

```
In [ ]: import matplotlib.pyplot as plt
```

```
In [ ]: station_name = "Barbados"
test_year = 2020
at_iterations = "650000"
has_no_time_context = False
```

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

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

    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:

    assert not (use_time_context and reconstruct_gaps), "Cannot use time context and reconstruct gaps at the same time yet, because we don't split the gap files yet by variab

    replace_dir = {
        "STATIONNAME" : station.lower() if not reconstruct_gaps else f"the_gaps_{station.lower()}",
        "ITERATION" : str(iter),
        "SNAPSHOTDIR" : snapshot_folder_name,
    }

    if reconstruct_gaps:
        replace_dir["_leading_trailing"] = ""
        replace_dir["cleaned_"] = "cleaned_era5_for_"

    template_file = "test_args_template_trailing.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"
train_folder_path = "/work/bm1159/XCES/xces-work/k203179/data/train"
output_file_path = "outputs/output_output.nc"
era5_file = test_folder_path + f"/era5_for_{station_name.lower()}_leading_trailing.nc"
era5_full_file = f"/work/bm1159/XCES/xces-work/k203179/data_sets/era5_for_{station_name.lower()}.nc"
era5_gaps_file = f"/work/bm1159/XCES/xces-work/k203179/data_sets/era5_gaps_for_{station_name.lower()}.nc"

# get measurements values

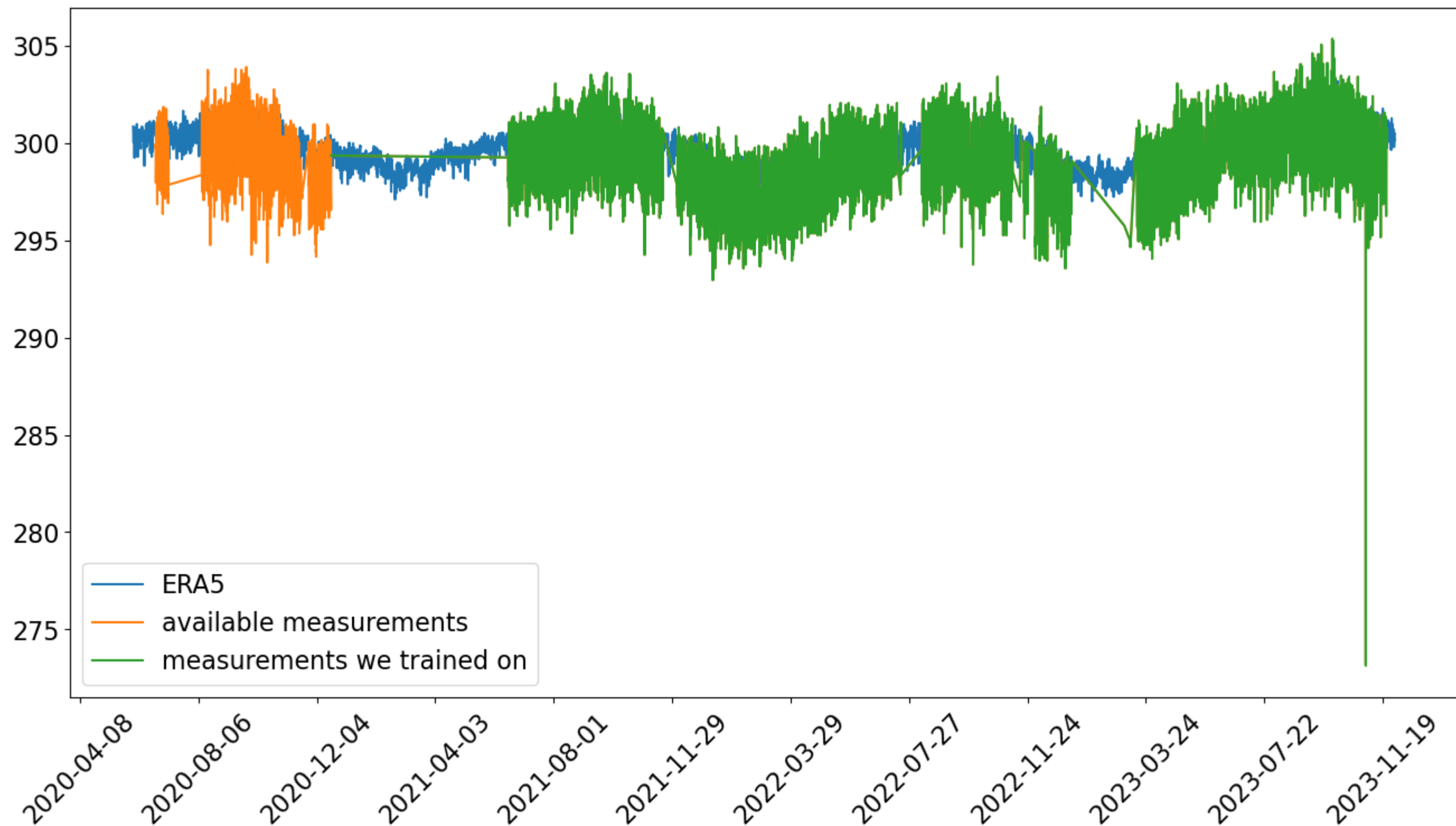
measurements_data = xr.open_dataset(test_folder_path + f"/{station_name.lower()}.nc")
era5_data = xr.open_dataset(era5_file)
measurements_data_trained = xr.open_dataset(train_folder_path + f"/expected_{station_name.lower()}.nc")

era5_full_data = xr.open_dataset(era5_full_file)

plt.plot(era5_full_data["time"], era5_full_data["tas"].mean(axis=(1,2)), label = "ERA5")
plt.plot(measurements_data["time"], measurements_data["tas"].mean(axis=(1,2)), label = "available measurements")
plt.plot(measurements_data_trained["time"], measurements_data_trained["tas"].mean(axis=(1,2)), label = "measurements we trained on")
plt.legend()

# rotate x-axis labels, and show them every 6 months
plt.xticks(rotation=45)
plt.gca().xaxis.set_major_locator(plt.MultipleLocator(120))

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

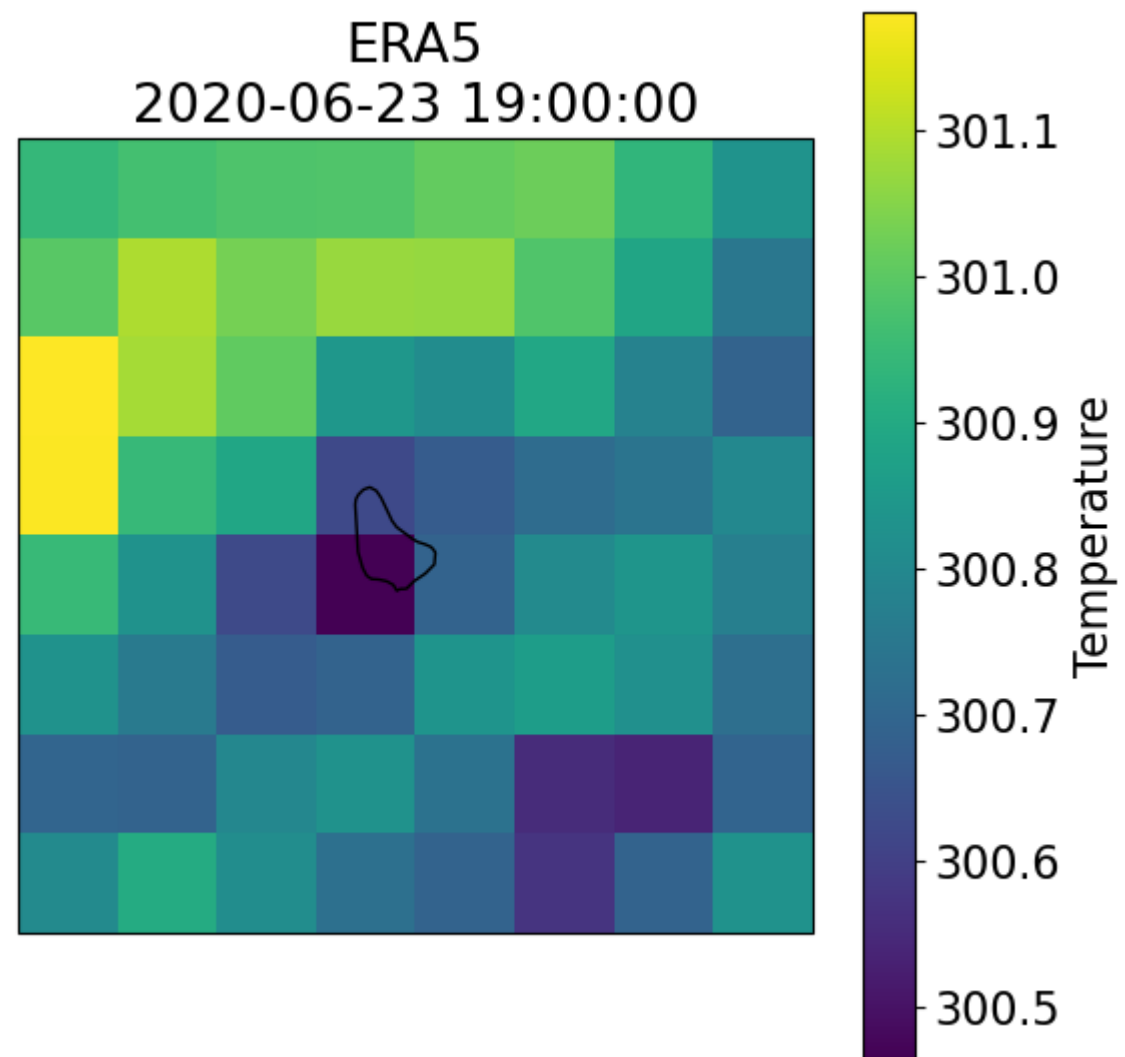


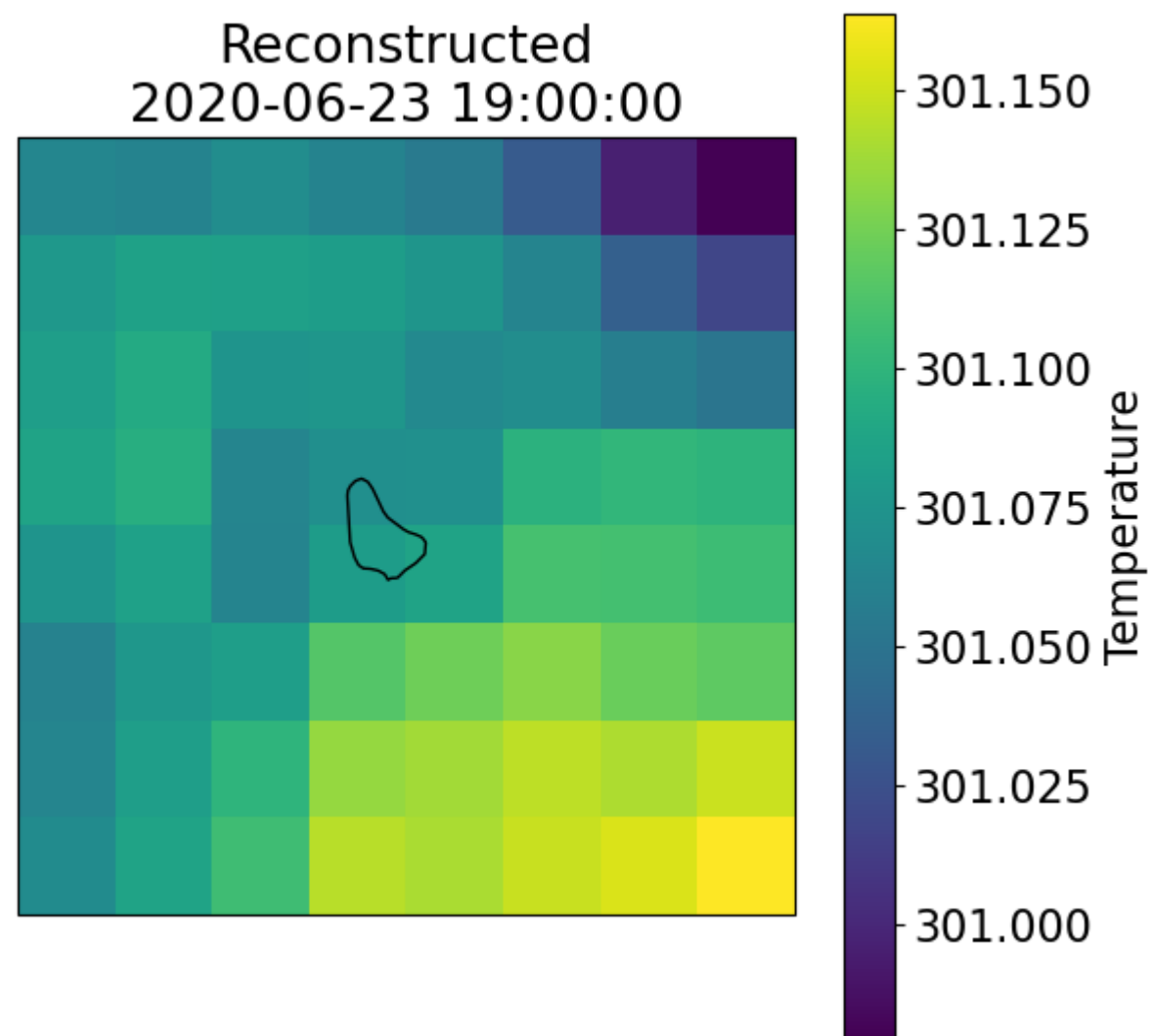
## Evaluate with no timecontext

```
In [ ]: evaluate_wrapper(use_time_context = False, iter = at_iterations, station = station_name)
        output_no_time_ds = DataSet(output_file_path, name = "Reconstructed")
        output_no_time_data = xr.open_dataset(output_file_path)
```

```
/home/k/k203179/.conda/envs/crai/lib/python3.10/site-packages/climateconstructionai/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 [00:08<00:00, 8.24s/it]
```

```
In [ ]: plotter1 = DatasetPlotter(DataSet(era5_file, name = "ERA5"))
plotter1.plot()
plotter2 = DatasetPlotter(output_no_time_ds)
plotter2.time_index_list = plotter1.time_index_list
plotter2.plot()
```





