Praktikum 1

Unsmoothing of AtmoRep Nowcasting Images to locate heavy-tails

Verfasser

Stefan Eger

Wien, 2023/2024

Studienkennzahl lt. Studienblatt:	UA 066 921
Fachrichtung:	Informatik
Betreuerin / Betreuer:	Univ.-Prof. Dipl.-Inform.Univ. Dr. Claudia Plant & Dr. Irene Schicker

Contents

Abstract

In the past year and months, data-driven machine learning models like PanguWeather or GraphCast gained a lot of attention and are even used experimentally at the European Center for Medium Range Weather Forecasting (ECMWF). Despite these models showing very good skills and being able to provide multiple forecast parameters, they are unable to deliver non-underdispersive probabilistic forecasts and underestimate extremes. The main focus will be on Nowcasting, which is the forecasting of short term meteorological data such as temperature into the future. The overarching goal is to create better forecasts using predictions from the trained AtmoRep model. Three milestones are planned to be achieved in this project:

(i) Getting familiar with AtmoRep: implement the code and use the ERA5 data

(ii) Adapt the code the work with data from GeoSphere Austria (formerly ZAMG): this includes retraining and entail the fixing of compability issues with GeoSphere training data and AtmoRep training data.

(iii) ~~Heavy tails / smoothing: this objective includes unsmoothing of the heavy tails (highest and lowest 10 % of weather extremes) of a resulting forecasts in order to see where weather extremes are located. This is an improvement to an overly smooth picture, which would obscure such extreme outliers. One way to achieve this is to modify the loss function, which is currently in AtmoRep a distance function between the instationary data distribution and the distribution modeled by AtmoRep, with an Monte Carlo estimate over the training data. Hopefully the modification of the loss function will yield more useful unsmoothed images.~~

1 Introduction

Currently AtmoRep [10] has only been trained with ERA5 Data[7]. The goal was to use a different data set, which is much more fine grained, in order to make better forecasts. COSMO REA 6[5] (AtmoRep is already able to process ERA5 data, more on that later 2) hourly data contains much more information than standard ERA5 data, thus it was chosen to be trained with AtmoRep. In particular we were interested in the following domain 4-22 longitude and 40-55 latitude, containing Austria and its surrounding neighbor states. After the initial setup of the AtmoRep Project, the Code had to be modified to access the COSMO Data, which has turned out to be much more difficult than anticipated 4. All changes which have been made, can be found in the following Git Repository [1]

2 COSMO - REA6 Setup

2.1 Download Data

COSMO - REA6 hourly 3D Temperature (T), water vapor content(Q), turbulent kinetic energy(TKE), wind velocity u and v direction (U) (V) and at last 2D data with total precipitation (TOT_PREC) from 1995-2018 has been downloaded from [6]. Here is an example command fetching the year 1995 for T:

```
wget -r --no-parent -nH --cut-dirs=5
    --reject "index.html*"
    https://opendata.dwd.de/climate_environment/REA/COSMO_REA6/hourly/3D/T/1995
```

The cut-dirs=5 is useful for achieving cutting away unnecessary overhead folders (Useful to keep in mind if Shell scripts have to be adapted).

2.2 Rotate Grid

COSMO-REA-6 Grid is rotated at the time of download and has to be rotated back, in order to cut out the domain 4-22 longitude and 40-55 latitude. For this task CDO 2.3.0 (Climate Data Operators) [3] has been used and the tutorial provided by COSMO has been followed[4]. The following 2 cdo commands have been used to achieve the goal mentioned before:

```
cdo -f nc4 -z zip=9 -copy -setgrid,griddes_REA6.txt <infile> <outfile>.nc4
cdo sellonlatbox,4,22,40,55 <infile> <outfile>.nc4
```

In the first cdo line you will note the griddes_REA6.txt, which is necessary to rotate the grid correctly. It can be found in the tutorial [4] or `https://opendata.dwd.de/climate_environment/REA/COSMO_REA6/constant/griddes_REA6.txt`. The first line rotates the grid with the given text file, but also ensures that the output is a netcdf (.nc4) file. The previously downloaded tar files contain .grib files which are the input files. This will become important later on for the AtmoRep code, because grib files are used by default and adjustments have to be made. The netcdf format has the advantage that the data can be verified for correctness via ncview [9]. Here is a small visualization how the process worksB.1 B.2. This process has been automatised with A.1 and is by far the most computational intensive process. It took around 20-22 minutes to process a single year, (even tough it is already parallelised) with 32 Threads at a time.

