PAPER

# Two Watts is all you need: enabling in-detector real-time machine learning for neutrino telescopes via edge computing

Journal of Cosmology and Astroparticle Physics
An IOP and SISSA journal

Received: *March 9, 2024*
Accepted: *May 17, 2024*
Published: *June 12, 2024*

# Two Watts is all you need: enabling in-detector real-time machine learning for neutrino telescopes via edge computing

**Miaochen Jin**[a], **Yushi Hu**[b] **and C.A. Argüelles**[a]

[a] *Department of Physics & Laboratory for Particle Physics and Cosmology, Harvard University, Cambridge, MA 02138, U.S.A.*

[b] *Department of Electrical and Computer Engineering & Natural Language Processing Group, University of Washington, Seattle, WA 98195, U.S.A.*

*E-mail:* miaochenjin@g.harvard.edu, yushihu@uw.edu, carguelles@fas.harvard.edu

Abstract: The use of machine learning techniques has significantly increased the physics discovery potential of neutrino telescopes. In the upcoming years, we are expecting upgrades of currently existing detectors and new telescopes with novel experimental hardware, yielding more statistics as well as more complicated data signals. This calls for an upgrade on the software side needed to handle this more complex data in a more efficient way. Specifically, we seek low power and fast software methods to achieve real-time signal processing, where current machine learning methods are too expensive to be deployed in the resource-constrained regions where these experiments are located. We present the first attempt at and a proof-of-concept for enabling machine learning methods to be deployed in-detector for water/ice neutrino telescopes via quantization and deployment on Google Edge Tensor Processing Units (TPUs). We design a recursive neural network with a residual convolutional embedding and adapt a quantization process to deploy the algorithm on a Google Edge TPU. This algorithm can achieve similar reconstruction accuracy compared with traditional GPU-based machine learning solutions while requiring the same amount of power compared with CPU-based regression solutions, combining the high accuracy and low power advantages and enabling real-time in-detector machine learning in even the most power-restricted environments.

Keywords: Machine learning , neutrino detectors, neutrino experiments

ArXiv ePrint: 2311.04983


https://doi.org/10.1088/1475-7516/2024/06/026

JCAP06(2024)026

## Contents



JCAP06(2024)026

## 1 Introduction

Neutrino telescopes are large-scale neutrino detectors built in naturally occurring media such as glaciers, mountains, lakes, and seas or even deployed in outer space. They aim to detect high-energy neutrinos produced in the collision of high-energy hadrons with ambient gas or radiation in astrophysical sources. The steeply falling ($E^{-2.5}$ [1, 2]) flux of these neutrinos, together with the smallness of the neutrino-nucleon cross-section [3], makes the detection of these neutrinos challenging. Approximately ten years ago, the IceCube Neutrino Observatory, a gigaton-ice-Cherenkov detector in Antarctica [4], discovered a diffuse astrophysical flux; see [5] for a historical review.

More recently, driven by reconstruction and event selection improvements made possible by the use of machine learning techniques, the IceCube collaboration has announced the detection of the first steady-state extragalactic neutrino source, NGC 1068 [6], and the observation of our galaxy in neutrinos [7]. These successes follow from prior searches for astrophysical neutrino sources, which, among other things, found evidence for emission from the TXS 0506+056 Blazar [8] in IceCube and hinted at emission from our galaxy by ANTARES [9], a smaller neutrino telescope which was deployed in the Mediterranean sea.

Detecting and studying these neutrinos can provide unique information about the cosmic accelerators that produce them and potentially resolve the 100-year-old problem of the origin of cosmic rays. Additionally, these neutrinos probe previously uncharted energy and distance regimes and thus constitute a unique probe of new physics, see [10], e.g., refs. [11–26] for specific examples. Furthermore, the field of neutrino astrophysics is growing with two optical neutrino telescopes under construction: Baikal-GVD [27] in Russia and KM3NeT [28] in the Mediterranean Sea. These are expected to be followed by next-generation optical neutrino telescopes such as IceCube-Gen2 in Antarctica [29], TRIDENT and HUNT in China [30, 31], and P-ONE [32] in Canada; as well as a breath of Earth-skimming neutrino detectors which focus on finding evidence for tau neutrinos using Cherenkov light, TRINITY [33], particle showers, TAMBO [34], or radio, Grand [35].

These experiments are expected to produce a large amount of data: IceCube currently produces data at approximately 3 kHz with similar data rates expected at KM3NeT and Baikal GVD. In the meantime, high-accuracy algorithms process these data at a much lower rate, such that to achieve real-time processing, we need much more efficient algorithms. See ref. [36] for a recent ML proposal to tackle these large rates. The large data rate is expected to increase in next-generation experiments significantly, e.g., IceCube-Gen2 is expected to have eight times the data rate of IceCube, while TRIDENT will be thirty times larger. For the detectors to tell interesting events apart from the backgrounds or to send real-time warnings on rare events, the need for real-time triggering and reconstruction algorithms becomes more prevalent. Ref. [36] provides a solution based on sparse convolutions. However, the problem of neutrino event reconstruction is not only that of large backgrounds; these algorithms also need to operate in resource-constraint environments. For example, the IceCube detector Main Array operates Digital Optical Modules (DOMs) at 5.7 watts per module [4]; additionally, other experiments, such as TAMBO or Grand, envision solar-power detection units that generate limited power. Under these restrictions, machine learning algorithms whose efficiency benefits from GPU parallelization cannot be deployed, and current real-time triggering algorithms are CPU-based fast regression that fit under the power restriction but are much less accurate: they serve only as a preliminary selection and more accurate reconstructions are performed off-line.

Edge computing refers to low-latency computing solutions that happen close to the source of the data, for example in real-time data processing situations. In 2018, Google announced an edge computing micro-architecture: the Edge Tensor Processing Unit (Edge TPU) Dev Board [37], which is a portable version of the TPU architecture that was developed and announced earlier [38]. Inheriting the *Matrix Multiplication Units* that enable fast machine learning inference from the TPU, this edge computing version runs inference on reduced size models, consuming only 3 watts of power in total for the Dev Board, with only 2 watts required by the TPU chip itself. The Edge TPU has since then enabled machine-learning-inference capability on many mobile computing devices and is under further development and optimization even today, see [39] and [40] for relevant discussions. With a versatile architecture that allows for easy interfacing, compiling, and deployments, and boosted with a software backend TensorFlow [41], the Edge TPU becomes a suitable edge computing solution to achieving real-time in-detector machine learning inference for neutrino telescopes.



JCAP06(2024)026

In this article, we introduce the first attempt at accelerating neutrino event reconstruction on edge computing devices using a recursive neural network (RNN) method with a residual convolutional input encoding, which enables an extremely low-power-consuming alternative to GPU-based machine learning algorithms. We will discuss the specific data pre-processing, network design, and quantization procedure that makes possible the deployment of this algorithm on the Edge TPU. In section 2, we introduce the detector geometries and data simulation used in this work; in section 3, we briefly discuss the various hardware architectures we test our algorithm on and their reported power consumption specifications and layout, specifically the constraints of the Edge TPU hardware, serving as the motivation of discussions of the software methods; in section 4, we lay out in detail the data pre-processing and network architecture in the context of edge-computing hardware limitations, and explain our methods and solutions, including a fine-tuning training procedure; in section 5, we evaluate the accuracy and power performance of our approach; finally, in section 6, we discuss the various future directions this work opens up, and encourage further exploration in the direction of low-power computing in neutrino detectors.

## 2 Detector and data simulation

In this work, we will test the performance and accuracy of our network on two example detectors of the same geometry deployed in water and in ice, to which we assigned the names of WaterHex and IceHex respectively, where the hexagonal geometry is inspired by that of IceCube. As with traditional and upcoming optical neutrino telescopes, the optical modules (OM) are arranged in vertical strings. The inter-string distance is set to 100 meters, and the inter-OM distance along a string is set to 17 meters. The arrays of each of the detectors constitute a total of 114 strings, which is similar to the expected KM3NeT final configuration [28], with each string containing 60 OMs, summing to a total of 6840 OMs.

