Advanced Techniques for Imputing, Forecasting, and Synthesizing Multivariate Meteorological Data

Jarren Briscoe* Helen Catanese† Matthew Cann† Assefaw Gebremedhin*

Abstract

This study introduces two novel deep learning-based neural frameworks: FORESEER and WeatherGen, designed to address the critical need for accurate weather prediction in agricultural microclimates. Farmers often face significant challenges due to biases in public forecasts that do not account for their unique microclimates, leading to potential losses in crop yield and quality. FORESEER serves as a novel network available to either impute or forecast meteorological data. WeatherGen, a multi-agent Generative Adversarial Network (GAN), generates representative synthetic weather data, derived from extensive data collected from multiple farms across the western United States. The synthetic dataset, created by WeatherGen and made publicly available, has demonstrated promising results in zero-shot trials, indicating its potential for real-world application. These innovative neural frameworks aim to refine and predict weather data in areas of interest, empowering farmers to make informed decisions that directly enhance the quality and quantity of their produce.

1 Background and Motivation

Farmers and their crops are vital to society. However, many farmers in the western United States operate in unique microclimates that are not accurately represented in public weather forecasts. This discrepancy can lead to significant differences between forecasted and actual weather conditions experienced on the farms. Farmers require accurate local real-time weather and forecasting to take preventative measures against weather events such as frost or heat stress.

Furthermore, many weather forecasting techniques do not consider human influences such as irrigation. As a result, these microclimates are unable to fully benefit from the wealth of spatiotemporal deep learning resources available [1–3]. Meanwhile, it is not just farmers that benefit from meteorological data mining. It also plays a pivotal role in safeguarding finances, infrastructure, and public health [4, 5]. However, comprehensive meteorological datasets are required for weather-based applications including climatology, degree day models, weather forecasting models, and more. Yet, complete, accurate observational datasets are rare due to hardware failure, sensor drift, and communication loss, making data imputation a necessity.

Numerical weather prediction (NWP) models commonly use a single categorical land use for the entire grid. The grid cells are generally one elevation of flat terrain (for most operational models), and have single values for surface albedo (the fraction of sunlight that is diffusely reflected by a body), roughness length (a parameter based on the topography and vegetation that impacts the ability of winds aloft to reach the surface), etc. This is disadvantageous toward regions of complex terrain and/or varied land with many microclimates. In the case of Washington fruit growers, their irrigated orchards are in stark contrast to the dry, sparsely vegetated terrain that surrounds them.

In the field of weather-data imputation, prior research has primarily focused on imputation using neighboring stations [6] or simpler machine learning methods such as MLR, KNN, SVM, random forests, and ARIMA [6–9]. Our proposed model accommodates a growing weather network across multiple regions, with varying lengths of data from weeks to years. This approach facilitates more complex and realistic network imputation and forecasting, without the constraints of extensive data records.

State-of-the-art (SoTA) forecasting models have used attention-based mechanics in the past. Such forecasting models include Informer [10], Autoformer [11], FEDformer [12], and PDTrans [13]. However, recent work with fully connected models have been shown to outperform attention-based models in some applications [14, 15].

Our novel forecaster, FORESEER (Forecasting Operations and Real-time Environmental System for Environmental Estimation and Response), uses attention, convolutional, recurrent, and fully connected layers in parallel to leverage the advantages that each provides. Our method is shown to outperform the current SoTA methods TiDE [14] and DMLR [15] on a heretofore unexplored dataset.

*Washington State University
†Addium Inc.

In FORESEER, we pipeline predictions from SoTA spatiotemporal forecasts, specifically the National Blend of Models' (NBM's) [16] hourly forecasts. FORESEER improves the forecast to a location of interest, a subset of the NBM's gridded forecast, and provides a forecast every fifteen minutes. In addition to NBM's forecast, FORESEER also uses local weather history and identifiable information of the farms. This farm information allows accurate zero-shot tests on new farm sites.

In time-series forecasting, the closest work to our own is Google's TiDE [14]. TiDE incorporates three inputs: lookback (history), attributes (static data), and known future variables such as holidays (dynamic covariates). We use forecasts as the dynamic covariates input since our weather data has no known future variables. This is a new and intuitive approach. However, TiDE assumes that dynamic covariates' time delta (change in time) must equal the lookback's time delta. Furthermore, TiDE's forecast is limited to the dynamic covariate's horizon. We overcome these limitations by padding the NBM forecasts to comply with TiDE's dimension restrictions.

WeatherGen (Weather Generator) addresses the critical issue of weather data collection. Data collection is expensive and can be subsidized with synthetic generation. This synthetic data helps in the development and testing new weather prediction models. To the best of our knowledge, the closest work to WeatherGen is the WGAN from [17]. However, the authors work with daily data while we use high-resolution temporal data of fifteen-minute intervals. Furthermore, we introduce the capability to query synthetic results by a user-given date and time.

In summary, our contributions in this work are as follows:

- We develop FORESEER, a tool designed for weather forecasting and imputation.
- We present WeatherGen, a synthetic data generator that takes in the day of the year and the time of day to produce realistic weather data.
- We show that FORESEER outperforms TiDE and DMLR, two SoTA forecasting models, using a large-scale weather dataset for farm microclimates. We also evaluate TiDE and DMLR's performance in imputation and find FORESEER outperforms these methods.
- We provide reference code for our models, developed using Pytorch [18], available on GitHub.

2 Methods and Technical Solutions

In this section, we present our data collection, preparation, processing, and inference methods.