## Evaluate with time contex

```
In [ ]: if has_no_time_context:
        output_with_time_data = output_no_time_data
    else:
        evaluate_wrapper(use_time_context = True, iter = at_iterations, 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 [00:10<00:00, 10.54s/it]
```

```
In [ ]: # 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 -59.54316, 13.16443
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 [ ]: def era_vs_reconstructed_comparision_to_df(era5_data, reconstructed_with_time_data, reconstructed_no_time_data, measurements_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 = era5_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, measurements_data)

# print a section of the df

```

```
start_date = test_year.__str__() + "-01-01"
end_date = test_year.__str__() + "-12-31"

hourly_df = hourly_df[start_date:end_date]
```

## Implement plotting method of dataframe

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

    plot_trailing_time = plot_trailing_time and not has_no_time_context

    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 for validation", color="blue")
    if plot_trailing_time:
        plt.plot(time, reconstructed_median_with_time_context_values[time_slice], label="Reconstructed for val. using also ERA5 step before & after", 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': 26})

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" if plot_trailing_time else "") +
        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" if plot_trailing_time else "") +
        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()

```

## Hourly Analysis

...over the full year

```

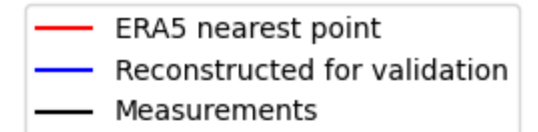
In [ ]: title = f"{station_name} - {test_year}"
plot_n_steps_of_df(hourly_df, as_delta=True, title=title)
plot_n_steps_of_df(hourly_df, as_delta=False, title=title)

```

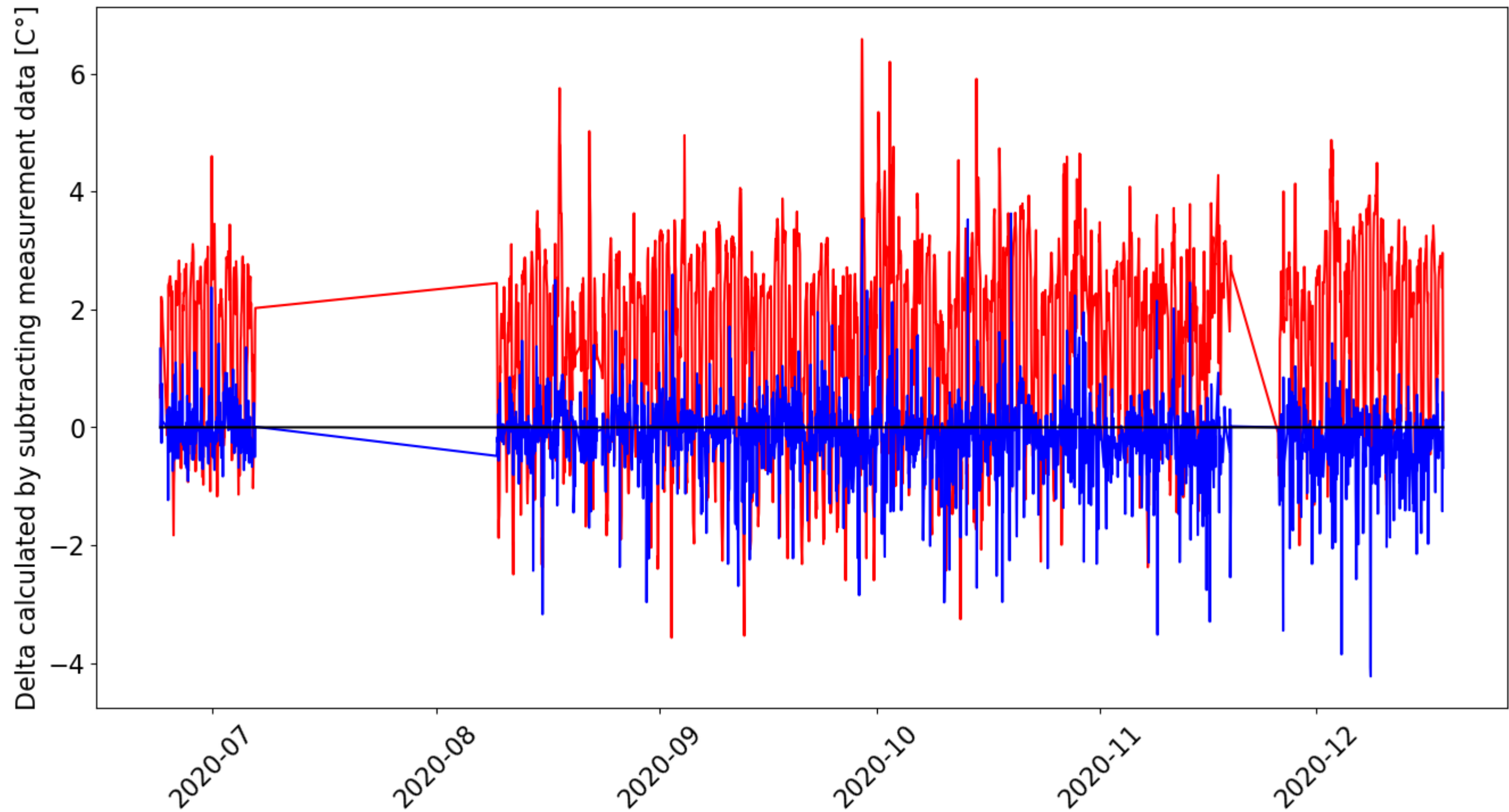


RMSE reconstructed: 0.68 C°  
RMSE ERA5 nearest point: 2.15 C°

Correlation reconstructed: 0.933  
Correlation ERA5 nearest point: 0.445



## Barbados - 2020

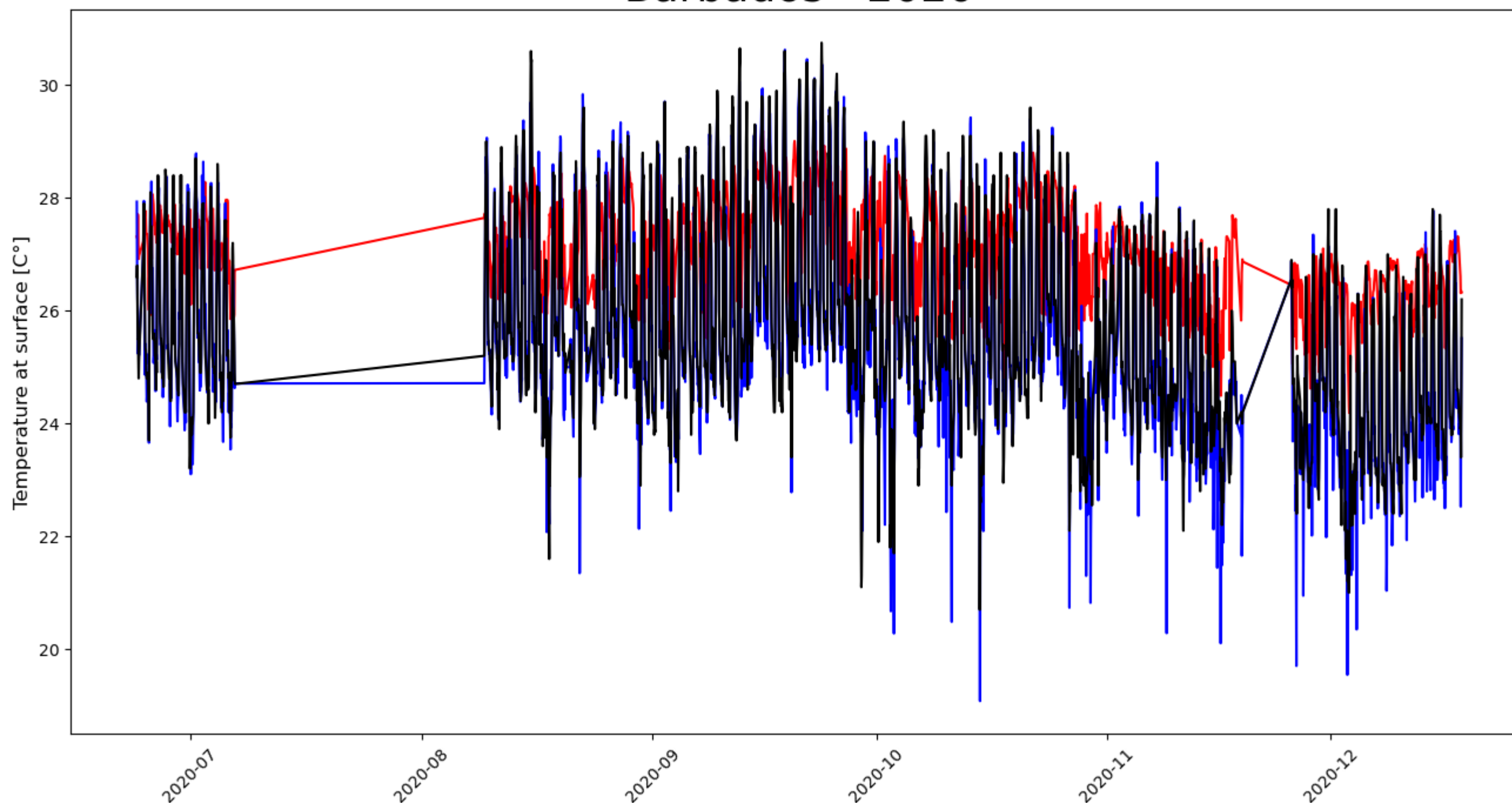


RMSE reconstructed: 0.68 C°  
RMSE ERA5 nearest point: 2.15 C°

Correlation reconstructed: 0.933  
Correlation ERA5 nearest point: 0.445

— ERA5 nearest point  
— Reconstructed for validation  
— Measurements

## Barbados - 2020

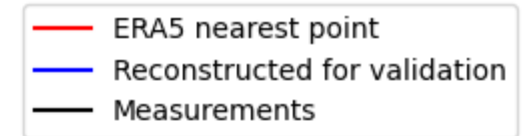


...over random weeks

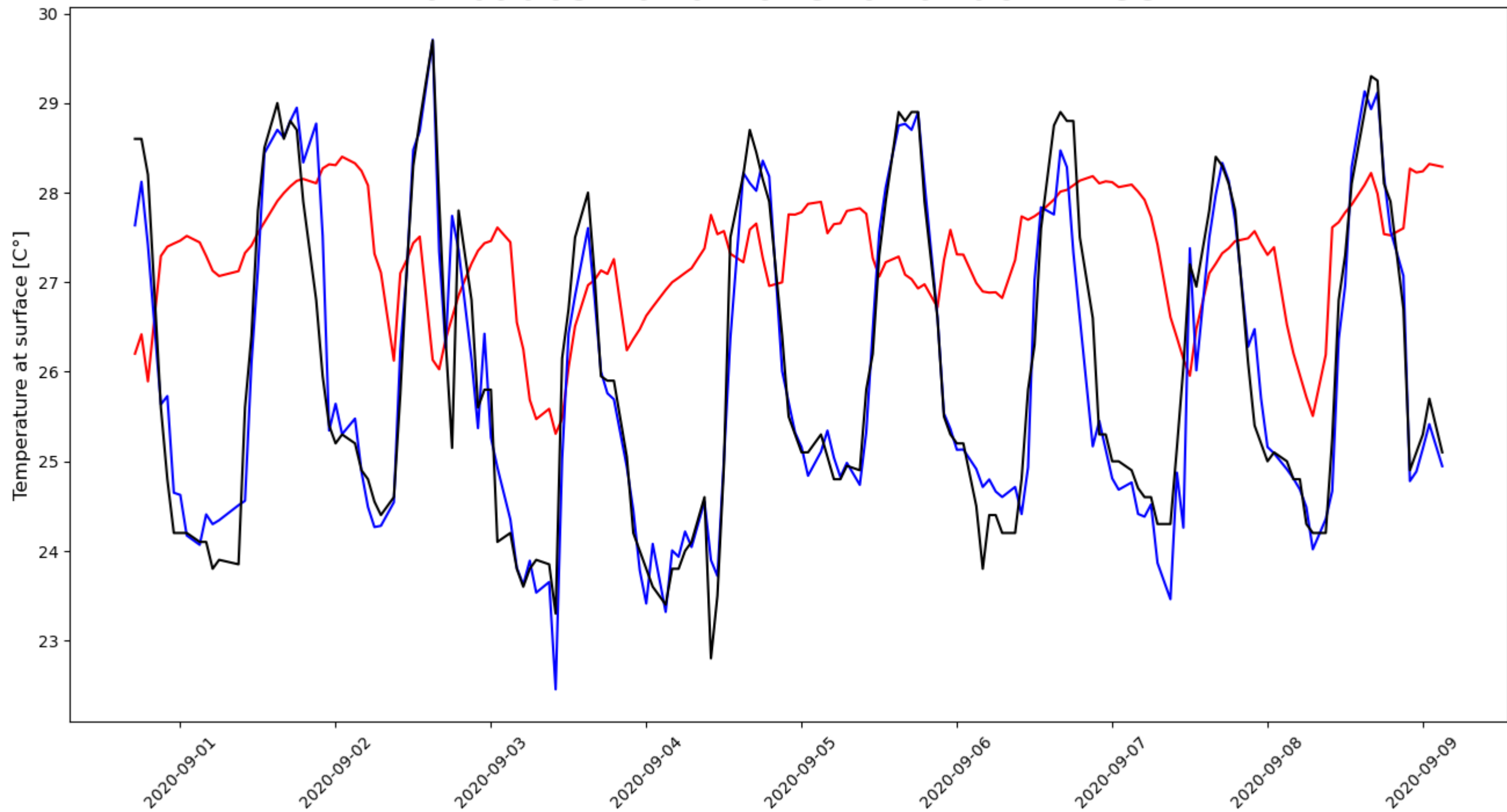
```
In [ ]: n = 168
if n == 168:
    title = f"{station_name} {test_year} - over a random week"
else:
    title = f"{station_name} {test_year} - {n} random consecutive hourly steps"
plot_n_steps_of_df(hourly_df, as_delta=False, n=n, title=title)
plot_n_steps_of_df(hourly_df, as_delta=False, n=n, title=title)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

