Eur. Phys. J. C (2024) 84:646
https://doi.org/10.1140/epjc/s10052-024-12977-2

THE EUROPEAN PHYSICAL JOURNAL C

Regular Article - Experimental Physics

IceCube – Neutrinos in Deep Ice

The top 3 solutions from the public Kaggle competition

Habib Bukhari[4], **Dipam Chakraborty**[5], **Philipp Eller**[1,2,a], **Takuya Ito**[6], **Maxim V. Shugaev**[3], **Rasmus Ørsøe**[1,2,b]

[1] Physics Department, TUM School of Natural Sciences, Technical University of Munich, 85748 Garching, Germany
[2] Munich Data Science Institute, Technical University of Munich, 85748 Garching, Germany
[3] Department of Materials Science and Engineering, University of Virginia, Charlottesville, VA 22904-4745, USA
[4] Ashburn, VA, USA
[5] Bangalore, India
[6] Tokyo, Japan

Received: 24 October 2023 / Accepted: 3 June 2024 / Published online: 25 June 2024

Abstract During the public Kaggle competition "IceCube – Neutrinos in Deep Ice", thousands of reconstruction algorithms were created and submitted, aiming to estimate the direction of neutrino events recorded by the IceCube detector. Here we describe in detail the three ultimate best, award-winning solutions. The data handling, architecture, and training process of each of these machine learning models is laid out, followed up by an in-depth comparison of the performance on the Kaggle datatset. We show that on cascade events in IceCube above 10 TeV, the best Kaggle solution is able to achieve an angular resolution of better than 5°, and for tracks correspondingly better than 0.5°. These results indicate that the Kaggle solutions perform at a level comparable to the current state-of-the-art in the field, and that they may even be able to outperform existing reconstruction resolutions for certain types of events.

1 Introduction

The IceCube Neutrino Observatory [1] consists of a detector installed deep within the antarctic glacier and spans a cubic kilometer of ice. Its mission is to probe the properties of fundamental particles and the astrophysics of these particles. The main subject of study, so-called neutrinos, are the most abundant matter particle in the universe. They are nearly massless and do not carry an electric charge, making them particularly difficult to detect. An important step in analysing the data collected by the detector is to estimate the direction the neutrinos came from based on the measurements of the faint traces of Cherenkov radiation resulting from neutrino interactions in the ice. This direction information is needed, for example, to unveil violent astrophysical neutrino sources [2,3], or to study neutrino properties [4–6].

1.1 Reconstruction in IceCube

Reconstruction of events is the process of turning the detector read-out data into high-level, physical quantities (such as the neutrino direction in our case), which is a parameter inference problem [7]. Traditional direction reconstruction algorithms in IceCube range from fast line fits [8] to increasingly sophisticated maximum likelihood estimators (MLEs). A key component to MLE techniques is the event reconstruction likelihood itself that describes the scattering and absorption of photons in the South Pole ice, which is considered intractable and therefore requires approximation. Simpler techniques rely on parameterized distributions as approximations [9], or simplify the likelihood by describing only the direction reconstruction by removing pulses originating from scattered light [10]. Such algorithms are often used as first-guesses and fast reconstruction of real-time alerts [11]. More accurate but slower approaches make use of the full reconstruction likelihood, as described in [12]. For instance, the method described in Ref. [13] is based on sophisticated photon ray tracing to approximate the full reconstruction likelihood, and can reconstruct all event topologies, but is limited to the GeV energy range. Beyond the GeV scale, similar approaches are followed that typically target a specific type of event [6,14]. Such methods have been widely used, most

[a] e-mail: philipp.eller@tum.de (corresponding author)
[b] e-mail: rasmus.orsoe@tum.de

recently in finding evidence for neutrino emissions from the NGC1068 Seyfert galaxy [15].

In recent years, new reconstruction techniques relying on neural networks (NNs) have emerged. In theory, NNs are able to approximate arbitrary functions, and once trained, they can reconstruct neutrino events many orders of magnitude faster than traditional methods. Techniques based on NNs are therefore both fast and flexible, as they can replace likelihoods, and can generalize to the entire energy range of IceCube. However, it has been an ongoing effort to identify model architectures that provide equal or superior reconstruction performance compared to the current traditional methods. This search is further complicated by the nature of neutrino telescope data; geometric time series is a data type that falls in-between established machine learning paradigms, making an a priori identification of ideal model architecture non-trivial. Some early attempts used convolutional neural networks (CNNs) to reconstruct energy and direction of high-energy cascade events [16]. A similar method was used for the energy reconstruction of neutrinos from NGC 1068 [17]. Also for the GeV energy range, an adaptation of CNNs have been used to perform a variety of reconstruction tasks [18]. Graph Neural Networks (GNNs) have been shown to further improve the reconstruction accuracy for low energy events [19], and have also been adapted for novel tasks such as pulse cleaning [20], but struggle to outperform traditional reconstruction methods beyond the GeV energy range. Hybrid approaches combine MLE techniques with deep learning, where the likelihood is either fully or partly approximated by neural networks [21,22]. Such a technique was recently used to find evidence of neutrino emissions from the galactic plane [23].

1.2 "Neutrinos in Deep Ice" Kaggle competition

Kaggle is an online platform where companies and institutions can present data science problems to the general public through competitions. In these competitions, members of the public can compete in teams to develop algorithms that perform best on a well-defined problem specified by the competition. We created the Kaggle competition "IceCube – Neutrinos in Deep Ice" [24], where the participants were tasked with developing direction reconstruction algorithms for various IceCube neutrino events. Given a detector response x which was induced by a neutrino with direction vector r_{truth}, the algorithms had to produce an estimate $r_{recon.}$ of the true neutrino direction. The scoring metric used to evaluate the quality of the algorithms was the mean opening angle between r_{truth} and $r_{recon.}$ computed over a large set of events. This particular choice of metric was based on the physics question of interest, and considerations on robustness imposed by Kaggle. A so-called "public" leader board score was computed on roughly half of the hidden test set upon submission and the score made available to the participants, while the "private" leader board score remained hidden from everyone until the completion finished. This private score determined the final ranking. This division of leaderboards is a technique used by Kaggle to encourage participants to produce solutions that generalize well to unseen data.

During the 3 month competition period from January 19th to April 19th 2023, a total of 6460 people entered the competition and 901 submitted at least one valid solution. At the end of the competition, the participants were distributed across 812 competing teams and a grand total of 11,206 solutions had been submitted [25], of which the top three, prize-winning solutions are presented in this article.

2 Competition details and dataset

The IceCube detector [1] consists of 5160 Digital Optical Modules (DOMs) [26] distributed on 86 strings at depths from 1450 m to 2450 m, see Fig. 1. The main array of the detector consists of 78 strings arranged in a near-hexagonal pattern each carrying 60 DOMs with a vertical separation of 17 m and an average horizontal distance between neighbouring strings of 125 m. Each DOM holds a 10$''$ Photomultiplier Tube (PMT) directed towards the center of the Earth. At a depth of around 2000 m, a layer of optical impurities lies embedded in the ice. The layer is mostly comprised of mineral dust, referred to as "the dust layer" and effects the scattering and absorption of light [27]. An additional 8 strings have been installed around the center string of the main array. On these strings, the DOMs are distributed differently than in the main array: Above the dust layer, at around 1750 m to 1850 m, 10 DOMs comprise the so-called "veto cap", a cluster of DOMs used to identify electrically charged particles entering the detector from above. Below the dust layer, from around 2100 m to 2450 m, a part of the ice where the optical transparency is highest, a second cluster of DOMs have been installed. This second cluster, named "DeepCore" [28], uses DOMs with higher quantum efficiency compared to the main array and the modules have a vertical spacing of just 7 ms, making it the detector volume with the highest density of DOMs.

When neutrinos interact in the ice, charged particles are produced that emit Cherenkov radiation as they transverse the ice. The PMTs in IceCube DOMs can detect these photons, and the amount of signal detected for a neutrino interaction may range from a few to more than 10^5 photons. The number of detected photons, however, is many orders of magnitudes lower than those emitted, and this signal is interspersed with noise from primarily radioactive decays in the glass housing of the DOMs.

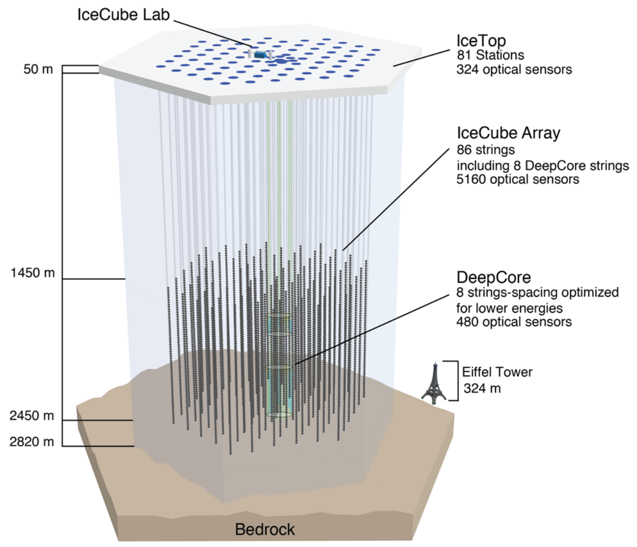

Fig. 1 Illustration showing the IceCube detector with its 86 strings vertically deployed in ice, instrumenting the depth between 1450 m and 2450 m. The 78 strings of the main array are arranged in a nearly hexagonal pattern, while the remaining 8 strings have a denser sensor spacing forming "DeepCore" and are highlighted in green. (Image courtesy of the IceCube collaboration.)

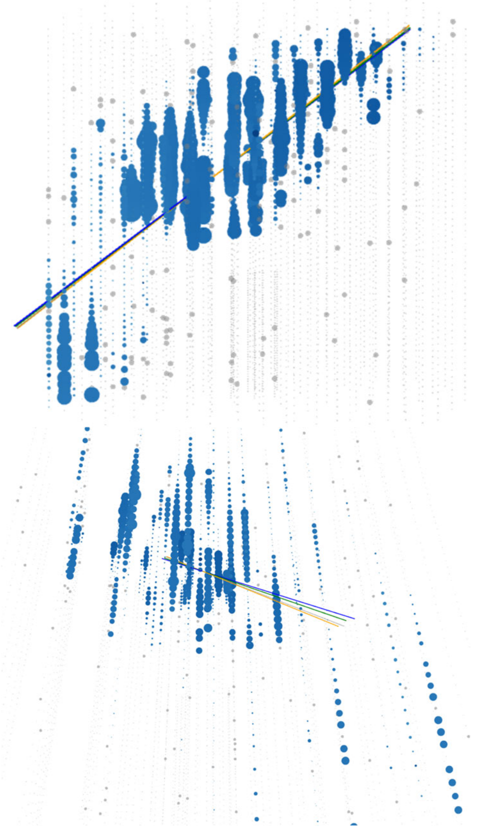

Fig. 2 A 22.4 PeV ν_μ CC event from the Kaggle dataset where the neutrino interacted outside the detector volume, allowing the muon to travel from above and through the detector with zenith and azimuth angles of 52° and 189°, respectively. HLC pulses are shown in blue. Solid grey points indicate non-HLC pulses. Not shown is the time information of the pulses. Neutrino direction reconstructions are shown with coloured lines, and the grey line depicts the true neutrino direction. The reconstructed directions were (53, 190), (52, 188), (53,189) for the 1st, 2nd and 3rd place solution respectively. Top: A side view of the event – the muon passes through the entire detector. Bottom: Close-up of the end of the track highlighting the differences in predictions

2.1 Neutrino events in IceCube

When photo-electrons produce a sufficiently large voltage at the PMT anode, a digitization process is triggered where the PMT waveforms are read out either partially or fully. If at least one neighboring DOM on the same string also records a signal within 1 μs, the hard local coincidence (HLC) condition is satisfied and the full waveform is read out. If this condition is not met, only minimal information around the peak voltage is read out [1]. The digitized waveforms are subject to an unfolding process [12] that estimates photon arrival times and the charge of individual photo electrons – each of which is referred to as a so-called "pulse". Based on the HLC hits recorded by the DOMs, event triggers set certain criteria for reading out all signal recorded by DOMs, including non-HLC pulses, for further processing. Different event trigger definitions are used in the IceCube online systems [29], and the events used in the Kaggle competition satisfy at least one of them.

