Objective: To verify gridded NBM 4.0 PQPF against gridded URMA QPE across WR (and some CR) CWAs using contingency tables, reliability metrics and diagrams, relative operating characteristics, and brier skill metrics.

Results Presented: (1) WR STID Science Round Table 2/1/2021, (2) WR STID Science Round Table 2/10/2021, (3) WR/WPC URMA Comparison 4/6/2021, 5/7/2021

Tools:

Anaconda - Easily manage python environment and packages/modules

JuPyter Notebook/Lab - Much of the development code was written in JuPyter notebooks, which are a great way to visualize and interact with the interactive iPython environment. These can be read as standalone notebooks or in the JuPyter "Lab" environment. Unfortunately, have not found a way to work with this on science3 or other WRH boxes due to security limitations.

Boto - Access, search, and download from Amazon AWS/S3 Buckets

NetCDF4 - Python module for reading/writing netCDF files (used as a backend)

Pygrib - Python module for reading/writing grib files (used both directly and as backend)

XArray - Effectively manage large gridded datasets in Python

Pandas - Effectively manage large tabular datasets in Python

Multiprocessing, Functools - Built-in module that allows for both multiple threads or processes to run functions and return a result. Large returns in speed when used for truly parallel tasks with low memory load. *Functools.Partial* allows for fixed arguments to be passed with the functions.

GeoPandas - Excellent utility for reading and managing shapefile data in python, including the ability to query and dissolve polygons

RasterIO - Handles geographic information, especially GeoTIFFs and GeoJSON, with functions for masking, querying, and regridding raster datasets.

RegionMask - Used in conjunction with a shapefile (via GeoPandas) to create a True/False mask in XArray that can be directly applied to an XArray Dataset or DataArray. Highly useful for subsetting data to a CWA or forecast zone. Can be used on the fly to calculate areal statistics on gridded data.

CartoPy - Actively supported mapping package (as opposed to the deprecated basemap) for plotting preconfigured political maps/boundaries, shapefiles, point data, and gridded data on multiple projections. Not an excellent solution, and there may be better plotting options going forward, but this is the community-supported package at this time.

Scikit-Learn (sklearn) - Used here for a few specific functions, this machine learning Python package is full of utility. *Sklearn.calibration_curve* produces reliability diagrams which have been verified correct when compared with custom reliability diagram code (derived directly from textbook equations). This provides a more compact and easily applied solution (easily applied to XArray) than some of the custom functions. *Sklearn.roc_curve, roc_auc_score* are used in producing the ROC diagrams and ROC/AUC scores, and have been similarly validated.

Dask - While not currently used in any of these scripts, dask allows for computing analyses over large datasets without having to read the entire dataset into memory at once. However, it remains fairly complex to apply and doesn't work incredibly well with many of the analysis tools used here. There may be some potential to use xarray, dask, and scikit-learn to greatly streamline the process in the future, cutting out a lot of the intermediate data processing that is done to reduce memory load.

Many others including **glob, datetime, system, pyplot** which are more standard packages and included within the base python install.

Process/Workflow:

- **Acquisition:** The Amazon Web Services (AWS) S3 repository of NBM 4.0 QMD QPF is a quick and easy way to access and download the NBM grids. Documentation can be found here: https://registry.opendata.aws/noaa-nbm/. The S3 resource name is: arn:aws:s3:::noaa-nbm-pds and is used within boto (or whichever client) to access the data. A web-based browser for the bucket can be found here: https://noaa-nbm-grib2-pds.s3.amazonaws.com/index.html.

  As the hosted files are in grib format, they can either be downloaded and then subset with wgrib2 locally, or can be subset before downloading using curl. An example of this can be found in **./scraps/scripts_latest/aws_NBM_get_subset.py** (described below in docs). It is suggested that any code developed going forward is subset inline like this as it is much less process and memory intensive. In all cases, it is recommended to write this data to disk rather than read from boto to memory given the limitations on the science boxes.

  No data acquisition script was needed for URMA data, as it is already being archived on /nas, though care should be taken not to use URMA data inside of the valid time +7 days (see note below). A simple script is then used to subset the URMA data to WR (and some CR CWAs).

- **Preprocessing:** URMA data are archived as 6-hourly accumulated precipitation. These need to be accumulated to verify the NBM 24-hourly PQPF. This is handled by a script that resamples the 6-hourly data in xarray to 24-hourly data (and verified to be accumulating to the correct valid time). The 24-hourly data was previously saved out to a single netCDF, which works fine for static applications but should be broken into monthly files more easily appended to if growing the dataset in real time/daily. Alternatively, if a memory-efficient way was used, it might be possible to just read and compile the 6-hourly files on the fly, though currently this imposes a significant memory load through XArray (dask may prove one option).

  NBM PQPF comes in as grib2, though to more easily interface with XArray, is repacked as netCDF. Currently these are packed in the archive as follows:

```
ncdump -h blend.202112.qmd.f024.WR.nc
netcdf blend.202112.qmd.f024.WR {
dimensions:
        valid = 62 ;
        y = 1051 ;
        x = 1132 ;
        threshold = 8 ;
variables:
        int64 valid(valid) ;
                valid:units = "hours since 2000-01-01" ;
                valid:calendar = "proleptic_gregorian" ;
        double lat(y, x) ;
                lat:_FillValue = NaN ;
        double lon(y, x) ;
                lon:_FillValue = NaN ;
        int64 init(valid) ;
                init:units = "hours since 2000-01-01" ;
                init:calendar = "proleptic_gregorian" ;
        int64 interval ;
        int64 step ;
                step:units = "days" ;
        int64 fhr ;
        double threshold(threshold) ;
                threshold:_FillValue = NaN ;
        double threshold_in(threshold) ;
                threshold_in:_FillValue = NaN ;
        float probx(threshold, valid, y, x) ;
                probx:_FillValue = NaNf ;
                probx:coordinates = "lon fhr init lat interval step threshold_in"}
```

  In addition to the lat,lon grid, the data are packed by valid time and threshold values (0.01”, 0.1”, etc..). These can then be read in by XArray and concatenated along the threshold or forecast hour dimension on the fly for analysis and plotting.

