys_to_extract, library="pd")
df = pandas_df.reset_index()
unique_extended = []
for index in df["entry"].values:
    unique_extended.append(int(unique_id[index]))
df["event_no"] = unique_extended
df = df.drop(["entry", "subentry"], axis=1)
df = creating_time_zero(df)

return df
```

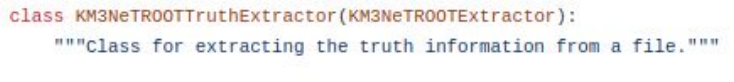


KM3NeT specific Readers

KM3NeT specific Extractors

GraphNeT generic writers

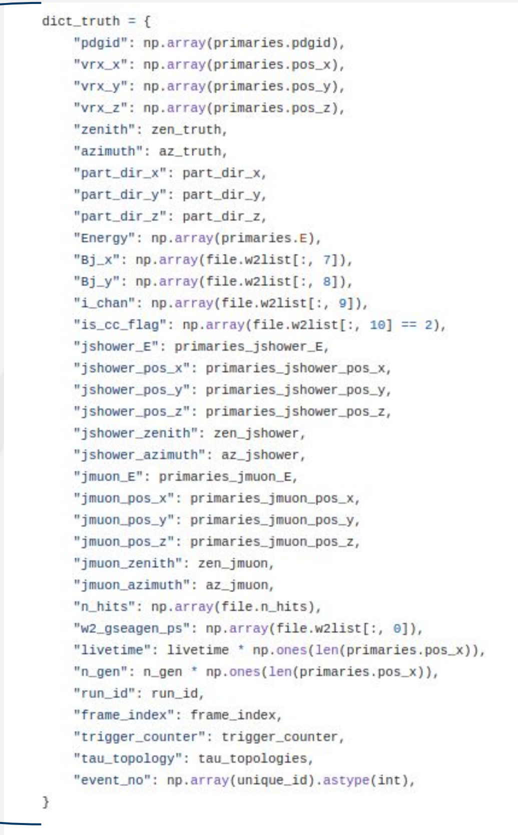
KM3NeT Extractors



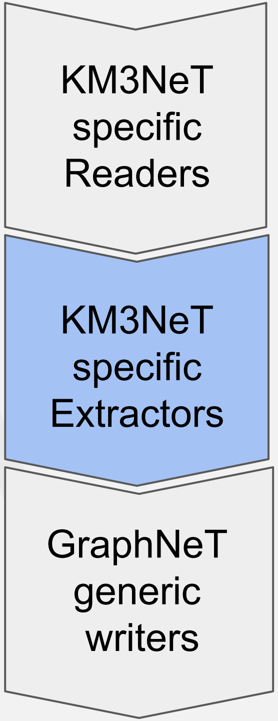
```
class KM3NeTROOTTruthExtractor(KM3NeTROOTExtractor):
    """Class for extracting the truth information from a file."""
```

Same working principle!

True features of the event extracted to be stored in the database.



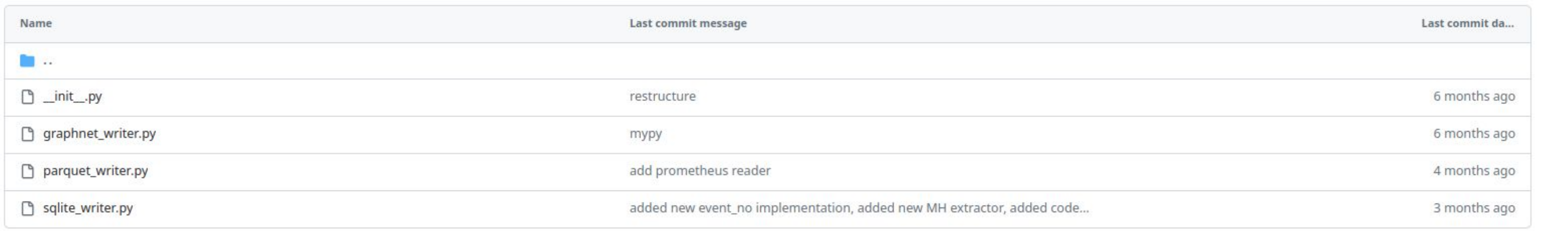
```
dict_truth = {
    "pdgid": np.array(primaries.pdgid),
    "vrx_x": np.array(primaries.pos_x),
    "vrx_y": np.array(primaries.pos_y),
    "vrx_z": np.array(primaries.pos_z),
    "zenith": zen_truth,
