SCALEABLE DATA SYSTEMS FOR WEATHER FORECASTING

Eron Neill

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science

University of North Carolina Wilmington

2024

Approved by

Advisory Committee

Dr. Gulustan Dogan

Dr. Minoo Modaresnezhad

Dr. Karl Ricanek

Chair

Contents

ABSTRACT

This paper presents a modular and reusable system for processing, modeling, and visualizing weather data, as well as a proof of concept implementation focusing specifically on rainfall forecasting. The system was designed to address the challenges of building and maintaining weather data systems by enabling flexible, scalable, and standardized workflows. The proof of concept utilizes ERA-5 Land, the same dataset used in our previous rainfall forecasting research. The objectives were twofold: to implement a generalized system architecture for weather data systems and to validate this architecture through practical implementation. The proof of concept involved creating a model ensemble using machine learning techniques such as autoencoders and transformers, while calculating a variety of error metrics for model evaluation. Results show promising accuracy in rainfall forecasts, with a mean absolute error of 0.23 mm/hour. While the study does not delve deeply into model optimization, it emphasizes data-driven insights and lays the groundwork for future, more refined weather forecasting systems.

ACKNOWLEDGMENTS

I would like to extend my sincerest thanks to my fantastic advisory committee, Dr. Karl Ricanek, Dr. Gulustan Dogan, and Dr. Minoo Modaresnezhad, for their wisdom, and their ongoing commitment to their students. I would also like to thank my family for loving and supporting me throughout this endeavor. It would not have been possible without them. I would also like to extend my thanks to the UNCW Applied AI Lab, the UNCW Department of Computer Science, and the Congdon School of Supply Chain, Business Analytics, and Information Systems for fostering such a wonderful learning environment where students have access to the resources and expertise needed to explore their curiosity.

1 Introduction

Effective weather forecasting relies on the ability to process, model, and visualize large volumes of data with precision and flexibility. This task often involves the integration of diverse data sources and the application of complex modeling techniques, as described by Bauer et al. in their comprehensive overview of weather prediction systems[1]. The inherent complexity of these tasks often leads to bespoke research solutions, which can be difficult to maintain or adapt for new applications. To address these challenges, this project aims to design and implement a reusable, modular system for managing the end-to-end pipeline of data processing, modeling, and visualization. This system is intended to provide a foundation that can be easily extended or adapted for diverse datasets and forecasting goals.

The project has two primary objectives. The first is to design and implement a reusable, modular system for managing the end-to-end pipeline of data processing, modeling, and visualization. Such modular systems have been highlighted as essential for scalable and flexible applications in big data contexts[2]. By implementing this architecture, the system can be tailored to a variety of forecasting contexts while maintaining consistency and maintainability. The second objective is to validate this architecture through a proof-of-concept implementation focused on weather forecasting. This implementation uses the same dataset as prior weather forecasting research, providing a direct comparison to previous methods and demonstrating the system's practical utility.

By balancing the development of a generalizable framework with a focused proof of concept, this project aims to contribute both a toolset for future forecasting applications and insights into best practices for building modular data systems. These contributions align with prior work emphasizing the importance of reusability and extensibility in software design for scientific research[3].

1.1 How to View the Dashboard

This paper is best when paired with its corresponding visualization dashboard that includes the most up to date results from this project as well as much of the information found within this paper. The dashboard can be viewed at http://datautopia.net.

1.2 Weather Forecasting

Weather forecasting plays a vital role in society, influencing decisions across sectors such as agriculture, transportation, disaster preparedness, and water resource management. Accurate forecasts empower individuals and organizations to mitigate risks associated with extreme weather events, optimize operations, and plan more effectively for future conditions[1, 4]. In particular, rainfall forecasting is critical for addressing challenges like flood prevention, drought management, and crop planning, as highlighted in studies on the socio-economic impact of accurate precipitation prediction[5]. By providing timely and reliable information, forecasting systems contribute to public safety, economic stability, and environmental sustainability[6].

Advances in weather forecasting increasingly depend on sophisticated data systems capable of processing large volumes of observational and modeled data[7, 2]. These systems must integrate diverse data sources, apply robust statistical or machine learning models, and deliver actionable insights in user-friendly formats. A modular and reusable architecture for weather forecasting data systems can enhance adaptability, scalability, and efficiency, making it easier to tackle the growing complexity of modern forecasting needs[3]. While this project emphasizes rainfall forecasting, the system it proposes is designed to generalize to other meteorological variables, supporting a wide range of forecasting applications[8].

1.3 Challenges in Weather Forecasting

Developing and comparing weather forecasting models is a complex process that involves multiple challenges, ranging from data quality issues to the need for rigorous model evalu-

ation. A key obstacle lies in the variability and sparseness of weather data. Observational datasets often contain gaps, inconsistencies, or biases due to limitations in measurement instruments or geographical coverage. These issues can hinder model training and reduce the reliability of forecasts, necessitating robust data preprocessing techniques to ensure that input data is clean and representative [9].

Another challenge is the selection of appropriate modeling techniques. Weather phenomena are influenced by a wide range of dynamic and interdependent factors, making it difficult to identify the right combination of models and features. Traditional statistical models, machine learning methods, and hybrid approaches all have strengths and weaknesses, depending on the type and scale of the forecasting problem. Furthermore, models must balance accuracy and interpretability, particularly in contexts where decisions based on forecasts can have significant consequences.