- **Analysis:** Once the data from URMA and NBM PQPF are read into XArray, analysis can begin. These datasets are on identical grids, making the analysis fairly simple, though there should be a failsafe step that checks the max/min lat/lon of each grid against each other and trims as needed. However, if clipping with a shapefile via regionmask, this issue will generally remedy itself. Valid time should also be clipped so that the datasets are aligned. At this point, both custom functions included in the scripts or canned functions included in the scikit-learn module can be used to perform the analysis.

In order to reduce the memory pressure, most of the example scripts subset the datasets to a CWA before concatenating them along the valid time or forecast hour dimension. In addition, it may be helpful to only process one threshold at a time if needed, then exporting the results to a CSV file and producing the plots from an aggregate of that text-based data. This is the intended solution to limited memory on the WRH science boxes but has not yet been applied.

Regionmask is used to subset the data spatially, and is fairly easy to use. There is a standalone example (see below) of how to read shapefiles with geopandas, which then can be transformed using regionmask into a True/False boolean grid. This grid is a mask that can be used to clip the data array before concatenating along valid time, forecast hour, or threshold dimensions to once again reduce the overall memory load. Ideally, it would be more appropriate to apply the mask on the fly rather than clip the extent of the data array so that analysis could be performed on multiple CWA's more easily. This would require changing the order in which things are done and would require appending to the statistics daily, rather than over the dataset as a whole.

In moving towards an automated, constantly updating analysis, some preliminary work was done to assess the possibility of calculating reliability/ROC/brier skill over a gridded area (specifically a CWA or forecast zone) for a single day's worth of data (4 runs out to 168h), then combining those over an unspecified number of days to achieve flexible statistics based on a user's query. This would mean that the memory overhead and processing time needed are greatly reduced, as these stats could be output and appended to either a CSV file each day by CWA or zone. From the data in CSV format, organized by CWA, lead time, and valid date, an aggregate of the data could be used to produce the reliability diagrams, ROC curves, or brier skill scores for a user-selected date range, CWA, or lead time.

- **Presentation:** Once the analysis is complete, the current set of scripts hold the data in memory and then derive numerous plots which are all custom coded. Examples of each are found below, and the notebook examples can be helpful in breaking down their development. The spatial plots are slightly more complex, as these are a non-conventional way of viewing reliability metrics, especially. For the reliability diagrams and ROC curves, canned functions exist via scikit-learn that can be helpful for making quick plots on the fly, especially if the forecast and verification data are both in similar xarray datasets. As previously mentioned, if exporting the statistics to a csv, the plots could be made on the fly/on the web and much more interactive than the current static set.

- **Automation:** Currently, data acquisition for the NBM QMD PQPF threshold probabilities and URMA 6-hourly precipitation are automated. The NBM PQPF archive is updated once daily, polling for the prior 4 runs (0, 6, 12, 18) out to FHR168. The data are packed into multidimensional netCDF files, segmented by month and forecast hour to keep file size manageable. The URMA data are being archived independent of this project. There is potential to automate the analysis and output/plotting with the code here as a foundation, though as it was written for a much more capable system, it needs to be streamlined and creatively restructured. Some of the scrap notebooks and scripts contain attempts to resolve this problem.

URMA archive validation issues, WPC vs. STID:

Initial results presented in Science Round Tables were found to be in conflict with WPC verification. In places, certain thresholds presented an opposing (wet vs dry) bias and overall poorer ROC and Brier scores. After assessing the 5km verification WPC runs, as well as their own 2.5 km NBM/URMA verification, that did not appear to be the issue. After exchanging some QPE stats from the STID URMA archive vs the WPC URMA archive, it was found that the root of the issue is that URMA can be amended with new data up to +7 days from the valid time. Substantial deficits in total mass (accumulated precipitation) were found in the STID archive vs the WPC archive, and the early archival of the URMA precipitation grids was the root cause. Archived URMA precipitation grids were obtained from WPC and used to backfill the local archive through the start of NBM 4.0 (10/1/2021).

Codebase: https://github.com/m-wessler/nbm-pqpf-deploy.git
git@github.com:m-wessler/nbm-pqpf-deploy.git

Ready-to-run, automated:

nbm_archive_automation.py	This script is currently running on science3.wrh.noaa.gov and actively archiving the NBM PQPF threshold grids over WR (and portions of CR in the Intermountain West). This script is called from cron once daily (or can be called directly by the user), checks the archive for existing data, and will backfill to the most recent archived model run (not gaps beyond that, but as long as the existing archive is not corrupted, will perform as expected). The archives are output as netcdf files, one for each forecast hour, for each month of the year. This keeps file size manageable and read/write speeds fast. It also ensures the entire archive does not have to be rebuilt from scratch if one particular month/lead time is corrupted. At this time only the threshold probabilities (up to 4.0”) are being exported, but there exists a placeholder in the code to also export percentiles using the same methodology as thresholds if desired. Ideally do not export all, but instead select representative levels (e.g. 1, 5, 10, 25, 50, 75, 90, 95, 99) to mitigate file size and memory limitaitons.
get_nbm_gribs_aws.py (called from: nbm_archive_automation.py)	Anonymously accesses AWS NBM repository using **Boto** client, a directory search for recent runs, and then subsetting using **wgrib2** after file is downloaded. See example scripts for newer code to subset NBM gribs inline on download using curl and byte ranges. Current subset: All QMD PQPF (Percentiles + Thresholds) nlon, xlon, nlat, xlat = -130, -100, 30, 50 Init_hours = 0, 6, 12, 18

Previously working code, not yet ported:
The following scripts were used to produce the research-style static plots in the science round tables. They were written to run over the dataset as a whole (or a predefined time range) and

subset_urma.py	Subsets the full CONUS URMA grids that are archived on /nas to a more manageable WR (and western CR) grid using a python command line call to wgrib2. Output files are tagged with **.WR.grib2*.

