Understanding and Modeling Marine Fog in Areas Offshore from Atlantic Canada

by

Piyush Teeloku

A thesis submitted to the faculty of graduate studies in partial fulfillment of the requirements for the degree of

Master of Science

Department of Earth and Space Science and Engineering

York University, Toronto, Ontario

April 2025

Abstract

In recent years, machine learning (ML) has gained popularity in the field of weather forecasting, particularly in areas where numerical weather prediction (NWP) models face challenges. One such area is fog prediction. A key factor in improving fog forecasts is the accurate representation of microphysics processes, which play a crucial role in fog formation and dissipation. In this study, we propose a post-processing approach that combines the Weather Research and Forecast (WRF) model with a machine learning classifier to distinguish between fog and no-fog conditions.

This research builds on previous work conducted during the Fog and Turbulence Interactions in the Marine Atmosphere (FATIMA) project, where a 3-day marine fog forecast over the Yellow Sea was provided using WRF. The project highlighted the importance of microphysics parameterization schemes in accurately forecasting marine fog. Our findings from the Thompson microphysics schemes reinforce the need to carefully evaluate these schemes to improve fog prediction, particularly in operational forecasting.

Our study spans eleven years, from 2012 to 2023, focusing on the months from April to August each year. We utilize the WRF Preprocessing System (WPS) with initial and boundary conditions provided by ERA5 reanalysis data. The features of our ML model include 2-meter temperature, U and V wind components, 2-meter relative humidity, surface pressure, and the month, day, and hour. The target variable is hourly reported visibility

data from Navigation Canada on the Environment and Climate Change Canada (ECCC) website for the locations of St. John's, Newfoundland and Labrador, and Yarmouth, Nova Scotia in Canada.

When tested with data in 2024, our ML model demonstrates better performance compared to predictions based solely on the liquid water content (LWC) from the WRF model. We used various metrics to evaluate the classification, with the F1 score being the most important metric. By using the ExtraTreesClassifier model, we were able to obtain a 11% increase in the F1-score (0.69 vs. 0.62)and a 2% increase in accuracy (0.90 vs. 0.88) compared to the WRF model at St John's. A similar performance was noted in Yarmouth. The accuracy there increased by 4% (0.86 vs. 0.82) and F1-score by 11% (0.59 vs. 0.53). This approach shows promise in enhancing the accuracy of fog prediction, offering valuable insights for aviation, marine operations, and transportation safety.

Acknowledgements

I would like to express my deepest gratitude to my professors, Prof. Peter Taylor and Prof. Yongsheng Chen for their invaluable patience, guidance, and insightful feedback throughout this journey. Their expertise and encouragement have been instrumental in shaping this work.

I am also sincerely thankful to my supervisory committee, my co-supervisors and Prof. Gary Klaassen, for their time, knowledge, and thoughtful contributions, which have greatly enriched my research.

A special thanks to my office mates for their support, both in providing valuable feedback and in fostering an environment of collaboration and motivation. I am particularly indebted to Zheqi Chen, a PhD candidate at York University, for his collaboration on predicting fog using Machine Learning. Our discussions and brainstorming sessions laid the foundation for this project, and his insights were invaluable as I carried out the model runs and explored various approaches.

Finally, I would be remiss not to acknowledge my family, especially my parents and my partner, whose unwavering support, patience, and belief in me have been my greatest source of strength. Their encouragement has carried me through the challenges of this journey, and for that, I am profoundly grateful.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background Work

1.1.1 Marine Fog

The weather conditions for fog to occur can be quite mild, but when it happens, it can become quite disruptive due to the visibility drop, therefore aviation and marine industries are very interested in how fog will affect their journeys (Flynn (2018)). It is estimated to be the second most common cause of weather-related aviation accidents behind strong winds (National Center for Atmospheric Research (2022)). Beyond its impact on transportation, fog plays a crucial role in atmospheric and ecological processes (Korain and Dorman (2017)). It carries water droplets containing ions, aerosols, and microorganisms, which influence coastal ecosystems' hydrological, thermodynamic, nutrient, and toxicological properties. In addition, fog regulates temperature and moisture conditions, shaping vegetation patterns and forest development in coastal regions.

Air quality degradation is also a concern during fog events, particularly under stable

and stagnant anticyclonic conditions. Predicting fog requires models that account for a wide range of physical processes, from microscopic aerosols ($\sim 10^{-7}$ m) to large-scale synoptic patterns ($\sim 10^{6}$ m), which span a wide range of scales ($\sim 10^{13}$ m). The key variables influencing fog formation and persistence include aerosols and condensation nuclei, microphysical and precipitation processes, advection, synoptic conditions, air-sea interactions, ocean currents, land-sea interactions, marine inversion characteristics, and coastal topography.

From the Glossary of Meteorology (2024), fog is defined as "a hydrometeor consisting of a visible aggregate of minute water droplets suspended in the atmosphere near the earths surface. According to international definitions, fog reduces visibility below 1 km (0.62 miles). Fog differs from cloud only in that the base of fog is at the earths surface while cloud bases are above the surface." Marine fog develops when the air near the ocean surface cools to its dew point, causing condensation and the formation of tiny water droplets that reduce visibility. Several mechanisms contribute to the formation of marine fog, with the primary factor being the interaction between atmospheric moisture and oceanic temperature variations. Wind, humidity, and large-scale weather patterns further influence its development and persistence.

There are various types of fog: advection fog, radiation fog, steam fog, and upslope fog, but for this research, we are mainly focused on advection and radiation fog. From Flynn (2018), radiation fog typically forms overnight or during the early morning hours when the ground loses heat through radiation under clear skies and light winds. As the ground cools, it lowers the temperature of the air just above it. If this cooling continues to the point where the air reaches its dew point, water vapor condenses into tiny droplets, resulting in fog. This type of fog is common over land, especially in valleys where cool air can settle, and often dissipates after sunrise as the ground warms.

Advection fog forms when warm, moist air moves over a cooler surface, causing the air temperature to drop to its dew point and condense into fog. This process is driven by

horizontal air movement (advection), which continually brings warm, humid air into contact with a cooler surface, such as cold ocean waters or landmasses. It is most commonly found in coastal areas where ocean currents influence temperature. Advection fog can be persistent, lasting for extended periods if the wind continues to supply moist air. Unlike radiation fog, advection fog can occur both during the day and at night and may persist as long as the moist air continues to flow over the cooler surface.

Understanding marine fog, its formation, types, and effects is crucial for improving forecasting methods and mitigating its impacts. This report examines the occurrences of marine fog in various coastal regions (St John's, Yarmouth, Sable Island, and the Yellow Sea), exploring the factors that influence its formation and evolution.

1.1.2 Using the Weather Research and Forecasting (WRF) model

WRF is a state-of-the-art numerical weather prediction (NWP) model designed for both mesoscale meteorological research and operational forecasting (Skamarock et al. (2019)). There are two main components of WRF: a data assimilation system (WRFDA), and a software architecture supporting parallel computation and system extensibility. Due to its flexibility and efficiency, WRF is suitable for use across scales, ranging from meters to thousands of kilometers, for various research purposes, including regional climate research, data assimilation, parameterization, and some functional purposes such as simulating ideal cases and real-time numerical weather prediction.

We also used the WRF Prepossessing System (WPS), which is used for real-case simulations. The 3 main functions of WPS are to define the domains suitable for your case, interpolate the terrestrial data to the simulation domain, and interpolate meteorological input data from an outside source (for our research it was mainly the ERA5 reanalysis data and the Global Forecast System (GFS) data) to the simulating domain. WPS primarily pre-

pares the initial and boundary conditions for real-case simulations in WRF. The processed meteorological data from the WPS serves as input for the real.exe program, which initializes the real-case simulation. Additionally, WRF can also run idealized simulations using the ideal.exe program, which generates a controlled environment from predefined input files or atmospheric soundings, typically assuming simpler atmospheric processes.

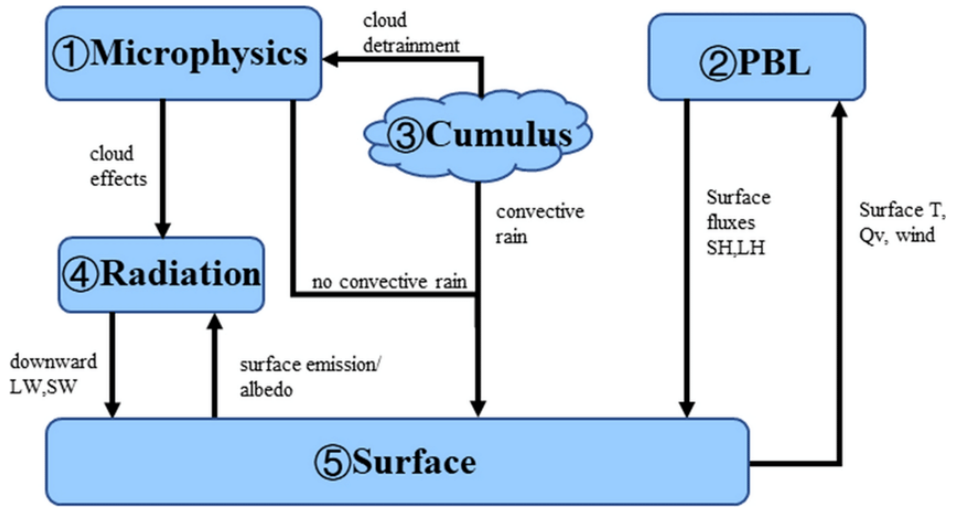


Figure 1.1: Interactions between the WRF parameterization schemes (Chen et al. (2021)) .

The WRF model incorporates various physical parameterization schemes, as shown in Fig. 1.1. Stensrud (2007) categorizes these schemes into key areas, including land surface processes, atmosphere-land interactions, water-atmosphere interactions, planetary boundary layer (PBL) dynamics and turbulence, convection, microphysics, and radiation. WRF provides multiple physics options within each category, offering a range of schemes that can be selected based on the specific needs of our research study or forecast application. The physics schemes give a comprehensive understanding of the atmospheric processes, and every choice made will change the outcome of the model's simulation, making parameterization schemes an aspect very important to consider when setting up the WRF model (Yarker Consulting (2015)).

We have used the WRF model for various research projects ranging from lake-breeze interactions in Southern Ontario to wind forecasting. We also used WRF as an operational forecasting model for marine fog over coastal regions for the Fog and Turbulence Interactions in the Marine Atmosphere (FATIMA) project. The main goal of the project was to have a much deeper understanding of the formation of marine fog over shallow seas and coasts together with a better ability to detect and predict fog (University of Notre Dame (2026)). The first time we participated in this project was during their research project on Sable Island in Nova Scotia, Canada. In the summer of 2022, they had a research cruise, the Atlantic Condor, that was equipped with various instruments relevant to fog and air quality. They collected various samples over Sable Island, which is one of the foggiest places in the world (VandenBoer (2022)).

The next year, we were more involved in the project as an operational forecaster of marine fog for the Yellow Sea project. This was the second field campaign of FATIMA with the same objective of understanding marine fog in the Yellow Sea, which is situated between mainland China and the Korean Peninsula (Lee et al. (2023)). The campaign was conducted in the summer of 2023, from June 20th to July 9th. The ship used was the R/V Onnuri of the Korea Institute of Ocean Science and Technology (KIOST) which cruised the Yellow Sea off the coast of South Korea. It played an important role as a non-stationary instrument by collecting various important variables such as liquid water in the air, precipitation rate, wind profiles, and more (Gonzalez et al. (2023)). Furthermore, time series data of ocean-atmospheric variables were collected on three Korea Ocean Research Platforms (KORS). Our role was to give fog forecasts over the Yellow Sea, follow the path of the ship, and determine fog-intensive operating periods (IOPs) using the WRF model operationally.