Model evaluation adds an additional layer of complexity. Forecasting models are often judged using a variety of metrics, such as mean absolute error (MAE), root mean square error (RMSE), and skill scores, but these metrics can yield conflicting conclusions about model performance. Additionally, different weather events or forecasting horizons may require tailored evaluation methods, making it challenging to compare models fairly across diverse scenarios. Establishing standardized benchmarks and testing models on consistent datasets is crucial for meaningful comparisons and for advancing the field [10].

These challenges highlight the importance of building flexible and modular systems for forecasting. Such systems can facilitate the integration of diverse datasets, support the experimentation and comparison of various modeling techniques, and provide tools for rigorous evaluation. By addressing these issues, developers and researchers can improve both the reliability and the applicability of weather forecasting models [9, 11].

1.4 Outcomes

This project presents a significant step forward in designing and implementing modular systems for weather forecasting. The key outcomes of this work include:

1. **Development of a Generalized Architecture:** A reusable, modular framework for weather forecasting data systems that emphasizes flexibility, scalability, and maintainability. This architecture can be easily adapted to different forecasting contexts, supporting diverse datasets and goals.

2. **Proof-of-Concept Implementation:** A demonstration of the architecture's functionality through a focused proof-of-concept in rainfall forecasting. This implementation serves as a direct comparison to prior methods, showcasing the system's potential for real-world applications.

3. **Enhanced Modularity:** By integrating components for data preprocessing, feature engineering, and visualization, the system allows for easy customization and extension, offering a flexible foundation for future research and applications in weather forecasting.

4. **Scalable and Maintainable Design:** The design prioritizes scalability, making it suitable for handling the increasing volume and complexity of weather data. It also facilitates maintenance, reducing the need for bespoke solutions in future applications.

5. **Improved Forecasting Accuracy:** Through the implementation of an ensemble of models and robust evaluation metrics, this project contributes to better performance in rainfall forecasting, as evidenced by the results compared to earlier methods.

2 Background

2.1 Prior Work

This project builds on a series of prior efforts focused on rainfall forecasting for Colombia, South America [12, 13]. Our earliest work utilized the CHIRPS (Climate Hazards Group InfraRed Precipitation with Station data) dataset, which provides high-resolution rainfall estimates derived from satellite observations and in situ measurements. This initial study applied relatively straightforward modeling techniques, yielding promising but limited results due to the constraints of both the dataset and the ad-hoc design of the forecasting system. One key challenge in this work was the difficulty in generating meaningful and standardized analysis metrics for comparing our results to those of other research efforts, a limitation that shaped subsequent projects.

Our more recent publication shifted to using the ERA-5 Land dataset, a reanalysis dataset that offers higher temporal and spatial resolution with improved data consistency. This transition marked an important step forward in both data quality and modeling approach. The study introduced more sophisticated modeling techniques, resulting in improved forecasting accuracy and greater insight into rainfall patterns over Colombia. However, like the earlier work, the system architecture was designed in an ad-hoc manner, which continued to limit scaleability and pose challenges for systematic evaluation and comparison.

These prior efforts provided valuable experience and insights, forming the foundation for the current project. This project seeks to address the limitations of earlier approaches by implementing a modular and reusable architecture that standardizes the forecasting pipeline and enables more robust evaluation. By maintaining the focus on the ERA-5 Land dataset and leveraging lessons learned from our previous work, this project aims to further advance rainfall forecasting for Colombia while improving the ability to generate and com-

pare performance metrics across studies.

2.2 Related Works

This section provides an overview of the foundational models and evaluation techniques employed in this study. As the focus of this paper lies primarily on the development of the infrastructure supporting weather forecasting, we include only a brief discussion of the models used in the ensemble. For a more comprehensive background on precipitation forecasting methodologies, readers are encouraged to refer to our prior research [12, 13].

2.2.1 Autoencoders

Autoencoders are a type of deep learning model designed to learn efficient representations of data by compressing and reconstructing it. Their architecture typically resembles an hourglass, where the input data is compressed into a smaller-dimensional latent space by the encoder, then reconstructed by the decoder. This characteristic structure makes them versatile for many data modalities, including images and text.[14]

Autoencoders are trained in an unsupervised manner, using the input sample as the target output. During training, the model minimizes the reconstruction error, effectively learning to encode the most relevant features of the input data. After training, the autoencoder is split into two standalone models: the encoder, which transforms input data into a latent representation, and the decoder, which reconstructs data from the latent space back into its original form.[14]

In our implementation, we trained a 2D convolutional variational autoencoder[16] for working with grayscale images to encode the gridded array of rainfall measurements that makes up our dataset. 2 shows some characteristics of a similar autoencoder that was

