GraphCast 2.0: Enhancing Performance and Precision in GraphCast

Abhijay Chelli, Ansh Aggarwal

IST DS340W, Penn State
201 Old Main, University Park, PA, 16802, United States
afc6135@psu.edu
aza6467@psu.edu

***Abstract*— This paper presents an improved and modernized approach to Graphcast. Graphcast is a machine learning based medium weather prediction model with a goal of predicting 10 day singular weather predictions in under a minute with high accuracy. This model competes with current numerical-based weather prediction models, and it needs to be faster and more efficient at prediction than the current model in usage. While the original Graphcast model delivered quite strong results with the current dataset from 2022 however its performance in its Root Mean Square Error(RMSE) and its Loss values could've been improved. These two metrics(RMSE and Loss) are factors that indirectly show a model's accuracy to its testing data. The primary focus of Graphcast 2.0(an improved version of Graphcast) is to improve these metrics within the model by using tested methods. Graphcast 2.0 incorporates data-specific feature selection, lead time optimization, and idea-pulling from both Gencast and PanguWeather. By using these various techniques in Graphcast 2.0, this model will be able to have a much better RMSE and Loss value to accurately represent the model that can produce medium ranged weather predictions. The expected outcome of these enhancements is to provide a more accurate and efficient alternative to traditional models, offering significant improvements in both forecasting accuracy and usability.**

***Keywords*— Machine Learning, ML-Based Weather Prediction, Model Optimization, Graphcast, Model Metrics(RMSE/Loss)**

I. Introduction

Weather prediction(in all of its forms) is something used on an hourly and daily basis by individuals and companies alike. The uses of weather prediction stem from students checking their weather for the day, to large companies planning shipping routes months in advance to take advantage of winds or currents. The uses for weather prediction are many, and it is what keeps the world running(and goods moving) while being in the shadows of how one lives.

The current models which are used are numerical-based weather prediction models. These models include examples such as the ECMWF(European Centre for medium-range weather forecasts), GFS(Global Forecasting system), and ICON(Icosahedral NonHydrostatic model). These are just a few examples of numerical-based weather prediction models currently used across the globe. Each of these models collects sensor data from a wide variety of sources, which is then processed through complex statistical formulas and fed into supercomputers to generate weather prediction outputs, such as graphs and visuals. These outputs are typically interpreted by human meteorologists, who verify the readings and make adjustments if necessary. However, this process is both expensive and time-consuming. Additionally, despite the vast amounts of data collected daily by these sensors, much of it remains unused, resulting in decades' worth of stored but untapped information.

However recently a new method for weather prediction became available to use: ML-based weather prediction models. These models are much more simple(yet advanced) compared to their numerical weather prediction counterparts. Machine Learning based weather prediction models use previous stored weather data along with complex machine learning models(such as Graphical neural networks) to properly predict weather for the next 10/15 days in a short period of time.

These machine learning based weather prediction models are cheaper to run and use, since they don't need the highly complex and expensive supercomputers which currently run the majority of the world's weather prediction. These machine learning based weather prediction models also use the unused data which has been stored since the 1940's. This data is highly detailed with specifics down to pressure data at a specific hour in a specific location.

The specific model which this paper will delve into will be the Graphcast model made in late 2022. This is a graphical neural network model with a multi-mesh representation which predicts 10 singular medium range forecasts in under 1 minute with high accuracy. The multi-mesh representation is used to properly represent the specific detail of the earth instead of in a rectangular graph format. This improvement in speed and accuracy in high-impact weather situations shows the power of a machine learning based weather prediction model.

However, this model is not without its flaws. While the model itself performs well, the data it uses holds it back and leads to lower accuracy than it should achieve. The dataset being used by Graphcast is the ERA5 dataset which is a large dataset which contains all weather data from the 1940's in a specific GRIB format, however this model uses data from 1979's till 2017. This dataset(when converted to CSV format) shows issues in the data itself, from insufficient columns to features that don't seem important to the weather prediction ability of Graphcast. By having an improper dataset, Graphcast's RMSE(Root mean square error) and loss values are much higher than they should be for a weather prediction model of this caliber.

The goal for Graphcast 2.0 is to resolve those issues which lower the RMSE and loss values(and in turn accuracy) by making changes to the datasets use for testing and training, by making changes to weather prediction lead times(the time between a prediction and when it actually happens), and techniques learned from both PanguWeather and Gencast(two other weather prediction models). By making these changes, Graphcast 2.0 should overcome the metric issues which the original model faced.