2.3 MergeTime hours to year-month

Each file, after using the first shell script A.1, should reference an hourly snapshot within a year. Since AtmoRep uses year-month format to load data, all hours within a month have to be merged into a single file A.2, but how do we know which file is part of a given month. For this action the COSMO filenames will be used. The command to facilitate this action can look like this:

```
cdo -f nc4 mergetime T/1995/T.3D.199501* <output>.nc4
```

This would ensure that all files which are part of january 1995 are merged into 1 file. This process is also automated A.2. In this script the variable name will be adjusted to a more meaningful name. OLD_VARNAME=var11 represents the current name of temperature data within all hourly files. This name will be changed to VARNAME=T for better readability and ensures intuitive understanding which data is stored within the .nc4 file without having to lookup what variable represents which var code. Here is a small lookup table.

OLD_VARNAME	VARNAME
var11	T
var51	Q
var152	TKE
var33	U
var34	V
var61	TOT_PRECIP

Finally the levels contained within one monthly file, has to be separated, since AtmoRep can only process year-month files with 1 single level, but COSMO Data stores 6 levels (35, 36, 37, 38, 39, 40), except TOT_PRECIP which does only contain model level 0. This script will also store the final files in a subfolder *ml* $<$ *level* $>$, which can be uploaded into AtmoRep data folder and should be ready to be used now. Small note here, before using this script, check manually if a given year contains the first hour of the next year. The current solution is to manually copy this hour to the next year, which could be further improved by modifying the merge script A.2. TOT_PRECIP has its own scriptA.6, because its folder and file structure is different, but will effectively yield the same result as A.2.

2.4 Mean and Standard Deviation

Atmorep requires besides the COSMO data mean and standard deviation to calculate the normalization. Choosing the local normalization (per grid point basis [10] [p.13]) the mean and standard deviation is required for each month in a given year. The main problem with this normalization file, containing mean and std, is that it is a binary file and it is not documented, as of yet, what information it exactly contains. Looking at the normalizer_local.py file in AtmoRep, it was possible to reverse engineer the binary file, but first cdo is used to calculate the mean and standard deviation A.3 with the following commands:

```
cdo monmean <inputfile>.nc4 <mean_filename>
cdo monstd <inputfile>.nc4 <std_filename>
merge_mean_std <mean_filename> <std_filename> <outputfile>.nc4
```

cdo monmean and monstd calculates the mean of each month contained within the input files, which have been created in the previous steps and then both resulting files will be merged into a single file. After this step the .bin file has to be generated. The format as far as I have understood it, is a matrix containing [[[mean_00, std_00], [mean_01, std_01], ..., [mean_0m, std_0m]], [[mean_1m, std_1m], ..., [mean_nm, std_nm]]] with the matrix shape (latitude_grid_points, longitude_grid_points, 2). In other words, per grid point is one mean float and one standard deviation float stored. The generation of bin files is also automatised A.4

After all scripts have run successfully, AtmoRep should have all the data it needs, in order to train a model.

3 AtmoRep Code

3.1 Setup

After building the AtmoRep Project [2] as the README describes, it is necessary to write a shell script A.5 to make it executable on MeluXina [8]. Pytorch has to be installed manually into your pyenv folder/environment. In this script the python environment is being activated and the Python GCC and OpenMPI module are being loaded. Python had to be loaded too, to gain access to the Python 3.10.4 version. Then all GPUs in a node have to be set visible. Run sbatch scripts/<scriptfile> to start the training process.