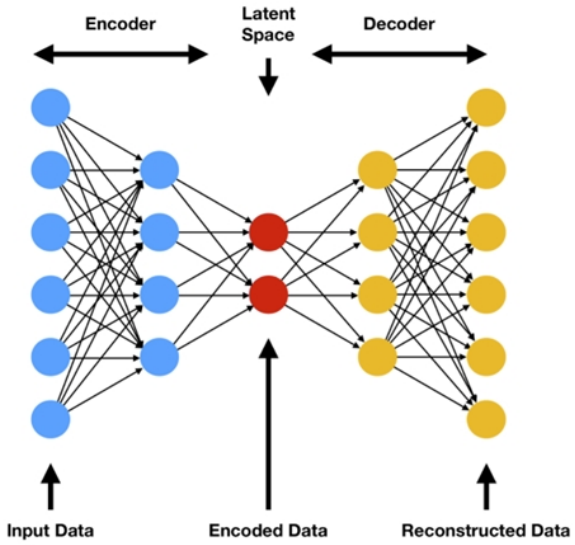
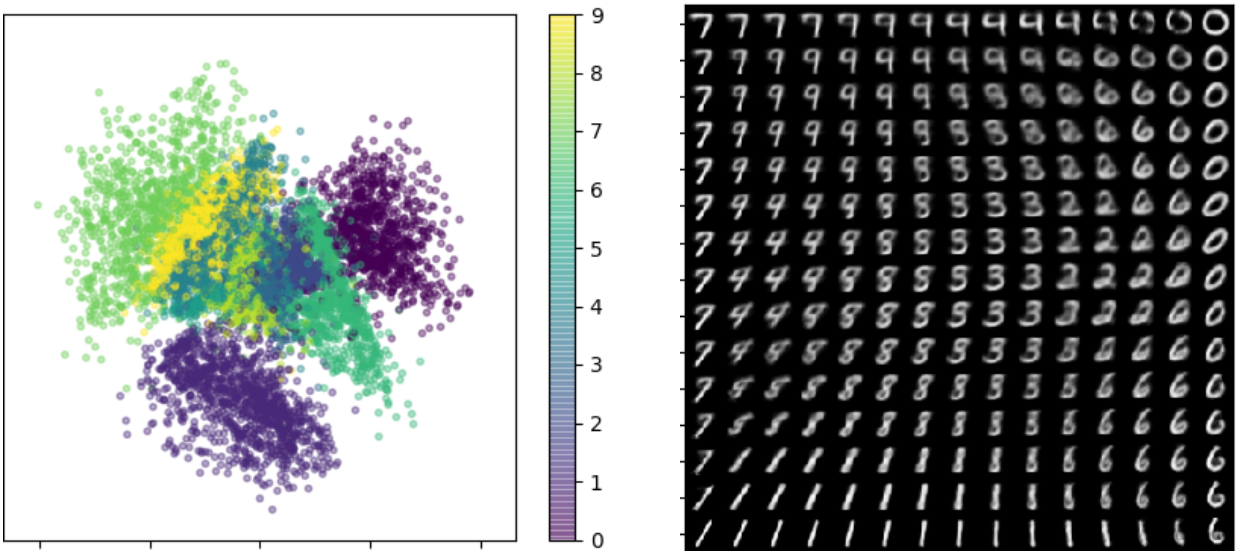


Figure 1: The distribution of samples in an autoencoder's latent space showing the learned class boundaries

trained on images of handwritten digits. In both, the encoder consists of multiple convolutional layers that progressively reduce the spatial dimensions while increasing the number of feature channels. This design allows the model to capture complex spatial patterns in the data, compressing the input into a compact latent representation. The decoder mirrors this process, employing transposed convolutional layers to reconstruct the input data from the latent space.[17]

The latent representation learned by our autoencoder is a vector, where each dimension corresponds to a feature learned from the input pixel space. These features are not only useful for reconstructing the input but also serve as a compressed representation that can be fed into subsequent models, such as our transformer. For example, interpolating between two latent vectors and decoding the results would produce a smooth transition between two weather patterns, illustrating how the model encodes and transforms spatial features.



(a) Latent space class boundaries

(b) Decoded samples from latent space

Figure 2: Graphics illustrating the workings of an image autoencoder's latent space.[15]

2.2.2 Transformers

Transformers[18] are a class of deep learning models well-suited for processing sequences, initially popularized for natural language processing but increasingly used in other domains. They leverage self-attention mechanisms to capture relationships between elements of a sequence, regardless of their distance, making them ideal for tasks involving temporal or sequential data.

In this project, the transformer was adapted from a translation model[19] and follows the standard encoder-decoder architecture. The encoder processes the input sequence, embedding it with positional information to preserve the order of elements. Through a series of self-attention layers, the encoder outputs a context representation that summarizes the sequence in a way that emphasizes relevant relationships within the data.

The decoder generates the target sequence step-by-step, starting with a predefined "start" token. At each step, the decoder combines the previously generated tokens with the

context from the encoder to predict the next token in the sequence. This process continues until the target sequence is complete. Positional encodings are applied to the decoder's input at every step to ensure that temporal ordering is preserved throughout the generation process.

In the future, it may be worth comparing this implementation to a decoder-only transformer to determine if that approach can achieve similar results in a more computationally efficient way.

3 Framework Architecture

The system is composed of a few different high level components, designed as a service oriented architecture. The services are organized in a monorepo, with a shared mysql database that warehouses the dataset, model outputs, and precalculated values as blob storage. Models are made using PyTorch, although the system would be fairly easy to integrate with other Python based machine learning frameworks. The visualization dashboard was written in Python using streamlit. All services that we built for this project were designed to have two parts, a generic framework, and a project specific implementation.