RMSE reconstructed: 0.53 C°  
RMSE ERA5 nearest point: 2.18 C°

Correlation reconstructed: 0.951  
Correlation ERA5 nearest point: 0.175

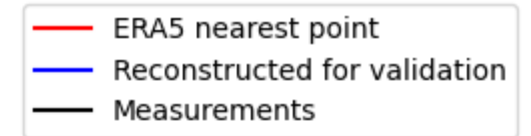


## Barbados 2020 - over a random week

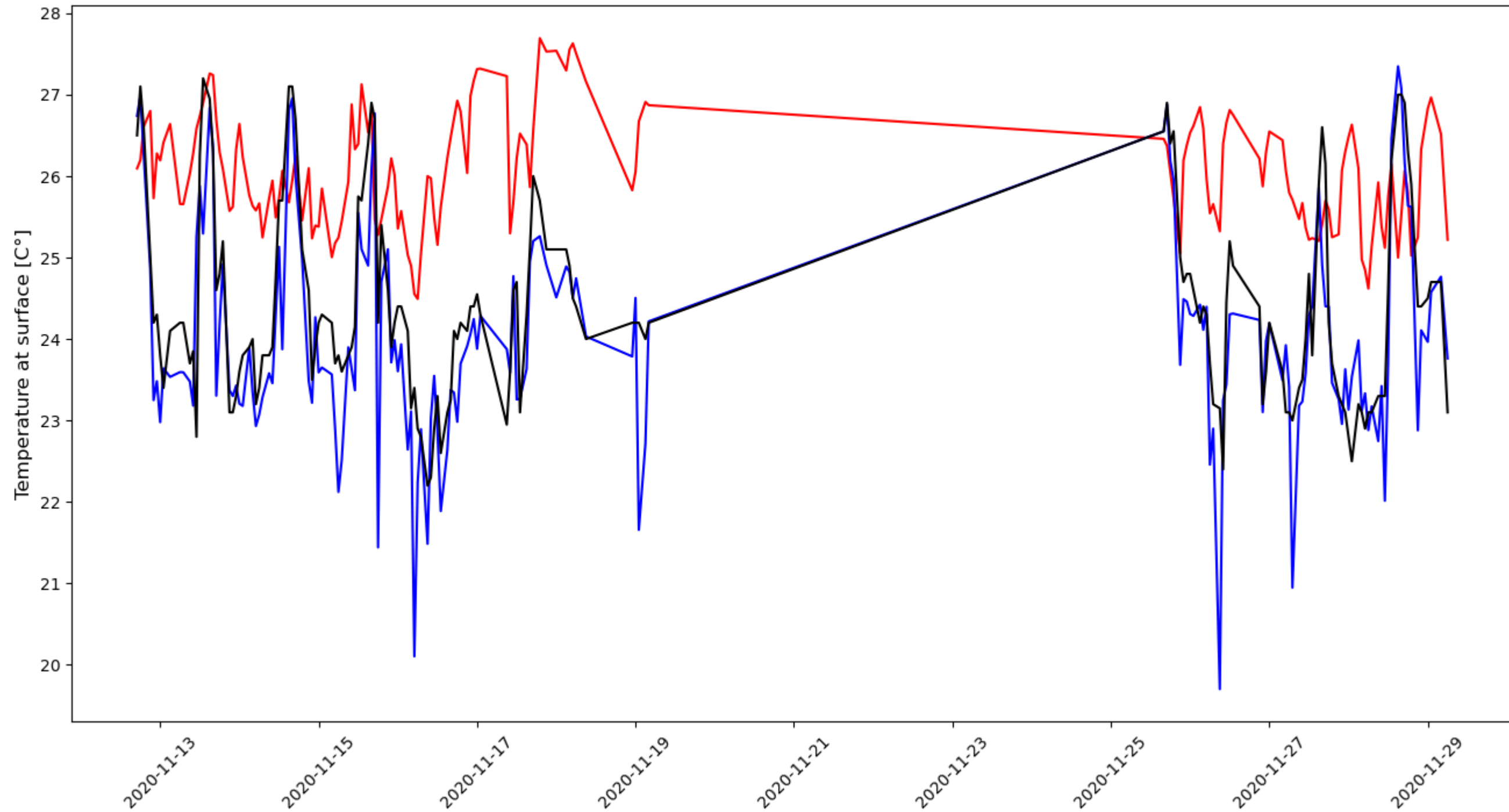


RMSE reconstructed: 0.83 C°  
RMSE ERA5 nearest point: 2.07 C°

Correlation reconstructed: 0.830  
Correlation ERA5 nearest point: 0.232



## Barbados 2020 - over a random week

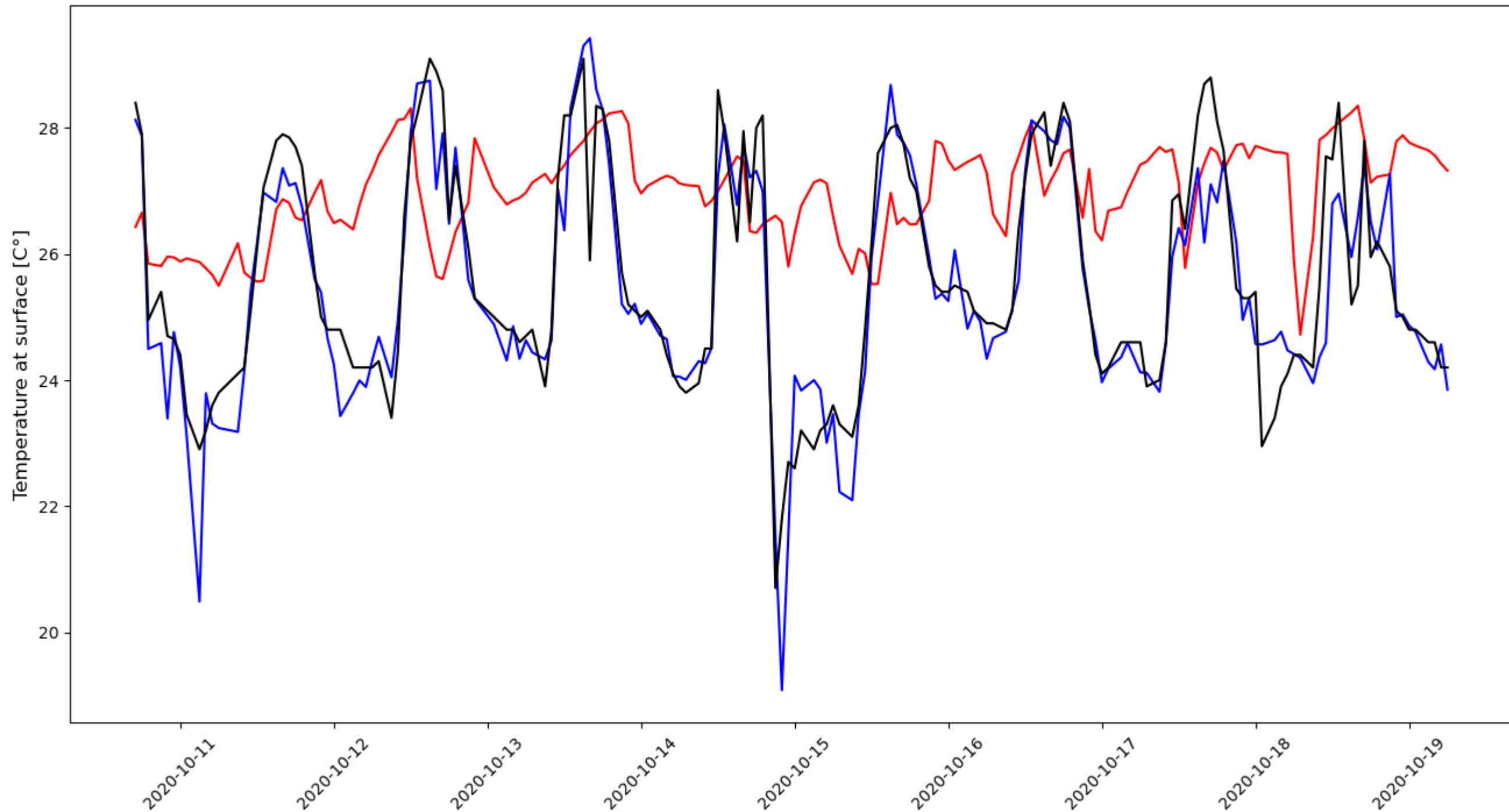


RMSE reconstructed: 0.77 C°  
RMSE ERA5 nearest point: 2.25 C°

Correlation reconstructed: 0.907  
Correlation ERA5 nearest point: 0.177

— ERA5 nearest point  
— Reconstructed for validation  
— Measurements

## Barbados 2020 - over a random week



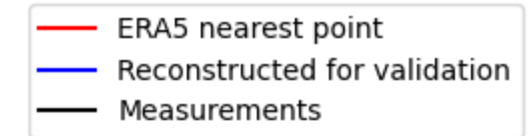
## Resample Data from hourly to daily or monthly

```
In [ ]: # resample to daily mean
daily_df = hourly_df.resample("D").mean()
# drop nans
daily_df = daily_df.dropna()
title = f"{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)
```