3.2 Code Changes

To make AtmoRep compatible with COSMO data a lot of changes had to be made. Many variables which appear in the config file in train.py seem to be hardcoded and have yet to be fixed, but training COSMO data is still possible after a lot of changes. Here are the most important ones and the reasoning behind these changes.

```
def train() :
    ...
  # format: list of fields where for each field the list is
  # [ name ,
  #   [ dynamic or static field { 1, 0 }, embedding dimension, , device id ],
  #   [ vertical levels ],
  #   [ num_tokens],
  #   [ token size],
  #   [ total masking rate, rate masking, rate noising, rate for multi-res distortion]
  #   normalizer {'global', 'local'}
  #   [data_type, file_shape, file_geo_range, file_format]
  # ]

  cf.fields = [ [ 'T', [ 1, 2048, [ ], 0 ],
                [ 35 ],
                [12, 6, 12], [3, 9, 9], [0.25, 0.9, 0.1, 0.05],
               'local',
               ['cosmo', (-1, 286, 254), [[40., 55.], [ 4., 22.]], 'netcdf4'] ] ]

    ...
```

The last two field elements had to be added to make AtmoRep run the local normalizer, instead of the global one. and 'cosmo' refers to the file prefix the preprocessed cosmo files have been given. e.g: cosmo_T_y2017_m01_ml35.nc4

```
def train() :
    ...
   cf.years_train = list(range(1996, 2017)) # list( range( 1980, 2018))
  cf.years_test = [2018]  #[2018]
  cf.file_format = 'netcdf4'
   #cf.geo_range_sampling = [[ -90., 90.], [ 0., 360.]]
   #will be mapped to [[0-180, 0-180][0-360, 0-360]]
  cf.geo_range_sampling = [ [ 40., 55.], [4., 22.]]
  cf.file_shape = (-1, 286, 254)

  # training params
  cf.batch_size_test = 32
  cf.batch_size_start = 8
  cf.batch_size_max = 16
  cf.batch_size_delta = 4
    ...
```

years_train refers to how many years are supposed to be trained, but the config file year_base = 1996 and year_last = 2018 still have to be set otherwise the local normalizer will crash, since the normalizer works with both parameters from train.py and the config file. The fileshape refers to how many lat and lon gridpoints are within a given .nc4 file. The most important change is the batch size, otherwise the GPU Memory will overflow. More on that issue later4.

```
def train() :
     def __init__(self, field_info,
        vlevel, file_shape, data_type = 'era5', level_type = 'ml') :

    fname_base = '{}/normalization/{}/{}{}/normalization_mean_var_{}_y{}_m{:02d}_{}{}.bin'

    self.corr_data = [ ]
    for year in range( config.year_base, config.year_last+1) :
      for month in range( 1, 12+1) :
        corr_fname = fname_base.format( str(config.path_data),
        field_info[0],
        level_type, vlevel,
        field_info[0], year,
        month, level_type, vlevel)

        x = np.fromfile( corr_fname, dtype=np.float32).reshape( (file_shape[1],
            file_shape[2], 2))

        x = xr.DataArray( x, [ ('lat', np.linspace( 40., 55., num=file_shape[1],
                endpoint=True)),
                ('lon', np.linspace( 4., 22., num=file_shape[2],
                endpoint=False)),
                ('data', ['mean', 'var']) ])
        self.corr_data.append(x)
```

It is necessary here to change the fname_base, so AtmoRep can find the mean and standard deviation bin files. Another change is in lat and lon in x, because the domain is sadly hardcoded in the original code and has to be changed accordingly. This change alone will not solve the problem though, because the same values have to changed in dynamic_field_level.py.

```
  def load_data( self, years_months, idxs_perm, batch_size = None) :
  ...
          if self.corr_type == 'local' :
        coords = [ np.linspace( 40., 55., num=self.file_shape[1], endpoint=True),
                   np.linspace( 4., 22., num=self.file_shape[2], endpoint=False) ]
    ...
```

In order to load the .nc4 with the correct dimensions, it is necessary to add ds.squeeze('lev') into file_io.py otherwise a dimensionality error will occur, since cdo will store the level even if there is only one model level within a given file.

```
 def netcdf_file_loader(fname, field,
    time_padding = [0,0,1], days_in_month = 0, static=False) :

  ds = xr.open_dataset(fname, engine='netcdf4')[field]
  ds = ds.squeeze('lev')
    ...
```