We use the open-source neutrino event simulation tool `Prometheus` [42] to generate the neutrino events and simulate the corresponding detector responses, where the default medium settings are employed. We generate all-sky muon neutrinos with $\cos\theta_{\text{True}} \in [-1, 1]$ and $E_\nu \in [10^3, 10^6]\,\text{GeV}$ with a power-law energy distribution that has a spectral index of $-1$. In figure 1, we show the energy and cosine zenith distribution of the simulated events at different levels. In each energy bin, we plot the fraction of all generated events ("generation level") that reach "light level" and "trigger level" respectively, where the levels are to be understood as follows:

- *Generation level*: includes all events generated by `Prometheus`. This is uniform in both energy and cosine of the zenith angle. In generating the events we use the volume injection option, where the neutrinos are injected such that they pass through a uniform column depth sphere centered at the detector. More specifically, this spherical volume is defined such that column depth is uniform in all directions, with a cutoff at the top of the atmosphere. See [42] for detailed documentation of this option setting.

- *Light level*: includes the events that contain at least one photon deposition in the array of OMs. The energy distribution as shown in figure 1 is because the injection of higher



JCAP06(2024)026

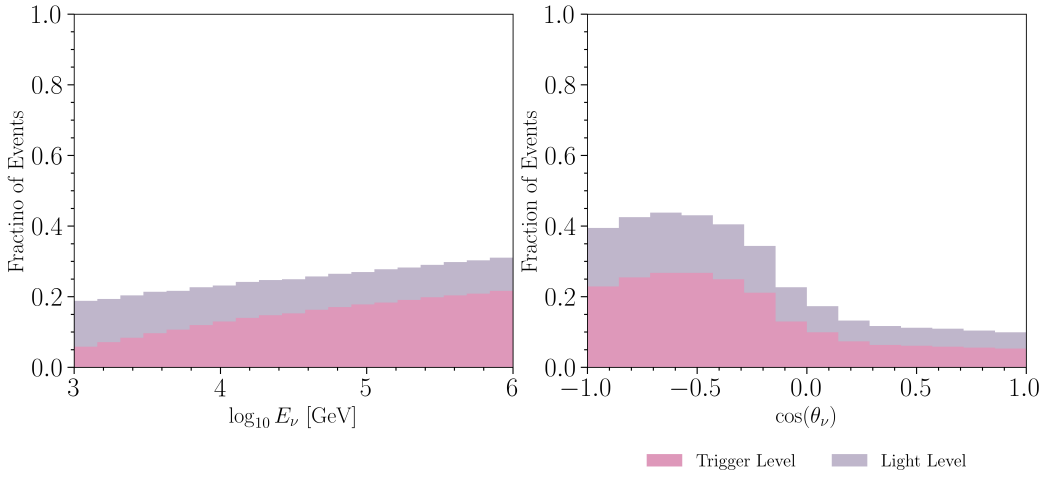


**Figure 1.** *Energy and Zenith angle distribution of data set at various selection levels.* Plotted for simulated WaterHex dataset with the IceHex data set being qualitatively similar.

energy neutrinos leads to the production of higher energy muons; these muons traverse longer distances in matter and are more likely to reach the detector region and emit more Cherenkov photons in the detector region thereon. The zenith distribution is skewed significantly towards the down-going direction. As discussed in the previous section, while `Prometheus` injection of neutrinos is configured such that the column depth of neutrino propagation before reaching the detector volume is uniform in all directions, the injector does have a hard cutoff at the top of the atmosphere. This implies that down-going neutrinos (those that are injected from the direction of the southern hemisphere atmosphere) traverse a much shorter column depth before reaching the detector compared with the up-going ones coming from the core/mantle of the Earth. Therefore, injected down-going muon neutrinos are more likely to interact and produce muons that deposit light in the detector compared with the up-going counterpart. This effect is boosted by the fact that the PMTs are assumed to face downwards and therefore have a better angular acceptance for events that are down-going, resulting in the final skewed zenith distribution as seen in the figure.

- *Trigger level*: this includes part of the light-level events that pass a trigger defined by a global simultaneous observation of local coincidences, similar to treatments performed in other relevant works [36]. In this work, as a proof of concept for edge computing, we do not include background events (as opposed to muon neutrino signal events) or background noise in muon neutrino events (such as $^{40}K$ in water) in our simulation of photon deposition in the detector. Therefore, in ice and water mediums alike, we relax the trigger criteria to the detection of at least 8 pairs of coincidental photon deposition within a 5 $\mu$s time window. Here, 5 $\mu$s is approximately the maximum time it takes for a muon to traverse the entire detector region, considering also a buffer for photon propagation, and coincidence is observed in OMs located at neighboring strings, at most separated by 2 OMs apart in the vertical direction.



JCAP06(2024)026

| Hardware Architecture | Reported Efficiency | Total Power | ML Accelerator Power |
|---|---|---|---|
| A100 GPU on Lenovo Server | 165 TFLOPS | 2400W | 400W |
| RTX3080 GPU on Alienware Workstation | 29.8 TFLOPS | 1000W | 320W |
| Apple MacBook Pro with M1Pro 16-core GPU | 5.3 TFLOPS | 100W | 15W |
| Google Edge TPU | 4TOPS | 3W | 2W |

**Table 1.** *Summary of the hardware utility data used in the evaluation of the model deployed by this work.* The power consumption specifications listed above are approximate values of peak consumption taken directly from manufacturer specs [37, 43–45] and therefore should be taken as a rough estimate and indication of the scale of the computing system to some extent of accuracy. Even considering this limitation in precision, the comparison listed above still shows significant differences between different classes of hardware architectures.

Using these definitions, for down-going muon neutrinos, approximately 33% deposit any light in the detector OMs, and about 25% passes the trigger selection; for up-going muon neutrinos, on the other hand, only about 10% deposits any light in the detector with 5% passing the trigger selection. At 1 TeV, about 50% of all simulated events deposit light in the detector OMs, and about 20% of all events pass the trigger selection. We simulated 2 million events for the WaterHex detector and 2.5 million events for the IceHex detector, out of which 300787 and 373079 events passed the trigger selection, respectively, both reaching about 15% passing rate. To facilitate cross-comparisons, we randomly choose 300000 events for each detector and split the set of events randomly into 240000 and 60000 as their respective training and validation datasets.

## 3 Hardware setup

### 3.1 Overview of architectures and power consumption

For this work, we evaluate the performance, both accuracy and power efficiency, of our method on four different computing units. Each hardware unit serves as a representative of a class of hardware architectures with different operating powers: server/cloud scale, workstation/desktop scale, laptop scale, and edge computing scale. These architectures span orders of magnitude in maximum power consumption. In table 1, we summarize the relevant specification parameters of the different hardware architectures: specifically, we show the reported Trillion Operations per Second (TOPS) / Trillion Floating Point Operations per Second (TFLOPS), total power, and ML accelerator power consumption. "ML accelerator power" counts only that of the GPU or TPU chip, whereas "total power" includes the consumption of the accompanying CPU and other parts of the cluster, PC, or Edge TPU Dev Board respectively for the hardware architectures.



JCAP06(2024)026

### 3.2 The Google Edge TPU

Specifically of interest to this article is the inference performance on the mobile, low-power machine learning accelerator: the Google Edge TPU. We devote this subsection to discussing the capabilities and limitations of the Edge TPU, which, as shown in table 1, consumes only 2 watts of power, as contrasted to large computing center-scale GPU clusters.

The TPU architecture is capable of such efficient performance thanks to Matrix Multiplication Units (MXU), which are systolic arrays, in place of Arithmetic Logic Units (ALU), which are employed by CPU and GPU architectures. On the one hand, this architecture, while applied on server-level scale, is capable of running at 68× incremental performance per watt compared to GPU-based servers [38]. On the other hand, the Edge TPU architecture runs on integer operations instead of floating point operations and is optimized for low-power, mobile computing, an important requirement for enabling online reconstruction work for neutrino telescopes and other experiments alike, often found in power-limited environments. While the Edge TPU inherits the performance advantages of its server-scale TPU relative, the capabilities come with trade-offs and harsh limitations on the software that can be run on it. While the Edge TPU documentation has an extensive list of requirements, including enabled layer and operation types [46], the two main restrictions that are crucial to the work of interest are as follows:

- *Dimensionality*: all tensors are restricted to $D \leq 3$, and in case where $D > 3$, the extra dimensions can only have length 1. This also implies convolutional layers are restricted to $D \leq 2$, which is very sup-optimal for optical module array type detectors where the data is inherently a 4-dimensional counting array and the time axis. Furthermore, the inability of the Edge TPU to handle high dimensional inputs disables batching, thereby resulting in expensive operations if one attempts to reduce 3D convolution to a batch of 2D convolutions. See section 4.2 for a detailed discussion.

