# Determining Antarctic Ice Sheet Volume from RADAR Measurements

Bastian Jockers, Malthe Wilson, Lea Keutmann

# Research Question:

Track trends in Antarctic ice
→ climate change and sea level rise

Decades of collected data: outliers

Flag outliers in ice thickness
→ score from 0 to 1
→ deviation from predicted thickness

# Tabular Data: 75 Mio. Rows/Datapoints

## Basic data set

- 19 columns/features:
  - Spatial
  - Temporal
  - Administrative
- 48k continuous Tracks (Track IDs)

**Flawed Data-entries:**

- Up to 64mio **"-9999"** in some columns

## Climate data set

- 9 additional features
  - Ice velocities (x, y, total)
  - temperature, modelled ice thickness, surface mass balance, slope, vertical height

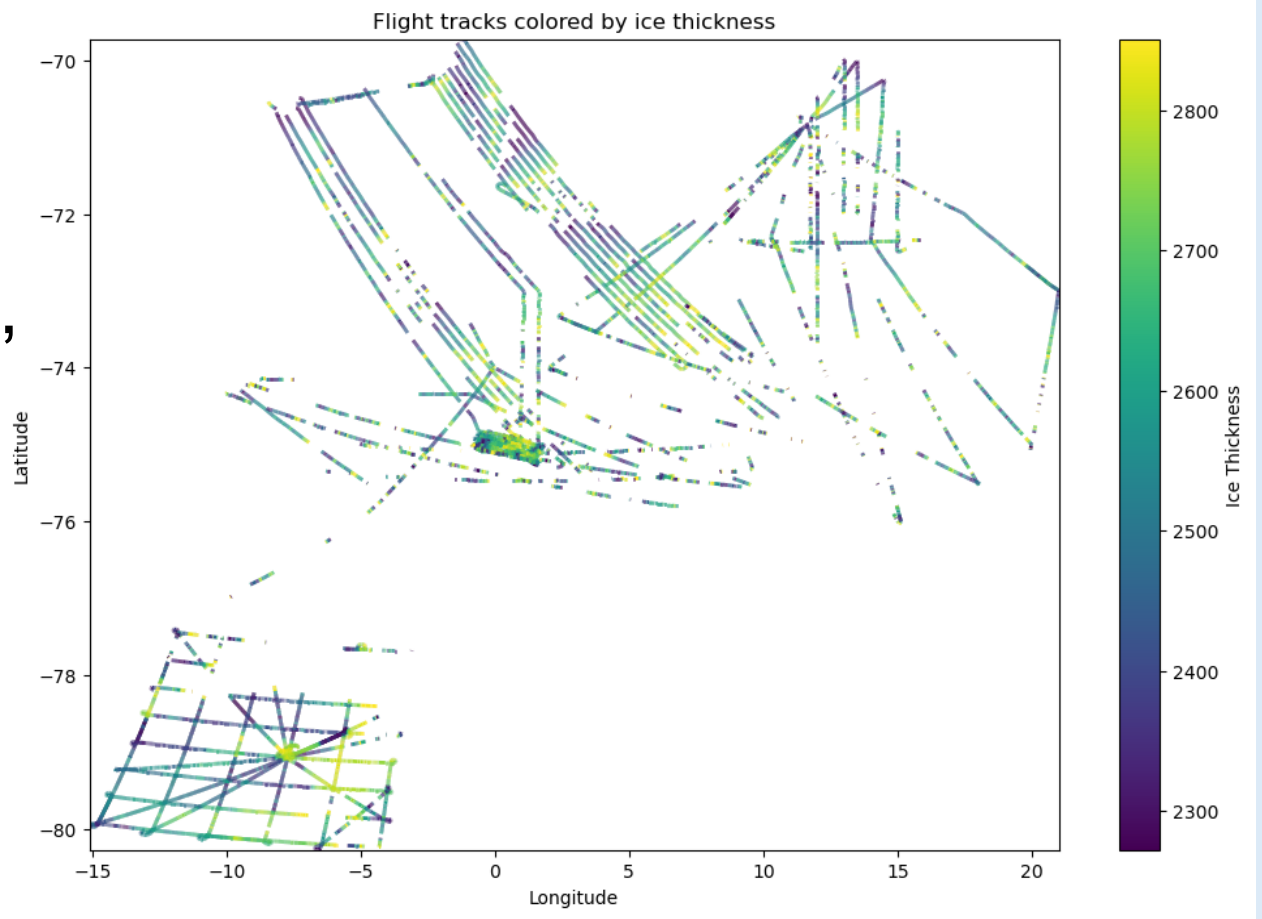
- **NaN**s in ca. 50k rows

**Target:** label datapoint (good/bad)

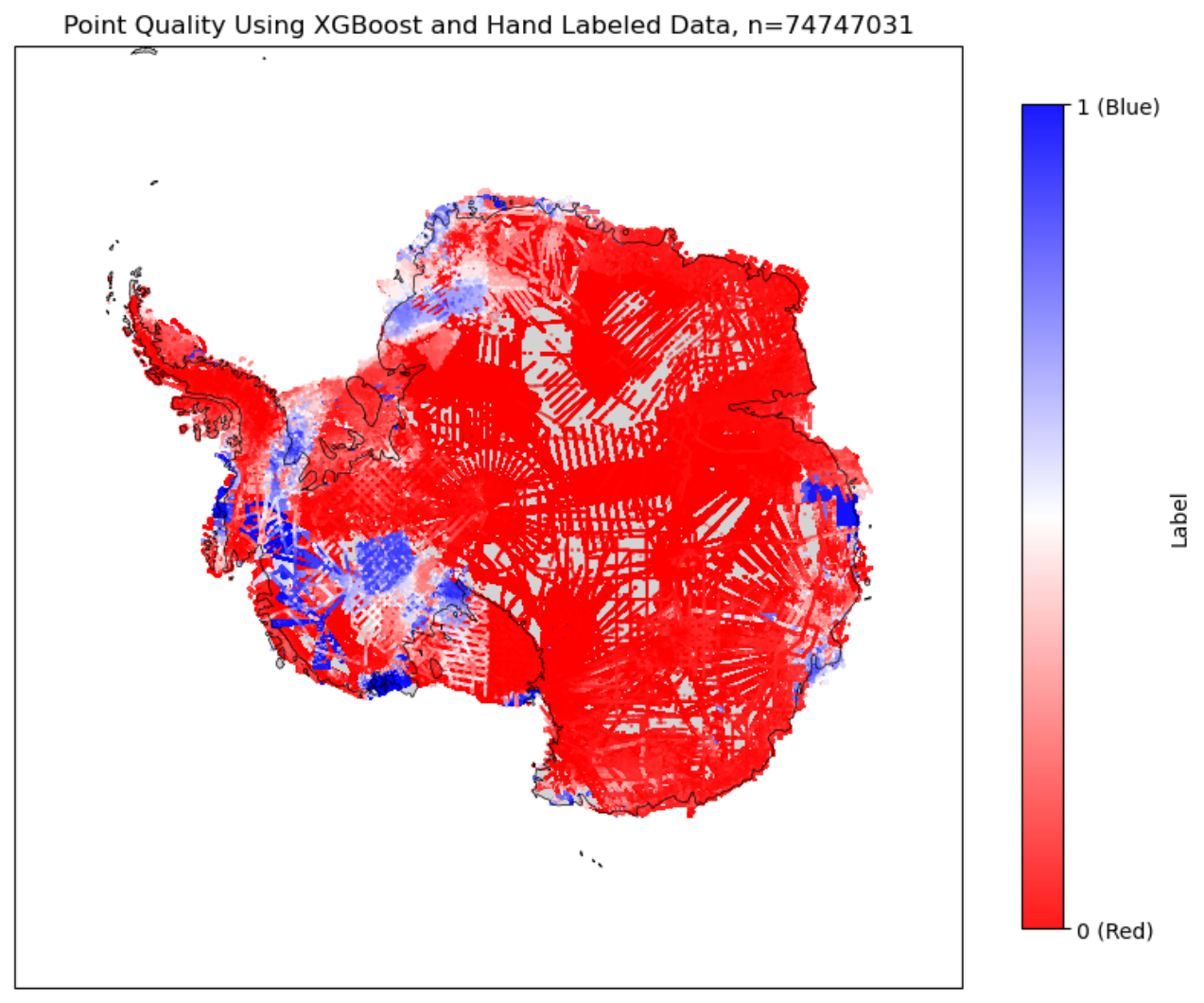
# Preprocessing: Tracks Stats

- Calculate all track crossings + height difference at crossing
- Statistics by track: nr. of crossings, mean, median, std of height differences at crossing

- Mean of mean difference: 166m
- Median of mean difference: 58m
- Mean crossings: 89
- Median crossings: 10



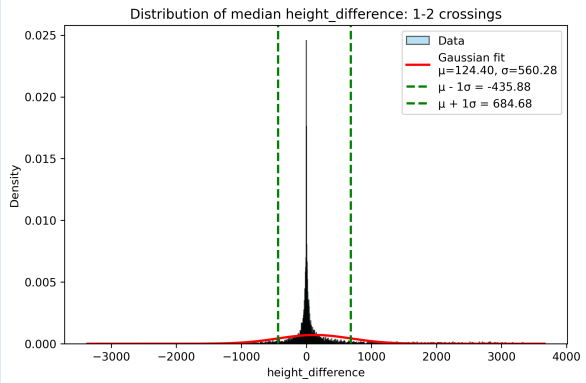
# Method 1: Quick and dirty , Classification

