In [ ]:
```
station_name = "Marshall"
test_year = 2021
```

In [ ]:
```
from climatereconstructionai import evaluate
import os
import regex as re

def evaluate_wrapper(use_time_context = True, iter = "final", station = station_name):

    snapshot_folder_name = f"snapshots_{station.lower()}"
    if not use_time_context:
        snapshot_folder_name += "_backup"

    # copy "test_args_template.txt" to "test_args_temp.txt"
    # while replacing:

    replace_dir = {
        "STATIONNAME" : station.lower(),
        "ITERATION" : str(iter),
        "SNAPSHOTDIR" : snapshot_folder_name,
    }

    template_file = "test_args_template.txt" if use_time_context else "test_args_template_no-time.txt"
    temp_file = "test_args_temp.txt"


    with open(template_file, "r") as f:
        lines = f.readlines()
        with open(temp_file, "w") as f2:
            for line in lines:
                for key in replace_dir:
                    line = re.sub(key, replace_dir[key], line)
                f2.write(line)

            # save and close
            f2.close()

    evaluate(temp_file)

    os.remove(temp_file)
```

In [ ]:
```
import xarray as xr
from utils import DataSet, DatasetPlotter
import numpy as np
import os

test_folder_path = "/work/bm1159/XCES/xces-work/k203179/data/test"
output_file_path = "outputs/output_output.nc"
era5_file = f"{test_folder_path}/era5_for_{station_name.lower()}.nc"

# get measurements values

measurements_data = xr.open_dataset(test_folder_path + f"/reality_{station_name.lower()}.nc")

# plot era5 and output at timesteps [x, ...]
```

```
era5_ds = DataSet(era5_file)
era5_data = xr.open_dataset(era5_file)
```

Evaluate with no timecontext

In [ ]:
```
evaluate_wrapper(use_time_context = False, iter = "400000", station = station_name)
output_no_time_ds = DataSet(output_file_path, name = "Reconstructed without time context")
output_no_time_data = xr.open_dataset(output_file_path)
```

```
/home/k/k203179/.conda/envs/crai/lib/python3.10/site-packages/climatereconstructionai/utils/normalizer.py:10: RuntimeWarning: Mean of empty slice
  img_mean.append(np.nanmean(np.array(img_data[i])))
/home/k/k203179/.conda/envs/crai/lib/python3.10/site-packages/numpy/lib/nanfunctions.py:1879: RuntimeWarning: Degrees of freedom <= 0 for slice.
  var = nanvar(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
  0%|          | 0/1 [00:00<?, ?it/s]100%|██████████| 1/1 [00:05<00:00,  5.40s/it]
```

Evaluate with timecontex

In [ ]:
```
evaluate_wrapper(use_time_context = True, iter = "400000", station = station_name)
output_with_time_ds = DataSet(output_file_path, name="Reconstructed with time context")
output_with_time_data = xr.open_dataset(output_file_path)
```

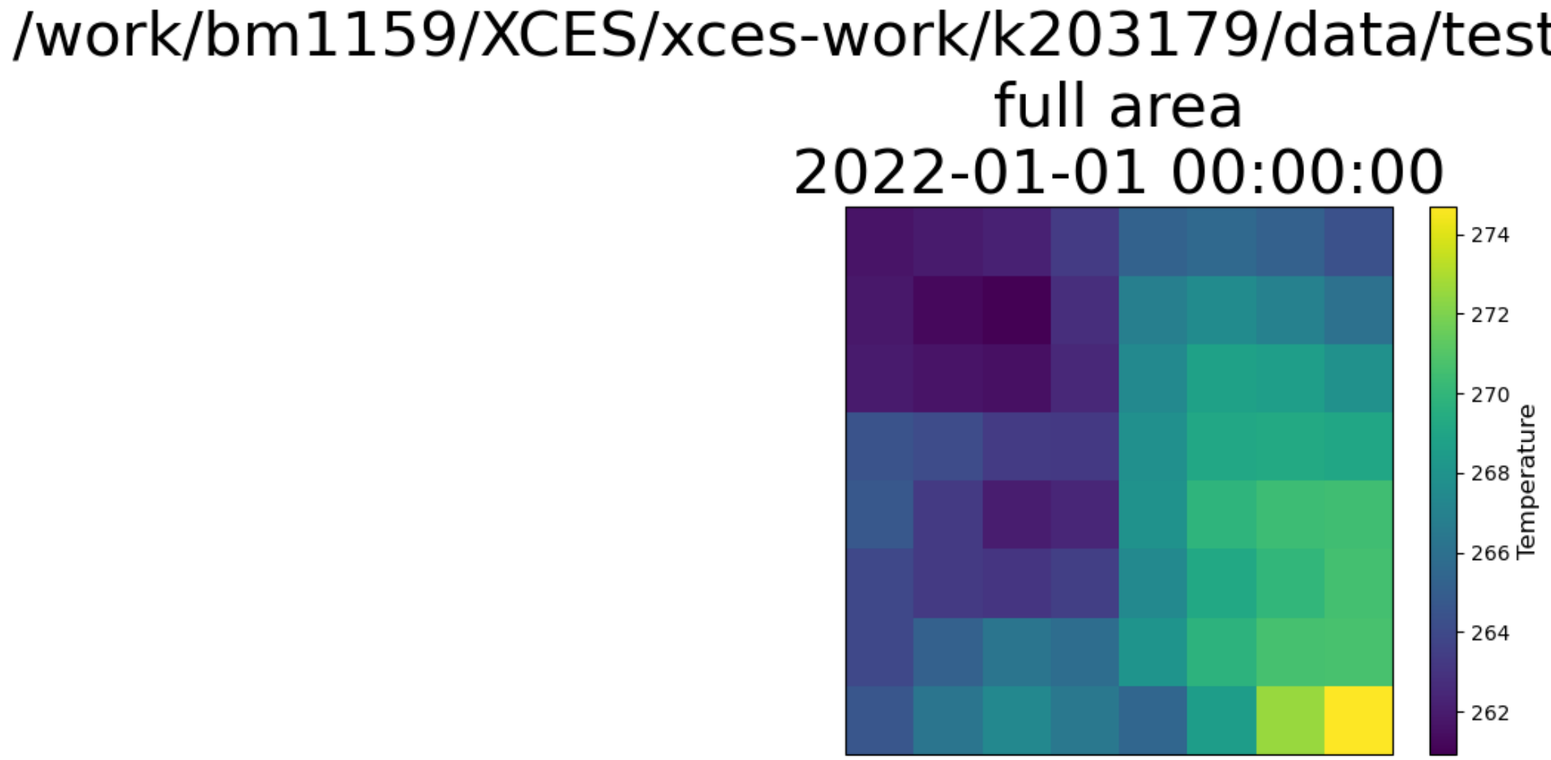
```
/home/k/k203179/.conda/envs/crai/lib/python3.10/site-packages/climatereconstructionai/utils/normalizer.py:10: RuntimeWarning: Mean of empty slice
  img_mean.append(np.nanmean(np.array(img_data[i])))
/home/k/k203179/.conda/envs/crai/lib/python3.10/site-packages/numpy/lib/nanfunctions.py:1879: RuntimeWarning: Degrees of freedom <= 0 for slice.
  var = nanvar(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
100%|██████████| 1/1 [00:06<00:00,  6.89s/it]
```

