Supporting Information for Improving Wind Forecasts in the Lower Stratosphere by Distilling an Analog Ensemble into a Deep Neural Network

Salvatore Candido[1], Aakanksha Singh[1], and Luca Delle Monache[1,2]

[1]Loon, Mountain View, California, USA

[2]Center for Western Weather and Water Extremes, La Jolla, California, USA

Contents of this file

Introduction

The supplemental data here covers a few disparate sets of material. The first two sections provide additional technical detail on the methods of the paper that are not necessary to understand the approach, but are useful when replicating the results. We then share statistical breakdowns of the Loon observations which can also be derived by

processing the data set at (Candido, 2020), but we include for convenience here. The following three sections contain views of our results that we do not include in the main text, but are of interest to some readers of early drafts of this paper. Finally, we present an additional validation set that leads to similar conclusions as the results in the main text. We include this for completeness as some of our early discussions of this work used this validation set, rather than the newer validation set that allows us to compare against the European Centre for Medium-Range Weather Forecasts (ECMWF) ensemble system (ENS).

Computing the Conventional Analog Ensemble with MapReduce

A barrier to operationalizing a global analog ensemble (AnEn) system is processing the corpus of analogs, which can easily grow to 100's of terabytes of data for three-dimensional global predictions over several years. The AnEn algorithm provides a natural partitioning as execution is independent for each $\mathcal{P}$ (grid point and lead time). Every historical forecast and the current prediction contain a piece of data for every grid point. The challenge is to organize the data so that the calculations can be efficiently executed across many datacenter computers.

Our approach is to use the MapReduce paradigm (Dean & Ghemawat, 2004), which allows the computation to run on a distributed computing (cloud) infrastructure like Google's Flume (Chambers et al., 2010). The idea is to break the computation into two subsequent `Map` and `Reduce` phases, each of which operate many times in parallel on different portions of the data and output key-value pairs. Once written this way, the framework can handle scheduling the program's execution across many machines and moving the various subsets of data to the appropriate machines.

The above procedure is accomplished as follows. Let latitude, longitude, pressure, and forecast lead time be the tuple k, which is unique for each grid point. The first `Map` phase scans all historical forecast files and generates a key-value pair $(k, t_f) \rightarrow x^i$ for each grid point and $(k, t_f) \rightarrow y^i$ for each analysis point. The k corresponds to the location and forecast lead time for the particular x^i (past forecast) and t_f (calendar time being forecast). For every y^i we generate multiple key-value pairs corresponding to a k for *every* lead time the system will forecast.

Notice that the above rule gives each forecast-observation pair a unique key. Prior to the reduce phase all identical keys are grouped into one `Reduce` call by the MapReduce framework. The `Reduce` phase joins these x^i and y^i pairs into a single record and saves them as new key-value pairs $k \rightarrow (x^i, y^i)$. This first MapReduce gives us a historical corpus. This corpus could be built in advance of receiving a new forecast to post-process.

The second MapReduce groups the data by grid point and runs the AnEn algorithm. A `Map` phase on the forecast data file from the ECMWF generates key-value pairs $k \rightarrow x^f$ for each grid point. The historical corpus key-value pairs are used directly. Note that x^f and every set of candidate analogs $\{(x^i, y^i)\}$ for a grid point have the same key. The data is grouped by key and fed to the `Reduce` phase that has all the data needed to apply equations (1) and (3) to generate the forecast.

Details of Training a Distilled Analog Ensemble Model

This section describes the low-level technical details of the distillation process.

Our training corpus is prepared by using a MapReduce similar to what is described in the previous section to process the set of forecast data files (both the 00Z and 12Z epochs) archived during the training period. Rather than running the AnEn algorithm

logic, i.e., equations (1) and (3), at each grid point given a new forecast, we instead save the candidate analogs (forecast-observation pairs) for a given grid point in a single record. These records are stored together on disk for retrieval by the training system, i.e., we have a set of records where each record corresponds to a unique latitude, longitude, pressure, and lead time grid point and contains all the viable forecast-observation pairs at this location at the appropriate lead time.

This data set is used to feed the training process of our deep neural network (DNN). We use a distributed architecture. We train our DNN based on the output of the AnEn, not directly from these forecast-observation pairs. Thus, we need to sample a hypothetical forecast to generate a training example of an input-output pair for the AnEn system. We use 10 datacenter worker processes that sample uniformly among grid points in records on disk, sample a hypothetical wind speed and wind heading forecast, and construct the input to the DNN (corresponding to this grid point and forecast) and the output (of the AnEn algorithm). In the results presented in this paper, we sample heading uniformly and wind speed from a beta distribution with $\alpha = 1.2$, $\beta = 3$, and a coefficient of 100. Effectively this creates a weighted distribution of wind speeds which seems generally applicable to the pressure altitudes ranges of interest in the stratosphere.