    "azimuth": az_truth,
    "part_dir_x": part_dir_x,
    "part_dir_y": part_dir_y,
    "part_dir_z": part_dir_z,
    "Energy": np.array(primaries.E),
    "Bj_x": np.array(file.w2list[:, 7]),
    "Bj_y": np.array(file.w2list[:, 8]),
    "i_chan": np.array(file.w2list[:, 9]),
    "is_cc_flag": np.array(file.w2list[:, 10] == 2),
    "jshower_E": primaries_jshower_E,
    "jshower_pos_x": primaries_jshower_pos_x,
    "jshower_pos_y": primaries_jshower_pos_y,
    "jshower_pos_z": primaries_jshower_pos_z,
    "jshower_zenith": zen_jshower,
    "jshower_azimuth": az_jshower,
    "jmuon_E": primaries_jmuon_E,
    "jmuon_pos_x": primaries_jmuon_pos_x,
    "jmuon_pos_y": primaries_jmuon_pos_y,
    "jmuon_pos_z": primaries_jmuon_pos_z,
    "jmuon_zenith": zen_jmuon,
    "jmuon_azimuth": az_jmuon,
    "n_hits": np.array(file.n_hits),
    "w2_gseagen_ps": np.array(file.w2list[:, 0]),
    "livetime": livetime * np.ones(len(primaries.pos_x)),
    "n_gen": n_gen * np.ones(len(primaries.pos_x)),