Now the grib_index in data_loader.py has to be modified, because even though we are loading netcdf data the grib_index is still used to retrieve the Value which we have assigned to our data within the .nc4. For reference the name of the temperature data var11 was changed to T and so on.

```
    self.grib_index = {
        'T': 'T', 'Q': 'Q', 'TKE': 'TKE', 'U': 'U', 'V': 'V', 'TOT_PRECIP': 'TOT_PRECIP',
          'vorticity' : 'vo', 'divergence' : 'd', 'geopotential' : 'z', 'orography' : 'z',
          'specific_humidity' : 'q',
          'mean_top_net_long_wave_radiation_flux' : 'mtnlwrf',
          'velocity_u' : 'u', 'velocity_v': 'v', 'velocity_z' : 'w',
          'total_precip' : 'tp', 'radar_precip' : 'yw_hourly',
          't2m' : 't_2m', 'u_10m' : 'u_10m', 'v_10m' : 'v_10m',  }
```

Almost finished now. The most dangerous and necessary hotfix was in multifield_data_sampler.py, because the fileshape resolution was hardcoded and a pole_offset was used although the domain is not global.

```
# reference grid, likely has to be the coarsest
# self.res = (geo_range[1][1]-geo_range[1][0]) / file_shape[2] #360. / file_shape[2]

self.res = (geo_range[1][1] - geo_range[1][0])
self.res /= file_shape[2] if self.is_global else (file_shape[2]-1)

# avoid wrap around at poles
pole_offset = 0# np.ceil(fields[0][3][1] * fields[0][4][1] / 2) * self.res
```

These were the major changes to AtmoRep so the code is able to train cosmo data, but there are still major bugs and performance issues and those will be discussed in the next chapter.

4 Known - Problems

4.1 Performance Issues

After successfully training temperature for 1996, it became clear that using 1 GPU per task (4 tasks per node) was not efficient. It was tried to train temperature 1996 to 2018, but due to the inefficiency the model only trained for 4 epochs of 128 after 24h with 32 nodes and 4 tasks per node and 1 GPU per task.

4.2 4 GPUs - 1 Task per node bug

After adjusting batch sizes, it was tried to increase the number of GPUs per task within a node and to train the model with multiple nodes (The same setting as seen in A.5), but this causes AtmoRep to crash because the batch size has an Index out of range error. More can be extracted from the log report, which is added to the Git repository. This setting should be more efficient, if the bug is fixed.

4.3 The year before bug

Another strange bug that has to be fixed, is that if you want to train the year 1996 for example, then you need the data for december 1995 . I was unable to find the cause for this bug in reasonable time.

5 Technology Stack

- Python version 3.10.4 (MeluXina) and Pycharm to edit the code
- Pytorch version 2.0.1+cu117
- 4x NVIDIA Ampere 40GB HBM GPUs per node were used[8]
- Ncview 2.1.8+ds-4build1 was used to verify and extract useful information about the netcdf files (.nc4)
- cdo 2.3.0 was used (Climate Data Operators) to edit cosmo data and generate mean and std files.

6 Conclusion

In conclusion not every milestone could be reached due to many unforeseeable circumstances, but AtmoRep can now produce models based on COSMO data, although it still needs some tweaks in order to make it work more efficient. This can probably be achieved by solving the IndexError, for using 4 GPUs for a single task. Sadly a lot of time had to be used to fix unknown bugs in the AtmoRep project and there was no time left to look into the third Milestone, but future likewise endeavours can be build upon these successes.

References

[1] Atmorep modified repo. `https://github.com/Stefan-Eger/P1-Atmorep`.

[2] Atmorep original code. `https://github.com/clessig/atmorep`.

[3] Cdo. `https://code.mpimet.mpg.de/projects/cdo/files`.

[4] Cdo tutorial. `https://opendata.dwd.de/climate_environment/REA/COSMO_REA6/help_COSMO_REA6/COSMO_REA6_Starthilfe.pdf`.

[5] Cosmo. `https://reanalysis.meteo.uni-bonn.de/?Overview`.

[6] Cosmo hourly data. `https://opendata.dwd.de/climate_environment/REA/COSMO_REA6/hourly/`.

[7] Era5. `https://datapub.fz-juelich.de/atmorep/data/`.

[8] Meluxina. `https://docs.lxp.lu/system/overview/`.

[9] Ncview. `https://cirrus.ucsd.edu/ncview/`.

[10] Lessig, C., Luise, I., Gong, B., Langguth, M., Stadtler, S., and Schultz, M. Atmorep: A stochastic model of atmosphere dynamics using large scale representation learning, 2023.