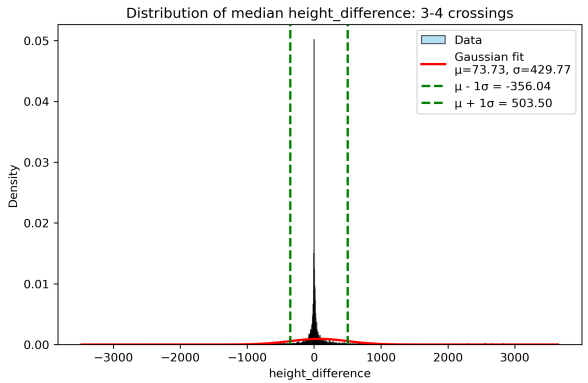


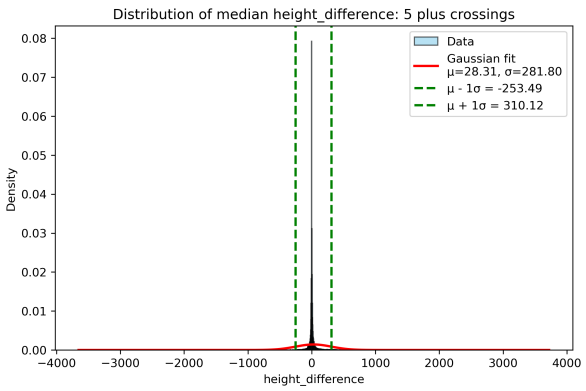

- XGBoost and Light GBM
- Manually labeled training data from track crossings
- Predict track quality for full data: 0-1

- Track_id not most important variable
- Overwhelmingly predicts bad points
- Trained on extremes

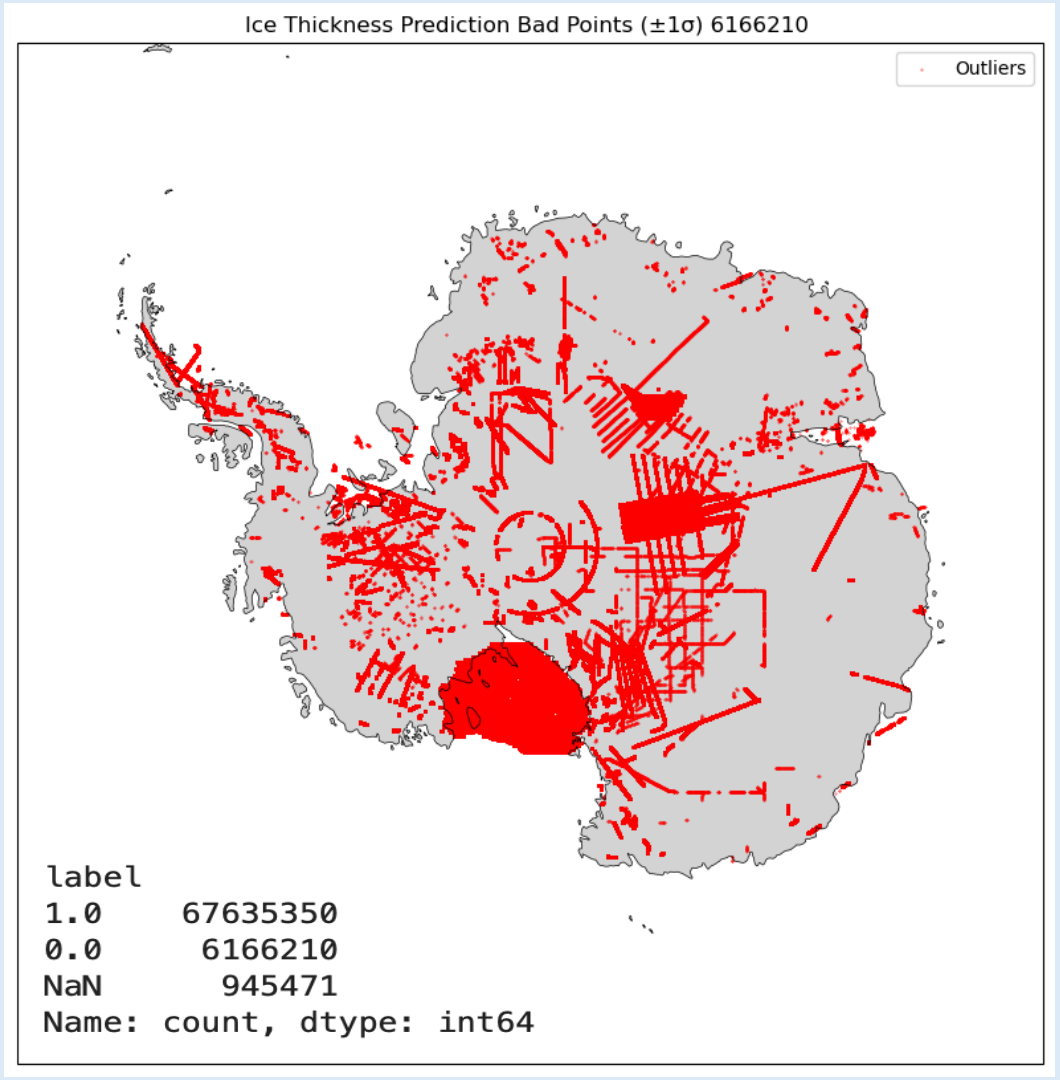
# Method 2: Labelling by Track Standard Deviation



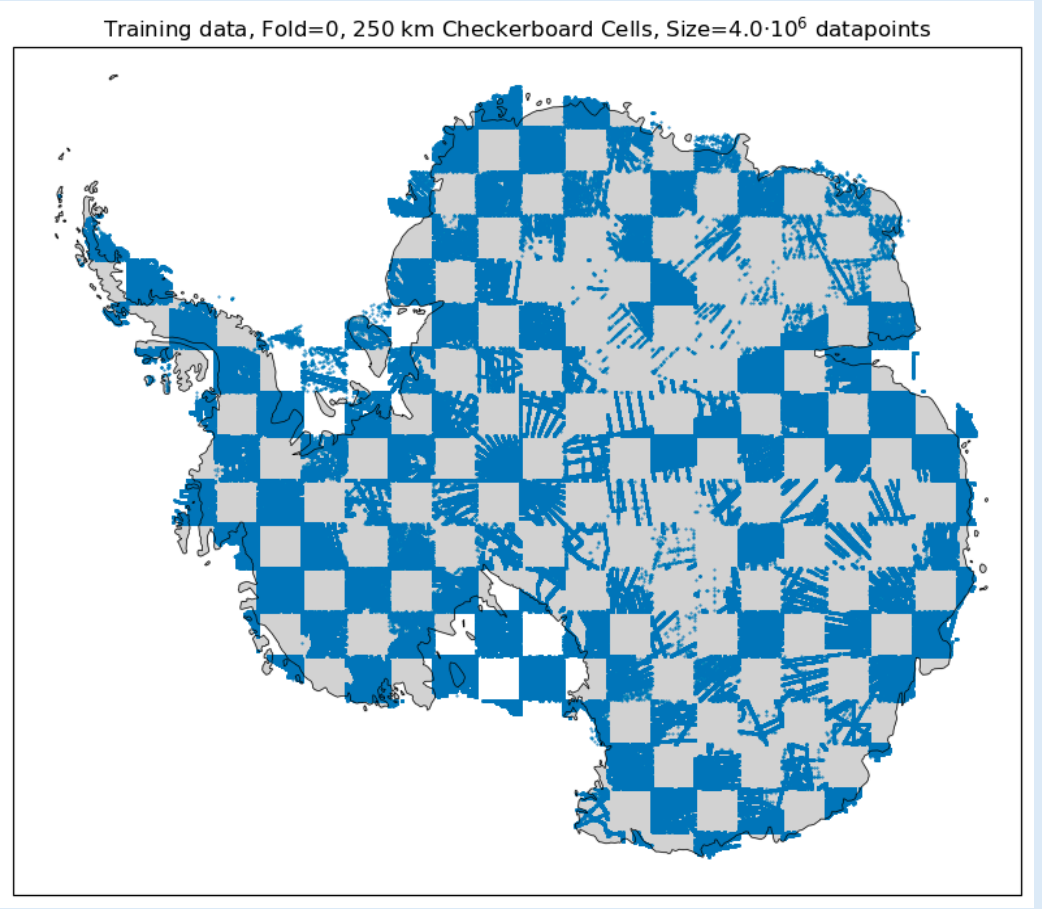

- From stats of track crossings,
  - 1-2 crossings (6.466 tracks)
  - 3-4 crossings (3.497 tracks)
  - 5 plus crossings (20.913 tracks)
- Distribution of median height difference
- Outside 1 standard deviation labelled as bad
- NaN for tracks w/o



# More Sophisticated Predictions: Regression

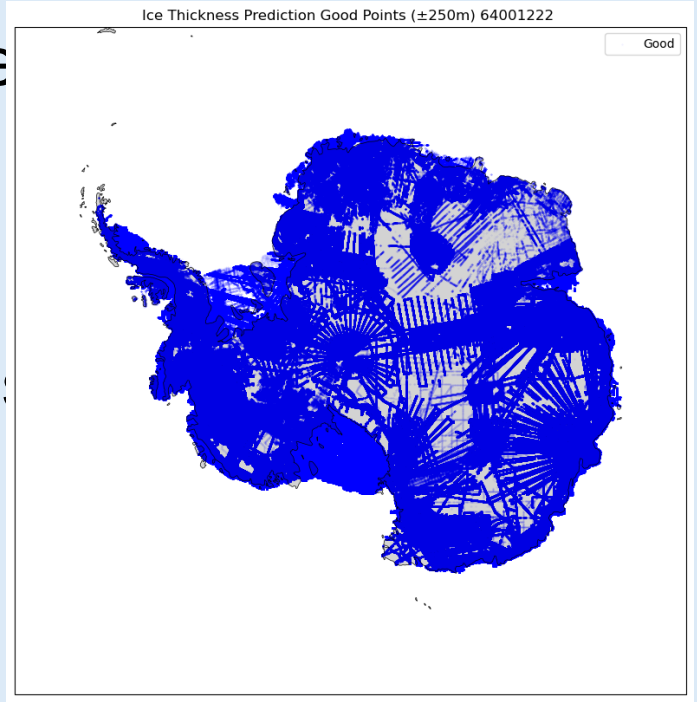
# Preprocessing: Grid for Cross Validation

