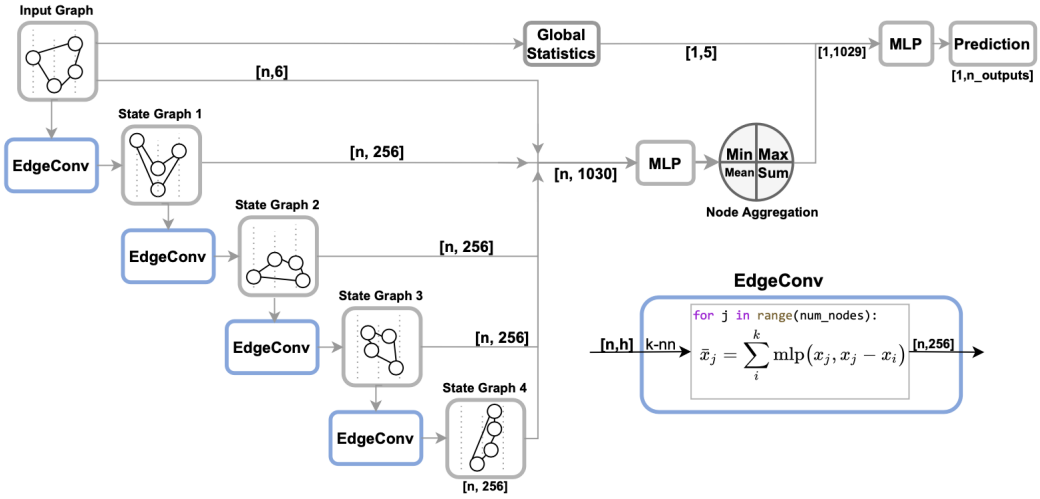

Figure 9: A diagram of the model dynedge.

6 IceCube Data

TODO: Data from simulation description

7 Icedata

This section describes the IceCube ice model data and the process of preparing (reading, converting, interpolating) it for supervised learning.

The properties of antarctic ice is not constant across the whole cubic kilometer of IceCube. The optical properties vary noticably due to changing densities of the ice, but also because of impurities like dust. In figure 2 a graph visualizing the absorption and scattering is shown, giving an overview of how the ice changes vertically.

The project SPICE measured the ice properties (icedata) and organized it into layers (ice of equal properties) indexed by depth. Measurements were taken along 6 strings every 10 m vertically. The recorded values are not constant horizontally, therefore the layers are (slightly) tilted. This tilt varies across depth, which complicates the interpolation process (section 7.3).

7.1 Coordinate Systems

Most annoyingly several types of coordinate system are commonly used within IceCube. This section will outline the differences between the ones relevant in this work and provide handy conversions to be used lateron when working with the icedata.

Spice

The Spice dataset is organized and labeled/keyed by depth, which is 0 at the surface of the ice and grows positive downwards. We will call this z-coordinate z_{Depth} or simply Depth.

IceCube

Most of IceCube uses the IceCube coordinate system which has its origin 1948,07 m below the surface, its z-coordinate will be called $z_{IceCube}$ and grows positive upwards.