	This can be run over a large file list as it feeds the wgrib2 call to process pools (with a limit of 32 simultaneous jobs). Currently this is run independent of the aggregation script, but could and likely should be called from **aggregate_urma.py**.
aggregate_urma.py	Reads from the output directory for the subset URMA grids and (1) compiles the 6h grids into 24h precipitation, (2) compiles the 24h precipitation into a single netCDF file. Care was taken to ensure that the **.resample** function used was configured properly so that the 24h precipitation ending at the valid time was correctly accumulated. This needs to have an ‘append’ function with backfill added in similar fashion to **nbm_archive_automation.py**. Likely that script could serve as a template for open, check, backfill, append. Calling **subset_urma.py** during this process will help streamline further. This could then be called from the cron along with the NBM PQPF archiver.
nbm_skill_verif.py	This is the script which reads the aggregated NBM and URMA data and produces the analysis, stats, and plots. It was designed to be run after the aggregate datasets are created, and can take some time to run, especially for longer datasets. This script will read the configuration file (date range, forecast hours/lead times, interval, thresholds, reliability bins, etc.), though care should be taken to keep the date range within the available data. The date range also needs to be sufficiently long to produce reliable stats. Requires **verif_funcs.py**, **verif_config.py**.
verif_funcs.py	Functions used in **nbm_skill_verif.py**. These are modular and all function on XArray DataArrays, so should be quite portable and usable in future iterations of the program.
verif_config.py	Configuration file for **nbm_skill_verif.py**. Sets directories for NBM, URMA, temp, and figures, as well as date ranges, thresholds, lead times, and more.

Notebooks, development code:

./notebooks/ multi_station_1Dqpf.ipynb	Originally written while 3.2 was operational, this polls data from the 1D viewer for a list of stations within a zone or CWA and verifies against observations from the Synoptic

	Data API. There are many useful code blocks in here, from the data ingest, to the mapping of metadata across a CWA, to bulk stats and traditional contingency tables (POD/FAR). This was the initial effort in verifying NBM PQPF before moving to URMA/gridded analysis and could likely be combined with the later grid-based work to bolster the results/compare point verification against URMA.
./notebooks/ nbm_reliability_plots.ipynb	First standalone iteration of custom reliability diagrams and ROC curves. These are manually calculated based on textbook equations and were verified correct against ‘canned’ functions to produce the same (calibration curves in scikit-learn). This may be helpful to see the code somewhat deconstructed into blocks before it was streamlined into the .py runtime.
./notebooks/ nbm_reliability_maps.ipynb	Standalone notebook with development of the spatial reliability plots, using the SEW CWA as an example. Breaks down the logic of developing the bins, calculating the reliability metrics, and then plotting a certain bin on the map with the “too wet”/”too dry” shading. This provided the groundwork for the final .py runtime and may be helpful in reproducing these in future work.
./notebooks/ reliability_new.ipynb	Standalone script that was used to compare the custom coded reliability diagram and ROC curve code to a set of canned functions from scikit-learn. Great example application of the canned functions *sklearn.calibration_curve* and *sklearn.roc_curve, sklearn.roc_auc_score* to an XArray dataset.
./notebooks/ netcdf4_writer.ipynb	A limited example, but gives a full breakdown of writing a custom NetCDF4 file from XArray without the use of XArray backends (which at this time do not create proper CDF compliant output). Provides a template for writing dimensions, coordinates, metadata, and data to NetCDF4.
./notebooks/ regionask_example.ipynb	A clean example of the use of *regionmask* and *cartopy* to subset and plot gridded datasets within a selected shapefile, with annotations. In this example NWS forecast zones are dissolved into their CWAs, which are then used to subset a single CWA’s geo-referenced data on a map.

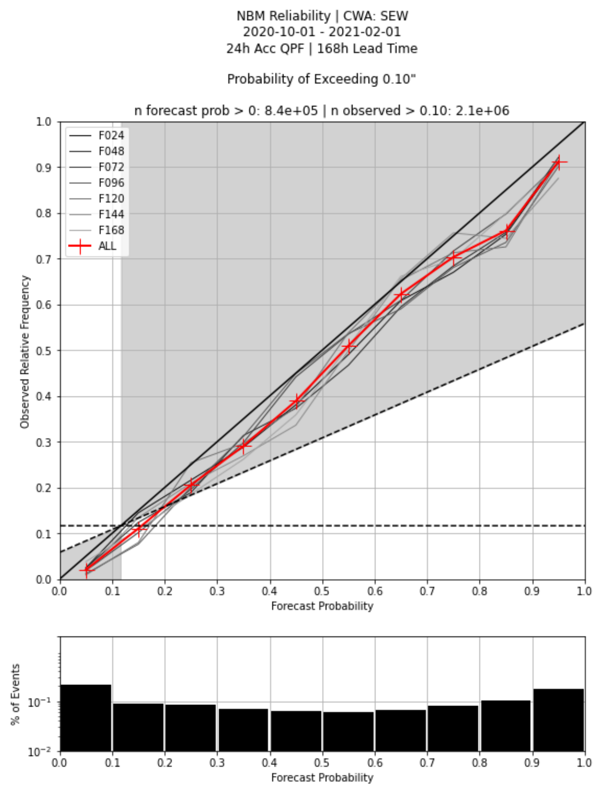
Code samples, extras:

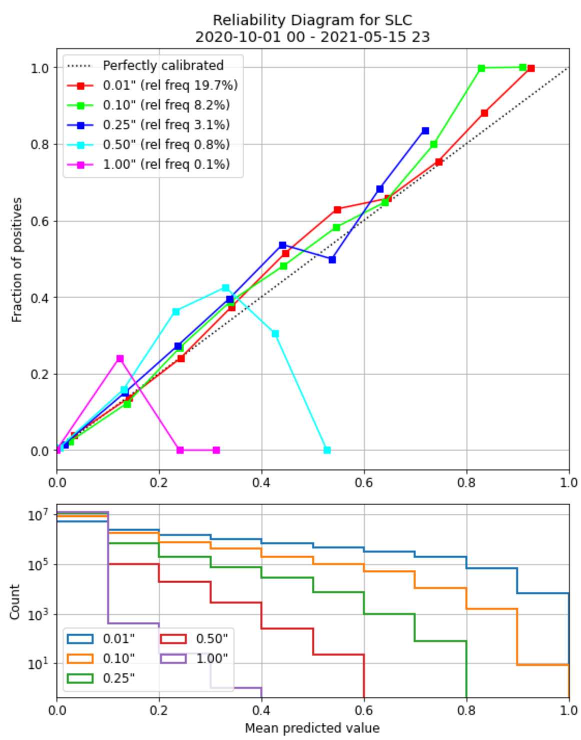
./scraps/scripts_latest/ ./scraps/scripts_all/	There is a considerable amount of old code from earlier iterations not covered elsewhere buried in these two folders. **./scripts_latest** has the most recent work, with **./scripts_all** containing prior iterations and lots of scratch

	work. There may be some useful bits here, which have been expanded upon below - the rest are unlikely to be worth much.
./scraps/scripts_latest/ extract_pdf_det.py	Earlier development code that was written for extracting percentile data from the NBM and archiving an aggregate in a NetCDFs. This has all the logic necessary to be applied to the **nbm_archive_automation.py** script to also archive percentiles alongside the threshold probabilities.
./scraps/scripts_latest/ nbm_skill_verif_expanded.py	An earlier iteration of **nbm_skill_verif.py** before functions were broken out into **verif_funcs.py** and the script was streamlined. There may be some utility here, but for the most part the content is covered in the updated script. There was an attempt to plot a gridded Brier Skill Score here, an example that may serve useful in future iterations.
./scraps/scripts_latest/ aws_NBM_get_subset.py	Sample script for downloading data from AWS S3 buckets using *boto* and *curl* (via command line call), sped up with *mutiprocessing.dummy* threadpools. Highly useful for developing new code going forward as this is much faster and less memory intensive than downloading the entire grid and subsetting locally.
./scraps/notebooks_latest/ ./scraps/notebooks_all/	Similar to the scrap scripts, these are old notebooks that have been used to develop the final scripts. There is likely some useful code here, especially within **./notebooks_latest**.
./forecast-zones-new/	(Current) Shapefile of NWS Forecast Zones, using new zone numbers, and trimmed coastline to exclude offshore points/islands. Includes select mountain west CR CWAs.
./forecast-zones-final/	(Old) Shapefile of NWS Forecast Zones, using new zone numbers, WR only, includes some bleed into offshore grid points and islands.

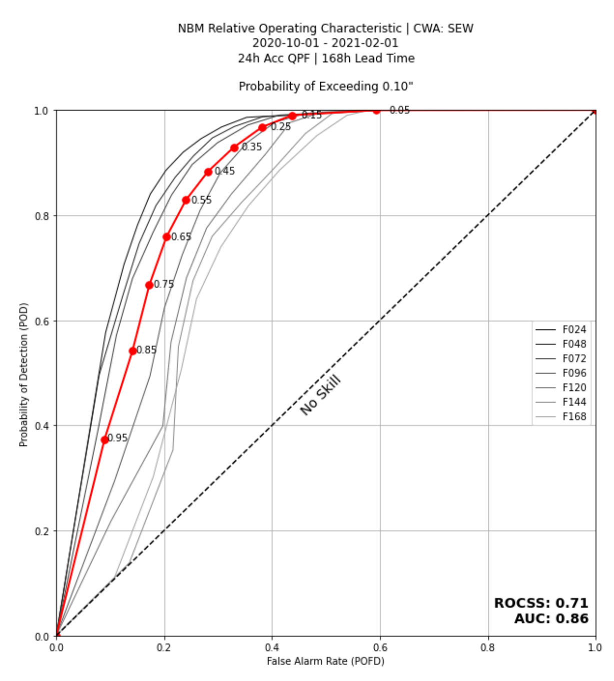
Sample output:

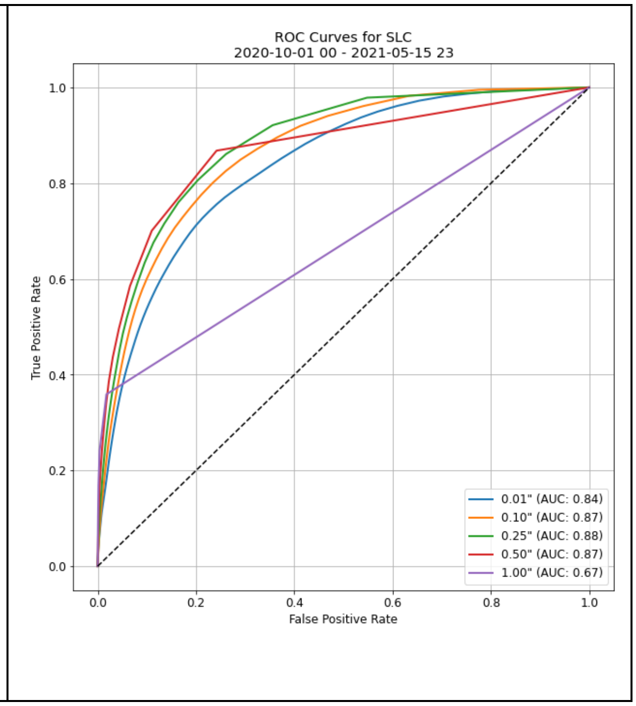
Reliability (2 Styles)