2.1 Data Collection and Processing

We collect our local farm data from ATMOS 41 and ATMOS 41W sensors within or near the orchards. These provide ground truths for local microclimate conditions.

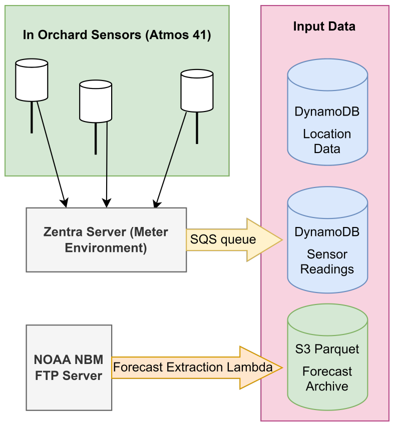

Figure 1: *An outline of the sources of our models' input and training data. All observational readings come from Atmos 41 sensors in or near orchard. Readings are sent directly to the Zentra server, which forwards them to an SQS queue ingested by into a dynamodb table.*

The Atmos 41 sensors we employ record temperature and other microclimate conditions with high precision. Table 1 details the measurements we use, along with their associated resolution, accuracy, and range.

For years, our group has constructed meteorological testing instruments used broadly in the western United States–predominately in Washington, Oregon, and California. Here, we detail our careful data processing that ensures the correct data for each site is combined and the time frames align. We detect corrupted data from sprinklers (that tend to skew the meteorological readings) and impute accordingly to ensure data quality. After carefully identifying many possible engineered features, we find that calculating dewpoint beforehand helps with predictions.

The architecture we rely on to collect both data sources is outlined in Figure 1. The ingest system is built in Amazon Web Services (AWS). Processing and transfer is handled by several lambdas. Data is stored in a combination of DynamoDB tables (for data that requires more timely reads and writes) and parquet files on Amazon S3. Parquet is an open source columnar storage file format that is compressed and indexed for efficient querying without downloading full files. Details on the parquet format can be found at https://parquet.apache.org/. Readings are reported every 15 minutes, with a 5 minute resolution. They are stored as 15 minute aggregates, with precipitation as an accumulation and all other features as averages. In cases

Table 1: *This table gives the resolution and accuracy of our observational data collected from the ATMOS 41.*

Variable	Resolution	Accuracy	Range
Air Temperature	$0.10°C$	$\pm0.60°C$	$-50 - 60°C$
Atmospheric Pressure	$0.01kPa$	$\pm0.05kPa$	$1 - 120kPa$
Vapor Pressure	$1W/m^2$	$\pm5\%$	$0 - 47kPa$
Wind Speed	$0.01m/s$	$\max(0.3m/s, 3\%)$	$0 - 30m/s$
Wind Direction	$1°$	$\pm1°$	$0° - 359°$
Wind Gusts	$0.01m/s$	$\max(0.3m/s, 3\%)$	$0 - 30m/s$
Solar Radiation	$1W/m^2$	$\pm5\%$	$0 - 1750W/m^2$

where the sensor does not report, or returns an error code (such as if it runs out of battery, or gets displaced), readings are dropped in that window. Observations are also extracted from the database into parquet format to be processed locally for use in our models.

Our static data contains each farm's identifiable information including latitude, longitude, elevation, slope, aspect, and exposure of the terrain. Slope (the angle of the terrain) and aspect (orientation that the terrain faces) are calculated using the 1/3 arc second (or 10 meters) National Elevation Dataset and third-order finite difference as in [19] and [20]. We calculate exposure (how sheltered the terrain is) using a 9x9 Digital Elevation Model (DEM) grid centered around the station by comparing the elevation of each grid cell compared to the center cell.

The local farm's data, which we refer to as "history", includes observations of air temperature, atmospheric pressure, vapor pressure, wind speed, wind direction, wind gusts, and solar radiation at approximately two meters above ground level, as recorded by the ATMOS 41. We drop the "TEMPC_RH_Sensor", lightning count, lightning distance, and accelerometer columns. The RH sensor temperature is dropped because it is a duplicate measurement and less accurate than the primary temperature probe. Lightning and accelerometer data are dropped because they are not consistently observed across the weather network.

We preprocess all features with the min-max scaling method from zero to one. For time, however, we take extra measures to properly capture its cyclical measure before scaling. We convert each date into a numerical day of year (DoY) (1 to 366) and each time of day into "seconds from midnight" (SFM) (1 to 86,400). Then, we separately scale DoY and SFM to $[-2\pi, 2\pi)$ before casting both time features into two pairs of sine and cosine. After the trigonometry conversions, we use the same min-max scaling method.

With local data collection, our sensors sometimes pick up local human-influenced climate changes such as sprinkler use. We detect the onset of sprinkler data corruption by identifying quick changes in dewpoint and temperature from a moving one-hour window. We then drop the data and impute.

2.2 FORESEER and WeatherGen In this section we present our novel weather-based neural architectures: FORESEER and WeatherGen. FORESEER, or "Forecasting Operations and Real-time Environmental System for Environmental Estimation and Response" is our weather forecasting and imputation tool. WeatherGen is our synthetic meteorological data generator.

FORESEER and WeatherGen share several common features. They both utilize advanced deep learning techniques such as casting to a learnable embedding space, multi-head attention layers (MHA) [21], recurrent neural networks with gated recurrent units (RNN-GRUs) [22], one-dimensional convolutional neural networks (1D-CNNs), leaky ReLU activations (LReLUs), data fusion, momentum-based optimizers [23], random weight dropouts [24], batch normalizations (BNs) [25], and layer normalizations (LNs) [26].