Unlike many applications, we do not simply loop over the dataset on disk a fixed number of times or until training error stabilizes. This is because every time we touch a record we sample a new forecast and generate a new input-output pair. Saving these pairs on disk versus generating them online during training is an engineering trade-off, and we have chosen the latter approach.

These examples from the 10 worker processes are injected into a reservoir datacenter process, whose job is essentially to receive new examples, store them in a limited size buffer, and respond to requests (from the learning process) for samples. Rather than choosing a circular buffer or some other first-in, first-out structure, we use a flat data array of 1 million examples and, for each new example, sample an index in the array at random to replace. This means some examples will persist in the buffer longer, and some for a shorter period of time. The typical dwell time of an example in the buffer can be characterized probabilistically. The learning process repeatedly queries the reservoir for batches of training examples, which are selected uniformly at random from examples in the data array. A slowly changing flow of examples where each batch (on average) tends to be drawn from disparate parts of the function mapping being learned is conceptually similar to the replay buffer in deep reinforcement learning (Lin, 1992; Mnih et al., 2015).

We use the Tensorflow (Abadi et al., 2015) library to create and train our DNN. Our network has inputs of latitude, longitude, pressure altitude, forecast lead time, forecast direction, and forecast speed. We transform these into a graph layer that is normalized using the following code snippet where the array 'domain' represents the inputs described above.

```
nlat = tf.multiply(domain[:, 0], 1. / 90.0)
coslng = tf.cos(tf.multiply(domain[:, 1], np.pi / 180.0))
sinlng = tf.sin(tf.multiply(domain[:, 1], np.pi / 180.0))
npre = tf.multiply(tf.subtract(domain[:, 2], 4799.), 1. / (14432. - 4799.))
nlea = tf.multiply(tf.subtract(domain[:, 3], 43200.), 1. / (864000. - 43200.))
coshead = tf.cos(domain[:, 4])
```

```
sinhead = tf.sin(domain[:, 4])
nspeed = tf.multiply(domain[:, 5], 1. / 100.)
normalized_domain = tf.transpose(
    tf.stack([nlat, coslng, sinlng, npre, nlea, coshead, sinhead, nspeeed]))
```

We do this to avoid the discontinuity in longitude being present in our DNN, and to make our inputs have roughly the same order of magnitude (which is a domain trick to decrease training time).

At this point the network consists of 10 fully-connected hidden layers with ReLu activation functions. Each layer has a width of 50 elements. These plus an ultimate layer containing the ultimate post-processed speed and heading forecast (width 2, fully-connected, no activation function) comprise the trained layers of the DNN. We can also include additional network outputs such as forecast uncertainty (standard deviation of the forecast) or ensemble members. This is not discussed in this paper.

To train the network we use stochastic gradient descent with a learning rate of 0.0001 and batch size 100. We train until the root mean square error between the DNN forecasts and the AnEn mean forecasts (from the training examples) stabilizes. In the distilled AnEn used to generate the results in this paper, we trained the DNN with about 6 billion examples.

This network architecture was not tuned for efficiency, but instead chosen to demonstrate how a fairly standard and basic deep learning approach could be used to implement this algorithm.

Statistics of the Loon Data Set

The following plots show the distribution of Loon's approximately 10.5 million observations used for one of the comparisons between algorithms shown in the main text of the paper. This is the intersection of Loon's dataset of observations of stratospheric winds from Loon (`http://www.loon.com`) high altitude balloons (Candido, 2020) and the region and time period for the validation paper used in our study.

Figure S1 shows the distribution of the data over pressure altitude and latitude.

Figure S2 shows the geographical distribution of the data.

Probabilistic Evaluation Metrics for Wind Speed

In the main text of the paper we presented the CRPS, Spread Skill, and Rank Histogram plots for comparing the ensemble systems predictions on wind direction, an omitted plots for wind speed given a similar pattern on skill between the approaches. We include the figures for wind speed in Figure S3.

Confidence Intervals on Deterministic Evaluations

Figures S4 and S5 show the same data as in Figure 2(a) in the main text, but include box plot views of the 90% bootstrap confidence intervals.

Algorithm Skill Comparison By Geography

Figure S6 show the CRMSE averaged across all lead times grouped by geography. One can observe that the Distilled AnEn has higher skill (lower CRMSE) than the baseline ECMWF HRES generally across the stratosphere globally.