- *Quantization*: for full utilization of the Edge TPU requires `uint8` type accuracy instead of floating point accuracy. This means mapping all input, output, as well as network weights to integers within the range of 0 to 255. As one would expect, this means decreased accuracy for the model, but luckily, benchmark examples show that with proper treatments, quantized networks can also achieve very high accuracy performances [47]. Various studies have also explored quantization methods that minimally reduce accuracy for different algorithms [39, 48, 49]

To overcome the hardships brought about by the limitations, we adapt our architecture design and employ a quantization procedure: these will be explored in section 4.2 and section 4.3 respectively.

## 4 Software methods

### 4.1 Data and network input

At the OM-level, water(ice)-Cherenkov detector data consists of the lowest level PMT waveforms recorded by optical modules in water or ice. Some detectors choose to store



JCAP06(2024)026

and send the entire waveforms to a centralized facility whereas others choose to process the waveforms and store compressed data. In either case, with real-time, in-OM processing, we can access a set of waveforms ($\{q_\alpha(t)\}_\alpha$) corresponding to each OM $\alpha$ located at $(x_\alpha, y_\alpha, z_\alpha)$. For machine learning algorithms to work with these data, at least some extent of pre-processing is needed, the specific method of which differs per ML algorithm and architecture employed. We take IceCube as an example, where the prominent reconstruction algorithm `DNNreco` [50] is a Convolutional Neural Network [51] (CNN) approach, where data is pre-processed to form a 4-dimensional tensor, treated equivalent to an "image." Spatial coordinates are embedded into a 3-dimensional tensor $(x_\alpha, y_\alpha, z_\alpha) \to V^{(i_\alpha, j_\alpha, k_\alpha)}$, while for the time component, the PMT waveforms $q_\alpha(t)$ are extracted to give nine distinct temporal features from the waveform of a DOM across the time of an event that describe the shape of the waveform. See [50] for a detailed discussion. This results in a 4-dimensional input where the time dimension vector. The network primarily sees the data as an image, where on each "pixel", now an OM location, the information of "color channels" is replaced by the discrete time parameters. In this work, we rethink the formulation of this problem, phrasing it primarily as a time series analysis and using Recursive Neural Network (RNN) architectures; this architecture choice will be elaborated upon in section 4.2. We take the detector data after pulse extraction to obtain the individual hit times of photons on OMs, $\{H_i = (t_i, x_i, y_i, z_i)\}_{i=1}^N$, and group them imagining that we are taking snapshots of the event at a fixed timestep. For an event consisting of $N$ hits that spans $\Sigma$ nanoseconds, we break it into $T$ separate frames each with $\sigma = \Sigma/T$ nanoseconds, containing an aggregate of $\{N_t\}_{t=1}^T$ hits with $\sum_t N_t = N$. This results in $T$ distinct three-dimensional arrays $A_t$, with each entry $A_t^{(X,Y,Z)}$ encoding the number of total hits on the corresponding DOM within the time window. Here $X, Y, Z$ are the three dimensions of the OM array, such that the total number of photon hits in the entire detector within any specific time window is $\sum_{(x,y,z)=(1,1,1)}^{(X,Y,Z)} A_t^{(x,y,z)} = N_t$. This data pre-processing results in a network input that resembles "snapshots" of the detector at different times, as shown in figure 2. A more detailed visualization of the simulated detector response and pre-processed data can be found in A.

Another advantage of such a data representation is the discrete nature of all the entries in $A_T^{(X,Y,Z)}$. For the network to be deployed on the TPU, not only the weights but also the network inputs have to be cast as full integers in the range of $[0, 255]$. This input encoding design allows us to adapt the input to type `uint8` without dealing with errors that originate from mapping a continuous distribution to a discrete one to suit the Edge TPU model requirements. A more detailed discussion on the full quantization of the network input will follow in section 4.3.

### 4.2 Recursive network with convolutional embedding

In most currently deployed machine learning-based reconstruction algorithms for neutrino telescopes, CNNs and graph neural networks [52] (GNNs) are employed. Given the data encoding, these architectures are straightforward and intuitive and yield satisfactory results. However, under the restrictions discussed in section 3, especially the restriction on input tensor dimension and allowed types of operations, these architectures cannot be deployed on the Edge TPU device.



JCAP06(2024)026

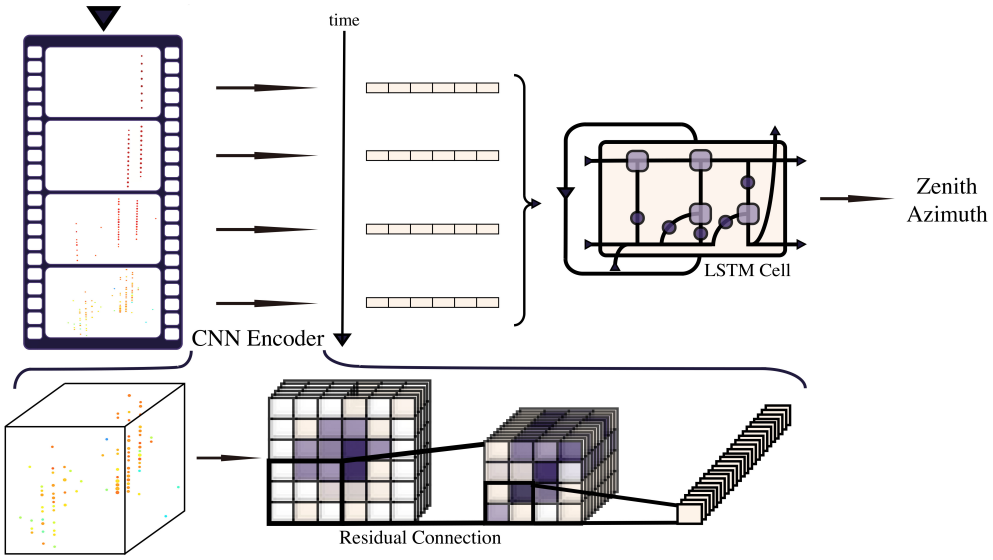


**Figure 2.** *Recursive neural network with convolutional encoding.* After input pre-processing, each time-step of the input data is encoded by a CNN encoder before being processed by the LSTM cell to generate a final prediction of the neutrino zenith and azimuth angles.

In figure 2 we show the network architecture developed in this work. This is a combination of residual convolutional and recursive neural network architectures. The design can be effectively summarized as a Long Short-Term Memory [53] (LSTM) time-series prediction using residual convolution as the input encoder. For each input array at timestep $t$, the CNN encoder transforms the array of image-like data $\{A_t^{(X,Y,Z)}\}_{t=1}^{T}$ into a sequence of vectors $\{L_t\}$ in the latent space, which becomes the input into the hidden layers of the RNN. Typically, for an event that spans 3 $\mu$s, choosing $T = 15$, each time-frame will contain all photon hits in a 20ns time window. The CNN encoder contains one initial convolution layer with five residual convolution blocks, each containing two convolution layers. The LSTM cell contains an LSTM layer and a dense layer that connects to the output. The output is the azimuthal and zenith angles of the incoming neutrino, from which we calculated the error by a dot product with the simulated true neutrino direction. See appendix B for the detailed layout of the network architecture.

It is worth noting that in the CNN encoder, we employed a very peculiar way of treatment that takes apart a 3-dimensional array of OM photons hits $A^{(X,Y,Z)}$ and breaks it apart into $\{A^{(X,Y)}\}_{z=1}^{Z}$, where the $z$ component is treated as channels in the network. This is due to TPU's restriction on vector dimensions and time complexity considerations. On the one hand, it is impossible to apply 3-dimensional convolution to the input data. On the other hand, time complexity without parallelization capability forbids us to apply CNN on the $z$ component in parallel: an alternative way to the channel treatment is applying the same two-dimensional CNN on all vertical slices $\{A_z^{(X,Y)}\}$ to obtain a set of output vectors and concatenate them, followed by another 2D CNN that transforms the resulting 2D array into a 1D vector, wrapping up the encoding process. While for a GPU, parallelization can be applied to the first set of CNN, and the entire process will consume the same time required for only two separate CNN networks, on the TPU architecture, where such parallelization



JCAP06(2024)026

is not allowed, this will take the same time as evaluating $(T+1)$ CNN networks separately. Due to these constraints and considerations, we resorted to the design shown in figure 2. We have tested the channel treatment versus the parallel treatment and found negligible differences in angular reconstruction accuracy on GPU architectures.