In [ ]:
```
print("Dimensions:", era5_ds.time[-1], output_no_time_ds.time[-1])

plot_timestep = len(era5_ds.time) - 1

vmin = min(
    np.min(era5_ds.dataset.variables['tas'][plot_timestep, :, :]),
    np.min(output_no_time_ds.dataset.variables['tas'][plot_timestep, :, :])
)

vmax = max(
    np.max(era5_ds.dataset.variables['tas'][plot_timestep, :, :]),
    np.max(output_no_time_ds.dataset.variables['tas'][plot_timestep, :, :])
)
```

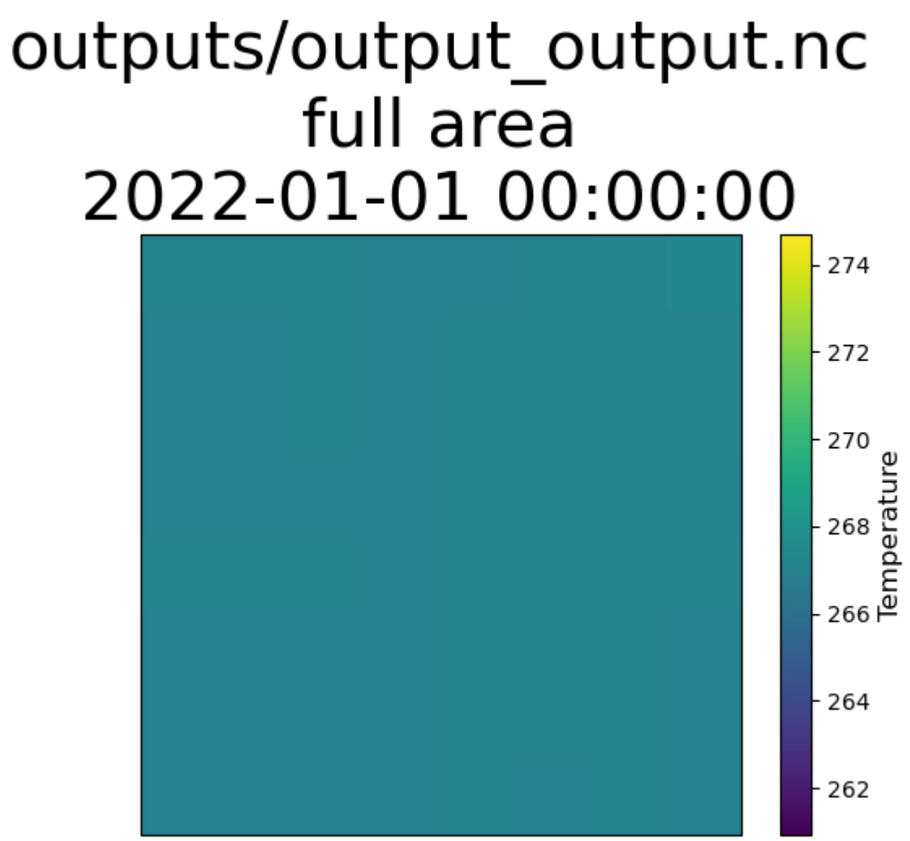
```
Dimensions: 8760.0 8760.0
```

Last Output point in time-context is far off:

In [ ]:
```
plotter = DatasetPlotter(era5_ds)
plotter.time_index_list = [plot_timestep]
plotter.vmin = vmin
plotter.vmax = vmax
plotter.plot()
```



In [ ]:
```
plotter = DatasetPlotter(output_no_time_ds)
plotter.time_index_list = [plot_timestep]
plotter.vmin = vmin
plotter.vmax = vmax
plotter.plot()
```



In [ ]:
```
plotter = DatasetPlotter(output_with_time_ds)
plotter.time_index_list = [plot_timestep]
plotter.vmin = vmin
plotter.vmax = vmax
plotter.plot()
```

In [ ]:
```python
# get coordinates from measurements nc file
import numpy as np

station_lon, station_lat = measurements_data.lon.values[0], measurements_data.lat.values[0]
print(f"station is at {station_lon}, {station_lat}")

# get nearest coordinates in era5
def get_left_right_nearest_elem_in_sorted_array(array, value):
    length = len(array)
    left = len(list(filter(lambda x: x <= value, array))) - 1
    right = length - len(list(filter(lambda x: x >= value, array)))
    nearest = min(left, right, key=lambda x: abs(array[x] - value))
    return left, right, nearest

test_array = [1, 2, 3, 4, 5, 6, 7, 8]
test_search = 5.51

print(f"searching for {test_search} in {test_array}")
left_idx, right_idx, nearest_idx =  get_left_right_nearest_elem_in_sorted_array(test_array, test_search)
print(f"idx left to {test_search} is {left_idx}, idx right to {test_search} is {right_idx}, nearest idx is {nearest_idx}")
print(f"mid crop: {test_array[left_idx:right_idx+1]}")
```

```
station is at -105.196, 39.9496
searching for 5.51 in [1, 2, 3, 4, 5, 6, 7, 8]
idx left to 5.51 is 4, idx right to 5.51 is 5, nearest idx is 5
mid crop: [5, 6]
```