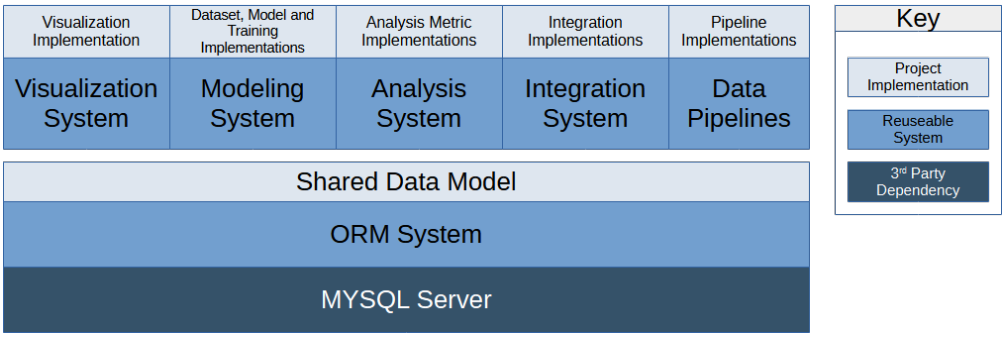


Figure 3: The layers of the framework

3.1 Data Layer

The data layer is an essential component of any software system using a Service-Oriented Architecture, acting as the bridge between services and the shared database. To build our system's data layer we used three primary components, a MYSQL database, an ORM system, and a shared data model. Our ORM system is defined within a framework called Datastore. Datastore is a open source library which we published separately from this project that is designed to simplify interactions with the database by wrapping SQLAlchemy, a popular ORM, and extending it with a set of basic yet practical functionalities. These include automating table creation, supporting password obfuscation, and providing a consistent structure for the shared data model. By addressing some common challenges in database

integration, Datastore helps streamline operations in SOA and microservice environments.

One of the main features of Datastore is its ability to automate table creation. This is achieved by utilizing SQLAlchemy's capabilities to generate and manage database schemas based on a shared data model. This shared model defines the schema in a way that aligns with Datastore's expectations, ensuring that tables and other shared objects are consistently implemented across services. While straightforward, this feature helps reduce repetitive coding tasks and supports consistency in database structure, which is particularly beneficial in distributed systems.

Another useful function provided by Datastore is password obfuscation. This feature adds a layer of security by handling password obfuscation before sensitive data reaches the database. Although it's not a comprehensive security solution, this functionality helps developers follow better practices and prevent shoulder surfing without requiring significant additional effort. By embedding this into the data layer, services can maintain a consistent approach to password handling, contributing to a more secure distributed system.

The shared data model is a central aspect of Datastore's design. It is defined for each distinct project that uses the datastore, but shared by all the services within that system. It utilizes a common schema definition that includes tables and DTOs. This approach ensures consistency in how data is represented and accessed across the system. In this project's implementation of the shared data model, many tables share the same overall design pattern which includes a indexed datetime column and a blob column that stores the array of values associated with each sample. This design pattern is deeply tied to the data pipeline framework which we developed for this weather forecasting system.

3.2 Services

Built upon the data layer are several services. These services rely on the shared data model defined in the data layer, but run independently of one another. That is not to say that they don't use the data output by another service though, just that they will not error out or generate incorrect output without another service being active. This aligns with the design goals of Service-Oriented Architectures, and helps the system maintain scaleability and robustness.

3.2.1 Data Pipelines

The system's backend relies on data calculation pipelines to transform and store precomputed data, significantly reducing visualization loading times and maintaining organization. However, this optimization comes with the cost of significantly increased database storage requirements. Much of the backend operates within this pipeline framework as a series of ETL (Extract, Transform, Load) processes, which extract data from one table, transform it, and store it in another. These pipelines use a modular framework: inputs and outputs are defined within the datastore's shared data model, while transformations are implemented via Python callback functions. Throughout the system numerous data pipelines are employed, with some being run as standalone services, and some being launched as small parts of a larger process.

3.2.2 Data Integration

The framework also includes support for 3rd party data integrations as well, although a project specific implementation to retrieve weather data was deemed unnecessary and cut from the project scope due to time constraints since we already had a copy of the dataset retrieved with an older integration for a prior project. However the system is more than

capable of implementing 3rd party integrations using a framework very similar in design to the data pipelines.

It should be able to support bulk, scheduled, and streaming integrations with the ability to synchronize the source data with the data that is already in the system and request only the input data that is needed. In the case of the integration we would have implemented with Copernicus Climate Data Store, it would be a scheduled integration that uses the date of the last data in our blob storage to determine what data to request.

3.2.3 Modeling

Once data has been ingested into blob storage and any needed transformations such as normalization have been completed, its time to implement an instance of the modeling service. The modeling service framework is heavily integrated with the machine learning framework PyTorch, although versions for other machine learning frameworks such as tensorflow or jax could be written in the future.

The framework includes two basic systems which are both extensions of PyTorch classes. The first is the dataset, which includes several classes that inherit from the PyTorch Dataset class. The most sophisticated of these dataset classes is the SynchronizedDataset, which synchronizes its local copy in file storage with the blob data in the MYSQL database upon initialization. This ensures that every training run is using up to date data from the database while taking advantage of the transfer speeds that local file storage offers. This system allows us to define a dataset for a model based on a SQL query, which offers a unique flexibility for datasets that can be grouped in many different ways by giving us access to the operators that SQL provides.

The second significant part of the model training framework is the Model class. It inherits from torch.nn.Module, and extends it with some commonly used features in a modular way. These features include automatic management of checkpoint files, real time logging with tensorboard, and a callback based training function that leaves the model creator full flexibility to customize the training loop without requiring them to write anything from scratch for a simple implementation.

After a model is trained to satisfaction, the model would typically be promoted to a production instance and a data pipeline used to inference the model across the entire input table. The output of this data pipeline is pushed into a new table that effectively contains precalculated values for the model's output. These values can then be accessed for display, analysis, training another model on them, or any other purpose.