    "run_id": run_id,
    "frame_index": frame_index,
    "trigger_counter": trigger_counter,
    "tau_topology": tau_topologies,
    "event_no": np.array(unique_id).astype(int),
}
```



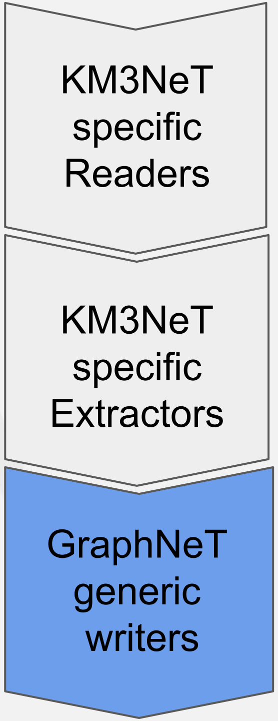
GraphNeT generic writers

- With GraphNeT already built in writers one can write into **sqlite** or **parquet** databases the information extracted with the KM3NeT readers and extractors.

- See Rasmus talk for advantages and disadvantages of these formats.



Name	Last commit message	Last commit da...
..		
__init__.py	restructure	6 months ago
graphnet_writer.py	mypy	6 months ago
parquet_writer.py	add prometheus reader	4 months ago
sqlite_writer.py	added new event_no implementation, added new MH extractor, added code...	3 months ago



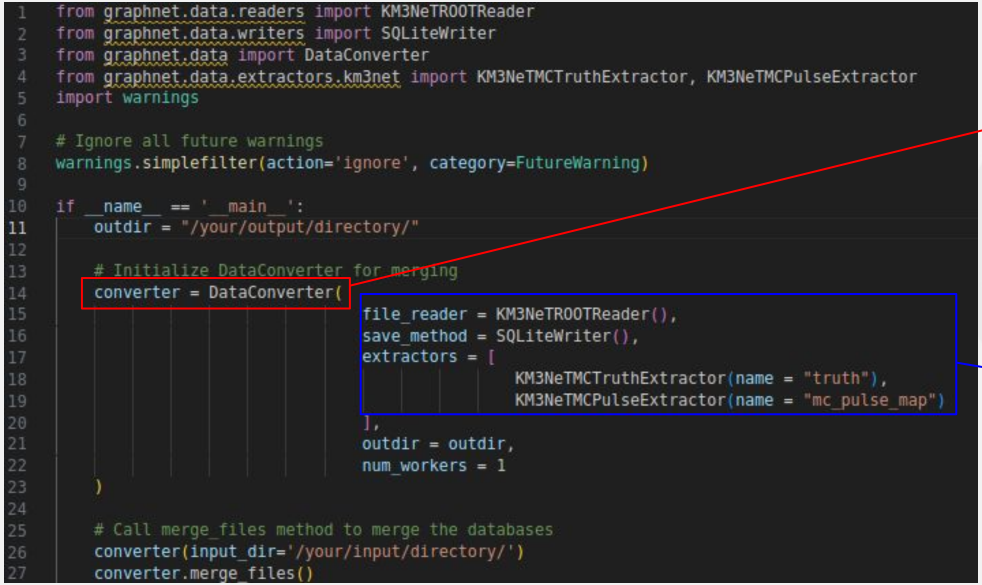
KM3NeT specific Readers

KM3NeT specific Extractors

GraphNeT generic writers

Processing a file

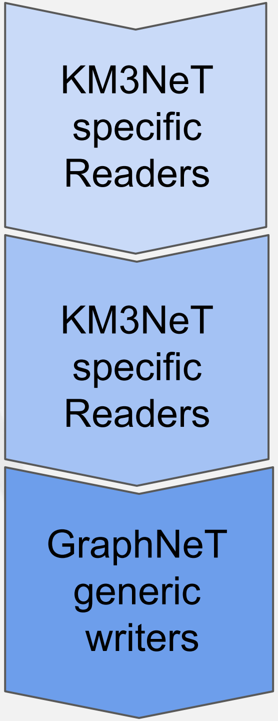
Only a few lines of code are needed!