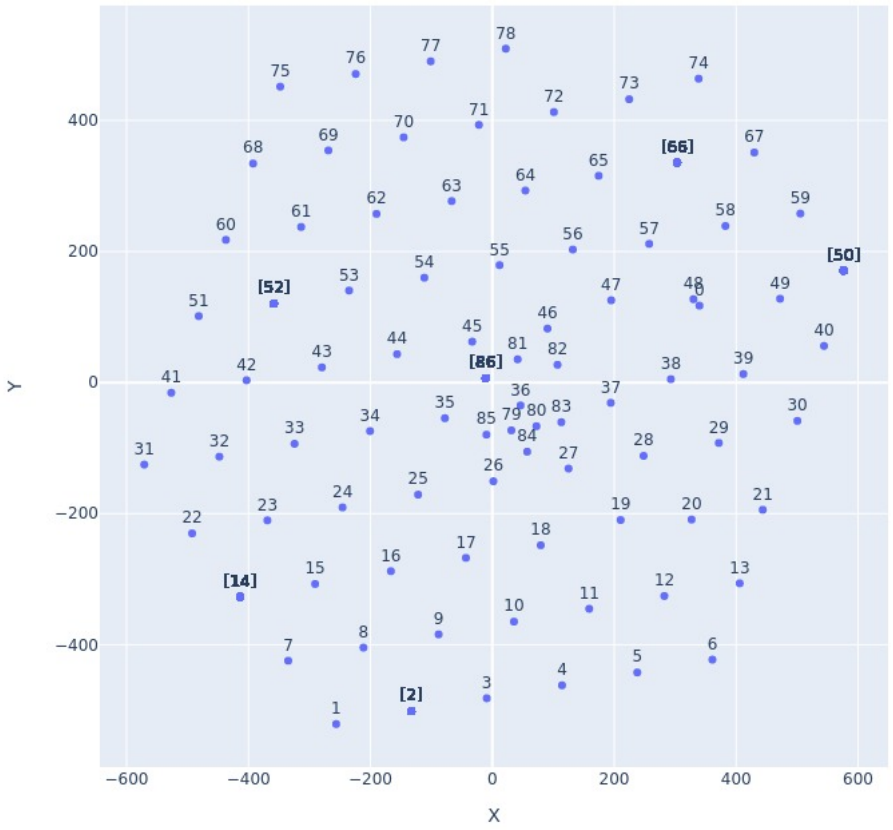

Figure 10: An overview of where SPICE measurements have been taken. Strings where measurements have been made are marked with square brackets.

Global

The z-coordinate we will be working with (and converting the other coordinates into) is z_{Global}, a coordinate system that has its origin at the surface and grows upwards. This definition results in the following conversions:

$$z_{Global} = -z_{Depth} \tag{9}$$

$$z_{Global} = z_{IceCube} - 1948{,}07\,\mathrm{m} \tag{10}$$

7.2 Spice files

All files of interest are space-seperated-values formatted and are read by using the Python `pandas` library. For example, the file `tilt.par` is read like so:

```
import pandas as pd

df_tilt_par = pd.read_csv('tilt.par', sep=' ', names=['string_no', 'dist_sw_225'])
```

icemodel.dat

This file contains all of the interesting data of ice properties. As loosely descibed above, the data is indexed by depth, which identifies the layer the row of data is related to.

The columns of data contained in this file represents the following (physical) properties:

- `depth`: depth below surface (used as layer identifier)
- `be(400)`:
- `adust(400)`: absorption at wavelength of 400nm
- `k1`:
- `k2`:
- `BFRA`: birefringence (axis 1)
- `BFRB`: birefringence (axis 2)

TODO: Describe properties more thurrowly

Depth	be(400)	adust(400)	k1	k2	BFRA	BFRB

tilt.par

This file describes where the ice layer tilt offsets are measures at, this means it gives a horizontal (xy) description of where the offsets are located. `SW_255` is the horizontal distance in the SouthWest (225deg) direction with its origin located at string 86.

Due to the natural relationship between string number and its location the data is redundant and only `SW_255` will be used from now on.

- `string_no`: string number
- `sw_225`: distance [m] into the south-west direction (225 °) direction

String	SW_225
50	-531.419
66	-454.882
86	0
52	165.202
2	445.477
14	520.77

tilt.dat

This file describes how the layers of ice are tilted relative to the flat assumption at the locations specified in `tilt.par`. It holds the 6 offset values at all depths. Together with the locations from above, it will be used to build a model of tilted layers.

Depth	Offset 50	Offset 66	Offset 86	Offset 52	Offset 2	Offset 14

7.3 Interpolation

As discussed in section 7 the layer tilt is not constant across depth, therefore the process of assigning a layer to an arbitrary point is non-trivial. Not only do the layers have to be interpolated over the horizontal axis, but also the icedata has to be interpolated vertically between the layers. This process and all decisions/assumptions made are described in this section.