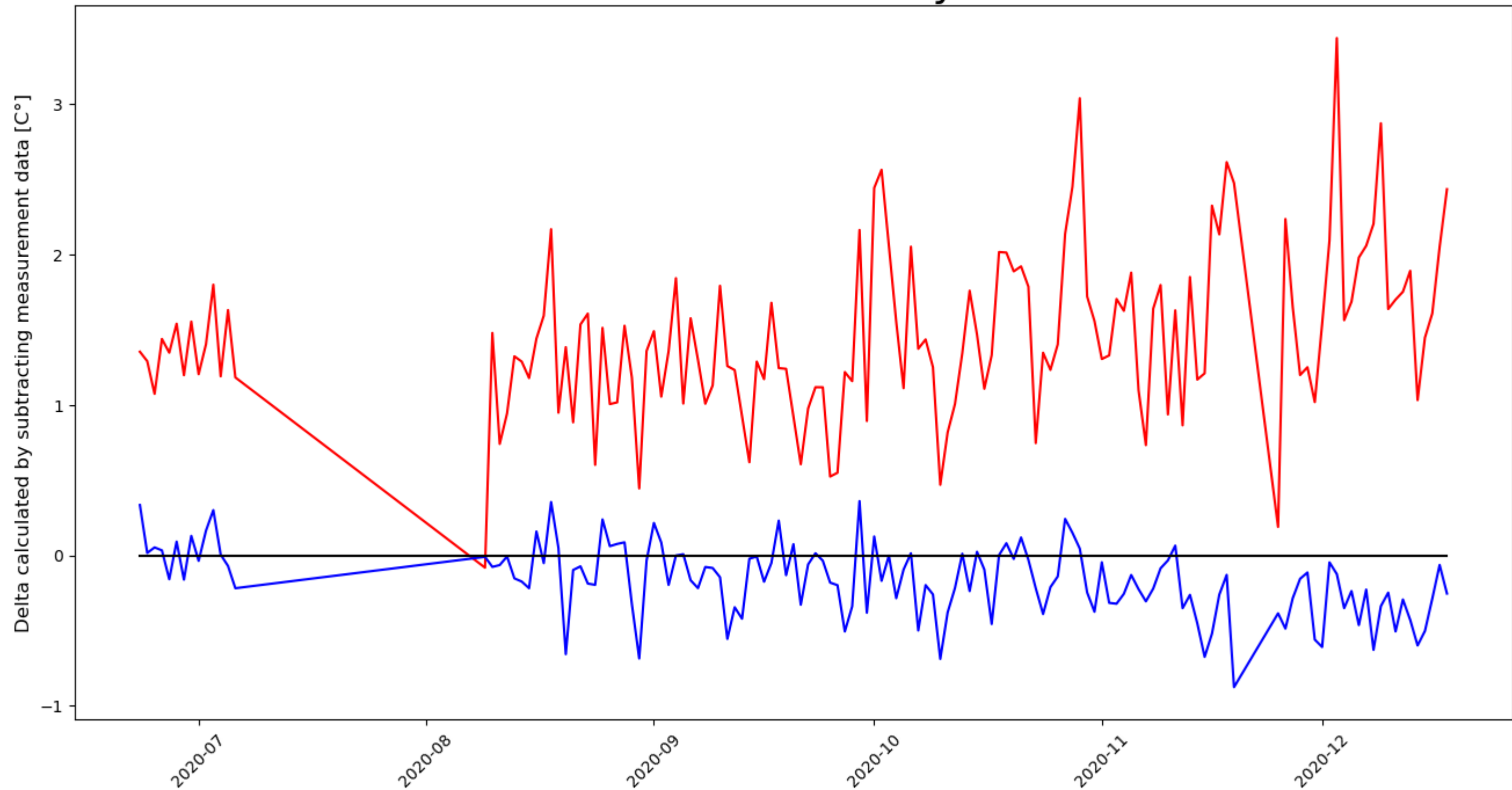


RMSE reconstructed: 0.29 C°  
RMSE ERA5 nearest point: 1.55 C°

Correlation reconstructed: 0.975  
Correlation ERA5 nearest point: 0.820



## Barbados 2020 - daily mean

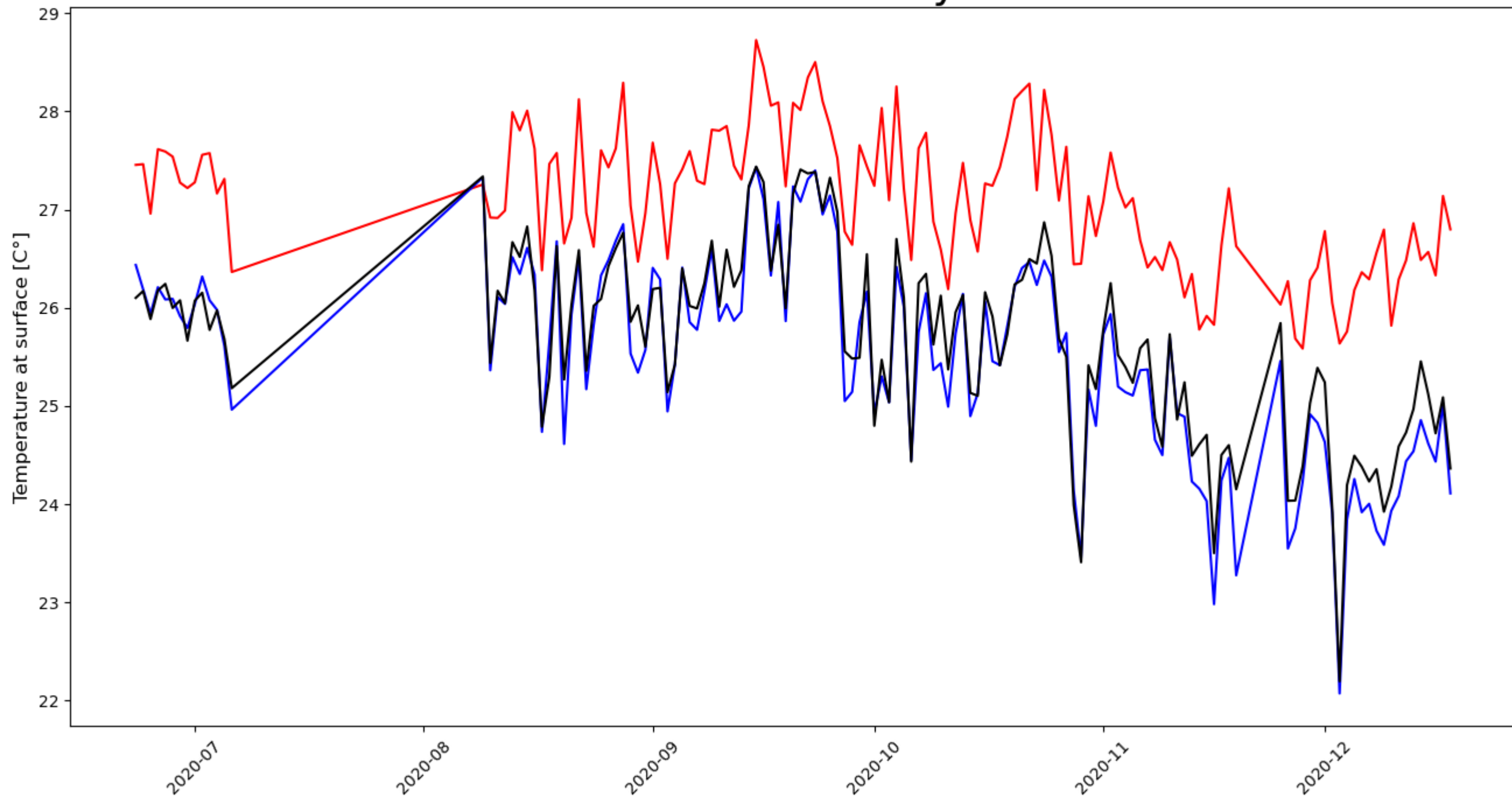


RMSE reconstructed: 0.29 C°  
RMSE ERA5 nearest point: 1.55 C°

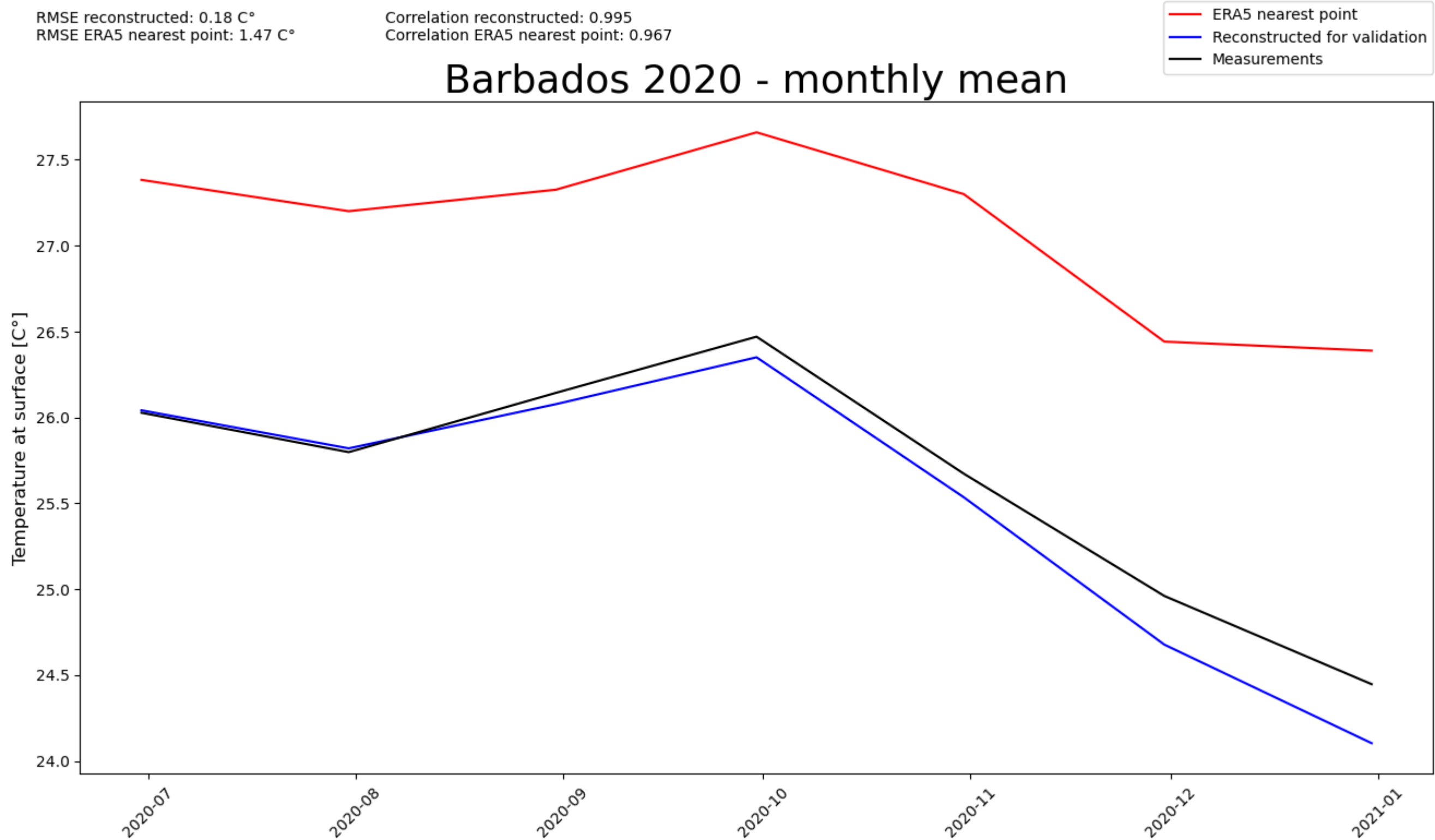
Correlation reconstructed: 0.975  
Correlation ERA5 nearest point: 0.820

