AI GOES MAD²

SMALL THINKS BIG
Transfer learning in KM3NeT/ORCA with transformers

15/10/2024, Caen

Iván Mozún Mateo
On behalf of the KM3NeT collaboration

KM3NeT

lpc caen

cnrs

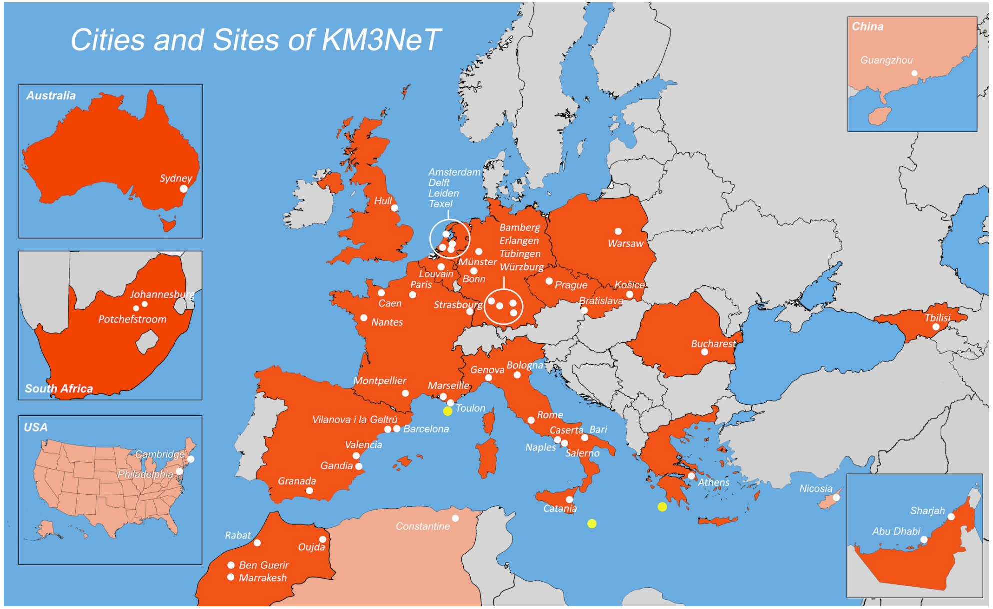
RÉGION NORMANDIE

Overview

1. KM3NeT: neutrino telescopes
2. Need for data in large language models
3. Why transfer learning?
4. Multi-detector configuration and multi-task for KM3NeT/ORCA
5. Summary & The road ahead

KM3NeT

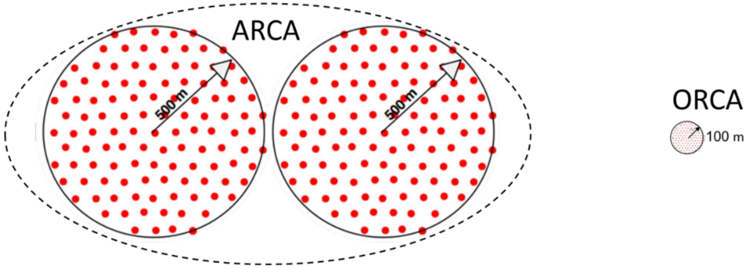
KM3NeT is an **international collaboration**
- 22 countries
- 65 partner institutes
- ~250 members



Two **undersea neutrino telescopes**

- **KM3NeT/ARCA**
  - Optimized for 1 TeV – 10 PeV
  - Identify high-energy neutrino sources in the Universe.
  - 36m vertical spacing and 90m horizontal spacing
- **KM3NeT/ORCA**
  - Optimized for 1 – 100 GeV
  - Determine the mass ordering of neutrinos.
  - 9m vertical spacing and 20m horizontal spacing

Currently under construction: ORCA23 (20%), ARCA28 (12%)



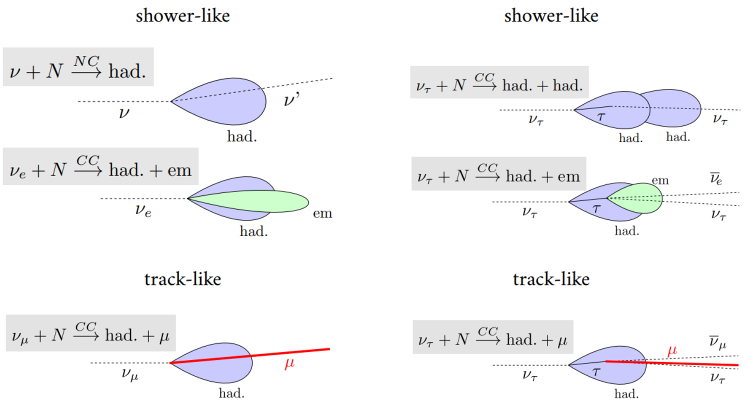
KM3NeT: neutrino telescopes

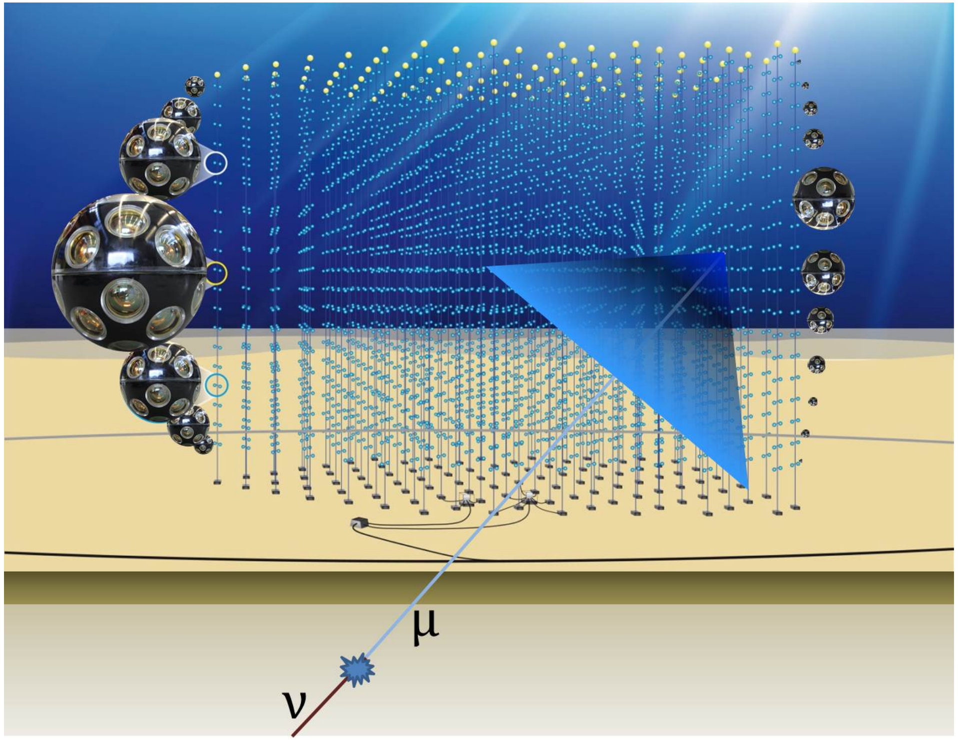
Same **technology**:

- 1 (2) building block(s) for ORCA (ARCA)
- 115 vertical detection units (DUs) per block
- 18 digital optical modules (DOMs) per DU
- 31” PMTs per DOM

Same **detection principle**:

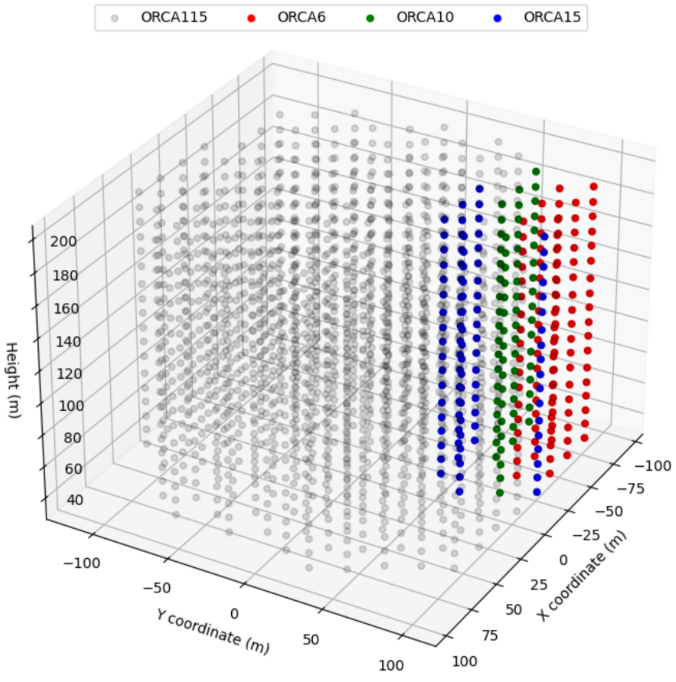
Light collection from ***Cherenkov radiation*** *emitted by particles traveling faster than the speed of light in water*

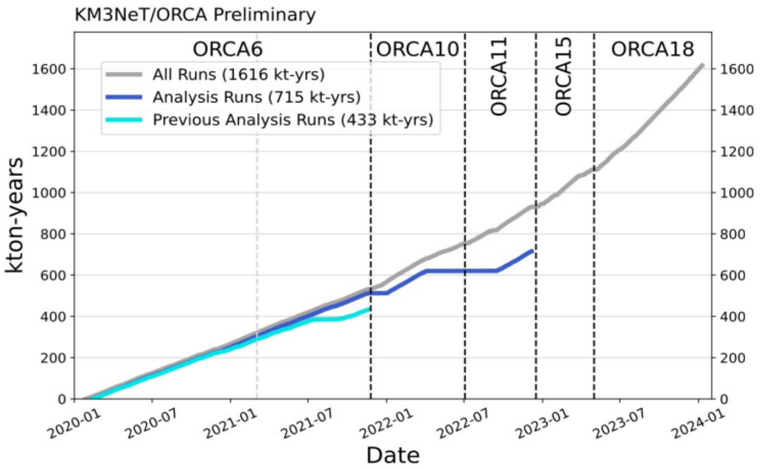




Building the detectors

KM3NeT telescopes collect, process and analyze data as they are being built.