3.2.4 Analysis

In addition to the unsophisticated metrics that are reported during model training, the framework also includes a much more comprehensive analysis service. The analysis service is based upon the core design pattern of the data pipeline, and extracts data from its source table in much the same way. An analysis service takes one or more tables as input, and iterates over the rows in it, calculating values from them using custom functions. Unlike the data pipeline which uses a callback function to do its transformation step, the data pipeline uses functions in an abstract base class to do its aggregation step.

The reason for this divergence is that the data pipeline only requires a single transformation function while the analysis service uses an add() function that operates on each row in the table and a finalize() function that is called after all rows have been iterated over. During this iteration metrics are aggregated and stored in the class, and in the finalize

function any final changes are applied such as dividing a sum by a list length to calculate an average. In the future the data pipeline may also be refactored to operate in a similar way, although for now its callback based pattern is more than capable of handling everything we have needed to do.

3.2.5 Visualization

Visualization of the data occurs in a web framework developed in Python using Streamlit[20]. It supports a range of data visualization tools, including Folium for mapping, Matplotlib for detailed plots, and Altair[21] for statistical and interactive visualizations. These tools provide a rich, flexible environment for exploring data and insights.

The visualization's architecture is loosely based on the model-view-controller design pattern to ensure modularity and maintainability. Each page's layout and interactivity are defined using declarative Python code representing the view. Shared logic and functionality, such as data transformations and common visualizations, are encapsulated in a central visualization_common module, serving as the controller layer. The database and ORM system define the model layer, enabling easy integration between the client and backend using the same common data model used by the other services.

This architecture allows for a clear separation of concerns, making it straightforward to extend or modify individual components without disrupting the overall functionality. By leveraging Streamlit's interactive capabilities and Python's visualization libraries, the client delivers a user-centric and reusable platform for data exploration and analysis.

4 Proof of Concept Implementation

To validate the modular architecture proposed in this project, we developed a proof-of-concept implementation focused on rainfall forecasting for the geographical region of Colombia, South America. It serves as a tangible demonstration of how the system can process, model, and visualize weather data using the same ERA-5 Land dataset employed in our previous work. By leveraging the reusable components of the architecture, this implementation aims to showcase the flexibility and scalability of the system, providing a clear example of how the modular approach can be adapted to different types of forecasting tasks. 4 shows the interactions between the different components of the system in the proof of concept implementation.

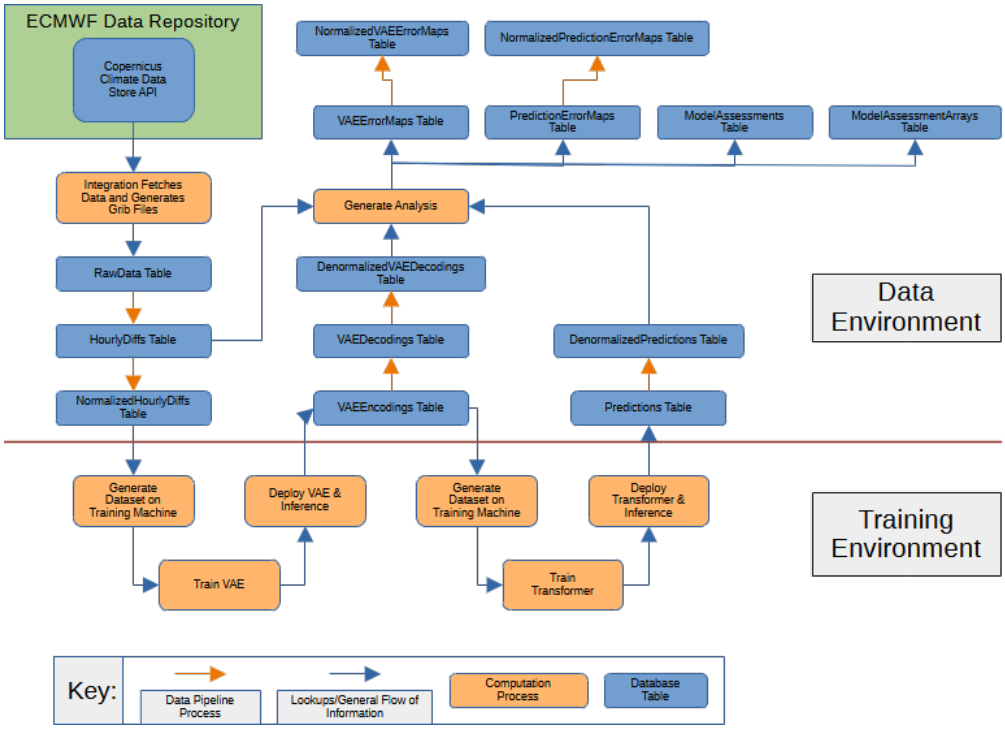


Figure 4: The flow of data through the system.

4.1 Dataset

The dataset for this project was derived from a subset of ECMWF's ERA5-Land dataset[8]. ERA5-Land is a global gridded weather dataset with a spatial resolution of 9 km^2 and an hourly temporal resolution. While ERA5-Land includes a wide range of weather-related metrics, this project focused solely on daily accumulated rainfall, which was de-accumulated to calculate hourly rainfall values.

For this project, we limited the spatial extent of the dataset to the region of South and Central America bounded by latitude/longitude coordinates (13, -80) and (-5, -62). The temporal range spanned from January 1, 1981, the start of ERA5-Land, to March 1, 2021. This subset of ERA5-Land data, used for model training and evaluation, consists of 331,998 2D arrays, each measuring 181x181 pixels and representing hourly rainfall in meters.