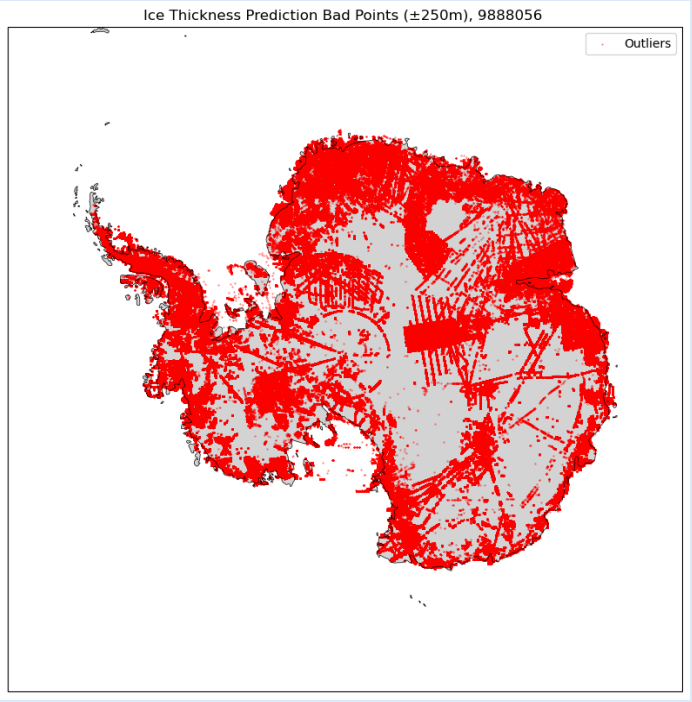
- Transform data into Antarctic Polar Stereographic coordinate system
- Make 250km grid
- Separate data into alternating folds
- Rows with NaNs in v, vx,  vy, smb deleted (50k)
- Train and validate with one fold: predict on the other
- Use sample of training fold for compute reasons (4mil/1mil)
- Plot to check for even spread



# Method 3: XGBoost Ice Thickness Prediction - grid fold

- Take 5 million random datapoints from 1 fold and split into train/ validation (4mil/1mil)
- No further preprocessing ne (BDT)
- Predict Ice Thickness: **regression task**
  - Calculate difference to mea values
  - Use difference threshold to label bad values



# Method 4: PyTorch

Test NNs on data, find issues and strengths  (reevaluate variables, try normal)

Optuna is used for optimization

Computations:

- Standardize all data:
  - Replace -9999 with column median.
- Lat, lon are cyclically encoded
  - -179 degrees is close to 179 degrees in real space but looks far away for the NN
- Distance to pole is computed
  - Proxy for close to water

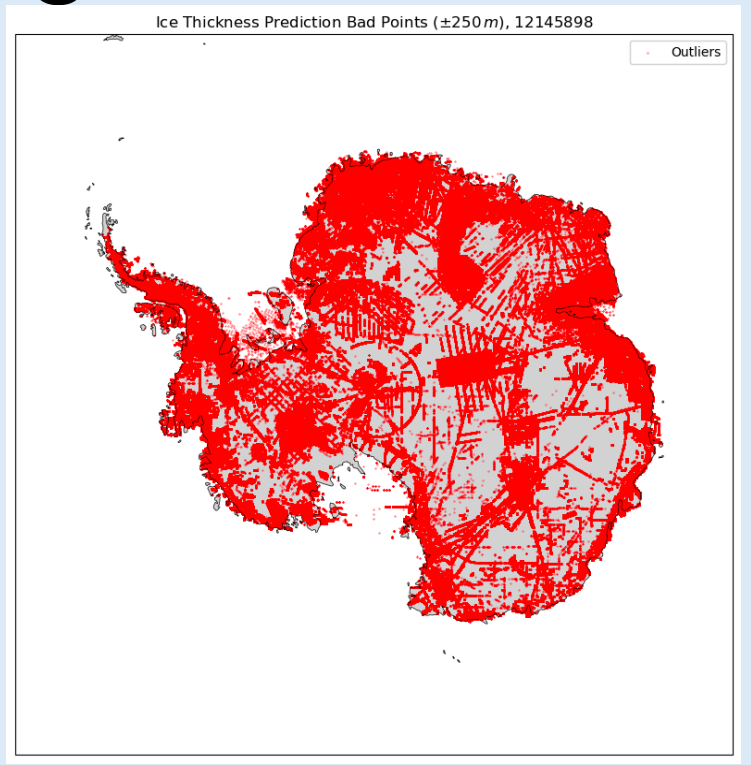
Uses two-fold cross-validation

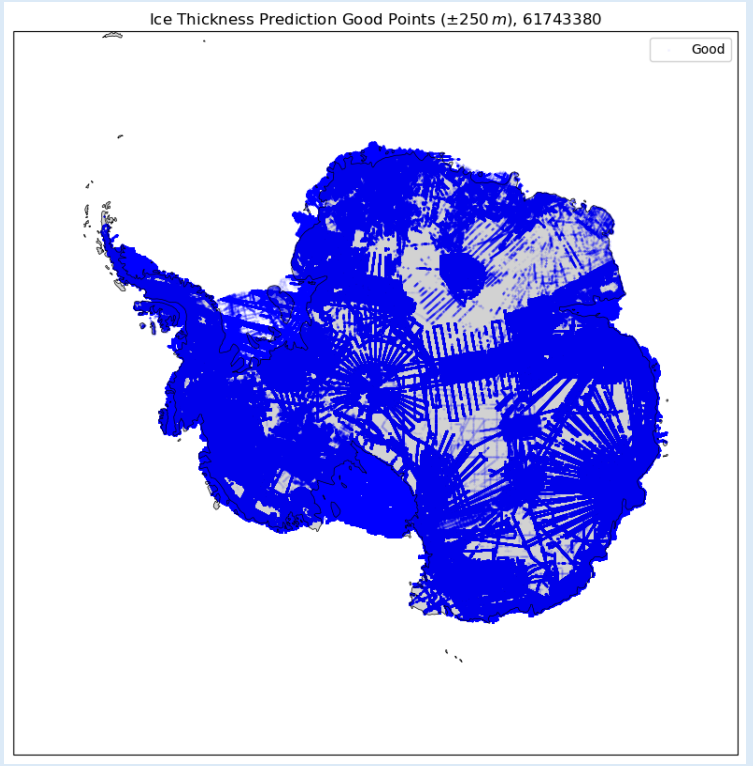
Predict ice_thickness in log

# Method 4: PyTorch Prediction - grid fold

MSEloss trained model

- 61 million "good" points
- 12 million "bad" points



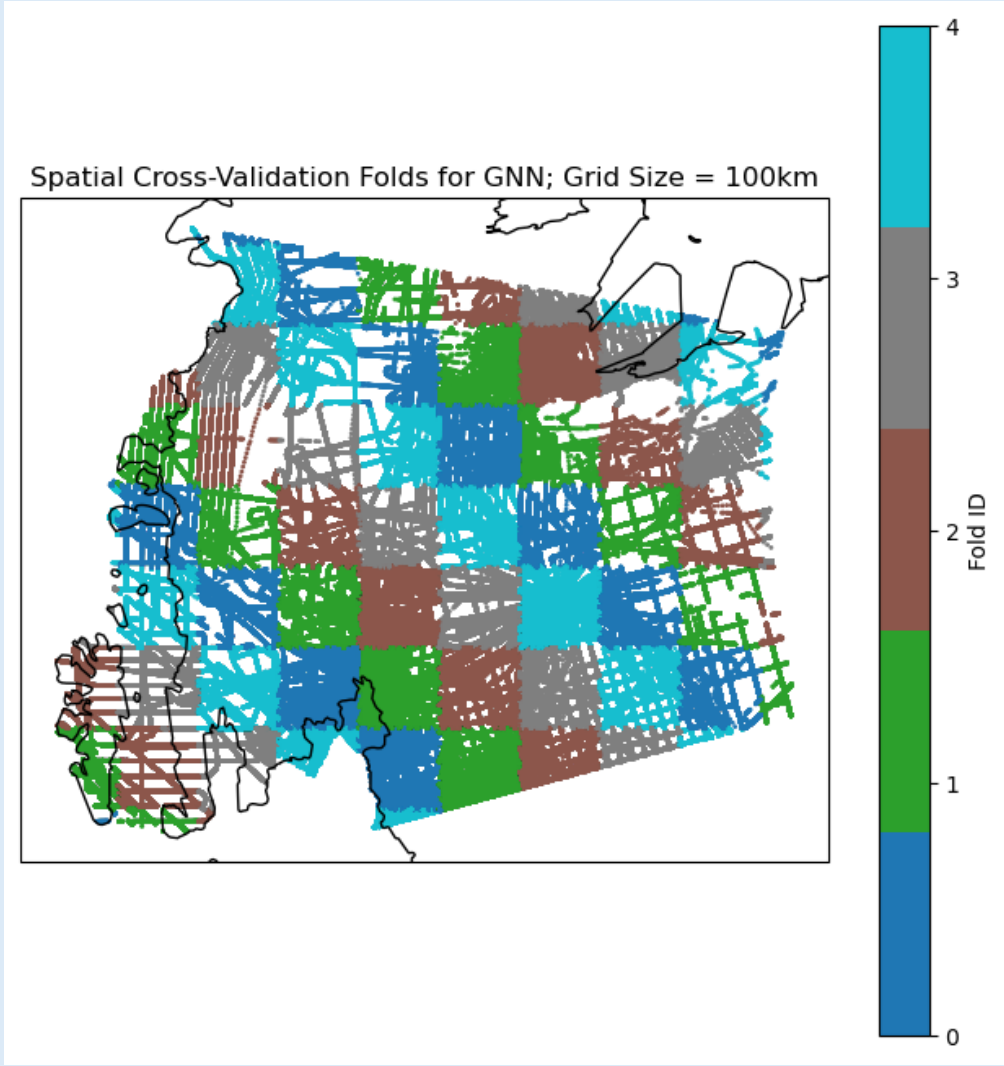
# Method 5: Graph Neural Network - GraphSAGE

- **Idea:** Take into account neighboring points to predict ice thickness
  → Take into account the 8 spatially nearest neighbors

- **Problem:** Data way too big!
  → Use subregion!
  - 7 Mio Nodes
  - 59 Mio Edges