These changes begin with a proper feature selection with the large dataset. By using R, a simple but effective feature selection will be used in order to properly see which features have the most impact on the model. Those features will then be highlighted in the Graphcast model to be used for the testing and training process in order to improve RMSE and loss. The next step is to experiment with lead times, finding the right mix between both long term forecasting but a higher rmse/loss value set. Finally we'll look into how we can incorporate various aspects of both Gencast and Pangu weather in order to improve our accuracy and our evaluation metrics.

This research paper is structured in a way that the process of improving Graphcast is highlighted. The paper will begin with a simple abstract, then delve into a proper introduction to what ML-based weather prediction is and our plans for Graphcast 2.0. The next part would be our literature review which dives into our 3 research papers that helped develop this paper. We'll then move onto our code changes and methodology. Finally we'll have our results and our conclusion with our references at the end.

This paper will take a deep dive into how we are able to improve Graphcast and make it a much more accurate functioning model, one that could possibly compete with the current numerical weather prediction models in use right now.

II. Methodology

To enhance the performance of the original, unchanged GraphCast model, both feature selection and lead time could be used for the purpose of finding out how input variables and time intervals can directly influence accuracy within predictions and their accuracy, using RMSE and Loss.

When specifically looking at Feature Selection, XGBoost library became useful, as it allowed us to incorporate the XGBRegressor model. This model had a specific role in terms of finding out what the least to most important attributes were. The main attribute that was targeted for this was 2m_temperature. This made us evaluate for which variables had the most important effect, ultimately lessening gaussian noise and leading to a faster training of models.

We also tested and experimented on RMSE and Loss values based on lead time optimization, overall looking at how accurate the model can be. These lead times were 5 hours, 10 hours, and 15 hours, respectively, with 10 hours being our baseline. RMSE values were ultimately used to measure the difference between the predicted and actual values. Loss values were seen if the model was able to learn well during the training phase, which feature selection was able to support.

III. Literature Review

To give a surface context once again on this topic, Machine Learning based weather forecasting can be defined as computers being able to learn certain patterns based on past weather data in order to predict weather in the future. Past models were using traditional methods, such as predicting weather using physics equations. Machine Learning wants to specifically find out how past weather data relates to future weather data. It looks at specific weather events and metrics, like rainfall, temperature, wind, and more to be able to look at patterns. This allows for a faster approach to reach correct predictions, when looking at short term and long term predicitions. Overall the goal of this literature review is to look at the recent comings of weather prediction that incorporate machine learning methods to better predict weather, specifically for forecast time in days.

There are usually two types of models introduced within the span of these papers - traditional models that take on NWP systems like IFS(Integrated Forecasting System) and ENS(Ensemble Prediction System) and MLWP models (Machine Learning based Weather Prediction Models). NWP has been seen throughout and known for a long time, and specifically IFS and ENS have, developed by European Centre for Medium Range Weather Forecasts, are being used world-wide. As mentioned previously, the IFS and ENS models rely on solving physics equations, as based on a model of Earth's atmosphere. This is able to produce very accurate results of forecasts throughout multiple variables. Due to the nature of these traditional models, these models tend to output an expensive computational cost, due to energy usage, long runtimes, and use of supercomputers. This is when the need for MLWP comes in, as it helps to solve the issues of high costs.

Looking at the first reading, *Learning skillful medium-range global weather forecasting*, which gives an overall view of GraphCast for the overall research project, a lot of things can be discussed regarding this model. GraphCast is a model which uses a GNN-Based architecture with a encoder-processor-decoder format, as it was also trained on almost four decades of the ERA5 dataset. The notable thing about this is that it is able to generate ten day forecasts, with a resolution of 0.25°. These ten day forecasts would then be able to be processed within less than a minute.

If we jump into why the GraphCast model is a lot more effective and produces better results compared to HRES (ECMWP's high-resolution deterministic model) on 90% of 1380 verification targets, while also incorporating surface and atmospheric variables. The way the GraphCast model works, to create full ten day forecasts, is that the model is rolled out over various multiple time steps, since the model is autoregressive.

