GenCast: Redefining Weather Forecasting with Probabilistic AI Models

Report submitted to the SASTRA Deemed to be University as the requirement for the course

CSE300 - MINI PROJECT

Submitted by

ANNAREDDY VENKATA VISHNU VARDHAN REDDY
(Reg No.: 126003026, Computer Science and Engineering)
CHALAPATI NARASIMHA YUVA PRANEETH
(Reg No.: 126003056, Computer Science and Engineering)
KESAMREDDY DEEPAK REDDY
(Reg No.: 126003130, Computer Science and Engineering)

May 2025

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act,1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING
THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled **“GenCast: Redifining Weather Forecasting with Probabilistic AI models”** submitted as a requirement for the course, CSE300: **MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Mr. ANNAREDDY VENKATA VISHNU VARDHAN REDDY (Reg No.: 126003026, Computer Science and Engineering), Mr. CHALAPATI NARASIMHA YUVA PRANEETH (Reg No.: 126003056, Computer Science and Engineering), Mr. KESAMREDDY DEEPAK REDDY (Reg No.: 126003130, Computer Science and Engineering)** during the academic year 2024-25, in the School of Computing, under my supervision.

Signature of Project Supervisor :

Name with Affiliation : Dr. MANJULA K R, SOC

Date : 17.04.2025

Mini Project *Viva voce* held on 20 / 05 / 25

Examiner 1 **Examiner 2**

Acknowledgements

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah,** Associate Dean, Research, **Dr. K.Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan,** Associate Dean, Infrastructure, **Dr.R. Alageswaran**, Associate Dean, Students Welfare

Our guide **Dr. Manjula K R**, Associate Professor, School of Computing was the driving force behind this whole idea from the start. Her deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing us an opportunity to showcase our skills through this project.

List of Figures

List of Tables

Abbreviations

ECMWF	European Centre for Medium-Range Weather Forecasts
ENS	Ensemble
ERA-5	ECMWF re-analysis
MLWP	Machine Learning-based Weather Prediction
NWP	Numerical Weather Prediction
ML	Machine Learning
CDS	Climate Data Store
RMSE	Root Mean Squared Error
CRPS	Continuous Ranked Probability Score
MAE	Mean Absolute Error
HRES	High Resolution
GNN	Graph Neural Networks
TPU	Tensor Processing Unit
JAX	Just After Execution
WWRP	World Weather Research Programme
WDC	World Data Center for Meteorology
AI	Artificial Intelligence

NOTATIONS

Greek Symbols

λ	Per-Noise-level Loss Weight
σ	Noise Level
θ	Model parameters

Miscellaneous Symbols

$\forall$	For all
$\in$	Belongs to
$\sum$	Summation
$\prod$	Product
$\approx$	Approximately equal

English Symbols

P	Conditional Probability
X	Atmospheric State
Z	A residual with respect to the most recent weather data
$D\theta$	The denoiser neural network with parameters
$f\ \theta$	The neural network function underlying the denoiser.
$r\ \theta$	The refinement function for the diffusion sampling process.
w	per-variable loss weight
a	area of longitude-latitude

ABSTRACT

Weather forecasting plays a critical role in enabling effective decision-making across various domains, from disaster management to renewable energy planning, Traditional approaches rely on Numerical Weather Prediction (NWP), which uses physics-based simulations to generate deterministic forecasts. Although ensemble NWP models, like the European Centre for Medium Range Weather Forecasts (ECMWF) ENS, represent uncertainties better, they are computationally intensive and exhibit limitations in accuracy for extreme weather events. Machine Learning-based Weather Prediction (MLWP) methods have emerged as faster alternatives but often fail to quantify forecast uncertainty accurately. Existing ML models tend to focus on minimizing forecast errors for deterministic scenarios, leading to blurry predictions and limited representation of probable weather trajectories. This paper introduces GenCast, a model based on diffusion models. Gencast surpasses the accuracy and efficiency of leading operational NWP ensemble systems by generating stochastic 15-day global forecasts at high resolution (0.25° latitude-longitude) within minutes. Trained on decades of reanalysis data, it produces realistic individual weather trajectories, ensuring better marginal and joint spatio-temporal forecast distributions. Gencast demonstrates superior performance in predicting tropical cyclone paths, extreme weather events, and wind power production. By addressing the limitations of traditional NWP and MLWP systems, GenCast paves the way for enhanced, operational forecasting and decision-making. This work showcases the transformative potential of generative AI in addressing high-dimensional, spatiotemporal forecasting challenges.

KEYWORDS: Diffusion models, ensemble forecasting, CRPS, GenCast, ERA5, ECMWF, RMSE, MLWP, NWP.

Table of Contents

CHAPTER 1

SUMMARY OF THE BASE PAPER

Base Paper Title : Probabilistic Weather forecasting with machine learning

Journal Name : Nature

Year of Publication : 2024

Authors : Ilan Price, Alvaro Sanchez-Gonzalez, Ferran Alet, Tom R. Andersson

Publisher : Nature

Base Paper Link : **https://doi.org/10.1038/s41586-024-08252-9**

The study uses the 2021 **ECMWF re-analysis (ERA-5)** dataset. GenCast is an ML weather prediction method, trained on decades of reanalysis data. GenCast generates an ensemble of stochastic 15-day global forecasts, at 12-h steps and 0.25° latitude–longitude resolution, for more than 80 surface and atmospheric variables. It has greater skill than ENS on 97.2% of 1,320 targets we evaluated and better predicts extreme weather, tropical cyclone tracks and wind power production.

Dataset Details:

Link:

https://console.cloud.google.com/storage/browser/dm_graphcast/gencast/dataset;tab=objects?pageState=(%22StorageObjectListTable%22:(%22f%22:%22%255B%255D%22))&prefix=&forceOnObjectsSortingFiltering=false

Our dataset covers the period 1979–2019. During the development phase of GenCast, we used dates from 1979 to 2017 for training and validated results in 2018. Before starting the test phase, we froze all model and training choices, retrained the model on data from 1979 to 2018 and evaluated results in 2019.

Table 1: ERA5 Dataset variables classification

Type	Variable name	Short name	ECMWF Parameter ID	Role (accumulation period, if applicable)
Atmospheric	Geopotential	z	129	Input/Predicted
Atmospheric	Specific humidity	q	133	Input/Predicted
Atmospheric	Temperature	t	130	Input/Predicted
Atmospheric	U component of wind	u	131	Input/Predicted
Atmospheric	V component of wind	v	132	Input/Predicted
Atmospheric	Vertical velocity	w	135	Input/Predicted
Single	2 metre temperature	2t	167	Input/Predicted
Single	10 metre u wind component	10u	165	Input/Predicted
Single	10 metre v wind component	10v	166	Input/Predicted
Single	Mean sea level pressure	msl	151	Input/Predicted
Single	Sea Surface Temperature	sst	34	Input/Predicted
Single	Total precipitation	tp	228	Predicted (12h)
Static	Geopotential at surface	z	129	Input

Static	Land-sea mask	lsm	172	Input
Static	Latitude	n/a	n/a	Input
Static	Longitude	n/a	n/a	Input
Clock	Local time of day	n/a	n/a	Input
Clock	Elapsed year progress	n/a	n/a	Input

The “Type” column indicates whether the variable represents a static property, a time- varying single-level property (e.g. surface variables are included), or a time-varying atmospheric property. The “Variable name” and “Short name” columns are ECMWF’s labels. The “ECMWF Parameter ID” column is ECMWF’s numeric label, and can be used to construct the URL for ECMWF’s description of the variable, by appending it as suffix to the following prefix, replacing “ID” with the numeric code: https://apps.ecmwf.int/codes/grib/param-db/?id=ID. The “Role” column indicates whether the variable is something our model takes as input and predicts, or only uses as input context (the double horizontal line separates predicted from input-only variables, to make the partitioning more visible). For atmospheric variables, the 13 atmospheric pressure levels are taken as input and predicted by GenCast are: 50, 100, 150, 200, 250, 300, 400, 500, 600, 700, 850, 925, and 1000 hPa.