### 4.3 Quantization procedure

Quantization for Edge TPU deployment is a necessary step where all network operations are cast into 8-bit full integer operations. This is a daunting process where we face a trade-off between efficiency and accuracy. We use TensorFlow's mobile library to realize the quantization process.

To optimize for unsigned integer type operations, an important step during the conversion from the full precision model to the reduced precision one, the Edge-TPU compatible model must provide the converter with a representative dataset. This is a set of input data, which we take from the training set, that helps the converter decide a reasonable mapping between `float32` and `uint8` by providing information regarding the range and distribution of the input, weights, and output. Therefore, it is important to have a set of inputs with similar distributions and a narrow spread in value to ensure a smooth conversion to the reduced precision format.

We initiate the quantization process by focusing on the inputs. Referring back to our earlier discussions in section 4.1, the detector data is encoded using only integers. Notably, there are key features essential to the quantization process. Firstly, as each entry represents the number of photons deposited within a sufficiently small time frame, only a minute fraction of entries surpass a certain threshold $A_t^{(x,y,z)} \leq 255$. Consequently, implementing a cutoff at 255 does not significantly impact the network's performance. Secondly, in addition to a simple cutoff, we uniformly map the entries of every input tensor to a Gaussian distribution. This approach proves beneficial for the quantization process as it ensures a more evenly distributed dataset.

The input quantization process is summarized as

$$x' = \mathrm{Clip}_{[0,255]}\Big(\texttt{uint8}(F_{(\mu,\sigma)}(x))\Big), \tag{4.1}$$

where $F_{(\mu,\sigma)}$ maps the entire population $\{A_t^{(x,y,s)}\}_{(T,X,Y,Z)}$ to a Gaussian distribution centered at $\mu = 90$ with standard deviation $\sigma = 45$. The choices of $\sigma$ and $\mu$ are selected after several trials such that the data is widely spread enough but minimally exceeding the range $[0, 255]$. Although this pre-process quantization of the data is observed to impact the accuracy performance of the network negatively, the original network that's used for quantization and deployment on the TPU will be trained directly on this processed dataset. In contrast, the same network for GPU-based training and inference will be trained on the original data input; a comparison between the two will be provided in section 5.1.

Ideally, quantization of the network weights will be performed by the converter from the TFLite library [54]; however, stable releases of TensorFlow do not yet support full quantization of networks with multiple subgraphs, such as the network developed by this work. Instead, we divide the quantization process into steps to accomplish a successful conversion. In each



JCAP06(2024)026

stage, we cast additional weights, activations, other layer components, and outputs to an 8-bit unsigned integer. These stages proceed as follows:

- ***Stage 1:*** *quantization of inputs.* To begin with, we have the inputs into the CNN encoder as `uint8`; all weights are `float32`.

- ***Stage 2:*** *quantization of CNN encoder.* We quantize the CNN encoder by providing a representative dataset that contains 1000 input samples. However, after the CNN encoder evaluation, we use the reverse mapping to "fallback" from `uint8` results to `float32` numbers, which then becomes the input to the LSTM. Worth noting is that at this stage, while the input into the LSTM is indeed `float32`, there exists only 256 distinct numbers, as they were mapped back from `uint8`-quantized CNN encoder outputs. At this stage, since the LSTM is unaware of the change of precision of the CNN Encoder, even in the best-case quantization scenario, we would not be able to recover the original reconstruction accuracy fully. This step, which is unnecessary if direct quantization of a multiple subgraph model is allowed, brings about a loss of accuracy that is avoidable with future software developments.

- ***Stage 2.5:*** *re-train the LSTM cell.* At this stage, we perform a fine-tuning re-training of the LSTM cell where the initial weights are directly transferred from the pre-trained original model as a solution to the artificial accuracy loss problem discussed in the previous stage. However, instead of training with the float-fallback CNN encoder outputs, we directly perform training on the quantized output from the CNN encoder, which is of type `uint8`. Upon finishing the re-training, the CNN encoder is fully quantized, and so is the input to the LSTM cell, but the LSTM cell still contains `float32` weights.

- ***Stage 3:*** *quantization of LSTM and dense layers.* At this final stage, we perform the quantization of the LSTM cell, providing a representative dataset that contains 1000 samples of CNN encoder outputs of inputs selected from the training sample. The output is naturally also represented by the network in `uint8`, which falls back to `float32` to provide us with the actual final prediction.

In section 5, we will show the network performance at each stage.

## 5 Results and discussions

### 5.1 Reconstruction accuracy

We first show the accuracy of the network as evaluated on GPU using double-precision floating-point computations, without mapping or imposing the cutoff at 256 for the entries of the input. This serves as a benchmark as it is the optimal reference of the algorithm. In figure 3, we show the median error distribution of the angular, zenith, and azimuthal reconstruction against true neutrino energy for both IceHex and WaterHex detectors at trigger level. We use the same architecture but train on both datasets separately. For the IceHex detector, we see the angular error reaching as low as 4.0°; and for the WaterHex detector, we



JCAP06(2024)026

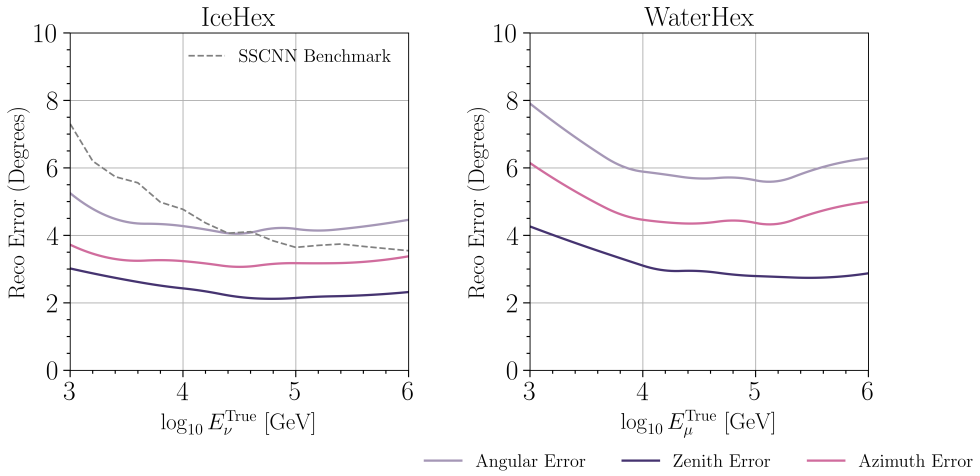


**Figure 3.** *Neutrino angular reconstruction resolution on full accuracy network as a function of the true simulated neutrino energy.* Left panel shows the performance of the network trained on the IceHex detector, right panel shows the performance of the network trained on the WaterHex Detector. The Sparse Submanifold Convolutional Neural Network (SSCNN) [36] angular resolution is shown on the IceHex performance plot as a benchmark.

see the angular error reaching as low as 5.6°. The performance for IceHex and WaterHex are comparable, where we do observe the detector in ice to have events that are easier to reconstruct, due to the uniform choice of 100 meters as the string spacing across the two mediums, whereas photons in water have a smaller absorption length, resulting in Cherenkov photons deposition in fewer, more clustered OMs. Additionally, the rising tail on the high energy end is due to event selection at triggering, where higher energies allow for the more poorly positioned track, cutting only the corner of the detector, for example, depositing fewer photons. On the other hand, as shown in the bottom panel of figure 4, the reconstructed error decreases monotonically with the increment in the number of photons hit depositions in the OMs. While this network is designed with limitations in architecture to enable further acceleration on TPUs, it is comparable in accuracy performance with other machine learning-based reconstruction algorithms for similar, but not completely identical, detectors [36, 50] at the trigger level. The recent high-speed algorithm using sparse submanifold CNN method [36] at the trigger level is labeled as a dashed gray line in figure 3.