Results for an Earlier Validation Period

Our original analysis of the methods included a comparison of AnEn mean against the ECMWF high-resolution deteriministic forecast (HRES) and, for the probabilistic predictions, against a persistence ensemble (PeEn) over a year long validation period

from October, 2017 to September, 2018. However, to add a comparison to the ECMWF ENS in a revised version of the manuscript, we changed our validation period in the main text due to data availability.

The PeEn is a simple way to generate an ensemble that consists of selecting the last N available ground-truth values to generate an N-member ensemble. It has been used in, e.g., Alessandrini, Delle Monache, Sperati, and Cervone (2015) and Cervone, Clemente-Harding, Alessandrini, and Monache (2017), as a probabilistic baseline forecast and can be interpreted as the probabilistic extension of a deterministic persistence forecast.

For the results shown in in Figures S7 and S8 the training dataset is the HRES forecasts produced from July, 2016, to September, 2017. We use this to choose weights used in the analog matching process. The validation period is over the HRES forecasts produced from October, 2017, to September, 2018. The data available in the AnEn matching includes all the forecasts in the training dataset plus any additional forecasts between the beginning of the validation time period but prior to the current forecast. This simulates operational use of an AnEn system. To evaluate the distilled AnEn we only use a DNN distilled from the training dataset.

Please refer to the main text of the paper where the relevance of the metrics shown in the below figures are explained in greater detail, albeit for a different validation period.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `https://www.tensorflow.org/` (Software available from tensorflow.org)

Alessandrini, S., Delle Monache, L., Sperati, S., & Cervone, G. (2015). An analog ensemble for short-term probabilistic solar power forecast. *Applied Energy*, *157*, 95–110. Retrieved 2016-11-21, from http://www.sciencedirect.com/science/article/pii/S0306261915009368

Candido, S. (2020, April). *Loon stratospheric sensor data.* Zenodo. Retrieved from https://doi.org/10.5281/zenodo.3763022 doi: 10.5281/zenodo.3763022

Cervone, G., Clemente-Harding, L., Alessandrini, S., & Monache, L. D. (2017). Short-term photovoltaic power forecasting using artificial neural networks and an analog ensemble. *Renewable Energy*, *108*, 274 - 286. Retrieved from http://www.sciencedirect.com/science/article/pii/S0960148117301386 doi: https://doi.org/10.1016/j.renene.2017.02.052

Chambers, C., Raniwala, A., Perry, F., Adams, S., Henry, R., Bradshaw, R., & Nathan. (2010). Flumejava: Easy, efficient data-parallel pipelines. In *Acm sigplan conference on programming language design and implementation (pldi)* (p. 363-375). 2 Penn Plaza, Suite 701 New York, NY 10121-0701. Retrieved from http://dl.acm.org/citation.cfm?id=1806638

Dean, J., & Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. In *Osdi'04: Sixth symposium on operating system design and implementation* (pp. 137–150). San Francisco, CA.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, *8*(3-4), 293–321.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *Nature*,

518(7540), 529.

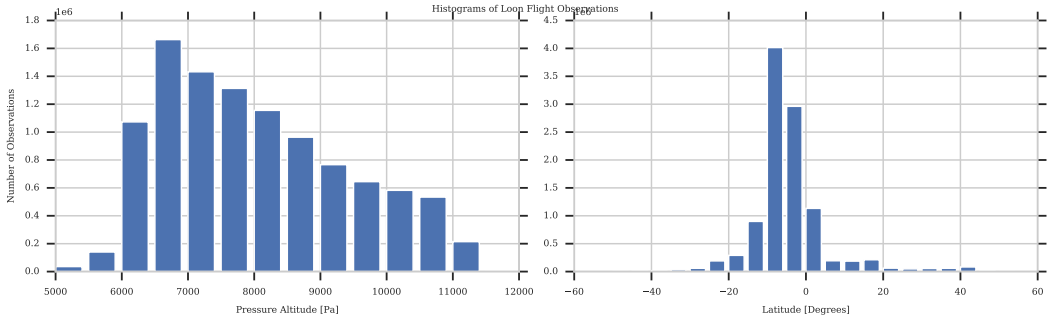


Figure S1. Distribution of Loon's measurements as a function of pressure altitude and latitude.