2.2.1 FORESEER We illustrate FORESEER in Figure 2. Its multi-modal input of static farm data (Static), farm weather history (History), and NBM's zero-time data (NBM) gives it unique properties as it can distinguish between sites and uses NBM as supervision.

As shown in Figure 2, History and NBM inputs are embedded using a dense layer, layer normalization, leaky ReLU activation, then a dropout (LLLD) followed by a 1D-CNN that convolves over the channels. Static is embedded using LBLD twice (LLLD with batch normalization instead of layer normalization).

We use MHAs with four heads to capture long-term dependencies and zone in on important information as some time periods tend to be more informative than others. Each attention layer is followed by a leaky ReLU activation. Our MHAs serve as peek or residual layers. Peek connections have significant precedence in the literature for improving models [14, 21].

Additionally, we use stacked bi-directional RNN-GRUs in parallel with the MHAs. Our RNN layer en-

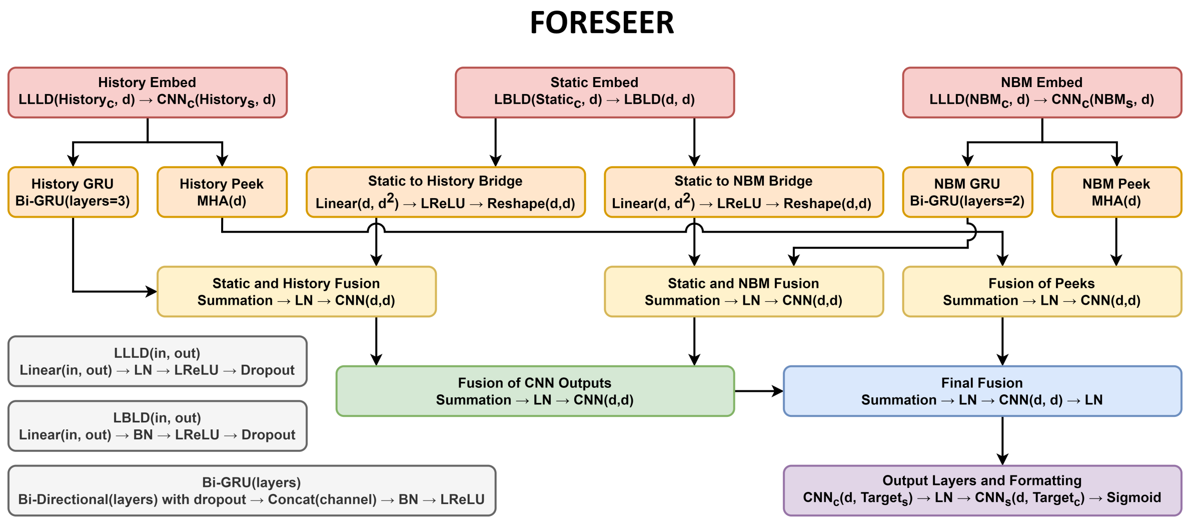
FORESEER
History Embed
LLLD(History_c, d) → CNN_c(History_s, d)
Static Embed
LBLD(Static_c, d) → LBLD(d, d)
NBM Embed
LLLD(NBM_c, d) → CNN_c(NBM_s, d)
History GRU
Bi-GRU(layers=3)
History Peek
MHA(d)
Static to History Bridge
Linear(d, d²) → LReLU → Reshape(d,d)
Static to NBM Bridge
Linear(d, d²) → LReLU → Reshape(d,d)
NBM GRU
Bi-GRU(layers=2)
NBM Peek
MHA(d)
Static and History Fusion
Summation → LN → CNN(d,d)
Static and NBM Fusion
Summation → LN → CNN(d,d)
Fusion of Peeks
Summation → LN → CNN(d,d)
LLLD(in, out)
Linear(in, out) → LN → LReLU → Dropout
Fusion of CNN Outputs
Summation → LN → CNN(d,d)
Final Fusion
Summation → LN → CNN(d, d) → LN
LBLD(in, out)
Linear(in, out) → BN → LReLU → Dropout
Bi-GRU(layers)
Bi-Directional(layers) with dropout → Concat(channel) → BN → LReLU
Output Layers and Formatting
CNN_c(d, Target_s) → LN → CNN_s(d, Target_c) → Sigmoid

Figure 2: *This is the architecture diagram for FORESEER. The light gray boxes in the bottom left are commonly repeated blocks. Each layer is followed by the leaky ReLU activation, except for the output which is a sigmoid.* CNN_c *is a convolution over the channel dimension and* CNN_s *convolves over the temporal dimension. Without the subscript, CNN convolves over a latent dimension. Furthermore,* $Input_c$ *stands for the channel dimension of the input and likewise with* $Input_s$*; for example,* $History_s$ *is the temporal dimension of the history input.*

forces a temporal bias (closer values in time hold more significance) while the MHA excels at long-term dependencies. For the GRUs, we use the usual activation functions of sigmoid for the reset and update gates and tanh for the new gate. We choose the GRU over the long-short-term memory unit (LSTM) since the GRU is simpler and LSTM's largest benefit is its state which we do not need. Furthermore, [22] finds that GRUs are comparable to LSTMs.