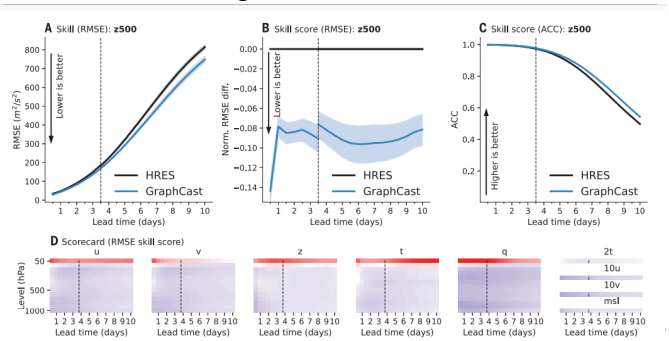

Fig. 1 This figure backs up the claim in regards to GraphCast outperforming HRES on 90% of 1,380 targets. This is due to GraphCast having a low RMSE across several lead times in A's Graph, the consistency of better performance over time in Graph B, along with strong pattern matching in Graph C, as well as the improved accuracy through various weather variables and atmospheric levels as seen in the graph of D.

When looking at the second paper, *Probabilistic Weather Forecasting With Machine Learning*, it gives an overall view of the GenCast ML based model, as it uses a generative diffusion based model. Several more ideas can be more discussed regarding this advanced ML model. Compared to GraphCast producing one single outcome, GenCast uses an ensemble approach. This is essentially capturing several multiple future outcomes of the atmosphere. The notable feature about this is that it tells researchers the uncertainty of the forecast. The GenCast model can be seen as important, especially for tracking very important weather based scenarios, as scaled from high importance to low importance.

Looking at the specific results for GenCast, it is able to be found that GenCast was able to predict weather for up to 15 days in just under 8 minutes. It also found that it performed better than the highly

regarded European ENS model, of 97.2% of evaluated targets. Why it is preferred in some methods today is because of its ability to show the wide range of various weather outcomes, as well as how confident it can be. GenCast can also be very useful for predicting common weather events, such as storms. Looking at the specific diffusion design of GenCast, while also blocking out all noise, allows it to be one of the best advanced weather-based models today.

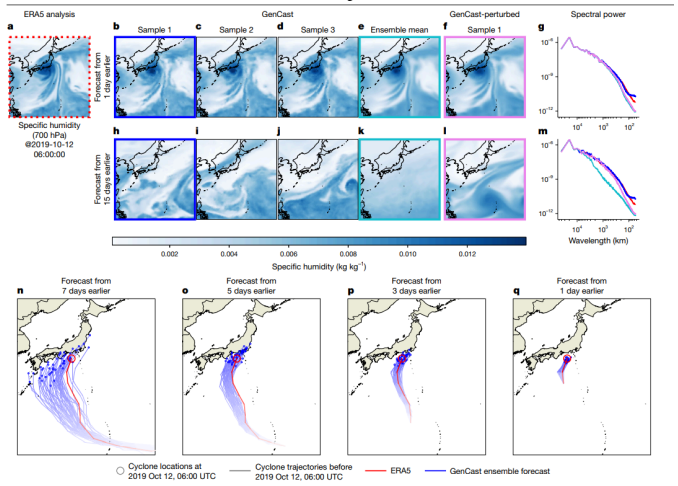

Fig. 2 The figure is able to specifically guide how it generates various realistic forecasts, as it has the ability to look at uncertainty and fine detail up to 15 days in advance. It specifically exhibits GenCast's ability to correctly predict the path of a specific storm that the figure looks at - Typhoon Hagibis. It shows the spread of possible outcomes which narrows closer to that weather event.

Finally looking at the third research paper, *The Rise of Data-Driven Weather Forecasting*, it can be seen focusing on PanguWeather, which is a ML model that is vision-transformer based. While GenCast compares itself to ENS, PanguWeather is compared to IFS on the same conditions from before. Results for PanguWeather show that it exceeds IFS accuracy on variables like temperature on the short lead times. However, there are some drawbacks within the PanguWeather as ML models introduce smooth outputs, bias increasing constantly, and also underperforms for forecasting tropical cyclone intensity. Forecast reasoning can also be important at times as several concerns for interpretability are introduced. This paper is important as it gives context to GraphCast and GenCast in several factors like trust and readiness in ML based systems for deploying these.