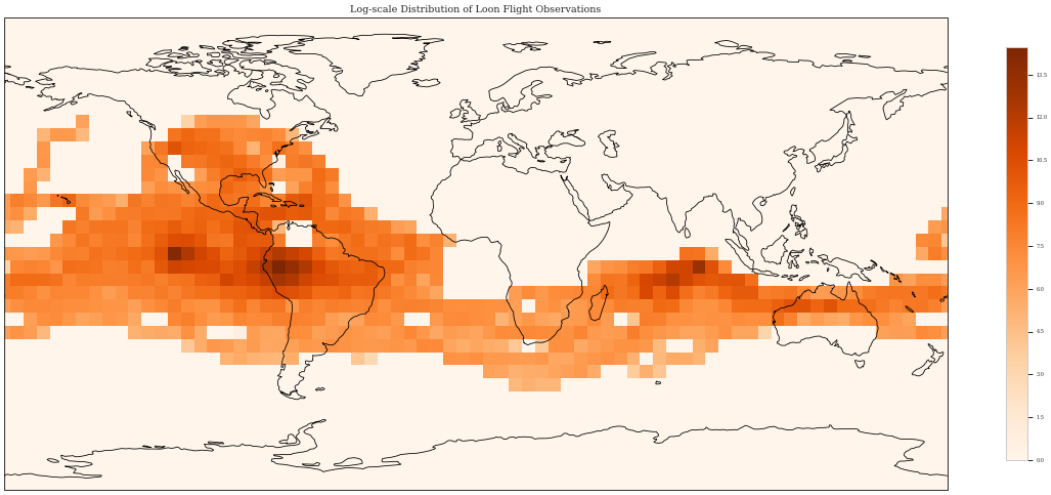


Figure S2. Geographical distribution of Loon's measurements.