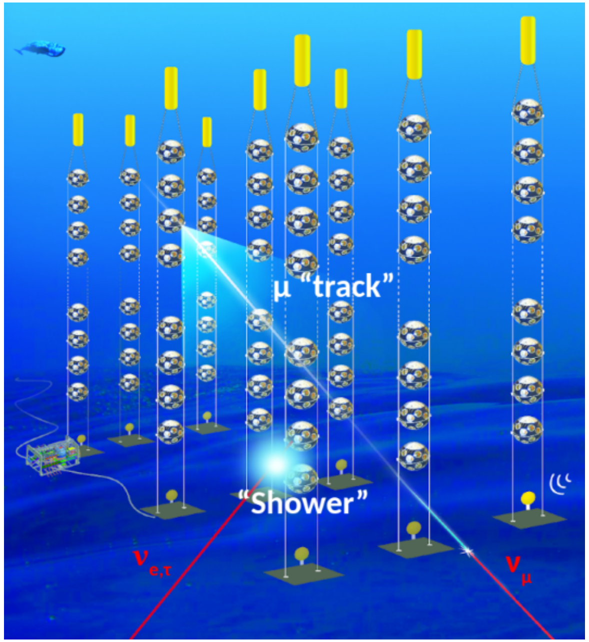


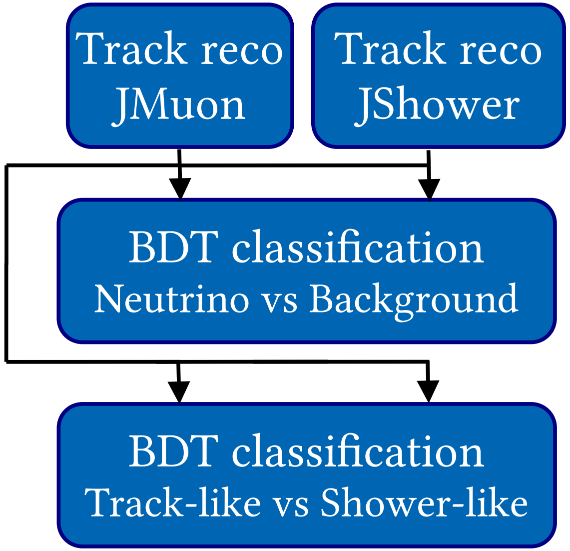
Foundation model in KM3NeT?

- learns as the detectors grow
- can handle multiple geometries
- can handle both KM3NeT/ORCA and KM3NeT/ARCA
- classification ↔ reconstruction

Reconstructing neutrino physics

The **official KM3NeT pipeline for reconstruction and classification** relies on algorithms that are applied separately for track-like event or shower-like events. Then, simple BDTs are applied on the reconstructed variables for classification tasks.

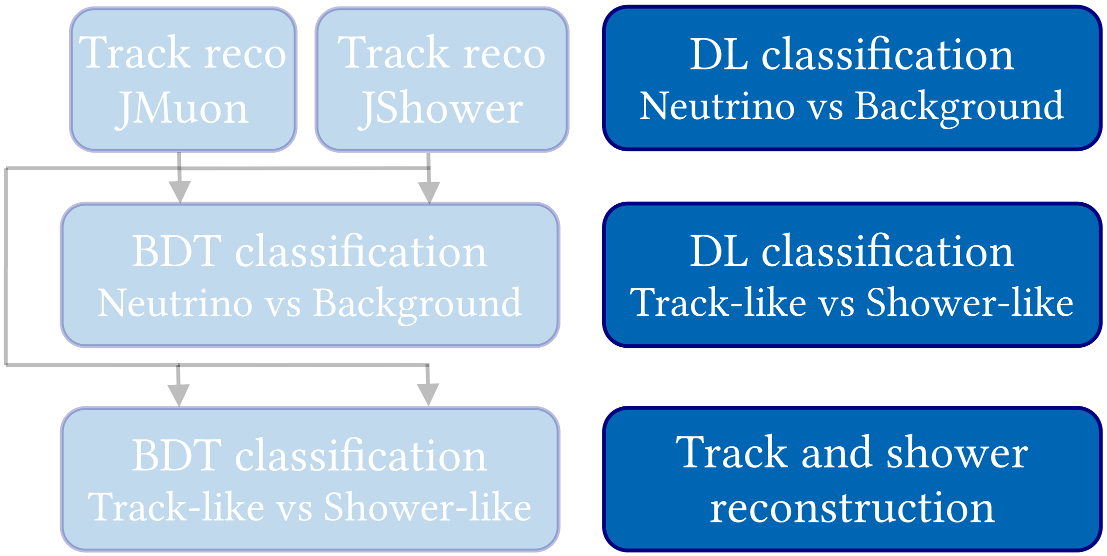




Classic approach

Reconstructing neutrino physics

The **official KM3NeT pipeline for reconstruction and classification** relies on algorithms that are applied separately for track-like event or shower-like events. Then, simple BDTs are applied on the reconstructed variables for classification tasks.



Classic approach

Deep Learning representation

Novel **deep learning** techniques use low-level information from the detector, i.e. light pulses to

1. Let the model decide the features to use
2. Generalise over a large input domain dimensions
3. Perform different tasks

Classification and reconstruction are performed **independently**, and for any type of event.

Large DL models needs **huge amounts of very diverse data** to generalize and interpolate, improving the performances of existing algorithms.

KM3NeT Deep Learning Outreach

Various DL models tested. So far, no one is considered for official analysis.

Convolutional Neural Networks

- Event reconstruction for KM3NeT/ORCA using convolutional neural networks (M. Moser, KM3NeT)
- Event Classification and Energy Reconstruction for ANTARES using Convolutional Neural Networks (N. Geißelbrecht, ANTARES)
- Deep learning reconstruction in ANTARES (J. García-Méndez et al., ANTARES)
- Dark matter search towards the Sun using Machine Learning reconstructions of single-line events in ANTARES (J. García-Méndez et al., ANTARES)

Deep Neural Networks

- Deep Neural Networks for combined neutrino energy estimate with KM3NeT/ORCA6 (S. Peña Martínez, KM3NeT)

Graph Neural Networks:

- Development of detector calibration and graph neural network-based selection and reconstruction algorithms for the measurement of oscillation parameters with KM3NeT/ORCA (D. Guderian, KM3NeT)
- Data reconstruction and classification with graph neural networks in KM3NeT/ARCA6-8 (F. Filippini et al., KM3NeT)
- Cosmic ray composition measurement using Graph Neural Networks for KM3NeT/ORCA (S. Reck, KM3NeT)
- Optimisation of energy regression with sample weights for GNNs in KM3NeT/ORCA (B. Setter, KM3NeT)
- Tau neutrino identification with Graph Neural Networks in KM3NeT/ORCA (L. Hennig, KM3NeT)

More details here: A Comprehensive Insight into Machine Learning Techniques in KM3NeT (J. Prado)

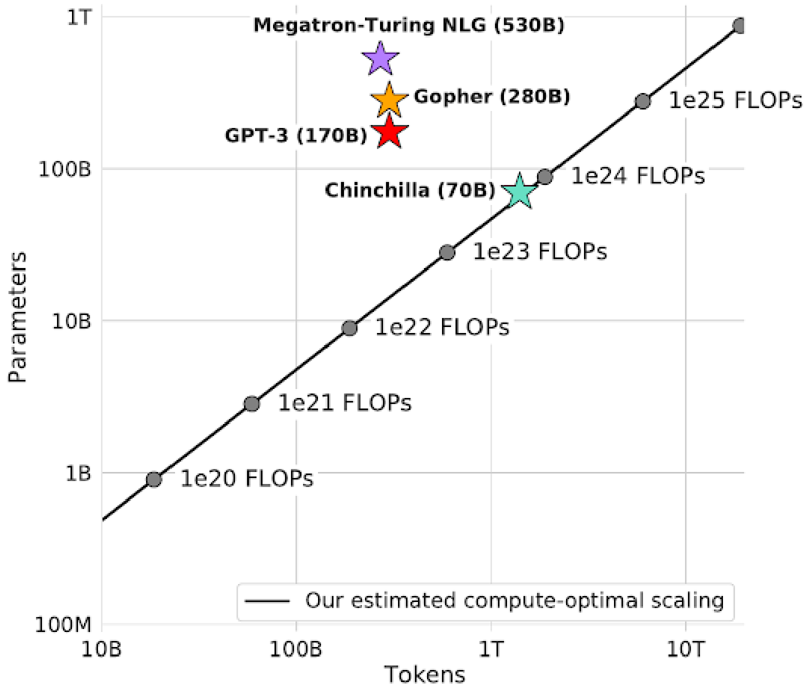
Need for data in large language models

Neutrino telescope data is described as a set of spatial points with timing & charge information (point-cloud data), hence, most developed DL architectures are based on GNNs.