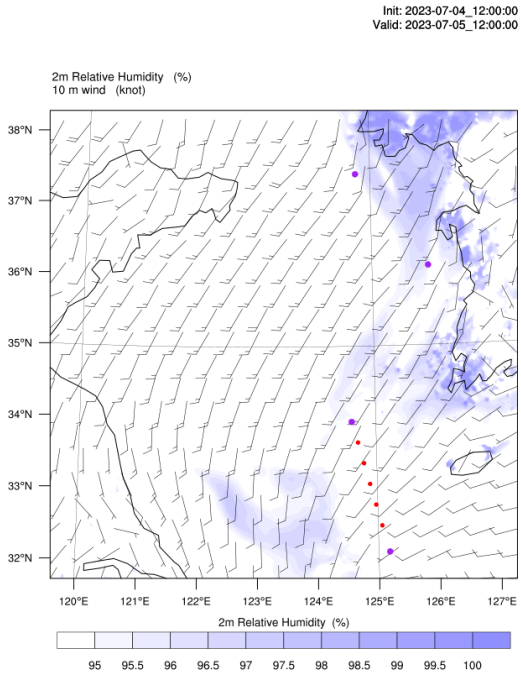
We used WRF as a 2-nested domain, with first domain (D01) having a 9 km resolution and D02 having 3 km. D01 had 150×150 grid points and D02 had 241×241 grid points. As

input data, we used the GFS forecast meteorological data and ran the simulation for 3 days and 12 hours. The first 12 hrs was spin up for the model to stabilize and adjust from the initial conditions (Liu et al. (2023)). We set the output variables interval to 1 hour, and set up a modified 34 eta levels, where there were more vertical levels at the surface for more accurate results near the surface. The physics options chosen and WRF configuration is given in Table 1.1 (-1 option for WRF means that WRF automatically chooses the suitable scheme based on the other options).

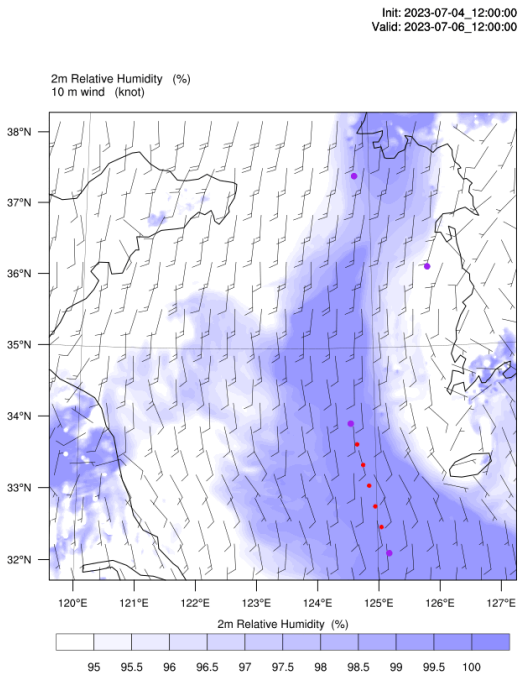
Domains	**D01**	**D02**
Resolution (km)	9	3
No. of grid points (lon $\times$ lat)	150 $\times$ 150	241 $\times$ 241
Microphysics	Thompson	Thompson
Longwave radiation	RRTMG	RRTMG
Shortwave radiation	RRTMG	RRTMG
Land surface model	Unified Noah	Unified Noah
Surface Layer model	MYNN	MYNN
Cumulus parameterization	-1	-1
Planetary boundary layer	MYNN 2.5	MYNN 2.5

Table 1.1: Configuration of the WRF simulation for the Yellow Sea campaign.

So, we held daily meetings for the FATIMA project during the month of June and July 2023 at 9 p.m. EST to give a 3-day forecasts for the ship. For the briefings, we ran the simulation and then plotted the 2m-relative humidity and the cloud mixing ratio (g/kg) for the Yellow sea in domain 2, together with 10 m wind barbs to see the interaction of fog with wind. Then, we also plotted the relative humidity against height to be able to better identify marine fog or low-level clouds. The graphs for relative humidity vs. height are from different platforms in the sea, as mentioned above, where measurements were being taken.



(a) Relative humidity at 2m plot for Yellow Sea on the 5th July, 2023 at 12:00 UTC

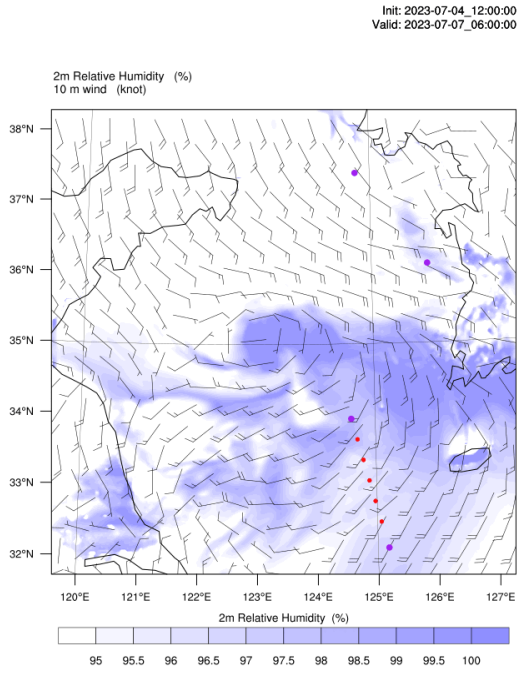


(b) Relative humidity at 2m plot for Yellow Sea on the 6th July, 2023 at 12:00 UTC

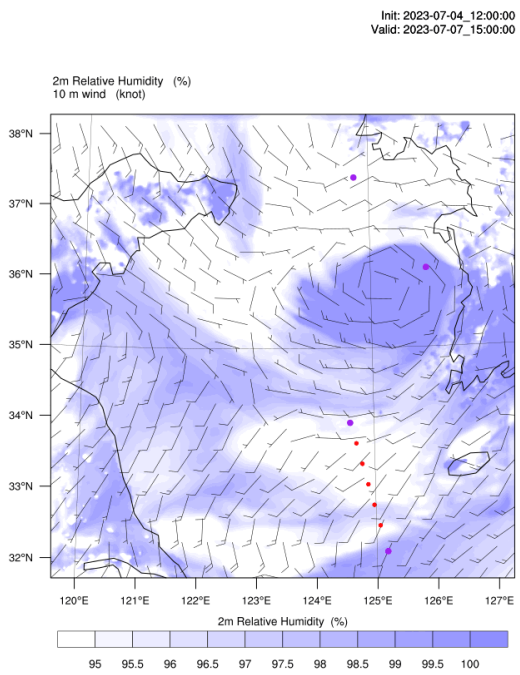
Figure 1.2: Hourly evolution of relative humidity at 2 m plot on domain 2 for Yellow Sea with wind barbs at 10 m, showing the first forecast day (5th July) and second (6th July) at 12:00 UTC.

In Fig. 1.2 to 1.3, we have an example of one of those meetings that occurred on July 5^{th}, 2023 to show how we acted as operational forecasters. In the 3-day forecast, there was a low-pressure system that came in which was then observed by the ship at the forecasted time.

In Fig. 1.2(a), we have the predictions for relative humidity at 2 m for 5^{th} July at 12:00 UTC. The forecast indicates a shift in wind direction from westerly to southerly(the figures are not shown but the winds were westerly before). Around 12:00 UTC, patches of moisture are observed along the coastlines, with relative humidity exceeding 98% in some areas. The presence of high humidity along with changing wind patterns suggests favorable conditions for fog formation near the coast.



(a) Relative humidity at 2m plot for the Yellow Sea on the 7th July 2023 at 06:00 UTC



(b) Relative humidity at 2m plot for the Yellow Sea on the 7th July 2023 at 15:00 UTC

Figure 1.3: Hourly evolution of relative humidity at 2 m plot on domain 2 for the Yellow Sea with wind barbs at 10 m, showing the third forecast day at two different times. In Fig. 1.3(a), it is at 06:00 UTC, and in Fig. 1.3(b), it is at 15:00 UTC.

In Fig. 1.2(b), we have a similar plot for 6^{th} July (the second day of the forecast) of relative humidity at 2 m at 12:00 UTC. We observed that by 12:00 UTC on 6th July, southerly winds had strengthened, transporting a moisture-laden air mass from the south to the north. This results in an extensive region of high relative humidity, increasing the likelihood of marine fog. Additionally, a developing low-pressure system approaches from the south, with intensifying wind speeds to 25-30 knots by the evening (plots are not shown), which could influence local fog dissipation and advection patterns.

Then, we showed two figures from the last day of the simulation where the low-pressure system was moving in. At 06:00 UTC on July 7th in Fig. 1.3(a), the forecast for the Yellow Sea shows a significant area of high relative humidity, exceeding 98%, indicating a high likelihood of dense marine fog. This concentrated zone of moisture is located primarily around 35°N latitude and 123°E longitude. Light winds, around 10 knots, are observed across the region, predominantly from the southwest. These gentle winds are contributing to the advection of the moist air mass northward and maintaining the fog bank's shape.

At 15:00 UTC on July 7th in Fig. 1.3(b), the forecast indicates a significant weakening and northward shift of the marine fog system in the Yellow Sea. The area of high relative humidity has diminished considerably, with the intensity of the humidity decreasing as well. The center of the high humidity zone has moved northward, now located roughly at 35°N latitude and around (125-126)°E longitude. The wind patterns remain generally light, though there may be a slight increase in speed, contributing to the dispersal of the fog. The observed dissipation of the fog is likely due to increased solar radiation and surface heating, leading to a decrease in relative humidity.

Finally, to get a good understanding of fog events relevant to the path of the ship, which for that day is denoted by the red dots, we plotted the relative humidity vs. height with time in Fig. 1.4 at the G-ORS platform which is indicated by the purple dot in the South,

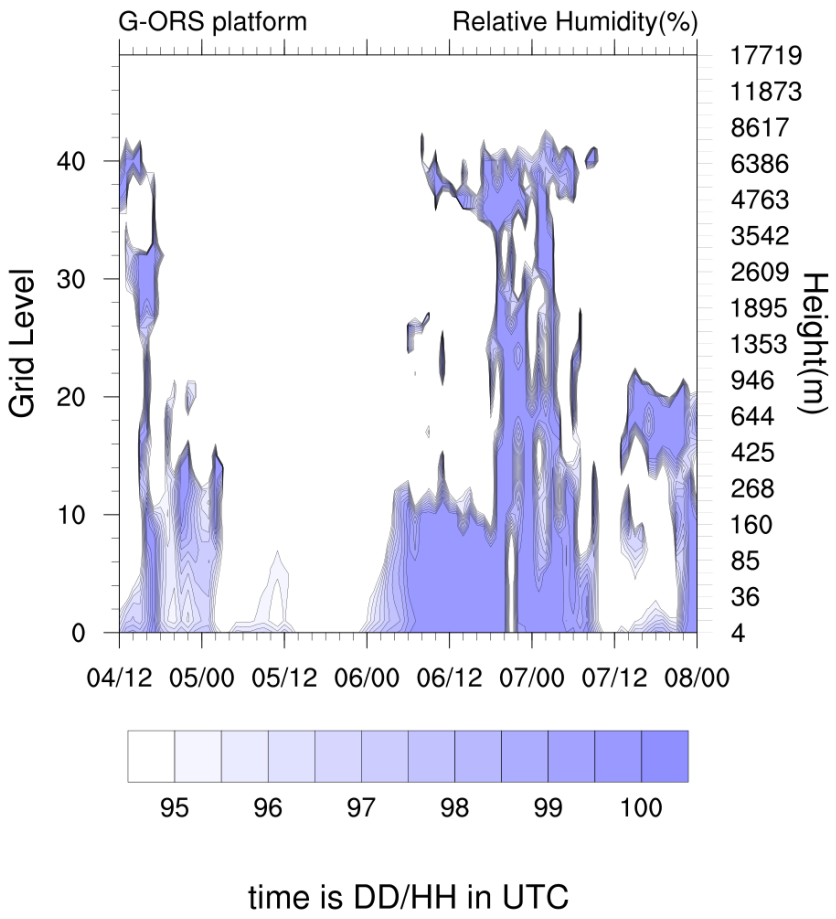


Figure 1.4: Relative humidity plot vs. Height (in both meters and vertical level) over time at the G-ORS platform, which is in the south, right above the red dots in Fig. 1.2 and Fig. 1.3.

above the red dots in Fig. 1.2 and Fig. 1.3 (around 34°N, 124.8°E). This gave us more insight into the height of the moist air and gave us a better idea if fog was present. We can see that on July 5^{th}, it was relatively dry throughout the atmospheric column. The shading is light, indicating values mostly below 95%. This suggests a dry air mass with minimal potential for fog or precipitation. July 6th exhibits a more complex pattern. The first 12 hours show fluctuating humidity levels, gradually intensifying. We see patches of higher relative humidity, particularly in the lower levels, indicating periods of increased moisture. The presence of high relative humidity extending to around 6 km suggests the potential for

cloud formation and possibly precipitation towards the end of the day. Furthermore, we see on July 7th, the early hours maintained high relative humidity, supporting the possibility of precipitation, while later in the day, a significant drying trend occurred, particularly in the lower levels, due to the advection of drier air associated with the northward movement of the observed low-pressure system. This plot helps to distinguish between periods of fog potential (high humidity at low levels) and precipitation potential (high humidity at higher levels).