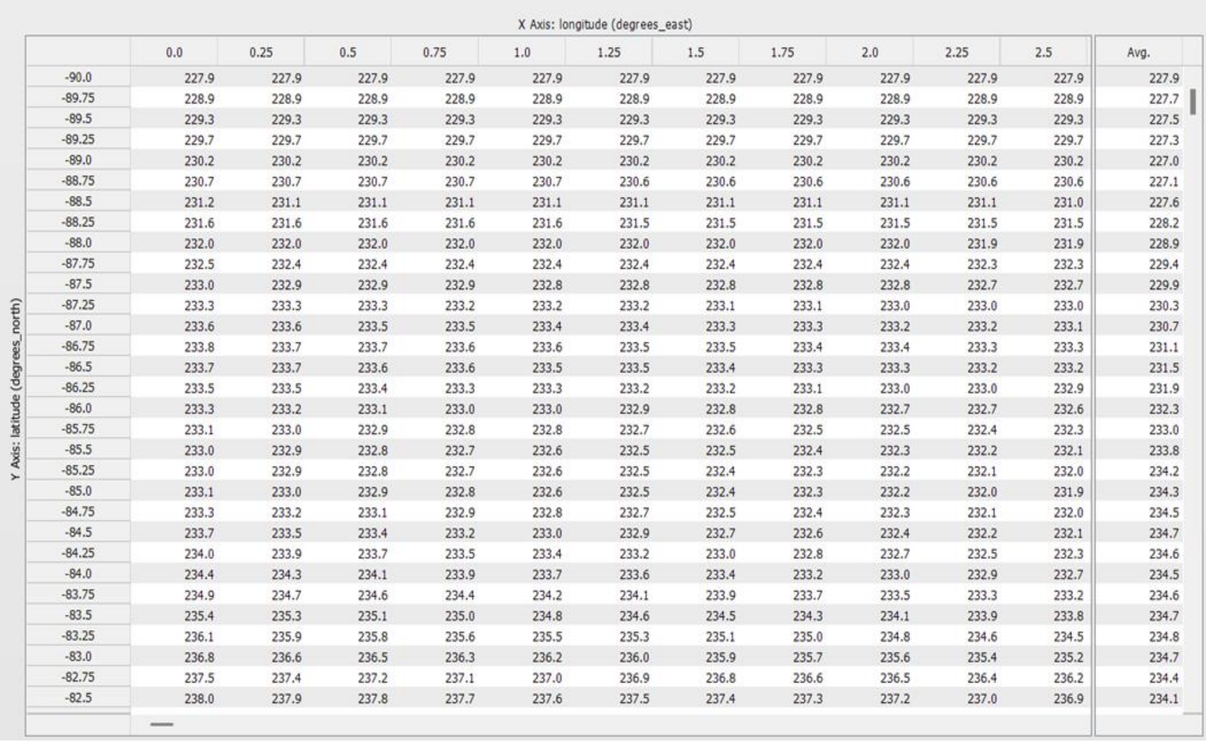
Data Preprocessing:

Unsampling EDA analysis:

To create ensemble forecasts, we used ERA5 EDA data, which is only available at 0.5° resolution. We upsampled it to 0.25° using bilinear interpolation. For sea surface temperature (SST), this caused a mismatch in land/sea (NaN) areas between EDA and ERA5. To fix this, we used ERA5's land/sea mask and filled any missing EDA SST values with ERA5's standard SST data.

Variables with NaNs:

ERA5 sea surface temperature (SST) data contains NaNs over land by default. As preprocessing of the SST training and evaluation data, values over land are replaced with the minimum sea surface temperature seen globally in a subset of ERA5.



X Axis: longitude (degrees_east)

Y Axis: latitude (degrees_north)

	0.0	0.25	0.5	0.75	1.0	1.25	1.5	1.75	2.0	2.25	2.5	Avg.
-90.0	227.9	227.9	227.9	227.9	227.9	227.9	227.9	227.9	227.9	227.9	227.9	227.9
-89.75	228.9	228.9	228.9	228.9	228.9	228.9	228.9	228.9	228.9	228.9	228.9	227.7
-89.5	229.3	229.3	229.3	229.3	229.3	229.3	229.3	229.3	229.3	229.3	229.3	227.5
-89.25	229.7	229.7	229.7	229.7	229.7	229.7	229.7	229.7	229.7	229.7	229.7	227.3
-89.0	230.2	230.2	230.2	230.2	230.2	230.2	230.2	230.2	230.2	230.2	230.2	227.0
-88.75	230.7	230.7	230.7	230.7	230.7	230.6	230.6	230.6	230.6	230.6	230.6	227.1
-88.5	231.2	231.1	231.1	231.1	231.1	231.1	231.1	231.1	231.1	231.1	231.0	227.6
-88.25	231.6	231.6	231.6	231.6	231.6	231.5	231.5	231.5	231.5	231.5	231.5	228.2
-88.0	232.0	232.0	232.0	232.0	232.0	232.0	232.0	232.0	232.0	231.9	231.9	228.9
-87.75	232.5	232.4	232.4	232.4	232.4	232.4	232.4	232.4	232.4	232.3	232.3	229.4
-87.5	233.0	232.9	232.9	232.9	232.8	232.8	232.8	232.8	232.8	232.7	232.7	229.9
-87.25	233.3	233.3	233.3	233.2	233.2	233.2	233.1	233.1	233.0	233.0	233.0	230.3
-87.0	233.6	233.6	233.5	233.5	233.4	233.4	233.3	233.3	233.2	233.2	233.1	230.7
-86.75	233.8	233.7	233.7	233.6	233.6	233.5	233.5	233.4	233.4	233.3	233.3	231.1
-86.5	233.7	233.7	233.6	233.6	233.5	233.5	233.4	233.3	233.3	233.2	233.2	231.5
-86.25	233.5	233.5	233.4	233.3	233.3	233.2	233.2	233.1	233.0	233.0	232.9	231.9
-86.0	233.3	233.2	233.1	233.0	233.0	232.9	232.8	232.8	232.7	232.7	232.6	232.3
-85.75	233.1	233.0	232.9	232.8	232.8	232.7	232.6	232.5	232.5	232.4	232.3	233.0
-85.5	233.0	232.9	232.8	232.7	232.6	232.5	232.5	232.4	232.3	232.2	232.1	233.8
-85.25	233.0	232.9	232.8	232.7	232.6	232.5	232.4	232.3	232.2	232.1	232.0	234.2
-85.0	233.1	233.0	232.9	232.8	232.6	232.5	232.4	232.3	232.2	232.0	231.9	234.3
-84.75	233.3	233.2	233.1	232.9	232.8	232.7	232.5	232.4	232.3	232.1	232.0	234.5
-84.5	233.7	233.5	233.4	233.2	233.0	232.9	232.7	232.6	232.4	232.2	232.1	234.7
-84.25	234.0	233.9	233.7	233.5	233.4	233.2	233.0	232.8	232.7	232.5	232.3	234.6
-84.0	234.4	234.3	234.1	233.9	233.7	233.6	233.4	233.2	233.0	232.9	232.7	234.5
-83.75	234.9	234.7	234.6	234.4	234.2	234.1	233.9	233.7	233.5	233.3	233.2	234.6
-83.5	235.4	235.3	235.1	235.0	234.8	234.6	234.5	234.3	234.1	233.9	233.8	234.7
-83.25	236.1	235.9	235.8	235.6	235.5	235.3	235.1	235.0	234.8	234.6	234.5	234.8
-83.0	236.8	236.6	236.5	236.3	236.2	236.0	235.9	235.7	235.6	235.4	235.2	234.7
-82.75	237.5	237.4	237.2	237.1	237.0	236.9	236.8	236.6	236.5	236.4	236.2	234.4
-82.5	238.0	237.9	237.8	237.7	237.6	237.5	237.4	237.3	237.2	237.0	236.9	234.1

Fig 1: Sample Data

Models Used in GenCast:

Diffusion-Based Generative Modelling:

Employs a diffusion model adapted to the Earth's spherical geometry. Models the conditional probability distribution P(X_{t+1}| X_t, X_{t-1}), where X_t represents the global weather state at time t. Generates ensemble forecasts by sampling from this distribution, capturing the uncertainty inherent in weather systems.

Schematic of how GenCast produces a forecast:

GenCast is implemented as a conditional diffusion model, a generative ML method that can model the probability distribution of complex data and generate new samples. Diffusion models underpin many of the recent advances in modelling natural images, sounds and videos under the umbrella of generative AI. Diffusion models work through a process of iterative refinement. A future atmospheric state, $\mathbf{X}^{t+1}$, is produced by iteratively refining a candidate state initialized as pure noise, Z0t+1, conditioned on the previous two atmospheric states ($\mathbf{X}^{t}$, $\mathbf{X}^{t-1}$).