In [ ]:
```python
def era_vs_reconstructed_comparision_to_df(era5_data, reconstructed_with_time_data, reconstructed_no_time_data):
```

```
    lon_left_idx, lon_right_idx, lon_nearest_idx = get_left_right_nearest_elem_in_sorted_array(era5_data.lon.values, station_lon % 360)
    lat_left_idx, lat_right_idx, lat_nearest_idx = get_left_right_nearest_elem_in_sorted_array(era5_data.lat.values, station_lat)

    era5_nearest_values = era5_data.variables["tas"][:, lon_nearest_idx, lat_nearest_idx]

    reconstructed_with_time_data_values = reconstructed_with_time_data.variables["tas"].stack(grid=['lat', 'lon']).values
    reconstructed_no_time_data_values = reconstructed_no_time_data.variables["tas"].stack(grid=['lat', 'lon']).values
    measurements_data_values = measurements_data.variables["tas"][...].mean(axis=(1,2))


    # timeaxis
    time = measurements_data.variables["time"][:]

    import pandas as pd

    # create dataframe with all values
    df = pd.DataFrame()

    df["time"] = time

    # index should be time
    df.set_index("time", inplace=True)

    df["era5_nearest"] = era5_nearest_values
    df["reconstructed_median"] = [np.median(x) for x in reconstructed_no_time_data_values]
    df["reconstructed_with_time_context_median"] = [np.median(x) for x in reconstructed_with_time_data_values]

    df["measurements"] = measurements_data_values

    return df
```

Generate Dataframe

- makes resampling easier

In [ ]:
```
hourly_df = era_vs_reconstructed_comparision_to_df(era5_data, output_with_time_data, output_no_time_data)

# drop last 1 hour
hourly_df = hourly_df[:-1]

# print a section of the df

start_print_date = "2021-12-31"
end_print_date = "2022-12-31"

hourly_df[start_print_date:end_print_date]
```

Out[ ]:

time	era5_nearest	reconstructed_median	reconstructed_with_time_context_median	measurements
2021-12-31 00:00:00	278.701538	278.405273	279.030029	277.210000
2021-12-31 01:00:00	278.443237	278.296936	279.160889	277.808333
2021-12-31 02:00:00	277.574158	278.703125	277.708374	278.211017
2021-12-31 03:00:00	277.030609	278.842529	277.351898	278.283333
2021-12-31 04:00:00	276.625641	278.398071	277.573242	278.460000
2021-12-31 05:00:00	273.666718	278.865906	277.058105	278.793333
2021-12-31 06:00:00	272.025665	278.628357	275.637085	278.346667
2021-12-31 07:00:00	272.586304	277.702332	276.955627	276.435000
2021-12-31 08:00:00	272.302032	277.179230	278.614197	276.745000
2021-12-31 09:00:00	271.156952	276.619965	277.795654	275.803333
2021-12-31 10:00:00	270.792542	274.366821	273.480621	275.028333
2021-12-31 11:00:00	270.880646	275.297699	274.081818	276.333333
2021-12-31 12:00:00	271.105133	275.895081	274.529938	274.403333
2021-12-31 13:00:00	270.799286	274.720734	274.774292	270.200000
2021-12-31 14:00:00	270.200378	272.777100	273.076813	270.596667
2021-12-31 15:00:00	270.555420	271.469360	272.061462	271.075000
2021-12-31 16:00:00	271.382416	272.516235	273.203552	271.183333
2021-12-31 17:00:00	272.423828	272.573547	275.059692	271.018333
2021-12-31 18:00:00	273.161255	273.008911	275.633057	271.145000
2021-12-31 19:00:00	272.459076	273.077454	275.113159	271.536667
2021-12-31 20:00:00	271.580505	272.360718	274.085327	271.795000
2021-12-31 21:00:00	271.244659	271.568085	273.217468	270.550000
2021-12-31 22:00:00	268.393768	268.839355	269.188660	269.468333
2021-12-31 23:00:00	268.587921	268.101318	268.911316	268.941667

Implement plotting method of dataframe

In [ ]:
```python
def plot_n_steps_of_df(df, as_delta, n=None, title=None):

    from matplotlib import pyplot as plt

    time = df.index.values
    if n is None:
        n = len(df)

    # random slice of n consecutive datapoints
    import random
```

```
    slice_start = random.randint(0, len(time) - n)
    time_slice = slice(slice_start, slice_start + n)

    time = time[time_slice]

   # era5_mid_values = df["era5_mid"].values -273.15
    era5_nearest_values = df["era5_nearest"].values - 273.15
    reconstructed_median_values = df["reconstructed_median"].values - 273.15
    reconstructed_median_with_time_context_values = df["reconstructed_with_time_context_median"].values - 273.15

    measurements_values = df["measurements"].values - 273.15

    rmse_reconstructed = np.sqrt(np.sum((reconstructed_median_values[time_slice] - measurements_values[time_slice])**2) / len(time))
    rmse_reconstructed_with_time_context = np.sqrt(np.sum((reconstructed_median_with_time_context_values[time_slice] - measurements_values[time_slice])**2) / len(time))
    rmse_era5_nearest = np.sqrt(np.sum((era5_nearest_values[time_slice] - measurements_values[time_slice])**2) / len(time))

    correlation_reconstructed = np.corrcoef(reconstructed_median_values[time_slice], measurements_values[time_slice])[0,1]
    correlation_reconstructed_with_time_context = np.corrcoef(reconstructed_median_with_time_context_values[time_slice], measurements_values[time_slice])[0,1]
    correlation_era5_nearest = np.corrcoef(era5_nearest_values[time_slice], measurements_values[time_slice])[0,1]

    if as_delta:
     #   era5_mid_values = era5_mid_values - measurements_values
        era5_nearest_values = era5_nearest_values - measurements_values
        reconstructed_median_values = reconstructed_median_values - measurements_values
        reconstructed_median_with_time_context_values = reconstructed_median_with_time_context_values - measurements_values
        measurements_values = measurements_values - measurements_values

        # y-axis title, temperature difference
        plt.ylabel("Delta calculated by subtracting measurement data [C°]")

    else:
        plt.ylabel("Temperature at surface [C°]")


    plt.plot(time, era5_nearest_values[time_slice], label="ERA5 nearest point", color="red")
    # plt.plot(time, era5_mid_values[time_slice], label="ERA5 nearest 4 points")


    plt.plot(time, reconstructed_median_values[time_slice], label="Reconstructed", color="blue")
    plt.plot(time, reconstructed_median_with_time_context_values[time_slice], label="Reconstructed with time context", color="green")
    plt.plot(time, measurements_values[time_slice], label="Measurements", color="black")

    # x-axis labels 90 degrees
    plt.xticks(rotation=45)

    # title
    if title is not None:
        plt.title(title)


    # font size of legend
    plt.rcParams.update({'font.size': 10})

    # font size of axis labels
    plt.rcParams.update({'axes.labelsize': 12})

    # font size of title
    plt.rcParams.update({'axes.titlesize': 30})

    plt.legend()
```

```
    # position legend below chart to the right
    plt.legend(bbox_to_anchor=(1, 1.15), loc='upper right', borderaxespad=0.)


    # text below diagram with RMSE and Correlation in fontsize 10
    plt.text(0.1,0.95, f"RMSE reconstructed: {rmse_reconstructed:.2f} C°\n" +
             f"RMSE reconstr. (time context): {rmse_reconstructed_with_time_context:.2f} C°\n" +
             f"RMSE ERA5 nearest point: {rmse_era5_nearest:.2f} C°",

             fontsize=10, transform=plt.gcf().transFigure)

    plt.text(0.3, 0.95, f"Correlation reconstructed: {correlation_reconstructed:.3f}\n" +
             f"Correlation reconstr. (time context): {correlation_reconstructed_with_time_context:.3f}\n" +
             f"Correlation ERA5 nearest point: {correlation_era5_nearest:.3f}",

             fontsize = 10, transform=plt.gcf().transFigure)

    # figure size A4 landscape
    plt.gcf().set_size_inches(16, 8)

    plt.show()
```