Language models are starting to overtake but…

- **lot of trainable parameters**
- **lot of training data**

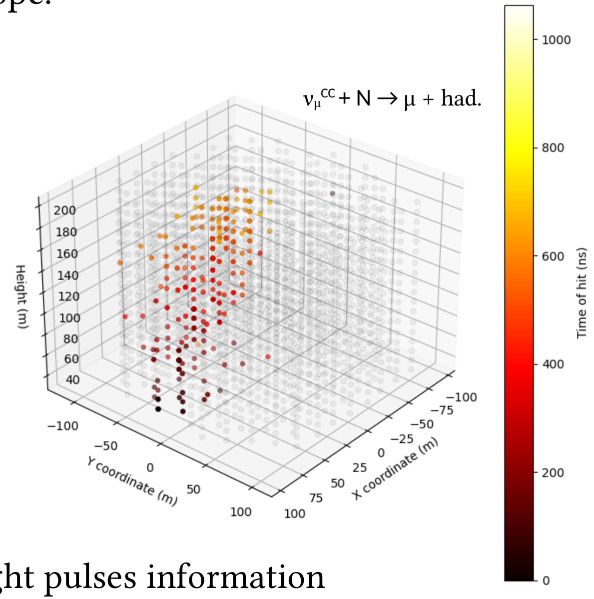
The data is too complex and requires a lot of computing resources to be produced and to encapsulate all the physics → **we must be efficient**



Scaling law for trainable parameters and tokens for large language models
arxiv.org/abs/2203.15556

Transformer architecture

The input data is the low-level hit information that composes the light pattern detected in the telescope.

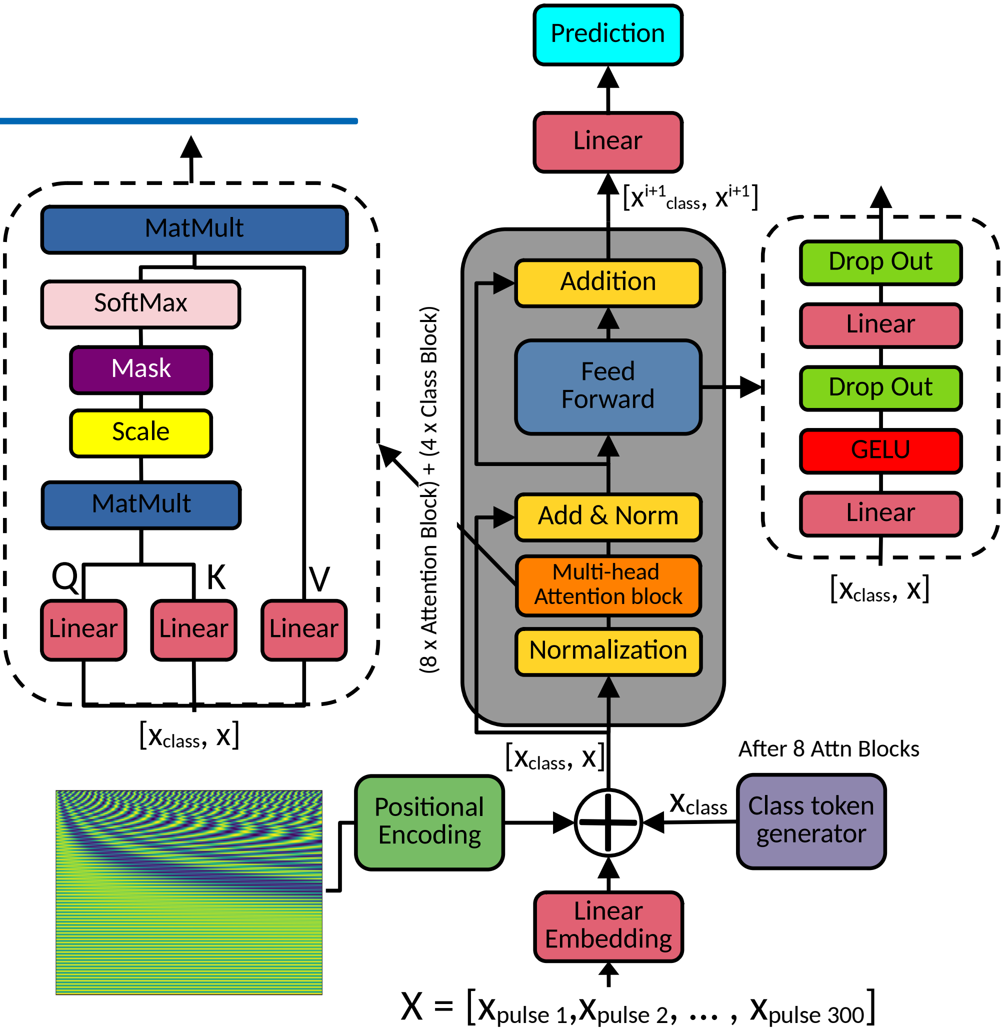


The light pulses information

$$X_{pulse} = [pos_x, pos_y, pos_z, dir_x, dir_y, dir_z, t, ToT]$$

is processed in parallel by the transformer and the high-level information is extracted in the attention blocks.

Model has ~1.6M trainable parameters.



Neutrinos in Deep Ice (Kaggle Competition): arxiv.org/abs/2310.15674
Particle Transformer for Jet Tagging: arxiv.org/abs/2202.03772

Transfer learning studies

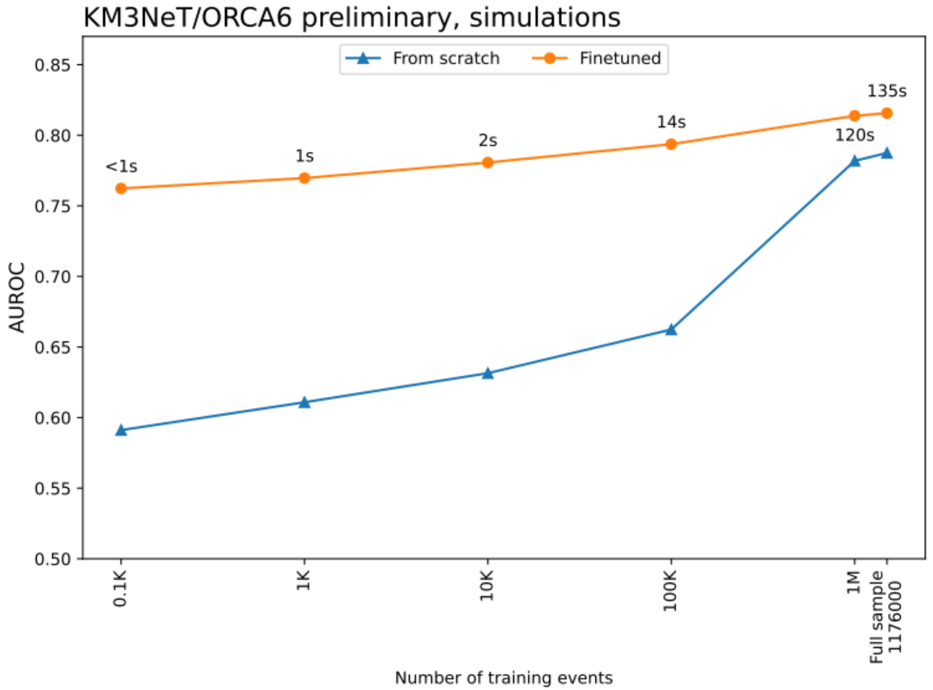
Multiple tasks with a single model
- Classification and reconstruction done together
- Test the capacity of the model

Efficient use of data
- Run-by-Run: simulates MC runs based on data runs to reduce discrepancies
- **Not enough data** to train large models for every time the detector response is updated

Missing detector information
- PMTs do not exist.
- PMTs correspond to DUs not deployed yet.

Efficient use of computing resources
- Saves time and increases performance
- The information is propagated across detectors

GraphNeT
Deep Learning for Neutrino Telescopes

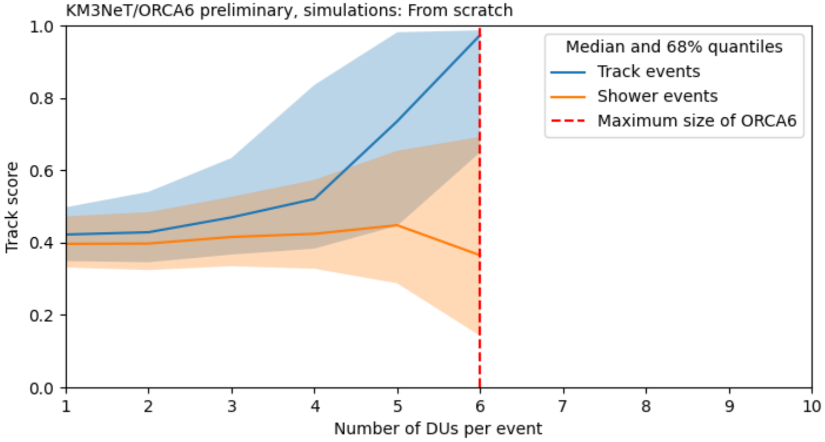


AUROC value for track-shower classification with KM3NeT/ORCA6 data.
The AUROC curves are shown as function of the training size sample.