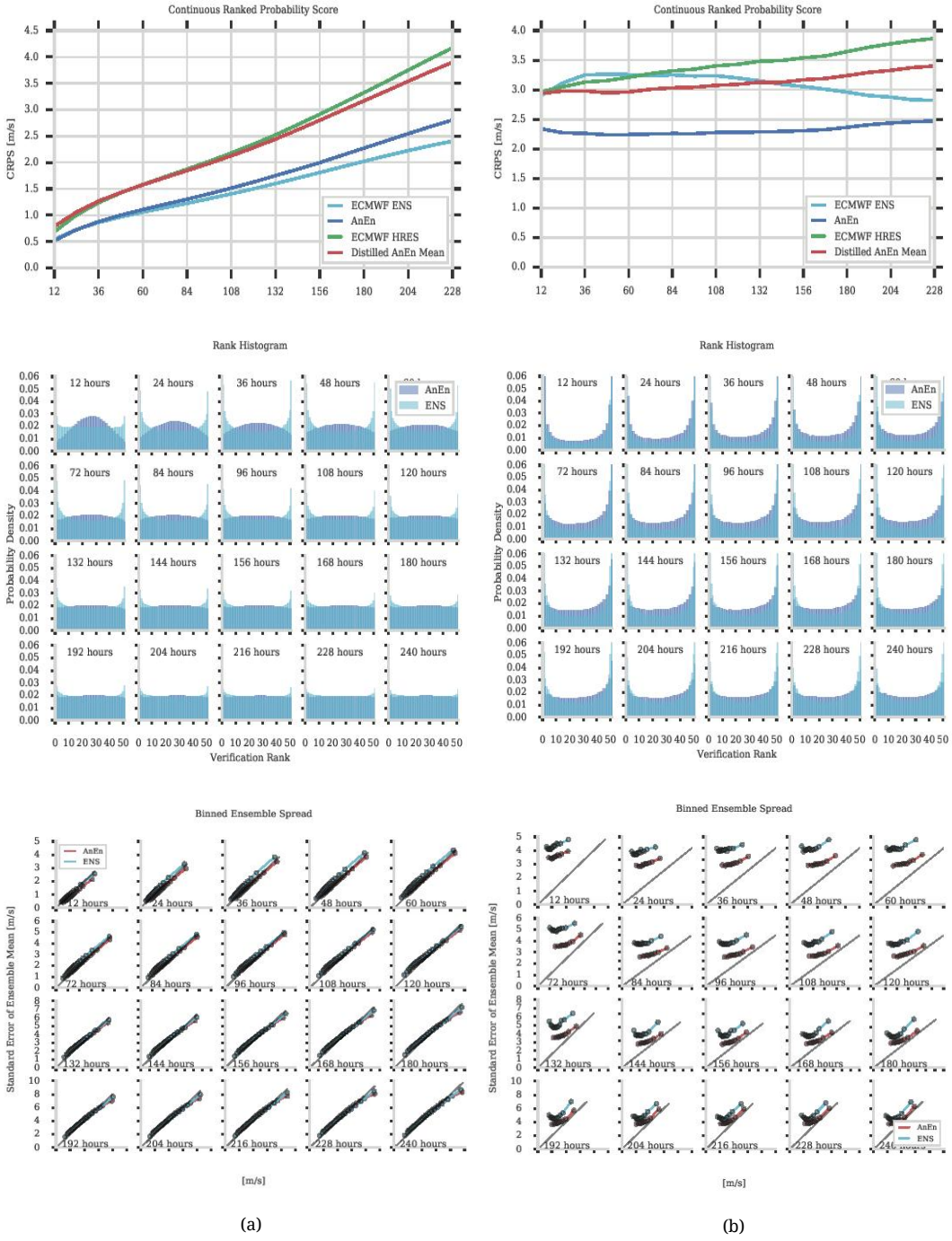


Figure S3. Probabilistic forecast evaluation metrics comparing the AnEn forecast of wind speed to forecasts produced by a ENS. Results with HRES analysis as ground truth are shown on the left (a), while results against Loon's measurements are on the right (b). From top to bottom, the metrics shown are CRPS, rank histogram, and binned-spread skill.

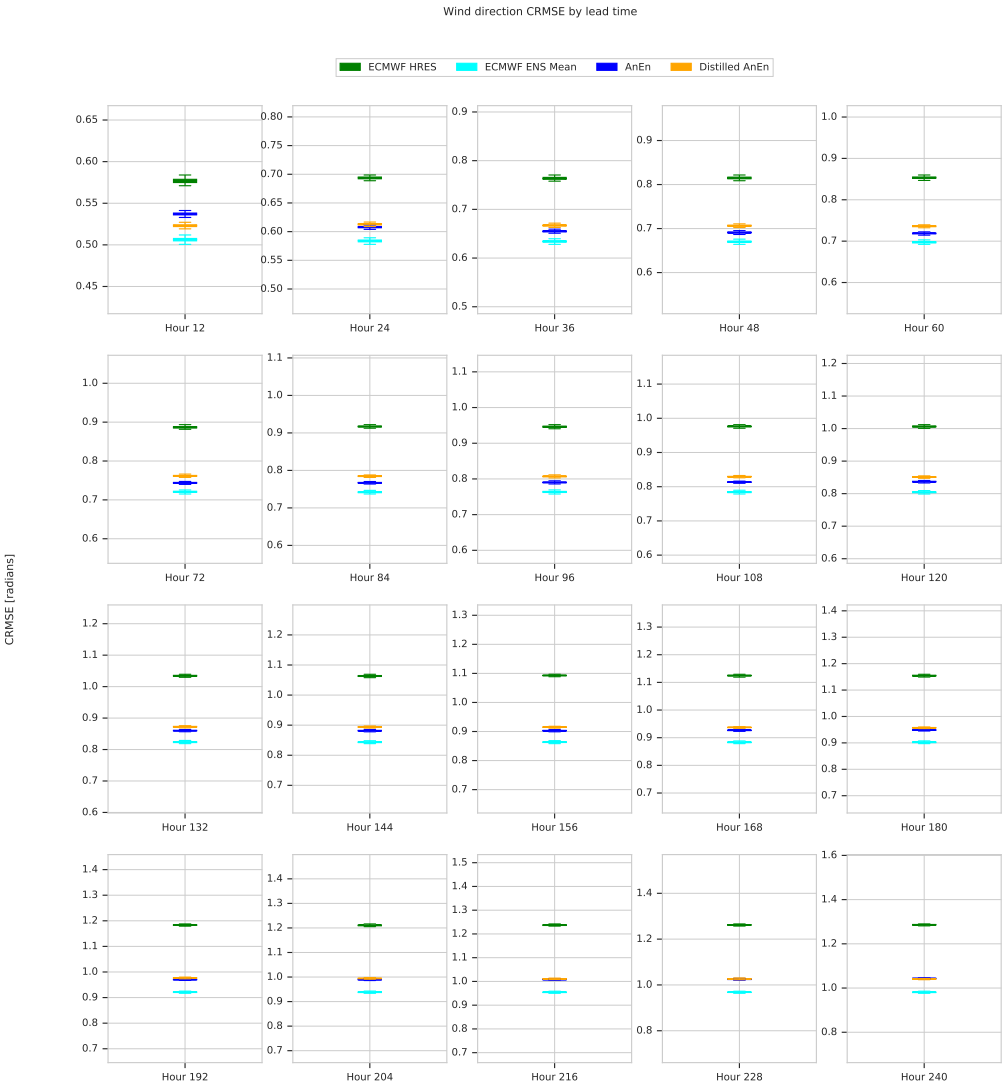


Figure S4. CRMSE for wind direction predictions including boxplots showing the bootstrap 90% confidence intervals.