Neutrino events in IceCube come mainly in two broad categories that have distinct geometric shapes. "Tracks": sufficiently energetic charged-current (CC) muon neutrinos (and roughly 17% of ν_τ interactions) produce a track signature. These events produce a muon that can travel long distances in the ice while emitting Cherenkov radiation, effectively producing a track-like signature."Cascades": other neutrino interactions that are not described by the track class. These interactions produce electromagnetic and hadronic particle showers in which the energy tends to be deposited over relatively small distances, effectively producing a localized deposit of light.

An illustration of a simulated track event is shown in Fig. 2. In this event, the neutrino interacted outside the detector volume, producing a muon that enters the detector from above. Because the neutrino had 22 PeV of total energy, the resulting muon is able to travel through most of the detector and a large number of DOMs measure Cherenkov radiation from the interaction. The true direction and reconstructed directions from each winning solution are added for comparison.

The participants of the kaggle competition were provided with nearly 140 million simulated neutrino events together with the true directions. These events span the energy range from 100 GeV to 100 PeV, and contain all flavours and interaction types, and hence both tracks and cascades. The neutrino events were simulated with an energy spectrum follow-

Table 1 Input data for Kaggle competition: neutrinos in deep ice

Feature	Description	Unit
(x, y, z)	Position of DOMs in IceCube coordinates	m
t	Pulse time relative to trigger time	ns
q	Charge of a pulse	P.E.
$Aux.$	If 0, the pulse is in HLC	–

ing a power law of $E^{-1.5}$ between 100 GeV and 1 PeV, and E^{-1} at the highest energies between 1 and 100 PeV. The ratio of neutrino flavors was roughly kept equal, resulting in a sample dominated by cascade events (∼2/3) and a sizable fraction of tracks (∼1/3). These events were grouped into 660 random sub-samples, so-called "data batches", each containing the detector response from 200,000 events. Each detector response x is a $[\mathrm{n}_{\mathrm{pulses}}, 6]$-dimensional array, where $\mathrm{n}_{\mathrm{pulses}}$ is the number of observed pulses in the trigger window, which can range from a few to several hundreds of thousands. For each pulse, the in-ice position of the PMT, arrival time, the associated charge and an auxiliary flag is provided. These six features are the input to the reconstruction algorithms and are shown in Table 1.

If the auxiliary flag is 1 it indicates that the specific pulse did not meet HLC criteria, and is thus more likely to originate from noise, but could also be scattered light. Each event is simulated using a unique set of nuisance parameters that represents the systematic uncertainties of the detector. This includes assumed scattering and absorption of light, and lets the collection of events represent a wide range of detection scenarios [30]. In addition, the detector responses have not undergone any pulse cleaning, a procedure that attempts to remove noise-induced pulses.

2.2 A competition baseline with DynEdge

To provide the participants with a baseline to compare against, we decided to train a GNN from GraphNeT [31], an open-source ML library for neutrino telescopes, on parts of the competition data. We submitted the predictions from the GNN to the leaderboard for comparison, and shared the trained model and technical material that allowed participants to fully tinker with every aspect of this method. Many solutions, including the winning solutions documented in this paper, took inspiration from the techniques in our baseline submission and produced their own versions aimed specifically for this competition. In this section we elaborate on some of those techniques. This baseline model also serves as a reference in the results section, where performance of the winning solutions is assessed and compared.

2.2.1 DynEdge

The specific GNN called "DynEdge" [19], is a flexible algorithm capable of reconstruction and classification for many different physics tasks on both a per-pulse and per-event level. DynEdge is a convolutional graph neural network which represents neutrino event as point cloud graphs. A graph is a collection of nodes n and edges e. In the competition baseline, we represented individual pulses of Cherenkov radiation as nodes, and edges are initially drawn to each node's k Nearest Neighbours (kNN) based on the Euclidean distance between the in-ice PMTs that measured the pulse. The data associated with each node is the pulse information shown in Table 1. The graphs are then convolved by an EdgeConv [32] layer, which updates the feature vector of the i'th node by adding the pair-wise differences between i'th node and each of its k neighbours. The layer is defined by

$$\mathbf{x}_i' = \sum_{j \in \mathcal{N}(i)}^{j=k} h_\Theta(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i) \tag{1}$$

where $\mathbf{x}_i$ represents the un-convolved feature vector of the i'th pulse, $\mathbf{x}_j$ the feature vector of the j'th neighbour of the i'th pulse and h_Θ is a learned function applied to all neighbourhoods. The convolved values of the i'th node is represented by $\mathbf{x}_i'$. DynEdge applies multiple of these layers in series, and between each layer the neighbourhoods are re-calculated based on each node's position in latent space, effectively letting the GNN learn the optimal edges for the given task. The most recent use of DynEdge in IceCube is in a study of IceCube Upgrade's expected sensitivity to atmospheric neutrino oscillations, where it was used to remove noise pulses, classify event topologies and for reconstruction [20].

2.2.2 Von Mises–Fisher loss

In our baseline submission we used the von Mises–Fisher (vMF) distribution for 3D vectors as a loss function [19]. By taking the natural logarithm, one obtains

$$\text{loss} = -\kappa \cdot \cos(\Delta\theta) - \ln(C_3(\kappa)) \tag{2}$$

where $\Delta\theta$ is the opening angle between the true and reconstructed 3D direction vector, and κ is a measure of uncertainty, analogous to $\frac{1}{\sigma}$ for normal distributions. The quantity $C_3(\kappa)$ represents the normalization constant of the vMF distribution, which requires numerical approximation for 3D vectors. The vMFs distribution for 3D vectors represents a 2-sphere embedded in $\mathbb{R}^3$ and is conceptually close to conventional choices in loss functions such as $1 - \cos(\Delta\theta)$ but has the added benefit of the uncertainty estimation through κ, which is estimated alongside the direction by DynEdge.

For the baseline, we trained DynEdge according to the procedure documented in [19] using the 3D vMF loss on 7.8% of the competition data without further optimization. This version was submitted to the leaderboard early in the competition, along with technical material, and achieved a score of around 1.018 on the public leaderboard, which is calculated as the mean opening angle taken over the evaluation dataset. When using the full dataset for training, the score drops to around 0.985.

3 Winning solutions

In this section, the technical details behind each winning solution are described. The details include core model architectures, choices in data preprocessing, standardization, training techniques and ensembling methods.

3.1 1st place solution

We designed a simple, lightweight model combining the EdgeConv [32] and the transformer architecture, allowing us to leverage the strengths of both architectures. By creating six variations of this model and ensembling them, we achieved a private leaderboard score of 0.960.

3.1.1 Preprocessing

Transformers can achieve a high accuracy even without extensive feature extraction when provided with adequate training data. Given this, our approach emphasized retaining the inherent characteristics of the data, applying minimal preprocessing.

The input features are as defined in Table 1. Due to computational constraints, the sequence for each event was truncated to a maximum of 6000 pulses for inference, and only events with a maximum of 200 to 500 (depends on model) pulses or less were used for training. Each input feature is scaled as follows.

$$x', y', z' = \frac{x}{500}, \frac{y}{500}, \frac{z}{500} \tag{3}$$

$$t' = \frac{t - 10^4}{115} \tag{4}$$

$$q' = \frac{\log_{10}(q)}{3} \tag{5}$$

In addition to these elementary features, the node homophily ratio [33] of (x, y, z, t) is extracted as an additional global statistic. These global statistics features are concatenated with the output from the global pooling layer shown in Fig. 3. These preprocessing steps were similar to those of DynEdge [19].

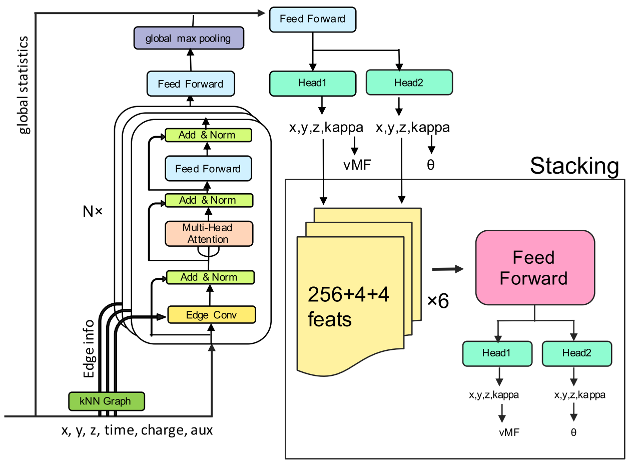

Fig. 3 This illustrates the overall architecture of the 1st place model. The left part represents the core structure of our method, combining EdgeConv and a Transformer layer, while the right part depicts the stacking process

3.1.2 Base model architecture

An overview of our model is shown in Fig. 3. The base model uses multiple blocks of EdgeConv followed by a transformer.

3.1.3 EdgeConv

In the original EdgeConv layer implementation, the feature vector $\mathbf{x}_i$ associated with DOM$_i$ is updated based on the difference $\mathbf{x}_j - \mathbf{x}_i$, where $\mathbf{x}_j$ represents the feature vector of the k-nearest neighbor DOM$_j$ to DOM$_i$, as seen in Eq. (1). This scheme works well for features like x, y, z, and time. But for features like charge and the auxiliary flag, its absolute values are important. Therefore, we updated $\mathbf{x}_i$ based on both the difference $\mathbf{x}_j - \mathbf{x}_i$ and $\mathbf{x}_j$ itself.

$$\mathbf{x}_i' = \sum_{j \in \mathcal{N}(i)}^{j=k} h_\Theta(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i, \mathbf{x}_j) \tag{6}$$

3.1.4 Edge selection

In the original EdgeConv implementation, edges are calculated in each layer dynamically by k-Nearest Neighbors (kNN). However, this edge selection scheme is not differentiable in itself and therefore does not have gradients. This methodology worked well for the segmentation task in the original EdgeConv paper [32], as points in the same segment are trained to be close in the latent space. However, direction reconstruction is fundamentally different, and therefore the edges used in the EdgeConv layers of this method are calculated based on the input features only once.