During convolution, sometimes the kernel tensor does not perfectly divide into the space it traverses. Adjusting accordingly, we use causal padding which adds zeros to the beginning of the data. This padding technique allows some retrocasting, and we find that this padding performs better than "same" or "valid" padding. Our versatile CNNs can convolve over the temporal dimension or the channel dimension.

Our data fusion techniques are summation and channel-wise concatenation. We only use channel-wise concatenation to combine the forward and backward RNN-GRUs in the bi-directional block.

Random dropouts (setting random input values to zero) regularize a network. In the complexity context, dropouts reduce the amount of neurons used which simplifies the model. Furthermore, dropouts help regularize by reducing heavy leaning on a few specific neurons. We use dropout frequencies in the range of $[0.1, 0.2]$.

We use batch normalization and layer normalizations. Batch normalizations mitigate internal covariate shift, where the distribution of each layer's inputs changes during training. Batch normalization works by taking the difference of the input and batch mean then dividing by the batch standard deviation. Layer normalization is like batch normalization, except it considers inputs to a layer on a single instance instead of a batch and it runs during evaluation. We use layer normalizations after data fusions by summation following the examples of [14, 21].

2.2.2 WeatherGen WeatherGen is our synthetic data generator that we create for public use. Due to privacy concerns, it produces weather data without static information such as latitude and longitude.

WeatherGen can be defined as a Conditional Wasserstein Generative Adversarial Network with Gradient Penalty (CWGAN+GP). This is a variant of the Wasserstein GAN (WGAN) [27] that builds upon the original GAN model. Traditionally, GANs are a two-player game with a discriminator and a generator where the discriminate differentiates between the real data and the synthetic data produced by the generator. Then the GAN optimizes using the binary cross-entropy loss function. GANs are prone to mode collapse—a convergence to an unfavorable local minima where the generator repeats the same data. Conditional GANs (CGANs) allow one to query a class to the GAN [28]. We combine the WGAN+GP and CGAN and apply it to our weather regression task, allowing us to query realistic weather given the date and time.

WeatherGen separately trains its two components: the critic C and the generator G. Because the critic struggles with differentiating real and fake data at first, we train the critic five times for every time the generator trains once.

Note that $\mathbb{E}[]$ denotes an expected value. In the generator loss L_G (2.1) and the critic loss L_C (2.2), this is done using the arithmetic mean. To find the losses, we use sample mini-batches of real-data samples $x \sim P_r$, extract their time labels t, and generate synthetic samples $\tilde{x} \sim P_G$. We calculate the expected gradient penalty (GP) using an interpolated sample $\hat{x}$ from the real and fake data (2.3). Additionally, we use $\lambda = 10$. Then, the critic's total loss is $L_C + GP$ (2.4).

$$L_G = -\mathbb{E}_{\tilde{x} \sim P_G}[C(\tilde{x}, t)] \tag{2.1}$$

$$L_C = \mathbb{E}_{\tilde{x} \sim P_G | t}[C(\tilde{x}, t)] - \mathbb{E}_{x \sim P_r}[C(x, t)] \tag{2.2}$$

$$GP = \lambda \mathbb{E}_{\hat{x} \sim P_{\hat{x}} | t}[(||\nabla_{\hat{x}} C(\hat{x}, t)||_2 - 1)^2] \tag{2.3}$$

$$L_{C_{total}} = L_C + GP \tag{2.4}$$

Generator We illustrate WeatherGen's generator in Figure 3. This generator takes a noise vector and time label as inputs. The noise vector is created by randomly sampling $d = 96$ instances from a normal distribution. The time label, represented as "t", is a quadruple consisting of the normalized sine and cosine pairs for the day of the year and seconds from midnight. Each sample in the minibatch has a single time label indicating the final time desired. Concurrently, while casting the time label into a $d \times d$ space and using positional encoding, we duplicate the noise in a $d \times d$ space. To mitigate the risk of mode collapse, we avoid adding a linear layer before summing noise with the labels. Otherwise, CGANs can zero out the noise and rely entirely on the inputted label. After data fusion, we apply multi-head attention and a convolution block in parallel before fusing and entering the final convolution block. All neural layers are followed by the Leaky ReLU activation, except for the output layer.

Critic We depict the neural architecture of WeatherGen's critic in Figure 4. In meteorological data generation, time is a soft label since temporally local groups are similar. For example, the expected weather on August 23 at noon is estimated to be similar to August 24 at 12:30PM in any year. To help the critic understand this, we add Gaussian noise (with $\mu = 0$ and $\sigma = 0.025$) to the time label.

3 Experiments and Results

In this section, we test the forecasting and imputing capabilities of FORESEER and the synthetic data quality

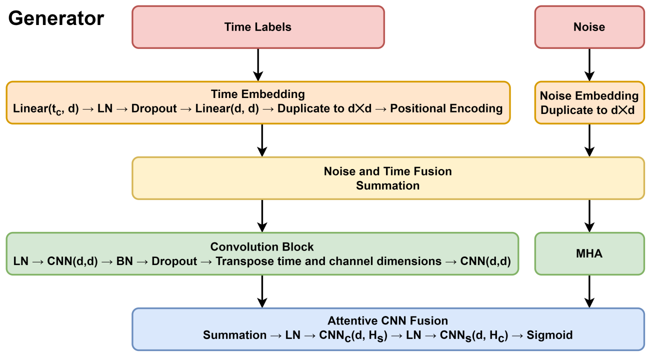
Generator
Time Labels
Noise
Time Embedding
Linear(t_C, d) → LN → Dropout → Linear(d, d) → Duplicate to d×d → Positional Encoding
Noise Embedding
Duplicate to d×d
Noise and Time Fusion
Summation
Convolution Block
LN → CNN(d,d) → BN → Dropout → Transpose time and channel dimensions → CNN(d,d)
MHA
Attentive CNN Fusion
Summation → LN → CNN_C(d, H_S) → LN → CNN_S(d, H_C) → Sigmoid