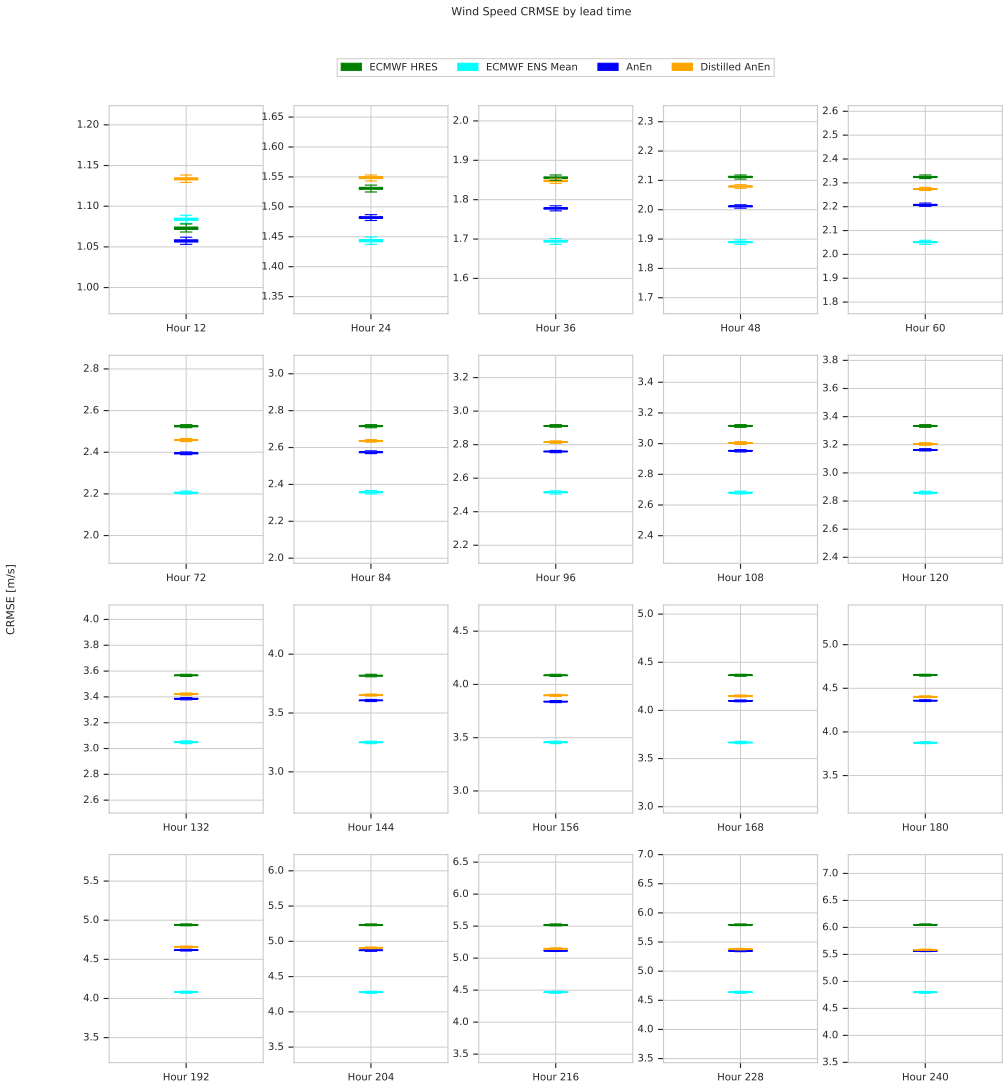


Figure S5. CRMSE for wind speed predictions including boxplots showing the bootstrap 90% confidence intervals.