— ERA5 nearest point  
— Reconstructed for validation  
— Measurements

## Barbados 2020 - daily mean



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



## Average Course of the day

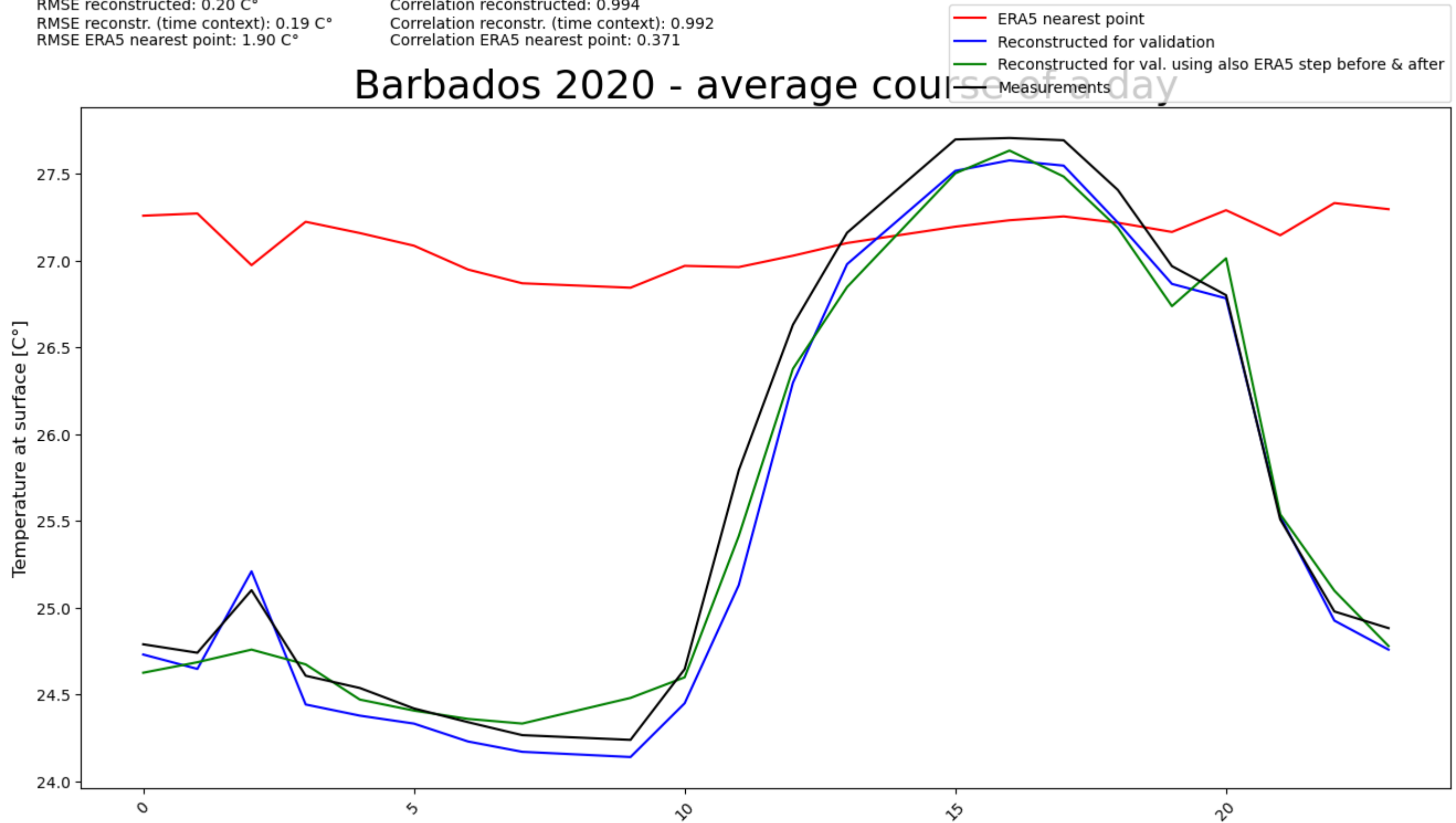
```
In [ ]: # calculate the mean of each 24 hours over the whole year

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



RMSE reconstructed: 0.20 C°  
 RMSE reconstr. (time context): 0.19 C°  
 RMSE ERA5 nearest point: 1.90 C°

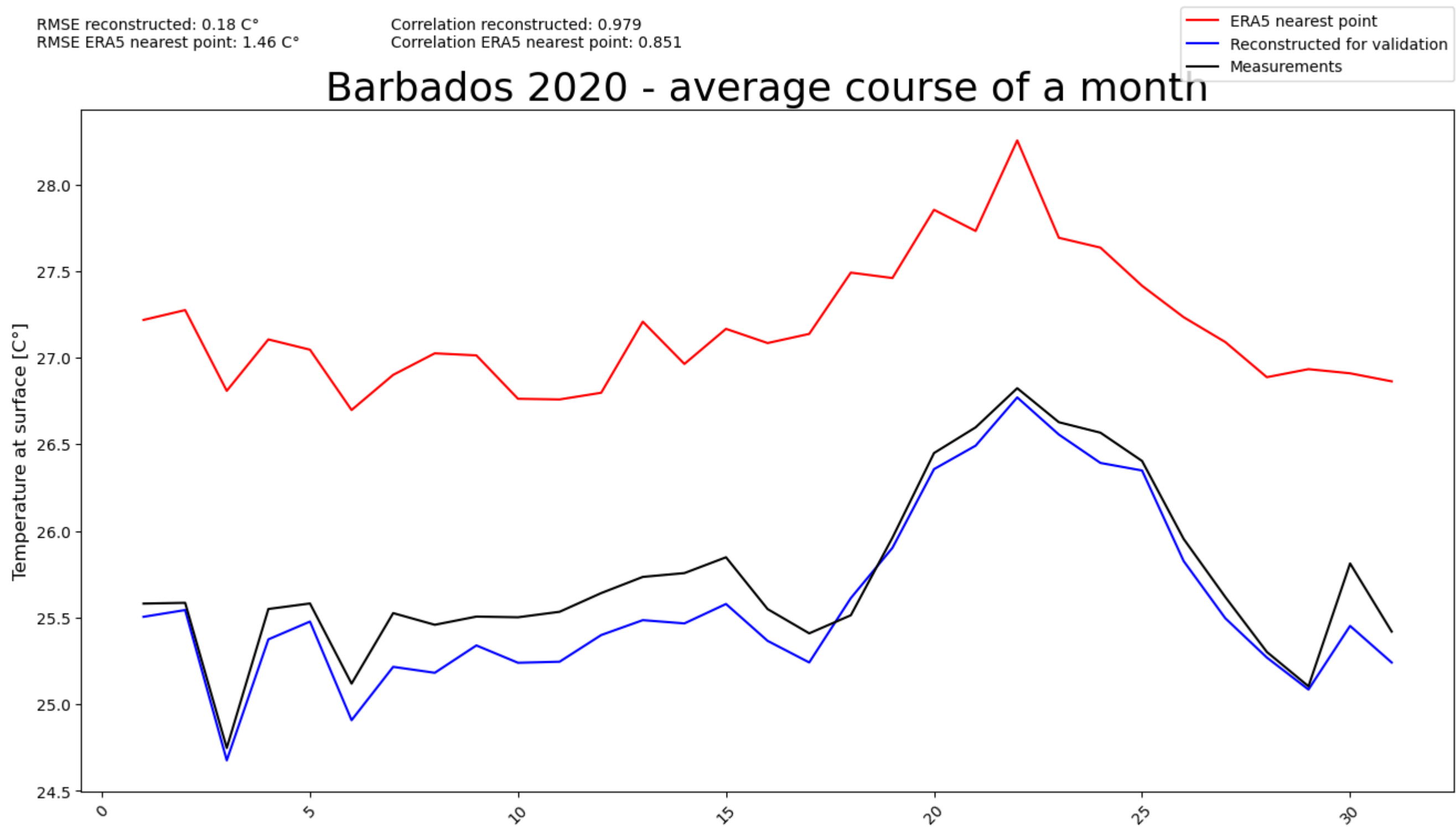
Correlation reconstructed: 0.994  
 Correlation reconstr. (time context): 0.992  
 Correlation ERA5 nearest point: 0.371



## 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"{station_name} {test_year} - average course of a month"
plot_n_steps_of_df(month_course_df, as_delta = False, title=title)
```



### Evaluate for the actually missing time steps

```
In [ ]: evaluate_wrapper(use_time_context = False, iter = at_iterations, station = station_name, reconstruct_gaps=True)
output_filled_gaps_ds = DataSet(output_file_path, name="Reconstructed gaps")
output_actual_gaps_data = xr.open_dataset(output_file_path)
```

```
/home/k/k203179/.conda/envs/crai/lib/python3.10/site-packages/climate reconstructionai/ 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, 6.71s/it]
```

```

In [ ]: def create_filled_in_gaps_df(full_era5_data, filled_gaps_data, measurements_data):

    import pandas as pd

    def single_df(time, tas, name):
        df = pd.DataFrame()
        df["time"] = time
        df.set_index("time", inplace=True)
        df[name] = tas
        return df

    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 = full_era5_data.variables["tas"][:, lon_nearest_idx, lat_nearest_idx]
    reconstructed_values = filled_gaps_data.variables["tas"].stack(grid=['lat', 'lon']).values
    reconstructed_values = [np.median(x) for x in reconstructed_values]

    measurements_values = measurements_data.variables["tas"].stack(grid=['lat', 'lon']).values
    measurements_values = [np.median(x) for x in measurements_values]

    era_df = single_df(full_era5_data.variables["time"], era5_nearest_values, "era5_nearest")
    reconstructed_df = single_df(filled_gaps_data.variables["time"], reconstructed_values, "filled_in_gaps")
    measurements_df = single_df(measurements_data.variables["time"], measurements_values, "measurements")

    # merge on time
    merged = pd.concat([era_df, reconstructed_df, measurements_df], axis=1)
    # drop rows where reconstructed and measurements are both nan simultaneously
    return merged.dropna(subset=["filled_in_gaps", "measurements"], how="all")

```

```

In [ ]: hourly_filled_gaps_df = create_filled_in_gaps_df(era5_full_data, output_actual_gaps_data, measurements_data)
hourly_filled_gaps_df

```

Out[ ]: era5\_nearest filled\_in\_gaps measurements

time			
2020-06-08 15:00:00	299.021698	300.837708	NaN
2020-06-08 16:00:00	299.369385	300.234680	NaN
2020-06-08 17:00:00	299.592957	302.066528	NaN
2020-06-08 18:00:00	299.607269	301.066132	NaN
2020-06-08 19:00:00	299.687927	302.042786	NaN
...	...	...	...
2023-11-22 14:00:00	300.397003	300.572327	NaN
2023-11-22 15:00:00	300.451721	NaN	300.45
2023-11-22 16:00:00	300.482544	NaN	300.75
2023-11-22 17:00:00	300.510742	NaN	300.55
2023-11-22 18:00:00	300.620239	NaN	300.10

30292 rows x 3 columns

```
In [ ]: def plot_n_steps_of_filled_in_df(df, n=None, title=None):

    time = df.index.values

    era5_nearest_values = df["era5_nearest"].values - 273.15
    reconstructed_median_values = df["filled_in_gaps"].values - 273.15

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

    import random

    if n is None:
        n = len(df)

    break_loop = 0
    we_have_measurements_and_reconstructed_values_in_this_timeframe = False
    while not we_have_measurements_and_reconstructed_values_in_this_timeframe and break_loop < 1000:

        # random slice of n consecutive datapoints

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

        if n == len(df):
            we_have_measurements_and_reconstructed_values_in_this_timeframe = True
        else:
            break_loop += 1
            # check in time slice if we have measurements and reconstructed values
            we_have_measurements_and_reconstructed_values_in_this_timeframe = not (np.all(np.isnan(measurements_values[time_slice])) or np.all(np.isnan(reconstructed_median_v

    time = time[time_slice]

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

```

plt.plot(time, era5_nearest_values[time_slice], label="ERA5 nearest point", color="grey", linestyle="--" if n <= 168 else "-")
# plt.plot(time, era5_mid_values[time_slice], label="ERA5 nearest 4 points")

plt.plot(time, reconstructed_median_values[time_slice], label="Reconstructed", color="#d13297", linestyle="-", marker="o" if n <= 168 else "")
plt.plot(time, measurements_values[time_slice], label="Measurements", color="black", linestyle="-", marker="o" if n <= 168 else "")

# 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': 26})

plt.legend()

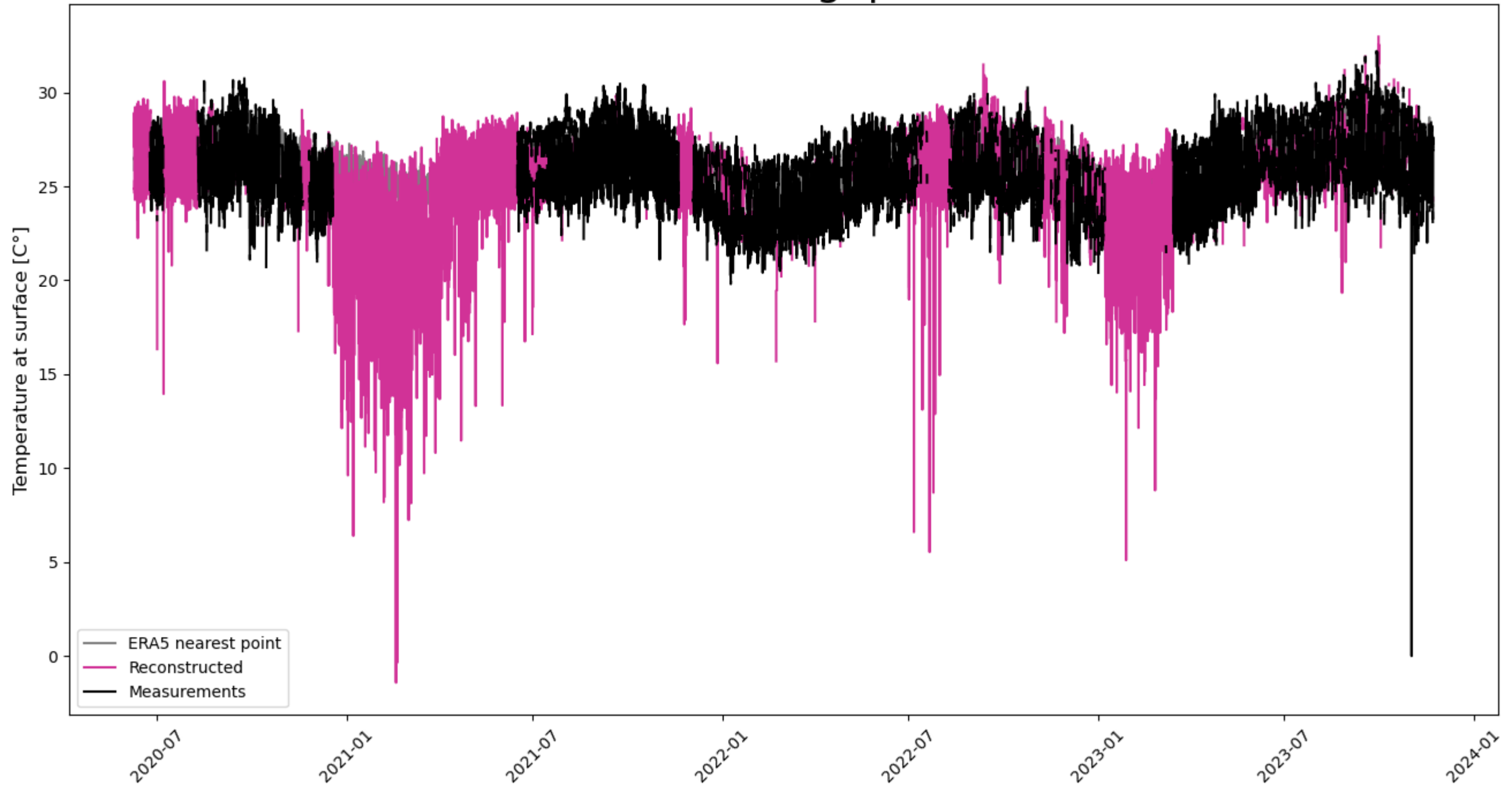
# figure size A4 landscape
plt.gcf().set_size_inches(16, 8)