As illustrated in Fig. 3, our method first calculates the edges of a neutrino event. The event graph containing node features x and edges e is then passed through EdgeConv which uses the edges to convolve the node features. These latent, convolved node features $\mathbf{x}'$ are then added together with the original, unconvolved node features x in a skip-connection, and the result is normalized. The normalized quantity is passed to a transformer with multiple attention heads, and this output is also subject to skip-connection addition and normalization. The output is given to a MLP block with a final addition and normalization skip connection. This combination of EdgeConv and a Transformer layer forms the backbone of our method and may be repeated in serial n times, as denoted in the diagram. The Number of layers n is a hyperparameter which has been subject to tuning, and our final method utilizes multiple instances of this base model with different choices in n in an ensemble. The output of this series of EdgeConv + Transformer blocks is fed through an MLP block and is subject to global max pooling, producing a latent column vector for each event. This column vector is concatenated with *global statistics*, i.e. a set of engineered features that describe the entire event. This column vector is given to a final MLP with two prediction heads.

3.1.5 Loss function

While the von Mises Fisher Loss shown in Eq. (2) serves as a reliable and consistent loss function, it represents θ using cosine values. However, the metric of this competition is the angle θ itself. To minimize θ itself, we defined the loss function as follows:

$$\text{Loss} = -\theta - \kappa\cos(\theta) - ln(C_3(\kappa)) \tag{7}$$

This simple modification resulted in a 0.005 decrease in opening angle compared to von Mises–Fisher Loss, which corresponds to a difference of several places on the final Kaggle leader board.

3.1.6 Ensemble members

We made an ensemble of six models, whose configuration can be seen in Table 2. The dimensions of both EdgeConv and transformer layers are set to 256 and the number of attention heads for the transformer is set to 8 for all ensemble members. The number of EdgeConv+Transformer block in each model is either 3 or 4. The variables used to calculate the distance used for edge selection by kNN are either the 3d position (x, y, z) or the 4d input (x, y, z, t) and 6 edges are extracted for each node. The maximum sequence length for the transformer layers ranges between 200 and 500 pulses per event for training, but is set to 6000 pulses per event for inference. A maximum of about 30 epochs was used as we found training beyond this point gave increasingly diminishing returns.

These models were selected from experiments conducted during the limited competition period and were not designed to achieve the best result. In particular, model dimension and depth of the EdgeConv+Transformer block is very small, model parameters are only 6 M for the largest model, and we believe that a larger model would give better performance.

3.1.7 Ensemble method: stacking

Our approach to ensembling incorporates a stacking strategy, where the predictions from the models listed in Table 2 are used as input for a final model. The six ensemble members have been trained with varying configurations, each capturing different nuances of the data. By using these as embedding extractors, the stacking model can benefit from the diverse representations and deliver a more comprehensive prediction.

The stacking model is a three-layered Multi-Layer Perceptron (MLP) with 512 dimensions. It takes as input the prediction of x, y, z, κ and the last 256-dimensional hidden layer from each of the ensemble members. The detailed architecture of this process is illustrated in Fig. 3. The same loss functions used in base models are used for this stacking model. Through stacking, a further decrease in average opening angle by $0.003°$ was achieved.

3.1.8 Training procedure

The Adam optimizer [34] was used for all models. For models M1, M3, M4, and M6, which were trained from scratch, the learning rate was set to 10^{-3} for the first 15 epochs, and then it was linearly decayed from 10^{-3} to 10^{-6} for the subsequent 15 epochs. The fine-tuned models M2 and M5 had a learning rate of 10^{-4} in the initial half of their training, and during the latter half, their learning rate was linearly decayed from 10^{-4} to 10^{-7}. To effectively train our models on this voluminous dataset, we employed several optimization techniques. Specifically, by incorporating Mixed-Precision Training [35] and Sequence Bucketing, we managed to reduce the GPU memory consumption to about a third, while achieving over three times faster computational speed. This approach allowed us to handle the data very efficiently while preserving the accuracy of our models.

3.1.9 Mixed-precision training

Mixed-precision training is a technique where input data and model weights are cast from the high-precision 32-bit to 16-bit floating point precision during training. This is known to both decrease memory usage and speed up computations. When we first used mixed-precision, it made the training unstable. However, by appropriately placing the batch normalization at specific positions as shown in Fig. 3, we achieved stable progression in the training process.

Table 2 Overview of ensemble members and their configurations. n_l: Number of EdgeConv+Transformer block layers in serial connection. k: The positional information used for computation of edges, 3d denoting the three spatial dimensions and 4d denoting the inclusion of time. b_s: Batch size used for training. s_l: Maximum input sequence length used for training. n_e Number of epochs. O: Origin of model, "–" indicates training from scratch. $S_{valid.}$: Local validation score. S_{public} and $S_{private.}$ are the public and private leaderboard scores, respectively

ID	l	k	b_s	s_l	n_e	$S_{valid.}$	S_{public}	$S_{private}$	O
M1	4	3d	1000	200	30	0.967	0.964	0.964	–
M2	4	3d	1000	200	2	0.967	0.964	0.963	M1
M3	3	3d	500	300	30	0.969	0.965	0.965	–
M4	3	4d	1000	250	30	0.971	0.968	0.968	–
M5	4	3d	1000	200	12	0.966	0.963	0.963	M2
M6	4	4d	500	500	30	0.968	0.964	0.964	–

3.1.10 *Sequence bucketing*

In transformer models, the self-attention mechanism inherently has a quadratic computational complexity with respect to sequence length. As sequence lengths increase, this leads to significant computational and memory challenges. Sequence bucketing addresses this issue by grouping sequences of similar lengths together in "buckets" [36]. As illustrated in Fig. 4, this approach minimizes the need for excessive padding, optimizes GPU memory use, and reduces the number of unnecessary computations. Consequently, using sequence bucketing can enhance the computational efficiency of training transformer models without compromising on performance. In our training, we chose the threshold a in Fig. 4 such that the samples within a batch are split in a $8:2$ ratio. This ratio was determined experimentally to minimize the GPU memory usage.

In addition, some ensemble members were not trained from scratch, but further trained from other ensemble members. M2 was fine-tuned from M1 and M5 was fine-tuned from M2. Out of the 660 batches of competition data, the last batch was used for local validation of ensemble members during training. The batch size was set as large as possible to fit in GPU memory, ranging from 500 to 1000. A maximum of about 30 epochs was used for all ensemble members as we found training beyond this point gave increasingly diminishing returns, however some models trained for fewer epochs as seen in Table 2.

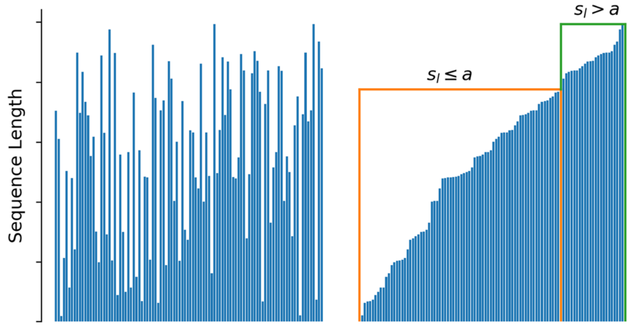

Fig. 4 Illustration of sequence bucketing for sequence data with varying lengths. The y-axis denotes the sequence lengths, whereas the x-axis denotes the order of appearance in a batch of training data. Left: Unordered set of sequences in a batch. Right: The set of sequences are sorted according to their lengths and sliced into two "buckets". S_l denotes the sequence length and a represents a threshold choice

3.2 2nd place solution

In our pursuit to better predict the direction of neutrino particles based on data from the IceCube Neutrino Observatory, we have critically assessed the constraints of conventional GNNs. These include their predominantly local function and unexpectedly slow computational processing in comparison to transformer models. In particular, in our tests, 1.4M parameters and a 4 layer DynEdge reference model requires the same computational budget for training as a 16 block and 7.4M parameters transformer T model, with the latter providing a significant improvement of the mean opening angle. Therefore, our proposed solution hinges on transformer models, which we see as evolved GNNs that operate on a fully connected graph, dynamically estimating edge weights via attention mechanisms [37]. In addition to transformers, a Fourier encoder is central to our method, which embeds the continuous input variables, shown in Table 1, into a multidimensional space used in the model. This addition significantly improved the quality of our direction reconstructions. Also, our method relies on a DynEdge-inspired feature extractor and a custom implementation of the Minkowski space-time line element as an attention bias, which refined our method further. Our code is available as open source [38].

3.2.1 *Preprocessing*

This section details the techniques used for data standardization, feature engineering, and data subselection.

3.2.2 Standardization techniques

Each of the detector coordinates was divided by 500. The time was shifted by 1×10^4 towards the beginning of the event and then was divided by 3×10^4. For the charge, we took base-10 logarithm and divided the result by 3. This standardization is similar to the reference DynEdge model [19]. For the length of the event, we took the base-10 logarithm of the total number of pulses. The Minkowski space-time interval used the same normalization as the detector coordinates. The ice properties data was taken from [39]. The depths were adjusted by subtracting 1950 m and subsequently normalized by dividing by 500 m. The scattering and absorption lengths were standardized using the `RobustScaler` from the scikit-learn library [40], ensuring the robustness of the transformation to outliers.

3.2.3 Feature engineering

The input to the model consists of a sequence of pulses described by the variables in Table 1. The base-10 logarithm of the total number of pulses is included as an event-level feature to provide the model with information on the event length in case of under-sampling of long events, as described in the next subsection.

We found it crucial to process the continuous variables, e.g. time and charge, into a representation suitable for transformers using the Fourier encoding method. Fourier encoding is a technique from signal processing and is frequently used for language models to describe the position of a word in a sentence [37]. This encoding method, however, generalizes beyond natural language processing and, when applied to a continuous signal, can be viewed as soft digitization of the input into a set of Fourier sine/cosine modes with $10{,}000^{2j/d}$ frequencies, where d is the embedding width, and integer j is changing from 0 to $d/2 - 1$ [37].

In our setup, the continuous input variables undergo the Fourier encoding method, while the discrete auxiliary variable is transformed using a learnable embedding. By multiplying the normalized time and position by 4096 and charge by 1024, we attained sufficient digitization resolution. Specifically, this resolution is dictated by the highest considered Fourier frequency, which is equal to 1. A change of the normalized input continuous variable by 2π results in the identical value in this lowest bit of the digitization. The model may capture lower variations of the input due to the continuous nature of the Fourier encoding, but the signal change may be too weak for the model to be treated effectively. For example, with the above-described scaling, we increased the sensitivity of the embedding to small input variations and can achieve 3×10^4 ns$/4096 \times 2\pi = 1.2$ ns digitization resolution for time (3×10^4 ns comes from the time normalization). This input up-scaling is critical, and the use of the Fourier encoding method with sufficiently large scaling coefficients significantly improved the model performance in our experiments.