IV. Novelty

Primarily in the research and the pure development with the improved model of GraphCast, GraphCast 2.0, we were able to discover that this new and improved model is able to display its improvement from its performance and accuracy, based on the original GraphCast model. Even though the GraphCast Model showed strong results, it lacked in areas and values like RMSE and loss, as they were higher than wanted. However, this is what the GraphCast 2.0 model did really well. So overall, our work is to optimize these results in such a way that allows for an accurate and well-working weather prediction model.

Our goal in having a different approach, as compared to the original parent paper, was based on two specific ideas. The first idea touched on feature selection for improving the training part of our research idea. Specifically, we wanted to apply data-specific feature selection within the ERA5 Dataset. When looking at how the original model dealt with this, they didn't really take advantage of improving these features when training. This led to inefficiencies within the research, which was surprisingly not that much of an issue as we also did the same. We found that it didn't really show a profound impact on the performance of the model. This specific finding could be due to the fact that the data was already well-prepared for the purpose of medium-range weather predictions. In fact, we still continued with tested feature selection and it didn't improve the values of RMSE or loss, as mentioned before, but still allowed the model to focus on what was of the utmost importance for the data during the training.

The second idea touched on the specific enhancement of optimizing the lead time. We felt that GraphCast was not able to fully support lead time duration and model accuracy at the same time. We experimented with multiple lead times to be able to find the close-to-perfect of long-term

forecasting and accuracy of predictions. There were three different lead times that were tested on. These were 5 hours, 10 hours, and 12 hours, respectively, with 10 hours being our baseline. This baseline was to be able to be used as a reference point, telling us if the performance is short (5 hours) or longer (12 hours). We were able to find that the lower lead times correlated with lower RMSE values and loss value, allowing for accurate predictions. When the lead times increased, it was found that RMSE and loss values were rising, telling us that lead times were one of the most important factors for determining how well the model performed. This allows for a better optimization of GraphCast 2.0.

When looking at how these improvements relate within the data science pipeline, we can see that it fits in with the modeling phase. Moreso, the original model really focuses on collecting, cleaning, and visualizing the data, whereas our focus relies heavily on using those techniques to be able to overall improve the performance of the model, giving us accurate weather predictions on GraphCast 2.0. GraphCast 2.0 is a more cost effective method to help compete with existing models today, and also present itself as a preferred model.

V. Implementation

The process of taking our ideas and turning them into actual code and experimentation was not as simple as we had initially planned. The original goal for our implementation was to try out a three-part system that would help lower the RMSE and loss values of the Graphcast 2.0 model. However, due to the memory limitations of Google Colaboratory and the overall time we had to work with, we decided to focus on two parts and leave the third for future work. The two ideas that we moved forward with were *feature selection* and *lead time experimentation*.

The first step in our implementation was feature selection. As we looked into the Graphcast model and the way it was set up, we noticed that there was no proper feature selection being done anywhere. For a dataset with this many variables, the lack of even basic feature filtering was a clear limitation. With limited features available in our dataset, we decided to use an XGBoost-based feature selection approach to figure out which variables were actually impacting our main target — 2m temperature. At first, we had planned to run the feature selection in R, since that platform can sometimes handle memory better. But to keep everything consistent and easier to manage later, we decided to stick with our colab code and optimize it slightly to access more memory.

In the code itself, we imported the necessary Python packages like XGBoost, Pandas, and Scikit-learn. We then loaded our CSV file (WeatherDataP), which was converted from the GRIB format, and ran an 80/20 split for training and testing. Our first version of the XGBoost feature selection automatically pulled the top features based on importance values related to 2m temperature. However, we wanted to make sure nothing was missed due to formatting or naming issues, so we ran a second version where we listed each feature name manually. Both versions gave us very similar results, which gave us confidence in the selection. To go a step further, we also generated a correlation matrix to see if any features were highly correlated to each other, which could affect performance.

The second part of our implementation was experimenting with different lead times. The default lead time in the Graphcast model was 10 hours, but we wanted to test how changing this would impact RMSE and loss. Originally, we planned to test 5 hours (short), 10 hours (baseline), and 15 hours (long). The tests for 5 and 10 hours worked without issue, but when we tried 15 hours, we ran into errors. After some time troubleshooting, we found that the model can only handle up to 12 hours of lead time. So we changed our highest value to 12

hours and reran the experiments. These tests were done by adjusting the values inside the "extract training and eval data" section of the model, specifically modifying two lines of code that control the slice range for lead times. We ran each test twice to double-check our results, and all three (5h, 10h, 12h) completed successfully within Colab's memory limits.