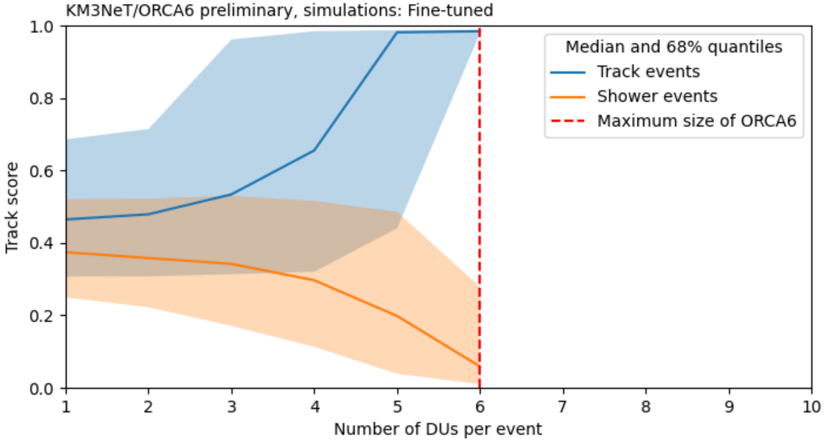
Preliminary results on multi-geometries for KM3NeT/ORCA6

The model is able to **interpolate** to **non-existing DUs information** because it pre-learned the full geometry.

From scratch



Fine-tuned



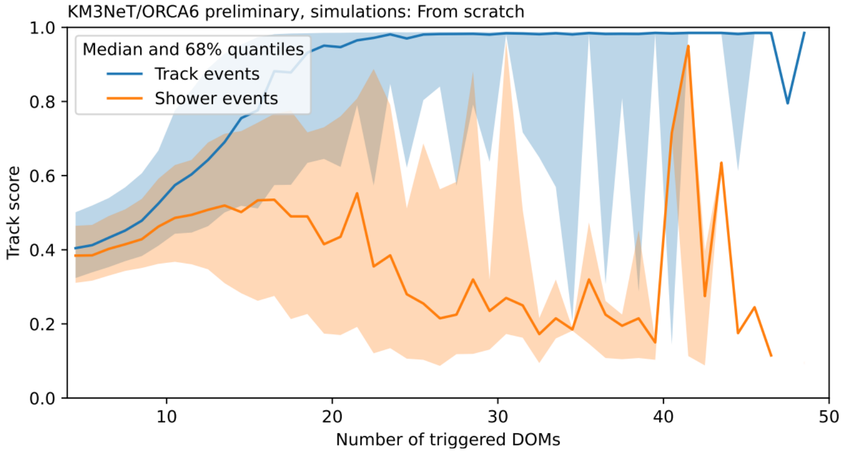
Track-shower classification:

- Limited data: 200k events for a detector with 6 lines is not enough to do a proper separation
- Performance: fine-tuned model works way better than the scratch one
- High dependence on event geometry: not enough discrimination with few lines

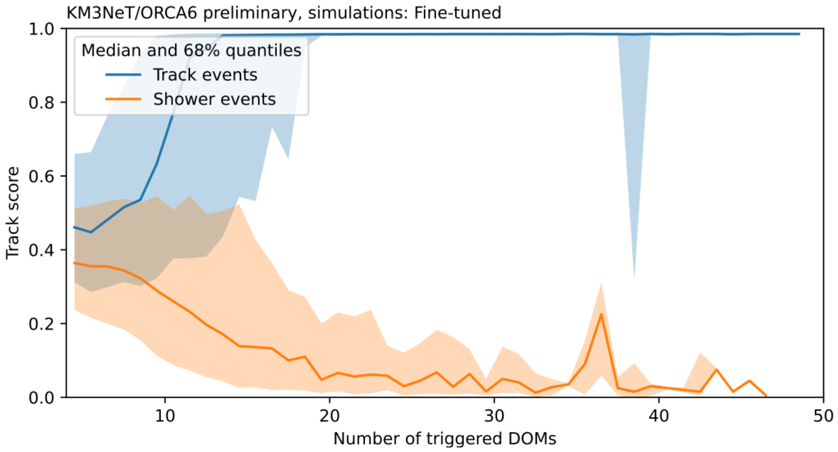
Preliminary results on multi-geometries for KM3NeT/ORCA6

The model is able to **interpolate** to **non-existing DUs information** because it pre-learned the full geometry.

From scratch



Fine-tuned



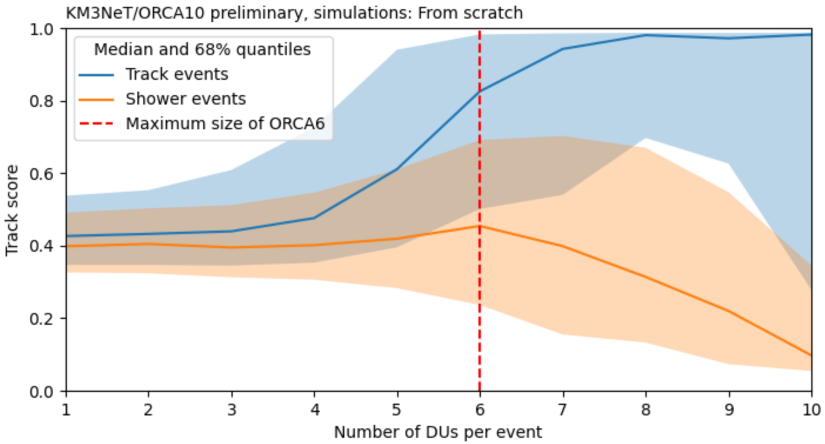
Track-shower classification:

- Limited data: 200k events for a detector with 6 lines is not enough to do a proper separation
- Performance: fine-tuned achieves separation in events with above 10 triggered DOMs
- Peak at ~40 triggered DOMs: fine-tuned model compensates the low statistics

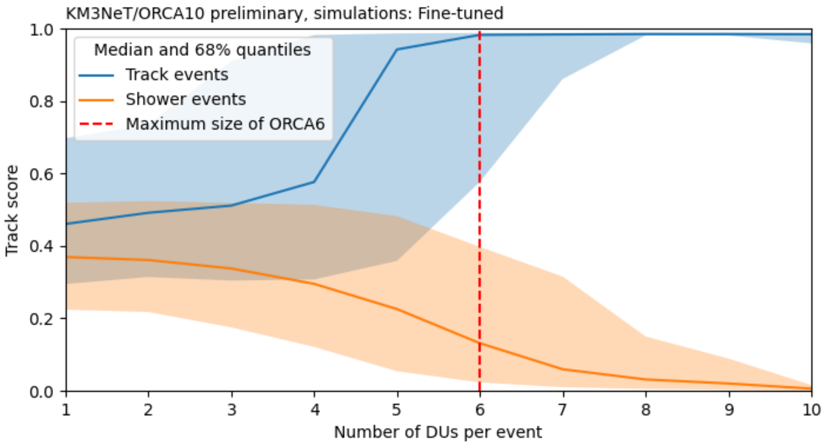
Preliminary results on multi-geometries for KM3NeT/ORCA10

The model is able to **interpolate** to **non-existing DUs information** because it pre-learned the full geometry.

From scratch



Fine-tuned

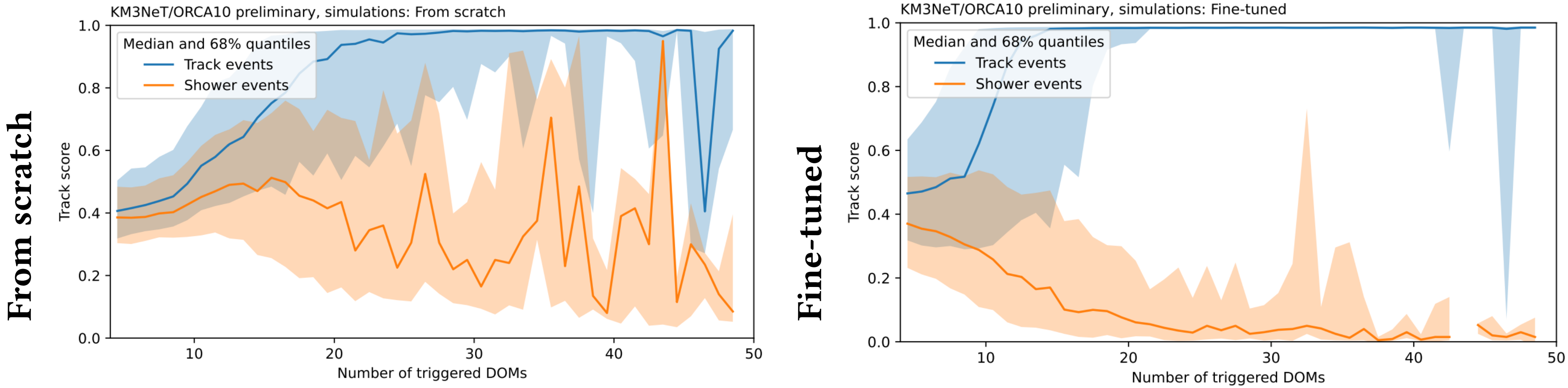


Track-shower classification:

- Limited data: 200k events for a detector with 6 lines is not enough to do a proper separation
- Performance: fine-tuned model works way better than the scratch one
- High dependence on event geometry: not enough discrimination with few lines
- **Major improvement with increasing detector size ↔ better event containment**

Preliminary results on multi-geometries for KM3NeT/ORCA10

The model is able to **interpolate** to **non-existing DUs information** because it pre-learned the full geometry.