In addition to Fourier encoding, in part of our experimental setups we have incorporated an optional DynEdge-inspired encoder with several adaptations, which provides an additional set of latent features as input to our base model.

Lastly, to combat the noise induced pulses, we further introduced a relative space-time interval bias based on the Minkowski metric given by $ds^2 = c^2dt^2 - dx^2 - dy^2 - dz^2$. The metric is used to compute the line element between all pulses in a given neutrino event and encodes their causality. We define the element as $ds = \text{sign}(ds^2) \cdot \sqrt{|ds^2|}$ clipped at $(-4, 4)$ and to achieve higher digitization resolution, we divide ds by 1024 before passing ds through the Fourier encoding. This engineered feature is added to the first transformer blocks in our base architecture as a relative attention bias [41], which provided a noticeable improvement in the performance.

In addition to the feature engineering above, we explored the impact of including ice transparency and absorption as additional features. Despite these extended features offering an alternate representation for the z-coordinate, because of their z-dependence, no noticeable improvement to the mean opening angle was found when including the optical properties of the ice.

3.2.4 Sub-selections and filtering

The IceCube challenge considers relatively short events with 62 median and 163.4 average number of pulses, and only 1.4% of events are longer than 768 pulses. Therefore, we trained our models with the maximum sequence length of 192 and used 512 lengths in validation and 768 in the final submission. In scenarios where events exceeded the pre-determined maximum sequence length, a tiered selection mechanism was employed. The primary preference was given to HLC pulses and in cases where the number of available HLC pulses fell below the pre-determined sequence length, non-HLC pulses were sampled. Other sampling techniques based on charge and arrival time were investigated but did not improve upon the choice described above. Consideration of longer sequences may improve the model performance on cascade events with a large number of pulses.

3.2.5 Base model architecture

Central to our method is a transformer model that interprets each event as a sequence of pulses, and the base model diagram can be seen in Fig. 5. We use transformer blocks with learnable shortcuts, similar to BEiT [42], and the first 4 transformer blocks are modified according to [41] in order to incorporate the relative space-time as attention bias. The continuous input variables are processed with the Fourier encoder and the optional DynEdge-inspired encoder to create features, which are then concatenated and fed to the transformer. In the DynEdge encoder, we switched all ReLU activations to GELU. Unlike the standard implementation, we opted not to employ pooling operations and instead utilized the latent features from the encoder. In the first 4 blocks of the transformer, relative space-time intervals are given to the Fourier-encoder as attention bias.

The first 4 transformer blocks are followed by 12 regular transformer blocks with learnable shortcuts. The input sequence to these blocks is expanded with a *cls* token [43], which is a special token often used in transformer architectures, primarily to represent the entire input sequence for tasks like classification. In the context of our model, the *cls* token is targeted to aggregate the information about the neutrino direction when the data propagates through the transformer blocks. After the last transformer blocks, this *cls* token is projected into a 3-dimensional vector, which characterizes the predicted direction of the neutrino and the model's confidence in the prediction (the vector length).

The details of the models, considered in our experiments, are summarized in Table 3. Our model sizes are referred to according to ViT [44] notation: *T* for tiny (192), *S* for small (384), and *B* for base (768). The computational cost of the smallest considered *T* model is similar to the competition DynEdge baseline.

3.2.6 Ensemble members

Several experiments were carried out to determine the effect of the model size, pooling mechanism, head size, and use of the Fourier and DynEdge-inspired encoders on the model performance, and to derive the optimal sets of hyperparameters for the base model architecture. Our experimental efforts are summarized in the table below for a series of key configurations. In our evaluations, the Cross-Validation (CV) score was gauged at $L = 512$ maximum sequence length (denoted as CV512). The last five data batches (655 through 659) served as our validation set. Columns S_{public} and $S_{private}$ refer to the evaluation on the test data of the IceCube challenge for the public and private leaderboard, respectively. The models included in our final ensemble are highlighted in bold.

T d32 is our baseline model used for the optimization of the training pipeline, which provides a significant boost over the DynEdge reference model. To disentangle the effect of the Fourier encoder on the performance, we have performed an experiment *T d32 no Fourier* using a simple projection of the continuous input variables into the transformer dimension. This experiment results in a significantly lower mean opening angle suggesting that it is crucial to convert the continuous input variables into a form suitable for transformers. The next experiment *T d32 avr pool* uses a simple masked average pooling over all tokens in the sequence instead of *cls* token based pooling. This experiment results in nearly identical performance to our *T d32* baseline, and we have chosen *cls* token setup for all further experiments. *S d32* and *B d32* experiments highlight the effect of the model size on the performance, suggesting an approximately 0.003 decrease in the mean opening angle with each increase of the model from *T* to *S* to *B* size. This up-scaling, however, carries the trade-off of a four-fold increase in model parameters and the computational cost. In addition, we have noticed the onset of over-fitting in training B models, and the size of the dataset provided for the IceCube challenge may be insufficient for training B and larger models. *B d64* experiment illustrates the effect of the head size on the model performance, suggesting that a smaller head size of 32 is preferable to 64, typically used in comparable size language and vision transformer models [43,45]. The next series of experiments (*B+DynEdge*, **S+DynEdge*, and **B+DynEdge*) are targeted at the study of the effect of the DynEdge-inspired encoder. These experiments exhibit rather contradictive results and may be affected by the use of different hardware, which permits a larger actual batch size. Overall we see a small improvement or no improvement from using DynEdge-inspired encoder. Our best final submission to the competition was an ensemble of 5 experiments highlighted in bold in Table 4 because their combination provided the best validation score.

3.2.7 Ensembling method

Our best final submission achieved a score of 0.9594 at the public and 0.9602 at the private test sets (0.9610 CV512). We considered a simple weighted average of the 3-d vectors predicted by the models with weights listed in Table 4, which are fitted to maximize the validation score. The use of the weighted average is partially dictated by the property of von Mises–Fisher Loss (applied in training) that enforces the model to correlate the vector length with the confidence [46]. Therefore, a weighted average of the model predictions automatically biases the final prediction toward the most confident direction. Ensembling only offered a marginal improvement over our best single model result, which stands at 0.9608 public and 0.9618 leaderboard (0.9627 CV512). We also considered stacking with fitting a second-level few-layer neural network model combining predictions of the first-level models (Table 4) and additional features, e.g. sequence length, first detection time, and the total charge. However, these experiments did not bring any improvement over the simple weighted average baseline.

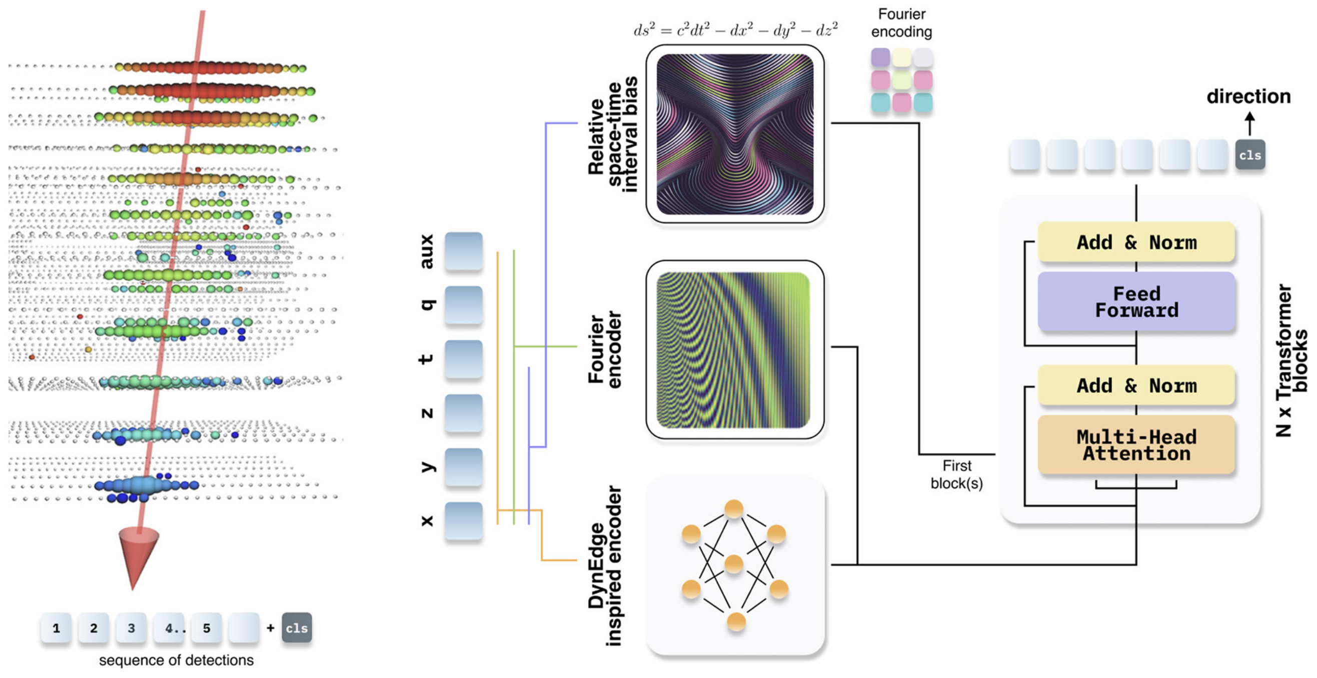

Fig. 5 Illustration of the 2nd place transformer-based setup for interpreting events as sequences of pulses. The flow begins with feeding pulses to the Fourier encoder to process continuous input variables. In parallel, there is an option to use the DynEdge-inspired encoder for additional feature extraction. The outputs from these stages are then concatenated. The first 4 transformer blocks employ relative space-time interval bias for clustering pulses based on their causality. In later blocks, the input sequence is expanded with a *cls* token, which is projected at the end into a 3-dimensional vector with the neutrino direction

Table 3 Model specifications summary. In the "Depth Conf." column, the depths are represented in the order: Relative Bias Blocks, Dynamic Edge Blocks, and Transformer Blocks. Each entry denotes the number of blocks (or layers for DynEdge-inspired encoder) in each respective component. A dash "–" indicates that a component is not present in the model

Model	Dims	Head size	Depth conf.	Parameters
T	192	32	4/-/12	7.57M
S	384	32	4/-/12	29.3M
B	768	32 or 64	4/-/12	115.6M
S+DynEdge	384	32	4/4/8	23.3M
B+DynEdge	768	32 or 48	4/4/12	116.5M

Table 4 Performance breakdown of model configurations across benchmarks. *d* in the Model column refers to the head size. Models with an asterisk (*) were trained on 2×A6000 hardware, a distinct choice from the RTX4090 that permits a significantly larger actual batch size. Models used in our final ensemble are highlighted in bold, and the W_e column represents the weight used for each ensemble member in the final linear combination of predictions

Model	W_e	CV512	S_{public}	$S_{private}$
T d32	–	0.9704	0.9693	0.9698
T d32 no Fourier	–	0.9900	0.9883	0.9871
T d32 avr pool	–	0.9705	0.9693	0.9692
S d32	–	0.9671	0.9654	0.9659
B d32	0.0825	0.9642	0.9623	0.9632
B d64	0.1535	0.9645	0.9635	0.9629
B+DynEdge d48	0.1937	0.9643	0.9624	0.9627
***S+DynEdge d32**	0.2360	0.9639	0.9620	0.9628
***B+DynEdge d32**	0.3343	0.9633	0.9609	0.9621

3.2.8 Training procedure

For the effective management and utilization of the dataset provided by the organizers, several data loading techniques were employed during training, each summarized below.