4.2 Model Ensemble

The forecasting ensemble has two primary components, a variational autoencoder (VAE) and an autoregressive transformer. The VAE embeds rainfall maps to single dimensional token vectors, and the transformer uses a series of tokens to predict the next one. The VAE and transformer are trained independently, and then combined into an ensemble for inference.

The VAE is trained first using the same value matrix for its input and target values with a combination of binary cross entropy and kullback-leibler divergence loss. Once trained, it is split into separate encoder and decoder networks so that it can be used as a tokenizer

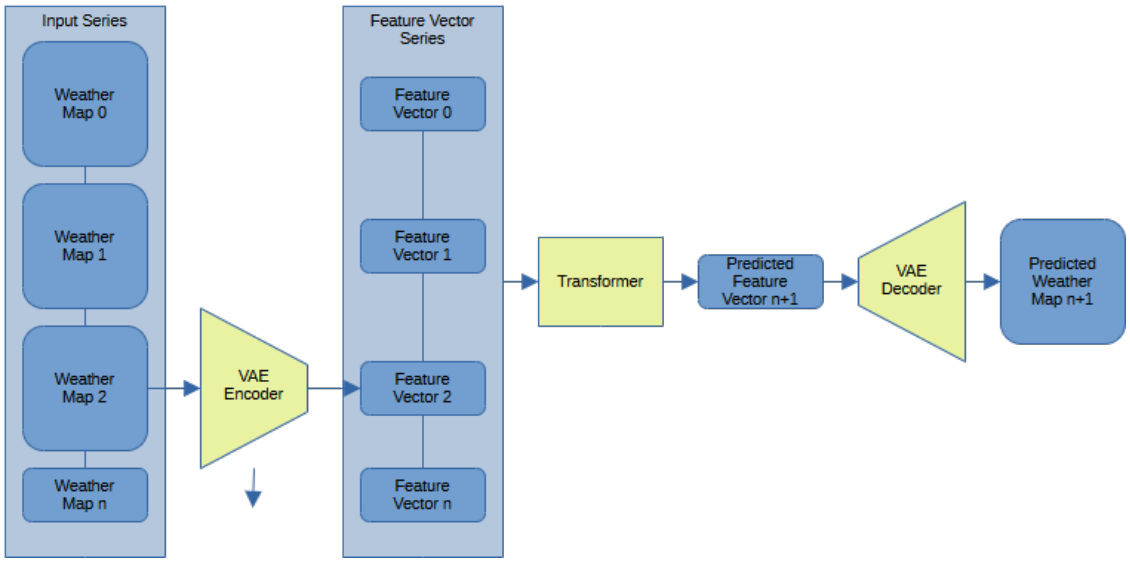


Figure 5: Diagram showing how the VAE is used to tokenize the input for the transformer

for the transformer. The VAE's encoder is inferenced on the entire dataset and the output is stored in a new dataset. This operation effectively generates a new copy of the dataset where each sample is now a vector of length 500 instead of the original 181*181 matrix. Next, the transformer is trained on this new dataset of tokens generated by the VAE encoder. The transformer is an autoregressive model, taking a series of 10 tokens as input, and predicting the 11th token in the sequence. The transformer is trained using the mean squared error as its loss.

After training, the models are saved and promoted from training to production. The ensemble is inferenced on the entire dataset, and the results are stored in the database.

4.3 Methodology

The primary objective of the proof-of-concept implementation was to demonstrate the functionality and flexibility of the proposed modular system for rainfall forecasting, rather than conducting a comprehensive study with in-depth model optimization. As such, the fo-

cus was not on exhaustive hyperparameter tuning or fine-tuning the models to achieve their peak performance. Instead, the models were trained to a satisfactory state that allowed for meaningful evaluation and analysis. This approach ensured that we could demonstrate the system's capability to handle data, perform modeling tasks, and facilitate model analysis, while leaving more detailed optimization for future iterations.

For model evaluation, we implemented a series of performance metrics to assess the quality of the forecasts produced by the system. All metrics were calculated in SI units to ensure consistency and standardization. The key metrics included:

1. Mean Absolute Error: A measure of the average magnitude of errors between predicted and observed values, without considering their direction. It provides a clear indication of overall model accuracy.

2. Remedian Absolute Error: Similar to mean absolute error, but uses the remedian[22] instead of the mean, providing a more robust measure in the presence of outliers.

3. Mean Error: The average difference between predicted and actual values. This metric can reveal any systemic bias in the model predictions.

4. Maximum Error: The largest difference between predicted and actual values, highlighting extreme discrepancies in the forecast.

5. Minimum Error: The smallest difference between predicted and actual values, identifying areas where the model performed best.

6. Remedian Error: Like mean error but calculated using the remedian[22], it gives a sense of central tendency for the errors while reducing the impact of extreme values.

7. Mean Standard Deviation: A measure of the variability of forecasted values, offering insight into the confidence of the model's predictions.

These metrics were implemented within the modular analysis service of the system and calculated across different data splits, including the training and test datasets, as well as the entire dataset. The results were used to gauge the model's performance and identify areas for potential improvement. Further details on these metrics, including both informal descriptions and formal definitions, as well as their specific role in weather forecasting, are provided on the data dashboard for easy reference.