```
from graphnet.data.readers import KM3NeTROOTReader
from graphnet.data.writers import SQLiteWriter
from graphnet.data import DataConverter
from graphnet.data.extractors.km3net import KM3NeTMCTruthExtractor, KM3NeTMCPulseExtractor
import warnings

# Ignore all future warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

if __name__ == '__main__':
    outdir = "/your/output/directory/"

    # Initialize DataConverter for merging
    converter = DataConverter(
                                file_reader = KM3NeTROOTReader(),
                                save_method = SQLiteWriter(),
                                extractors = [
                                                KM3NeTMCTruthExtractor(name = "truth"),
                                                KM3NeTMCPulseExtractor(name = "mc_pulse_map")
                                ],
                                outdir = outdir,
                                num_workers = 1
    )

    # Call merge_files method to merge the databases
    converter(input_dir='/your/input/directory/')
    converter.merge_files()
```

Call and start the **DataConverter** class

Provide it with your choice of classes for reading, extracting and writing down your data into a database

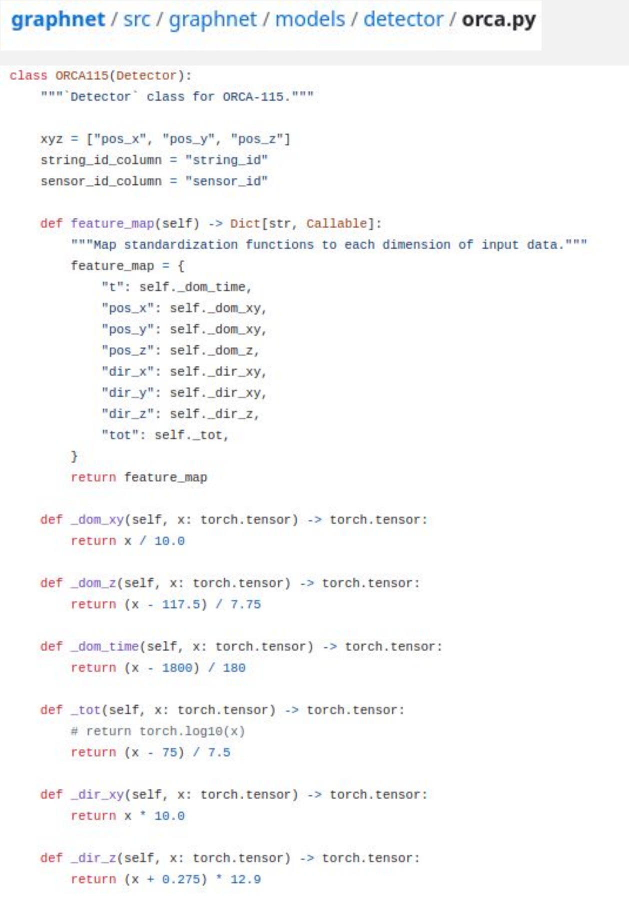


KM3NeT specific Readers

KM3NeT specific Readers

GraphNeT generic writers

Including your detector in GraphNeT



graphnet / src / graphnet / models / detector / **orca.py**

```
class ORCA115(Detector):
    """`Detector` class for ORCA-115."""

    xyz = ["pos_x", "pos_y", "pos_z"]
    string_id_column = "string_id"
    sensor_id_column = "sensor_id"

    def feature_map(self) -> Dict[str, Callable]:
        """Map standardization functions to each dimension of input data."""
        feature_map = {
            "t": self._dom_time,
            "pos_x": self._dom_xy,
            "pos_y": self._dom_xy,
            "pos_z": self._dom_z,
            "dir_x": self._dir_xy,
            "dir_y": self._dir_xy,
            "dir_z": self._dir_z,
            "tot": self._tot,
        }
        return feature_map

    def _dom_xy(self, x: torch.tensor) -> torch.tensor:
        return x / 10.0

    def _dom_z(self, x: torch.tensor) -> torch.tensor:
        return (x - 117.5) / 7.75

    def _dom_time(self, x: torch.tensor) -> torch.tensor:
        return (x - 1800) / 180

    def _tot(self, x: torch.tensor) -> torch.tensor:
        # return torch.log10(x)
        return (x - 75) / 7.5