- **Data chunk caching**: The provided dataset [24] is too large to be fully stored in RAM on small workstations. On the other hand, loading each data chunk and grouping the pulses based on the event takes a substantial time and makes it prohibitively expensive to perform random access to the dataset during training. To mitigate the computational overhead from data loading and preprocessing, a caching mechanism was employed, i.e. recently used data chunks are stored in RAM, while the older chunks are removed. This strategy enables fast random access to data from recent chunks and at the same time drastically reduces the RAM requirement compared to storing the entire dataset.
- **Chunk-based random sampling**: The limitation of the plant chunk caching, however, is that under random sampling there is only little chance that the same chunk is sampled multiple times within a short time, and the computational overhead on data loading is still significant. To enable effective cache utilization, we have implemented a strategy that selects a random data chunk first and then samples randomly all events within this chunk before going to the next one, i.e. each data chunk is loaded only once per epoch while the sampling is randomized. This strategy provides a good balance between random access and utilization of computational resources.
- **Sequence bucketing**: All sequences in the input batch (not to be confused with "data batches" used above to refer to individual files in the dataset) must be padded to the longest sequence length in the batch before feeding to a transformer model. Meanwhile, the processing of these padding tokens brings an additional computational overhead. For random batch sampling (the default option in deep learning libraries, e.g. Pytorch [47]) each batch has a substantial chance of having at least one sequence significantly exceeding the median sequence length of the dataset. Processing of padding tokens in a such batch may take several times more computational budget than processing of actual sequence tokens. For example, the competition dataset has a median sequence length of 62 while the maximum sequence length in our training is set to 192. To minimize this computational overhead we adopted a length-matching batch sampling strategy that constructs batches of events with approximately the same length (buckets of the length multiple of 16). This sequence bucketing has enabled training with the maximum sequence length of 192 and inference at lengths up to 768 without a substantial increase in computational time. Using a sequence length of 768 during inference yielded a noticeable performance improvement when compared to the 192 length.

Data chunks 655 to 659 were reserved for local validation, while the remaining data chunks were used in training. In all experiments, the AdamW optimizer [48] with a weight decay of 0.05 was utilized. We employed a cosine annealing scheduler with a warm-up, adjusting the learning rate according to a cosine function over the course of training. The scheduler was reset at each epoch. The effective batch size was maintained at 4096 using the gradient accumulation [49] technique to navigate hardware limitations.

In our training in the first 2–3 epochs, we used the von Mises–Fisher loss. In the remaining epochs, the model is fine-tuned with a loss function defined as

$$\text{Loss} = \theta + 0.05 \cdot \text{vMF} \tag{8}$$

where θ denotes the opening angle, and vMF represents the loss defined in Eq. (2). This definition of loss emphasises the competition metric, while retaining a small contribution from the vMF loss, which gives access to the prediction confidence in terms of the κ-parameter, which is a useful variable for ensembling multiple models.

The training was carried out for 4–5 epochs depending on the loss plateau, but the most notable performance improvement occurred within the first epochs. At the initial epoch, T and S models had the maximum learning rate of 5×10^{-4}, whereas the B models utilized a rate of 1×10^{-4}. In the following epochs, the maximum learning rate was reduced from 2×10^{-5} to 0.5×10^{-5}. Mixed precision is used to accelerate the training. From the second epoch onward, the incorporation of Stochastic Weight Averaging (SWA) [50], a method that averages model weights across multiple training steps for better generalization, was helpful in improving the model performance.

Training time varies according to the model configuration. Specifically, the T model with 7.57M parameters takes about 10 h per epoch, whereas the *B+DynEdge* model with 116.5M parameters required 56 h per epoch on an NVIDIA RTX4090.

3.3 3rd place solution

Recent literature that applies machine learning to IceCube data often use Graph Neural Networks (GNNs). DynEdge presents neutrino events as point cloud graphs where edges are dynamically learned in small neighbourhoods, thus treating reconstruction of neutrino events as a dominantly geometric learning problem. We posit that treating IceCube data as a sequence learning problem is the optimal choice. Following this, we make use of the transformer architecture in our solution. Empirically we found transformers to have higher utilization of GPUs, and run faster than GNNs both during training and inference, despite doing more computations. We also find that transformers scale well with the amount of data, closely following known scaling laws that guide the training of large language models [51]. We model angle prediction of IceCube data as both a classification and regression task in our solution. We found each to have their own strengths and weaknesses. Finally, we combine both predictions using scikit-learn's HistGradientBoostingClassifier [40], a method that uses an ensemble of decision trees. The training[1] and inference[2] code for our solution is open-sourced. Figure 6 depicts the high level architecture of the solution.

3.3.1 Preprocessing

We treat each event as a sequence of pulses sorted by their arrival time from early to late. We represent the neutrino events as sequences with lengths up to 3072. If an event has more than 3072 pulses, we prioritize HLC pulses. For events with more than 3072 HLC pulses, we randomly sample 3072 HLC pulses. For each event, we set the time of first pulse of the sampled sequence to 0. Each element in the sequence is a vector of pre-processed data about each pulse.

3.3.2 Standardisation techniques

The x, y, and z coordinates are each divided by 500. Relative time values are divided by 30,000. We take the logarithm of the charge values and divide it by 3. These choices are similar to the standardization done for DynEdge.

3.3.3 Feature engineering

We have engineered features specific for each ensemble member in our final method. First, for the transformer model we add quantum efficiency of the PMTs, and optical properties of the ice near each sensor taken from [26]. While we use the features mentioned in the final models, we note the performance without these features in early experiments done with small transformers only gave a small benefit than that without the features.

Second, for the gradient boosting model, we added many features, as shown in Table 5. Most of the features do not seem to affect the performance. SHAP values indicate only few are important, namely κ, first pulse absolute time, and disagreement between predicted angles. Third, we prepared the zenith and azimuthal angles as binned truth values for the angle classifiers. The angles are quantized into 128 bins and the spacing between azimuth bins is uniform. The spacing between zenith bins is uniform in cosine space.

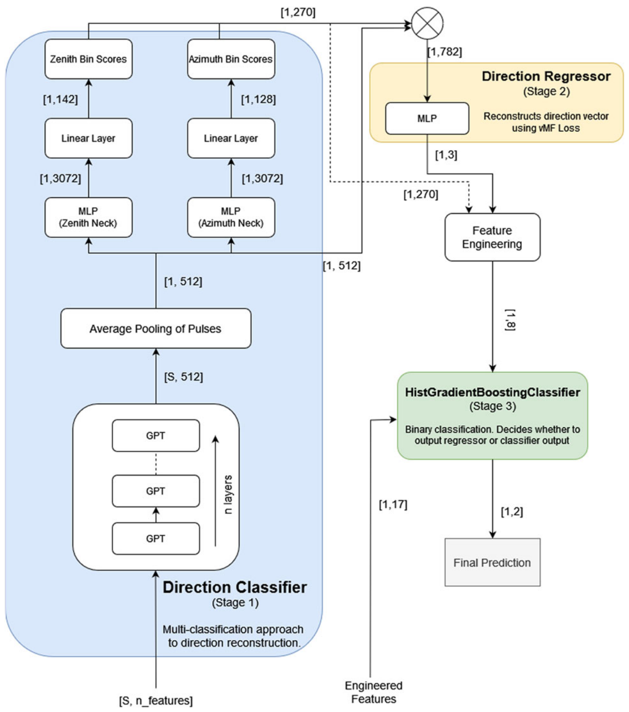

Fig. 6 Model architecture of the 3rd place solution. We use a multi-staged architecture to produce the angle for each event. The main component is the transformer backbone, which is trained from scratch on preprocessed events. Each event is modelled as a sequence of pulses for which the transformer produces a sequence of embeddings. These embeddings are then average pooled and used to produce independent classification predictions of azimuth and zenith. The same embeddings are also used to do perform regression for the 3D vector corresponding to the angle. Finally, the regression and classification predictions are combined using a gradient boosting classifier, which takes the predictions as well as some hand engineered features as it inputs. The final solution uses three independent copies of this architecture, and their predictions are combined to produce the predictions we submitted to the competition

3.3.4 Base model architecture

We use a multi-stage approach to predict the angles for each event, as shown in Fig. 6. The main component of our model is the transformer. We use standard GPT[3] [52] layers as the main building block. We preprocess and optionally subsample each sequence to as the input to the transformer, which produces a sequence of latent vectors for each element in the sequence. This sequence is then average pooled to produce a single vector, that feeds into classification heads. The transformer predicts the classification of azimuth and zenith with separate classification heads. Subsequently, the classification outputs and averaged pooled vector are passed into the

[1] https://github.com/dipamc/kaggle-icecube-neutrinos.

[2] https://www.kaggle.com/code/dipamc77/3rd-place-attention-xgboost-ensembler.

[3] https://github.com/karpathy/minGPT.

Table 5 Features used in the gradient boosting model. Along with statistics of the pulse information, and the actual predicted angles from both the classifier and regressor, we included some hand engineered features. Features marked with * indicate custom hand engineered features. The primary motivation for the features is to mitigate the bias of the zenith classifier. We categorized HLC pulses that are on vertically adjacent sensors and close in time as HLC pulse pairs. Then we compute the features based on this categorization, note that we ignore HLC pulses in this categorization if they do not have a corresponding paired pulse. From the regressor predictions, we compute kappa, which is the square root of the inverse norm of the predicted vector

Feature	Description
z_{stats}	Min, max, mean and std z-coordinate
t_{stats}	Min, max, mean and std of absolute values arrival time
q_{stats}	Std and sum of charge
P_{hlc}	Percent of total pulses being HLC
$S_{\text{avg.}}$	Average speed between pulses
P_{pair} *	Percentage of pulses that are HLC pulse pairs
Δ_{12} *	Difference of time and z-coordinate between first HLC pulse pair and second HLC pulse pair
Z_1 *	Z value of first HLC pulse pair
θ	Actual angles predicted by the classifier and regressor
κ	kappa – Proxy for confidence of prediction
S_A	Softmax of azimuth classifier predictions
S_Z	Softmax of zenith classifier predictions
$\Delta\theta$	Angular distance between all direction predictions

regression head, which predicts a 3D vector corresponding to the angle of the event. Finally, the predictions of both the regression and classification heads concatenated with a set of hand-engineered features, and passed to an Gradient boosting classifier, which is essentially an ensemble of decision trees.

For the transformers, each layer contains 8 attention heads, with an embedding size of 512 (64 per head). The transformer layers form the backbone of the model, producing an embedding for each pulse in the sequence. These embeddings are then averaged to make a single embedding per sequence. This embedded sequence is passed to an MLP with hidden dimension 12288 and output dimension 3072, which we call the neck. The neck for the classification head and regression head are separate. The neck embeddings are passed to the final layers for classification and regression.