Figure 3: *The neural architecture of WeatherGen's generator. Aside from color, the notations from Figure 2 apply.*

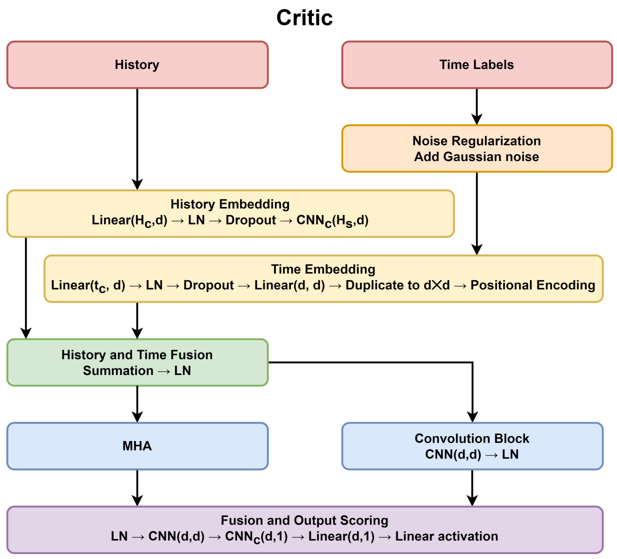
Critic
History
Time Labels
Noise Regularization
Add Gaussian noise
History Embedding
Linear(H_C,d) → LN → Dropout → CNN_C(H_S,d)
Time Embedding
Linear(t_C, d) → LN → Dropout → Linear(d, d) → Duplicate to d×d → Positional Encoding
History and Time Fusion
Summation → LN
MHA
Convolution Block
CNN(d,d) → LN
Fusion and Output Scoring
LN → CNN(d,d) → CNN_C(d,1) → Linear(d,1) → Linear activation

Figure 4: *The neural architecture of WeatherGen's critic. Aside from color, the notations from Figure 2 apply.*

produced by WeatherGen.

3.1 Hyperparameters and Metrics In our forecasting and imputing experiments, we use the AdamW optimizer [23] for all models and baselines. Our momentum hyperparameters for AdamW are (0.9, 0.999). Since we use a learning rate scheduler that reduces the learning rate upon reaching a plateau in the loss of the validation data, we begin with a higher learning rate of 0.001. With three consecutive validation losses greater than the best, the learning rate is decreased tenfold. If the validation loss increases for six consecutive epochs, training is stopped. No experiments require more than 150 epochs. We use minibatch sizes of 128, with 25 minibatches per epoch. We optimize all models for the mean-squared-error (MSE) metric.

For FORESEER's hyperparameters, we notice negligible difference between four and eight heads for the MHA layers so we choose the simpler four. For kernel sizes, we experiment odd numbers between one and fifteen. For the RNNs, we stack two bi-directional layers as it performs similarly to three. Our embedding space is a $d \times d = 128 \times 128$ tensor. We choose d^2 tensors for efficient computation in GPUs. We find that $d = 128$ is comparable to $d = 256$ and is modestly better than $d = 64$.

We find that [14]'s implementation did not include the static features they mention in their paper and is created in Tensorflow. As such, we present a PyTorch version that includes static features and overcomes the dimension restrictions for the input we use to feed in predictions and they use for dynamic covariates. Our implementation does not consider categorical features.

For TiDE's hyperparameters, we use suggestions from the original work plus the trial and error method. The important hyperparameters are the encoding dimensions and the number of residual blocks. We find that encoding dimensions of 64×64 and 3 residual blocks each for encoding and decoding performs best. For DMLR, we use layers with 512 hidden units each.

We analyze model performance by mean-absolute error (MAE), MSE, root-MSE (RMSE), and mean-absolute-scaled error (MASE). MASE is less well known so we describe the metric in detail.

MASE is scale-independent, allowing for comparisons across various scales. Additionally, it is based on a naïve forecast, which is a simple forward fill of data. MASE is in $[0, \infty)$ and lower scores indicate better performance. MASE of 0 is a perfect forecast, MASE in $[0, 1)$ means the forecast does better than the naïve forecast, MASE of 1 occurs when the forecast is on par with the naïve forecast, and MASE in $(1, \infty)$ is when the forecast is worse than the naïve forecast.

Now, we formally define MASE. Let J be the total number of forecasts made, Y_j be the actual value, F_j be the forecasted value, T be the total number of observations in the time series, and Y_t be the actual value at time t.

$$\text{MASE} = \frac{J^{-1} \sum_{j=1}^{J} |Y_j - F_j|}{(T-1)^{-1} \sum_{t=2}^{T} |Y_t - Y_{t-1}|} \tag{3.5}$$

For WeatherGen, we also use the AdamW optimizer. However, we change the hyperparameters to a constant learning rate of 0.0002 and the momentum parameters to (0.5, 0.95). We use lower momentum parameters to help the critic and generator converge as one may improve too quickly otherwise. We train the critic five times for every generator training. Our embedding space for the critic and generator is 96×96, the number of heads for the multihead-attention layer is eight, and the kernel sizes are eight, except for the generator's final CNN layer which has size four. We train WeatherGen for 200 epochs, saving every 10 epochs. We find epoch 170 yields the most realistic data.