plt.show()

```

```
In [ ]: plot_n_steps_of_filled_in_df(hourly_filled_gaps_df, title=f"{station_name} - reconstructed gaps in measurements")
```

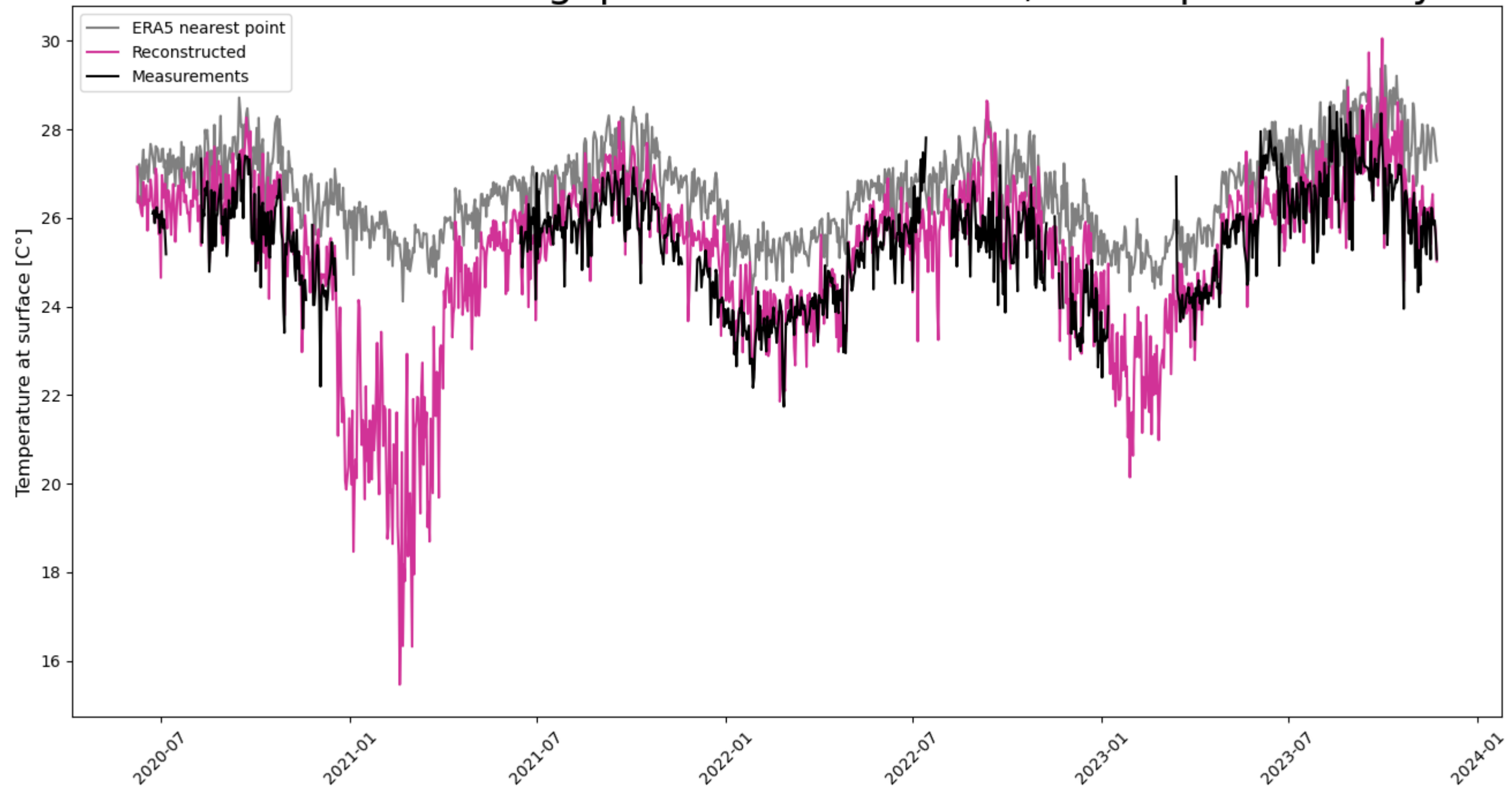
## Barbados - reconstructed gaps in measurements



### Resampled to Daily Mean