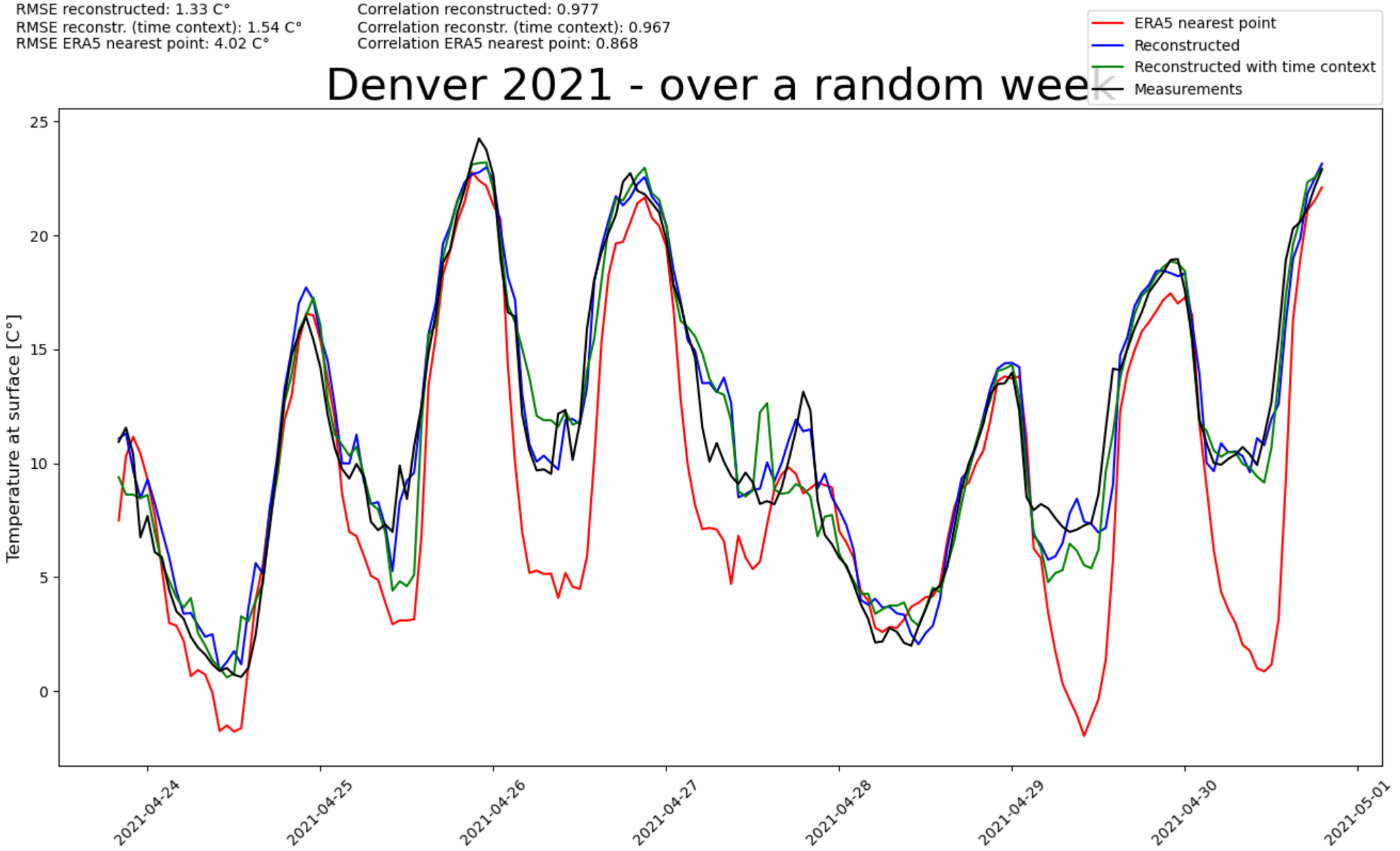
Plot Hourly (deltas), so errors against real measurements

In [ ]:

```
n = 168
if n == 168:
    title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - over a random week"
else:
    title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - {n} random consecutive hourly steps"
plot_n_steps_of_df(hourly_df, as_delta=False, n=n, title=title)
plot_n_steps_of_df(hourly_df, as_delta=False, n=n, title=title)
```