Overall, our implementation was centered around two main strategies: performing a feature selection to clean and prioritize our input variables, and experimenting with different lead times to see which range gave the most accurate predictions. While we initially set out to try three ideas, the two we moved forward with gave us useful insights and results — and the third idea, involving spatial resolution, will be saved for a future extension of this project.

VI. Results, Analysis, and Discussions

After following the specific instructions and processes set in the *Implementation* section(IV), the results can be split into three sections each following the two steps from the novelty process. See Figure 3 for proper steps.

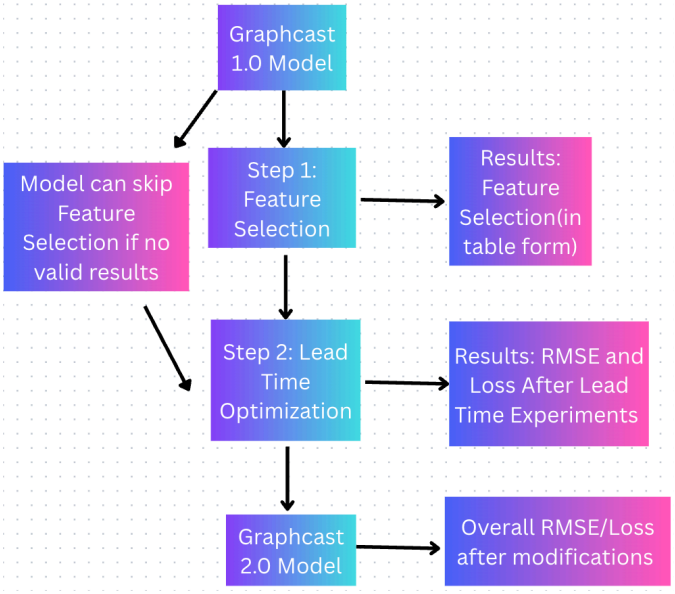

Fig. 3: Experimental flow for Graphcast 2.0 model pipeline. The diagram outlines the two-step improvement process starting from the original Graphcast 1.0 model. It includes conditional handling of feature selection, lead time optimization, and highlights where intermediate and final performance results are gathered.

Following those steps in Fig. 3 show that each step in the process from Graphcast 1.0 to Graphcast 2.0 show results for the feature selection, the lead time experimentation, but also that the feature selection step can be skipped if the results are not valid or useful to improving the model RMSE and Loss.

The first result in the steps to reaching Graphcast 2.0 from Graphcast 1.0 is to see what the double feature selection implementation shows us. Each feature selection within this model does the same thing on the same dataset(WeatherDataP) however the two differ in the way they pull the column headings/variable names(more detail in IV - Implementation). The first XGBoost feature selection automatically pulled all the variable names and had the main variable set as 2m_temperature. This initial feature selection outputs its results in a tabular format(Figure 4) and shows the importance ranked from most important to least important in relation to 2m_temperature.

```
batch_temp: 0.0
lat: 0.0
lon: 0.0
batch_pressure: 0.0
mean_sea_level_pressure: 0.0
batch_x: 0.0
level_x: 0.0
specific_humidity: 0.0
batch_y: 0.0
level_y: 0.0
geopotential: 0.0
```

Fig. 4: Output of the feature selection step, presented in table form. In this instance, all evaluated features returned an importance score of 0.0, indicating that no features were selected as significantly impactful. As outlined in the experimental flow, the model is configured to skip this step when no valid results are produced.

Fig. 4 shows the output of the first feature selection. Every parameter in the dataset (WeatherDataP) outputs a importance of 0.0 in relation to 2m_temperature. The reason for this result can be explained due to the data itself. The dataset(when taken in detail) shows that there isn't

much variability at all between data points. This is because the sensors are recording the data per different level of pressure, however the readings can be the same even with various pressure levels. This is why the feature selection wasn't able to find importance between the variables just due to how close they are together.

The next step was running the feature selection again however with manual inputs of the variable names instead of the machine pulling the variable names from the csv file. However this result is also the same as Figure 4 that shows the importance of all the variables in relation to 2m_temperature still at 0.0.