### 5.2 Post-quantization accuracy

In this section, we present and discuss the network performance at each stage after the quantization sub-steps described in section 4.3 are applied. We show the energy distribution of angular reconstruction error at various stages in figure 5.

For the IceHex detector, performance is nearly unaffected by the quantization of inputs, keeping the network in floating point precision. At this stage, the median error is 4.6°, with the high energy end of the spectrum reaching a 4.4° median error. Upon quantization of the CNN encoder, applying a float fallback before inputting into the LSTM cell, which is kept at floating point precision, the median angular error reaches 5.0°. Disabling floating point fallback and thereby quantizing the entire CNN encoder as well as the LSTM inputs,



JCAP06(2024)026

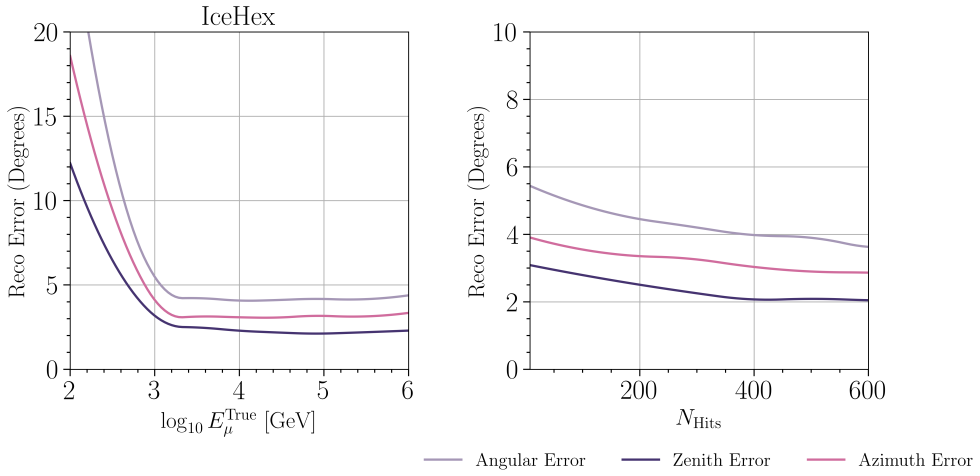


**Figure 4.** *Neutrino angular reconstruction resolution on full accuracy network for IceHex a function of the true simulated lepton energy and number of photon hits.* Left panel shows the error as a function of the muon, right panel shows that as a function of the number of photon hit deposition in OMs.

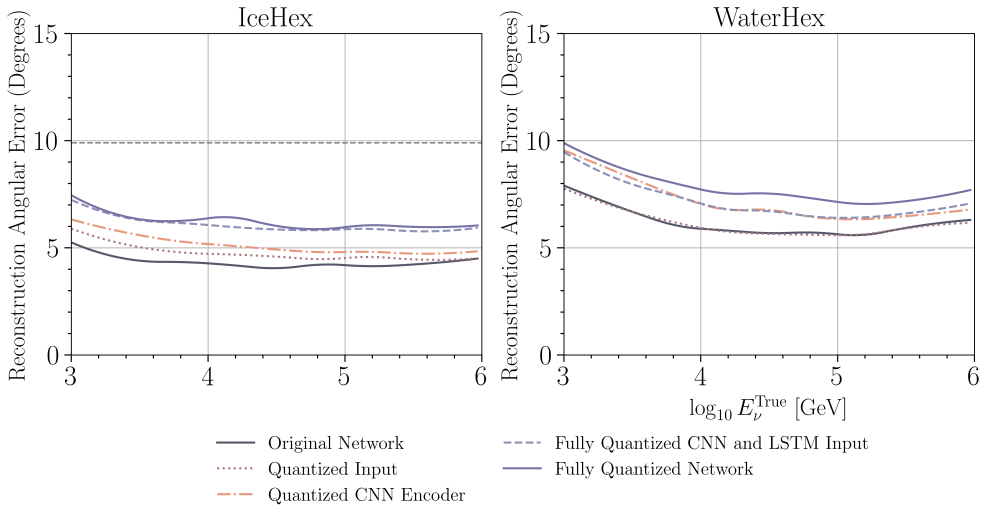


**Figure 5.** *Post-quantization accuracy of the network.* Aside from the baseline performance of the original network on the original input, we show the median angular reconstruction error of the network after the quantization of input, input and CNN encoder, and the entire network, respectively. Dashed lines show performances at intermediate quantization steps. For the IceHex detector, the dotted gray line shows the IceCube reported median angular error using the LineFit algorithm [55] at trigger level, the current real-time reconstruction method under the resources restrictions.



JCAP06(2024)026

upon retraining the LSTM cell, we see the median reconstructed angular error reach 6.0°. This increment in error is avoidable by bringing in software that is capable of quantizing the entire network without having to quantize by parts and fine-tune. Upon quantizing the entire network and evaluating the error after falling back from the integer prediction to their respective original floating point numbers via the reverse quantization mapping, we see the median error reach 6.1°. This still beats the median angular error of 9.9° of the currently employed real-time solution employed by IceCube [55], which is a regression algorithm that requires only CPU computing power, as opposed to the ML-based reconstruction we introduce in this work. This implies a very well-tuned network that works well with the quantization scheme since this error already accounts for the dead-weight loss that comes with low-power edge computing: the prediction being discrete as opposed to the continuous nature of the simulated ground truth.

For the WaterHex geometry, we observe a similar behavior across the different stages of quantization. The median reconstructed angular error being 6.0°, 6.9°, 7.0°, 7.7° respectively for the 4 quantization stages in order.

### 5.3 Inference frequency performance

With this benchmark network accuracy performance, we test the network on the various hardware architectures and hereby report the inference frequencies. For the edge-inference performance testing, we compile the fully quantized versions with the `PyCoral` compiler [56] and deploy them on the Google Edge TPU DevBoard; for the GPU run-time measurements, we run the full precision models on the various GPU architectures. It is important to note that non-fully quantized versions with TPU-incompatible operations but integer precision computations can be run on GPU architectures with a reduced run-time, but the model developed for this work is specifically designed to be run on the Edge TPU, satisfying its limitations, to tackle the power limitation problem. Therefore, while accelerating machine learning inference on GPUs with quantization can be of interest to future work, in this work we only focus on the comparison between the original performance on GPU with full precision on the one end of the spectrum and fully quantized network performance on TPU. Figure 6 shows how the performance per watt increases as the scale of the computing system decreases for single-event inference.

In a real-time, in-detector environment, for a triggering task, inferences are made on single events: this goes in the opposite direction of the strength of GPUs, which is its speed on large batch size parallel computation. In figure 6, we are considering this scenario of single event inference, which results in the very poor performance of large-scale computing systems towards the left side of the plot, and favors the smaller-sized architectures on the other end of the spectrum. We observe that for single event inference, the A100 GPU, RTX3080 GPU, and the core GPU on M1 chip demonstrate similar inference speeds at around 13 milliseconds per inference, the Google Edge TPU Dev Board hits 5 milliseconds in inference, reaching 100 Hz/watt performance while operating at 3 watts, accounting for the power consumption of other hardware parts aside from the TPU chip itself. This enables us to perform machine learning algorithms in real-time inside the detector, where power is limited to, for example for IceCube, 5.7 watts per optical Module [4], while for KM3NeT it is 7 watts per module [57].