A Appendix-Code

A.1 Rotate and Cut out COSMO Data

```bash
#!/bin/bash

START_TIME=$SECONDS

SRC_DIR=./V/2017
DST_DIR=./V/2017
REA6_GRID=./griddes_REA6.txt
LON1=4
LON2=22
LAT1=40
LAT2=55
N=32

if [ ! -d "$SRC_DIR" ]; then
        echo "Error: Source directory does not exist"
        echo "$SRC_DIR"
        exit 1
fi

# checking if destination directory exists if not it will be created
if [ ! -d "$DST_DIR" ]; then
        echo "DST_DIR is being created"
        mkdir -p "$DST_DIR"
fi

unpack_tar(){
        #echo "£1 will be unpacked"
        #tar -xvf "£1" -C "£DST_DIR"
        tar -xf "$1" -C "$DST_DIR"

        rm "$1"
}

rotate_grid(){
        #echo "£1 will be rotated"
        cdo -f nc4 -z zip=9 -copy -setgrid,"$REA6_GRID" "$1" "$1.nc4"
        rm "$1"
}
cut_out(){
        #echo "£1 will be cut out"
        cdo sellonlatbox,"$LON1","$LON2","$LAT1","$LAT2" "$1" "$1.nc4"
        rm "$1"
}



# Loops over each zipped file in the SRC_DIR.
echo "unpack tar files..."
(
for file in "$SRC_DIR"/*; do
   ((i=i%N)); ((i++==0)) && wait
        unpack_tar "$file" &
done; wait
) & all_tars=$!

wait $all_tars

# Loops over each grib file in the SRC_DIR and rotates the grid.
echo "rotate grids..."
(
for file in "$SRC_DIR"/*; do
   ((i=i%N)); ((i++==0)) && wait
        rotate_grid "$file" &
done; wait
) & all_rotations=$!

wait $all_rotations

# Loops over each nc4 file in the SRC_DIR and cuts out specified Longitude and Latitude.
echo "cutting out specified Longitude and Latitude..."
(
for file in "$SRC_DIR"/*; do
   ((i=i%N)); ((i++==0)) && wait
        cut_out "$file" &
done; wait
) & all_cuts=$!

wait $all_cuts

echo "FINISHED: "$SRC_DIR""
#output duration
duration=$(($SECONDS - $START_TIME))
echo "$(($duration / 60)) minutes and $(($duration % 60)) seconds elapsed."
```

A.2 Merge hours to Year-Month

```bash
#!/bin/bash
START_TIME=$SECONDS

#change as necessary
OLD_VARNAME=var34
VARNAME=V
YEAR=2017
N=32                #Tasks simultaniously allowed
declare -a LEVELS=(35 36 37 38 39 40)

#do not change
SRC_DIR=./"$VARNAME"/"$YEAR"
DST_DIR=./"$VARNAME"/"$YEAR"
ML_DST_DIR=./"$VARNAME"
FILE_PREFIX="$VARNAME".3D."$YEAR"

declare -a MONTHS=(01 02 03 04 05 06 07 08 09 10 11 12)

if [ ! -d "$SRC_DIR" ]; then
        echo "Error: Source directory does not exist"
        echo "$SRC_DIR"
        exit 1
fi

# checking if destination directory exists if not it will be created
if [ ! -d "$DST_DIR" ]; then
        echo "DST_DIR is being created"
        mkdir -p "$DST_DIR"
fi


merge_files(){
        echo "Month Data $1* will be merged to $2"
        cdo -f nc4 mergetime "$1*" "$2.nc4"
        rm -f "$1"*
}

chname_files(){

        filename=$(basename "$1")
        filename="${filename%.*}"

        cdo chname,"$2","$3" "$1" "$DST_DIR/${filename}_.nc4"
        rm "$1"
}
select_level(){

        filename=$(basename "$1")
        filename="${filename%.*}"

        for level in "${LEVELS[@]}"; do

                if [ ! -d "$ML_DST_DIR/ml${level}" ]; then
                        echo "ml$level Dir is being created"
                        mkdir -p "$ML_DST_DIR/ml${level}"
                fi

                cdo sellevel,"$level" $1
                    "$ML_DST_DIR/ml${level}/$ {filename}ml$level.nc4"
        done

        rm "$1"
}

#Merging all hour files to a single month file
echo "merge hours to month.nc4"
(
for month in "${MONTHS[@]}"; do
   ((i=i%N)); ((i++==0)) && wait
        merge_files "$SRC_DIR/$FILE_PREFIX$month"
            "$DST_DIR/cosmo_${VARNAME}_y${YEAR}_m${month}" &
done; wait
) & all_months=$!

wait $all_months


#updating variable Name for later extraction
echo "Update Variable Name"
(
for file in "$SRC_DIR"/*; do
   ((i=i%N)); ((i++==0)) && wait
        chname_files "$file" "$OLD_VARNAME" "$VARNAME" &
done; wait
) & all_vars=$!

wait $all_vars

#separating model Levels for atmorep
echo "Select Model Level ml<level>"
```

```
(
for file in "$SRC_DIR"/*; do
   ((i=i%N)); ((i++==0)) && wait
        select_level "$file" &
done; wait
) & all_vars=$!

wait $all_vars
echo "FINISHED: "$YEAR""
#output duration
duration=$(($SECONDS - $START_TIME))
echo "$(($duration / 60)) minutes and $(($duration % 60)) seconds elapsed."
```