Table 2: *Comparison of Models for Forecasting*

Model Name	MAE	RMSE	MASE
TiDE	0.0388	0.0495	0.247
DMLR	0.0348	0.0454	0.221
FORESEER	**0.0326**	**0.0422**	**0.206**

Table 3: *Comparison of Models for Imputation*

Model Name	MAE	RMSE	MASE
TiDE	0.0269	0.0407	0.1120
DMLR	0.0163	0.0278	0.0677
FORESEER	**0.0149**	**0.0254**	**0.0620**

3.2 FORESEER Results In this section, we compare FORESEER with the SoTA methods, TiDE and DMLR, in forecasting (Table 2) and imputing (Table 3). FORESEER outperforms both alternative methods in all metrics (MAE, RMSE, MASE) for forecasting and imputing.

Using a RTX 2060 GPU, FORESEER trains quicker than TiDE but slower than DMLR. We present the averaged times in minutes:seconds format. For forecast training time, FORESEER takes 3:17, TiDE takes 5:17, and DMLR takes 2:47. For imputation training time, FORESEER takes 5:17, TiDE takes 9:55, and DMLR takes 3:28. Regarding epoch count, TiDE typically takes between 80 and 110 epochs, FORESEER between 50 and 60 epochs, and DMLR between 40 and 50 epochs.

To show proper fitting, we provide two bar graphs showing each model's training loss, validation loss, best validation loss, and test loss in Figure 5 and Figure 6. Recall that our loss is MSE.

Figure 5 shows the results from models trained on all sites with train/validation/test datasets partitioned randomly by time. The zero-shot illustration, Figure 6, shows results from models trained on all datetimes but each train/validation/test dataset contains unique and random sites. FORESEER outperforms the other models in both cases, but does so much more in the zero-shot example. This suggests FORESEER excels at transferring datetime knowledge to other sites as compared to the other methods. Recall that TiDE and FORESEER consider static data (site information)

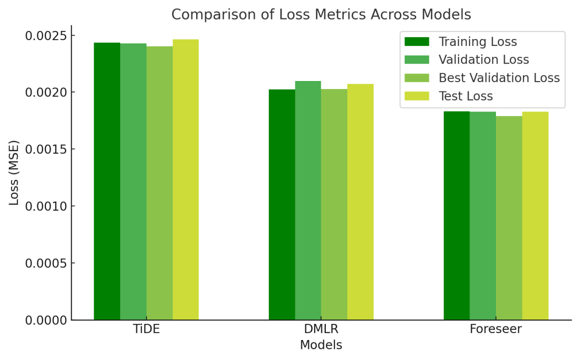

Figure 5: *MSE for each model. Models are trained for all sites, but not on all available times.*

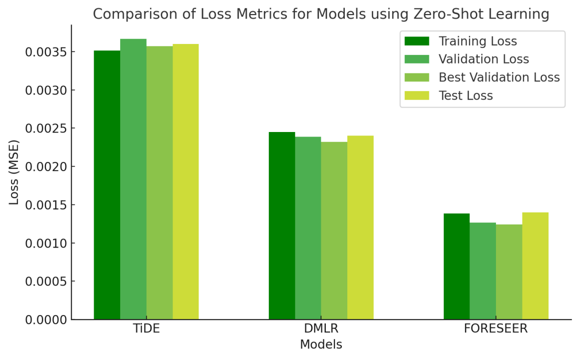

Figure 6: *MSE on for each model. Models are trained for all available times, but not on all sites.*

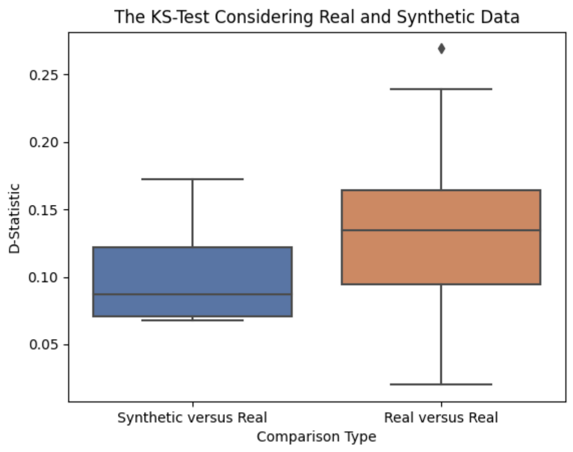

Figure 7: *Box plots comparing the distribution of distances between synthetic and real data and the distribution of distances between each real site pair (excluding self comparison). We see that the synthetic data is often more representative of each site than any other single real site.*

while DMLR does not.

3.3 WeatherGen Results In this section, we analyze WeatherGen's synthetic data generation.

To show the quality of the synthetic data, we compare the distance (D-Statistic) between the synthetic and real data to the distance between real data from site i and site j in Figure 7.

The D-Statistic is derived from the two-tailed Kolmogorov-Smirnov test (KS-test) [29,30], which compares the cumulative density function (CDF) of two distributions. The D-Statistic is the measure between the distributions, along with a p-value that indicates the likelihood that the difference is due to chance. For all of our tests, $p < 0.001$.

In Figure 7's "Synthetic versus Real" box plot, we compare the synthetic data's distribution to the distribution of data from each site. The "Real versus Real" part of the figure compares each unordered pair of sites, excluding self comparison. We see that the synthetic data's distribution is usually more representative for a given individual site than any other single real site to site comparison.