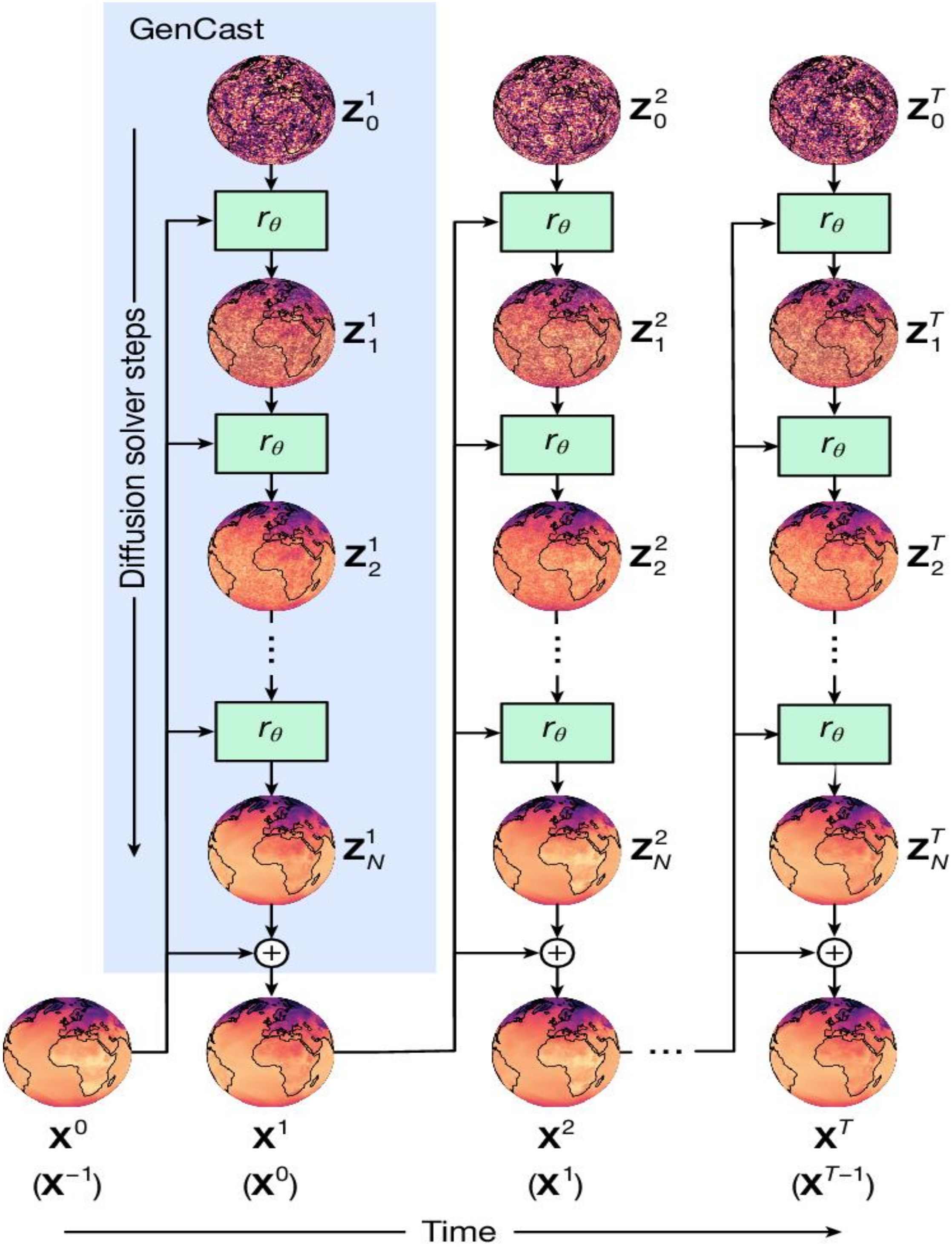


Fig 2: Autoregressive Sampling

Ensemble Forecast Generation:

Produces ensembles of stochastic 15-day global forecasts at 12-hour intervals. Each ensemble member represents a plausible future weather trajectory, allowing for probabilistic forecasting. Forecasts are generated in approximately 8 minutes on a single Cloud TPU v5 device.

Spatial Representation:

Utilizes refined icosahedral meshes to represent the Earth's surface, enabling efficient and accurate modeling of global weather patterns.

Architecture:

Encoder: Maps noisy weather states and conditioning inputs (previous weather states) to an internal representation.

Processor: A graph transformer that processes the encoded data on a spherical mesh.

Decoder: Maps the processed data back to the original grid to produce the denoised weather state.

CHAPTER 2
MERITS AND DEMERITS OF THE BASE PAPER

Merits:

1. Superior Performance
   - Outperforms the state-of-the-art ECMWF ENS ensemble (97.2% of 1,320 targets evaluated).
   - Better skill in extreme weather prediction, tropical cyclone tracking, and wind power forecasting.
2. Computational Efficiency
   - Generates 15-day global forecasts at 0.25° resolution in 8 minutes, significantly faster than traditional NWP models.
3. Probabilistic Forecasting
   - Uses a diffusion model to generate ensemble forecasts, capturing uncertainty better than deterministic ML models (e.g., GraphCast).
   - Produces sharp, realistic weather trajectories (unlike blurred deterministic forecasts).
4. Better Calibration & Skill
   - Well-calibrated (spread/skill ratio ≈ 1, flat rank histograms).
   - Higher Relative Economic Value (REV) for extreme events (e.g., heatwaves, cyclones).
5. Handling of Spatial-Temporal Dependencies
   - Effectively models joint spatio-temporal structure, improving cyclone track and wind power predictions.

Demerits

1. Computational Cost of Diffusion Models
   - Requires 39 function evaluations per time step, making it slower than deterministic ML models (e.g., GraphCast).
2. Dependence on NWP Initialization
   - Relies on ERA5 reanalysis for initialization, which may inherit biases from NWP data assimilation.
3. Limited Evaluation on Precipitation
   - Excluded from main results due to uncertainty in ERA5 precipitation data.
4. No Cyclogenesis Prediction
   - Evaluated only on existing cyclones, not cyclone formation.
5. Training Complexity
   - Two-stage training (1° pre-training → 0.25° pre-training) increases development effort and requires more memory.
6. Operational Deployment Challenges
   - Needs further optimization (e.g., distillation) for real-time high-resolution forecasting.

Proposed methodology:

Technologies used:

1. Python
2. TensorFlow
3. Pandas / NumPy
4. Matplotlib
5. Jax/Haiku
6. Xarray
7. IpyWidgets
8. Typing
9. Ipython/display

CHAPTER 3

SOURCE CODE

1. Installation and Initialization

```
# @title Upgrade packages (kernel needs to be restarted after running this cell).
%pip install -U importlib_metadata
```

2. Import necessary Github repo

```
# @title Pip install repo and dependencies

%pip install --upgrade https://github.com/deepmind/graphcast/archive/master.zip
```

3. Reconfigure JAX if running on TPU

```
# @title Reconfigure jax if running on TPU.
# This is required due to outdated jax and libtpu versions in Colab TPU images.
%pip uninstall -y jax jaxlib libtpu libtpu-nightly
%pip install -U "jax[tpu]==0.5.1" -f https://storage.googleapis.com/jax-
releases/libtpu_releases.html
```

4. Import necessary modules

```
# @title Imports
import dataclasses
import datetime
import math
from google.cloud import storage
from typing import Optional
import haiku as hk
from IPython.display import HTML
from IPython import display
import ipywidgets as widgets
import jax
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import animation
import numpy as np
import xarray
from graphcast import rollout
from graphcast import xarray_jax
from graphcast import normalization
from graphcast import checkpoint
from graphcast import data_utils
from graphcast import xarray_tree
from graphcast import gencast
from graphcast import denoiser
from graphcast import nan_cleaning
```