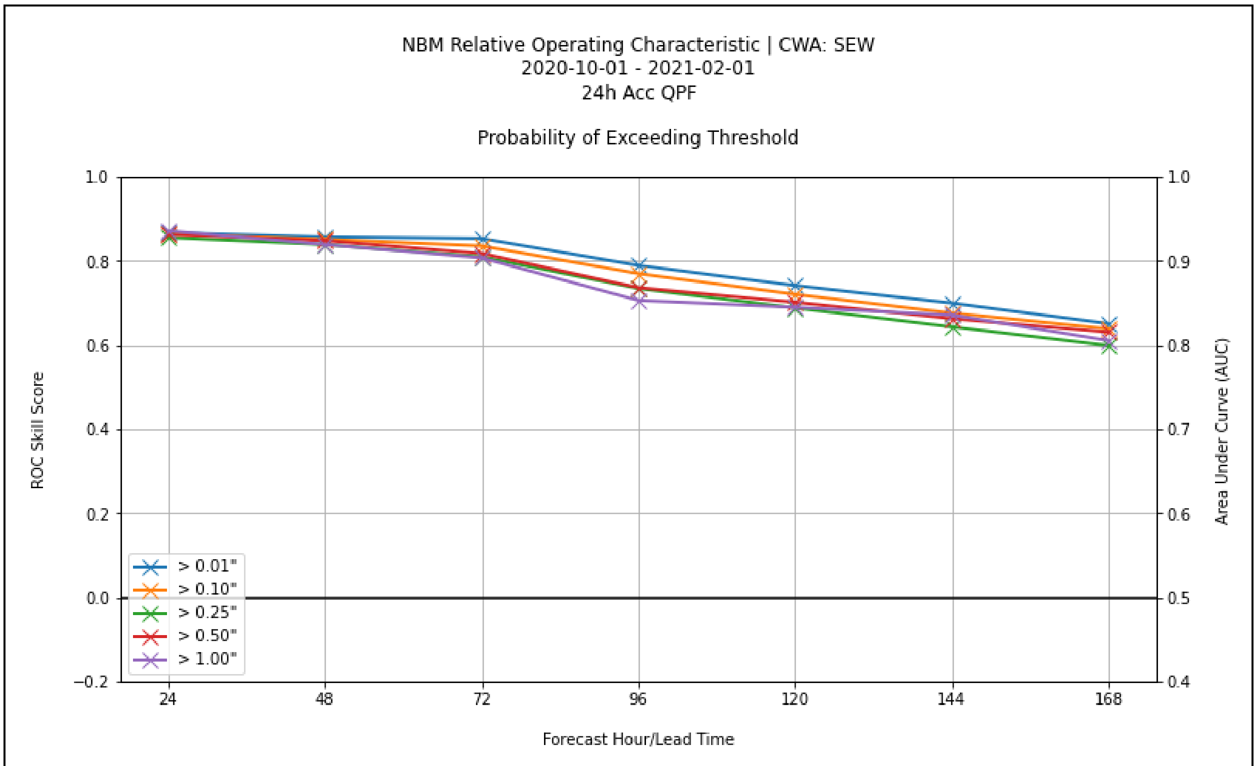


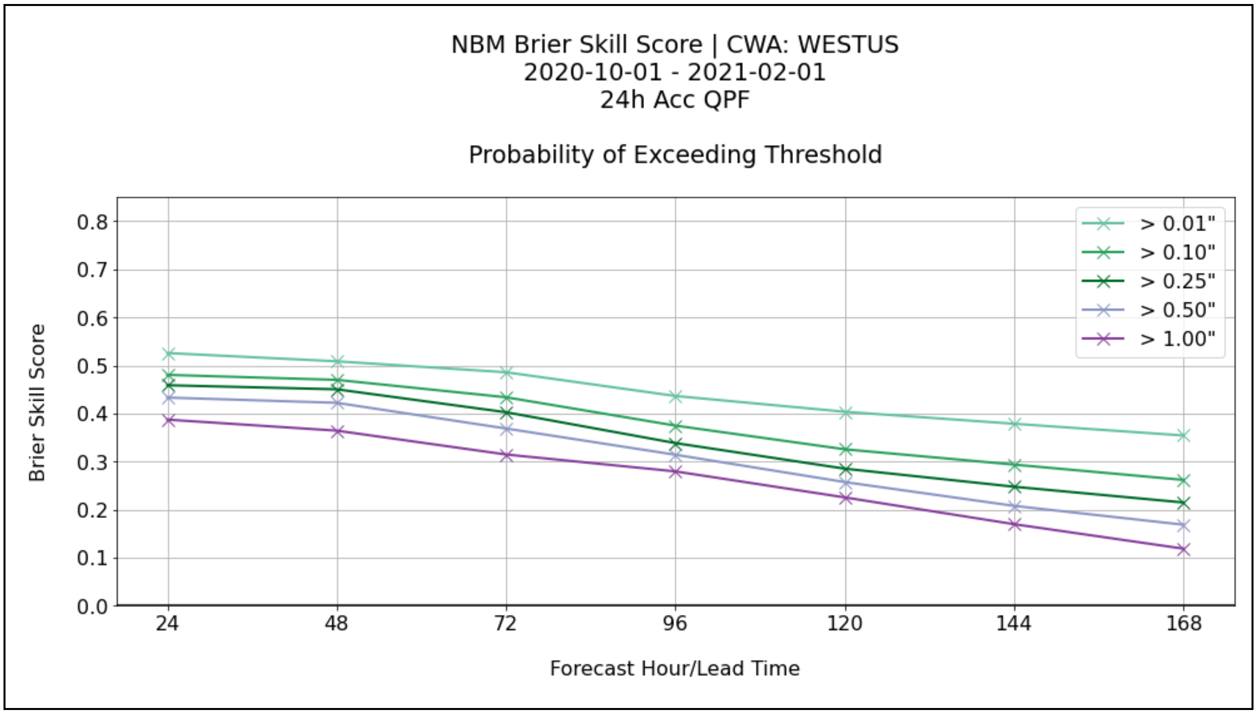
ROC Curves (2 Styles)