This section presents comparisons between synthetic and real meteorological data, as illustrated in Figure 8. Each graph displays temperature and solar radiation over a 24-hour period.

In the summer examples, typical diurnal temperature variations are evident, with a morning minimum and afternoon maximum. Furthermore, we see the presence of cloud cover reducing solar radiation from its typical bell shape in both examples. Cloud cover is more prevalent in the synthetic figure, which would act to moderate daytime temperatures.

The winter examples illustrate small diurnal temperature ranges common during precipitation systems (real winter captures a rain/freezing rain event) and during cold pools common to the region [31]. We see that the synthetic winter data in the top right shows a warm inflection around midnight. This inflection may occur due to increased cloud cover or wind speed, mixing the nocturnal boundary layer. Despite an apparent lack of cloud cover, there is no clear temperature response to solar radiation, suggesting the presence of a cold air pool. It is more difficult to maintain a cold air pool with small amounts of cloud cover, but does remain possible. However, this could also indicate that WeatherGen is struggling to generate wintertime cold pool events due to their complexity and lack of time-dependent data.

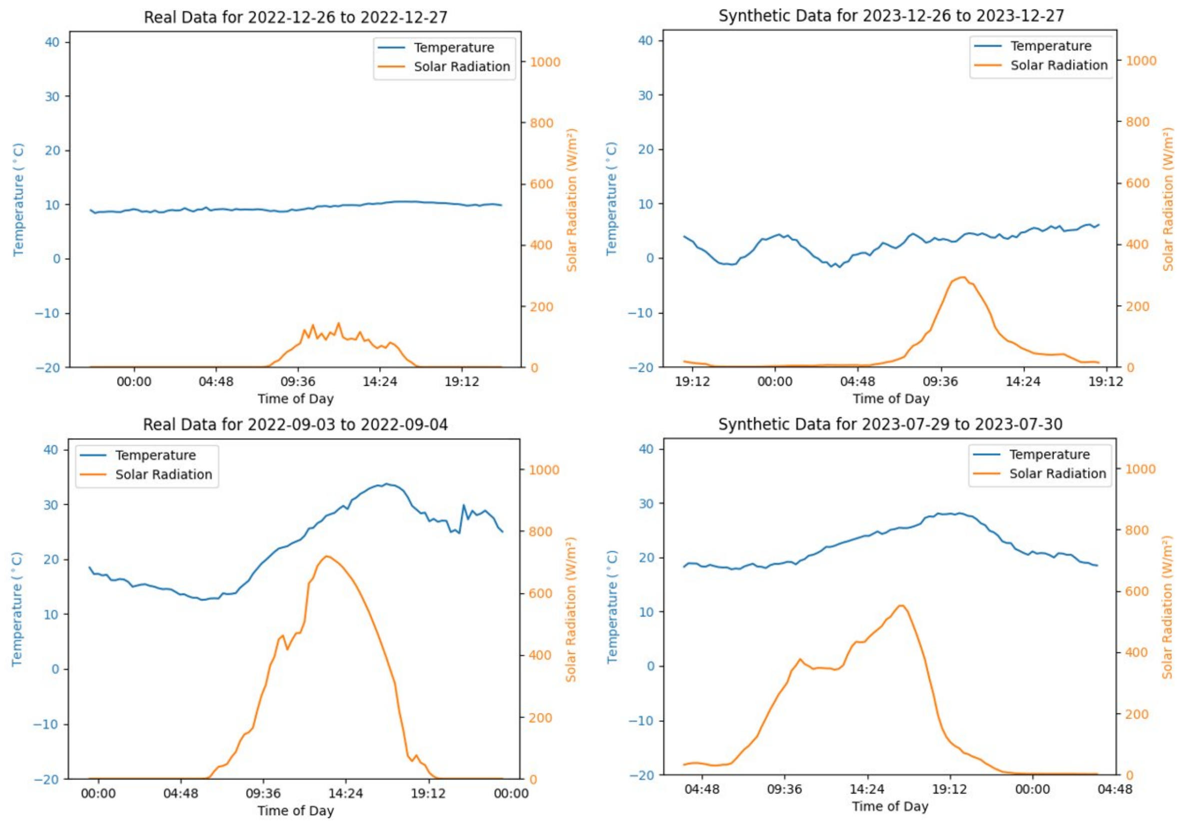

Figure 8: *Line plots of temperature and solar radiation for twenty-four hours for anonymous farm sites. The two graphs on the left are from real data and the two on the right are from synthetic data. The two top graphs are from winter and the two bottom graphs are from summer.*

4 Significance and Impact

We address the issues of biased weather forecasting, corrupted weather data, and insufficient weather data using two innovative neural frameworks specifically designed for weather data: FORESEER, a multi-modal weather forecasting and imputing model; and WeatherGen, a generator of synthetic weather data. These frameworks have been rigorously tested using data collected from various farms across the western United States, a region characterized by complex canopies and a susceptibility to bias in public gridded forecasts. We build upon the spatio-temporal SoTA weather forecasters by feeding forecasts into our model. Then, we outperform notable deep-learning architectures in forecasting and imputing weather data in farm microclimates. Finally, we create synthetic weather data for these microclimates and publish the code for our neural frameworks.