JCAP06(2024)026

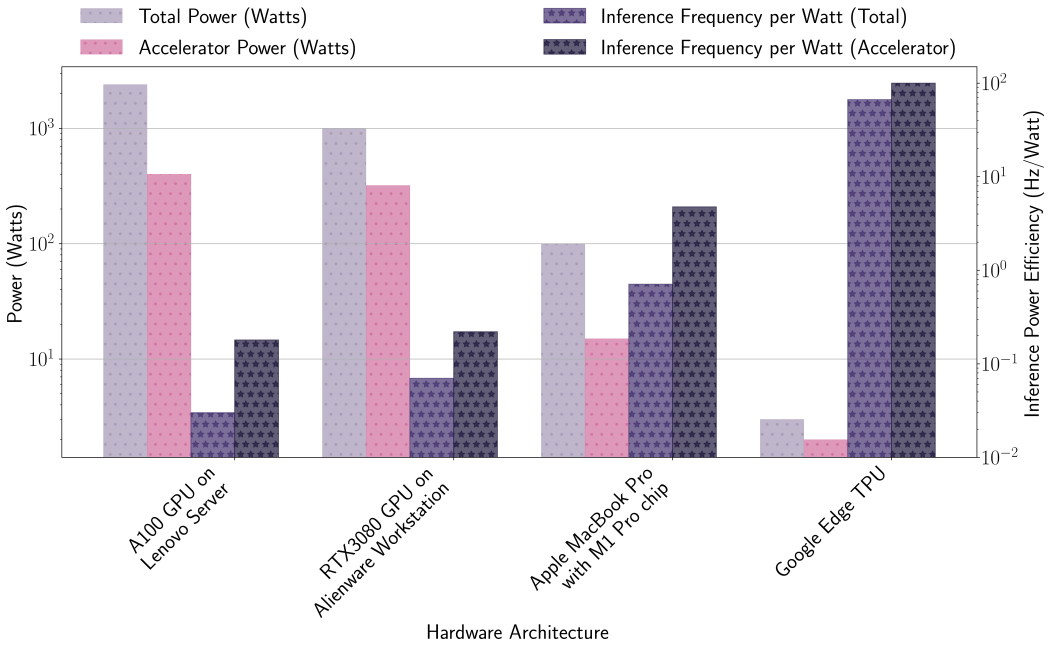


**Figure 6.** *Network power efficiency when run on different architectures.* lighter-colored, dotted bars show the total and accelerator power, star-filled, darker-colored bars show the power efficiency in inference frequency per watt.

Our work by no means undermines the capability of large-scale computing systems: if we increase the batch size to 100, then we observe that the time per inference decreases to 0.4 milliseconds for the A100 card, 1.7 milliseconds for the RTX 3080 card, and 3.8 milliseconds for the M1 Pro Chip. For algorithms with a more efficient data encoding, like that of Sparse-Submanifold CNNs [36], that enables a very large batch size, A100 cards are capable of performing inference at a rate of 9901 Hz on a batch size of 12288. Edge TPUs, on the other hand, are incapable of performing inference on a batch, but it is exactly the gains of power efficiency on single events that allow us to enable real-time in-detector machine learning inference using these edge devices. Thus GPU-designed algorithms and associated hardware are well-suited for off-line event filtering, and particle identification and reconstruction.

## 6 Summary and outlook

In this article, we have shown the accuracy and power efficiency of the TPU-tailored deep neural network for water- and ice-Cherenkov neutrino telescope event reconstruction. Our work serves as a proof of concept for the feasibility of real-time low-power in-situ machine learning tuned for ongoing and next-generation neutrino experiments. We have demonstrated the capability of a low-power machine learning algorithm following a specifically designed input pre-processing and quantization procedure, capable of reaching peak power efficiency while maintaining a competent accuracy, beating non-machine learning algorithms that are currently being deployed in real-time reconstructions or trigger systems. However, our work is far from a realization of the full potential of edge computing. To begin with, since the



JCAP06(2024)026

outputs are quantized, angular reconstruction, a continuous value prediction problem in nature, is not an optimal problem for such an algorithm to tackle. Alternatively, classification problems are better suited for deployment on edge computing, which is a future direction left for work. Additionally, the field of edge computing is under very fast development on both the software and the hardware fronts, some recent developments include the recent release of the `PyTorch ExecuTorch` that enables a new end-to-end solution for edge computing, supporting more edge devices [58].

Looking forward, for next-generation detectors, a new variety of situations will appear that calls out for the need for low-power real-time machine-learning-based data handling, including but not limited to the following scenarios:

- *Intelligent in-situ data encoding*: in some detectors, real-time waveform information is not completely saved and transmitted ashore, instead they are highly compressed [59]. In these scenarios, the deployment of such real-time machine learning accelerators will allow for more intelligent compressing methods.

- *Multiple-PMT optical modules*: among designs of next-generation optical modules, many have incorporated multiple PMTs into single modules [60]. This opens up the space for an algorithm that processes the multiple waveforms of this single module as time series data. In this scenario, the incorporation of an edge computing device would enable us to deploy machine learning-based sophisticated algorithms assisting this processing of local data.

- *Real time triggering*: next-generation neutrino telescopes will typically be much larger in geometric and effective areas, looking at IceCube-Gen2, for example [29]. This implies a larger amount of data transmission and storage requirement if we keep using the same simple cutoff-like triggers. The inclusion of real-time machine learning capability will allow us to instead develop machine learning-based triggering, which will not only help us in detecting rare events in real-time but also assist us in data selection and therefore alleviate the stressed data transmission and storage system.

- *Other power-limited facilities*: aside from water-/ice- Cherenkov neutrino telescopes, there are other experiments in similarly, extremely, power-limited environments, such as satellite detectors [61]. Enabling machine learning in such environments will allow for a real-time data handling system in these scenarios as well.

- *Other edge computing devices*: many more edge computing alternatives are low-power and efficient, and they usually face the same set of restrictions. Using similar quantization ideas developed in this work, we can explore more variations of micro-architectures, evaluating the pros and cons of each before incorporating them into the next-generation intelligent detector hardware.

As such, through this first demonstration of machine learning effort on TPUs, we would like to motivate similar further exploration into this direction of low-power computing alternatives. Various other improvements and applications are waiting out there to be explored along this newly opened gateway.



JCAP06(2024)026

**Code availability.** The code used to train the network and produce the plots in this work can be found in GitHub Repository.


## Acknowledgments

We thank Simone Francescato for painstaking comments and suggestions. MJ would also like to thank Nicholas Kamp, Felix Yu, Lihao Yan, Yidi Qi and Jinzheng Li for useful discussions. CAA is supported by the Faculty of Arts and Sciences of Harvard University and the National Science Foundation (NSF). Through part of this work, they were also supported by the Alfred P. Sloan Foundation. MJ was supported by the Harvard Physics Department Purcell Fellowship for part of this work. The NSF partially supported this work under Cooperative Agreement PHY-2019786 (The NSF AI Institute for Artificial Intelligence and Fundamental Interactions, http://iaifi.org/) Finally, CAA and MJ are supported by NSF CAREER Award PHY-2239795.


## A Data input and pre-processing visualization

Here in figure 7 we show a visualization of a typical event in the water detector with its pre-processing. Here, the total number of timesteps is chosen to be $T = 6$ for the sake of simplicity. The left panel contains information equivalent to the 3-dimensional accumulative photon hits tensor $A^{(X,Y,Z)}$, while the right 6 panels are, respectively, the time-ordered separated photon hits tensors $A_t^{(X,Y,Z)}$ for $t \in [0, 6]$ with $A^{(X,Y,Z)} = \sum_{t=0}^{6} A_t^{(X,Y,Z)}$.

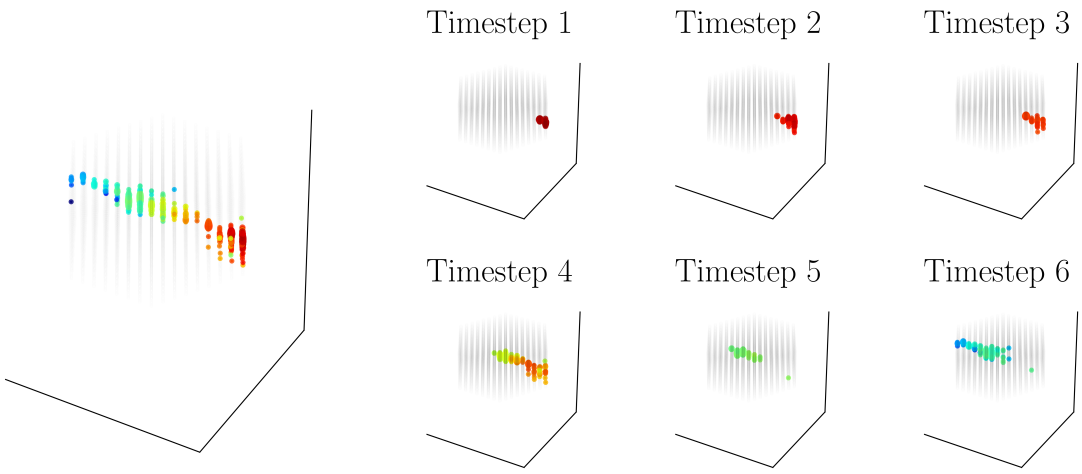


**Figure 7.** *Event visualization.* The left panel shows a visualization of the detector response to a typical simulated event in `Prometheus`. Light gray dots are the locations of the OMs; colored dots denote OMs that received photon hits: the red end of the spectrum signifies earlier hits, while the blue end of the spectrum signifies later hits; the sizes of the dots denote the number of photons seen. The right 6 panels are the photon hits received in each time step, respectively. For the right panels, we keep the coloring (timing) information for better visualization, whereas, in reality, such time information is no longer included in the pre-processed input data tensors.