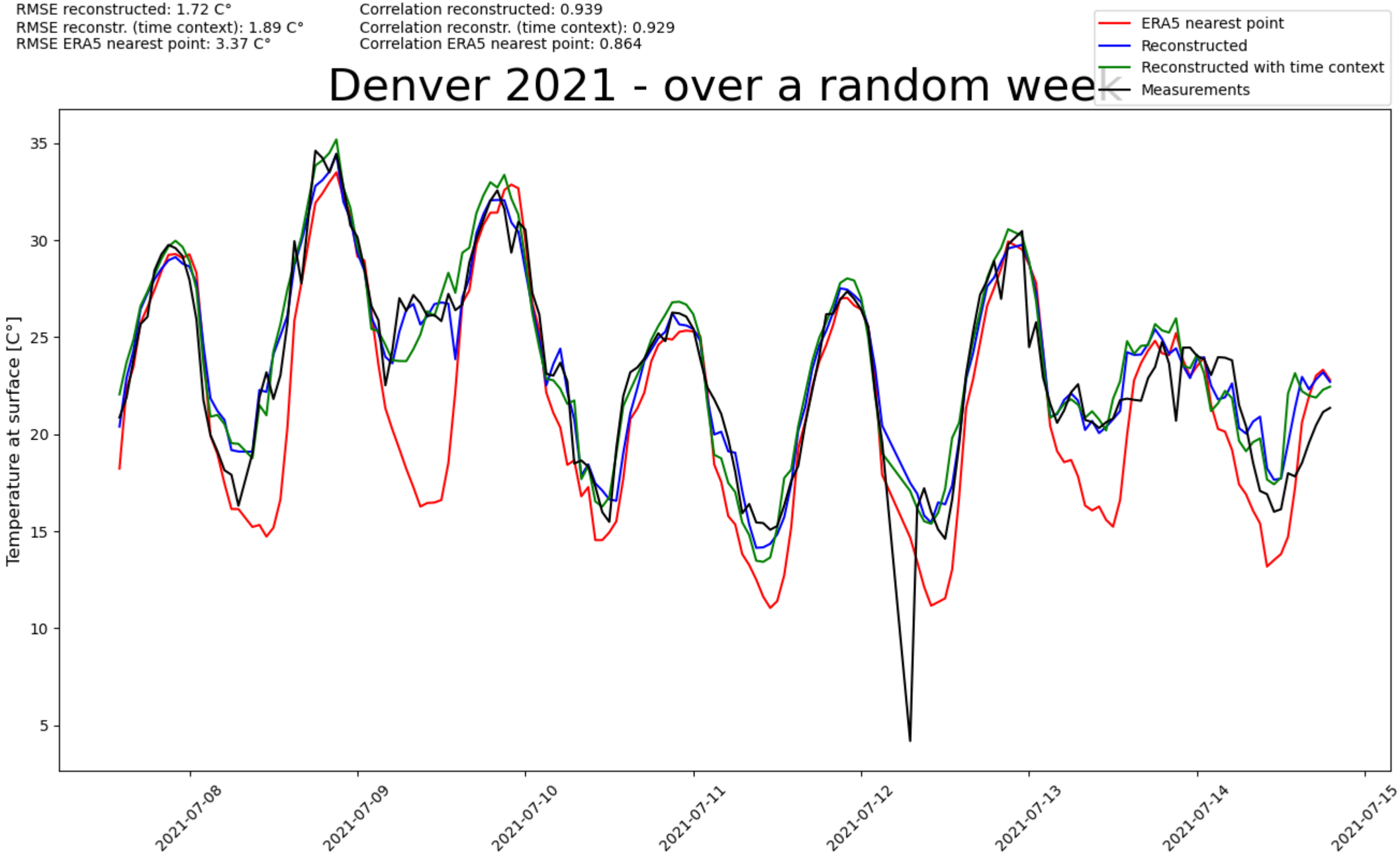
RMSE reconstructed: 1.33 C°
RMSE reconstr. (time context): 1.54 C°
RMSE ERA5 nearest point: 4.02 C°

Correlation reconstructed: 0.977
Correlation reconstr. (time context): 0.967
Correlation ERA5 nearest point: 0.868



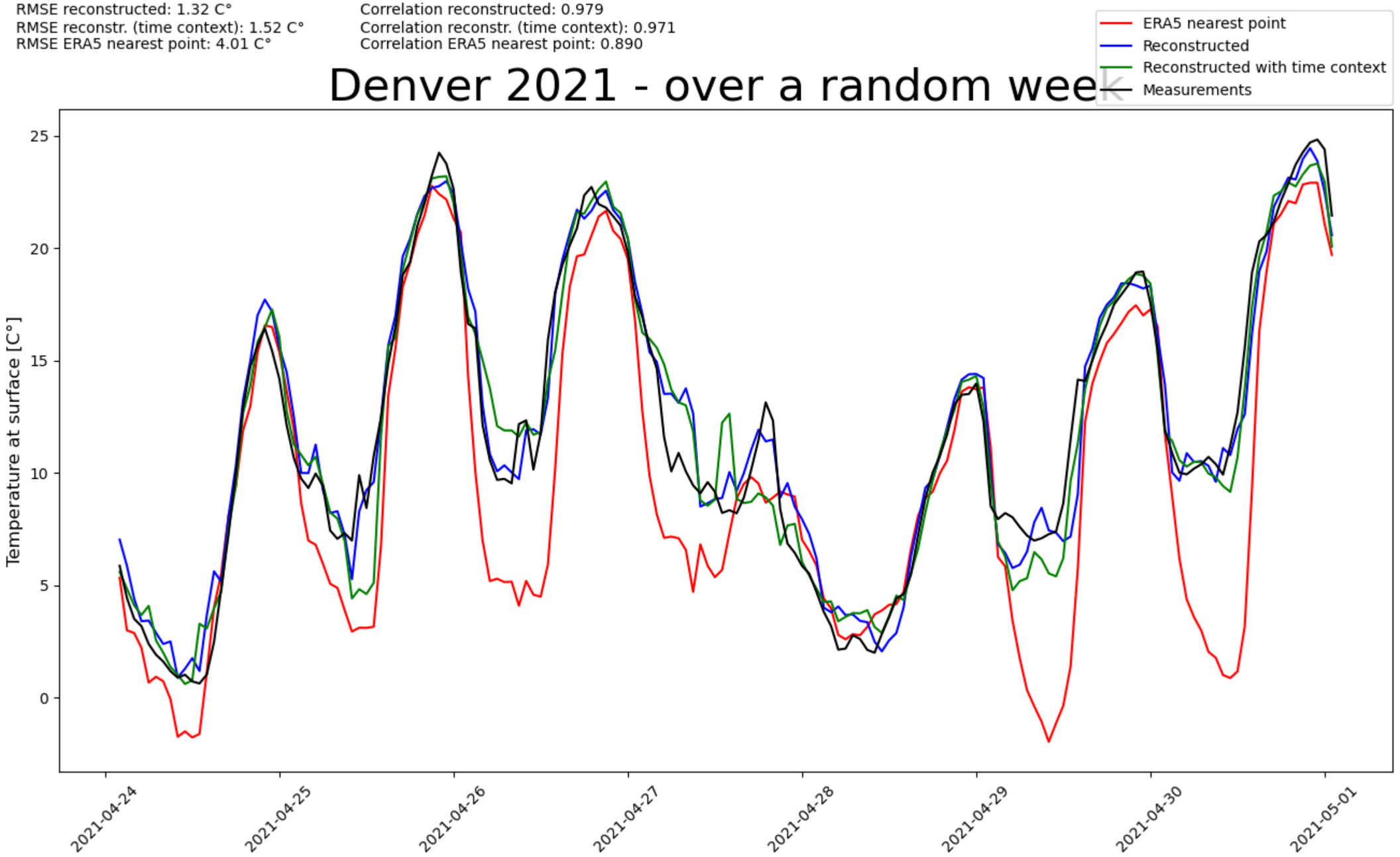
RMSE reconstructed: 1.72 C°
RMSE reconstr. (time context): 1.89 C°
RMSE ERA5 nearest point: 3.37 C°