The total number of trainable parameters of the 18 layer classification model is 72 M. The transformer backbone has 57 M parameters and the neck has 15 M parameters. The classification head is comprised of two independent fully connected layers of 128 values for azimuth and 142 (128 + 14) values for zenith. Note that extra 14 zenith values are due to the specific design of our loss function, which we describe in the training details subsection below. The regression head is comprised of a single fully connected layer of 3 values, for predicting the 3D vector corresponding to the azimuth and zenith. Additional to the averaged embeddings of the sequence from the transformer backbone, the regression head gets the outputs of the prediction head as inputs, for a total of 782 $(= 512 + 128 + 128 + 14)$ inputs.

To combine the predictions of classification and regression heads, we use a gradient boosting model, specifically scikit-learn's HistGradientBoostingClassifier [40]. The boosting model receives hand engineered features built from the predictions and the event data, and does a binary classification of whether to use the regression or classification head for a given event. These engineered features are shown in Table 5.

We noticed that a significant boost in performance is also obtained due to the feature of first pulse absolute time. Note that the transformers are trained only with relative time information, setting the first pulse time to 0. We did not experiment with absolute time values as inputs to the transformers, but results of other participants indicates that using absolute time gives a significant improvement.

3.3.5 *Ensembling*

Our final solution was an ensemble of three copies of the same architecture, which is a full instance of the model shown in Fig. 6. One of the models is a 15 layer transformer, and the other two are 18 layer transformers. The 18 layer transformers had weights taken from the same training run, where one was taken when 90 percent of the training was complete, and the other when the the training was fully completed, hence their predictions had high agreement. The final predictions of each model are combined using the HistGradientBoostingClassifier, note that this is separate from the gradient boosting classifier used for combining classification and regression heads.

Each of the three ensemble members independently produce the azimuth and zenith angles for each event. The predictions, along with all the hand engineered features of each event described in the feature engineering subsection above, are passed into the classifier, which only decides which of the three predictions to use for each event. We tried other ensemble methods such as averaging the predictions, but found the gradient boosting model to be most effective.

3.3.6 Training procedure

The training of the 18 layer backbone transformer and classifier model was done on a single NVIDIA RTX4080 GPU over approximately 5 days. The 15 layer model takes about 10 percent less time to train than the 18 layer. For efficiency, we train the transformer and the classifier separately, which takes nearly all the training time. We also perform fine tuning of the trained model on longer sequences, which takes an additional 4 h. The regression head is trained after the classification model training is complete, and the backbone is kept frozen, training the regression head takes 6 h on the same hardware. The gradient boosting model takes only a few minutes to train. The training procedure can be broken into a few stages, outlined below.

- **Stage 1**: In the first stage, we train the transformer from scratch for 4 epochs on 650 of the 660 batches of data provided by the organizers. We train for a maximum sequence length of 256, this was a practical choice due to the constraints of our hardware. We use the softmax cross-entropy loss with a custom smoothing operation for the classification loss. Unlike the standard softmax cross entropy setting, our classes are adjacent in angular space, and hence are not independent. To address this, we apply smoothing using a 1D Gaussian kernel for the local bins. The Gaussian kernel is of length 15 and sigma 3. For azimuth, since the space forms a closed loop, the smoothing operation also wraps around. For zenith, we pad the space with extra values to allow the Gaussian kernel to smooth beyond the boundary angles. At inference time, any predictions beyond the zenith boundary limits are clipped. The learning rate scheduler, weight decay and optimizer play an important role in the final performance. For classification models, we tune these parameters on smaller models and scale down the learning rate for larger models. We use a maximum learning rate of 1×10^4 and a weight decay of 1×10^5, with a OneCycle learning rate schedule, the final decayed learning rate is $2g \times 10^6$. We use AdamW optimizer for all the models. We keep a batch size of 384 for the sequence length of 256, and use gradient accumulation where needed, to fit out VRAM constraints. We also do gradient clipping with norm of 0.5 to stabilize training.
- **Stage 1 Fine-tuning**: The classification models are fine-tuned for longer sequences with 20 data batches for 1 epoch. We increase the maximum sequence length of events from 256 to 3072 in this stage, and only train on the events that are higher than 256 in length. During inference, we only use the weights of the fine-tuned model for the sequences longer than 256. Since the subset of data above 3072 is quite small, the total number of events that was used for fine-tuning is small, and the training is completed in 4 h.
- **Stage 2**: After completing the classifier training, we run inference on 100 data batches to save the outputs. Regression model is trained on these 100 data batches for 35 epochs. Since the regression head was trained with the backbone frozen, it is significantly faster to cache the outputs of the backbone and directly train the regression head. Caching outputs training the regression head takes 6 h. For the regression head, we use the von Mises–Fisher loss shown in Eq. (2). In our experiments, we found it to be unstable when training the full transformer backbone. Hence we opted to the freeze the backbone after training of the classification head, and only train the regression neck and head. The vMF also allows to use the kappa feature as a proxy for prediction confidence. Kappa is analogous to the inverse of the norm of the predicted vector. The choice of hyperparameters didn't affect performance much for the regression models. We keep the same hyperparameters as the classification models, except the batch size which was increased to 1024.
- **Stage 3**: In the final stage the gradient boosting classifier is trained with the outputs of both the regressor and classifier. We save predictions of 2 data batches from both models, and train the HISTGRADIENTBOOSTINGCLASSIFIER with the labels set to the predictions of whichever model was closer to the ground truth for every event. Our primary motivation to add this stage was the observation that the distribution of predictions for zenith and azimuth had unwanted structure for both the classifier and regressor. Through experimentation, we found that these predictions were independently not close to the ground truth, but when combined using the HISTGRADIENTBOOSTINGCLASSIFIER in this stage, the performance was significantly improved.

In addition, we employ several training optimizations to reduce the training time. We used Flash Attention v1 [53] during transformer training, which reduced the memory requirement and training time. The memory reduction allowed usage of longer sequence length for fine tuning. We train the transformer in float16 precision. Approximately halfway through the training, we switch to float32 precision for stability. We also used sequence bucketing by sorting the dataset by sequence length and packing mini-batches by sequence length, the longest sequence in a batch will determine the padding for the rest of the batch.

4 Comparison of solutions

The three solutions were evaluated on a sample of nearly 1 million neutrino events of all flavours and interaction types

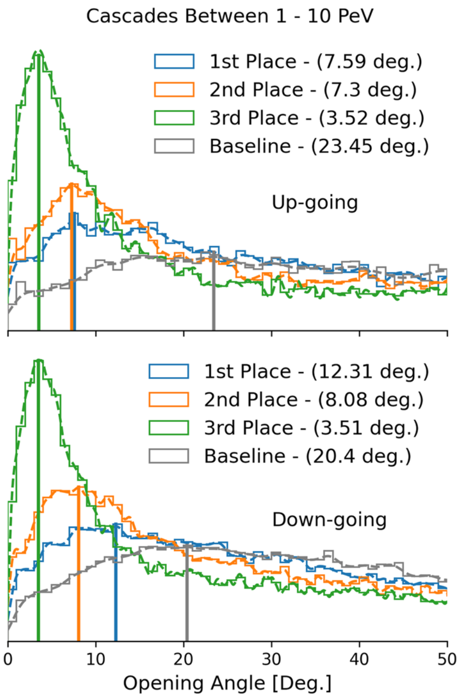

Fig. 7 Example of KDEs constructed for each solution on cascade events between 1 and 10 PeV. Estimated primary mode is denoted with the vertical lines. Top: Up-going cascade events. Bottom: Down-going cascade events

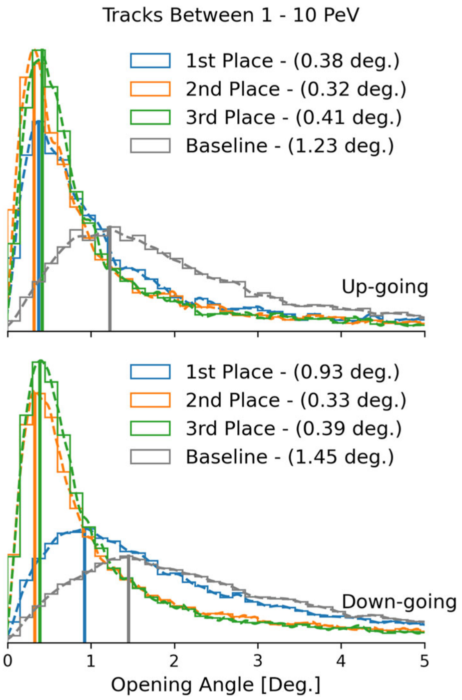

Fig. 8 Example of KDEs constructed for each solution on track events between 1 and 10 PeV. Estimated primary mode is denoted with the vertical lines. Top: Up-going track events. Bottom: Down-going track events

that originates from the same simulation used for the competition, but is sub-sampled such that it includes more events at the lower and higher energy range to facilitate comparisons there. Since this event sample, like the training data in the competition, contains many events that consist of only noise pulses, or events that are highly contaminated with atmospheric muons, the distribution of opening angles between reconstructed and true neutrino directions are relatively wide (see Figs. 7 and 8). In order to report numbers that correspond to the resolution of the well-reconstructable events, we decided to use the mode of the opening angle distribution instead of other summary statistics like the median or the mean which are strongly affected by the non-reconstructable events that span almost uniformly all opening angles between 0° and 180°. This procedure gives an approximation of the expected resolution on analysis level neutrino samples, where low quality events that occupy the wide tails have been excluded.

The opening angle distribution is binned in true neutrino energy and for each energy bin, a *kernel density estimator* (KDE) is used to estimate the primary mode of the distribution. Examples of such KDEs are also shown in Figs. 7 and 8.

The comparisons of the achieved resolutions as a function of neutrino energy are provided for up- and down-going events separately, and are shown for cascade events in Fig. 9 and tracks in Fig. 10.

The corresponding curves from the baseline model (Sect. 2.2) are also included in all figures to provide a reference.

As seen in Fig. 9, all methods have difficulties reconstructing cascade events up to energies of a few TeV, which is likely due to the composition of the competition dataset that provides only few events in this energy range, and that the signal-to-noise ratio becomes increasingly unfavourable as energy decreases. Beyond a few TeV, the 1st place solution achieves its best of around 5° for up- and down-going cascade events between 10 to 100 TeV but worsens as the energy increases. Similarly, the 2nd place solution reaches around 7° for both regions, but stays below 10° towards the higher energy end. The 3rd place solution performs best across the energy range and reaches an opening angle of less than 5° for the events with highest energies for both up- and down-going events. In comparison, the provided baseline stagnates at opening angles of around 20°, which is a factor 3−4× worse.