5. Define Plotting Functions

```
# @title Plotting functions
def select(
    data: xarray.Dataset,
    variable: str,    level: Optional[int] = None,
max_steps: Optional[int] = None
) -> xarray.Dataset:
  data = data[variable]
  if "batch" in data.dims:
    data = data.isel(batch=0)
  if max_steps is not None and "time" in data.sizes and max_steps < data.sizes["time"]:
    data = data.isel(time=range(0, max_steps))
  if level is not None and "level" in data.coords:
    data = data.sel(level=level)
  return data

def scale(
    data: xarray.Dataset,
    center: Optional[float] = None,
    robust: bool = False,
    ) -> tuple[xarray.Dataset, matplotlib.colors.Normalize, str]:
  vmin = np.nanpercentile(data, (2 if robust else 0))
  vmax = np.nanpercentile(data, (98 if robust else 100))
  if center is not None:
    diff = max(vmax - center, center - vmin)
    vmin = center - diff
    vmax = center + diff
  return (data, matplotlib.colors.Normalize(vmin, vmax),
          ("RdBu_r" if center is not None else "viridis"))

def plot_data(
    data: dict[str, xarray.Dataset],
    fig_title: str,
    plot_size: float = 5,
    robust: bool = False,
    cols: int = 4
    ) -> tuple[xarray.Dataset, matplotlib.colors.Normalize, str]:

  first_data = next(iter(data.values()))[0]
  max_steps = first_data.sizes.get("time", 1)
  assert all(max_steps == d.sizes.get("time", 1) for d, _, _ in data.values())
```

```
cols = min(cols, len(data))
rows = math.ceil(len(data) / cols)
figure = plt.figure(figsize=(plot_size * 2 * cols,
                             plot_size * rows))
figure.suptitle(fig_title, fontsize=16)
figure.subplots_adjust(wspace=0, hspace=0)
figure.tight_layout()

images = []
for i, (title, (plot_data, norm, cmap)) in enumerate(data.items()):
  ax = figure.add_subplot(rows, cols, i+1)
  ax.set_xticks([])
  ax.set_yticks([])
  x.set_title(title)
  im = ax.imshow(
      plot_data.isel(time=0, missing_dims="ignore"), norm=norm,
      origin="lower", cmap=cmap)
  plt.colorbar(
      mappable=im,
      ax=ax,
      orientation="vertical",
      pad=0.02,
      aspect=16,
      shrink=0.75,
      cmap=cmap,
      extend=("both" if robust else "neither"))
  images.append(im)

def update(frame):
  if "time" in first_data.dims:
    td = datetime.timedelta(microseconds=first_data["time"][frame].item() / 1000)
    figure.suptitle(f"{fig_title}, {td}", fontsize=16)
  else:
    figure.suptitle(fig_title, fontsize=16)
  for im, (plot_data, norm, cmap) in zip(images, data.values()):
    im.set_data(plot_data.isel(time=frame, missing_dims="ignore"))

ani = animation.FuncAnimation(
    fig=figure, func=update, frames=max_steps, interval=250)
plt.close(figure.number)
return HTML(ani.to_jshtml())
```

6. Load the Data and Initialize the Model

```
# @title Authenticate with Google Cloud Storage

# Gives you an authenticated client, in case you want to use a private bucket.
gcs_client = storage.Client.create_anonymous_client()
gcs_bucket = gcs_client.get_bucket("dm_graphcast")
dir_prefix = "gencast/"
```

7. Load the Model Params

```
# @title Choose the model

params_file_options = [
    name for blob in gcs_bucket.list_blobs(prefix=(dir_prefix+"params/"))
    if (name := blob.name.removeprefix(dir_prefix+"params/"))]  # Drop empty string.

latent_value_options = [int(2**i) for i in range(4, 10)]
random_latent_size = widgets.Dropdown(
    options=latent_value_options, value=512,description="Latent size:")
random_attention_type = widgets.Dropdown(
    options=["splash_mha", "triblockdiag_mha", "mha"], value="splash_mha",
description="Attention:")
random_mesh_size = widgets.IntSlider(
    value=4, min=4, max=6, description="Mesh size:")
random_num_heads = widgets.Dropdown(
    options=[int(2**i) for i in range(0, 3)], value=4,description="Num heads:")
random_attention_k_hop = widgets.Dropdown(
    options=[int(2**i) for i in range(2, 5)], value=16,description="Attn k hop:")
random_resolution = widgets.Dropdown(
    options=["1p0", "0p25"], value="1p0", description="Resolution:")

def update_latent_options(*args):
  def _latent_valid_for_attn(attn, latent, heads):
    head_dim, rem = divmod(latent, heads)
    if rem != 0:
      return False
    # Required for splash attn.
    if head_dim % 128 != 0:
      return attn != "splash_mha"
    return True
  attn = random_attention_type.value
  heads = random_num_heads.value
  random_latent_size.options = [
      latent for latent in latent_value_options
      if _latent_valid_for_attn(attn, latent, heads)]
```

```
# Observe changes to only allow for valid combinations.
random_attention_type.observe(update_latent_options, "value")
random_latent_size.observe(update_latent_options, "value")
random_num_heads.observe(update_latent_options, "value")

params_file = widgets.Dropdown(
options=[f for f in params_file_options if('1p0deg Mini' in f or ('0p25deg' in f
and'Operational' not in f))],
    description="Params file:",
    layout={"width": "max-content"})

source_tab = widgets.Tab([
    params_file,
    widgets.VBox([
        random_attention_type,
        random_mesh_size,
        random_num_heads,
        random_latent_size,
        random_attention_k_hop,
        random_resolution
    ]),
])
source_tab.set_title(0, "Checkpoint")
source_tab.set_title(1, "Random")
widgets.VBox([
    source_tab,
    widgets.Label(value="Run the next cell to load the model. Rerunning this cell clears your
selection.")
])
```

8. **Load the Model**

```
# @title Load the model

source = source_tab.get_title(source_tab.selected_index)

if source == "Random":
  params = None  # Filled in below
  state = {}
  task_config = gencast.TASK
  # Use default values.
  sampler_config = gencast.SamplerConfig()
  noise_config = gencast.NoiseConfig()
  noise_encoder_config = denoiser.NoiseEncoderConfig()
  # Configure, otherwise use default values.
  denoiser_architecture_config = denoiser.DenoiserArchitectureConfig(
    sparse_transformer_config = denoiser.SparseTransformerConfig(
        attention_k_hop=random_attention_k_hop.value,
        attention_type=random_attention_type.value,
        d_model=random_latent_size.value,
        num_heads=random_num_heads.value
        ),
    mesh_size=random_mesh_size.value,
    latent_size=random_latent_size.value,
  )
else:
  assert source == "Checkpoint"
  with gcs_bucket.blob(dir_prefix + f"params/{params_file.value}").open("rb") as f:
    ckpt = checkpoint.load(f, gencast.CheckPoint)
  params = ckpt.params
  state = {}

  task_config = ckpt.task_config
  sampler_config = ckpt.sampler_config
  noise_config = ckpt.noise_config
  noise_encoder_config = ckpt.noise_encoder_config
  denoiser_architecture_config = ckpt.denoiser_architecture_config
  print("Model description:\n", ckpt.description, "\n")
 print("Model license:\n", ckpt.license, "\n")
```

9. **Load the Example Data**

```
# @title Get and filter the list of available example datasets

dataset_file_options = [
    name for blob in gcs_bucket.list_blobs(prefix=(dir_prefix + "dataset/"))
    if (name := blob.name.removeprefix(dir_prefix+"dataset/"))]  # Drop empty string.

def parse_file_parts(file_name):
  return dict(part.split("-", 1) for part in file_name.split("_"))

def data_valid_for_model(file_name: str, params_file_name: str):
  """Check data type and resolution matches."""
  data_file_parts = parse_file_parts(file_name.removesuffix(".nc"))
  data_res = data_file_parts["res"].replace(".", "p")
  if source == "Random":
    return random_resolution.value == data_res
  res_matches = data_res in params_file_name.lower()
  source_matches = "Operational" in params_file_name
  if data_file_parts["source"] == "era5":
    source_matches = not source_matches
  return res_matches and source_matches

dataset_file = widgets.Dropdown(
    options=[
        (", ".join([f"{k}: {v}" for k, v in parse_file_parts(option.removesuffix(".nc")).items()]),
option)
        for option in dataset_file_options
        if data_valid_for_model(option, params_file.value)
    ],
    description="Dataset file:",
    layout={"width": "max-content"})
widgets.VBox([
    dataset_file,
    widgets.Label(value="Run the next cell to load the dataset. Rerunning this cell clears your
selection and refilters the datasets that match your model.")
])
```

10. **Load the Weather Data**

```
# @title Load weather data

with gcs_bucket.blob(dir_prefix+f"dataset/{dataset_file.value}").open("rb") as f:
  example_batch = xarray.load_dataset(f).compute()

assert example_batch.dims["time"] >= 3  # 2 for input, >=1 for targets

print(", ".join([f"{k}: {v}" for k, v in
parse_file_parts(dataset_file.value.removesuffix(".nc")).items()]))
example_batch
```