    def _dir_xy(self, x: torch.tensor) -> torch.tensor:
        return x * 10.0

    def _dir_z(self, x: torch.tensor) -> torch.tensor:
        return (x + 0.275) * 12.9
```

- Before training a model on KM3NeT data it is still needed to create your **detector class**.

- Provide information about the geometry of the detector (not strictly needed, only for some functionalities).

- Provide a way to **normalize the features** of the pulse map that is going to be used for the training.

Status

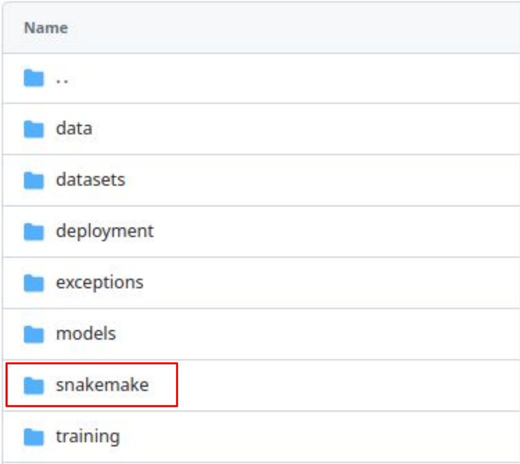
- So far this changes live in a **private repo** in KM3NeT gitlab.

- **Waiting for internal approval** to make a pull request into GraphNeT public software.

- Also other “smaller versions” in which any reconstructed true variables with internal software are extracted and no geometry of the detector is provided.

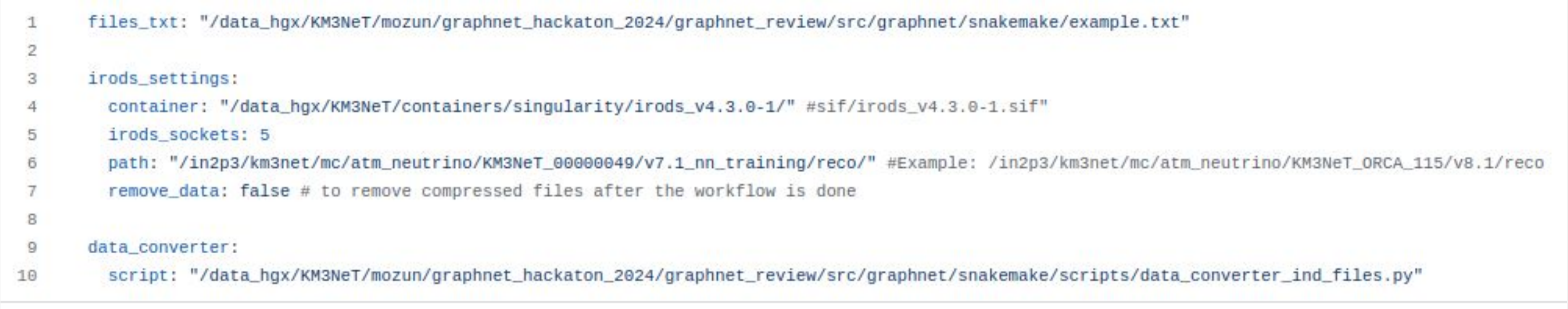
- **Software fully functional**. Already performed some internal trainings and achieved preliminary results presented in Neutrino Physics and Machine Learning 2024 conference.

Other utilities - Snakemake and GraphNeT



Name
..
data
datasets
deployment
exceptions
models
snakemake
training

- **Snakemake pipeline** to fetch files from IRODS and convert them into sqlite/parquet databases **fully functional**.
- Possibility of **parallelization** so it is quite fast to process an entire mass production.

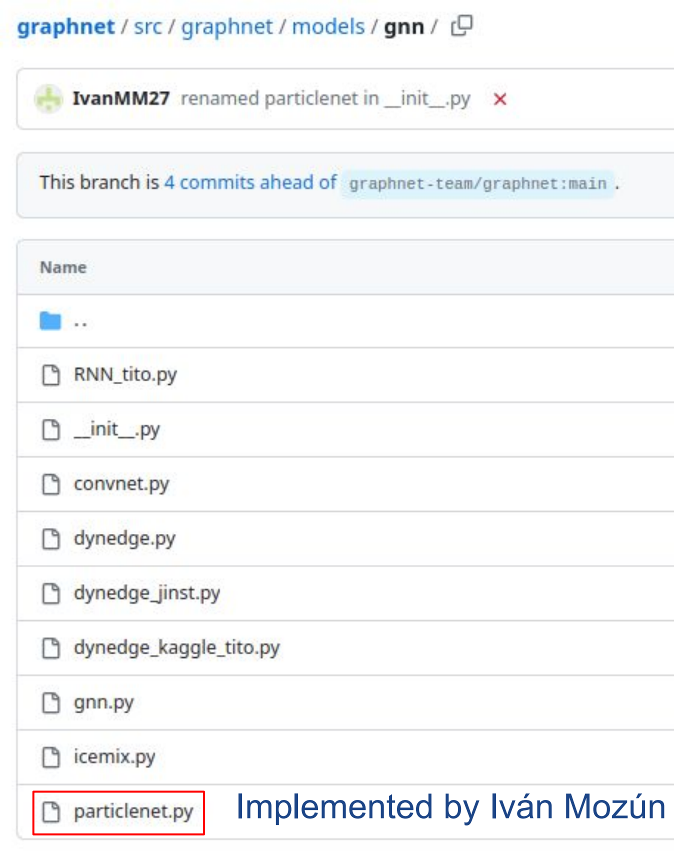