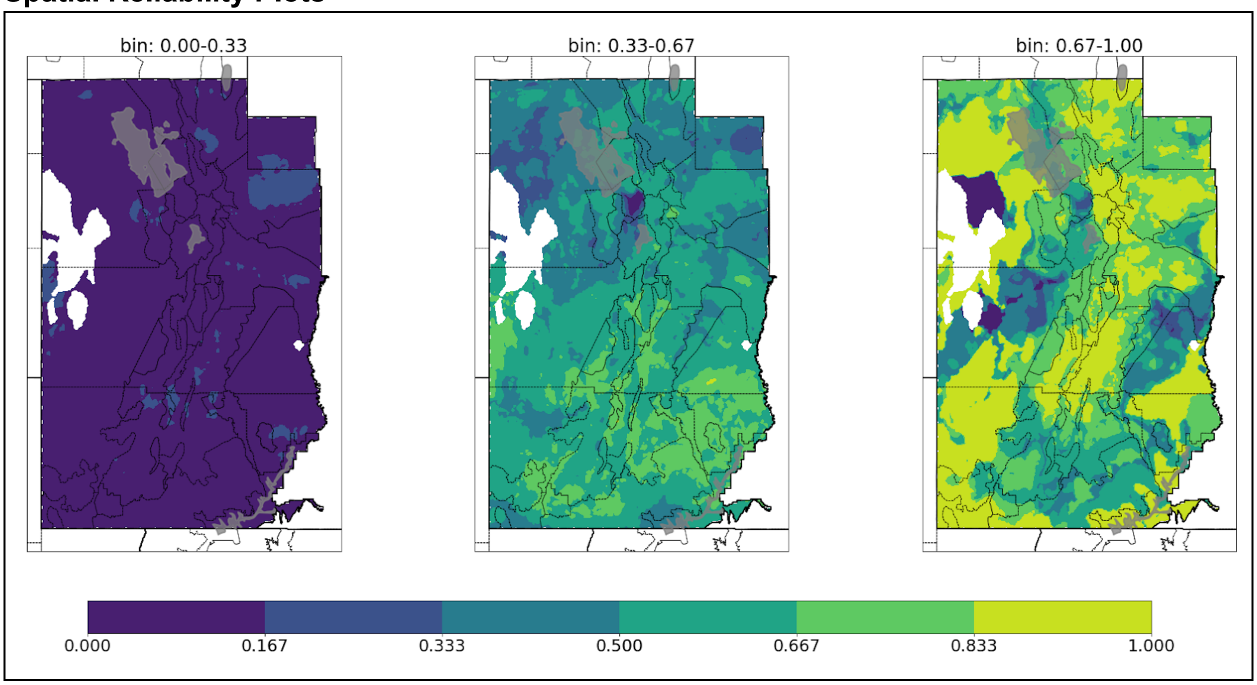


ROC/BSS vs Lead Time





Spatial Reliability Plots



Anaconda environment/package list:

```
# packages in environment at ~/anaconda3/envs/xlab:
#
# Name Version Build
Channel
_libgcc_mutex 0.1 main
_tflow_select 2.3.0 eigen
absl-py 0.11.0 py38h578d9bd_0
conda-forge
affine 2.3.0 py_0
conda-forge
aiohttp 3.7.3 py38h25fe258_0
conda-forge
appdirs 1.4.3 py_1
conda-forge
argon2-cffi 20.1.0 py38h7b6447c_1
arm_pyart 1.11.2 py38hab2c0dc_0
conda-forge
astor 0.8.1 pyh9f0ad1d_0
conda-forge
astunparse 1.6.3 pyhd8ed1ab_0
conda-forge
async-timeout 3.0.1 py_1000
conda-forge
async_generator 1.10 py_0
attrs 20.2.0 py_0
backcall 0.2.0 py_0
blas 1.0 mkl
bleach 3.2.1 py_0
blinker 1.4 py_1
conda-forge
bokeh 2.2.1 py38_0
boost-cpp 1.72.0 h8e57a91_0
conda-forge
boto3 1.15.13 py_0
botocore 1.18.13 py_0
brotlipy 0.7.0 py38h7b6447c_1000
bzip2 1.0.8 h7b6447c_0
c-ares 1.16.1 h516909a_3
conda-forge
ca-certificates 2020.12.5 ha878542_0
conda-forge
cachetools 4.2.1 pyhd8ed1ab_0
conda-forge
cairo 1.16.0 hcf35c78_1003
conda-forge
cartopy 0.18.0 py38h172510d_0
conda-forge
certifi 2020.12.5 py38h578d9bd_1
conda-forge
cffi 1.14.3 py38h5bae8af_0
conda-forge
cfgrib 0.9.8.4 py_0
conda-forge
cfitsio 3.470 hce51eda_6
conda-forge
cftime 1.2.1 py38h8790de6_0
conda-forge
chardet 3.0.4 py38_1003
click 7.1.2 py_0
click-plugins 1.1.1 py_0
conda-forge
cligj 0.5.0 py_0
conda-forge
cloudpickle 1.6.0 py_0
cryptography 3.1.1 py38h1ba5d50_0
curl 7.71.1 he644dc0_8
conda-forge
cycler 0.10.0 py38_0
cytoolz 0.11.0 py38h7b6447c_0
dask 2.30.0 py_0
dask-core 2.30.0 py_0
dbus 1.13.16 hb2f20db_0
decorator 4.4.2 py_0
defusedxml 0.6.0 py_0
descartes 1.1.0 py_4
conda-forge
distributed 2.30.0 py38_0
docutils 0.15.2 py38_0
eccodes 2.17.0 h59f7be3_1
conda-forge
entrypoints 0.3 py38_0
esmf 8.0.0 nompi_hb0fcdcb_6
conda-forge
esmpy 8.0.0
nompi_py38hf0e99fa_1 conda-forge
expat 2.2.9 he6710b0_2
fiona 1.8.13.post1 py38hc820daa_0
fontconfig 2.13.1 h86ecdb6_1001
conda-forge
freetype 2.10.2 h5ab3b9f_0
freexl 1.0.5 h516909a_1002
conda-forge
fsspec 0.8.0 py_0
g2clib 1.6.0 hf3f1b0b_9
conda-forge
gast 0.3.3 py_0
conda-forge
gdal 3.0.4 py38h172510d_6
conda-forge
geopandas 0.8.1 py_0
geos 3.8.1 he1b5a44_0
conda-forge
geotiff 1.5.1 h05acad5_10
conda-forge
gettext 0.19.8.1 hc5be6a0_1002
conda-forge
```