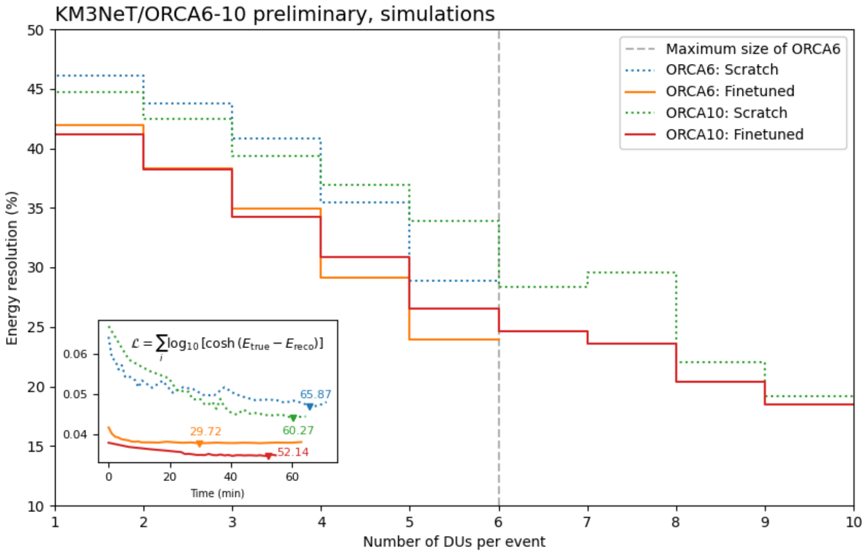
Track-shower classification:

- Limited data: 200k events for a detector with 6 lines is not enough to do a proper separation
- Performance: fine-tuned achieves separation in events with above 10 triggered DOMs
- Multiple peaks: fine-tuned model compensates the low statistics

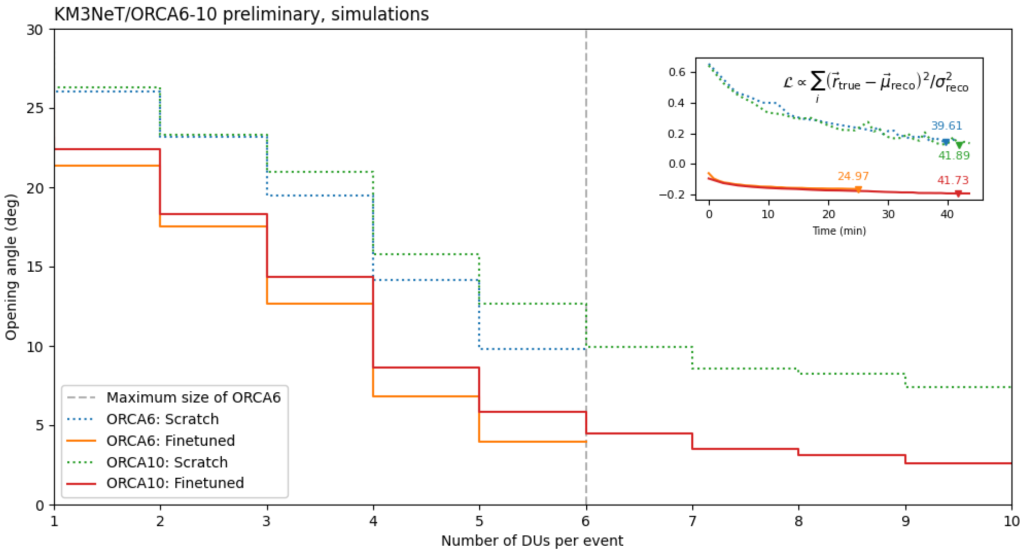
Preliminary results on multi-geometries for KM3NeT/ORCA6-10

The model is able to **interpolate** to **non-existing DUs information** because it pre-learned the full geometry.

Energy reconstruction



Direction reconstruction



Energy and direction reconstruction:

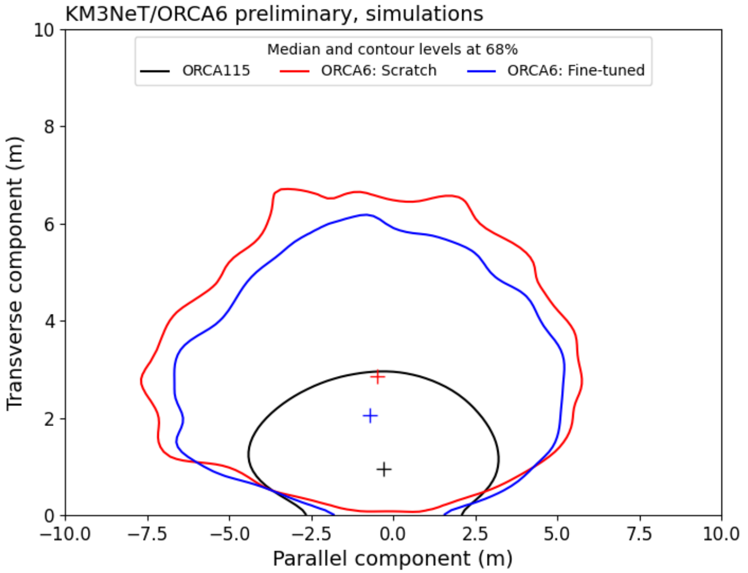
- Loss curves reveal fine-tuning's performance boost
- Similar resolution in energy reconstruction, but in less time!
- Direction reconstruction improved significantly

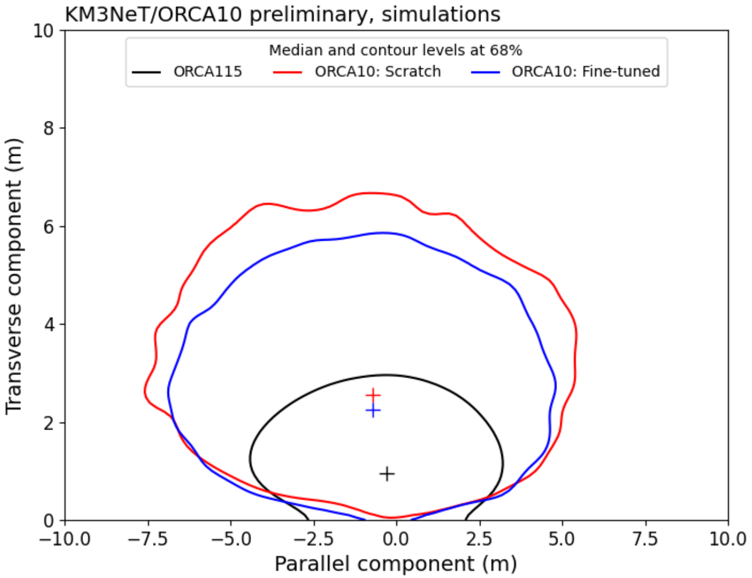
Preliminary results on multi-geometries for KM3NeT/ORCA

Interaction reconstruction vertex:

- The hardest task
- Dynamic detector coordinates from rbr approach
- Small fiducial volume burdens the reconstruction

Better data representation?
Build a latent space to effectively accommodate dynamic coordinates and detector conditions & geometries





Interaction vertex reconstruction at KM3NeT/ORCA6 (left) and KM3NeT/ORCA10 (right) projected over the neutrino direction for 1-100 GeV atmospheric neutrinos.