```yaml
files_txt: "/data_hgx/KM3NeT/mozun/graphnet_hackaton_2024/graphnet_review/src/graphnet/snakemake/example.txt"

irods_settings:
  container: "/data_hgx/KM3NeT/containers/singularity/irods_v4.3.0-1/" #sif/irods_v4.3.0-1.sif"
  irods_sockets: 5
  path: "/in2p3/km3net/mc/atm_neutrino/KM3NeT_00000049/v7.1_nn_training/reco/" #Example: /in2p3/km3net/mc/atm_neutrino/KM3NeT_ORCA_115/v8.1/reco
  remove_data: false # to remove compressed files after the workflow is done

data_converter:
  script: "/data_hgx/KM3NeT/mozun/graphnet_hackaton_2024/graphnet_review/src/graphnet/snakemake/scripts/data_converter_ind_files.py"
```

Implemented by Iván Mozún

ORCANeT in GraphNeT

- **ORCANeT model already implemented!**
- Easy to train ORCANeT models on KM3NeT data before processed with ORCASong framework now in GraphNeT.
- This pull request has already been made to the GraphNeT team and waiting for approval to be public.



graphnet / src / graphnet / models / **gnn** /

IvanMM27 renamed particlenet in __init__.py ✗

This branch is 4 commits ahead of `graphnet-team/graphnet:main`.

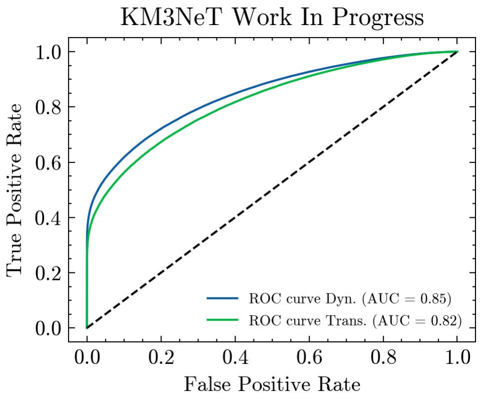
Name
..
RNN_tito.py
__init__.py
convnet.py
dynedge.py
dynedge_jinst.py
dynedge_kaggle_tito.py
gnn.py
icemix.py
particlenet.py

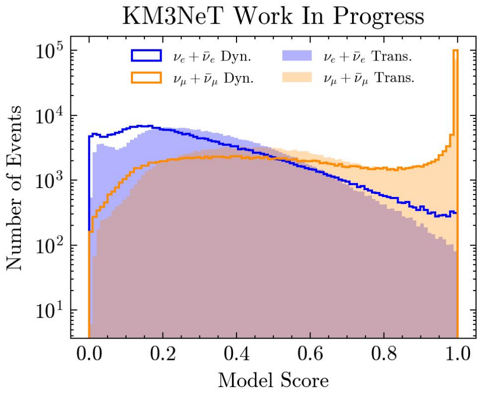
Implemented by Iván Mozún

(*) The model ORCANeT is just a version of the existing a public model ParticleNet

Final remarks and perspectives

- Using GraphNeT provides a way to **easily compare our models** having used the **same training sample** with the exact same processing and **testing on the same events**.
- Using GraphNeT ensures that there is a **well maintained software** frame that will be functional and kept up to date.
- There are several **open projects** involving GraphNeT and KM3NeT:
  - Comparison of models (Dynedge, ORCANeT, transformers…)
  - Transfer learning with transformers
  - GNN-based pulse cleaning
  - Reconstruction of exotic-particle-driven signals
- Easy and fast to perform inference. (time of 3.1 ms per event for dynedge model in a trained track-shower classifier using 1 GPU).





Thank you!