The briefing for that day concluded that the following three days were unlikely to be significant for collecting ship-based fog observations. However, precautionary measures were advised due to the approaching low-pressure system and the relatively strong winds forecasted for July 6^{th}. The forecast was later validated, as the system was indeed observed moving in from the south, and no fog events occurred along the ship's path during this period.

From this project, we gained valuable insights into marine fog dynamics and the critical role of physics parameterizations in improving fog forecasts. In particular, we concluded that further investigation into the microphysics scheme, which plays a crucial role in the formation and evolution of marine fog, would be highly beneficial.

1.1.3 Investigating the Thompson Microphysics scheme

For the Yellow Sea campaign, we used the Thompson microphysics scheme (Thompson et al. (2008)). It is a single-moment scheme that only predicts the mixing ratios of five liquid and ice species: cloud water, rain, cloud ice, snow, and graupel. One thing we noticed is that the model always predicted very high relative humidity at 2 m, $> 95\%$ or there was no moisture present. This always leads to a very high chance of fog or none at all and the fog layer being very dense. In addition, Taylor et al. (2021) suggests that many weather

models, including WRF, overestimate the liquid water content (LWC) in fog layers. We are also using the same paper (Taylor et al. (2021)) to do our own research on the influence of tuning the microphysics scheme for marine fog.

Their main idea was to investigate the surface layer parametrization in WRF. Since we are investigating marine fog, one needs to take into account the sea surface in the weather model; however, this type of deposition of fog droplets to the surface is not accounted for in WRF. Taylor et al. (2021) hypothesized that the sea surface is an effective sink for the fog droplets since turbulence within the fog layer will increase the probability of the droplets colliding and coalescing with the water surface. This will decrease the LWC near the surface and effectively decrease the thickness of the layer. They decided to use the MellorYamadaNakanishiNiino (MYNN) surface scheme and they introduced a turbulent deposition velocity variable inside the MYNN scheme module in WRF. They parametrized the deposition velocity by the roughness length variable which will be dependent on the fog-cloud droplets and the equation is given below:

$$V_d = \frac{ku_*}{ln(\frac{z1+z0c}{z0c})} \tag{1.1}$$

, where k is the Karman constant (=0.4), u is the friction velocity, z0c is the roughness length and z1 is the lowest vertical level. Their addition of the turbulent deposition velocity did decrease the LWC in the layer of marine fog compared to using the original schemes. In the paper, Taylor et al. (2021) also used the Thompson microphysics scheme which deals with moisture tendencies and atmospheric heat of water vapor, cloud droplets, rain, ice, and graupels. It deals with the different processes for each of those aerosols. For cloud droplets, there will be collision and coalescence, and for water vapor, there will be condensation or evaporation which are relevant to clouds. For this project, we wanted to investigate the cloud droplet number concentration (N_c) as varying this number will impact the formation

and evolution of clouds. Therefore, we looked at the Thompson microphysics single-moment scheme and the Thompson aerosol-aware double-moment scheme as a way to investigate N_c. To fully investigate their modifications, Taylor et al. (2021) used the single-column ideal case of WRF and we decided to have the same approach.

Moreover, we will be using the WRF- Single Column model (SCM-WRF), which is an ideal case in WRF. The ideal case in WRF uses idealized inputs rather than observational data like in real cases. The SCM-WRF runs on a 3x3 domain with boundary conditions on X and Y. There is no horizontal gradient and each variable runs through the single column. The initial conditions are given in the input sounding and input soil files (Josh (2009)). The reason we chose to use the SCM-WRF ideal case is that when you run an ideal case, it has fewer variables to worry about and we can easily access the performance of any changes being made to specific physical parametrizations. We chose the single-column model because most of the physical processes mentioned above mostly interact vertically (Developmental Testbed Centre (2009)). Before doing the modifications, we needed to investigate the Thompson microphysics scheme. For this scheme, we observed that each aerosol (except for snow) follows a generalized gamma function instead of the exponential distribution.

$$N(D) = \frac{N_t}{\Gamma(\mu+1)} \lambda^{\mu+1} D^{\mu} e^{-\lambda D} \tag{1.2}$$

where N_t is the total number of particles in the distribution, D is the particle diameter, λ is the distributions slope, and μ is the shape parameter. The idea behind the Thompson microphysics scheme is that instead of doing double moment schemes, they concentrated their efforts on understanding the fault in the other previous single moment schemes and improving on them. The Thompson scheme then, being a single moment, can behave as accurately as a double-moment scheme. For the cloud droplet number concentration, Nt = Nc and it is a constant in this scheme, that is, it is preset in the code and does not vary

with time and the default value is 100 cm^{-3} which represents relatively clean air.

From Thompson and Eidhammer (2014), the Thompson Aerosol-Aware Paper, it is understood that aerosols play a role in cloud formation and microphysics through heterogeneous nucleation of cloud and ice particles. When aerosol concentration increases, liquid water content increases and will often lead to a larger number of liquid droplets, which are smaller in size which increases the cloud albedo, called the first indirect effect. The second indirect effect of having smaller cloud droplets is that rain droplets are reduced or even take more time to develop, causing a delay in precipitation and both of those effects may affect the fog layer. Different regions will have different aerosol types and distribution, which can impact some weather phenomena. For example, continental regions have relatively dry air while maritime regions have relatively moist air. Those aerosols are important for different cloud properties such as radiation, precipitation, and dynamics which in turn is important for weather applications such as fog formation and other applications sensitive to LWC or ice content.

In contrast to the older scheme, N_c is solved prognostically, that is, it is made to vary with time in the simulation and is a predicted variable. This means that the Thompson Aerosol-Aware is a double-moment scheme where the cloud mixing ratio and cloud concentration both vary with time. The equation for calculating the concentration of the cloud droplets is given below:

$$\begin{aligned}\frac{dN_c}{dt} = &- (\text{rain, snow, graupel collecting droplets}) - (\text{freezing into cloud ice}) \\ &- (\text{collide/coalesce into rain}) - (\text{evaporation}) \\ &+ (\text{CCN activation}) + (\text{cloud ice melting})\end{aligned} \tag{1.3}$$

, where we observe different processes that will either contribute to or decrease the Nc. This is why the Thompson Aerosol-Aware is a two-moment scheme for the cloud droplet whereas the Thompson scheme is a single-moment scheme.

To run the SCM-WRF ideal case, we needed the same input variables as the ones in the Surface Deposition paper. They were initialized in the input sounding file which contained the initial speed u and v in ms^{-1}, potential temperature in Kelvin, and water vapor mixing ratio in kg/kg. The input soil file had the input for skin temperature. Then, we simulated for four days, from August 15th to August 19th, 2018 by inputting the days in the namelist.input file. This file gives all the necessary inputs, from the grid sizes, and timesteps to all the relevant physical parametrizations. The MYNN surface layer scheme was used with the z0c modifications (for marine fog, z0c = 0.01 m), and no radiation schemes were used as we wanted to mainly investigate the microphysics scheme. Also, to generate fog at the surface, we introduced a 6-hour surface cooling of 3 K/hr, which decreased the skin temperature from 300 K to 282 K. After investigating the Thompson microphysics scheme, we identified that the Nc is a constant and can be changed in the code, so we changed the cloud number from 100 cm^{-3} to 75 cm^{-3} which is more suitable for maritime cases (Thompson et al. (2008)). The code block that was changed is inside the WRF and is called the module_mp_thompson.F and the important part of the code is given below:

```
!.. Prescribed number of cloud droplets. Set according to known data or
!.. roughly 100 per cc (100. E6 m^-3) for Maritime cases and
!.. 300 per cc (300. E6 m^-3) for Continental. Gamma shape parameter,
!.. mu_c , calculated based on Nt_c is important in autoconversion
!.. scheme. In 2- moment cloud water , Nt_c represents a maximum of
!.. droplet concentration and nu_c is also variable depending on local
!.. droplet number concentration.
REAL , PARAMETER , PRIVATE :: Nt_c = 100. E6
REAL , PARAMETER , PRIVATE :: Nt_c_max = 1999. E6
```

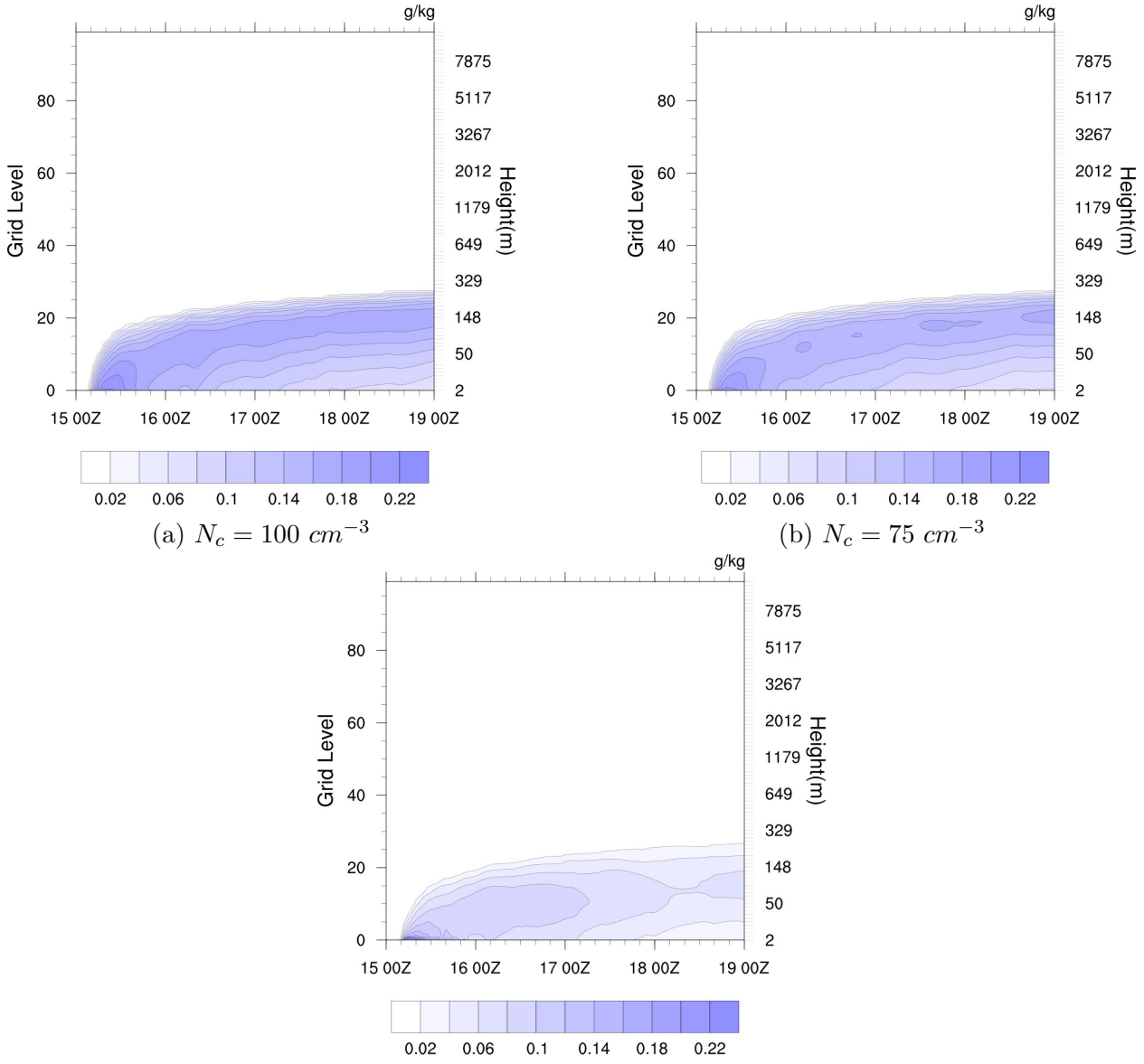
We can see that the parameter Nt_c can be easily changed to 75 cm^{-3} and the run can be recompiled for that case. Then, we change the microphysics from mp_physics = 8 to mp_physics= 28 which indicates the aerosol-aware in the namelist.input file.