# Method 5: Preprocessing and Training

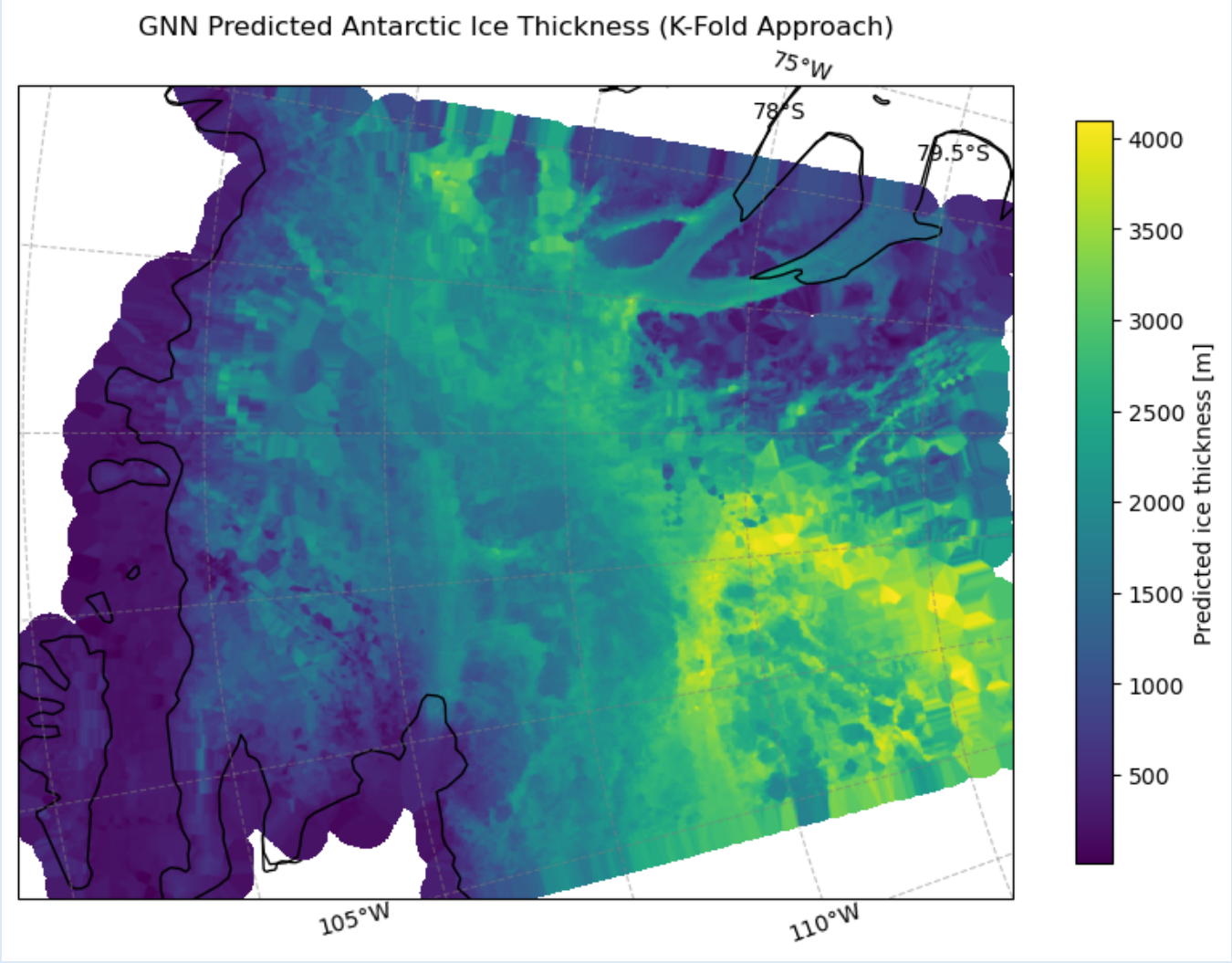
- Trained Model on 4 Folds and predicted the 5th one
- Mini-Batch-Training using NeighborLoader not to blow up the memory



# Results Comparison w/ 2 Standard Deviation

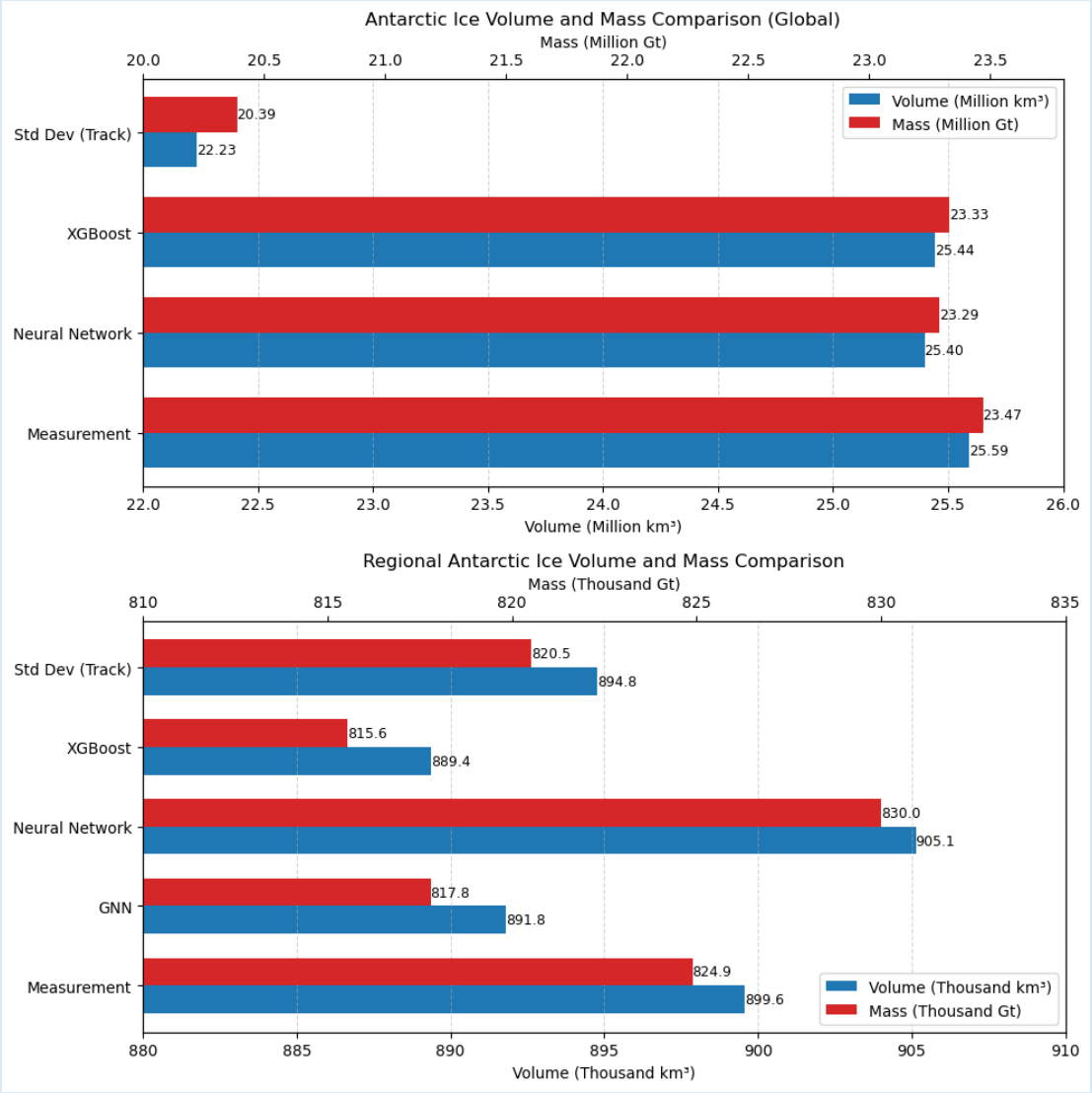
# Ice Thickness/Volume Through KNN-Mean

- Split prediction data into 1.8x1.5 km grids
- Calculate 16NN-mean ice thickness and multiply by grid area
- Sum over all grids
- For outermost point interpolate 25 km out



# Calculation of ice sheet volume

- Differences in order of 1%
- Regional scale
  - $NN_{mass}$ > $Measurement_{mass}$
- Global scale
  - $NN_{mass}$ < $Measurement_{mass}$
- Global std dev method much lower
  - Potentially missing sea ice



# Learnings from Lots of Data + Improvements

**Learnings: Memory**

- Parquet: speed, size, slicability
- DASK Dataframe (dd) alternative to pandas, with later compute
- DMatrix for optimization, xgb-inbuilt
- Batching
- Save everything
- One kernel at a time

**Future improvements**

- GNN on all data
  - Improved hyperparameter selection
- BDT regression training on all data
- Run on more features
- Feature selection specifically for regression

# Appendix

We solemly swear we all contributed equally

| Method | Description | RMSE on full data [m] | STD of Pred. & Measurent Dff erence [m] | Number of Bad Points (at 250m) | Note |
|---|---|---|---|---|---|
| XGBoost Classification | Trained from points of hand-labelled tracks. Worst and best 2500 tracks with 5 or more crossing according to median crossings at track. Optimized with Optuna. Prediction: 0-1. | - | - | 61mil (full) | Terrible algorithm, predicts r points as bad, trained on extremes, does however wo individual points |
| Labelling by standard deviation | Labeling by tracks, split tracks by number of track crossings (1-2, 3-4, 5+), label all outside 1std as bad | - | - | 6.2mil (full) | Fine estimate of quality, ver quick solution, highlights ce regions as bad, cannot adre tracks with no crossings |
| Ice Thickness Prediction with PyTorch Regression Random points | Trained 2 million random points. No optimisation done. Difference between prediction and measurement is computed. | 367.68 | 352.94 | 32,5 million | Really poor performance wh might be because we're trai on bad data. |
| Ice Thickness Prediction with PyTorch Regression Good points | Trained 2 million points "hand labelled" as good. Optimisation done using optuna, which made the NN a lot wider than before. Difference between prediction and measurement is computed. | 180.32 | 179.37 | 7 mil | Clear improvement compare the model that trained on random points. An issue wit approach is that we definite trained on datapoint and predicted on datapoint with same track_id. This will alm be the same as if we predic on our traning data. |
| Ice Thickness Prediction with | Used 2 cross fold validation by splitting the entire dataset into a grid. From one fold 5 million points | 1481.29 (SmoothL1 | 1480.39 (SmoothL1Loss | 17mil (SmoothL1Los | The models performance worsened a lot. This might h |

# Research Question:

To study and make predictions regarding **climate change and sea level rise**, we need to **track trends in Antarctic ice**.

However in the **decades of collected data** from many different exploration flights there are **outliers** that distort our predictions.