11. **Choose Data to plot**

```
# @title Choose data to plot

plot_example_variable = widgets.Dropdown(
    options=example_batch.data_vars.keys(),
    value="2m_temperature",
    description="Variable")
plot_example_level = widgets.Dropdown(
    options=example_batch.coords["level"].values,
    value=500,
    description="Level")
plot_example_robust = widgets.Checkbox(value=True, description="Robust")
plot_example_max_steps = widgets.IntSlider(
    min=1, max=example_batch.dims["time"], value=example_batch.dims["time"],
    description="Max steps")
```

```
widgets.VBox([
    plot_example_variable,
    plot_example_level,
    plot_example_robust,
    plot_example_max_steps,
    widgets.Label(value="Run the next cell to plot the data. Rerunning this cell clears your
selection.")
])
```

12. Plot Example Data

```
# @title Plot example data

plot_size = 7
data = {
    " ": scale(select(example_batch, plot_example_variable.value, plot_example_level.value,
plot_example_max_steps.value),
              robust=plot_example_robust.value),
}
fig_title = plot_example_variable.value
if "level" in example_batch[plot_example_variable.value].coords:
  fig_title += f" at {plot_example_level.value} hPa"

plot_data(data, fig_title, plot_size, plot_example_robust.value)
```

13. Extract Training and Eval Data

```
# @title Extract training and eval data

train_inputs, train_targets, train_forcings = data_utils.extract_inputs_targets_forcings(
    example_batch, target_lead_times=slice("12h", "12h"), # Only 1AR training.
    **dataclasses.asdict(task_config))

eval_inputs, eval_targets, eval_forcings = data_utils.extract_inputs_targets_forcings(
    example_batch, target_lead_times=slice("12h", f"{(example_batch.dims['time']-2)*12}h"),
# All but 2 input frames.
    **dataclasses.asdict(task_config))

print("All Examples:  ", example_batch.dims.mapping)
print("Train Inputs:  ", train_inputs.dims.mapping)
print("Train Targets: ", train_targets.dims.mapping)
print("Train Forcings:", train_forcings.dims.mapping)
print("Eval Inputs:   ", eval_inputs.dims.mapping)
print("Eval Targets:  ", eval_targets.dims.mapping)
print("Eval Forcings: ", eval_forcings.dims.mapping)
```

14. **Load Normalization Data**

```
# @title Load normalization data

with gcs_bucket.blob(dir_prefix+"stats/diffs_stddev_by_level.nc").open("rb") as f:
  diffs_stddev_by_level = xarray.load_dataset(f).compute()
with gcs_bucket.blob(dir_prefix+"stats/mean_by_level.nc").open("rb") as f:
  mean_by_level = xarray.load_dataset(f).compute()
with gcs_bucket.blob(dir_prefix+"stats/stddev_by_level.nc").open("rb") as f:
  stddev_by_level = xarray.load_dataset(f).compute()
with gcs_bucket.blob(dir_prefix+"stats/min_by_level.nc").open("rb") as f:
  min_by_level = xarray.load_dataset(f).compute()
```

15. **Build jitted functions, and possibly initialize random weights**

```
# @title Build jitted functions, and possibly initialize random weights

def construct_wrapped_gencast():
  """Constructs and wraps the GenCast Predictor."""
  predictor = gencast.GenCast(
      sampler_config=sampler_config,
      task_config=task_config,
      denoiser_architecture_config=denoiser_architecture_config,
      noise_config=noise_config,
      noise_encoder_config=noise_encoder_config,
  )

  predictor = normalization.InputsAndResiduals(
      predictor,
      diffs_stddev_by_level=diffs_stddev_by_level,
      mean_by_level=mean_by_level,
      stddev_by_level=stddev_by_level,
  )

  predictor = nan_cleaning.NaNCleaner(
      predictor=predictor,
      reintroduce_nans=True,
      fill_value=min_by_level,
      var_to_clean='sea_surface_temperature',
  )

  return predictor

@hk.transform_with_state
def run_forward(inputs, targets_template, forcings):
  predictor = construct_wrapped_gencast()
  return predictor(inputs, targets_template=targets_template, forcings=forcings)

@hk.transform_with_state
def loss_fn(inputs, targets, forcings):
  predictor = construct_wrapped_gencast()
  loss, diagnostics = predictor.loss(inputs, targets, forcings)
  return xarray_tree.map_structure(
      lambda x: xarray_jax.unwrap_data(x.mean(), require_jax=True),
      (loss, diagnostics),
  )
def grads_fn(params, state, inputs, targets, forcings):
  def _aux(params, state, i, t, f):
    (loss, diagnostics), next_state = loss_fn.apply(
        params, state, jax.random.PRNGKey(0), i, t, f
    )
    return loss, (diagnostics, next_state)

  (loss, (diagnostics, next_state)), grads = jax.value_and_grad(
      _aux, has_aux=True
  )(params, state, inputs, targets, forcings)
  return loss, diagnostics, next_state, grads

if params is None:
  init_jitted = jax.jit(loss_fn.init)
  params, state = init_jitted(
      rng=jax.random.PRNGKey(0),
      inputs=train_inputs,
      targets=train_targets,
      forcings=train_forcings,
  )

loss_fn_jitted = jax.jit(
    lambda rng, i, t, f: loss_fn.apply(params, state, rng, i, t, f)[0]
)
grads_fn_jitted = jax.jit(grads_fn)
run_forward_jitted = jax.jit(
    lambda rng, i, t, f: run_forward.apply(params, state, rng, i, t, f)[0]
)
# We also produce a pmapped version for running in parallel.
run_forward_pmap = xarray_jax.pmap(run_forward_jitted, dim="sample")
```

16. Run the Model

```
# The number of ensemble members should be a multiple of the number of devices.
print(f"Number of local devices {len(jax.local_devices())}")
```

17. **Autoregressive rollout**

```
# @title Autoregressive rollout (loop in python)

print("Inputs:  ", eval_inputs.dims.mapping)
print("Targets: ", eval_targets.dims.mapping)
print("Forcings:", eval_forcings.dims.mapping)

num_ensemble_members = 8 # @param int
rng = jax.random.PRNGKey(0)
# We fold-in the ensemble member, this way the first N members should always
# match across different runs which use take the same inputs, regardless of
# total ensemble size.
rngs = np.stack(
  [jax.random.fold_in(rng, i) for i in range(num_ensemble_members)], axis=0)
chunks = []
for chunk in rollout.chunked_prediction_generator_multiple_runs(
  # Use pmapped version to parallelise across devices.
  predictor_fn=run_forward_pmap,
  rngs=rngs,
  inputs=eval_inputs,
  targets_template=eval_targets * np.nan,
  forcings=eval_forcings,
  num_steps_per_chunk = 1,
  num_samples = num_ensemble_members,
  pmap_devices=jax.local_devices()
  ):
  chunks.append(chunk)
predictions = xarray.combine_by_coords(chunks)
```

18. **Choose predictions to plot**

```
# @title Choose predictions to plot

plot_pred_variable = widgets.Dropdown(
  options=predictions.data_vars.keys(),
  value="2m_temperature",
  description="Variable")
plot_pred_level = widgets.Dropdown(
  options=predictions.coords["level"].values,
  value=500,
  description="Level")
plot_pred_robust = widgets.Checkbox(value=True, description="Robust")
plot_pred_max_steps = widgets.IntSlider(
  min=1,
  max=predictions.dims["time"],
  value=predictions.dims["time"],
  description="Max steps")
plot_pred_samples = widgets.IntSlider(
  min=1,
  max=num_ensemble_members,
  value=num_ensemble_members,
  description="Samples")

widgets.VBox([
  plot_pred_variable,
  plot_pred_level,
  plot_pred_robust,
  plot_pred_max_steps,
  plot_pred_samples,
  widgets.Label(value="Run the next cell to plot the predictions. Rerunning this cell clears
your selection.")
])
```