In Fig. 1.5, we plotted the cloud mixing ratio in g/kg against height for the entire time of the simulation. We used the same modifications as the paper with z0c= 0.01 m. There were no radiation schemes, with the 6 hr surface cooling. We can see that fog forms in all 3 plots after the 6hr and gradually grows to around 300 m at the end of the simulation while remaining quite stratified. There are also no upper layer clouds due to turning off the radiation schemes but this simplifies accessing the performance of our changes. In Fig. 1.5(a), we have the original Thompson scheme, with $N_c = 100\ cm^{-3}$. Since the goal was to reduce the LWC, in Fig. 1.5(b), we decrease the cloud number concentration to 75 cm^{-3} and we can see that the shading (indicating the amount of Q_c) became lighter throughout the layers and the days. In Fig. 1.5(c), we used the Thompson Aerosol-Aware, the two-moment scheme, and there is a much more apparent decrease in LWC throughout the days.

When doing the comparison between Fig 1.5 (a) and 1.5 b), it is quite hard to distinguish the decrease in LWC content (there is minimal change in the shading), so we decided to calculate the LWC decrease for each day to fully understand the contribution of changing the cloud droplet number and the Thompson Aerosol-Aware scheme. To be able to calculate the percentage decrease for each day, we simply used the percentage decrease formula on the cloud mixing ratio given below:

$$\%decrease = \frac{Qc_{Nc100} - Qc_{\text{Nc(75 or aerosol-aware)}}}{Qc_{Nc100}} \times 100 \tag{1.4}$$

, where Qc is the cloud mixing ratio. We extracted the eta levels (vertical levels in the model) for each day until the LWC was 0 and then plugged in the values for Qc_{Nc100} and



(a) $N_c = 100\ cm^{-3}$

(b) $N_c = 75\ cm^{-3}$

(c) Thompson Aerosol-Aware scheme

Figure 1.5: Plotting Q_c in g/kg against Height (m) for 4 days with a 6 hr surface cooling at the start of the simulation. Fig 1.5 a) The Thompson scheme with N_c= 100 cm^{-3}, b) The Thompson scheme with N_c= 75 cm^{-3} and c) The Thompson Aerosol-Aware scheme which has a varying N_c.

Qc_{Nc75}. Then, we calculated the percentage decrease at each vertical level for one day and did an average for that day. This was done for the 16th, 17th, 18th, and 19th and the results are given in Table 1.2.

% Decrease	**24hr**	**48hr**	**72hr**	**96hr**
Maritime Case(N_c= 75 cm^{-3})	9.43	9.01	5.98	5.56
Thompson Aerosol-Aware	50.3	49.9	57.6	59.6

Table 1.2: Table of mean percentage decrease for each day comparing original with N_c=75 cm^{-3} for the Thompson scheme and with the Thompson Aerosol-Aware scheme.

From Table 1.2, we see that in the Maritime case ($N_c = 75\ cm^{-3}$), there is a 9% decrease in LWC on the first two days and a 6% decrease on the last two days. For the Thompson Aerosol-Aware, there is a 50% decrease on the first two days and a 60% decrease on the last two days. Now, that we verified that the LWC did decrease by changing the cloud droplet number concentration, the next objective is to find the implications of the changes in the real world. We decided to calculate the visibility inside the fog layer and do a comparison between the three cases. We decided to calculate the visibility for each day again at 00:00 UTC. To be able to calculate the visibility, we used the daytime visibility equation based on the "Characterizing and Predicting Marine Fog Offshore Newfoundland and Labrador" paper by Isaac et al. (2020). We used the daytime visibility equation for simplicity, given below:

$$V_k = \frac{1.24\rho_w^{2/3}}{LWC^2 N^{1/3}} \tag{1.5}$$

, where V_k is the visibility in m, ρ_w is the density of water, LWC is the cloud mixing ratio and N is the cloud droplet number concentration. To calculate visibility, we chose to use LWC at the lowest eta level which corresponded to z $\approx$ 1.6 m. The cloud droplet

concentration, for the original case and the maritime case, was set as a constant, and that number was simply used for the calculations. However, for the Thompson Aerosol-Aware (N_c was around 20 cm^{-3}), as the cloud droplet number was varying with time, we got the values for each day from the WRF output as it is a predicted variable and plugged that value into equation 1.5.

	Visibility/ m			
	24hr	**48hr**	**72hr**	**96hr**
Original Case (N_c= 100 cm^{-3})	80.7	99.6	121.8	140.9
Maritime Case (N_c= 75 cm^{-3})	94.5	116.1	137.5	164.5
Thompson Aerosol-Aware	224.8	235.8	401.4	491.9

Table 1.3: Table containing visibility at 00:00 UTC for each day of the simulation at height z= 1.6 m for each case (original, when N_c is changed to 75 cm^{-3} and Thompson aerosol-aware).

Across all three cases, visibility remained below 1 km, confirming fog conditions. In the original case ($N_c = 100\,\mathrm{cm}^{-3}$), visibility started at 80.7 m at 24 hours and gradually increased to 140.9 m by 96 hours. The maritime case ($N_c = 75\,\mathrm{cm}^{-3}$) showed slightly higher visibility, beginning at 94.5 m and reaching 164.5 m by the final day, showing an increase of 24.4 m in visibility. In contrast, the Thompson Aerosol-Aware case exhibited a much more pronounced increase in visibility. Compared to the original case, at 24 hr, the visibility was 225 m which is an increase of around 150 m. This scheme showed the same gradual increase of visibility throughout the days, reaching around 492 m at 96 hr. This is due to the significant decrease in liquid water content (LWC) compared to the other cases ($\approx$ 20 cm^{-3} vs. 100 cm^{-3}). This increased visibility by nearly 350 m. Given that typical fog visibility is in the range of 100 to 1000 m, the Thompson Aerosol-Aware scheme may provide a more accurate depiction of the fog layer.

Therefore, by changing the cloud droplet number concentration to be more suitable for

a maritime case in the Thompson microphysics scheme, that is changing the N_c from 100 cm^{-3} to 75 cm^{-3}), we were able to decrease the cloud mixing ratio or LWC by 7%. This increased visibility by approximately 20 m. When we changed the Thompson microphysics scheme to its newer version, the Thompson Aerosol-Aware microphysics scheme, which has N_c varying with time, we saw the LWC decrease by 55%. The resulting increase in visibility was around 300 m.

1.2 Literature review

While using the NWP model such as WRF has been instrumental in understanding the microphysics and evolution of fog, accurately predicting fog occurrence has been challenging for many weather models (Taylor et al. (2021)). According to Lam et al. (2023), traditional NWP models have experienced a steady increase in accuracy due to a better understanding of atmospheric physics and higher computing power. However, this method can be slow and computationally expensive and does not improve with the amount of historical data available. Alternatively, machine learning-based weather prediction (MLWP) can use a vast amount of historical data while remaining cost-effective. Therefore, MLWP has gained a lot of interest in the atmospheric science field, which is used to improve the prediction or forecasting of different weather phenomena. Furthermore, many modeling studies experienced similar difficulties when researching marine fog (Chen et al. (2020)). As mentioned in Korain et al. (2014), although sea fog modeling has gained popularity since WWII, accurately predicting when sea fog will begin and how long it will last has typically been unsuccessful.

There have been various NWP approaches to modeling and predicting marine fog. As mentioned earlier, Taylor et al. (2021) introduced a deposition velocity in the surface layer physics scheme in WRF. They ran the 3D WRF model with the roughness length for North

Atlantic during the summer 2018 on a domain extending from eastern Canada out beyond the Grand Banks and including Sable Island. They were able to have a better LWC representation as well as visibility measurements compared with observations. However, they do not guarantee that this modifications improves the prediction of the onset and dissipation of fog. More recently, Gao et al. (2024) used another NWP approach where they used microwave radiometer-retrieved cloud water path observations to improve numerical simulations of sea fog over the Yellow Sea in May 2018. They used an ensemble method with WRF and a Grid-point Statistical Interpolation/EnKF assimilation system. Then, they had more accurate initial conditions by retrieving the cloud water content from satellite. By doing so, they were able to have improvements on sea fog forecasting. Their method worked very well with areas of high chance of fog and had an average false alarm rate of 0.463, meaning that they were overestimating the number of fog occurrence by roughly 46%. So, we are still dealing with the same problem that the NWP are overestimating the LWC and doing a poor job at predicting fog occurrence.

Another approach to improve fog prediction is to use machine learning and AI techniques. Therefore, we decided to take an alternative approach by using both WRF and Machine Learning to improve the hourly forecast of fog. Early efforts in fog prediction can be traced back to the 1980s, when researchers explored linear regression models for this task, though with limited success (Koziara et al. (1983)). In response to the shortcomings of traditional approaches like linear models, machine learning (ML) techniques have gained traction over the past decade for tackling fog prediction challenges. For example, Fabbian et al. (2007) applied a multi-layer perceptron (MLP) to forecast fog events at Canberra International Airport using meteorological data. Their neural network model, trained and tested with data from the Australian Bureau of Meteorology, showed promising performance. They used eight meteorological features including visibility and obtained decent results for 3 hour forecasting. Bartokov et al. (2015) improved nowcasting of coastal fog in the desert area

of Dubai for up to 6 hours using decision trees and showed improvements compared to the WRF model and the PAFOG fog model. They further increased the accuracy by including the output of the numerical fog forecasting models in the training database of the decision tree.

Colabone et al. (2015) investigated the use of multi-layer perceptrons (MLPs) trained with back-propagation to forecast fog events at the Academia da Força Area in Brazil. Their study focused on evaluating how well MLPs could learn from meteorological data to identify fog occurrence, and they demonstrated encouraging results in capturing fog conditions in that setting. Boneh et al. (2015) applied a Bayesian network model to forecast fog at Melbourne Airport, structuring the problem around an 8-hour prediction window. Leveraging 34 years of historical meteorological observations, the authors trained the network and found it to outperform the existing operational forecasting system. As a result, their model was adopted for real-time fog forecasting at the airport. Cornejo-Bueno et al. (2017) explored fog forecasting at Valladolid Airport in Spain by testing a range of ML regression algorithms. Since radiation fog is the dominant type in that region, their study focused exclusively on winter months when such events are most frequent. Among the methods evaluated, Support Vector Regression (SVR) and Extreme Learning Machines (ELM) showed particularly strong performance in predicting fog events. Durán-Rosal et al. (2018) examined the classification of fog events using neural networks trained on meteorological input features. They experimented with various activation functionsincluding sigmoidal, product, and radial types, and employed a multi-target learning strategy. Their results indicated that these networks were highly effective in accurately classifying fog occurrences.Guijo-Rubio et al. (2018) addressed the challenge of forecasting low-visibility events caused by fog using ordinal classification methods. Their model categorized events into three levels (fog, mist, and no fog). They tested several ordinal classifiers for this task. The results demonstrated the potential of these approaches in distinguishing between varying degrees of visibility reduction.