Since the layers are described at the locations defined in `tilt.par` it is most sensible to work in the z-southwest coordinate system. More specifically we use z_{Global} as the y-axis and distance in the south-west direction as the x-axis with string 86 as the origin. Onward we will use the function `sw_225` to calculate the sw-coordinate of any given point.

```
import numpy as np

SW_225 = np.array([-1, -1]) / np.sqrt(2)  # normalized direction vector
ORIGIN = XY_STRING[86]                    # string 86 is zero of ‘tilt.dat‘

def sw_225(xy):
    return np.dot(xy - ORIGIN, SW_225)
```

7.3.1 Layers

First a function/algorithm is required to decide on what layer a given point is located in, or more accurately, what layers a given point is located between. To build such a function, we interpolate the measured layer tilt offsets across the sw-coordinate. By adding/subtracting the base layer depth to the interpolated offset we get the actual layer heights at any point. Doing this for all positions, we get the resulting layers shown in figure 11.

Extrapolation

As is visible in figure 10 the strings 7 and 1 are further into the south-west direction than the furthest measured string 14. This is also the case with strings 74, 67 and 59, which extend past the last measured string in that direction, string 50. This makes the choice to extrapolate the layers in both directions easy and involuntary.

Quadratic Interpolation

We interpolate between the measured offsets quadratically. This decision is clearly biased, but hopefully also the better choice over a linear interpolation. The resoning being, that the icelayers are probably not spikey at the points of measurement. Therefore a quadratic interpolation will provide for more even and flowy layers, instead of spikey, undifferentiable layers.

Continued Layers

Strangely the last layer contained in `tilt.dat` is at a depth of 2448,47 m, which is a few dozen meters above the lowest DOMs. Though the data contained in `icemodel.dat` extends all the way to a depth of 2798,47 m. To incorporate this existing data, the decision was made to simply copy the shape of the lowest layer 7 times. This allows for a interpolation that respects the given icedata while hopefully being as close to the actual shape of layers as possible. These additional layers are visible in figure 11.

Code

To realize our functionality in code, the 1D interpolation function of `scipy` is used. The data being used is stored in `df_tilt_par` and `df_tilt_dat`, which comes from the corresponding files described in 7.2.

Since `interpolation_dzs` only interpolates the offsets, we create another funtion `interpolation_zs`, which adds the z-coordinate of every layer to the offsets and converts between the Spice coordinates and the global coordinates (see 7.1). It will return an array of the heights of all layers at the location it is given `sw`.

```
from scipy.interpolate import interp1d

interpolation_dzs = interp1d(
    df_tilt_par.dist_sw_225.to_numpy(),  # positions where offsets were measured (6,)
    df_tilt_dat.iloc[:, 1:].to_numpy(),  # offsets = dzs (125, 6)
    kind='quadratic',
    axis=1,
    bounds_error=False,
    fill_value='extrapolate',
)
```

```
def interpolation_zs(sw): return -(df_tilt_dat.z - interpolation_dzs(sw))
```