In Fig. 10, the resolutions on track events is shown. An additional curve has been added that depicts the median kinematic opening angle between the initial neutrino and the outgoing muon from the interaction, representing the expected

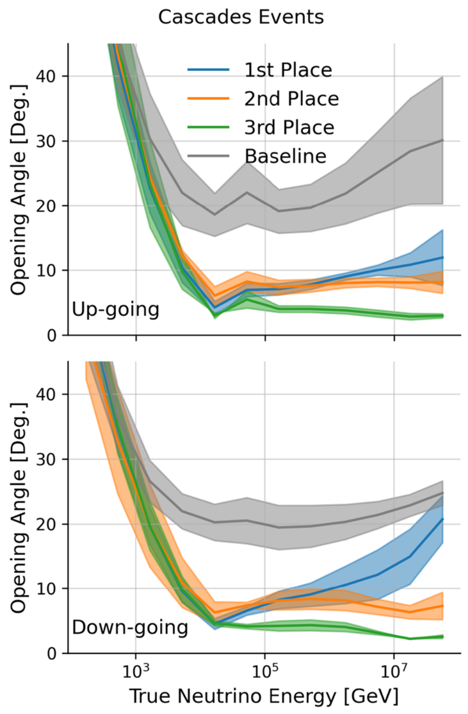

Fig. 9 Estimated primary mode of opening angle vs. neutrino energy for cascade events. Each curve represents a solution. Solid lines depict estimated modes and bands denote 1 sigma uncertainty. Top: Estimated primary mode for up-going cascade events. Bottom: Estimated primary mode for down-going events

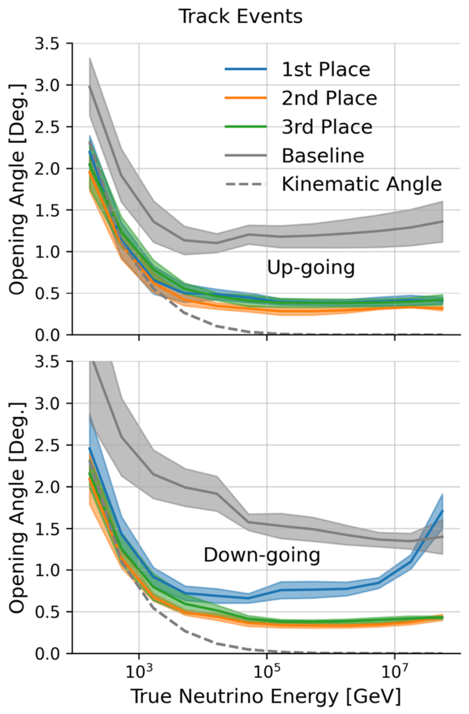

Fig. 10 Estimated primary mode of opening angle vs. neutrino energy for track events. Each curve represents a solution, and the kinematic angle is shown in grey

information limit from only reconstructing the track direction.

Around 1 TeV and lower, all methods follow the kinematic angle closely but flatten beyond a few TeV for both up- and down-going tracks. The 1st place solution settles below half a degree for up-going track events after around 10 TeV but for down-going tracks it reaches a low of around 0.6° at 10 TeV where after the resolution worsens considerably. The 2nd place solution settles under 0.5° after around 5 TeV and reaches a low of around 0.4° for both up- and down-going events. Although the 3rd place solution performs similarly to the 1st place solution on the up-going tracks, it is closer to the 2nd place solution on the down-going tracks. At the same time, the provided baseline is not able to achieve sub-degree resolutions anywhere, and has in general resolution that are worse by a factor $2-3\times$.

From Figs. 9 and 10 it is evident that the three solutions find different ways of minimizing the average opening angle: The 1st place solution appears to resolve up-going tracks significantly better than down-going, the 2nd place solution reconstructs tracks very well, whereas the 3rd place solution appears to resolve cascade events significantly better than the other methods.

It can be noted from the figures discussed in this section, that the 1st place solution may not be the first choice when evaluated based on the mode of the opening angle, but rather the 3rd place for cascade events and the 2nd place for tracks. There is a strong correlation between mean and mode of the opening angle, and therefore all three solutions provide largely improved reconstruction compared to the baseline reference, but the differences seen are expected.

The 1st, 2nd and 3rd solutions used sequence lengths of up to 6000, 768 and 3072 for inference with their transformers, respectively. These choices in hyper-parameters were conditioned on the competition dataset, where less than 2% of the events had a sequence length larger than 800, making the number of events subject to significant sub-sampling small. Because the sequence length increases with neutrino energy, these choices in sequence length might be sub-optimal for the high-energy neutrinos shown in Figs. 9, 10 and further improvements in the high energy range might be possible.

5 Conclusions

While thousands of solutions were submitted during the 3-month “IceCube – Neutrinos in Deep Ice” Kaggle competition, here we focused on the three winning ones. Data processing, model architectures, and training procedures for each of the top three solutions were described in this article, providing insight into these novel ways of applying machine learning to IceCube data. While the three approaches have several aspects in common, they also differ in many ways. The final Kaggle scores of the three solutions were almost on par, but differences can be observed when analyzing and comparing the performance in greater detail. In this work, a comparison differentiating between track and cascade events, as well as up- and down-going events, and shown as a function of the neutrino energy was provided. These comparisons reveal, although all three solutions provided highly accurate reconstructions in general, that there are stark differences. Overall, the 3rd place solution delivers the best cascade reconstruction with a resolution consistently below 5° for neutrino events with energies above 10 TeV. For the track events, the 2nd place solution performs overall best with resolutions that are consistently below the 0.5° mark for neutrino events with energies above 10 TeV.

The current IceCube state-of-the-art reconstruction that specifically targets cascade events reports a median angular resolution of 5°–15° for a comparable energy range as quoted above [16]. Another recently published IceCube reconstruction optimized for track events quotes median opening angles of around 0.2–0.6° [14] in a similar energy range. Although these numbers cannot be compared one-to-one with those reported by our comparisons, as these are computed on different event samples and the median is not necessarily equal to the mode, they indicate that the solutions found in the Kaggle competition are strong contenders and possibly able to outperform existing IceCube approaches, in particular for cascade events.

Another strong suit of the ML-based Kaggle solutions is their high inference speed and their applicability to different event types in IceCube, including noise and other backgrounds. This means that the methods can, in principle, be applied to the online IceCube data stream and provide unprecedented real-time reconstruction quality.

It is also noteworthy that in the top 20 best performing solutions, the vast majority relied on a combination of graph convolutional neural networks and transformers. When comparing the three solutions against the baseline, it is worth noting that all three methods outperform the DynEdge model in this task by quite large margins across the energy range considered in this work. Although the DynEdge baseline provided here was not optimized for this competition or its energy range, comparison of model architectures can provide clues for future iterations of deep learning techniques in IceCube. As suggested in Sect. 3.1, the dynamic re-computation of edges in the graph representation of events originates from point cloud segmentation problems in computer vision and provides a convenient way of applying convolutions on neutrino events of varying size and geometry. However, to project the [n,d]-dimensional input into a [1,k]-dimensional event-level prediction, aggregation of latent predictions is required. In DynEdge, this many-to-one projection is done using simple statistical measures such as averages, minimum and maximum, and this aggregated information is processed by MLP-blocks with relatively few learnable parameters. Two of the winning solutions provide more sophisticated ways of producing this projection; the 3rd place solution uses an elaborate set of engineered features in addition to the averages of its GPT-encoder layers and the 2nd place solution uses a CLS-token as its many-to-one projection. Another point of comparison is the use of transformers, which all three methods relied on. As mentioned in Sect. 3.2, transformers can be viewed as graph convolutional neural networks on fully connected graphs, but where the edges are weighted with attention scores, effectively letting the transformer decide on a per-event level the optimal neighbourhood size and the magnitude of each contribution of the pulses within. In contrast, DynEdge operates on a fixed neighbourhood size for all events (8 for this baseline) and uses shared weights for all neighbourhoods. Another interesting point is the large size of the Kaggle training dataset. To this end, ML based reconstruction algorithms in IceCube were trained on much smaller datasets containing only few million events, which has left large-scale training unexplored. Considering the role of transformers in NLP and other fields with very large datasets, it is not surprising to see such methods also scale well to larger training data sets for IceCube. However, a clear shortcoming of transformer-based solutions is their quadratic scaling with sequence length in terms of memory requirements and computation speed, making it intractable to train on events with very long sequences of pulses, which often are the events of highest interest for certain physics analyses within IceCube. While the winning solutions can be evaluated beyond the sequence lengths they were trained on, it remains a topic for further study to characterize the accuracy of such methods when evaluated on sequences with lengths many times larger than what they were trained on. This question is an active area of research in the deep learning community, and recent developments, such as [54], offer the promising ability to scale to very large sequence lengths.

While further studies are needed from the IceCube collaboration that directly compare to existing IceCube algorithms on an equal footing, we were able to demonstrate the performance of the competition winners on the Kaggle dataset. Our resolution figures show an exciting potential of these new methods.

Acknowledgements This project was supported by the IceCube collaboration, kaggle.com and the Kaggle competition research grants program, the Munich Data Science Institute (MDSI), the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2094 – 390783311, the Sonderforschungsbereich (Collaborative Research Center) SFB1258 'Neutrinos and Dark Matter in Astro- and Particle Physics', and the PUNCH4-NFDI consortium fund "NFDI 39/1".

Author contributions T.I. is the author of the 1st place solution and detailed Sect. 3.1, H.B. and M.S. are the authors of the 2nd place solution and detailed Sect. 3.2, D.C. is the author of the 3rd place solution and detailed Sect. 3.3. R.Ø. is the author of the DynEdge baseline. P.E. and R.Ø. contributed all remaining text and edited the manuscript. R.Ø. generated the model comparisons including the handling of all necessary data. P.E. is the main organizer of the original kaggle competition. All authors reviewed and discussed the full manuscript.

Data Availability Statement This manuscript has associated data in a data repository. [Authors' comment: The datasets used for the current study are publically available in the [kaggle competition] repository, [https://www.kaggle.com/competitions/icecube-neutrinos-in-deep-ice/data.]

Code Availability Statement This manuscript has associated code/software in a data repository. [Author's comment: The code/software of the 2nd place solution is available in the following [github] repository, [https://github.com/DrHB/icecube-2nd-place], and of the 3rd place in the following [github] repository, [https://github.com/dipamc/kaggle-icecube-neutrinos]. The code/software of the 1st place solution is not publically available but can be provided upon reasonable request.]

Funded by SCOAP^3.