19. Plot prediction samples and diffs

```
# @title Plot prediction samples and diffs

plot_size = 5
plot_max_steps = min(predictions.dims["time"], plot_pred_max_steps.value)

fig_title = plot_pred_variable.value
if "level" in predictions[plot_pred_variable.value].coords:
  fig_title += f" at {plot_pred_level.value} hPa"

for sample_idx in range(plot_pred_samples.value):
  data = {
      "Targets": scale(select(eval_targets, plot_pred_variable.value, plot_pred_level.value,
plot_max_steps), robust=plot_pred_robust.value),
      "Predictions": scale(select(predictions.isel(sample=sample_idx),
plot_pred_variable.value, plot_pred_level.value, plot_max_steps),
robust=plot_pred_robust.value),
      "Diff": scale((select(eval_targets, plot_pred_variable.value, plot_pred_level.value,
plot_max_steps) -
                     select(predictions.isel(sample=sample_idx), plot_pred_variable.value,
plot_pred_level.value, plot_max_steps)),
                    robust=plot_pred_robust.value, center=0),
  }
  display.display(plot_data(data, fig_title + f", Sample {sample_idx}", plot_size,
plot_pred_robust.value))
```

20. Plot Ensemble Mean and CRPS

```
# @title Plot ensemble mean and CRPS

def crps(targets, predictions, bias_corrected = True):
  if predictions.sizes.get("sample", 1) < 2:
    raise ValueError(
        "predictions must have dim 'sample' with size at least 2.")
  sum_dims = ["sample", "sample2"]
  preds2 = predictions.rename({"sample": "sample2"})
  num_samps = predictions.sizes["sample"]
  num_samps2 = (num_samps - 1) if bias_corrected else num_samps
  mean_abs_diff = np.abs(
     predictions - preds2).sum(
          dim=sum_dims, skipna=False) / (num_samps * num_samps2)
  mean_abs_err = np.abs(targets - predictions).sum(dim="sample", skipna=False) /
num_samps
  return mean_abs_err - 0.5 * mean_abs_diff

plot_size = 5
plot_max_steps = min(predictions.dims["time"], plot_pred_max_steps.value)

fig_title = plot_pred_variable.value
if "level" in predictions[plot_pred_variable.value].coords:
  fig_title += f" at {plot_pred_level.value} hPa"

data = {
    "Targets": scale(select(eval_targets, plot_pred_variable.value, plot_pred_level.value,
plot_max_steps), robust=plot_pred_robust.value),
    "Ensemble Mean": scale(select(predictions.mean(dim=["sample"]),
plot_pred_variable.value, plot_pred_level.value, plot_max_steps),
robust=plot_pred_robust.value),
    "Ensemble CRPS": scale(crps((select(eval_targets, plot_pred_variable.value,
plot_pred_level.value, plot_max_steps)),
                        select(predictions, plot_pred_variable.value, plot_pred_level.value,
plot_max_steps)),
                      robust=plot_pred_robust.value, center=0),
}
display.display(plot_data(data, fig_title, plot_size, plot_pred_robust.value))
```

21. Loss Computation

```
# @title Loss computation
loss, diagnostics = loss_fn_jitted(
    jax.random.PRNGKey(0),
    train_inputs,
    train_targets,
    train_forcings)
print("Loss:", float(loss))
```

22. **Gradient Computation**

```
# @title Gradient computation
loss, diagnostics, next_state, grads = grads_fn_jitted(
    params=params,
    state=state,
    inputs=train_inputs,
    targets=train_targets,
    forcings=train_forcings)
mean_grad = np.mean(jax.tree_util.tree_flatten(jax.tree_util.tree_map(lambda x: np.abs(x).mean(), grads))[0])
print(f"Loss: {loss:.4f}, Mean |grad|: {mean_grad:.6f}")
```

23. **Calculation Evaluation metrics (MAE, RMSE, Accuracy)**

```
# @title Calculate MAE, RMSE, and Accuracy

def calculate_mae(targets, predictions):
    """Calculate Mean Absolute Error (MAE)."""
    return np.abs(targets - predictions).mean(dim=["lat", "lon", "time"])

def calculate_rmse(targets, predictions):
    """Calculate Root Mean Squared Error (RMSE)."""
    return np.sqrt(((targets - predictions) ** 2).mean(dim=["lat", "lon", "time"]))

def calculate_accuracy(targets, predictions, threshold):
    """Calculate accuracy within a given threshold."""
    return (np.abs(targets - predictions) <= threshold).mean(dim=["lat", "lon", "time"])
# @title Function to Suggest Reasonable Threshold for Selected Variable

def suggest_threshold(variable):
    """
    Suggest a reasonable threshold for the selected variable.

    Args:
        variable: The name of the variable (e.g., "2m_temperature").

    Returns:
        A suggested threshold value.
    """
    threshold_suggestions = {
        "10m_u_component_of_wind": 3.0,  # m/s
        "10m_v_component_of_wind": 3.0,  # m/s
        "2m_temperature": 2.0,           # K or °C
        "geopotential": 80.0,          # meters
        "mean_sea_level_pressure": 140.0,  # Pa
        "sea_surface_temperature": 1.0,  # K or °C
        "specific_humidity": 0.0011,      # kg/kg
        "temperature": 2.0,              # K or °C
        "total_precipitation_12hr": 0.05,  # mm
        "u_component_of_wind": 5.0,      # m/s
        "v_component_of_wind": 5.0,      # m/s
        "vertical_velocity": 0.2,        # Pa/s
    }
    return threshold_suggestions.get(variable, 1.0)  # Default threshold if variable not found

# Example usage:
variable = plot_pred_variable.value  # Choose the variable to evaluate
level = plot_pred_level.value  # Choose the pressure level (if applicable)

# Suggest a reasonable threshold
threshold = suggest_threshold(variable)
print(f"Suggested threshold for {variable}: {threshold}")

# Select the target and prediction data
target_data = select(eval_targets, variable, level)
prediction_data = select(predictions.mean(dim=["sample"]), variable, level)

# Calculate MAE, RMSE, and Accuracy
mae = calculate_mae(target_data, prediction_data)
rmse = calculate_rmse(target_data, prediction_data)
accuracy = calculate_accuracy(target_data, prediction_data, threshold)

# Print results
print(f"MAE for {variable} at {level} hPa: {float(mae):.4f}")
print(f"RMSE for {variable} at {level} hPa: {float(rmse):.4f}")
print(f"Accuracy for {variable} at {level} hPa (±{threshold}): {float(accuracy) *
100:.2f}%")
```