Preliminary results on multi-geometries for KM3NeT/ORCA

Interaction reconstruction vertex:

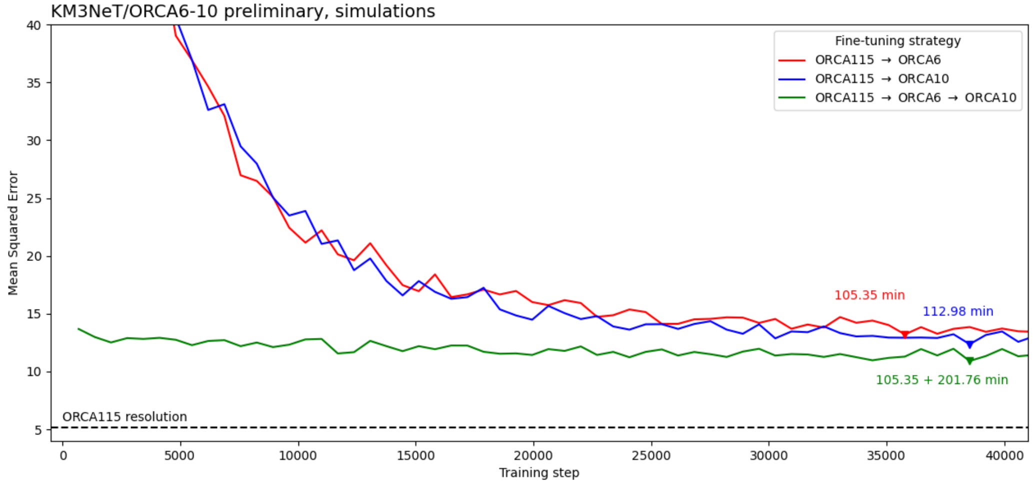
- The hardest task
- Dynamic detector coordinates from rbr approach
- Small fiducial volume burdens the reconstruction

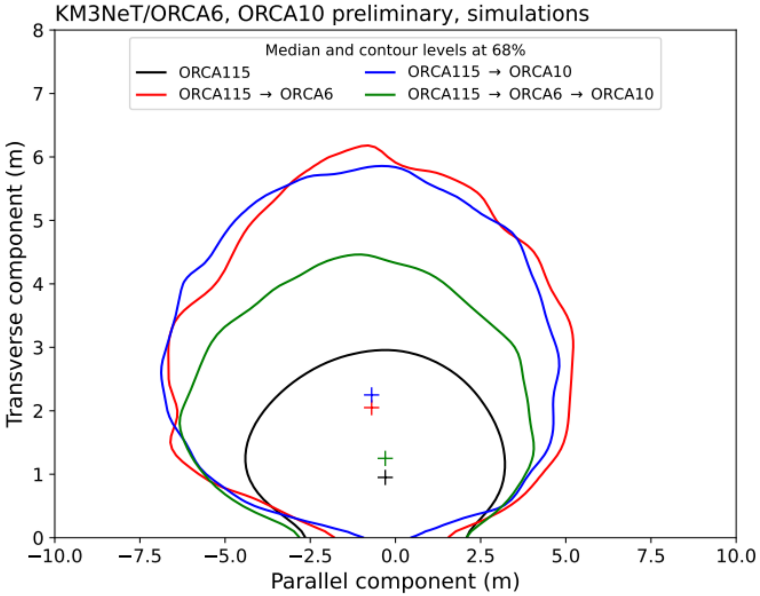
From big to small...

ORCA115 ⟶ ORCA6, ORCA10, etc.

...and vice versa

ORCA115 ⟶ ORCA6 ⟶ ORCA10 ⟶ etc.





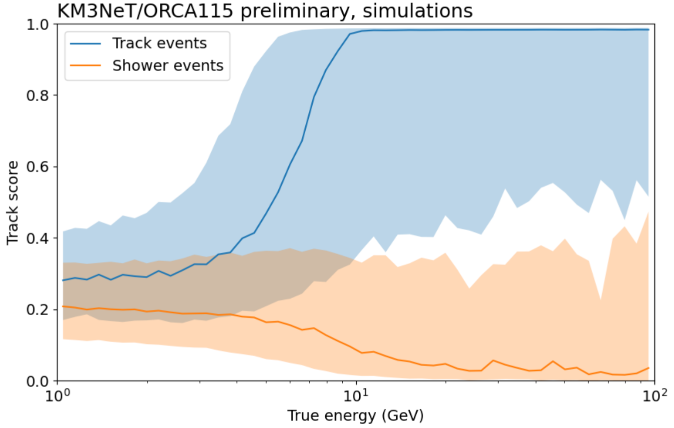
We need to propagate the knowledge between detectors!

Preliminary results on multi-task for KM3NeT/ORCA

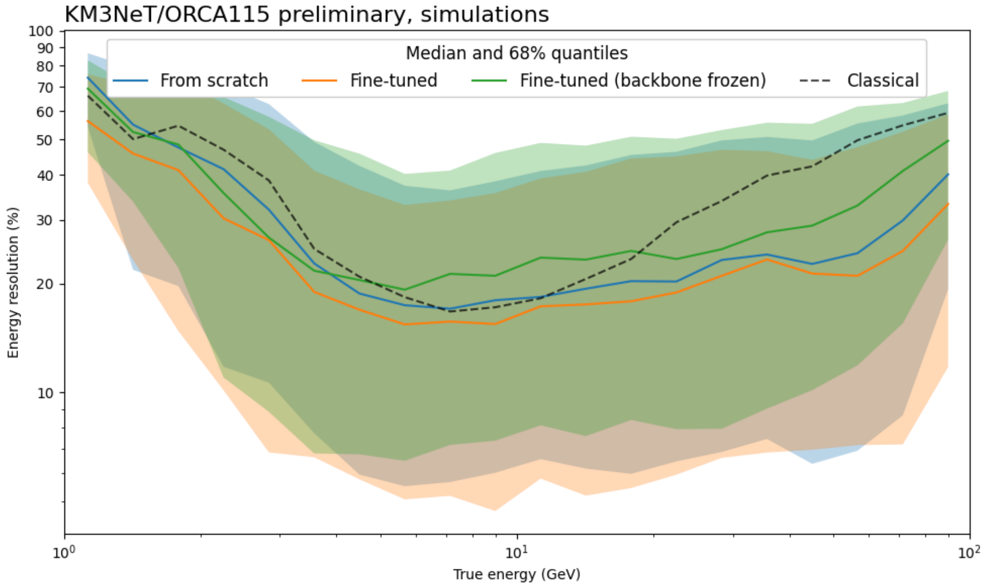
Classification to energy reconstruction

- A model that can handle multiple tasks
- A dataset 1 unrelated to dataset 2 (different detectors or water properties or atmospheric muons), helps a model into performing another task and makes it more robust

Multi-task study
ORCA115 dataset 1: track-shower
ORCA115 dataset 2: energy
850k tracks & 850k showers each



Track-score in function of neutrino energy in KM3NeT/ORCA115 for **dataset 1**



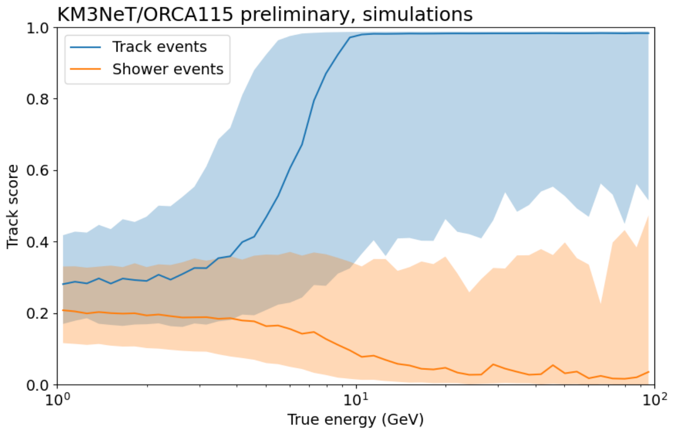
Energy resolution in function of neutrino energy in KM3NeT/ORCA115 for tracks from **dataset 2**

Preliminary results on multi-task for KM3NeT/ORCA

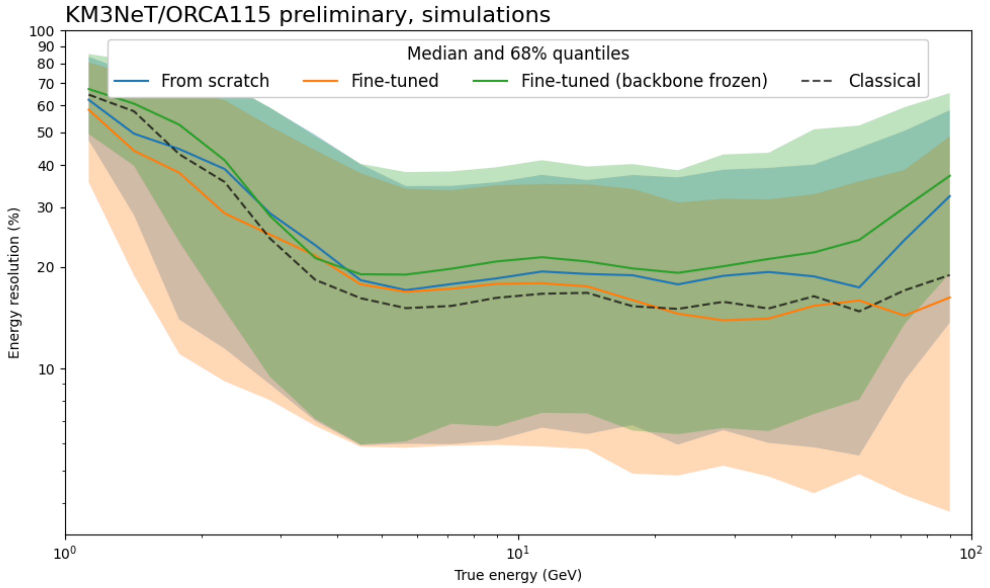
Classification to energy reconstruction

- A model that can handle multiple tasks
- A dataset 1 unrelated to dataset 2 (different detectors or water properties or atmospheric muons), helps a model into performing another task and makes it more robust