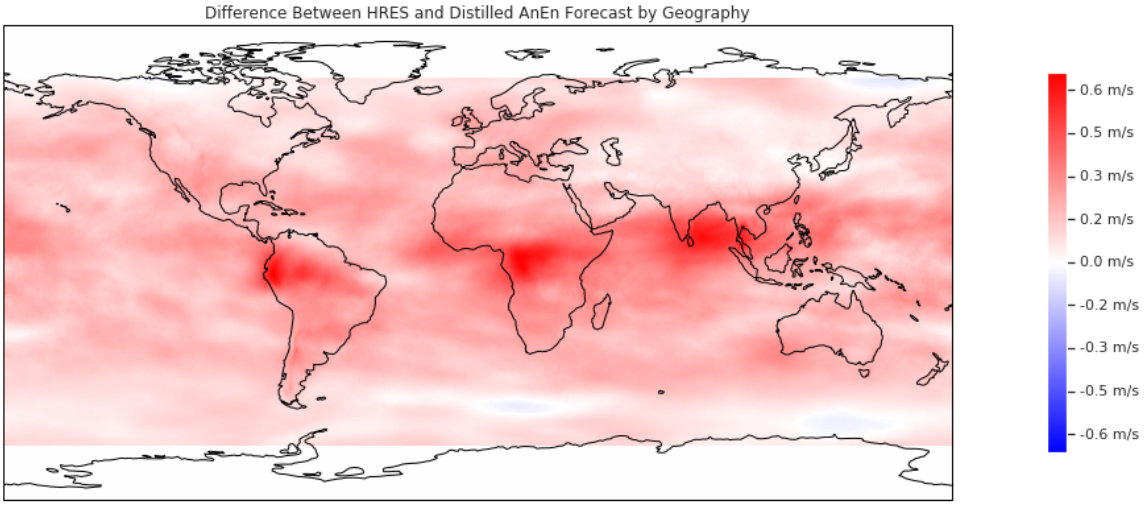
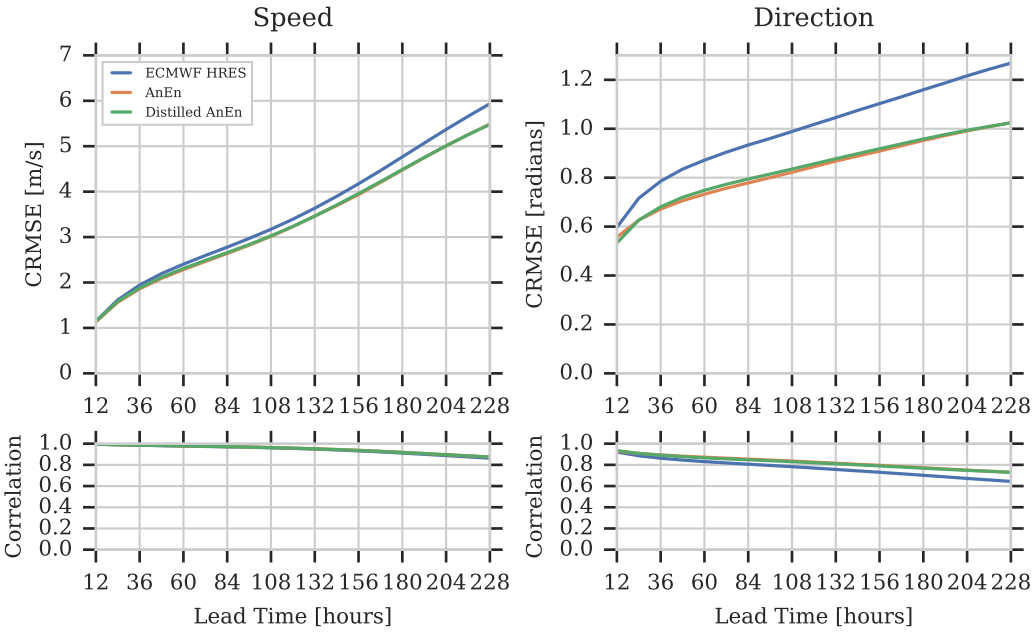
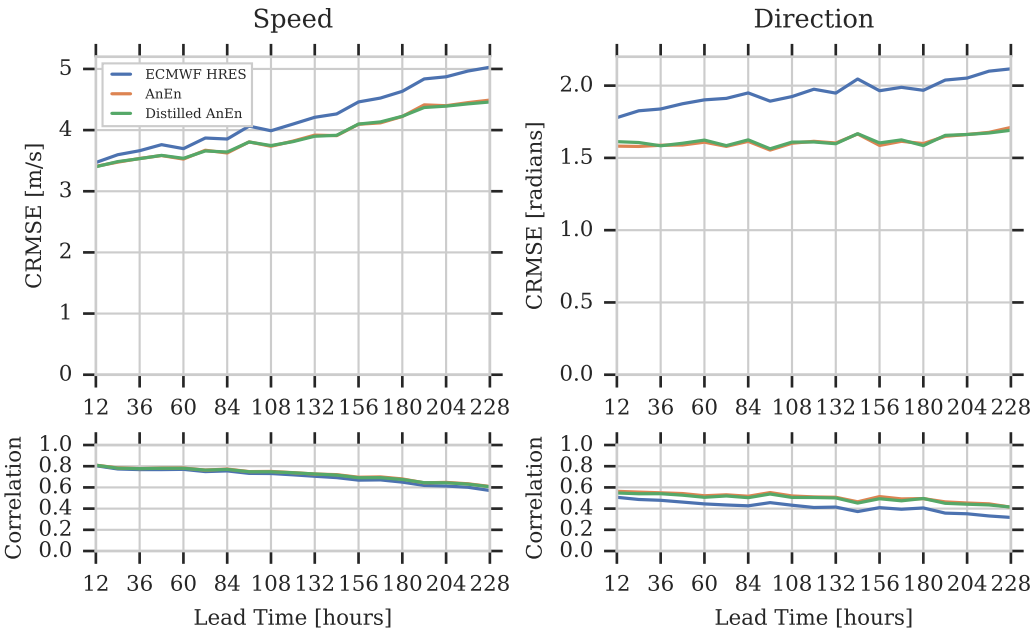


Figure S6. Geographical distribution of CRMSE for the distilled AnEn prediction of wind speed with HRES analysis as ground truth.