Correlation reconstructed: 0.939
Correlation reconstr. (time context): 0.929
Correlation ERA5 nearest point: 0.864



RMSE reconstructed: 1.32 C°
RMSE reconstr. (time context): 1.52 C°
RMSE ERA5 nearest point: 4.01 C°

Correlation reconstructed: 0.979
Correlation reconstr. (time context): 0.971
Correlation ERA5 nearest point: 0.890

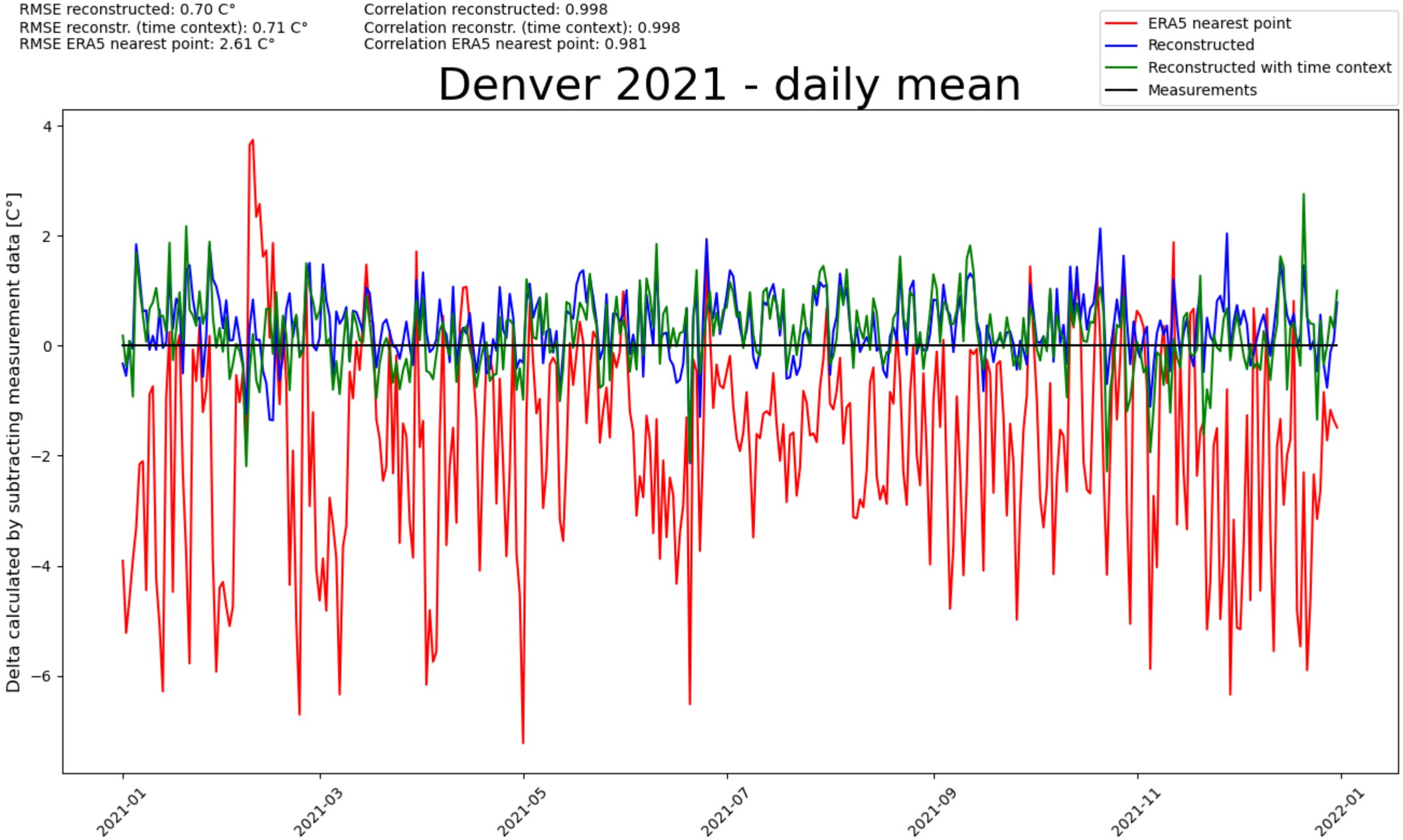


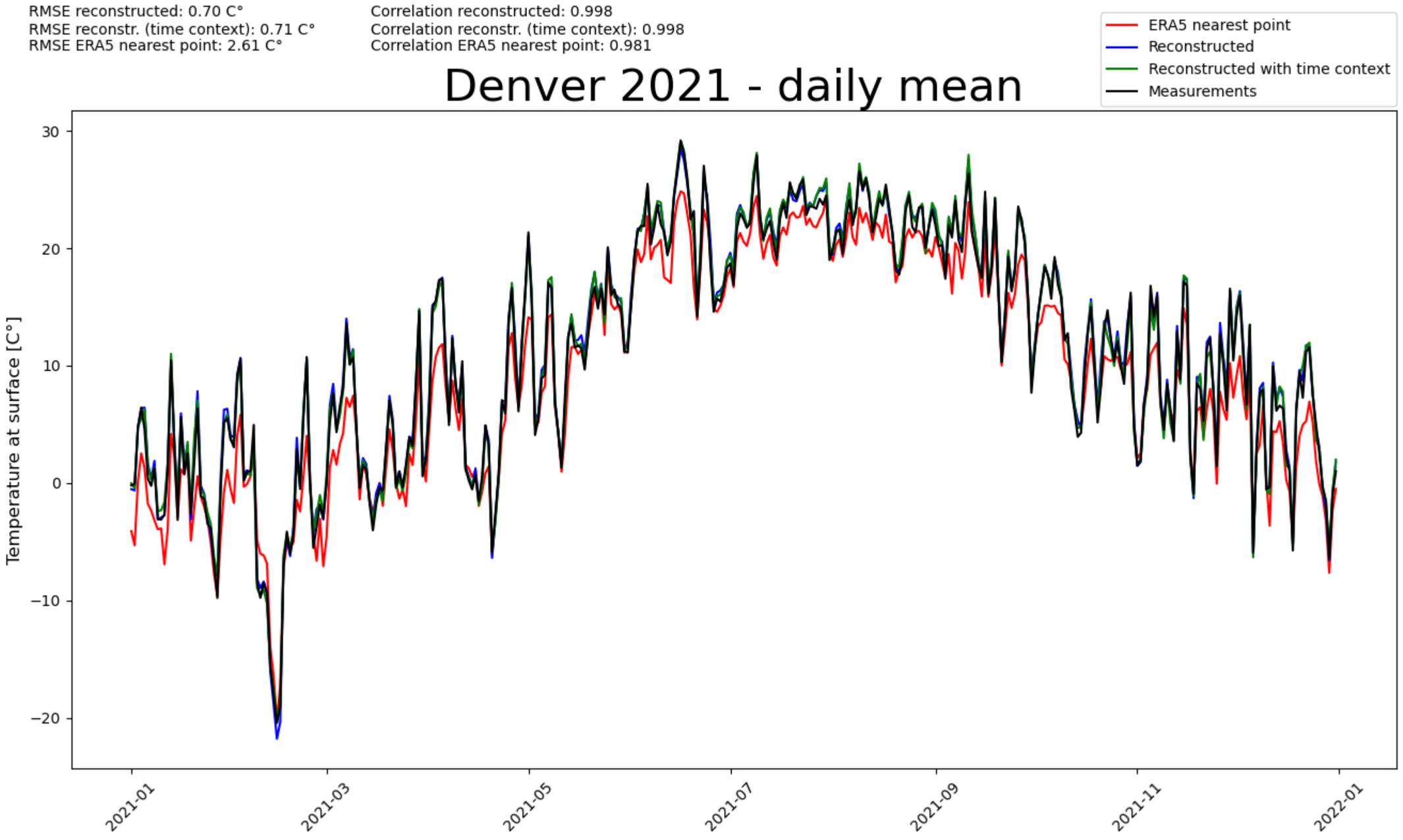
Resample Data from hourly to daily or monthly

In [ ]:
```
# resample to daily mean
daily_df = hourly_df.resample("D").mean()
# drop nans
daily_df = daily_df.dropna()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - daily mean"
plot_n_steps_of_df(daily_df, as_delta=True, title=title)
plot_n_steps_of_df(daily_df, as_delta=False, title=title)
```

RMSE reconstructed: 0.70 C°
RMSE reconstr. (time context): 0.71 C°
RMSE ERA5 nearest point: 2.61 C°

Correlation reconstructed: 0.998
Correlation reconstr. (time context): 0.998
Correlation ERA5 nearest point: 0.981