A.3 Calculate Mean and Standard Deviation

```bash
#!/bin/bash

START_TIME=$SECONDS

YEAR=2017
VARNAME=U
N=32
declare -a LEVELS=(ml35 ml36 ml37 ml38 ml39 ml40)

SRC_DIR="$VARNAME"
NORM_DIR=normalization/"$VARNAME"/
IN_FILE_PREFIX="cosmo_${VARNAME}_y${YEAR}"
OUT_FILE_PREFIX="normalization_mean_var_${VARNAME}_y${YEAR}"
declare -a MONTHS=(01 02 03 04 05 06 07 08 09 10 11 12)

if [ ! -d "$SRC_DIR" ]; then
        echo "Error: Source directory does not exist"
        echo "$SRC_DIR"
        exit 1
fi

# checking if destination directory exists if not it will be created
if [ ! -d "$NORM_DIR" ]; then
        echo "NORM_DIR is being created $NORM_DIR"
        mkdir -p "$NORM_DIR"
fi

merge_mean_std(){
        cdo merge $1 $2 $3
        rm $1 $2
}

chname_variable(){
        cdo chname,"$VARNAME",$2 "$1" "$3"
        rm $1
}

calc_mean_and_std_all_months_per_year(){
        mean_filename="${2}_mean.nc4"
        mean_filename_named="${2}_mean_named.nc4"

        std_filename="${2}_std.nc4"
        std_filename_named="${2}_std_named.nc4"
        cdo monmean "$1.nc4" "$mean_filename"
        cdo monstd "$1.nc4" "$std_filename"

        chname_variable "$mean_filename" "mean" "$mean_filename_named"
        chname_variable "$std_filename" "std" "$std_filename_named"

        merge_mean_std "$mean_filename_named" "$std_filename_named" "$2.nc4"
}

#calculating mean and std for all month in a given year for all model levels
echo "Calculating mean and std for $YEAR"
(
for level in "${LEVELS[@]}"; do
        DST_DIR="$NORM_DIR/$level"

        if [ ! -d "$DST_DIR" ]; then
                echo "DST_DIR is being created $DST_DIR"
                mkdir -p "$DST_DIR"
        fi

        for month in "${MONTHS[@]}"; do
                ((i=i%N)); ((i++==0)) && wait
                calc_mean_and_std_all_months_per_year
                    "$SRC_DIR/$level/${IN_FILE_PREFIX}_m${month}_${level}"
                    "$DST_DIR/${OUT_FILE_PREFIX}_m${month}_${level}" &
        done
done; wait
) & all_levels=$!

wait $all_levels


echo "FINISHED: "$YEAR""
#output duration
duration=$(($SECONDS - $START_TIME))
echo "$(($duration / 60)) minutes and $(($duration % 60)) seconds elapsed."
```