JCAP06(2024)026

## B Network architecture

Here we show in detail the network architecture developed in this work. Firstly, in table 2 we show a cellular basic component of the CNN encoder: the residual convolution block. This block is used extensively in the CNN encoder, whose architecture and hyperparameters are shown in table 3. We finally show the entire network architecture, consisting of tensor reshaping layers, the CNN encoder, and the LSTM block together with a final fully connected layer in table 4.

| Layer Name | Layer Specs |
|---|---|
| `Input` | $X_{\text{in}}$ |
| $\texttt{Conv2D}_1$ | Kernel Size $(k_x, k_y)$; Stride $s_1$; Channels $c$; Padding |
| $\texttt{Conv2D}_2$ | Kernel Size $(k_x, k_y)$; Stride $s_2$; Channles $c$; Padding |
| `Addition` | $\texttt{Add}(X_{\text{in}} + \texttt{Conv2D}_1(\texttt{Conv2D}_2(X_{\text{in}})))$ |

**Table 2.** *Residual block.* Architecture of a single cellular residual block with given kernel size $(k_x, k_y)$, stride $(s_1, s_2)$, and some padding strategy, used extensively in the CNN encoder of the network. $s_1 \neq s_2$ implies downsampling.

| Layer Name | Layer Specs |
|---|---|
| `Input` | $X_{\text{in}}$ |
| $\texttt{Conv2D}_1$ | Kernel Size $(3, 3)$; Stride 2; Channels: 32; Padding: `"same"` |
| $\texttt{ResBlock}_1$ | $k_x = k_y = 3$; $s_1 = s_2 = 1$; $c = 32$; Padding = `"same"` |
| $\texttt{ResBlock}_2$ | $k_x = k_y = 3$; $s_1 = 2, s_2 = 1$; $c = 32$; Padding = `"same"` |
| $\texttt{ResBlock}_3$ | $k_x = k_y = 3$; $s_1 = s_2 = 1$; $c = 64$; Padding = `"same"` |
| $\texttt{ResBlock}_4$ | $k_x = k_y = 3$; $s_1 = 2, s_2 = 1$; $c = 64$; Padding = `"same"` |
| $\texttt{ResBlock}_5$ | $k_x = k_y = 3$; $s_1 = s_2 = 1$; $c = 128$; Padding = `"same"` |

**Table 3.** *CNN encoder.* Architecture of the CNN encoding network that encodes the input at any time step $t$: $A_t^{(X,Y,Z)}$ into a 1-dimensional vector in the hidden latent space, which in turn gets processed by the LSTM Block.



JCAP06(2024)026

| Layer Name | Layer Specs |
|---|---|
| `Input` | $\{A_T^{(X,Y,Z)}\}_B$ |
| $\texttt{Reshape}_1$ | $\texttt{Reshape}(\{A_T^{(X,Y,Z)}\}_B, [B \times T, X, Y, Z])$ |
| `CNN Encoder` | See table 2 |
| $\texttt{Reshape}_2$ | $\texttt{Reshape}(\texttt{CNN Encoder}(\texttt{Reshape}(\{A_T^{(X,Y,Z)}\}_B, [B \times T, X, Y, Z])), [B, T, 128])$ |
| `LSTM` | LSTM Dimension: 128; $n_{\text{hidden}}$: 128 |
| `Dense` | To Output |

**Table 4.** *Network architecture.* Architecture of the entire network, with given input sizes depending on the data pre-processing and detector geometry parameters $T, X, Y, Z$.

## References


[1] IceCube collaboration, *A combined maximum-likelihood analysis of the high-energy astrophysical neutrino flux measured with IceCube*, *Astrophys. J.* **809** (2015) 98 [arXiv:1507.03991] [INSPIRE].

[2] IceCube collaboration, *The IceCube high-energy starting event sample: Description and flux characterization with 7.5 years of data*, *Phys. Rev. D* **104** (2021) 022002 [arXiv:2011.03545] [INSPIRE].

[3] J.A. Formaggio and G.P. Zeller, *From eV to EeV: Neutrino Cross Sections Across Energy Scales*, *Rev. Mod. Phys.* **84** (2012) 1307 [arXiv:1305.7513] [INSPIRE].

[4] IceCube collaboration, *The IceCube Neutrino Observatory: Instrumentation and Online Systems*, 2017 *JINST* **12** P03012 [*Erratum ibid.* **19** (2024) E05001] [arXiv:1612.05093] [INSPIRE].

[5] C. Spiering, *Towards High-Energy Neutrino Astronomy. A Historical Review*, *Eur. Phys. J. H* **37** (2012) 515 [arXiv:1207.4952] [INSPIRE].

[6] IceCube collaboration, *Evidence for neutrino emission from the nearby active galaxy NGC 1068*, *Science* **378** (2022) 538 [arXiv:2211.09972] [INSPIRE].

[7] IceCube collaboration, *Observation of high-energy neutrinos from the Galactic plane*, *Science* **380** (2023) adc9818 [arXiv:2307.04427] [INSPIRE].

[8] IceCube collaboration, *Neutrino emission from the direction of the blazar TXS 0506+056 prior to the IceCube-170922A alert*, *Science* **361** (2018) 147 [arXiv:1807.08794] [INSPIRE].

[9] ANTARES collaboration, *Hint for a TeV neutrino emission from the Galactic Ridge with ANTARES*, *Phys. Lett. B* **841** (2023) 137951 [arXiv:2212.11876] [INSPIRE].

[10] C.A. Argüelles et al., *Fundamental physics with high-energy cosmic neutrinos today and in the future*, *PoS* **ICRC2019** (2020) 849 [arXiv:1907.08690] [INSPIRE].

[11] M. Bustamante and S.K. Agarwalla, *Universe's Worth of Electrons to Probe Long-Range Interactions of High-Energy Astrophysical Neutrinos*, *Phys. Rev. Lett.* **122** (2019) 061103 [arXiv:1808.02042] [INSPIRE].

[12] C.A. Argüelles, T. Katori and J. Salvado, *New Physics in Astrophysical Neutrino Flavor*, *Phys. Rev. Lett.* **115** (2015) 161303 [arXiv:1506.02043] [INSPIRE].




JCAP06(2024)026

[13] IceCube collaboration, *Search for quantum gravity using astrophysical neutrino flavour with IceCube*, *Nature Phys.* **18** (2022) 1287 [arXiv:2111.04654] [INSPIRE].

[14] I.M. Shoemaker and K. Murase, *Probing BSM Neutrino Physics with Flavor and Spectral Distortions: Prospects for Future High-Energy Neutrino Telescopes*, *Phys. Rev. D* **93** (2016) 085004 [arXiv:1512.07228] [INSPIRE].

[15] M. Bustamante, J.F. Beacom and K. Murase, *Testing decay of astrophysical neutrinos with incomplete information*, *Phys. Rev. D* **95** (2017) 063013 [arXiv:1610.02096] [INSPIRE].

[16] N. Song et al., *The Future of High-Energy Astrophysical Neutrino Flavor Measurements*, *JCAP* **04** (2021) 054 [arXiv:2012.12893] [INSPIRE].

[17] A. Abdullahi and P.B. Denton, *Visible Decay of Astrophysical Neutrinos at IceCube*, *Phys. Rev. D* **102** (2020) 023018 [arXiv:2005.07200] [INSPIRE].

[18] Y. Farzan and S. Palomares-Ruiz, *Flavor of cosmic neutrinos preserved by ultralight dark matter*, *Phys. Rev. D* **99** (2019) 051702 [arXiv:1810.00892] [INSPIRE].