In addition to these global metrics, we also calculated pixel-wise metrics to analyze spatial patterns in the forecast. These metrics were visualized as maps showing temporal aggregates, allowing us to identify spatially localized phenomena, such as regional weather patterns or areas with particularly high forecast errors. This spatial analysis is crucial in weather forecasting, as it enables the identification of "hot spots" where predictions may need further refinement.

While these metrics provide a comprehensive overview of the model's performance, there are additional metrics we plan to include in future studies. Notably, we aim to visualize spatial aggregates over time, which would provide insights into the evolving patterns of forecast accuracy. Furthermore, we intend to implement categorical metrics by converting continuous forecast values into binary predictions based on a specific threshold. These metrics include Heidke Skill Score, F1 score, precision, and recall, which are commonly used in classification tasks and help evaluate the model's ability to predict specific weather events (e.g., rainfall occurrence) accurately. These additions will further enrich the analysis and offer a more complete view of the model's forecasting capabilities.

4.4 Results

The results of the proof of concept implementation demonstrate competitive performance in several key metrics, though some limitations and areas for improvement remain. The Mean Absolute Error for the model was found to be 0.23 mm/hour, which is comparable to other models in its class. However, it is important to note that this metric may not provide a fully accurate comparison due to differences in the datasets used, particularly in terms of geographic coverage and data resolution.

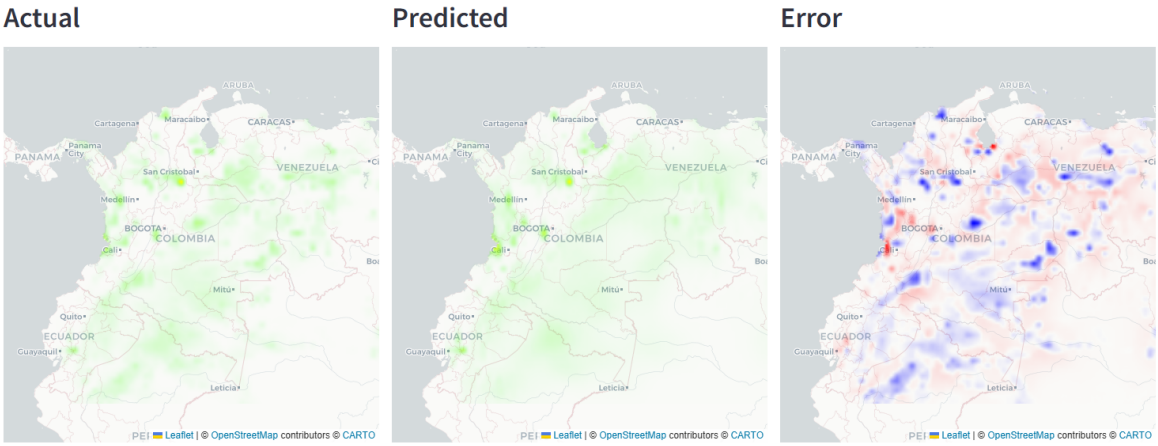
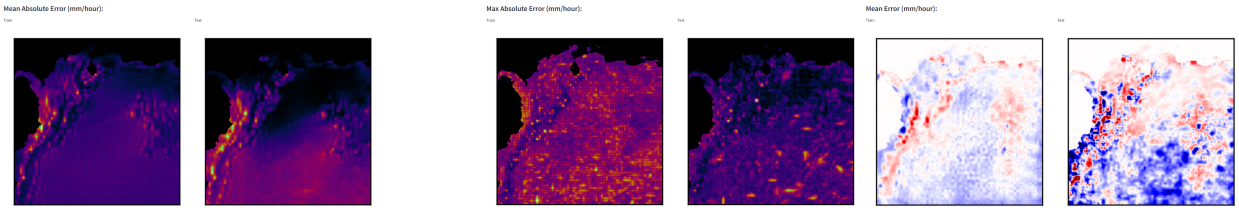


Figure 6: A prediction from the dataset along with its spatial error distribution. Red indicates a bias towards overprediction and blue indicates a bias toward underprediction

The test set used for evaluation may have been too small to be fully representative of the model's performance across diverse weather conditions, potentially introducing biases in the results. The analysis of model bias revealed some interesting insights in both the test and training sets. The mean error was biased toward underprediction, while median error was biased toward overprediction, suggesting that the model most often overpredicted rainfall, but tended to underpredict more severely in certain cases.

A more detailed look at the error distribution revealed that approximately 99.7% of measured error values had a magnitude less than 1.4 mm/hour, suggesting that the model performed well for the majority of forecasts, with most predictions falling within a relatively narrow error range. However, the most inaccurately forecasted value over the entire dataset

had an error of 40.5 mm/hour, highlighting the presence of extreme outliers in the data that were more difficult for the model to predict accurately. This extreme value suggests the potential for further refinement, particularly in handling extreme weather events or outlier conditions.



(a) Mean Absolute Error

(b) Max Absolute Error

(c) Mean Error (Red indicates a bias towards overprediction and blue indicates a bias toward underprediction)

Figure 7: Temporally aggregated error metrics calculated pixelwise

Overall, these results suggest that while the system demonstrates reasonable performance and accuracy, there are areas where model improvement is possible, particularly in terms of handling extreme events and reducing prediction biases. For detailed results, please refer to the dashboard's model analysis and result set pages.