More recently, Miao et al. (2020) evaluated the performance of a Long Short-Term Memory (LSTM) neural network for fog prediction up to 4 hours in Anhui, China. They treated the task as a classification problem, leveraging LSTM's ability to model temporal dependencies in meteorological time series data. The study demonstrated the potential of deep learning for fog classification in complex atmospheric conditions. FogNet, proposed by Kamangir et al. (2021), is a deep learning-based framework developed for fog prediction that utilizes a multiscale 3D Convolutional Neural Network (CNN) architecture, enhanced with a double-branch dense block and an attention mechanism. This design enables the model to effectively capture both spatial and temporal dependencies from high-dimensional meteorological data. The study focused on data from Mustang Beach Airport in Port Aransas, Texas, covering the period from 2009 to 2020. To construct the input features, the authors used forecasts from the North American Mesoscale (NAM) model alongside local weather station observations. Their approach involved post-processing the visibility outputs from NAM using the station data, enabling fog predictions up to 24 hours in advance. The results demonstrated the models superior performance in visibility forecasting compared to conventional methods.

Castillo-Botn et al. (2022) focused on low-visibility event prediction using data from the Mondoñedo weather station in Galicia, Spain. They applied both regression and classification approaches to model visibility conditions. They collected visibility data for 23 months, from 1st January 2018 to 30th November 2019 and evaluated both ML techniques to provide nowcasting of fog in the next 30 minutes. Their work highlighted the benefits of using supervised learning techniques for both continuous and categorical forecasting of visibility reduction events. Pelez-Rodrguez et al. (2023) extended previous work on low-visibility forecasting at the Mondoñedo weather station in Galicia, Spain (the same site used in the Castillo-Botn et al. (2022) study) by incorporating ERA5 reanalysis data alongside local observations. They explored the use of both machine learning models and evolutionary

algorithms to enhance the prediction of fog-related visibility reductions. The study focused on short-term forecasts of 1hr, 3 hr and 6 hr, showing a 17% improvement.

More closely related to our marine fog research, Gultepe et al. (2024a) applied machine learning techniques to nowcast marine fog visibility using observational data collected during the FATIMA campaign in July 2022. The campaign took place in the Grand Banks region and near Sable Island off the northeast coast of Canada, using measurements from the Research Vessel Atlantic Condor. Key meteorological variables were used alongside derived droplet number concentration to analyze fog characteristics. Nowcasting was performed for lead times of 30 and 60 minutes using support vector regression (SVR), least-squares boosting (LSB), and deep learning models, targeting visibility thresholds below 1 km and 10 km. Their results emphasized the value of ship-based observations and ML models in supporting short-term fog forecasting in complex marine environments. Another sea fog ML approach was proposed by Chen et al. (2024)) developed a deep learning framework to improve the short-term prediction of sea fog regions in the Yellow Sea, with an emphasis on maritime navigational safety. The study introduced the Multivariable Sea Fog Forecast (MV-SFF) dataset, which contains 122 sea fog events from 2010 to 2020. Each event includes hourly geostationary satellite images and reanalysis-based meteorological variables, providing a rich data source for spatial fog forecasting. Using this dataset, the authors proposed the Rich-Element Aggregated (REA) modela deep learning architecture designed to extract and integrate diverse meteorological and satellite-derived features across different times and spatial scales. The REA model includes a position-aware edge detection mechanism to accurately localize sea fog regions. A seven-hour ahead forecast starting from 09:16 local time demonstrated strong performance, with the REA model outperforming other state-of-the-art deep learning methods in both accuracy and consistency. This work highlights the potential of combining remote sensing with deep learning to enhance spatial sea fog forecasting capabilities.

In this study, binary classification was chosen as the machine learning framework to post-process numerical output from the WRF model at St John's and Yarmouth due to its proven effectiveness in prior research. The binary approach was used to identify fog events (typically defined by visibility below 1 km) while accounting for the inherent variability in visibility measurements. From other research, we observed that using ERA5 data for the meteorological features helped increase the performance of the ML approaches and compared to other research that used features to complement their ML models, we used a post-processing approach. Many existing studies have employed classification techniques to distinguish between fog and no-fog conditions and have reported reliable performance in identifying fog onset and duration. Classification is also particularly advantageous when the primary operational need is to issue timely alerts for fog events, rather than estimating precise visibility values. Our approach builds on this by integrating WRF model forecasts with site-specific visibility observations to improve local fog prediction.

Unlike many previous works that focus on very short-term forecasts (e.g., 30 to 60 minutes or up to 6 hour lead time), our method extends the forecast horizon to 24 hours and provides predictions at an hourly resolution, offering greater lead time for decision-making. This also enables us to identify when during the day fog occurs and how long it can last compared to using only time windows of 3 hr, 6 hr and 12 hr. Additionally, by leveraging a high-resolution regional NWP model like WRF, our method benefits from better meteorological context, particularly in capturing key physical drivers of fog. While the model does not resolve the full areal extent of fog, particularly in complex coastal terrain, the combination of WRF-based physical guidance and site-specific ML classification offers an optimal balance between accuracy, interpretability, and operational utility for localized fog forecasting.

Chapter 2

Methodology

2.1 Collecting data

A key factor in successfully applying machine learning techniques is having access to large amounts of high-quality data. As noted by Google For Developers (2024g), ML practitioners often spend significantly more time collecting, preprocessing, and transforming data than on using the models themselves to generate results. In our case, since we are feeding substantial weather data into our model to detect instances of fog, we employ a supervised learning approach. Supervised learning is ideal for well-defined tasks (Google For Developers (2024e)), and we opted for the classification method. Classification, a type of supervised learning, aims to categorize data into predefined classes (Google For Developers (2024g)). For our purposes, we use both binary classification (fog or no fog) and multi-classification (fog, mist, and clear). We opted for binary classification because it is important to identify the occurrence of fog accurately and this method reduces the variability in predicting visibility. If we were predicting visibility directly for example, we would need to get accurate predictions of visibility on top of correctly getting the fog events (Visibility below 1.2 km). This increases

the chances of missing fog and a higher percentage of error (this is explored in the discussion chapter).

Regarding data collection, we divide our dataset into two parts: simulating the features and collecting the target (visibility). The features are generated using the Weather Research and Forecasting (WRF) model with a two-nested domain and the WRF Preprocessing System (WPS).

2.2 Target

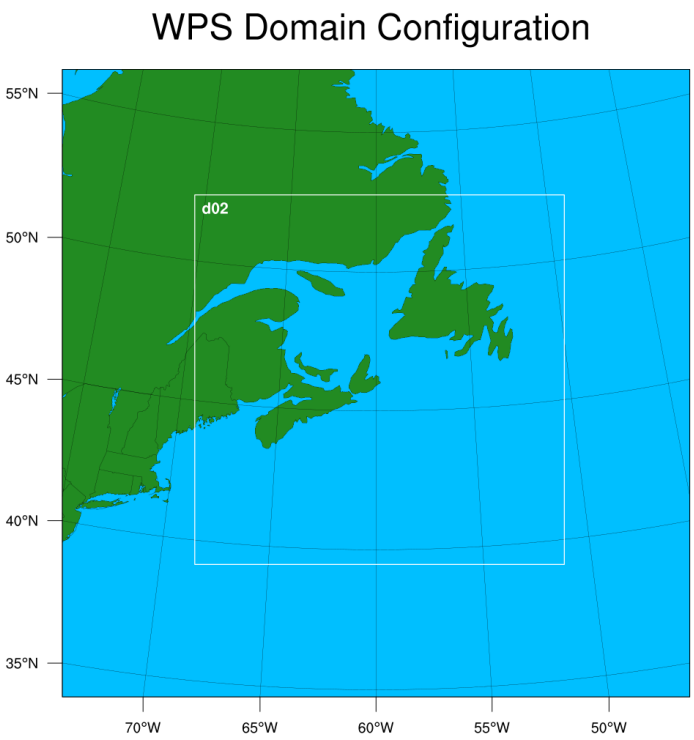
The target is visibility in kilometers measured at weather stations. From Figure 2.1b and 2.1c, we see two locations where the weather stations are located. We will show results for St John's, Newfoundland and Labrador, and Yarmouth, Nova Scotia. The St John's weather station is located at latitude 47°37'07.000" N and latitude 52°45'09.000" W with an elevation of 140.5 m, and the Yarmouth weather station is located at latitude 43°49'37.000" N and latitude 66°05'17.000" W with an elevation of 42.9 m. We also tried to use Sable Island weather station data. For the summer of 2022, the FATIMA campaign was at the Sable Island and it was a location of interest for us because it is known as the foggiest location on Earth. However, due to poor observations from that weather station (a lot of missing data), we omitted the results. The weather stations can use an Automated Weather Observation System (AWOS) to measure visibility hourly (NavCanada (2017)). AWOS calculates visibility using sensors at approximately 3 m in height. The sensors have a transmitter and a receiver about 1 m apart that measure infrared pulses to obtain visibility readings (Iowa Department of Transportation (2008)). The visibility reports can also be reported by qualified human observers and are under the METAR and SPECI category. They basically report every hour daily and are listed as METAR (H24). The weather stations both

report various surface observations either using instruments or humans every hour, including visibility (Environment and Natural Resource (2024)).

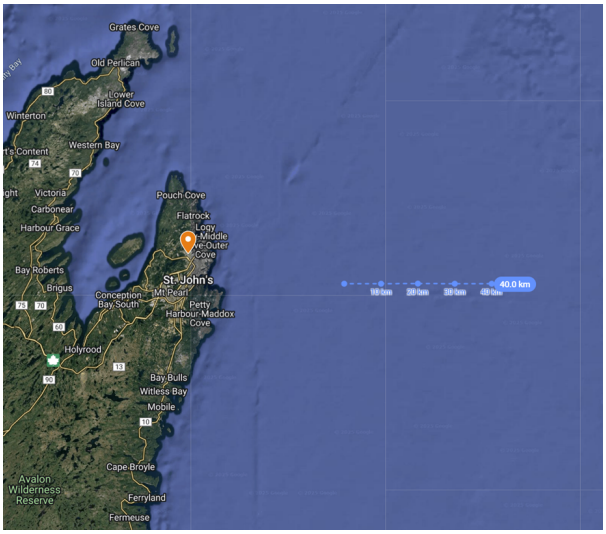
As seen from the weather stations map (Fig. 2.1), these sites are influenced by marine air masses and complex coastal terrain, both of which are key contributors to fog formation. Both St John's and Yarmouth are situated along the Atlantic coast of Canada, where the interaction of major ocean currents, that is, the warm Gulf Stream and the cold Labrador Current, creates ideal conditions for sea advection fog. Advection fog occurs when warm, moist air travels over a cooler surface (sea surface), and the air cools, causing condensation and therefore fog (NavCanada (2024)). In this region, warm, humid air masses transported by the Gulf Stream often move over the colder waters carried southward by the Labrador Current (Bullock et al. (2016)). This temperature contrast is particularly pronounced during the spring and early summer months, when sea surface temperatures remain low while atmospheric temperatures begin to rise. As a result, dense fog frequently develops offshore and is advected inland by prevailing winds, especially under stable marine layer conditions. The convergence of these ocean currents is therefore a critical driver of marine fog formation and plays a central role in the fog climatology of both coastal sites.