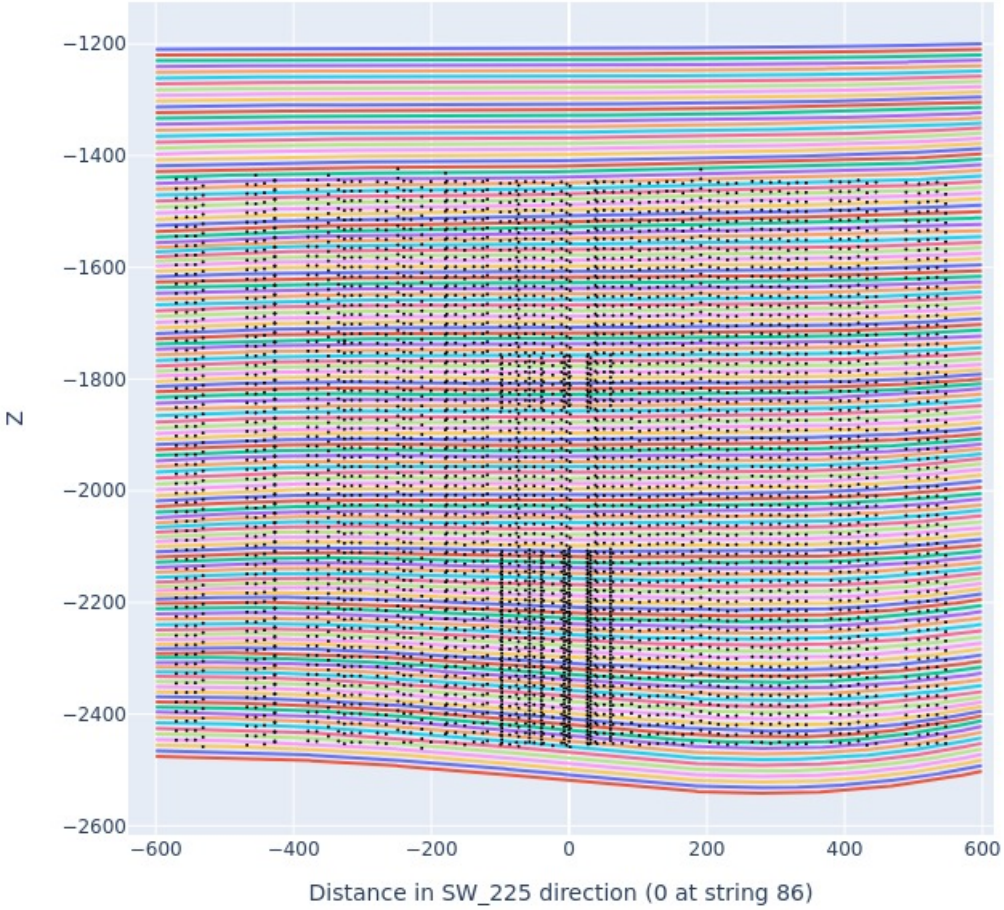

Figure 11: The interpolated ice layers (color) and all DOMs (black).

7.3.2 Ice Properties

To now compute the icedata at any point we first use `interpolation_zs` designed above to get the heights of all layers at the location point of interest. Then we do another interpolation, this time interpolating the icedata over the just calculated layers heights. This gives us a function that will interpolate the icedata between the layer above and below the point of interest at a given height.

Quadratic Interpolation

A quadratic interpolation was chosen for similar reasons as before for the layers. It is intended to smooth over outliers in the data and make it less spikey. Having a differentiable function to describe the properties is probably also a more correct approach.

Extrapolation

If we had not previously continued the layers in `tilt.dat`, we would have to use extrapolation in order to calculate the icedata of the lowest DOMs. But because we have extended the layers in order to contain all DOMs between them, extrapolation is not required here.

Code

The functionality discussed above relies on the layer heights at the point of interest. Therefore we run the function `interpolation_zs` created before in order to get the layer heights. We use use this data together with the icedata from `icemodel.dat` to build the second interpolation, again using the 1D interpolation function from `scipy`.

The function `interpolation_features` will return a 2D array of icedata specific to the point of interest identified by its sw- and z-coordinate.

```
layer_zs_global = interpolation_zs(sw)

interpolation_features = interp1d(
    layer_zs_global,  # layer heights from before (125,)
    layer_features,   # icedata from icemodel.dat aligned with layers from tilt.dat (125, 6)
    kind='quadratic',
    axis=0,
    bounds_error=False,
    fill_value='extrapolate',
)
```

7.4 Lookup Table

To make our set of interpolation functions more usable and performant during training, a lookup table is built. So instead of interpolating the features on every occurence while training, we precalculate the values for all DOMs beforehand. In order to achieve this, we loop over all DOMs and their positions, calculate their sw-coordinate and then calculate their icedata. The lookup table itself is implemented as a simple key-value store (Python dictionary), which uses the DOM locations (xyz) as its key and returns a 6 element array containing the icedata for that DOM.