The goal of this project is to **flag outliers in ice thickness**, e.g. by providing a **score from 0 to 1** in the BedMap dataset of ice thickness in **Antarctica.**

# Tabular data: 75 Mio. Rows/Datapoints

Basic data set

## 75mil rows x 19 columns

- **48k Tracks**, premade from raw data
- Up to 64mio **"-9999"** in some columns

Climate data set

- **9 more features**
- Ca. 50k **NaNs** rows

**Features:**

Basic Data Set

- spatial (lat, long, east, north, heights)
- time (data, time, year)
- administrative (flight id, track nr., track method, file num)

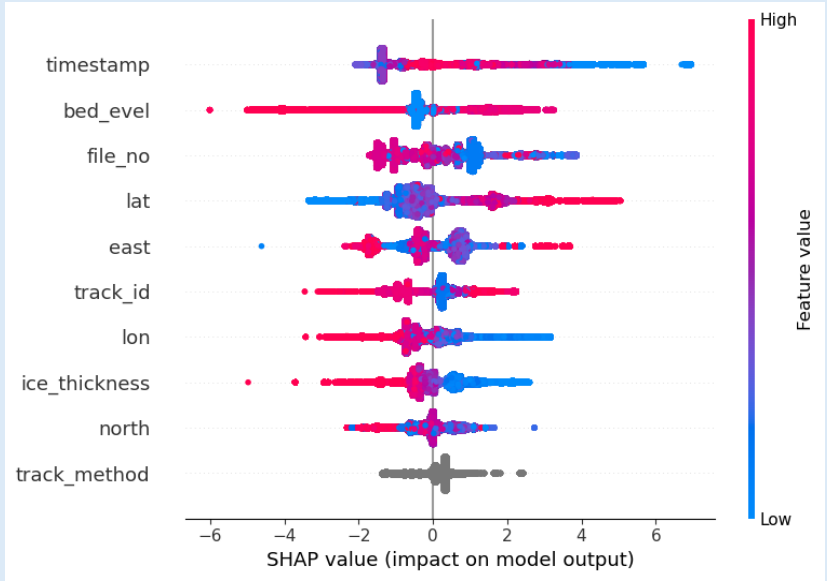
Climate data set

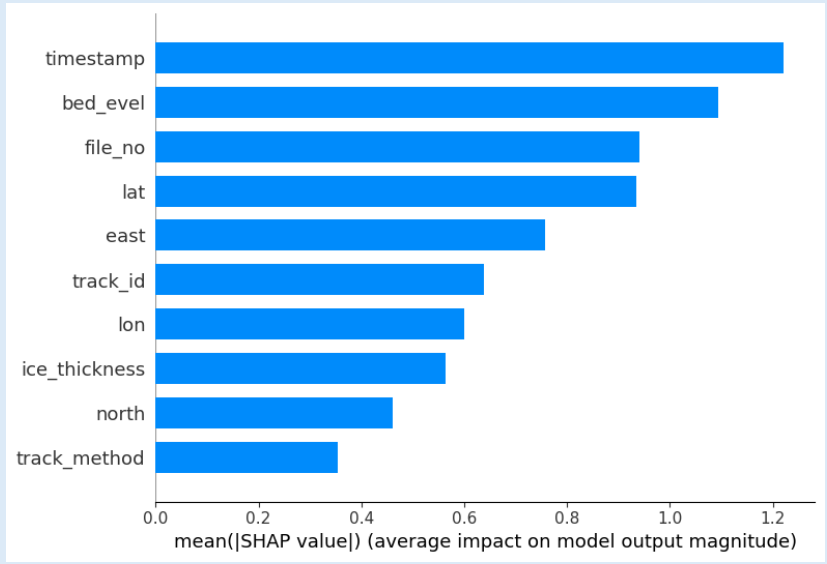
- Same as tracks plus:
- velocities (x, y, total)
- temperature, modelled ice thickness, surface mass balance, slope, vertical height

# Preprocessing: Feature Selection

## Exclusion of non-numeric features

| | variable | mean_importance | std_importance | mean_logloss |
|---|---|---|---|---|
| 0 | date | 0.105608 | 0.000201 | 0.170096 |
| 1 | point_id | 0.102662 | 0.000140 | 0.167149 |
| 2 | file_no | 0.073051 | 0.000211 | 0.137539 |
| 3 | lon | 0.058504 | 0.000113 | 0.122991 |
| 4 | ice_thickness | 0.046984 | 0.000082 | 0.111471 |
| 5 | lat | 0.045603 | 0.000115 | 0.110091 |
| 6 | bed_evel | 0.029782 | 0.000094 | 0.094269 |
| 7 | year_end | 0.025653 | 0.000104 | 0.090140 |
| 8 | year_start | 0.012963 | 0.000071 | 0.077451 |
| 9 | datetime | 0.007625 | 0.000045 | 0.072112 |
| 10 | file | 0.006869 | 0.000048 | 0.071357 |
| 11 | atd | 0.005321 | 0.000044 | 0.069808 |
| 12 | flight_id | -0.000132 | 0.000002 | 0.064356 |
| 13 | time_utc | -0.000475 | 0.000006 | 0.064013 |



# Preprocessing: Variable Selection

- Exclusion of non-numeric variables:
  - "flight_id"
  - "file"
  - "date"
  - "time_utc"
  - "datetime"
- Further variable reduction using permutation, shap and XGBoost feature ranking, dropping:
  - "atd"
  - "point_id"
  - "year_start"
  - "year_end"
- Stripped from climate dataset:  "z" (NaNs), "ith_bm" (proxy for target)

# Method 1: Quick and dirty , Classification

- BDT, due to –9999 points (XGBoost and Light GBM)
- Hand labeled training data from track crossings:
  - Only tracks with 5+ crossings, label top and bottom 2500 tracks according to median diff at crossing
  - split into train, val and test, keeping tracks separate
- Variable evaluation using permutation, shap and inbuilt XGBoost feature ranking, discard 4 more variables (for ease to compute)
- Optimized with optuna
- Predict track quality for full data: 0-1

# Method 1: Results and problems

Strengths:

- Track_id is not most important variable, model potentially picks up on other indicators of quality
- Applied to individual points: can make predictions even for tracks without crossings

Problems:

- Only 13 million good points, 61 million bad points
- Only trained with extremes
- Only trained on 5+ crossings
- Training data suggests that quality is a track property, which is not necessarily the case

# Method 2: Labelling by Standard Deviation

- From stats of track crossings, data is split by number of crossings:
  - 1-2 crossings (6.466 tracks)
  - 3-4 crossings (3.497 tracks)
  - 5 plus crossings (20.913 tracks)
- Distribution of median height difference plotted
- All data outside 1 standard deviation labelled as bad
- All data within labelled as good

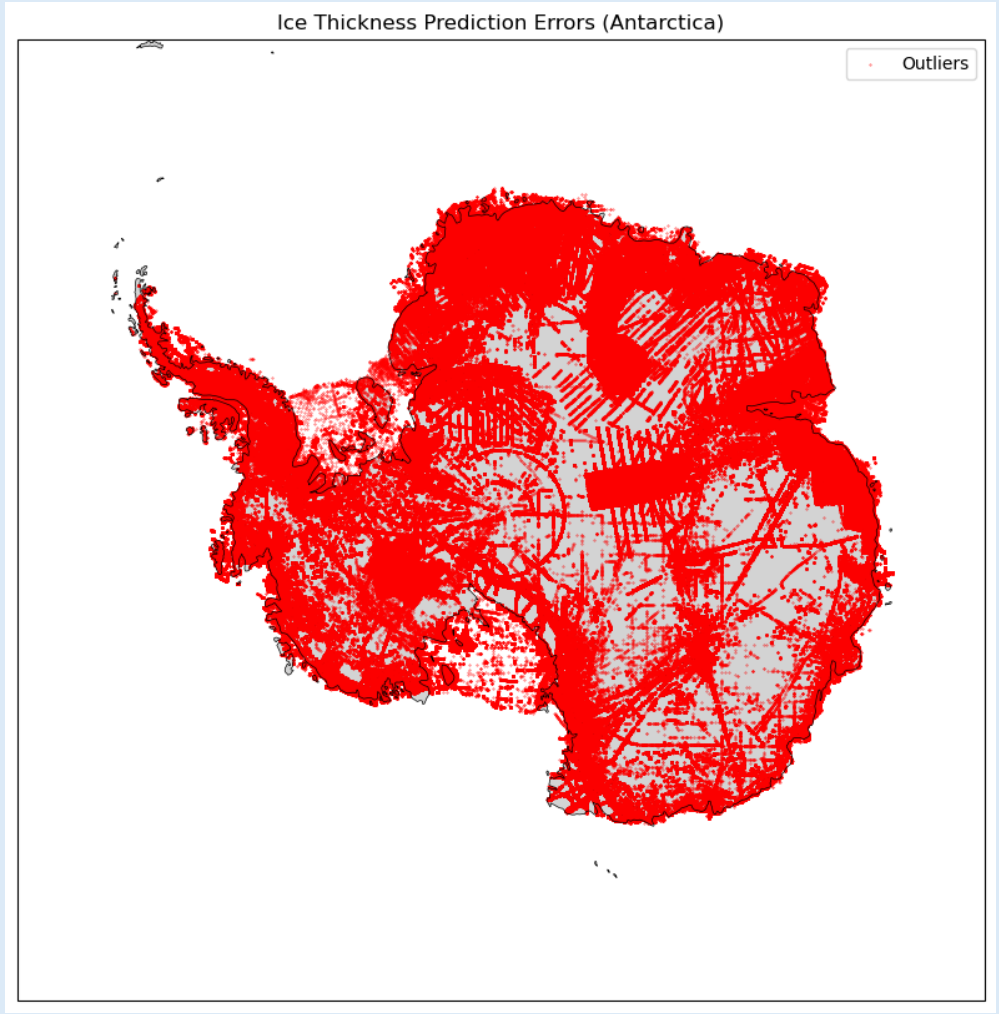
```
label
1.0    67635350
0.0     6166210
NaN      945471
Name: count, dtype: int64
```