```
In [ ]: daily_filled_gaps_df = hourly_filled_gaps_df.resample("D").mean()  
plot_n_steps_of_filled_in_df(daily_filled_gaps_df, title=f"{station_name} - reconstructed gaps in measurements, resampled to daily mean")
```

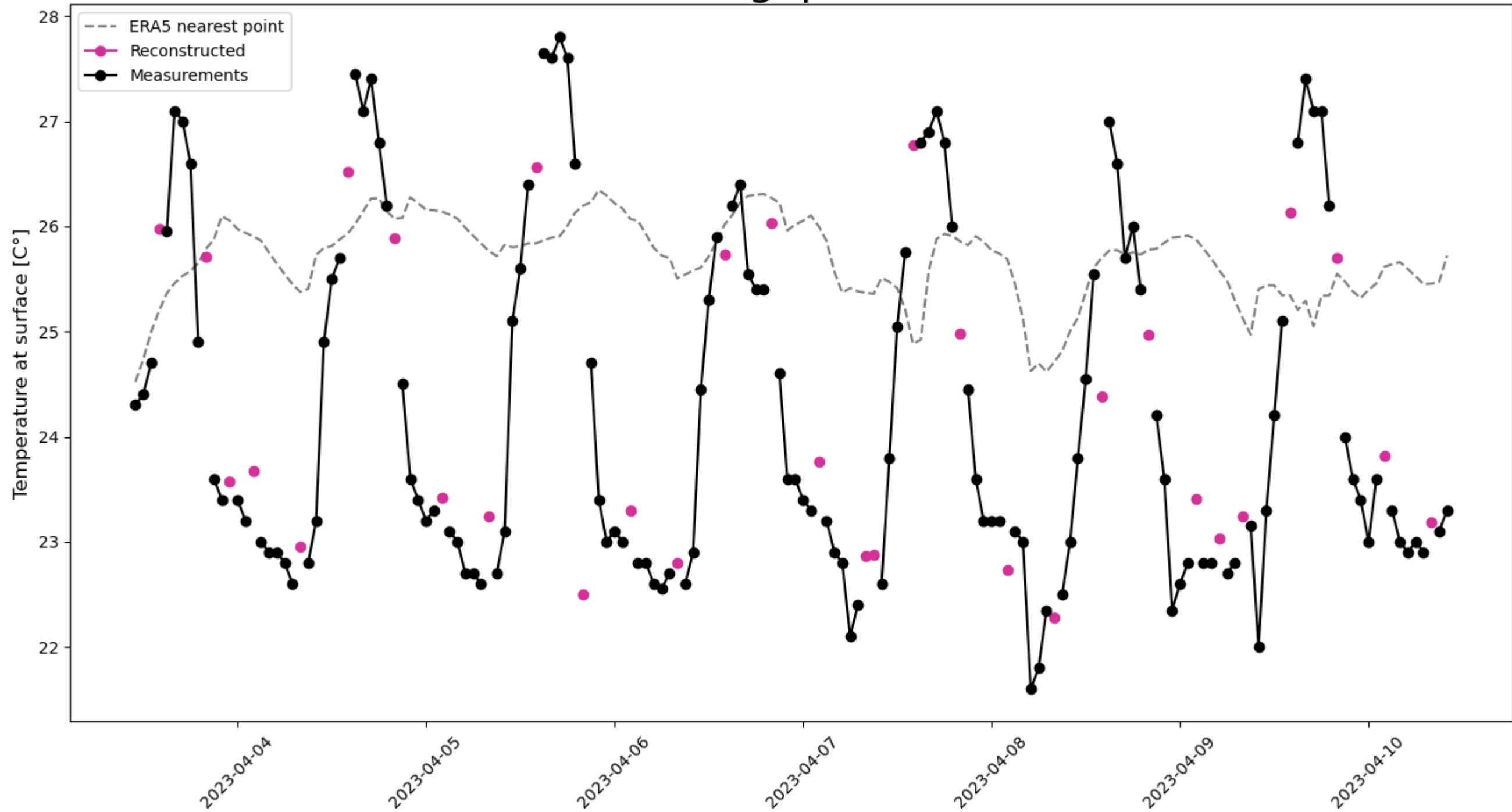
# Barbados - reconstructed gaps in measurements, resampled to daily mean



## Plot random weeks

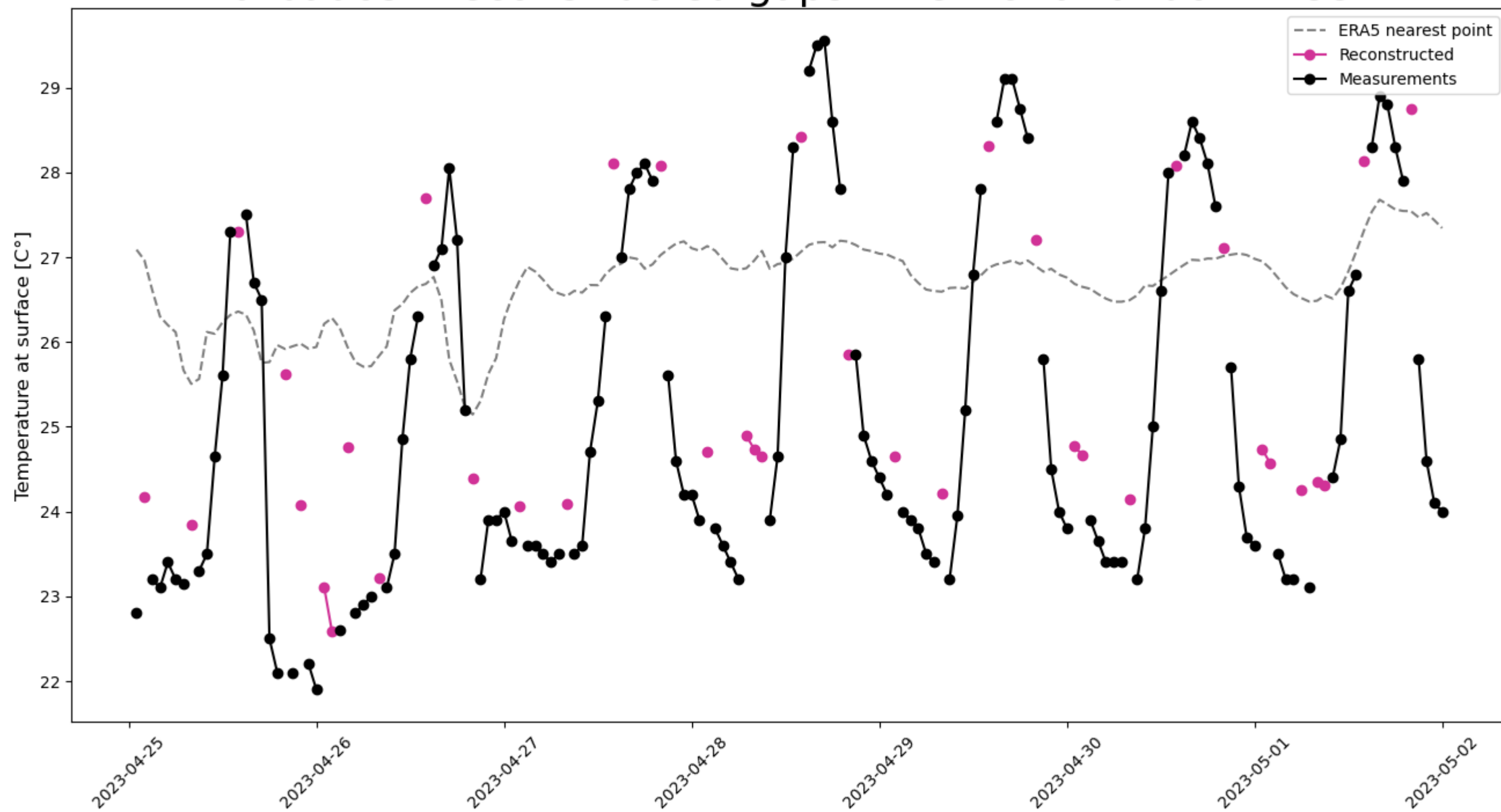
```
In [ ]: for i in range(10):  
        plot_n_steps_of_filled_in_df(hourly_filled_gaps_df, n=168, title=f"{station_name} - reconstructed gaps - view of a random week")
```

# Barbados - reconstructed gaps - view of a random week

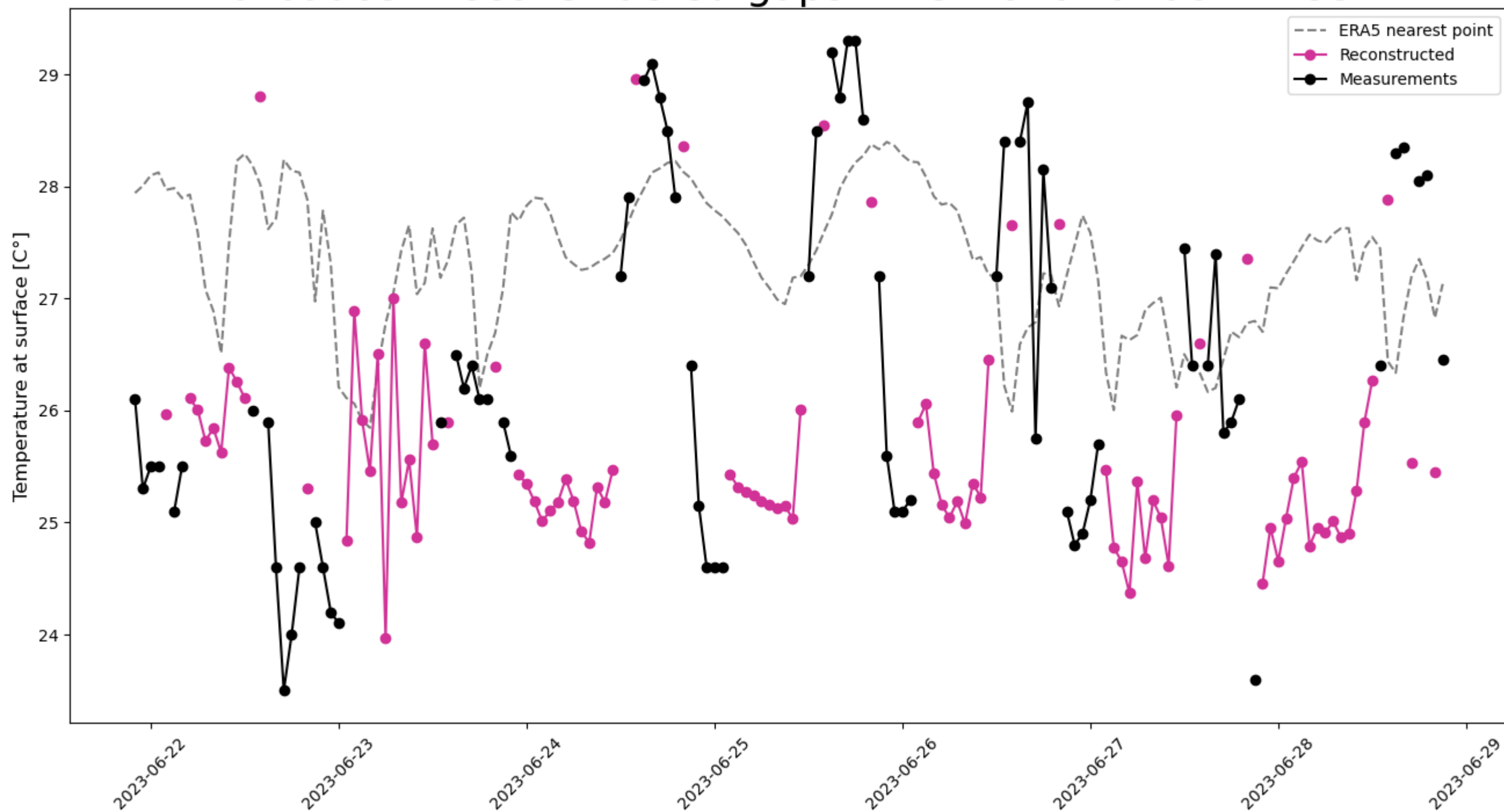




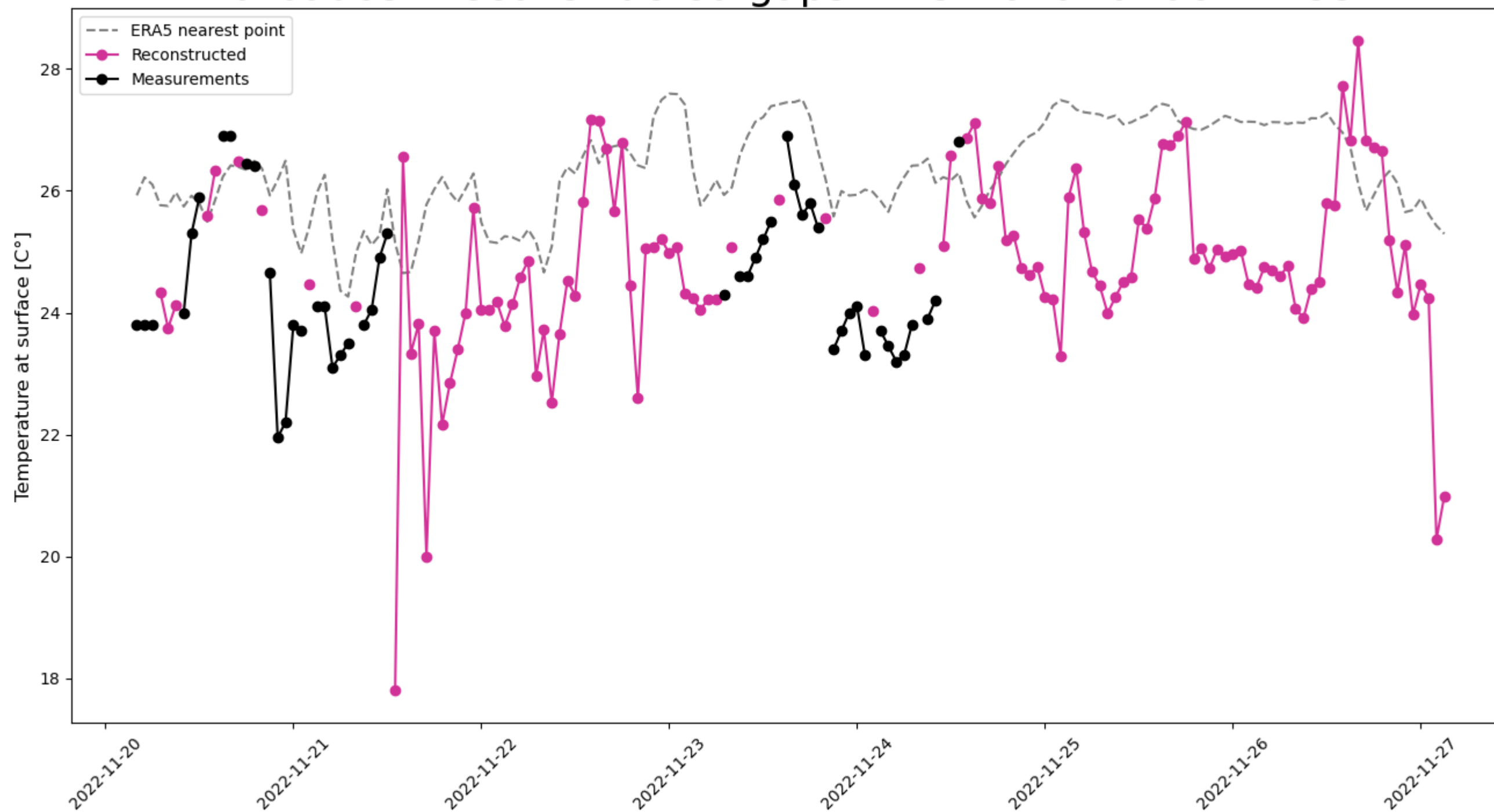