(a)



(b)

Figure S7. A deterministic wind speed and direction forecast skill comparison between the HRES, AnEn, and Distilled AnEn over all lead times is shown using as ground truth (a) HRES analysis and (b) Loon observations of stratospheric winds.

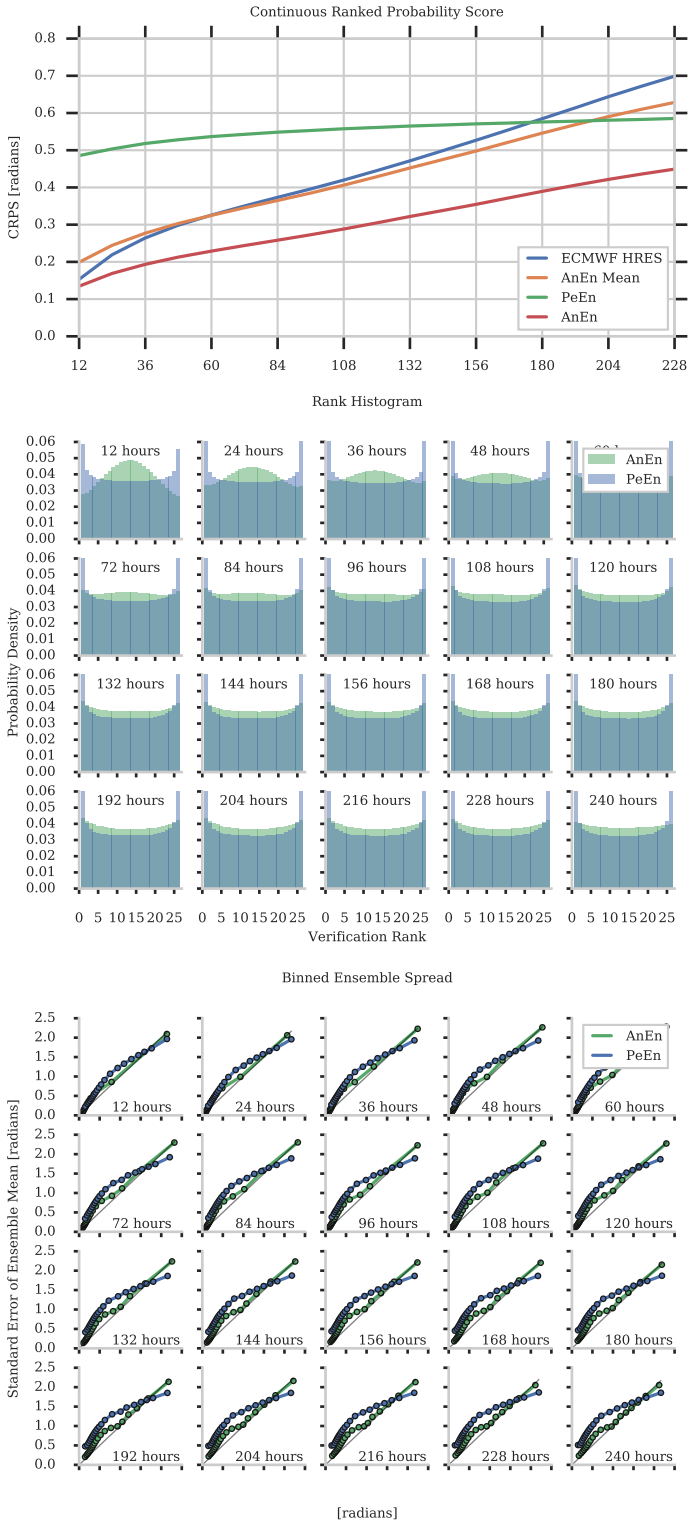


Figure S8. Probabilistic forecast evaluation metrics comparing the AnEn forecast of wind direction to forecasts produced by HRES, AnEn mean, and PeEn using HRES analysis as the ground truth. From top to bottom, the metrics shown are CRPS, rank histogram, and binned-spread skill.