Step 2 in this process(refer to Fig. 3) is the lead time optimization and how that impacts our RMSE and Loss values for the Graphcast model. Lead times are explained above, however simply put, they are just the time between the forecast and when the forecast was made. Graphcast 2.0 attempted 3 different values for lead time in the range between 5-12 hours.

The initial lead-time base value was 10 hours, and Graphcast 2.0 outputted a standard RMSE and Loss Value of 4.54 and 20.625 respectively and is shown in Fig. 5.

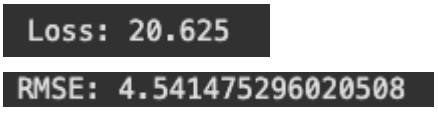

Fig. 5: Output of the model after training with a baseline lead time of 10 hours. The loss value is reported as 20.625 and RMSE as 4.54, establishing a reference point for performance comparison against shorter and longer lead times. This baseline result was used to evaluate the relative impact of lead time optimization on model accuracy.

The next value down the line compared was 5 hours lead time, this time is closer to the lower end of the lead time range. This is to see how a lower lead time impacts RMSE and Loss and how it could change how Graphcast 2.0 operates. After running Graphcast 2.0 with the lead time of 5 hours, RMSE and Loss did drop significantly from the baseline of 10 hours of lead time and is shown in Fig. 6 below. RMSE of 5 hours was 3.64 and the loss was 13.25.

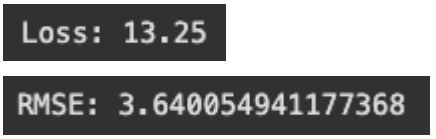

Fig. 6: Output of the model after training with an experimental lead time of 5 hours. The loss value is reported as 13.25 and RMSE as 3.64, establishing a reference point for performance comparison against baseline and longer lead times.

We can see based on Fig. 6 that Loss and RMSE both improve with a lower lead time compared to the longer baseline of 10 hours. Now to compare this to a longer lead time, picking 12 hours of lead time. This is the maximum lead time based on the range of the model in Graphcast 1.0. By testing the top of the range of the lead time possibilities, this can test if there is a true loss or increase in RMSE/loss, or if something else happens. By testing 12 hours of lead time on Graphcast 2.0, the results in Fig. 7 show us that the loss and RMSE values are identical to when the lead time was 10 hours, showing a plateau in both loss and RMSE, and in turn accuracy.

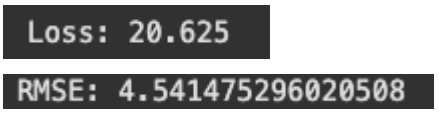

Fig. 7: Output of the model after training with a lead time of 12 hours. The loss value is reported as 20.625 and RMSE as 4.54, establishing a reference point for performance comparison against shorter lead times.. This result was used to evaluate the relative impact of lead time optimization on model accuracy and shows the plateau of the values compared to 10 hours.

Fig. 7 shows that values above 10 hours of lead time actually plateau in both RMSE and loss, indicating diminishing returns on lead time versus accuracy.

By comparing the three lead times, 5 hours/10 hours/12 hours, we can see that a lower lead time in comparison to the baseline does lead to a better RMSE and loss value, but shows that any value above the baseline shows diminishing returns. Fig. 8 below shows the three lead times and their RMSE/Loss values in a line chart that shows where the plateau happens.

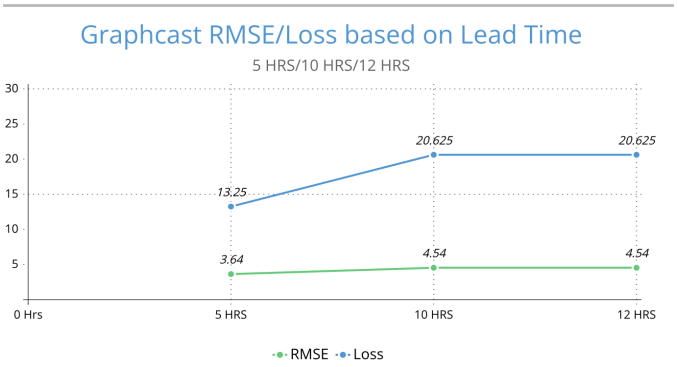

Fig. 8: Line chart showing the 3 lead times of 5/10/12 hours and how the RMSE and Loss changes as lead time increases, showing an increase then a plateau.