Multi-task study
ORCA115 dataset 1: track-shower
ORCA115 dataset 2: energy
850k tracks & 850k showers each



Track-score in function of neutrino energy in KM3NeT/ORCA115 for **dataset 1**

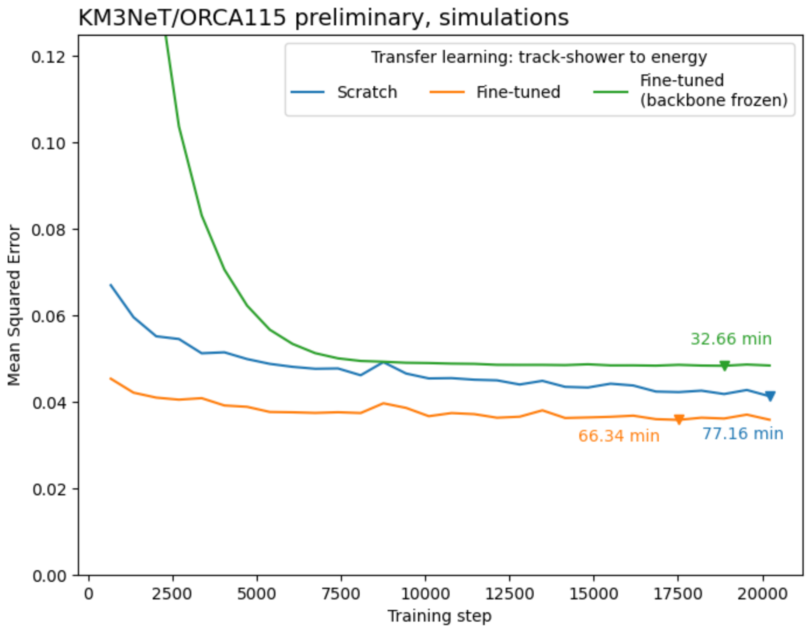


Energy resolution in function of neutrino energy in KM3NeT/ORCA115 for showers from **dataset 2**

Preliminary results on multi-task for KM3NeT/ORCA

Classification to energy reconstruction
- A model that can handle multiple tasks
- A dataset 1 unrelated to dataset 2 (different detectors or water properties), helps a model into performing another task and makes it more robust

Multi-task study
ORCA115 dataset 1: track-shower
ORCA115 dataset 2: energy
850k tracks & 850k showers each



Fine-tuned model shows **faster convergence and efficiency**.

Freezing the backbone has a **trade-off** between training speed-up and accuracy ⟶ suboptimal feature representation

Overall, the three cases achieve show improvements with respect to classical reconstruction methods.

Summary

Transfer Learning in multiple-detectors

- Transformers are particularly effective to deal with small detectors and very limited data
- Further optimization is still needed in vertex reconstruction

Transfer Learning for multi-task

- Speeds up training and boosts model robustness
- Leverages knowledge from different tasks

The road ahead

- **From simulations to data:** ensure consistency and accuracy when transitioning to real detector data
- **Robustness tests & uncertainties:** validate model reliability across different conditions and detectors
- Estimate improvements as the detector grows to optimize **scalability**
- Develop **common benchmark** with state-of-the-art models → On the way, see Jorge's talk
- Implement any deep learning reconstruction in the **official data processing pipeline** → Almost there
- Start testing pre-training models (BERT-like, GPT-like) with neutrino telescope data

Thank you for your attention!

Preliminary results on KM3NeT/ORCA115

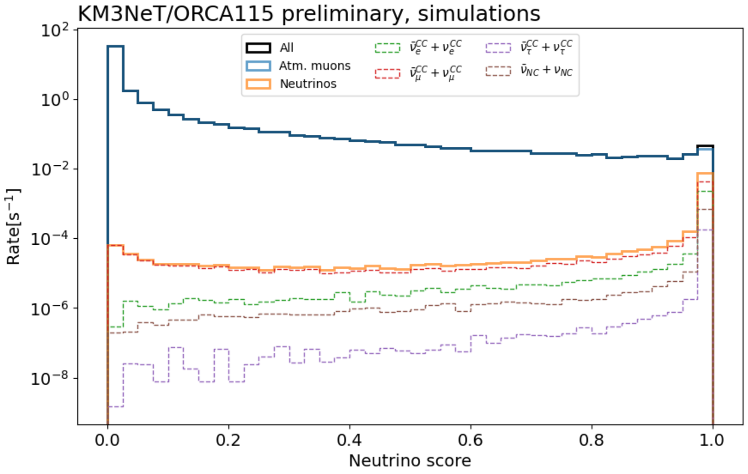
Motivation: the transformer is a language model

- KM3NeT/ORCA115 is the final detector, having all the possible neutrino physics encapsulated
- We can think of other configurations as similar languages to learn
- The information about KM3NeT/ORCA115 is used to understand our current detector

DL classification
Neutrino vs Background

DL classification
Track-like vs Shower-like

Track and shower reconstruction



Event rate for neutrino score (0 for atmospheric muons, 1 for neutrinos).

Purpose: reject background data (atm. muon) from neutrino signal.

Atmospheric muons are more energetic, having their starting & ending points in most of the cases, out of the fiducial volume.

The model easily isolates **neutrino events** as they are mostly **fully contained** in the detector.

Preliminary results on KM3NeT/ORCA115

AI GOES MAD²

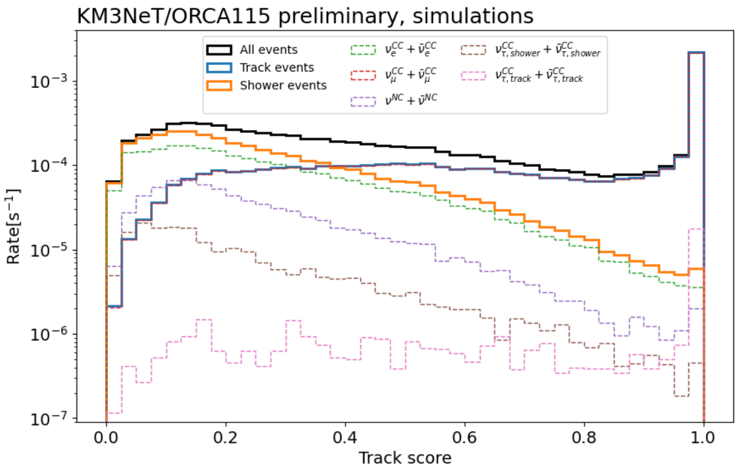
Motivation: the transformer is a language model

- KM3NeT/ORCA115 is the final detector, having all the possible neutrino physics encapsulated
- We can think of other configurations as similar languages to learn
- The information about KM3NeT/ORCA115 is used to understand our current detector

DL classification
Neutrino vs Background

**DL classification
Track-like vs Shower-like**

Track and shower
reconstruction



Event rate for track score (0 for showers, 1 for tracks) for 1-100 GeV atmospheric neutrinos.

Purpose: separate the two neutrino event topologies, track-like and shower-like.

Enough separation power below 10 GeV (AUROC = 0.82)

High separation power above 10 GeV (AUROC = 0.91).

Low energy events do not contain enough pulses to properly separate these two categories

Preliminary results on KM3NeT/ORCA115

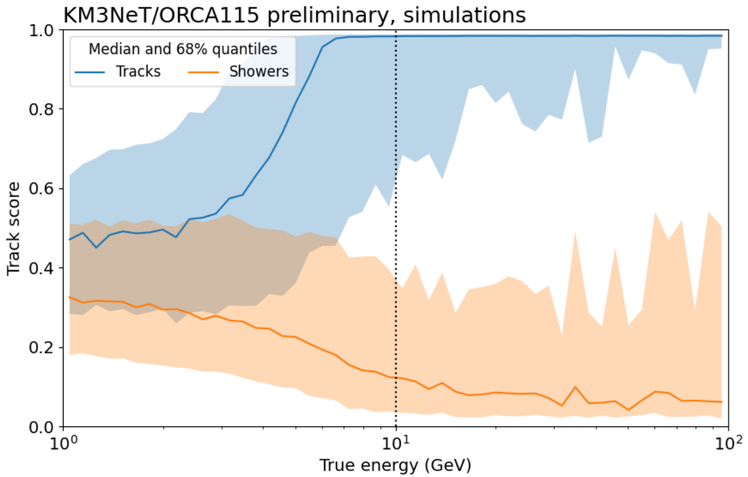
Motivation: the transformer is a language model

- KM3NeT/ORCA115 is the final detector, having all the possible neutrino physics encapsulated
- We can think of other configurations as similar languages to learn
- The information about KM3NeT/ORCA115 is used to understand our current detector

DL classification
Neutrino vs Background

**DL classification
Track-like vs Shower-like**

Track and shower
reconstruction



Track score (0 for showers, 1 for tracks) as funtion of neutrino energy for 1-100 GeV atmospheric neutrinos.

Purpose: separate the two neutrino event topologies, track-like and shower-like.

Enough separation power below 10 GeV (AUROC = 0.82)

High separation power above 10 GeV (AUROC = 0.91).