This makes St John's and Yarmouth particularly prone to frequent fog occurrences compared to other coastal regions (Canavan and Sanford (2007)). The unique oceanographic and atmospheric conditions along this part of the Atlantic coastline result in higher fog occurrence, drawing scientific and operational interest. As a result, these locations are especially valuable for studying marine fog processes and developing predictive tools, such as high-resolution modeling and machine learning frameworks, to improve fog forecasting in regions with significant maritime and aviation activity.

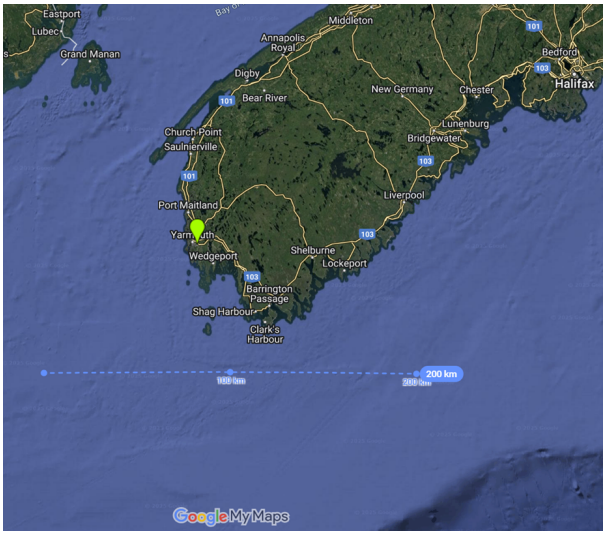
According to Glossary of Meteorology (2024), fog is defined as visibility at the surface dropping below 1 km. So we decided to collect visibility data from the NAVCAN weather



(a) WRF domain setup



(b) Locations of the St John's weather station (orange pin)



(c) location of the Yarmouth weather station (green pin)

Figure 2.1: (a) Two-nested WRF domain configuration with 27 km and 9 km resolutions. (b) Google Maps view of St. John's and (c) for Yarmouth (Google (2024)).

reports on the ECCC historical weather website. We used data from 2012 to 2023 and only selected summer (April to August) as it avoids complications with snow/ice and we are looking mainly at advection fog in the coastal areas where the highest frequency of fog at these regions is during the summer. When looking at the visibility reports, we noticed that there was a clear distinction between 1.2 km and 1.6 km. No visibility were reported at 1.4 km at all for both locations, and we concluded that by classifying fog at 1.2 km instead of 1 km, we would not lose valuable fog information. We decided to classify fog when Vis $\leq$ 1.2 km. For binary classification, it was fog when Vis $\leq$ 1.2 km and clear $>$ 1.2 km. For multi-class, fog is Vis $\leq$ 1.2 km, 1.2 km $<$ mist $\leq$ 5 km, and clear is $>$ 5 km.

2.3 Features

We chose features that are relevant to marine fog occurrence. From Gultepe et al. (2024b), they tested ten different variables and performed a pairwise correlation analysis that was visualized on a heat map. So, we did the same process with a Pearson correlation analysis for temperature, dewpoint temperature at 2 m, relative humidity at 2 m, wind components U and V at 10 m, and surface pressure. We also tried dewpoint depression (T2-TD2), which was also mentioned in the article. Figure 2.2 shows how the chosen predictors were correlated with visibility. From Wikipedia contributors (2024c), the Pearson Correlation Coefficient (PCC) is defined as the ratio between the covariance of two variables and the product of their standard deviations, meaning that it is a normalized measure of covariance between two data sets. The analysis results will range from -1 to 1, with -1 having a negative correlation and 1 having a positive correlation. The equation for a given population and 2 selected variables, X and Y, is given below:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{2.1}$$

, where σ_X and σ_Y are the standard deviation of X and Y, respectively. cov(X,Y) is the covariance of X and Y and is calculated using equation 2.2.

$$cov(X,Y) = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)] \tag{2.2}$$

, where μ_X and μ_Y are the mean of each variable and $\mathbb{E}$ is the expectation which is the generalization of the weighted average.

PCC gives a linear correlation between two variables and we have tried various variables that WRF offered. According to Ratner (2009), for a feature to be relevant, the correlation should be greater than 0.3. This indicates a moderately positive or negative linear relationship according to the fuzzy-firm linear rule. It is worth mentioning that the heat map will not capture any other type of relationship and so, some features may have a higher correlation than shown but we used the PCC as a benchmark to proceed with the feature selection. In Fig. 2.2, we have plotted the heatmap for both locations, St John's and Yarmouth. We looked at the correlation of the features with visibility and there are some differences between both locations, suggesting differences in weather patterns, which will be investigated more in the data exploration section. For Yarmouth, all the selected features are highly correlated to visibility with the exception of U at 10 m. Surface pressure and temperature and dewpoint temperature at 2 m are relatively close to 0.3 showing moderate correlation. Dewpoint depression together with relative humidity at 2 m shows a strong correlation. For St John's, the correlation is not that strong with the temperature variables, except for dewpoint depression, however, U and V at 10 m show a moderate correlation. Surface pressure is similar to Yarmouth and again strong correlation with relative humidity.

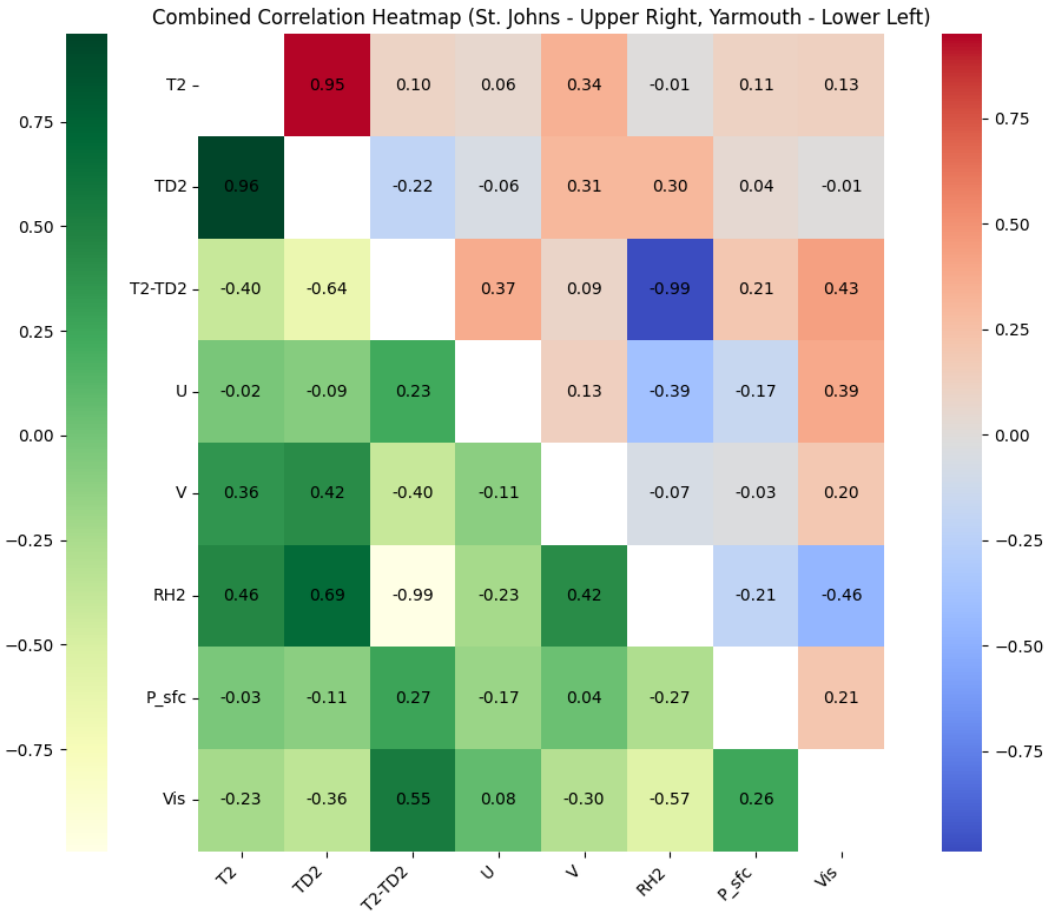


Figure 2.2: Correlation heatmap (Bothma (2024)) of the features relevant for fog prediction using the ERA5 data from 2012 to 2023 for St John's, Newfoundland and Labrador on the upper right in cool warm colors and Yarmouth, Nova Scotia on the lower left in green colors.

Even if the correlation for temperature at 2 m is not as strong, we decided to use it and did not use dewpoint temperature and dewpoint depression due to its high correlation with relative humidity for both locations. For example, at St John's, dewpoint depression has a correlation of -0.99 with relative humidity.

We also explored different variables but ended up not using them. For example, instead of wind components at 10 m, we used wind speed and wind direction. We also tried to use the components of the wind direction in the north-south and east-west and calculated the temperature advection, the dewpoint temperature advection, and the pressure advection. In

addition, we calculated the advections for temperature, pressure, and dewpoint temperature at the lowest vertical level of the model, which was 4 m. Since the results were calculated using variables like temperature, surface pressure, and wind components U and V at the lowest level, there was a high correlation with the wind components at 10 m, as well as with the temperature and dewpoint temperature at 2m and pressure. Therefore, we chose only features that were sufficiently independent from each other.

We decided to collect the data in two ways. Firstly, we ran WRF from 2016 to 2023 using the GFS Final Analysis (National Centers for Environmental Prediction, National Weather Service, NOAA, U.S. Department of Commerce (2015a)) data as input. As the GFS Final Analysis data are only available as of May 2015, we started from 2016. The GFS final analysis data are on a 0.25-degree by 0.25-degree grid and are available every six hours. The Global Forecast System (GFS) data and Final Analysis (FNL) data are similar in the sense that they are run through the same model but the FNL data are initialized three hours later, allowing about 10% more observational data to be integrated. As the data have a relatively coarse resolution, around 25 km, we ran WPS in a 2-nested domain to down-scale the grid size to 9 km. This is because the size of the fog can vary in size from a few hundred meters to 30-50 kilometers (Dorman et al. (2022)). The domain setup is shown in Fig. 2.1(a).

2.3.1 Method with GFS initial data

We ran the WRF model v4.5.2 continuously from March to August for each year (2016 to 2023) and so, ran the model 8 times. We started the simulation 12 hours before (spin-up time), allowing the model to stabilize and adjust from the initial conditions (Liu et al. (2023)). The physics options for the configuration of the WRF simulation are given in Table 2.1 (WRF Users Page (2024)). As visibility data were collected hourly, we collected WRF output variables hourly, which are set up in the namelist.input file.

Domains	D01	D02
Resolution (km)	27	9
No. of grid points (lon × lat)	91 × 91	160 × 160
Microphysics	Thompson Aerosol-Aware	Thompson Aerosol-Aware
Longwave radiation	RRTMG	RRTMG
Shortwave radiation	RRTMG	RRTMG
Land surface model	Unified Noah	Unified Noah
Surface Layer model	MYNN	MYNN
Cumulus parameterization	Tiedtke	Off
Planetary boundary layer	MYNN 2.5	MYNN 2.5

Table 2.1: Configuration of the WRF model for the yearly 6-month simulation from March to August, 2016 to 2023.

Since this is a long simulation (around 6 months), we use the spectral nudging option in WRF. From Reen (2016), spectral nudging is a form of Newtonian relaxation where extra tendency terms are added to slowly push the model towards the GFS data during long runs such that the results remain coherent, and since it is applied continuously through the simulation, it is a type of four-dimensional data assimilation (FDDA). Another important option to use for long simulations is the Sea Surface Temperature (SST) update. This is because the WRF model does not predict sea surface temperature, which remains constant throughout the run. Although this is good for short runs, for our domain and the length of the simulation, this will lead to a higher margin of error. So, from WRF-ARW Online Tutorial (2024), to implement the varying SST, we need to ungrib the SST input data and interpolate to our domain, and then run metgrid. In doing so, the SST input data varied every 6 hours for the entire run.