giflib 5.2.1 h516909a_2
conda-forge
glib 2.66.1 h680cd38_0
conda-forge
google-auth 1.24.0 pyhd3deb0d_0
conda-forge
google-auth-oauthlib 0.4.1 py_2
conda-forge
google-pasta 0.2.0 pyh8c360ce_0
conda-forge
grpcio 1.33.2 py38heead2fc_2
conda-forge
gst-plugins-base 1.14.5 h0935bb2_2
conda-forge
gstreamer 1.14.5 h36ae1b5_2
conda-forge
h5netcdf 0.8.1 py_0
conda-forge
h5py 2.10.0
nompi_py38h513d04c_102 conda-forge
hdf4 4.2.13 h3ca952b_2
hdf5 1.10.5 nompi_h3c11f04_1104
conda-forge
hdfeos2 2.20 h64bfcee_1000
conda-forge
hdfeos5 5.1.16 h8b6279f_6
conda-forge
heapdict 1.0.1 py_0
icu 64.2 he1b5a44_1
conda-forge
idna 2.10 py_0
importlib-metadata 1.7.0 py38_0
importlib_metadata 1.7.0 0
importlib_resources 3.0.0 py38h32f6830_0
conda-forge
intel-openmp 2020.2 254
ipykernel 5.3.4 py38h5ca1d4c_0
ipython 7.18.1 py38h5ca1d4c_0
ipython_genutils 0.2.0 py38_0
ipywidgets 7.5.1 py_1
jasper 1.900.1 hd497a04_4
jedi 0.17.2 py38_0
jinja2 2.11.2 py_0
jmespath 0.10.0 py_0
joblib 0.17.0 py_0
jpeg 9d h516909a_0
conda-forge
json-c 0.13.1 hbfbb72e_1002
conda-forge
json5 0.9.5 py_0
jsonschema 3.2.0 py38_1
jupyter 1.0.0 py38_7
jupyter_client 6.1.7 py_0
jupyter_console 6.2.0 py_0
jupyter_core 4.6.3 py38_0
jupyterlab 2.2.6 py_0
jupyterlab_pygments 0.1.2 py_0
jupyterlab_server 1.2.0 py_0
kealib 1.4.13 hec59c27_0
conda-forge
keras 2.4.3 py_0
conda-forge
keras-preprocessing 1.1.2 pyhd8ed1ab_0
conda-forge
kiwisolver 1.2.0 py38hfd86e86_0
krb5 1.17.1 hfafb76e_3
conda-forge
lcms2 2.11 h396b838_0
ld_impl_linux-64 2.33.1 h53a641e_7
libaec 1.0.4 he6710b0_1
libcurl 7.71.1 hcdd3856_8
conda-forge
libdap4 3.20.6 h1d1bd15_1
conda-forge
libedit 3.1.20191231 h14c3975_1
libev 4.33 h516909a_1
conda-forge
libffi 3.2.1 he1b5a44_1007
conda-forge
libgcc-ng 9.1.0 hdf63c60_0
libgdal 3.0.4 h3dfc09a_6
conda-forge
libgfortran-ng 7.3.0 hdf63c60_0
libiconv 1.16 h516909a_0
conda-forge
libkml 1.3.0 hd79254b_1012
conda-forge
libnetcdf 4.7.4 nompi_h9f9fd6a_101
conda-forge
libnghttp2 1.41.0 h8cfc5f6_2
conda-forge
libpng 1.6.37 hbc83047_0
libpq 12.3 h5513abc_0
conda-forge
libprotobuf 3.13.0.1 h8b12597_0
conda-forge
libsodium 1.0.18 h7b6447c_0
libspatialindex 1.9.3 he6710b0_0
libspatialite 4.3.0a h2482549_1038
conda-forge
libssh2 1.9.0 h1ba5d50_1
libstdcxx-ng 9.1.0 hdf63c60_0
libtiff 4.1.0 h2733197_1
libuuid 2.32.1 h14c3975_1000
conda-forge
libwebp-base 1.1.0 h516909a_3
conda-forge
libxcb 1.14 h7b6447c_0