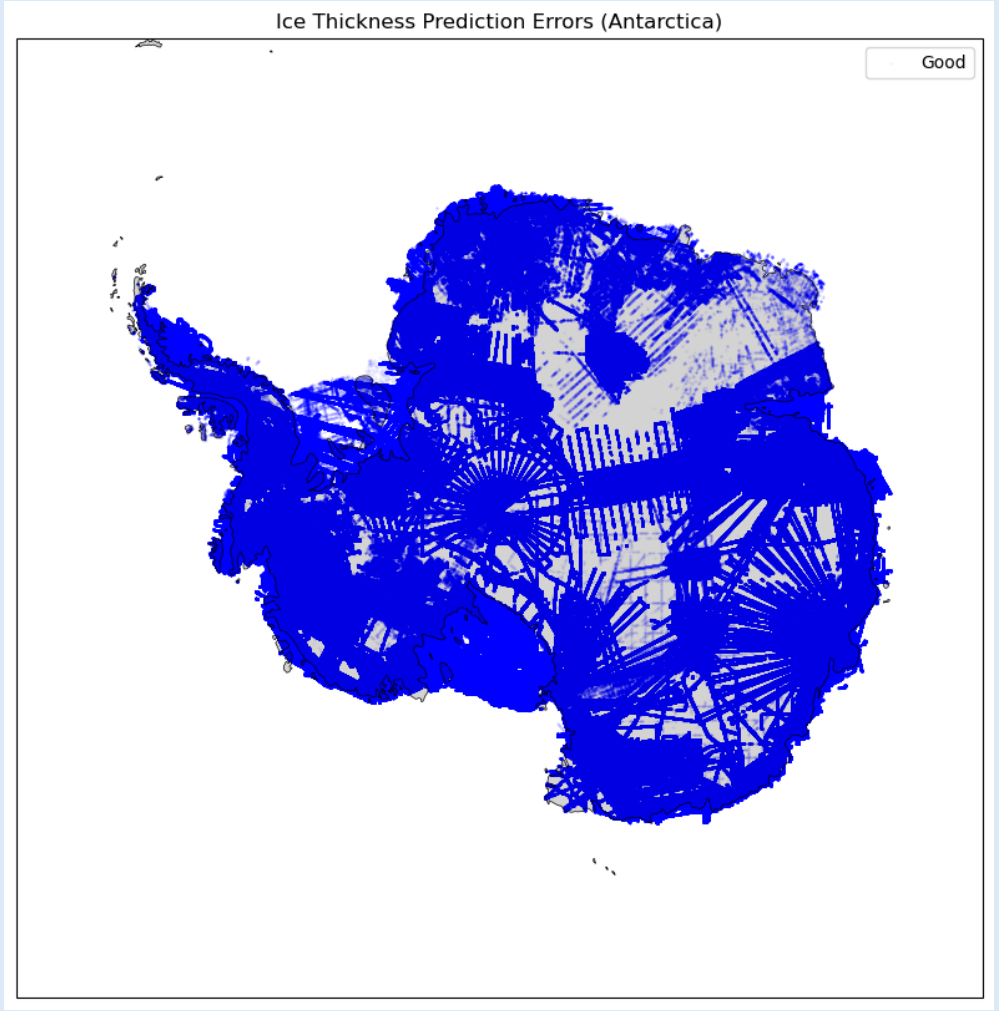
# Method 3: XGBoost Ice Thickness - random good points

- Take 2 million good points (from standard deviation method)
  - Split into 1.4mil train/ 300k val/ 300k test data
- No filtering of NaNs/ log of calue/ further preprocessing needed
- Predict Ice Thickness: regression task
  - Loss: root mean squared error
  - Remove "label" (and all other previously removed variables)
  - Optimize with Optuna
  - Predict on full data
  - Final rsme full dataset: 93.361880
- Compare with measured values
- Use difference to determine bad values

# Method 3: XGBoost Ice Thickness - random good points

## Points classified by 1 Standard Deviation 93.3

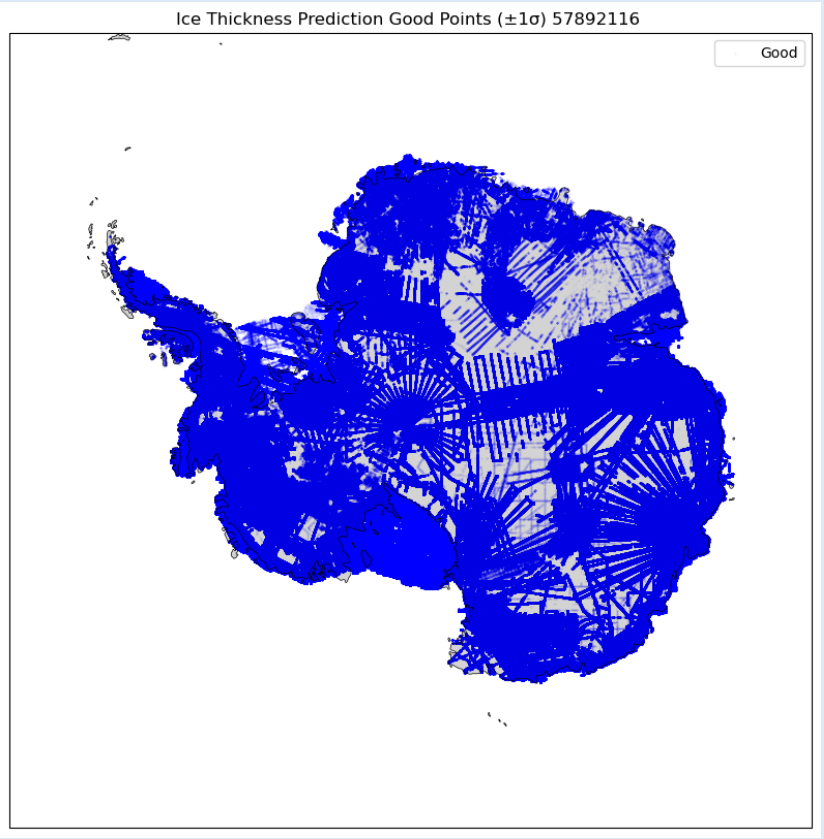


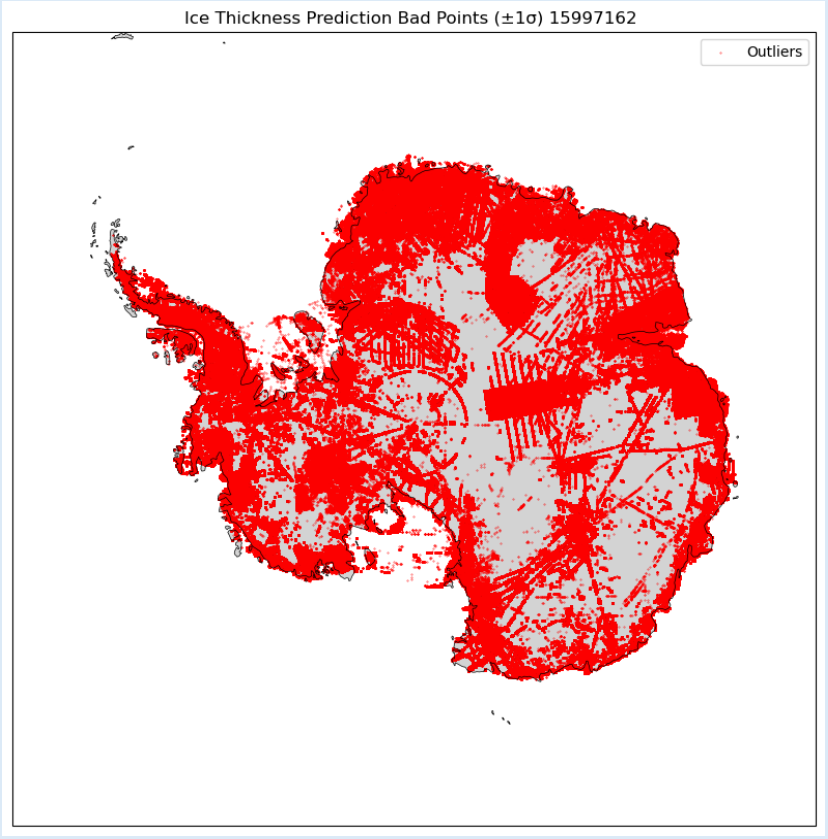



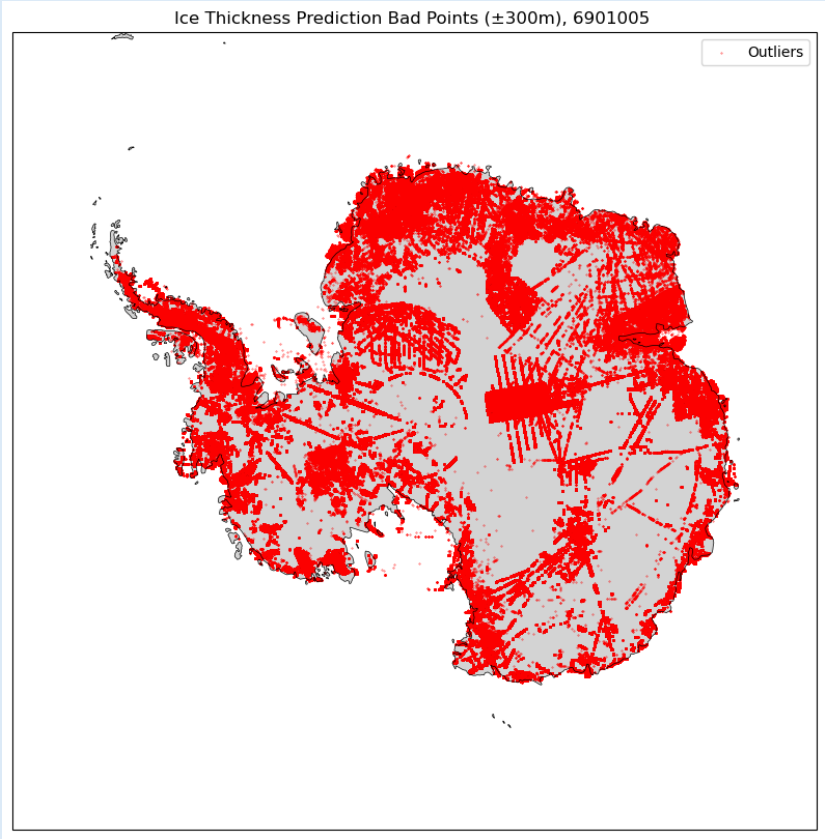
# Method 3: XGBoost Ice Thickness Prediction - grid fold