2.3.2 Method with ERA5 initial data

There are a few key differences between the GFS and the ERA5 initial data. GFS is run on the Global Data Assimilation System (GDAS), which collects observational data, and

ERA is the 5th generation of the European Center for Medium-Range Weather Forecasts (ECMWF) reanalysis data. Their purpose varies in the sense that GFS is used primarily for forecasting. It provides weather forecasts several times a day, extending up to 16 days in advance. ERA5 reanalysis data combine past observations with the ECMWF model to provide a consistent historical record of atmospheric, oceanic, and land variables from 1940 to the present (Hersbach et al. (2020)). So, ERA5 is used primarily for climate studies, research, and understanding historical weather patterns. They also differ in spatial resolution. As mentioned, GFS has a horizontal resolution of 25 km and has 26 vertical levels, while ERA5 has a resolution of 31 km and 137 vertical levels. The way the data are assimilated is slightly different as ERA5 uses 4D-Var assimilation and combines model results and observations. ERA5 data are also available every hour compared to every 6 hours. This provides a slight advantage compared to GFS as we can just run the WPS to get the features we need, and thus getting the data much faster.

The reason why we tested two different ways is to understand the quality of our data and also to understand if there are any significant differences between the models. Again, ERA5 data, which are available every hour, provide a great advantage in terms of computation cost and efficiency. Also, ERA5 data are available from 1940 compared to GFS FNL which are available from May 2015. This allowed us to use all the weather stations visibility data that were available as of 2012 and so having an extra four more years of data to use. ERA5 is also considered the most accurate set of data for historical weather and climate analysis due to its consistency and quality of the data across a wide range of variables. For example, from Ibebuchi et al. (2024), ERA5 outperformed other reanalysis data sets when evaluating apparent temperature in the USA. According to Malakar et al. (2020), the GFS analysis data provided a more accurate representation of the evolution of tropical cyclones in the North Indian Ocean compared to ERA5. Therefore, depending on the specific context or region, one data set may be more appropriate. For our study, we tried both cases with the

ML models and concluded that ERA5 provided more consistent and slightly more accurate results than GFS FNL with WRF. However, this may be due to the use of more data with the ERA5 data and the quality might be better. So, we will show results mainly using the ERA5 reanalysis data.

2.4 Method: Data Cleaning and Data Exploration

2.4.1 Cleaning the data

Since we obtained data for the features from WRF and WPS, cleaning the data was rather simple. We stored all the data from ERA5 and visibility in a CSV (Comma-Separated Values) file. This made it easier to read and work with the files in Python. The main step in the cleaning was to remove all the data where visibility was missing from the weather reports. We also tried to account for rain as it also affects visibility. However, we decided to leave it as is due to the ambiguity of finding a threshold cut-off for the precipitation rate.

2.4.2 Data Exploration

Another essential part of data analysis is understanding and visualizing your data. Tukey (1977) introduced Exploratory Data Analysis (EDA) as he believed that statistical conclusions relied too heavily on hypothesis testing. EDA emphasizes exploring and visualizing the data to allow the data itself to guide which hypotheses should be tested. Instead of relying solely on traditional measures like the mean and standard deviation, EDA often uses more robust statistics such as the median and quartiles to summarize the dataset. In addition, EDA incorporates graphical techniques to help visualize and understand the structure of the data (Hsieh (2023)).

	St John's	St John's	Yarmouth	Yarmouth
Variable	**Skewness**	**Kurtosis**	**Skewness**	**Kurtosis**
T2	-0.0088	-0.8972	-0.5476	-0.4803
RH2	-0.8997	0.2082	-0.8594	0.0589
U	-0.1690	0.0237	-0.3048	0.8032
V	-0.4240	-0.0572	-0.2611	-0.0507
P_sfc	-0.5982	0.8659	-0.3619	0.9389

Table 2.2: Skewness and Kurtosis for the predictors using ERA5 data from 2012 to 2023, April to August at St John's and Yarmouth.

Table 2.2 contains the skewness and kurtosis values for each variable that we use for the ML model. Skewness and kurtosis give us an indication of how the data are distributed. Skewness indicates how the tails of the distributions are and kurtosis indicates what shape is the distribution, relative to the normal distribution, and how tall is the peak (Tuychiev (2023)). The equation for skewness and kurtosis is given below:

$$Skewness = \frac{3(\text{mean} - \text{median})}{\text{Standard Deviation}} \tag{2.3}$$

$$Kurtosis = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - mean}{StdDev}\right)^4 - 3 \tag{2.4}$$

, where x_i denotes the individual data point and n is the number of observations in the dataset (Tuychiev (2023)).

For T2 (temperature at 2 m), we have a skewness of -0.0088, which is close to 0, indicating that both tails are almost similar, with the left tail slightly longer. RH2 (Relative Humidity at 2 meters) has a moderate negative skew (-0.8997), meaning its distribution has a longer left tail, with a kurtosis of 0.2082 indicating a shape close to normal. U (U-component of wind at 10 m) shows a slight left skew (-0.1690) and nearly normal kurtosis (0.0237), while V (V-component of wind at 10 m) has a mild left skew (-0.4240) and a kurtosis (-0.0572)

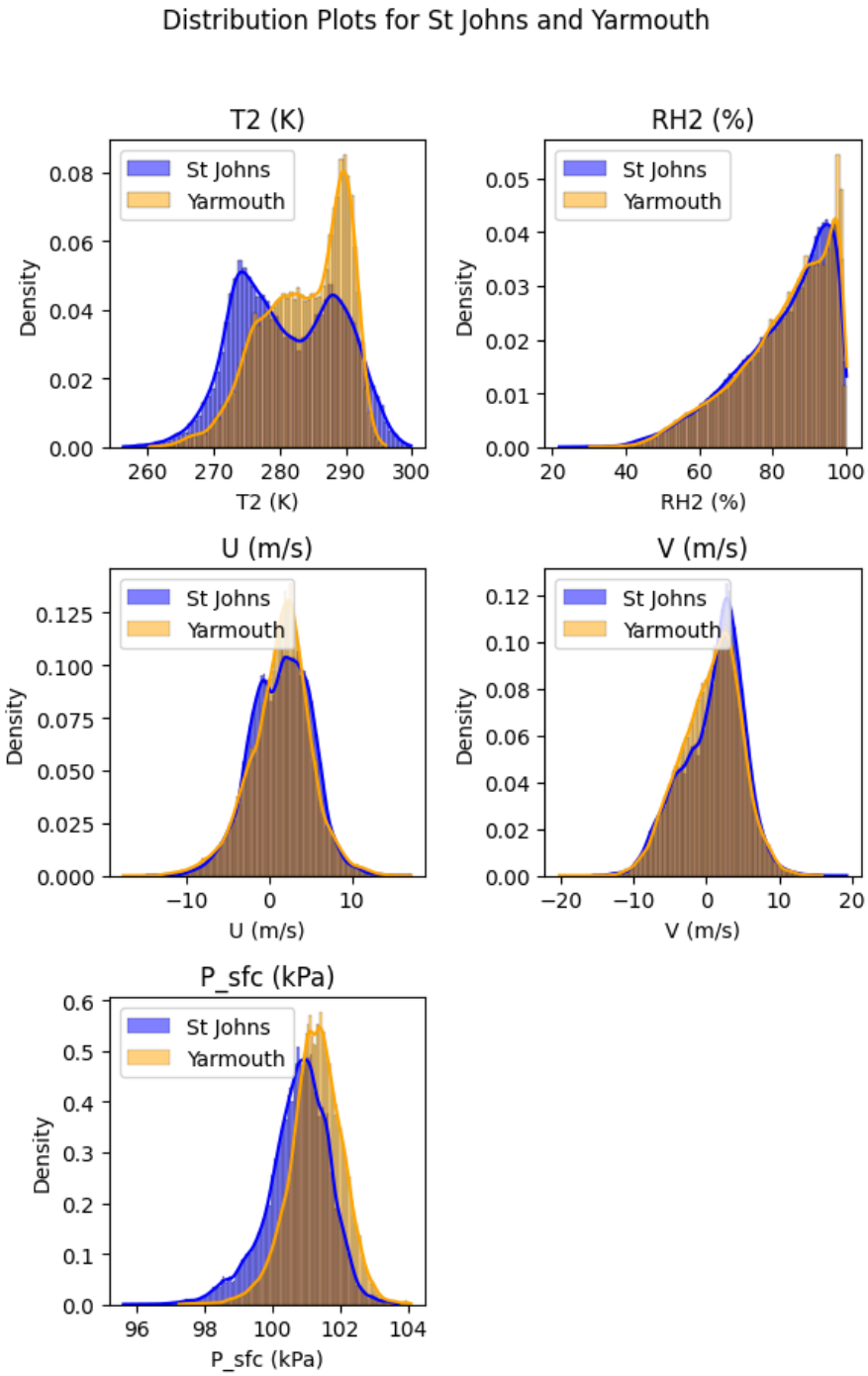


Figure 2.3: EDA: Histograms representing distribution plots for T at 2 m, RH at 2 m, V and U at 10 m, and surface pressure for St John's and Yarmouth, Canada (March to August 2012-2023).

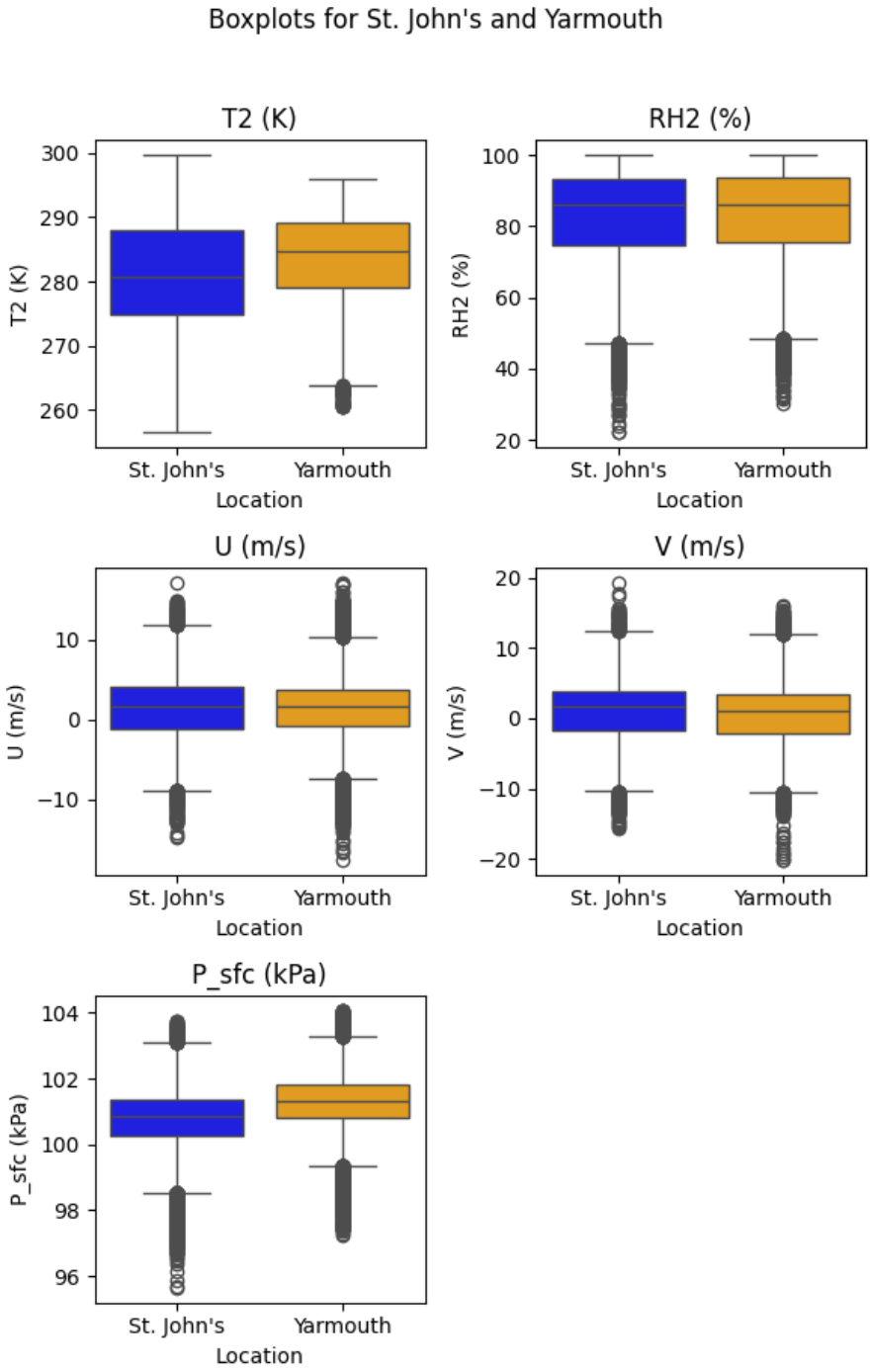


Figure 2.4: Comparison of the boxplots for T at 2 m, RH at 2 m, V and U at 10 m, and surface pressure for St John's and Yarmouth, Canada that indicate the five-number summary, March to August 2012-2023. From Beyer (1981), the five-number summary is the median, the lower and upper quartiles (i.e. q0.25 and q0.75), and the minimum and maximum values.