# Barbados - reconstructed gaps - view of a random week



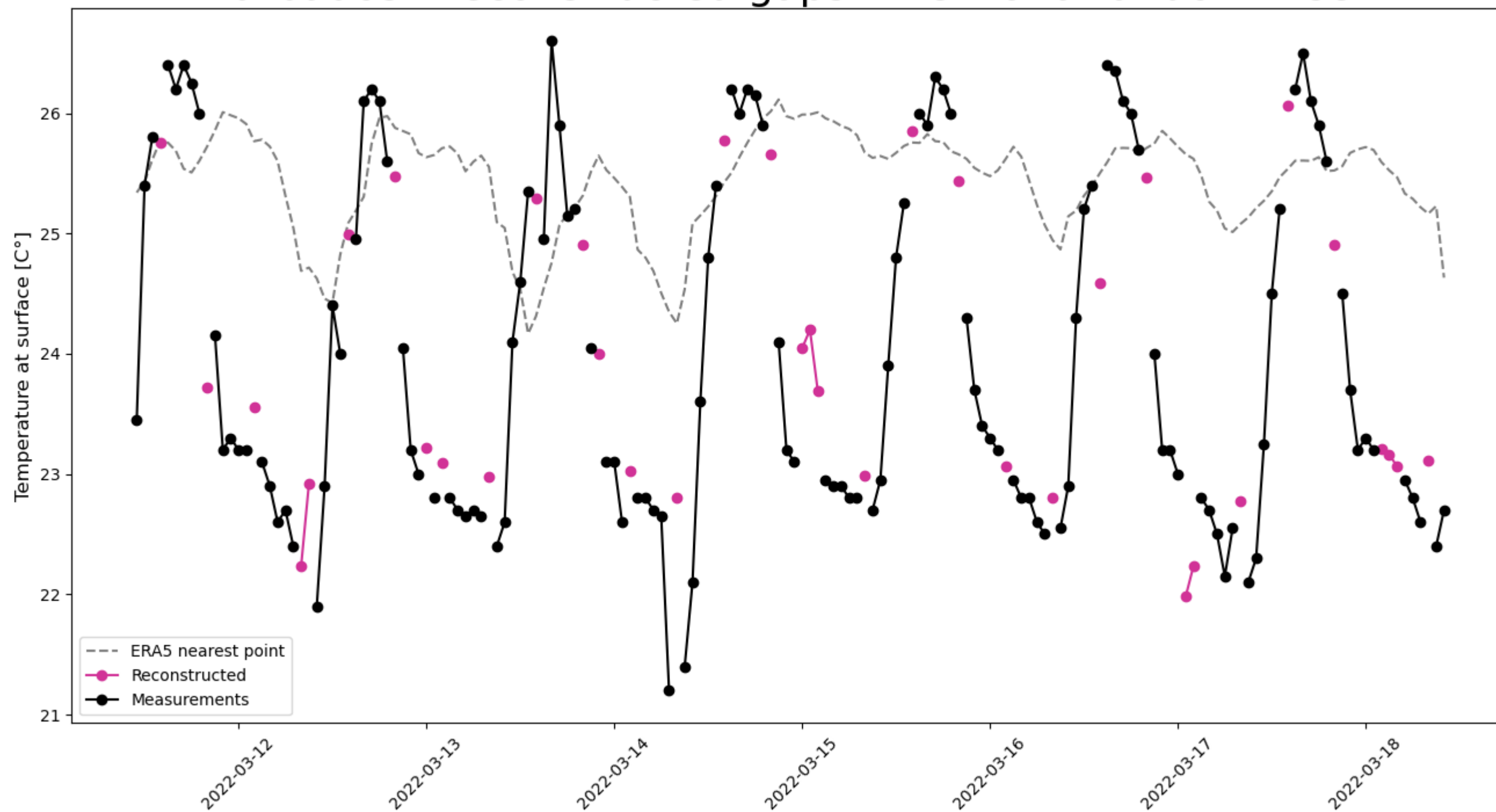
# Barbados - reconstructed gaps - view of a random week



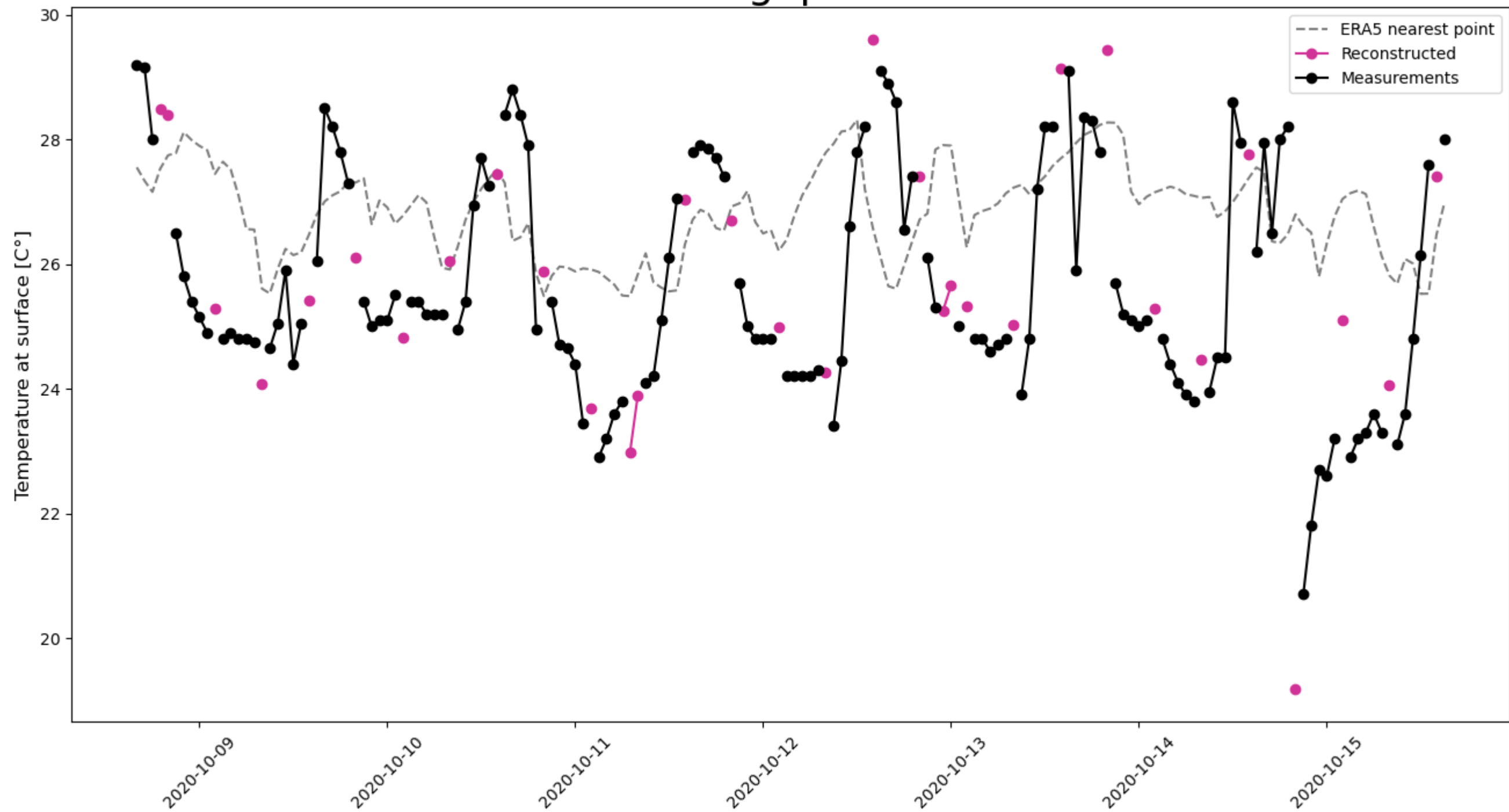
## Barbados - reconstructed gaps - view of a random week



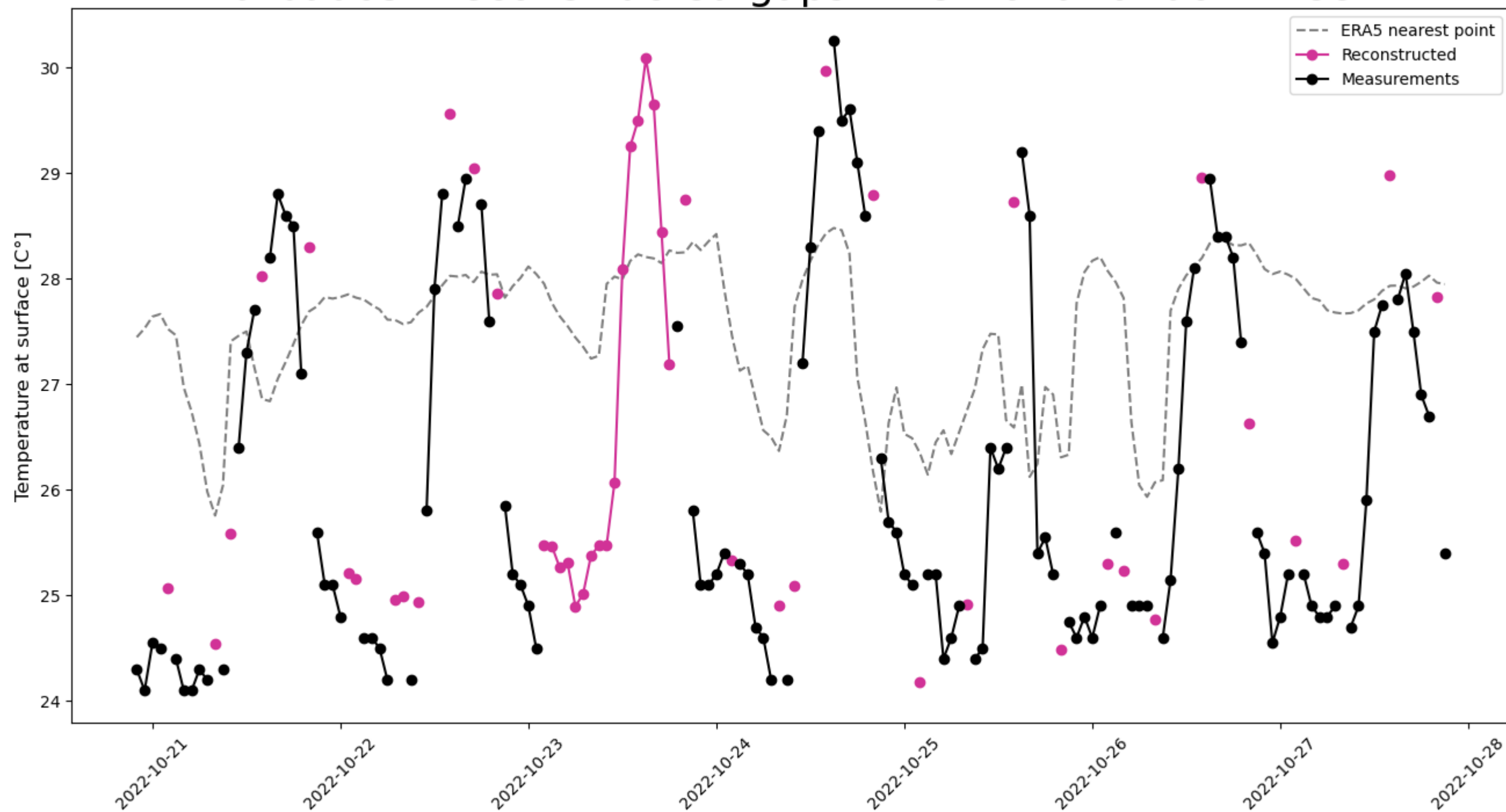
Barbados - reconstructed gaps - view of a random week



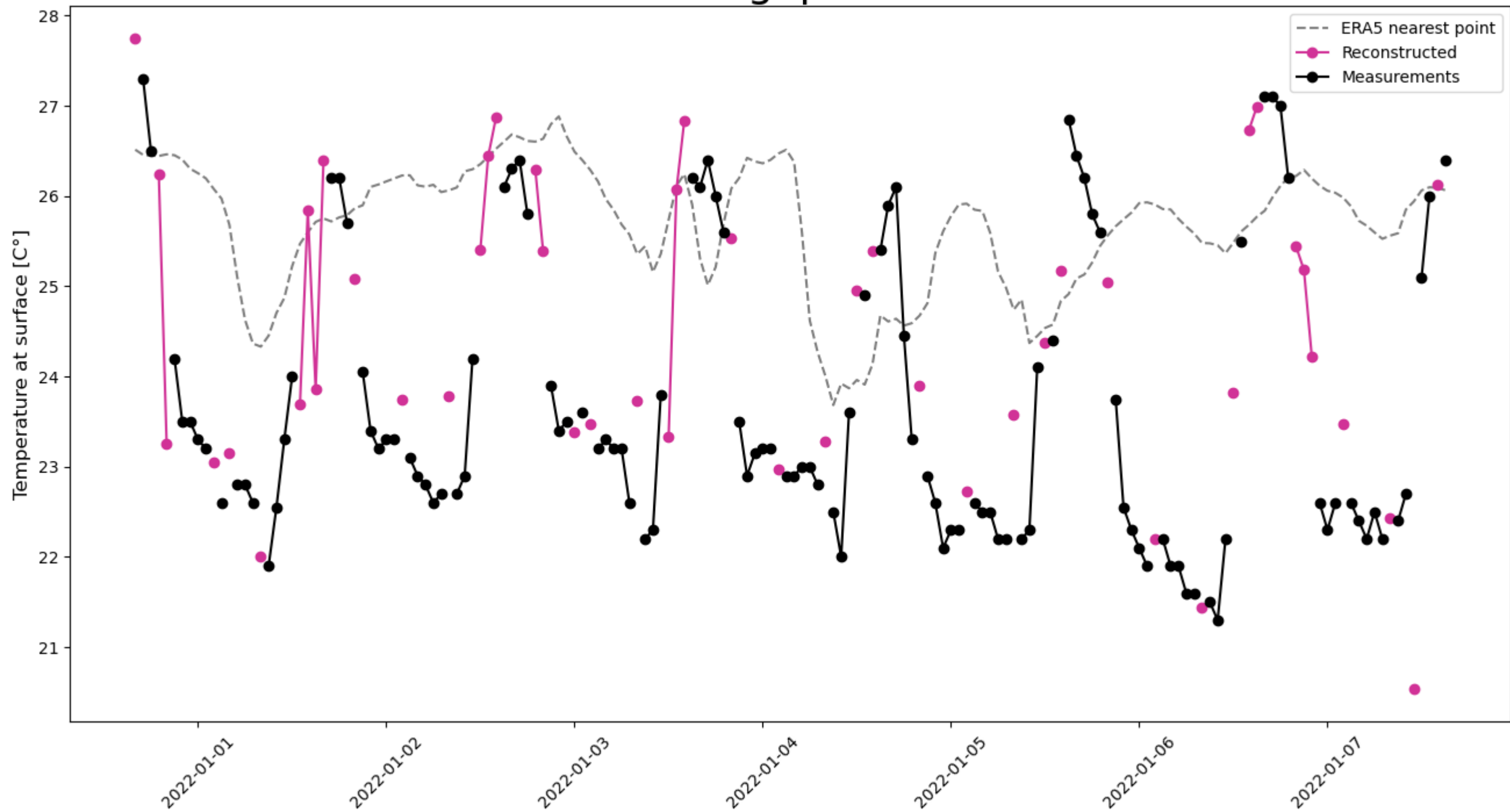
# Barbados - reconstructed gaps - view of a random week



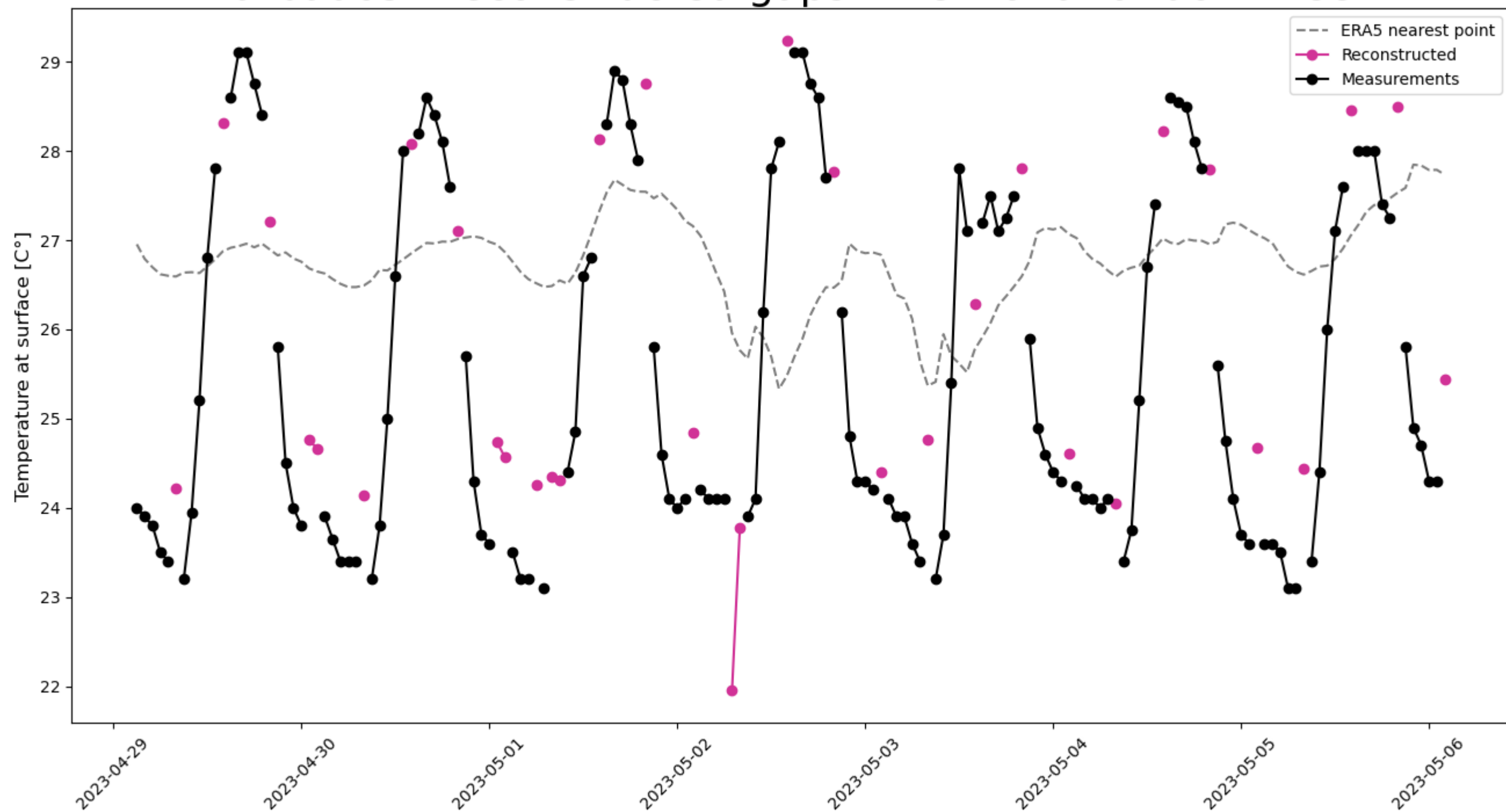
# Barbados - reconstructed gaps - view of a random week



# Barbados - reconstructed gaps - view of a random week



# Barbados - reconstructed gaps - view of a random week





# Barbados - reconstructed gaps - view of a random week