CHAPTER 4

RESULTS AND SNAPSOTS

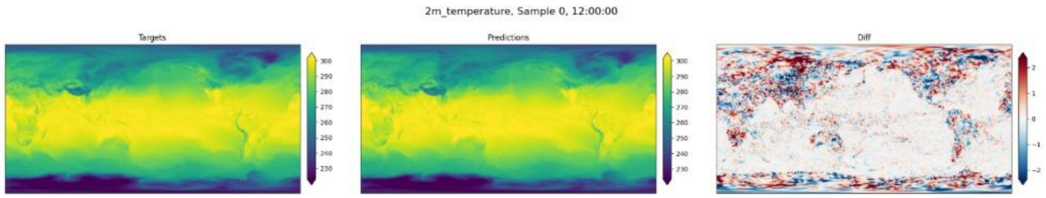


Fig 3: 2m temperature

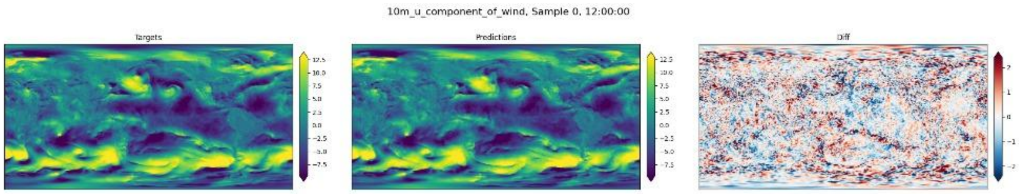


Fig 4: 10m u component of wind

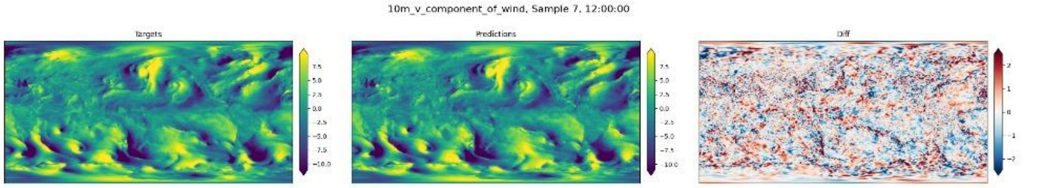


Fig 5: 10m v component of wind

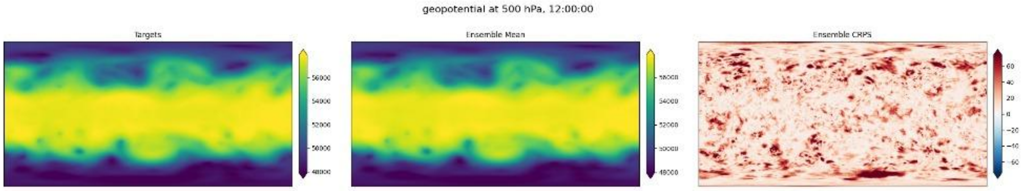


Fig 6: Geopotential

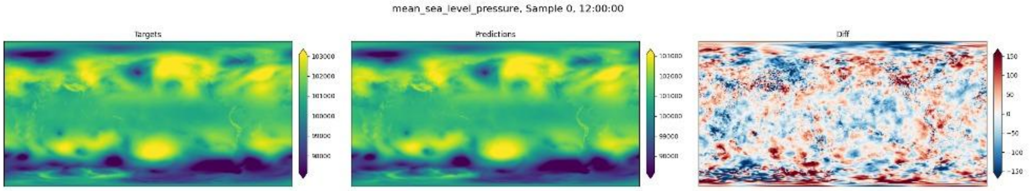


Fig 7: Mean Sea level pressure

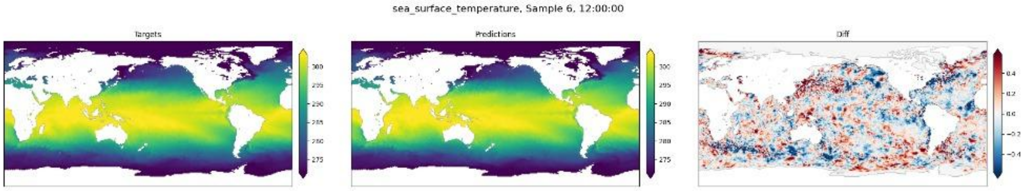


Fig 8: Sea Surface Temperature

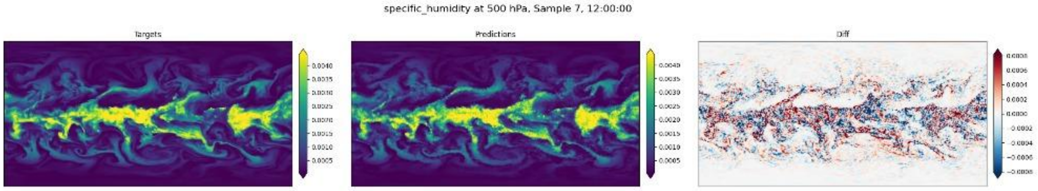


Fig 9: Specific Humidity

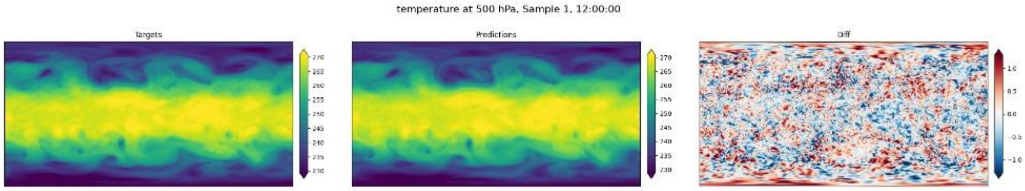


Fig 10: Temperature

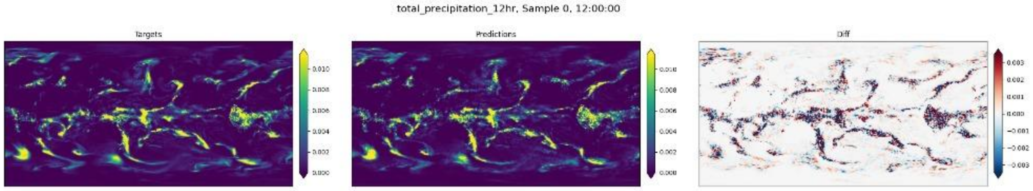


Fig 11: Total precipitation

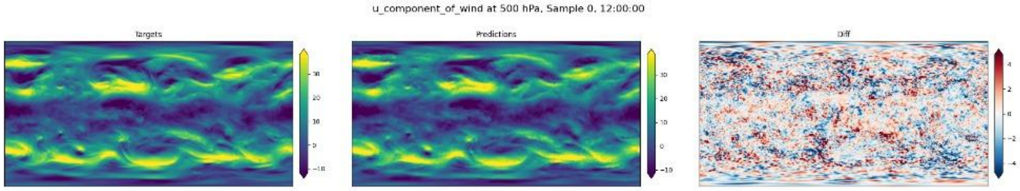


Fig 12: u component of wind

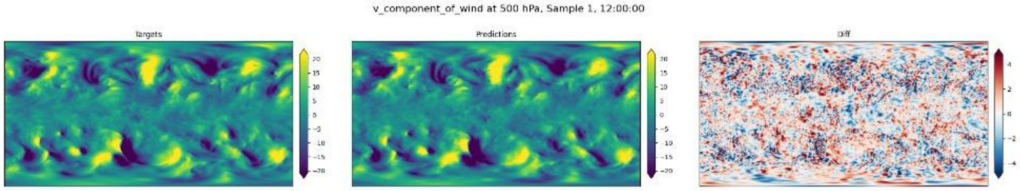


Fig 13: v component of wind

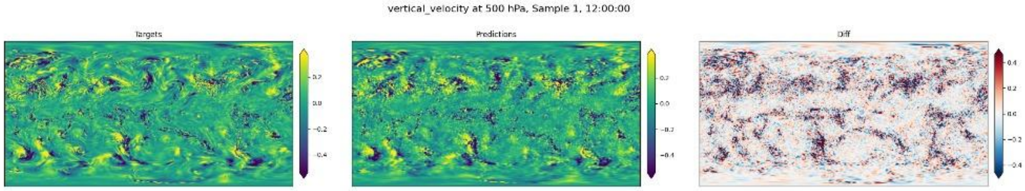


Fig 14: Vertical velocity

Table 2:

Evaluation metrics for each variable

S. no	Variable	Accuracy (%)	Loss	Mean Gradient	MAE	RMSE
1.	2m temperature	97.59	6.0487	0.015977	0.4383	0.7017
2.	10m u component of wind	97.06	6.0487	0.015977	0.5954	0.8403
3.	10m v component of wind	97.75	6.0487	0.015977	0.6085	0.8593
4.	Geopotential	96.28	6.0487	0.015977	27.3102	36.8686
5.	Mean sea level pressure	97.82	6.0487	0.015977	36.5418	51.6657
6.	Sea surface temperature	66.06	6.0487	0.015977	0.1358	0.2104
7.	Specific humidity	98.16	6.0487	0.015977	0.0001	0.0002
8.	Temperature	98.30	6.0487	0.015977	0.3430	0.4522
9.	Total Precipitation_12hr	98.46	6.0487	0.015977	0.0005	0.0017
10.	u component of wind	98.98	6.0487	0.015977	1.1648	1.5795
11.	v component of wind	98.55	6.0487	0.015977	1.2256	1.6775
12.	Vertical velocity	97.86	6.0487	0.015977	0.0953	0.1626