- Take 5 million random datapoints from 1 fold and split into train/ validation (4mil/1mil)
- No further preprocessing needed (BDT)
- Predict Ice Thickness: **regression task**
  - Loss: root mean squared error
  - Use model optimised for ice thickness prediction
  - Predict on other fold
  - Final RMSE full dataset: 192.76 m
- Calculate difference to measured values
- Use difference threshold to label bad values

# Method 3: XGBoost Ice Thickness Prediction - grid fold



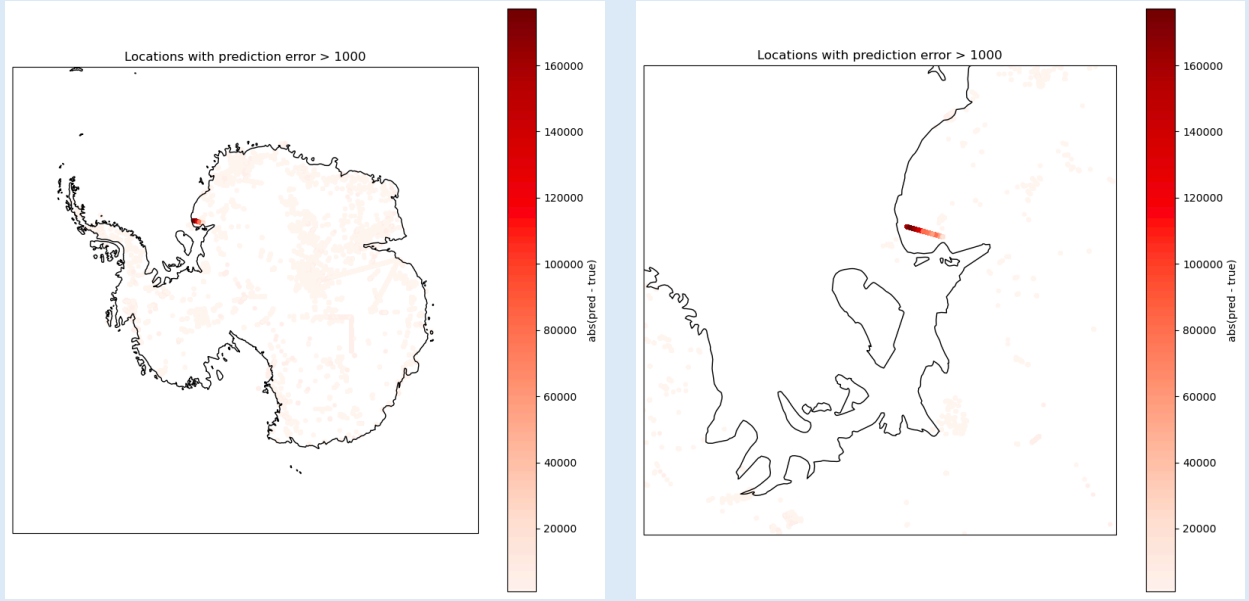



# Method 4: PyTorch - Good labels

- Here we used 2 million datapoints which we labeled as good
- First with criterion = SmoothL1Loss
  - Optuna optimized
    - Returned
      - 4 layers,
        - 'h0': 329
        - 'h1': 263
        - 'h2': 370
        - 'h3': 249
      - 'lr': 0.0010773129035919651
      - 'wd': 2.8054845679972438e-06
  - Ended up giving a std on difference between prediction and measured value at 179 m

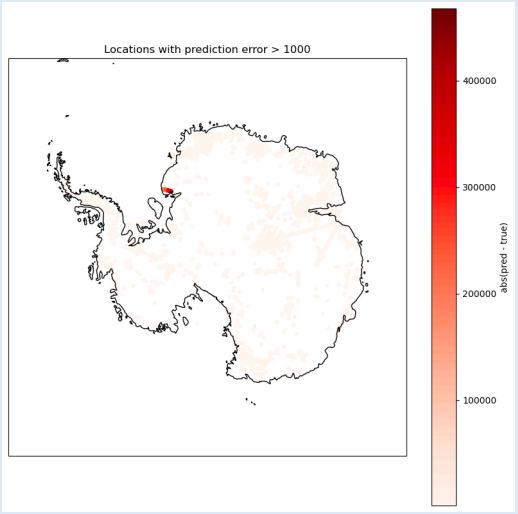
# Method 4: PyTorch - grid fold

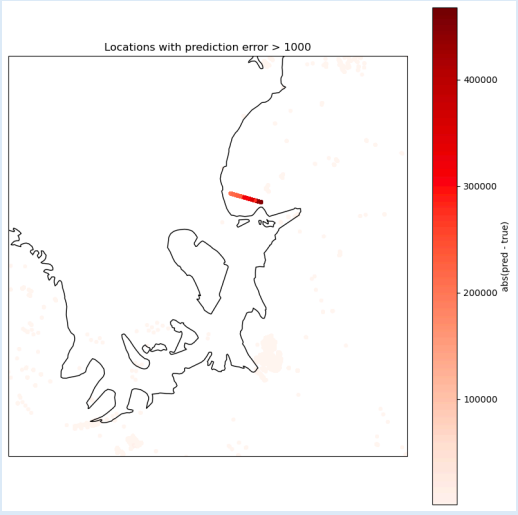
- Here we used the optimized model from the good label, but trained two models (one with each fold).
- Doing so we were able to predict on the entire dataset
- Ended up giving a std on difference between prediction and measured value at 1480 m



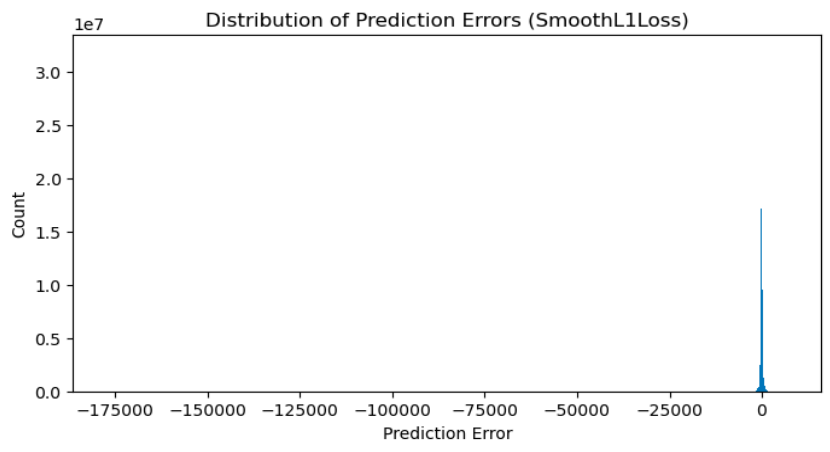
# Method 4: PyTorch - grid fold

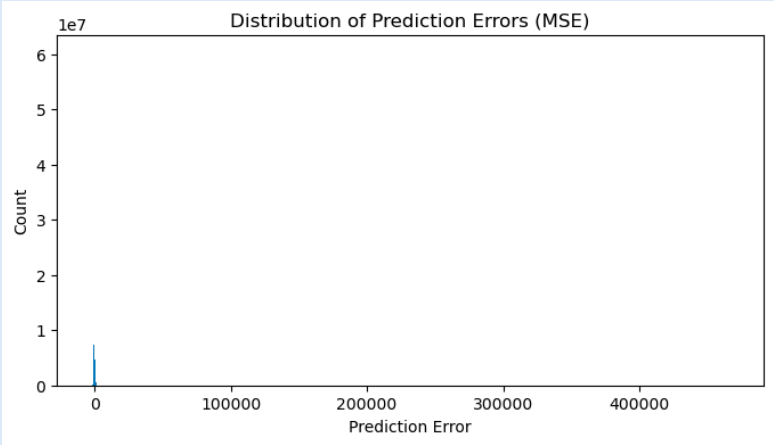
- Different approach used because we got really high std. MSEloss should penalise big errors better.
  - First with criterion = MSEloss
  - Optuna optimized
    - Returned
      - 5 layers,
        - 'h0': 208
        - 'h1': 476
        - 'h2': 397
        - 'h3': 332
        - 'h4': 496
      - 'lr'=0.001297088179561048,
      - 'wd'=0.00033852771377530154
      - Ended up giving a std on difference between prediction and measured value at 4088m
        - From the plot it is clear that it has a really hard time predicting on one specific track.
        - If we compare it with the plots in the previous slide, we see that the two models struggles with some of the same tracks/points.

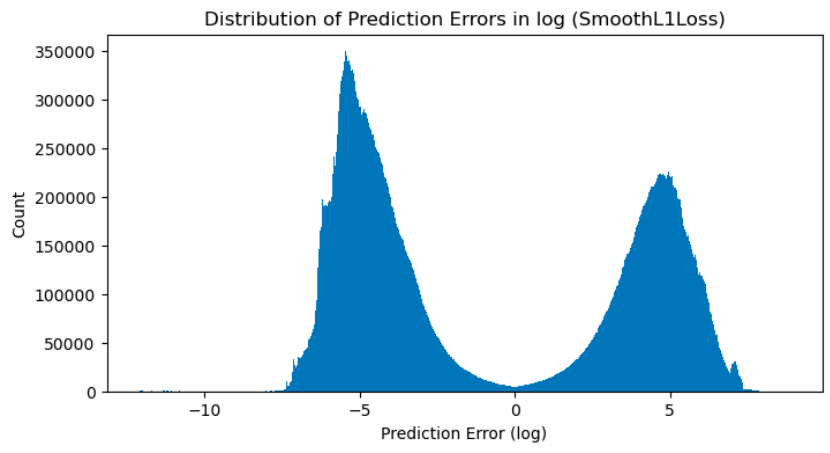


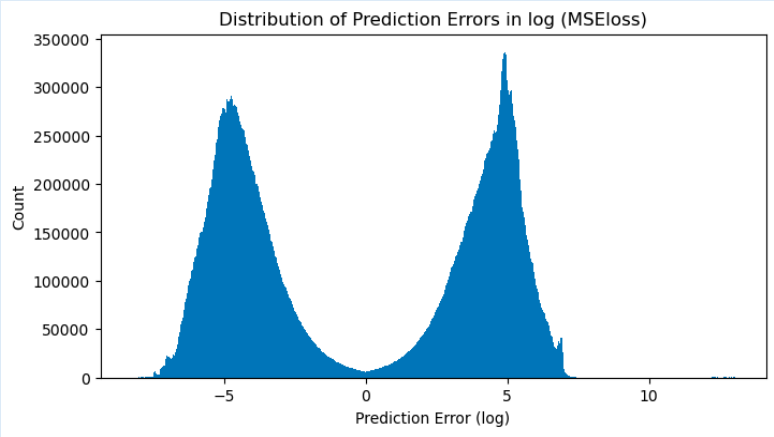



# Method 4: PyTorch - grid fold Histograms




Here we see that most of the difference are within a reasonable range, but we have some extreme outliers.