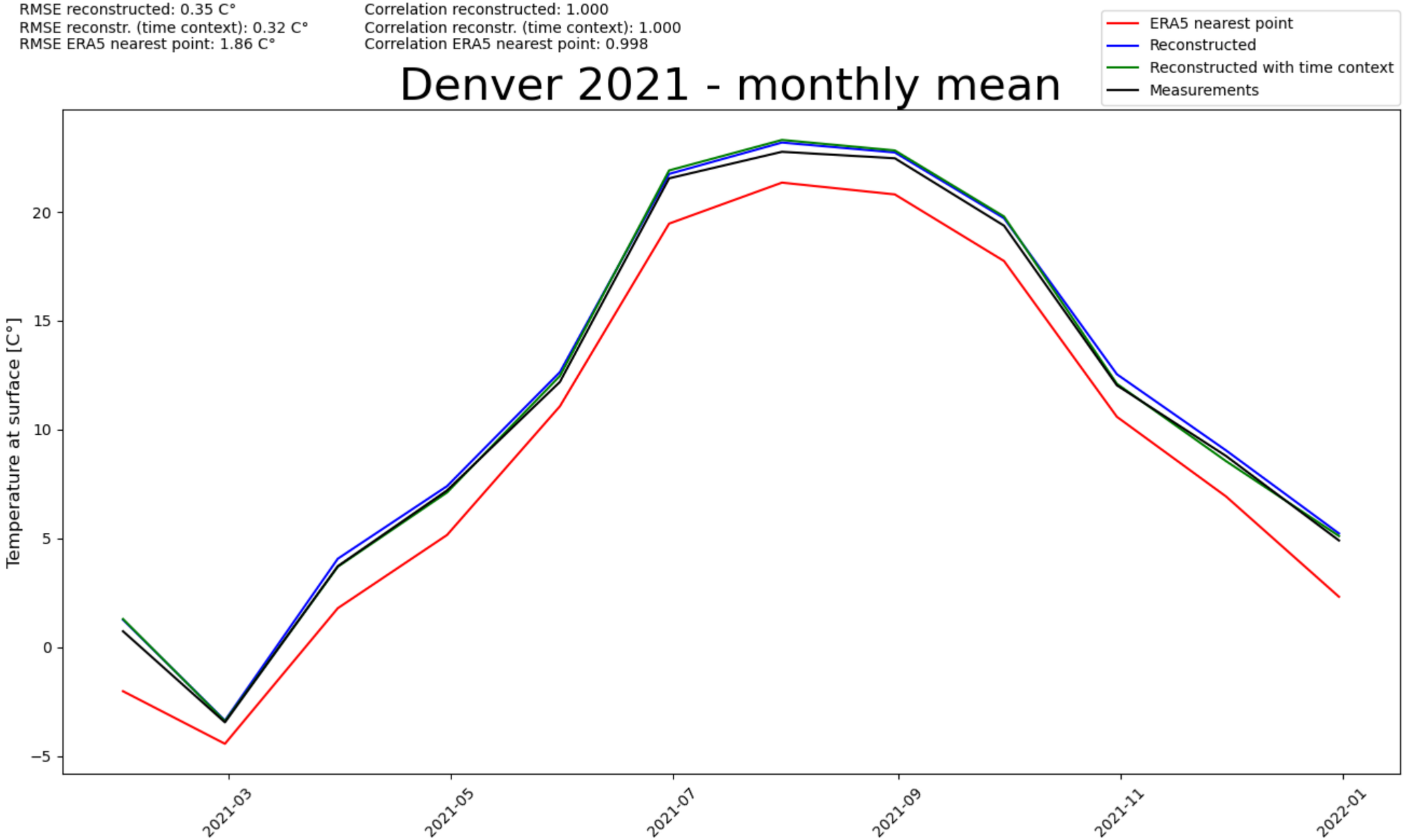




In [ ]:
```
# resample rows to monthly mean
df = hourly_df.resample("M").mean()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - monthly mean"
plot_n_steps_of_df(df, as_delta = False, title=title)
```

RMSE reconstructed: 0.35 C°
RMSE reconstr. (time context): 0.32 C°
RMSE ERA5 nearest point: 1.86 C°

Correlation reconstructed: 1.000
Correlation reconstr. (time context): 1.000
Correlation ERA5 nearest point: 0.998

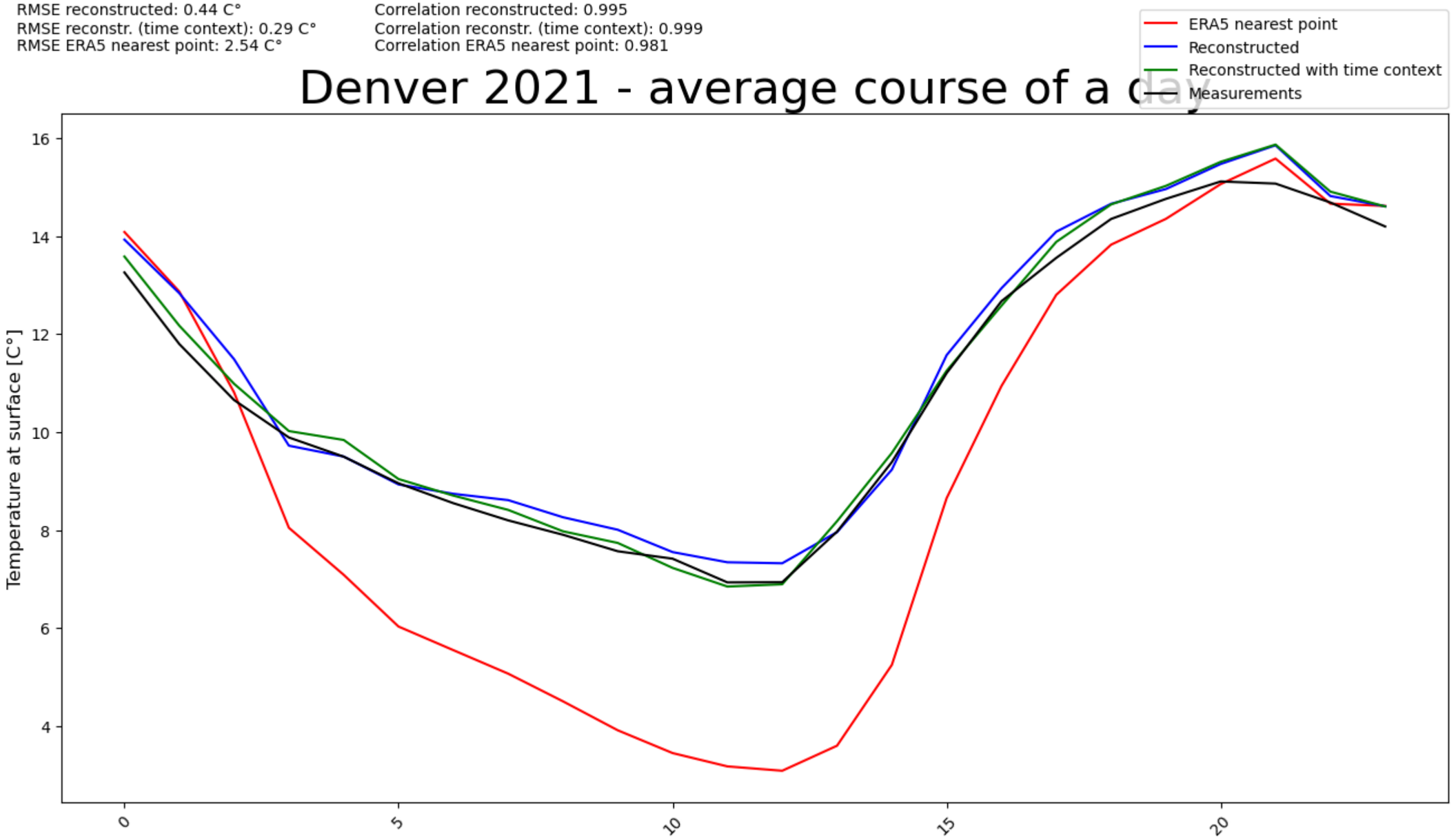


Average Course of the day

In [ ]:
```
# calculate the mean of each 24 hours over the whole year

day_course_df = hourly_df.groupby(hourly_df.index.hour).mean()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - average course of a day"
plot_n_steps_of_df(day_course_df, as_delta = False, title=title)
```

RMSE reconstructed: 0.44 C°
RMSE reconstr. (time context): 0.29 C°
RMSE ERA5 nearest point: 2.54 C°

Correlation reconstructed: 0.995
Correlation reconstr. (time context): 0.999
Correlation ERA5 nearest point: 0.981



Average Course of the month

In [ ]:
```
# calculate the mean of each day of a month over the whole year

month_course_df = hourly_df.groupby(hourly_df.index.day).mean()
title = f"{'Denver' if station_name == 'Marshall' else station_name} {test_year} - average course of a month"
plot_n_steps_of_df(month_course_df, as_delta = False, title=title)
```

RMSE reconstructed: 0.37 C°
RMSE reconstr. (time context): 0.29 C°
RMSE ERA5 nearest point: 1.91 C°

Correlation reconstructed: 0.996
Correlation reconstr. (time context): 0.995
Correlation ERA5 nearest point: 0.959