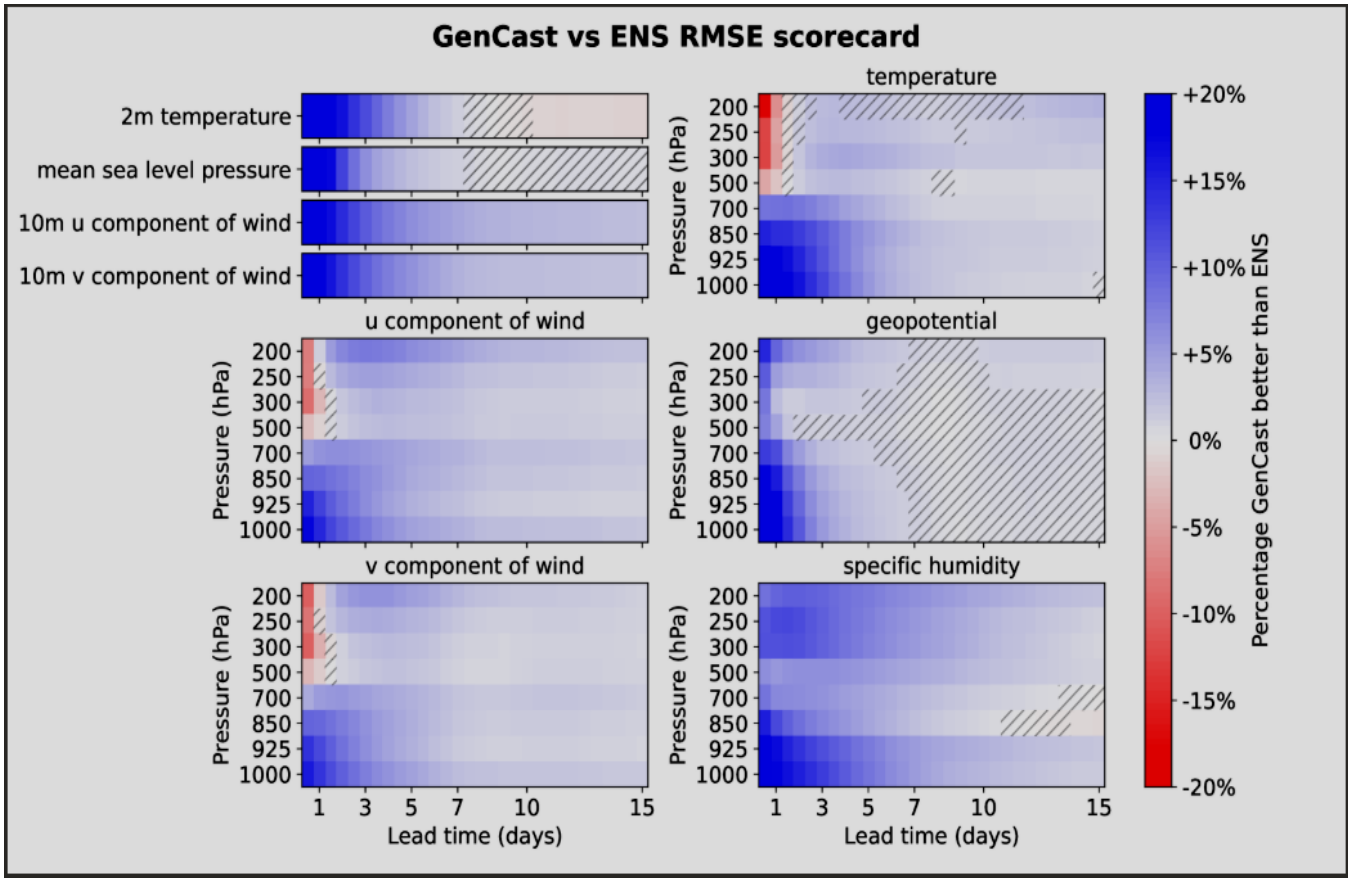


Fig 15: RMSE scorecard

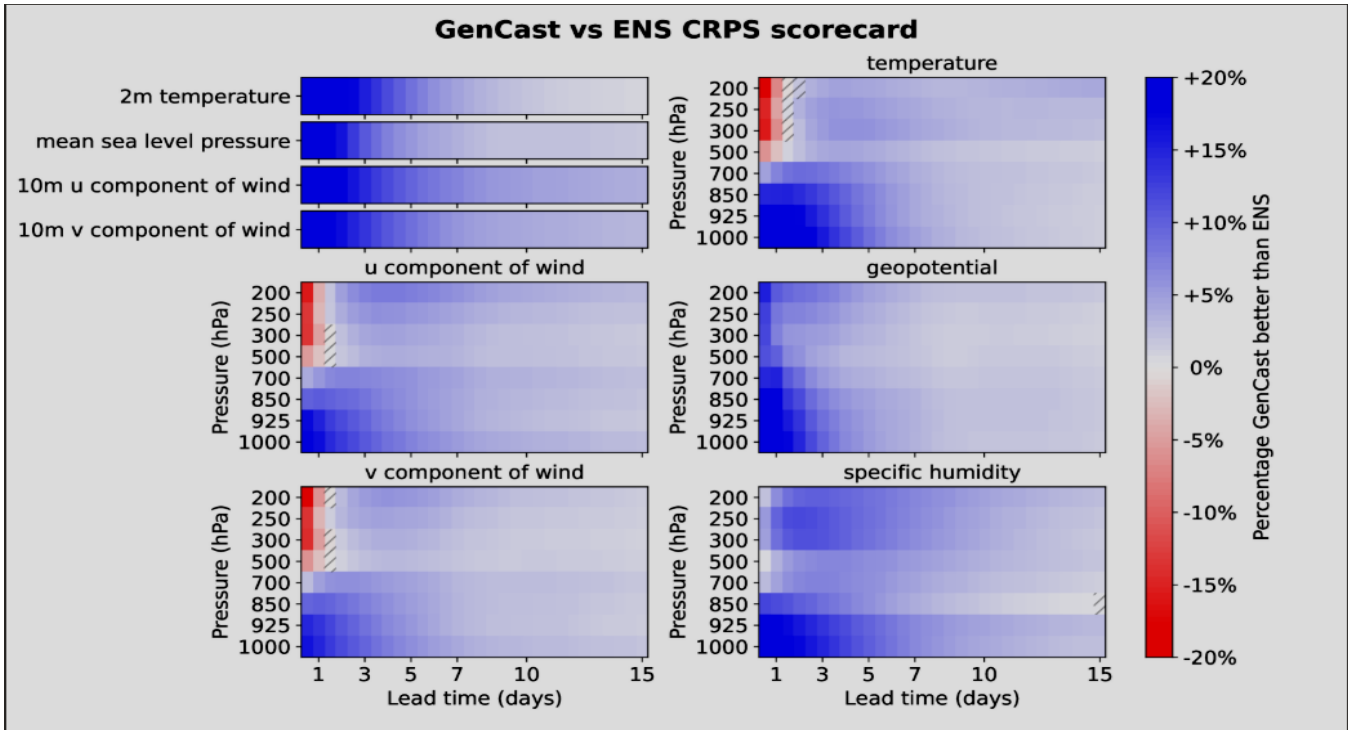


Fig 16: CRPS scorecard

Model Comparison:

Table 3: Comparison of 0p25deg and 1p0deg

Feature	0.25 deg Resolution	1.0 deg Resolution
Spatial Resolution	0.25 deg latitude-longitude grid	1.0 deg latitude-longitude grid
Detail Level	High (fine-scale patterns)	Low(large-scale patterns)
Variables Predicted	6 surface + 6 atmospheric	6 surface + 6 atmospheric
Training-Data	ERA5 at 0.25 deg Resolution	Down sampled at 1.0 deg Resolution
Computational Cost	Higher(8 minutes per forecast)	Lower(faster training)

CHAPTER 5

CONCLUSION’S AND FUTURE PLAN

Our project involves probabilistic weather forecasting with machine learning with the goal of enhancing accuracy efficiency and uncertainty estimation over numerical weather prediction models (NWP). Utilizing newer machine learning methods such as diffusion models and ensemble forecasting, our ability to make more accurate weather predictions can aid in applications such as disaster management as well as renewable power planning. The project provides the platform for future enhancements such as increased resolution models as well as real-time integration of data in order to further drive the advancement of AI-based weather forecasting.

Future Plan

1. Generating Ensemble Weather forecasts for higher degree resolution like 0.25 deg or 0.1 deg.
2. Creating region-specific GenCast model for any region in the whole world.
3. Wind power forecasting , cyclone tracking etc.

CHAPTER 6

REFERENCES

[1] Price, I., Sanchez-Gonzalez, A., Alet, F. *et al*. Probabilistic weather forecasting with machine learning. *Nature* **637**, 84–90 (2025).

[2] Ho, J., Jain, A. & Abbeel, P. Denoising diffusion probabilistic models. *Adv. Neural Inf. Process. Syst.* **33**, 6840–6851 (2020).

[3] Dunion, J. P. et al. Recommendations for improved tropical cyclone formation and position probabilistic forecast products. *Trop. Cyclone Res. Rev.* **12**, 241–258 (2023).

[4] Gielen, D. et al. The role of renewable energy in the global energy transformation. *Energy Strategy Rev.* **24**, 38–50 (2019).

[5] Ebert, E. E. Fuzzy verification of high-resolution gridded forecasts: a review and proposed framework. *Meteorol. Appl.* **15**, 51–64 (2008

CHAPTER 7

APPENDIX AND BASE PAPER

Base paper: Probabilistic weather forecasting with machine learning by using ERA5 dataset.

URL: **https://doi.org/10.1038/s41586-024-08252-9**