Graphcast 1.0 to Graphcast 2.0 follows a series of steps that require model changes and optimizations(see Fig. 3). The first of these steps include a double feature selection, double due to variable checking. However the feature selection returned no beneficial ways to improve our dataset due to the importance of all the variables to 2m_temperature being stuck at 0.0. However this did not impact step 2 of the process which is to do lead time experimentations. These experimentations show that a lower lead time(5 hours; Fig. 6), compared to a baseline of 10 hours, did have a significant improvement of RMSE and loss at 3.64 and 13.25 respectively. However, values higher than the baseline(12 hours; Fig. 7) show proof of diminishing returns, where the values are identical to the baseline of RMSE and loss at 4.54 and 20.625 respectively. This shows that based on our novel idea, step 2 of Graphcast 1.0 to Graphcast 2.0 has a much larger impact on RMSE and loss values, where a lower lead time improves our accuracy(by loss and RMSE) and improves the model overall.

VII. Discussion and Conclusion

Overall our research looked at enhancing the accuracy of the GraphCast model, specifically GraphCast 2.0. GraphCast is one of the original and important Machine Learning models used for approaching weather prediction, using data all the way from the 1970's. What we specifically focused and touched on was the use of XGBoost for feature selection, and testing on various lead times throughout our research (5 Hours, 10 Hours, and 12 Hours). Even though feature selection didn't present a big effect on feature selection, it did help with the training inputs and make that process more valuable. More importantly, our research found and discovered that altering the lead times allowed our GraphCast 2.0 model to work well for accuracy from metrics like RMSE and Loss.

What we found and discovered from our results was that it helped with the original hypotheses on how lower lead times, such as 5 hours, can lead to accurate predictions. When we set the lead time to 5 hours, it was able to result in accuracy that made it the highest. Feature selection was very helpful in also reducing gaussian noise. This can be useful for being able to deploy models at an adaptable rate for the future. We specifically found that model architecture and the preprocessing of data were two of the main factors in the execution of our model. This technically shows how even though we cannot enhance the original GraphCast model for all the metrics, there is a chance at improving based on just what we have discovered in this research.

Our project also looked at three future directions this research and proposal could take. The first part of this future work looks at the use of regionally based normalization, incorporating the use of latitude and longitude areas, while not focusing on the global scale of normalization data, as our current model focuses on. For regionally based normalization, calculating mean and standard deviation can be used in the process for this, on each specific region. On top of this, it allows for predictions to factor in patterns locally. This is another way to decrease the RMSE values, while also improving the prediction of weather in regions and not globally.

Another future direction this research could delve into is incorporating the use of Transformers, changing the original use of Graphical Neural Networks, specifically GNNs. Using a transformer

would affect the model by being able to better understand specific patterns over time and space, importantly when extreme weather events occur. Other weather-based ML models that we see today, such as GenCast and PanguWeather, exceed their expectations, as they are able to produce those results in terms of accuracy.

Finally, another direction taken for another approach in the future is making the resolution forecasting smaller. This is specifically 0.25 degrees to around 1-2.5 degrees. This allows for noise to be lessened that is produced out of nowhere. This then approaches a lower RMSE, on a large scale weather forecast.

All in all, our enhanced version of GraphCast, GraphCash 2.0, is able to visualize important enhancements. This is done on short term and forecasts, and many different approaches can be taken to further look at how weather prediction can evolve. As discussed, methods like: region-based normalization, using Transformer architecture, and reducing resolution downsizing, can improve this GraphCast 2.0, improving metrics on top of accuracy.

References

[1] R. Lam *et al.*, "Learning skillful medium-range global weather forecasting," *Science*, vol. 382, no. 6677, Nov. 2023, doi: https://doi.org/10.1126/science.adi2336.

[2] I. Price *et al.*, "Probabilistic weather forecasting with machine learning," *Nature*, Dec. 2024, doi: https://doi.org/10.1038/s41586-024-08252-9.

[3] Zied Ben Bouallègue *et al.*, "The rise of data-driven weather forecasting: A first statistical assessment of machine learning-based weather forecasts in an operational-like context," *Bulletin of the American Meteorological Society*, Feb. 2024, doi: https://doi.org/10.1175/bams-d-23-0162.1.