5 Conclusion

In conclusion, this paper introduces a modular, reusable architecture for weather forecasting systems that facilitates efficient data processing, model development, and visualization. The proof-of-concept implementation demonstrates the system's utility in generating rainfall forecasts using the ERA-5 Land dataset, with promising results in model accuracy despite some limitations in the methodology. The study highlights the importance of systematizing the forecasting pipeline to facilitate future research and improve model evaluation. Although aspects of model optimization were not fully explored, the results suggest that the framework can be adapted to other meteorological variables and can serve as a foundation for more sophisticated forecasting models. Future work will focus on refining the architecture, incorporating additional error metrics, and expanding the system's capabilities for real-world applications.

6 Future Work

The significant focus on infrastructure during this project has opened up a plethora of avenues for future work. Much of this system is general purpose and intended to be reused, so significant effort will be put into optimizing it during the course of future projects. Listed below are a few of the many goals we have in mind for this system:

1. Extract reusable code into standalone libraries
2. Set up the hosting cluster to use MAAS, LXC, and Ray
3. Optimized speed and parallelization of data pipelines
4. Implement high quality 3rd party integration
5. More metrics like Heidke Skill Score, Mean Squared Error, Root Mean Squared Error
6. Inferencing and visualization for output sequences longer than 1
7. Incorporate vector databases

References

[1] P. Bauer, A. Thorpe, and G. Brunet, "The quiet revolution of numerical weather prediction," *Nature*, vol. 525, pp. 47–55, 2015.

[2] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, no. 2, pp. 137–144, 2015.

[3] G. Wilson and et al., "Best practices for scientific computing," *PLoS biology*, vol. 12, no. 1, p. e1001745, 2014.

[4] A. H. Murphy, "What is a good forecast? an essay on the nature of goodness in weather forecasting," *Weather and Forecasting*, vol. 8, no. 2, pp. 281–293, 1993.

[5] S. D. Changnon, "Measures of economic impacts of weather extremes," *Bulletin of the American Meteorological Society*, vol. 84, no. 9, p. 1231–1236, Sep 2003.

[6] R. A. Pielke, "Future economic damage from tropical cyclones: Sensitivities to societal and climate changes," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 365, no. 1860, p. 2717–2729, Jul 2007.

[7] M. Ghil, M. R. Allen, M. D. Dettinger, K. Ide, D. Kondrashov, M. E. Mann, A. W. Robertson, A. Saunders, Y. Tian, F. Varadi, and et al., "Advanced spectral methods for climatic time series," *Reviews of Geophysics*, vol. 40, no. 1, Feb 2002.

[8] S. Muñoz, "Era5-land hourly data from 1981 to present." [Online]. Available: https://confluence.ecmwf.int/display/CKB/ERA5-Land%3A+data+documentation

[9] M. Taillardat, J. Van den Bergh, B. Van Schaeybroeck, K. Whan, and J. Ylhaisi, "Statistical postprocessing for weather forecasts: Review, challenges, and avenues in a big data world," *Bulletin of the American Meteorological Society*, vol. 102, no. 3, pp. E681–E699, 2021.

[10] S. Rasp, P. D. Dueben, S. Scher, J. A. Weyn, S. Mouatadid, and N. Thuerey, "Weatherbench: A benchmark data set for data-driven weather forecasting," *Journal of Advances in Modeling Earth Systems*, vol. 12, no. 11, Nov 2020.

[11] Y. Wu and W. Xue, "Data-driven weather forecasting and climate modeling from the perspective of development," *Atmosphere*, vol. 15, no. 6, p. 689, 2024.

[12] E. Neill and G. Dogan, "Rainfall forecasting with variational autoencoders and lstms," *2023 6th International Conference on Information and Computer Technologies (ICICT)*, p. 35–40, Mar 2023.

[13] E. Neill, J. Edwards, T. Reitz, E. Cook, G. Dogan, and N. Pricope, "Short term prediction of rainfall in columbia using a generative modeling approach," *2023 Congress in Computer Science, Computer Engineering, and Applied Computing (CSCE)*, p. 332–336, Jul 2023.

[14] S. Flores, "Variational autoencoders are beautiful." [Online]. Available: https://www.compthree.com/blog/autoencoder/

[15] L. Tiao, "A tutorial on variational autoencoders with a concise keras implementation," Oct 2022. [Online]. Available: https://tiao.io/post/tutorial-on-variational-autoencoders-with-a-concise-keras-implementation/

[16] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," Dec 2022. [Online]. Available: https://arxiv.org/abs/1312.6114

[17] D. Foster and K. Friston, *Generative deep learning: Teaching machines to paint, write, compose, and play.* O'Reilly Media, Incorporated, 2023.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," Aug 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[19] A. Sarkar, "Build your own transformer from scratch using pytorch," Apr 2023. [Online]. Available: https://towardsdatascience.com/build-your-own-transformer-from-scratch-using-pytorch-84c850470dcb

[20] I. Streamlit, "Streamlit: The fastest way to build and share data apps," 2019. [Online]. Available: https://streamlit.io/

[21] J. VanderPlas, B. Granger, J. Heer, D. Moritz, K. Wongsuphasawat, A. Satyanarayan, E. Lees, I. Timofeev, B. Welsh, and S. Sievert, "Altair: Interactive statistical visualizations for python," *Journal of open source software*, vol. 3, no. 32, p. 1057, 2018.

[22] P. J. Rousseeuw and G. W. Bassett, "The remedian: A robust averaging method for large data sets," *Journal of the American Statistical Association*, vol. 85, no. 409, p. 97–104, Mar 1990.