A.4 Transform .nc4 to .bin

```python
import os
import sys
import netCDF4
import numpy as np


def convert_file(filepath):
    print("Processing " + filepath + "...")
    file = netCDF4.Dataset(filepath + '.nc4')

    # output dimension (timestamp, model_level, lat, lon)
    mean = file.variables['mean'][:]
    std = file.variables['std'][:]
    if(len(mean.shape) == 3 and len(std.shape) == 3):
        output = np.stack((mean[0].data, std[0].data), axis=2)
    else:
        output = np.stack((mean[0][0].data, std[0][0].data), axis=2)

    #print(f"output shape: {output.shape}")

    output.astype('float32').tofile(filepath + ".bin")

    # comment this line if .nc4 files should remain
    os.remove(filepath + '.nc4')



def create_filepaths(year: int, variable_name: str, level: int):
    filepaths = []

    fdir_base = 'normalization/{}/{}{}/'
    fname_base = 'normalization_mean_var_{}_y{}_m{:02d}_{}{}'

    filedir = fdir_base.format(variable_name, 'ml', level)

    for month in range(1, 12 + 1):
        filename = fname_base.format(variable_name, year, month, 'ml', level)
        filepaths.append(filedir + filename)
    return filepaths


def convert_nc4_to_bin():
    print(f"Arguments: {sys.argv[1:]}")
    years = list(map(int, sys.argv[1].split(',')))
    variable_name = sys.argv[2]
    levels = list(map(int, sys.argv[3].split(',')))

    print(f"YEARS: {years}")
    print(f"VARIABLE_NAMES: {variable_name}")
    print(f"MODEL_LEVELS: {levels}")

    for year in years:
        for level in levels:
            filepaths = create_filepaths(year, variable_name, level)
            for filepath in filepaths:
                convert_file(filepath)
            print(f"FINISHED YEAR{year} - ml{level}")


# Arguments YEARS, VARIABLENAME MODEL_LEVELS
# example
# python3 convert_nc4_to_bin.py 1995,1996 T 35,36,37,38,39,40
if __name__ == '__main__':
    convert_nc4_to_bin()
```

A.5 Training Atmorep on MeluXina

```
#!/bin/bash -x
#SBATCH --account=p200233
#SBATCH --time=11:59:59
#SBATCH --nodes=32
#SBATCH --ntasks=32                      # number of tasks
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=64                # number of cores
#SBATCH --gpus-per-task=4                 # number of gpu per task
#SBATCH --partition=gpu
#SBATCH --qos=default
#SBATCH --output=logs/forecast-%x.%j.out
#SBATCH --error=logs/forecast-%x.%j.err

#export PYTHONPATH=£{SLURM_SUBMIT_DIR}

source ${SLURM_SUBMIT_DIR}/pyenv/bin/activate

ml purge
ml Python
ml GCC
ml OpenMPI
#ml PyTorch

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export CUDA_VISIBLE_DEVICES=0,1,2,3

# so processes know who to talk to
MASTER_ADDR="$(scontrol show hostnames "$SLURM_JOB_NODELIST" | head -n 1)"
# Get IP for hostname.
export MASTER_ADDR="$(nslookup "$MASTER_ADDR" | grep -oP '(?<=Address: ).*')"

export NCCL_DEBUG=TRACE

echo "Starting job."
echo "Number of Nodes: $SLURM_JOB_NUM_NODES"
echo "Number of Tasks: $SLURM_NTASKS"
date

srun --label --cpu-bind=v --accel-bind=v python
    -u ${SLURM_SUBMIT_DIR}/atmorep/core/train.py > output/output_${SLURM_JOBID}.txt

echo "Finished job."
date
```