libxml2 2.9.10 hee79883_0
conda-forge
locket 0.2.0 py38_1
lz4-c 1.9.2 he6710b0_1
markdown 3.3.3 pyh9f0ad1d_0
conda-forge
markupsafe 1.1.1 py38h7b6447c_0
matplotlib 3.3.1 0
matplotlib-base 3.3.1 py38h817c723_0
metpy 0.12.2 py_0
conda-forge
mistune 0.8.4 py38h7b6447c_1000
mkl 2020.2 256
mkl-service 2.3.0 py38he904b0f_0
mkl_fft 1.2.0 py38h23d657b_0
mkl_random 1.1.1 py38h0573a6f_0
msgpack-python 1.0.0 py38hfd86e86_1
multidict 4.7.5 py38h1e0a361_2
conda-forge
munch 2.5.0 py_0
nbclient 0.5.0 py_0
nbconvert 6.0.7 py38_0
nbformat 5.0.7 py_0
ncurses 6.2 he6710b0_1
nest-asyncio 1.4.1 py_0
netcdf-fortran 4.5.2
nompi_h45d7149_104 conda-forge
netcdf4 1.5.3
nompi_py38heb6102f_103 conda-forge
notebook 6.1.4 py38_0
numpy 1.19.1 py38hbc911f0_0
numpy-base 1.19.1 py38hfa32c7d_0
oauthlib 3.0.1 py_0
conda-forge
olefile 0.46 py_0
openjpeg 2.3.1 h981e76c_3
conda-forge
openssl 1.1.1h h516909a_0
conda-forge
opt_einsum 3.3.0 py_0
conda-forge
owslib 0.20.0 py_0
conda-forge
packaging 20.4 py_0
pandas 1.1.3 py38he6710b0_0
pandoc 2.10.1 0
pandocfilters 1.4.2 py38_1
parso 0.7.0 py_0
partd 1.1.0 py_0
pcre 8.44 he6710b0_0
pexpect 4.8.0 py38_0
pickleshare 0.7.5 py38_1000
pillow 7.2.0 py38hb39fc2d_0
pint 0.16.1 py_0
conda-forge
pip 20.2.3 py38_0
pixman 0.38.0 h516909a_1003
conda-forge
pooch 1.2.0 py_0
conda-forge
poppler 0.67.0 h14e79db_8
conda-forge
poppler-data 0.4.9 1
conda-forge
postgresql 12.3 h8573dbc_0
conda-forge
proj 7.0.0 h966b41f_5
conda-forge
prometheus_client 0.8.0 py_0
prompt-toolkit 3.0.7 py_0
prompt_toolkit 3.0.7 0
protobuf 3.13.0.1 py38hadf7658_1
conda-forge
psutil 5.7.2 py38h7b6447c_0
ptyprocess 0.6.0 py38_0
pyasn1 0.4.8 py_0
conda-forge
pyasn1-modules 0.2.7 py_0
conda-forge
pycparser 2.20 py_2
pyepsg 0.4.0 py_0
conda-forge
pygments 2.7.1 py_0
pygrib 2.0.5 py38hfcef17a_0
conda-forge
pyjwt 2.0.1 pyhd8ed1ab_0
conda-forge
pynio 1.5.5 py38h031d99c_12
conda-forge
pyopenssl 19.1.0 py_1
pyparsing 2.4.7 py_0
pyproj 2.6.1.post1 py38h7521cb9_0
conda-forge
pyqt 5.9.2 py38h05f1152_4
pyrsistent 0.17.3 py38h7b6447c_0
pyshp 2.1.2 pyh9f0ad1d_0
conda-forge
pysocks 1.7.1 py38_0
python 3.8.3 cpython_he5300dc_0
conda-forge
python-dateutil 2.8.1 py_0
python_abi 3.8 1_cp38
conda-forge
pytz 2020.1 py_0
pyyaml 5.3.1 py38h7b6447c_1
pyzmq 19.0.2 py38he6710b0_1

qt 5.9.7 h0c104cb_3
conda-forge
qtconsole 4.7.7 py_0
qtpy 1.9.0 py_0
rasterio 1.1.5 py38h033e0f6_1
conda-forge
readline 8.0 h7b6447c_0
regionmask 0.6.1 py_1
conda-forge
requests 2.24.0 py_0
requests-oauthlib 1.3.0 pyh9f0ad1d_0
conda-forge
rsa 4.7 pyhd3deb0d_0
conda-forge
rtree 0.9.4 py38_1
s3fs 0.3.0 py_0
conda-forge
s3transfer 0.3.3 py38_0
scikit-learn 0.23.2 py38h0573a6f_0
scipy 1.5.2 py38h0b6359f_0
seaborn 0.11.0 py_0
send2trash 1.5.0 py38_0
setuptools 50.3.0 py38hb0f4dca_1
shapely 1.7.1 py38hc7361b7_0
conda-forge
sip 4.19.13 py38he6710b0_0
six 1.15.0 py_0
snuggs 1.4.7 py_0
conda-forge
sortedcontainers 2.2.2 py_0
sqlite 3.33.0 h62c20be_0
tbb 2020.2 hc9558a2_0
conda-forge
tblib 1.7.0 py_0
tensorboard 2.4.1 pyhd8ed1ab_0
conda-forge
tensorboard-plugin-wit 1.8.0 pyh44b312d_0
conda-forge
tensorflow 2.3.0
eigen_py38h71ff20e_0
tensorflow-base 2.3.0
eigen_py38hb57a387_0
tensorflow-estimator 2.4.0 pyh9656e83_0
conda-forge
termcolor 1.1.0 py_2
conda-forge
terminado 0.8.3 py38_0
testpath 0.4.4 py_0
threadpoolctl 2.1.0 pyh5ca1d4c_0
tiledb 1.7.7 h8efa9f0_3
conda-forge
tk 8.6.10 hbc83047_0
toolz 0.11.1 py_0
tornado 6.0.4 py38h7b6447c_1
traitlets 5.0.4 py38_0
trmm_rsl 1.49 3
conda-forge
typing-extensions 3.7.4.3 0
conda-forge
typing_extensions 3.7.4.3 py_0
tzcode 2020a h516909a_0
conda-forge
urllib3 1.25.10 py_0
wcwidth 0.2.5 py_0
webencodings 0.5.1 py38_1
werkzeug 1.0.1 pyh9f0ad1d_0
conda-forge
wheel 0.35.1 py_0
widgetsnbextension 3.5.1 py38_0
wrapt 1.12.1 py38h1e0a361_1
conda-forge
wrf-python 1.3.2 py38h7eb8c7e_1
conda-forge
xarray 0.16.1 py_0
xerces-c 3.2.2 h8412b87_1004
conda-forge
xesmf 0.4.0 pyhd8ed1ab_0
conda-forge
xlrd 1.2.0 py_0
xorg-kbproto 1.0.7 h14c3975_1002
conda-forge
xorg-libice 1.0.10 h516909a_0
conda-forge
xorg-libsm 1.2.3 h84519dc_1000
conda-forge
xorg-libx11 1.6.12 h516909a_0
conda-forge
xorg-libxext 1.3.4 h516909a_0
conda-forge
xorg-libxrender 0.9.10 h516909a_1002
conda-forge
xorg-renderproto 0.11.1 h14c3975_1002
conda-forge
xorg-xextproto 7.3.0 h14c3975_1002
conda-forge
xorg-xproto 7.0.31 h14c3975_1007
conda-forge
xz 5.2.5 h7b6447c_0
yaml 0.2.5 h7b6447c_0
yarl 1.6.3 py38h25fe258_0
conda-forge
zeromq 4.3.2 he6710b0_3
zict 2.0.0 py_0
zipp 3.3.0 py_0
zlib 1.2.11 h7b6447c_3
zstd 1.4.5 h9ceee32_0