Our empirical evaluations demonstrate that FORESEER has substantial improvements over the SoTA baselines for training on all sites and zero-shot learning on withheld sites. Additionally, FORESEER's low RMSE score indicates its reduced susceptibility to bias towards underrepresented distributions. This is especially advantageous when predicting rare weather events, which are naturally underrepresented and thus provide limited training instances. Given the potential risks to life and property, accurate prediction of these events is of paramount importance. This accurate forecasting can immediately impact farmers by seeing future events and taking preventative measures, such as reacting to forecasted frost or heat stress. Both events can be devastating to crops and livestock if not anticipated.

One direction for future work involves the collection of farm data from geolocations outside of the western United States, thereby ensuring a more globally representative sample of farms. For FORESEER, sensitivity analysis, temporal knowledge graphs, and other methods to understand the model's reasoning can be researched. Furthermore, the problem of learning rate scheduling for WeatherGen, a WGAN variant, remains open and warrants further investigation. The potential of a dynamic critic training ratio also merits exploration.

References

[1] T. Yuan, J. Zhu, K. Ren, W. Wang, X. Wang, and X. Li, “Neural network driven by space-time partial differential equation for predicting sea surface temperature,” in *2022 IEEE International Conference on Data Mining (ICDM)*, pp. 656–665, 2022.

[2] A. Bojesomo, H. Al-Marzouqi, and P. Liatsis, “Spatiotemporal vision transformer for short time weather forecasting,” in *2021 IEEE International Conference on Big Data (Big Data)*, pp. 5741–5746, 2021.

[3] A. Diehl, L. Pelorosso, C. Delrieux, C. Saulo, J. Ruiz, M. E. Gröller, and S. Bruckner, “Visual analysis of spatio-temporal data: Applications in weather forecasting,” in *Computer Graphics Forum*, vol. 34, pp. 381–390, Wiley Online Library, 2015.

[4] J. Cifuentes, G. Marulanda, A. Bello, and J. Reneses, “Air temperature forecasting using machine learning techniques: A review,” *Energies*, vol. 13, no. 16, 2020.

[5] R. Abdel-Aal, “Hourly temperature forecasting using abductive networks,” *Engineering Applications of Artificial Intelligence*, vol. 17, no. 5, pp. 543–556, 2004.

[6] F. Rafii and T. Kechadi, “Collection of historical weather data: Issues with missing values,” in *Proceedings of the 4th International Conference on Smart City Applications*, SCA ’19, (New York, NY, USA), Association for Computing Machinery, 2019.

[7] Doreswamy, I. Gad, and B. Manjunatha, “Performance evaluation of predictive models for missing data imputation in weather data,” in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1327–1334, 2017.

[8] E. Afrifa-Yamoah, U. A. Mueller, S. M. Taylor, and A. J. Fisher, “Missing data imputation of high-resolution temporal climate time series data,” *Meteorological Applications*, vol. 27, no. 1, p. e1873, 2020.

[9] J. P. Boomgard-Zagrodnik and D. J. Brown, “Machine learning imputation of missing mesonet temperature observations,” *Computers and Electronics in Agriculture*, vol. 192, p. 106580, 2022.

[10] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, “Informer: Beyond efficient transformer for long sequence time-series forecasting,” *CoRR*, vol. abs/2012.07436, 2020.

[11] H. Wu, J. Xu, J. Wang, and M. Long, “Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting,” 2021.

[12] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, “Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting,” 2022.

[13] J. Tong, L. Xie, W. Yang, and K. Zhang, “Probabilistic decomposition transformer for time series forecasting,” 2022.

[14] A. Das, W. Kong, A. Leach, S. Mathur, R. Sen, and R. Yu, “Long-term forecasting with tide: Time-series dense encoder,” 2023.

[15] A. Zeng, M. Chen, L. Zhang, and Q. Xu, “Are transformers effective for time series forecasting?,” 2023.

[16] J. P. CRAVEN, D. E. RUDACK, and P. E. SHAFER, “National blend of models: a statistically post-processed multi-model ensemble.,” *Journal of Operational Meteorology*, vol. 8, no. 1, 2020.

[17] C. Besombes, O. Pannekoucke, C. Lapeyre, B. Sanderson, and O. Thual, “Producing realistic climate data with generative adversarial networks,” *Nonlinear Processes in Geophysics*, vol. 28, no. 3, pp. 347–370, 2021.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[19] A. Skidmore, *A comparison of techniques for calculating gradient and aspect from a gridded digital elevation model.* Chapman and Hall/CRC, 2006.

[20] M. Dunn and R. Hickey, “The effect of slope algorithms on slope estimates within a gis,” *Cartography*, vol. 27, pp. 9–15, 06 1998.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2017.

[22] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” 2014.

[23] I. Loshchilov and F. Hutter, “Fixing weight decay regularization in adam,” *CoRR*, vol. abs/1711.05101, 2017.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[25] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*, pp. 448–456, pmlr, 2015.

[26] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” 2016.

[27] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein gan,” 2017.

[28] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *CoRR*, vol. abs/1411.1784, 2014.

[29] A. N. Kolmogorov, “Sulla determinazione empirica di una legge didistribuzione,” *Giorn Dell’inst Ital Degli Att*, vol. 4, pp. 89–91, 1933.

[30] N. Smirnov, “Table for estimating the goodness of fit of empirical distributions,” *The annals of mathematical statistics*, vol. 19, no. 2, pp. 279–281, 1948.

[31] C. D. Whiteman, S. Zhong, W. J. Shaw, J. M. Hubbe, X. Bian, and J. Mittelstadt, “Cold pools in the columbia basin,” *Weather and Forecasting*, vol. 16, no. 4, pp. 432 – 447, 2001.