Low energy events do not contain enough pulses to properly separate these two categories

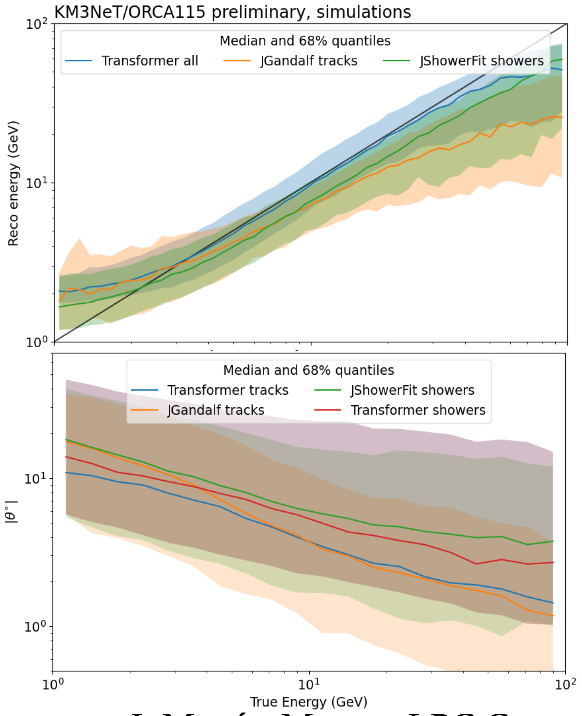
Preliminary results on KM3NeT/ORCA115

Motivation: the transformer is a language model
- KM3NeT/ORCA115 is the final detector, having all the possible neutrino physics encapsulated
- We can think of other configurations as similar languages to learn
- The information about KM3NeT/ORCA115 is used to understand our current detector

DL classification
Neutrino vs Background

DL classification
Track-like vs Shower-like

Track and shower reconstruction



True and reconstructed neutrino energy and zenith angle
Resolution for 1-100 GeV atmospheric neutrinos.

Purpose: reconstruct neutrino energy and neutrino direction.

Reconstruction done simultaneously for both track-like and shower-like events.

Saturation at high energies due to event containment.

Underestimation at low energies due to limited number of pulses.

Preliminary results on multi-geometries for KM3NeT/ORCA

Thanks to the modular structure of GraphNeT: **different detector configuration are easily handled.**

Multi-detector study
ORCA6 (Feb20 – Nov21): 4075 runs
ORCA10 (Dec21 – May22): 1889 runs
100k tracks & 100k showers

```python
detector = ORCA(
                    configuration = detector_config['path'],
                    du_selection = (
                                        detector_config['selection']['ORCA115'],
                                        detector_config['selection']['ORCAX'],
                    ),
                    shift_coordinates = True
)

data_definition = HitsSequence(
                                detector = detector,
                                node_definition = NodesAsHitsTimeSeries(
                                                                        input_feature_names = features,
                                                                        max_hits = config["data"]["max_hits"],
                                                                        trig_name = config["data"]["trig_name"],
                                ),
                                input_feature_names = features,
)

dataset = SQLiteDataset(
                            path = detector_config['path'],
                            truth_table = config['data']['truth_table_name'],
                            pulsemaps = config['data']['pulsemap'],
                            truth = truth,
                            features = features,
                            graph_definition = data_definition,
                            selection = config['selection'],
)

train_dataset, val_dataset = train_test_split(
                                                dataset,
                                                train_size = 1 - config['data']['validation_size'],
                                                test_size = config['data']['validation_size'],
                                                random_state = config['training']['seed'],
)
```

```python
class ORCA(Detector):
    """`Detector` class for ORCA."""

    def __init__(
        self,
        configuration: Optional[str],
        du_selection: Optional[Tuple[List[int], List[int]]],
        shift_coordinates: Optional[bool] = True,
    ) -> None:
        super().__init__()
        self.configuration = os.path.join(KM3NeT_GEOMETRY_TABLE_DIR, configuration)
        self.du_selection_ORCA115 = du_selection[0]
        self.du_selection_ORCAX = du_selection[1]
        self.shift = shift_coordinates

        self.geometry_table_path = os.path.join(
            KM3NeT_GEOMETRY_TABLE_DIR, "ORCA115.parquet"
        )

        if self.shift:
            self.x_shift, self.y_shift, self.z_shift = self._shift_to_ORCA115()

    geometry_table_path = os.path.join(
        KM3NeT_GEOMETRY_TABLE_DIR, "ORCA115.parquet"
    )
    xyz = ["pos_x", "pos_y", "pos_z"]
    string_id_column = "DU_id"
    floor_id_column = "floor_id"
    sensor_id_column = "dom_id"

    def _shift_to_ORCA115(self):
        ORCA115_df = pd.read_parquet(self.geometry_table_path)
        ORCAX_df = pd.read_parquet(self.configuration)

        ORCA115_DUs = ORCA115_df[ORCA115_df['DU_id'].isin(self.du_selection_ORCA115)]
        ORCAX_DUs = ORCAX_df[ORCAX_df['DU_id'].isin(self.du_selection_ORCAX)]

        x_shift = ORCAX_DUs['pos_x'].mean() - ORCA115_DUs['pos_x'].mean()
        y_shift = ORCAX_DUs['pos_y'].mean() - ORCA115_DUs['pos_y'].mean()
        z_shift = ORCAX_DUs['pos_z'].mean() - ORCA115_DUs['pos_z'].mean()

        return (x_shift, y_shift, z_shift)
```

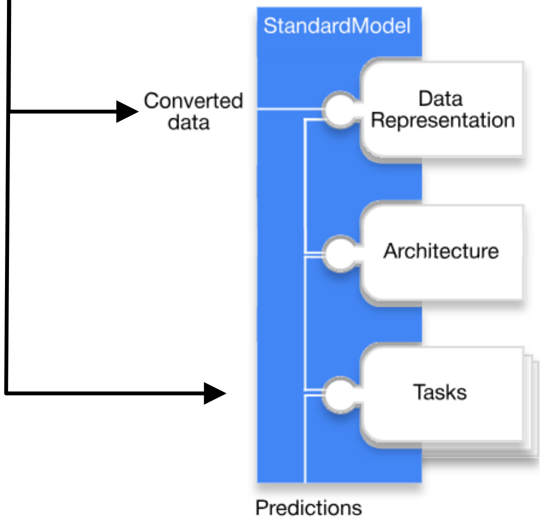
```python
def feature_map(self) -> Dict[str, Callable]:
    """Map standardization functions to each dimension of input data."""

    feature_map = {
        "t": self._dom_time,
        "pos_x": self._dom_x,
        "pos_y": self._dom_y,
        "pos_z": self._dom_z,
        "dir_x": self._dir_xy,
        "dir_y": self._dir_xy,
        "dir_z": self._dir_z,
        "tot": self._tot,
        "trig": self._identity,
        "channel_id": self._identity,
        "dom_id": self._identity,
        "du_id": self._identity,
    }

    return feature_map

def _dom_x(self, x: torch.tensor) -> torch.tensor:
    if self.shift:
        x = x - self.x_shift
    return x / 10.0

def _dom_y(self, x: torch.tensor) -> torch.tensor:
    if self.shift:
        x = x - self.y_shift
    return x / 10.0

def _dom_z(self, x: torch.tensor) -> torch.tensor:
    if self.shift:
        x = x - self.z_shift
    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 (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
```

Preliminary results on multi-task for KM3NeT/ORCA

AI GOES MAD²

Thanks to the modular structure of GraphNeT: **fine-tuning between tasks is as well possible.**

Multi-task study
ORCA115 dataset 1: track-shower
ORCA115 dataset 2: energy
850k tracks & 850k showers each




```python
# The model backbone
backbone = Transformer(
                            seq_length = config["backbone"]["seq_length"],
                            n_features = config["backbone"]["n_features"],
                            position_encoding = config["backbone"]["position_encoding"],
                            emb_dims = config["backbone"]["emb_dims"],
                            num_heads = config["backbone"]["num_heads"],
                            dropout_attn = config["backbone"]["dropout_attn"],
                            hidden_dim = config["backbone"]["hidden_dim"],
                            dropout_FFNN = config["backbone"]["dropout_FFNN"],
                            no_hits_blocks = config["backbone"]["no_hits_blocks"],
                            no_evt_blocks = config["backbone"]["no_evt_blocks"],
)

# The task definition
task = BinaryClassificationTask(
                                    hidden_size = backbone.nb_outputs,
                                    target_labels = config["task"]["target"],
                                    loss_function = BinaryCrossEntropyLoss(),
)

# The model: data + backbone + task(s)
model = StandardModel(
                        graph_definition = data_definition,
                        backbone = backbone,
                        tasks = [task],
                        optimizer_class = AdamW,
                        optimizer_kwargs = config["optimizer"]["parameters"],
                        scheduler_class = None,
                        scheduler_kwargs = None,
                        scheduler_config = None,
)

if config['training']['fine_tune']:
    backbone_weights = torch.load(config["pretrained"])['state_dict']

    model.load_state_dict(backbone_weights, strict = False)

    if config['training']['freeze_backbone']:
        for name, param in model.named_parameters():
            if name.startswith('backbone'):
                param.requires_grad = False
```