References

1. R. Abbasi et al., The IceCube data acquisition system: signal capture, digitization, and timestamping. Nucl. Instrum. Meth. A **601**, 294–316 (2009). https://doi.org/10.1016/j.nima.2009.01.001. arXiv:0810.4930
2. M.G. Aartsen et al., Observation of high-energy astrophysical neutrinos in three years of IceCube data. Phys. Rev. Lett. **113**, 101101 (2014). https://doi.org/10.1103/PhysRevLett.113.101101. arXiv:1405.5303 [astro-ph.HE]
3. M.G. Aartsen et al., Neutrino emission from the direction of the blazar TXS 0506+056 prior to the IceCube-170922A alert. Science **361**(6398), 147–151 (2018). arXiv:1807.08794 [astro-ph.HE]
4. M.G. Aartsen, M. Ackermann, J. Adams et al., Measurement of atmospheric tau neutrino appearance with IceCube DeepCore. Phys. Rev. D **99**, 032007 (2019). https://doi.org/10.1103/PhysRevD.99.032007
5. R. Abbasi et al., Measurement of atmospheric neutrino mixing with improved IceCube DeepCore calibration and data processing. Phys. Rev. D **108**(1), 012014 (2023). https://doi.org/10.1103/PhysRevD.108.012014. arXiv:2304.12236 [hep-ex]
6. M.G. Aartsen et al., Detection of a particle shower at the Glashow resonance with IceCube. Nature **591**(7849), 220–224 (2021) [Erratum: Nature 592, E11 (2021)]. https://doi.org/10.1038/s41586-021-03256-1. arXiv:2110.15051 [hep-ex]
7. P. Eller, in Machine Learning for Astrophysics, Astrophysics and Space Science Proceedings, ch. Event Reconstruction for Neutrino Telescopes (2023)
8. M. Wellons, Robust Statistics in IceCube Initial Muon Reconstruction, in International Cosmic Ray Conference, ser. International Cosmic Ray Conference, vol. 33, p. 3414 (2013)
9. J. Ahrens et al., Muon track reconstruction and data selection techniques in AMANDA. Nucl. Instrum. Meth. A **524**, 169–194 (2004). https://doi.org/10.1016/j.nima.2004.01.065. arXiv:astro-ph/0407044
10. J. Aguilar, I. Al Samarai, A. Albert et al., A fast algorithm for muon track reconstruction and its application to the ANTARES neutrino telescope. Astropart. Phys. **34**(9), 652–662 (2011), ISSN: 0927-6505. https://doi.org/10.1016/j.astropartphys.2011.01.003. https://www.sciencedirect.com/science/article/pii/S0927650511000053
11. M. Aartsen, M. Ackermann, J. Adams et al., The IceCube realtime alert system. Astropart. Phys. **92** (2016). https://doi.org/10.1016/j.astropartphys.2017.05.002
12. M.G. Aartsen, R. Abbasi, M. Ackermann et al., Energy reconstruction methods in the IceCube neutrino telescope. J. Instrum. **9**(03), P03009 (2014). https://doi.org/10.1088/1748-0221/9/03/P03009
13. R. Abbasi, M. Ackermann, J. Adams et al., Low energy event reconstruction in IceCube DeepCore. Eur. Phys. J. C **82**(9), 807 (2022). https://doi.org/10.1140/epjc/s10052-022-10721-2
14. R. Abbasi et al., A muon-track reconstruction exploiting stochastic losses for large-scale Cherenkov detectors. JINST **16**(08), P08034 (2021). https://doi.org/10.1088/1748-0221/16/08/P08034. arXiv:2103.16931 [hep-ex]
15. R. Abbasi, M. Ackermann, J. Adams et al., Evidence for neutrino emission from the nearby active galaxy NGC 1068. Science **378**(6619), 538–543 (2022). https://doi.org/10.1126/science.abg3395
16. R. Abbasi, M. Ackermann, J. Adams et al., A convolutional neural network based cascade reconstruction for the IceCube Neutrino Observatory. J. Instrum. **16**(07), P07041 (2021). https://doi.org/10.1088/1748-0221/16/07/P07041
17. R. Abbasi et al., Evidence for neutrino emission from the nearby active galaxy NGC 1068. Science **378**(6619), 538–543 (2022). https://doi.org/10.1126/science.abg3395
18. J. Micallef and on behalf of the IceCube collaboration, Using convolutional neural networks to reconstruct energy of gev scale icecube neutrinos. J. Instrum. **16**(09), C09019 (2021) [Online]. https://doi.org/10.1088/1748-0221/16/09/C09019
19. R. Abbasi, M. Ackermann, J. Adams et al., Graph neural networks for low-energy event classification and reconstruction in icecube. J. Instrum. **17**(11), P11003 (2022) [Online]. https://doi.org/10.1088/1748-0221/17/11/P11003
20. P. Eller, K. DeHolton, J. Weldert et al., Sensitivity of the IceCube upgrade to atmospheric neutrino oscillations, p. 1036 (2023). https://doi.org/10.22323/1.444.1036
21. P. Eller, A.T. Fienberg, J. Weldert, G. Wendel, S. Böser, D. Cowen, A flexible event reconstruction based on machine learning and likelihood principles. Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrom. Detect. Assoc. Equip. **1048**, 168011 (2023) ISSN: 0168-9002. https://doi.org/10.1016/j.nima.2023.168011 [Online]. https://www.sciencedirect.com/science/article/pii/S0168900223000013
22. M. Huennefeld et al., Combining maximum-likelihood with deep learning for event reconstruction in IceCube. PoS **ICRC2021**, 1065 (2021). arXiv:2107.12110 [astro-ph.HE]
23. I. Collaboration*†, R. Abbasi, M. Ackermann et al., Observation of high-energy neutrinos from the galactic plane. Science **380**(6652), 1338–1343 (2023). https://doi.org/10.1126/science.adc9818
24. A. Chow, L. Heinrich, P. Eller, R. Ørsøe, S. Dane, IceCube - Neutrinos in Deep Ice (2023) [Online]. https://kaggle.com/competitions/icecube-neutrinos-in-deep-ice

25. P. Eller et al., Public Kaggle Competition "IceCube – Neutrinos in Deep ice". PoS **ICRC2023**, 1609 (2023). https://doi.org/10.22323/1.444.1609
26. R. Abbasi, M. Ackermann, J. Adams et al., The IceCube data acquisition system: signal capture, digitization, and timestamping. Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrom. Detect. Assoc. Equip. **601**(3), 294–316 (2009), ISSN: 0168-9002. https://doi.org/10.1016/j.nima.2009.01.001. https://www.sciencedirect.com/science/article/pii/S0168900209000084
27. M. Aartsen, M. Ackermann, J. Adams et al., Efficient propagation of systematic uncertainties from calibration to analysis with the SnowStorm method in IceCube. J. Cosmol. Astropart. Phys. **2019**(10), 048 (2019). https://doi.org/10.1088/1475-7516/2019/10/048
28. R. Abbasi, Y. Abdou, T. Abu-Zayyad et al., The design and performance of IceCube DeepCore. Astropart. Phys. **35**(10), 615–624 (2012), ISSN: 0927-6505. https://doi.org/10.1016/j.astropartphys.2012.01.004. https://www.sciencedirect.com/science/article/pii/S0927650512000254
29. J.L. Kelley and I. Collaboration, Event triggering in the IceCube data acquisition system. AIP Conf. Proc. **1630**(1), 154–157 (2014), ISSN: 0094-243X. eprint: https://pubs.aip.org/aip/acp/article-pdf/1630/1/154/12124858/154_1_online.pdf [Online]. https://doi.org/10.1063/1.4902795
30. M. Aartsen, M. Ackermann, J. Adams et al., Efficient propagation of systematic uncertainties from calibration to analysis with the snowstorm method in icecube. J. Cosmol. Astropart. Phys. **2019**(10), 048 (2019). https://doi.org/10.1088/1475-7516/2019/10/048
31. A. Søgaard, R.F. Ørsøe, L. Bozianu et al., Graphnet: graph neural networks for neutrino telescope event reconstruction (2022). arXiv:2210.12194 [astro-ph.IM]
32. Y. Wang, Y. Sun, Z. Liu, S.E. Sarma, M.M. Bronstein, Dynamic graph CNN for learning on point clouds. ACM Trans. Graph. **38**. https://doi.org/10.1145/3326362
33. H. Pei, B. Wei, K.C.-C. Chang, Y. Lei, B. Yang, Geom-gcn: geometric graph convolutional networks [Online]. https://openreview.net/forum?id=S1e2agrFvS
34. D.P. Kingma, J. Ba, Adam: a method for stochastic optimization (2014). arXiv:1412.6980 [cs.LG]
35. P. Micikevicius, S. Narang, J. Alben et al., Mixed precision training. In: *International Conference on Learning Representations* (2018) [Online]. https://openreview.net/forum?id=r1gs9JgRZ
36. V. Khomenko, O. Shyshkov, O. Radyvonenko, K. Bokhan, Accelerating recurrent neural network training using sequence bucketing and multi-GPU data parallelization. In: *IEEE First International Conference on Data Stream Mining and Processing (DSMP)*, vol. 2016, pp. 100–103 (2016). https://doi.org/10.1109/DSMP.2016.7583516
37. A. Vaswani, N. Shazeer, N. Parmar et al., Attention is all you need. CoRR, vol. abs/1706.03762 (2017). arXiv:1706.03762
38. M.S. Habib Bukhari, IceCube – Neutrinos in Deep Ice (2023) [Online]. https://github.com/DrHB/icecube-2nd-place
39. M.G. Aartsen et al., Measurement of South Pole ice transparency with the IceCube LED calibration system. Nucl. Instrum. Meth. A **711**, 73–89 (2013). https://doi.org/10.1016/j.nima.2013.01.054. arXiv:1301.5361 [astro-ph.IM]
40. F. Pedregosa, G. Varoquaux, A. Gramfort et al., Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
41. P. Shaw, J. Uszkoreit, A. Vaswani, Self-attention with relative position representations. CoRR, vol. abs/1803.02155 (2018) [Online]
42. Z. Peng, L. Dong, H. Bao, Q. Ye, F. Wei, Beit v2: masked image modeling with vector-quantized visual tokenizers (2022). arXiv:2208.06366 [cs.CV]
43. J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding (2019). arXiv:1810.04805 [cs.CL]
44. H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, H. Jégou, Training data-efficient image transformers and distillation through attention (2021). arXiv:2012.12877 [cs.CV]
45. A. Dosovitskiy, L. Beyer, A. Kolesnikov et al., An image is worth 16×16 words: transformers for image recognition at scale (2021). arXiv:2010.11929
46. T.R. Scott, A.C. Gallagher, M.C. Mozer, Von mises-fisher loss: an exploration of embedding geometries for supervised learning. CoRR, vol. abs/2103.15718 (2021). arXiv:2103.15718
47. A. Paszke, S. Gross, S. Chintala et al., Automatic differentiation in PyTorch (2017)
48. I. Loshchilov, F. Hutter, Fixing weight decay regularization in Adam. CoRR, vol. abs/1711.05101 (2017). arXiv:1711.05101
49. J. Lamy-Poirier, Layered gradient accumulation and modular pipeline parallelism: fast and efficient training of large language models (2021). arXiv:2106.02679
50. P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, A.G. Wilson, Averaging weights leads to wider optima and better generalization (2019). arXiv:1803.05407
51. J. Hoffmann, S. Borgeaud, A. Mensch et al., An empirical analysis of compute-optimal large language model training. Adv. Neural Inf. Process. Syst. **35**, 30016–30030 (2022)
52. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, Language models are unsupervised multitask learners (2019)
53. T. Dao, D.Y. Fu, S. Ermon, A. Rudra, C. Ré, FlashAttention: fast and memory-efficient exact attention with IO-awareness. Adv. Neural Inf. Process. Syst. (2022)
54. T.D. Albert Gu, Mamba: linear-time sequence modeling with selective state spaces (2024) [Online]. https://openreview.net/forum?id=AL1fq05o7H