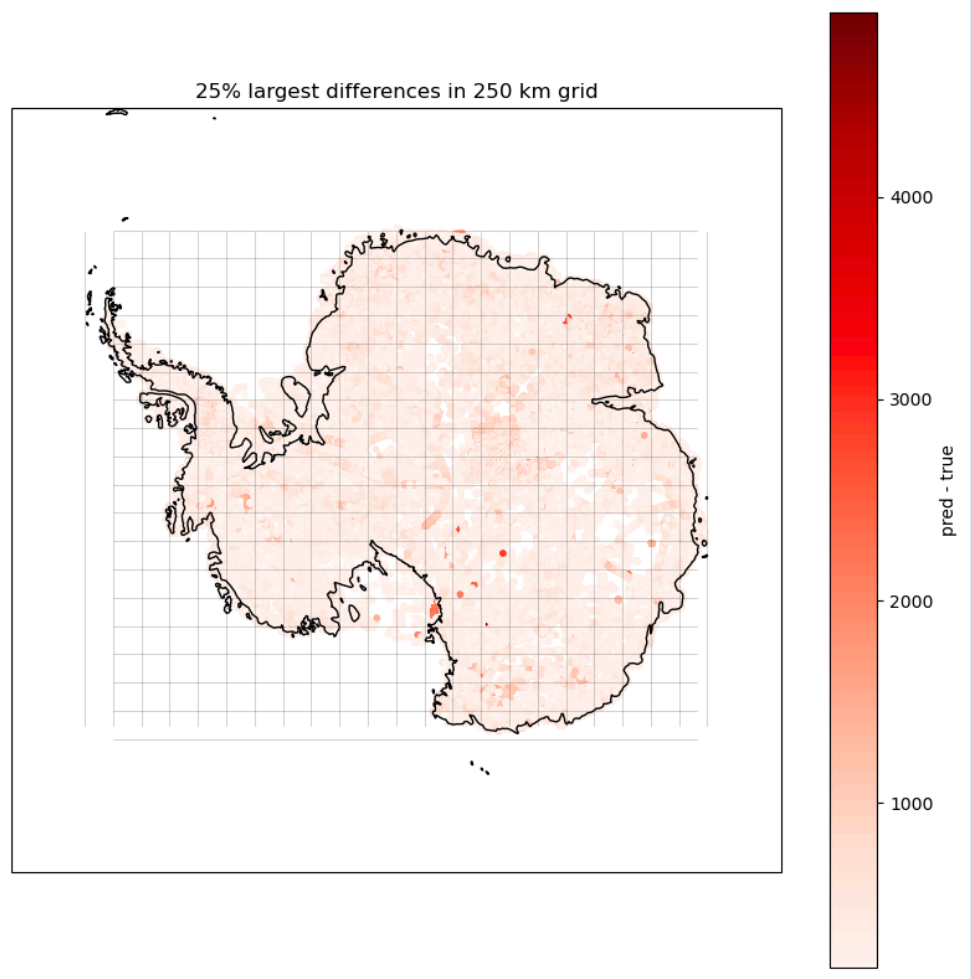
If we print the numbers of differences above 10.000 meters, we see that number is the exact same for the two models (12.771 datapoints)

This tells us that the model probably predict the same datapoints wrong, but the regression output is very different.

We suspect that this is due to the model seeing 12.771 datapoint with a very unique entrance, which the model never saw before in its training.

# Method 4: PyTorch - grid fold

- Remove 12771 datapoints for the predictions made by model with (MSEloss)
  - All differences over 10.000 m
- Looks similar to XGBoost results
- When doing that we end up with a std of 232 m
- We also see that the wrong data is not piled up in the center of each grid, which you could suspect to happen, since there could be unique data entries that the training did not include.



# Method 5: Graph Neural Network- Memory Saving

NeighborLoader was used to sample mini-batch subgraphs of 256 target

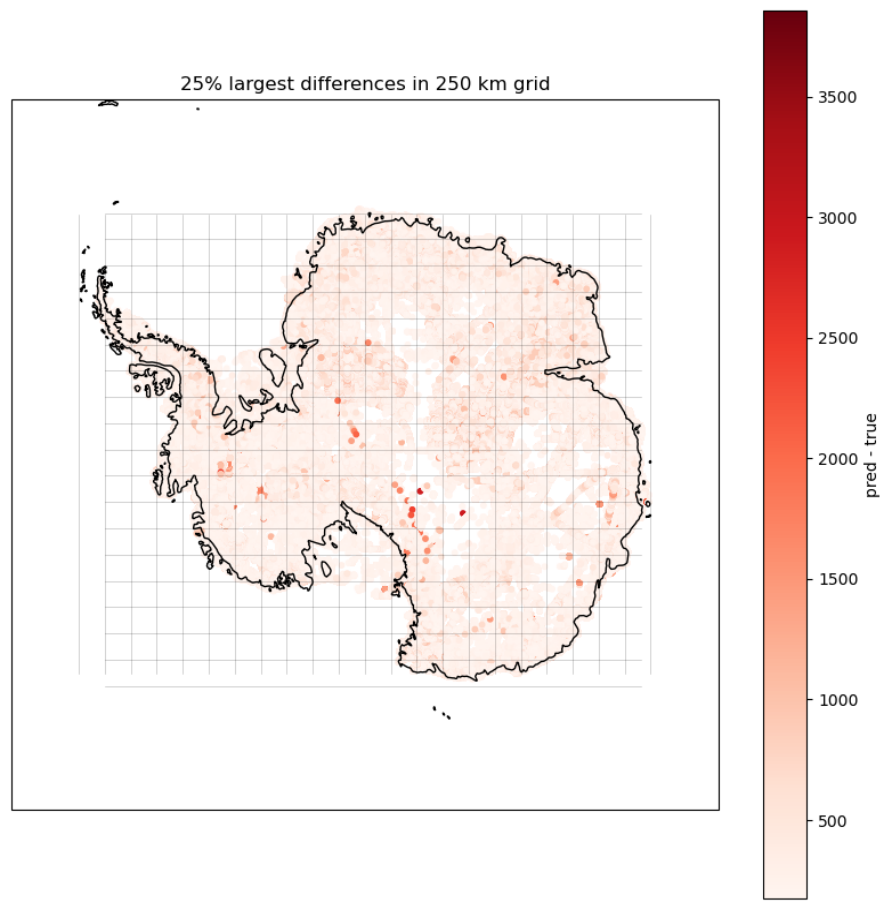
nodes, including their 2-hop neighborhoods by selecting up to 8 neighbors

at the first layer and 10 neighbors at the second layer for efficient

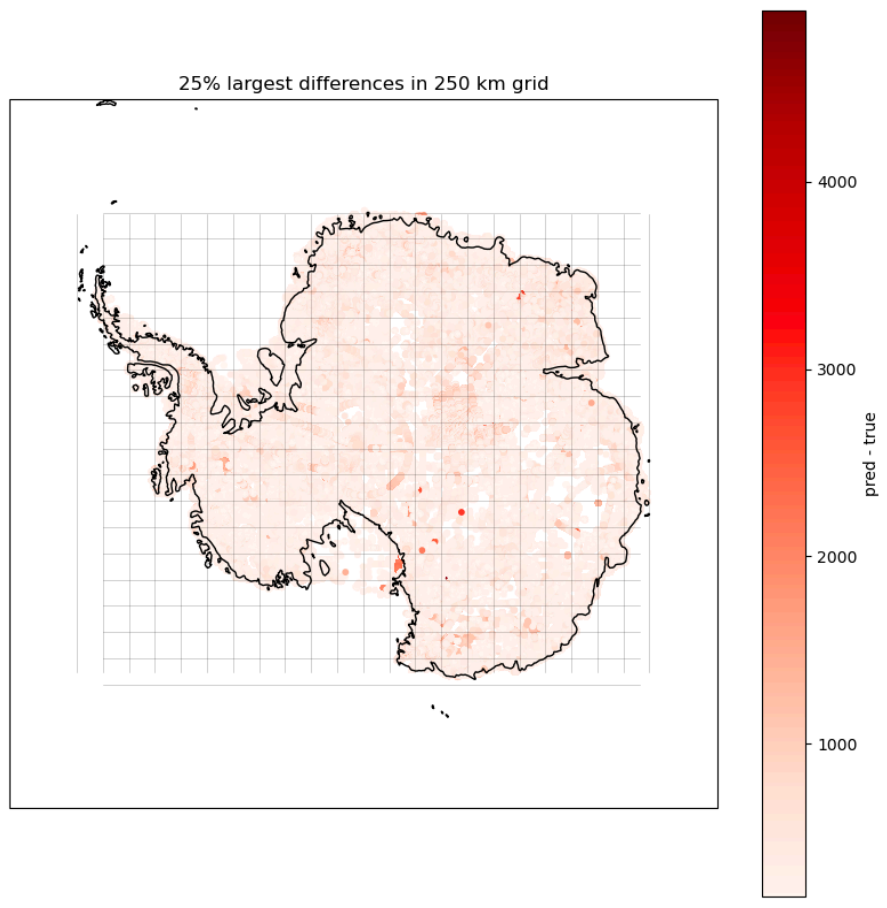
GraphSAGE training.

# Comparison

## XGBoost (BDT)




## PyTorch (NN)



# Results Comparison Absolute Error

# Results Comparison w/ 3 Standard Deviation

# Points classified with diff 10% (167m), diff 250m and diff 500m