A.6 TOT_PRECIP data processing

```
#!/bin/bash

START_TIME=$SECONDS

OLD_VARNAME=var61
VARNAME=TOT_PRECIP

SRC_DIR=./$VARNAME
DST_DIR=./$VARNAME
REA6_GRID=./griddes_REA6.txt
LON1=4
LON2=22
LAT1=40
LAT2=55
N=32


if [ ! -d "$SRC_DIR" ]; then
        echo "Error: Source directory does not exist"
        echo "$SRC_DIR"
        exit 1
fi

# checking if destination directory exists if not it will be created
if [ ! -d "$DST_DIR" ]; then
        echo "DST_DIR is being created"
        mkdir -p "$DST_DIR"
fi

unpack_bzip(){
        #echo "£1 will be unpacked"
        #tar -xvf "£1" -C "£DST_DIR"
        #tar -xf "£1" -C "£DST_DIR"
        bzip2 -d "$1"
}

rotate_grid(){
        #echo "£1 will be rotated"
        cdo -f nc4 -z zip=9 -copy -setgrid,"$REA6_GRID" "$1" "$1.nc4"
        rm "$1"
}
cut_out(){

        cdo sellonlatbox,"$LON1","$LON2","$LAT1","$LAT2" "$1" "$1.nc4"
        rm "$1"
}

chname_files(){

        filename=$(basename "$1")
        #remove extensions
        filename="${filename%.*.*.*}"
        #remove previous dots until year and month are left
        filename="${filename#*.*.}"
        year="${filename:0:4}"
        month="${filename:4:6}"

        #echo "YEAR: £month"

        if [ ! -d "$DST_DIR/ml0" ]; then
                echo "DST_DIR is being created"
                mkdir -p "$DST_DIR/ml0"
        fi

        cdo chname,"$2","$3" "$1"
            "$DST_DIR/ml0/cosmo_${VARNAME}_y${year}_m${month}_ml0.nc4"
        rm "$1"
}



# Loops over each zipped file in the SRC_DIR.
echo "unpack bzip files..."
(
for file in "$SRC_DIR"/*; do
   ((i=i%N)); ((i++==0)) && wait
        unpack_bzip "$file" &
done; wait
) & all_tars=$!

wait $all_tars

# Loops over each grib file in the SRC_DIR and rotates the grid.
echo "rotate grids..."
(
for file in "$SRC_DIR"/*; do
   ((i=i%N)); ((i++==0)) && wait
        rotate_grid "$file" &
done; wait
) & all_rotations=$!
```

```
wait $all_rotations

# Loops over each nc4 file in the SRC_DIR and cuts out specified Longitude and Latitude.
echo "cutting out specified Longitude and Latitude..."
(
for file in "$SRC_DIR"/*; do
   ((i=i%N)); ((i++==0)) && wait
        cut_out "$file" &
done; wait
) & all_cuts=$!

wait $all_cuts

#updating variable Name for later extraction
echo "Update Variable Name"
(
for file in "$SRC_DIR"/*; do
   ((i=i%N)); ((i++==0)) && wait
        chname_files "$file" "$OLD_VARNAME" "$VARNAME" &
done; wait
) & all_vars=$!

wait $all_vars
echo "FINISHED: "$SRC_DIR""
#output duration
duration=$(($SECONDS - $START_TIME))
echo "$(($duration / 60)) minutes and $(($duration % 60)) seconds elapsed."
```

B Appendix-Pictures

B.1 Rotated REA6 Temperature

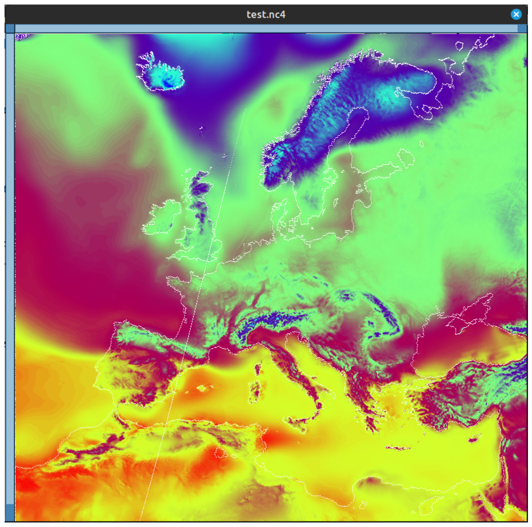

Figure 1: Rotated REA6 Temperature

B.2 Rotated and cut out REA6 Temperature

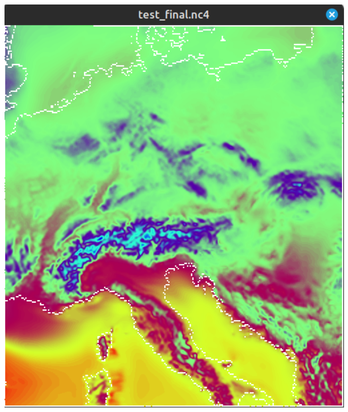

Figure 2: Rotated and cut out REA6 Temperature