[19] M.M. Reynoso, O.A. Sampayo and A.M. Carulli, *Neutrino interactions with ultralight axion-like dark matter*, *Eur. Phys. J. C* **82** (2022) 274 [arXiv:2203.11642] [INSPIRE].

[20] C.A. Argüelles, K. Farrag and T. Katori, *Ultra-light Dark Matter Limits from Astrophysical Neutrino Flavour*, *PoS* **ICRC2023** (2023) 1415 [arXiv:2402.18126] [INSPIRE].

[21] C.A. Argüelles et al., *Sterile neutrinos in astrophysical neutrino flavor*, *JCAP* **02** (2020) 015 [arXiv:1909.05341] [INSPIRE].

[22] K. Carloni et al., *Probing Pseudo-Dirac Neutrinos with Astrophysical Sources at IceCube*, *PoS* **ICRC2023** (2023) 1040 [arXiv:2212.00737] [INSPIRE].

[23] C.A. Argüelles et al., *Snowmass white paper: beyond the standard model effects on neutrino flavor: Submitted to the proceedings of the US community study on the future of particle physics (Snowmass 2021)*, *Eur. Phys. J. C* **83** (2023) 15 [arXiv:2203.10811] [INSPIRE].

[24] K. Murase and I.M. Shoemaker, *Neutrino Echoes from Multimessenger Transient Sources*, *Phys. Rev. Lett.* **123** (2019) 241102 [arXiv:1903.08607] [INSPIRE].

[25] K. Murase and I. Bartos, *High-Energy Multimessenger Transient Astrophysics*, *Ann. Rev. Nucl. Part. Sci.* **69** (2019) 477 [arXiv:1907.12506] [INSPIRE].

[26] C. Guépin, K. Kotera and F. Oikonomou, *High-energy neutrino transients and the future of multi-messenger astronomy*, *Nature Rev. Phys.* **4** (2022) 697 [arXiv:2207.12205] [INSPIRE].

[27] Baikal-GVD collaboration, *Baikal-GVD: status and first results*, *PoS* **ICHEP2020** (2021) 606 [arXiv:2012.03373] [INSPIRE].

[28] KM3Net collaboration, *Letter of intent for KM3NeT 2.0*, *J. Phys. G* **43** (2016) 084001 [arXiv:1601.07459] [INSPIRE].

[29] IceCube-Gen2 collaboration, *IceCube-Gen2: the window to the extreme Universe*, *J. Phys. G* **48** (2021) 060501 [arXiv:2008.04323] [INSPIRE].

[30] Z.P. Ye et al., *A multi-cubic-kilometre neutrino telescope in the western Pacific Ocean*, arXiv:2207.04519 [INSPIRE].

[31] T.-Q. Huang et al., *Proposal for the High Energy Neutrino Telescope*, *PoS* **ICRC2023** (2023) 1080 [INSPIRE].

[32] P-ONE collaboration, *The Pacific Ocean Neutrino Experiment*, *Nature Astron.* **4** (2020) 913 [arXiv:2005.09493] [INSPIRE].

JCAP06(2024)026

[33] A.M. Brown et al., *Trinity: An Imaging Air Cherenkov Telescope to Search for Ultra-High-Energy Neutrinos*, in the proceedings of the *37th International Cosmic Ray Conference*, Berlin, Germany, 15–22 July 2021 [arXiv:2109.03125] [INSPIRE].

[34] TAMBO collaboration, *TAMBO: Searching for Tau Neutrinos in the Peruvian Andes*, in the proceedings of the *38th International Cosmic Ray Conference*, Nagoya, Japan, 26 July–03 August 2023 [arXiv:2308.09753] [INSPIRE].

[35] GRAND collaboration, *The Giant Radio Array for Neutrino Detection (GRAND): Science and Design*, *Sci. China Phys. Mech. Astron.* **63** (2020) 219501 [arXiv:1810.09994] [INSPIRE].

[36] F.J. Yu, J. Lazar and C.A. Argüelles, *Trigger-level event reconstruction for neutrino telescopes using sparse submanifold convolutional neural networks*, *Phys. Rev. D* **108** (2023) 063017 [arXiv:2303.08812] [INSPIRE].

[37] *The edge tpu dev board*, https://coral.ai/products/dev-board.

[38] N.P. Jouppi et al., *In-Datacenter Performance Analysis of a Tensor Processing Unit*, arXiv:1704.04760.

[39] H.-H. Chin, R.-S. Tsay and H.-I. Wu, *A High-Performance Adaptive Quantization Approach for Edge CNN Applications*, arXiv:2107.08382.

[40] C.J.S. Schaefer, S. Joshi, S. Li and R. Blazquez, *Edge Inference with Fully Differentiable Quantized Mixed Precision Neural Networks*, arXiv:2206.07741.

[41] M. Abadi et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*, arXiv:1603.04467.

[42] J. Lazar et al., *Prometheus: An Open-Source Neutrino Telescope Simulation*, arXiv:2304.14526 [INSPIRE].

[43] *A100 gpu specs*, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet.pdf.

[44] *Rtx 3080 gpu specs*, https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3080-3080ti/.

[45] *Apple macbook pro specs*, https://support.apple.com/en-us/111901.

[46] *Edge tpu supported operations*, https://coral.ai/docs/edgetpu/models-intro/#supported-operations.

[47] K. Seshadri et al., *An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks*, arXiv:2102.10423.

[48] W. Chen et al., *Quantization of Deep Neural Networks for Accurate Edge Computing*, arXiv:2104.12046.

[49] B. Jacob et al., *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*, arXiv:1712.05877.

[50] R. Abbasi et al., *A Convolutional Neural Network based Cascade Reconstruction for the IceCube Neutrino Observatory*, 2021 *JINST* **16** P07041 [arXiv:2101.11589] [INSPIRE].

[51] L. Alzubaidi et al., *Review of deep learning: concepts, cnn architectures, challenges, applications, future directions*, *J. Big Data* **8** (2021) 53.

[52] F. Scarselli et al., *The Graph Neural Network Model*, *IEEE Trans. Neural Networks* **20** (2009) 61 [INSPIRE].

[53] S. Hochreiter and J. Schmidhuber, *Long Short-Term Memory*, *Neural Comput.* **9** (1997) 1735 [INSPIRE].

[54] *Tensorflow lite library*, https://www.tensorflow.org/lite.

[55] M.G. Aartsen et al., *Improvement in Fast Particle Track Reconstruction with Robust Statistics*, *Nucl. Instrum. Meth. A* **736** (2014) 143 [arXiv:1308.5501] [INSPIRE].

[56] *Pycoral edgetpu compiler*, https://coral.ai/docs/edgetpu/compiler/.

[57] KM3NeT collaboration, *The Digital Optical Module of KM3NeT*, *J. Phys. Conf. Ser.* **1056** (2018) 012031 [INSPIRE].

[58] *Pytorch executorch*, https://pytorch.org/executorch-overview.

[59] KM3NeT collaboration, *Characterisation of the Hamamatsu photomultipliers for the KM3NeT Neutrino Telescope*, 2018 *JINST* **13** P05035 [INSPIRE].

[60] IceCube collaboration, *Design and performance of the multi-PMT optical module for IceCube Upgrade*, *PoS* **ICRC2021** (2021) 1070 [arXiv:2107.11383] [INSPIRE].

[61] LiteBIRD collaboration, *Probing Cosmic Inflation with the LiteBIRD Cosmic Microwave Background Polarization Survey*, *PTEP* **2023** (2023) 042F01 [arXiv:2202.02773] [INSPIRE].

[62] N.P. Jouppi et al., *TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings*, arXiv:2304.01433.

[63] IceCube collaboration, *Graph Neural Networks for low-energy event classification & reconstruction in IceCube*, 2022 *JINST* **17** P11003 [arXiv:2209.03042] [INSPIRE].

[64] *Edge tpu performance benchmarks*, https://coral.ai/docs/edgetpu/benchmarks/.

[65] *Edge tpu quantization*, https://coral.ai/docs/edgetpu/models-intro/#quantization.

[66] *Tensorflow quantization aware training*, https://www.tensorflow.org/model_optimization/guide/quantization/training.

[67] M.Z. Alom et al., *Effective Quantization Approaches for Recurrent Neural Networks*, arXiv:1802.02615.

[68] Q. He et al., *Effective Quantization Methods for Recurrent Neural Networks*, arXiv:1611.10176.