close to zero, suggesting a nearly normal distribution. Lastly, P_{sfc} (Surface Pressure) has a moderate left skew (-0.5982) and a positive kurtosis (0.8659), implying a more peaked distribution with potential outliers. Overall, the skewness values indicate a slight left tilt in the distributions, while the kurtosis values suggest that most variables have tail heaviness close to that of a normal distribution.

The next step in our analysis is to visualize the data using both histogram distribution plots and box plots. Histograms, as introduced by Pearson and Henrici (1895), are effective in visualizing the probability distribution of a variable by plotting the frequency of occurrences against its domain (Hsieh (2023)). From Fig. 2.3, we observe that the temperature at 2 meters for St. John's has a bimodal distribution, indicating two dominant temperature ranges. In contrast, Yarmouth's temperature distribution is more unimodal, centered around a slightly higher temperature range. The box plots in Fig. 2.4 further support this observation, showing that St. John's experiences a broader range of temperatures, with a lower median than Yarmouth, which has a slightly higher and more tightly grouped temperature range. When examining relative humidity (RH2), the histogram shows that the data for both locations are skewed to the right, with a concentration near 100%. This suggests consistently high humidity levels in both regions. However, the box plots highlight that Yarmouth has a slightly wider interquartile range, suggesting more variability in its humidity levels compared to St. John's. Both locations display significant outliers (not in a meteorological sense but rather statistically) in the lower humidity range, but St. John's has a noticeably higher number of low-humidity outliers. The wind components (U and V at 10 meters) reveal interesting contrasts between the two locations. The histograms for both wind components indicate near-normal distributions, but the V-component at St. John's has a longer left tail, indicating more instances of extreme negative wind speeds, which is also visible in the box plots as a greater number of outliers on the negative end of the distribution. On the other hand, Yarmouth shows a more symmetric distribution for both components,

with fewer outliers. Finally, surface pressure (P_{sfc}) shows a fairly symmetric and normally distributed pattern at both locations, as indicated by the histograms. The box plots reveal that St. John's has a marginally wider range of surface pressure values, while Yarmouth's distribution is more compact. However, both locations have a notable number of outliers at the lower end of the surface pressure spectrum, with St. John's showing a more spread-out distribution.

In summary, these visualizations provide valuable insight into the meteorological differences between St. John's and Yarmouth. Although the temperature and humidity distributions show similar trends, Yarmouth generally experiences slightly more consistent conditions. St. John's, however, demonstrates more variability, particularly in wind and surface pressure, which could be attributed to local environmental factors. These comparisons highlight the importance of visual tools in understanding the weather patterns of a region and enable us to better prepare our data for the ML model.

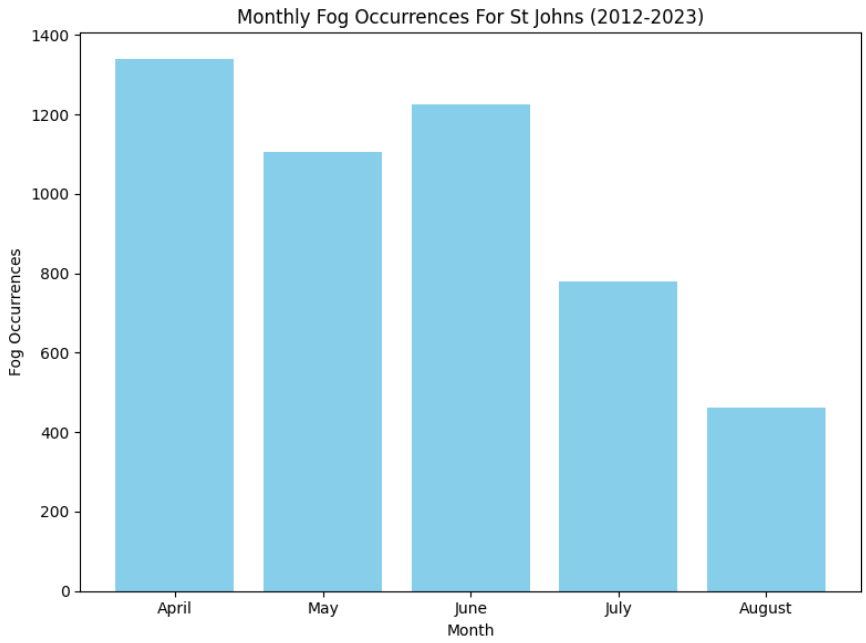
2.4.3 Types of Fog

Now that we have looked more deeply at our features, the next step is to understand the visibility data that we collected. As mentioned above, we have collected visibility data hourly, from weather stations at each location. The dataset is from 2012 to 2023, from March to August. Then we classified fog when visibility dropped below 1.2 km. So, to understand how fog evolved in those areas, we plotted the fog occurrence monthly for all the years combined on a bar chart. In Fig. 2.5, we have the plot for St John's and Yarmouth. We observe in Fig. 2.5(a) that, for St John's, fog occurs more frequently during early summer, with the highest count in April. We have the highest number of fog in the first three months, around 1100 to 1300 and it drops by almost half in July and August. However, from Fig. 2.5(b), we see that the behavior of fog is almost the opposite for Yarmouth. Fog occurs mostly at the end

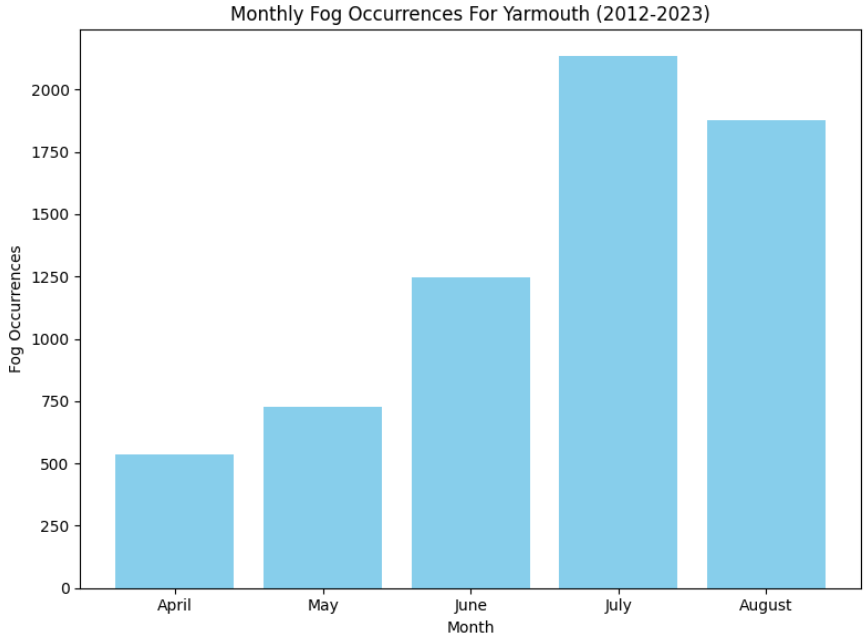
of summer with the highest count in July, around 2200 followed by August. Similarly to St John's, fog occurrence is almost halved in the other month, and we notice almost a mirror image of the St John's plot. We also notice that fog occurs more frequently in Yarmouth overall, although it might not always be the case yearly. This further highlights the difference in weather between the two locations we chose.

With the data we have collected, we can determine what type of fog those locations are producing, that is, either radiation or advection fog. There are many different types of fog, but we will mainly focus on radiation and advection. Haby (2024) describes both types of fog. Radiation fog typically forms during the night and early morning hours, usually after sunset and just before sunrise. It forms when the ground cools rapidly through radiational cooling, causing the air near the surface to reach its dew point and become saturated. Radiation fog will dissipate from daytime heating by mid-morning. This is due to surface air warming by conduction and becoming unsaturated. Radiation fog usually dissipates. In contrast, advection fog occurs when warm, moist air moves (advects) over a cooler surface, causing the air to cool and reach its dew point. This type of fog can occur at any time of the day or night but is more common when there is a significant temperature difference between the air and the surface, often due to the movement of air masses. So, radiation fog tends to form during the nighttime and early morning when cooling conditions are favorable, whereas advection fog can form at any time when moist air is transported over a cooler surface. Advection fog usually dissipates when the winds increase (turbulent mixing) or the cooler air warms up.

So, using the data we collected, we can better understand what type of fog occurred more frequently by plotting the joint probability density functions (PDFs) of two variables. Joint PDFs estimate the likelihood of two variables co-occurring. Wikipedia contributors (2024b) describes joint PDFs as the marginal distribution and the conditional probability

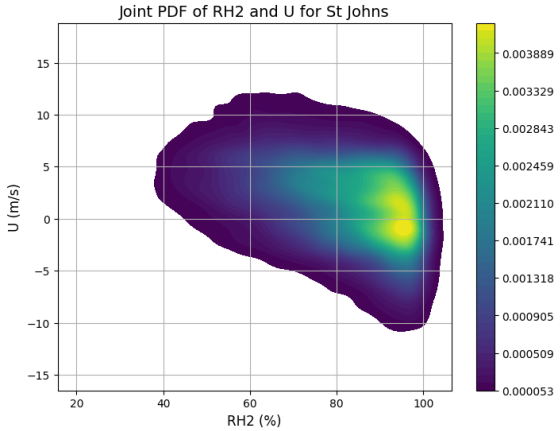


(a) Hourly fog occurrences count for each month, from 2012 to 2023 for St John's, Canada.

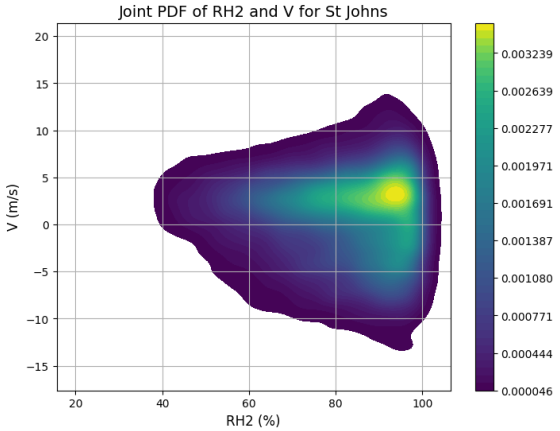


(b) Hourly fog occurrences count for each month, from 2012 to 2023 for Yarmouth, Canada.

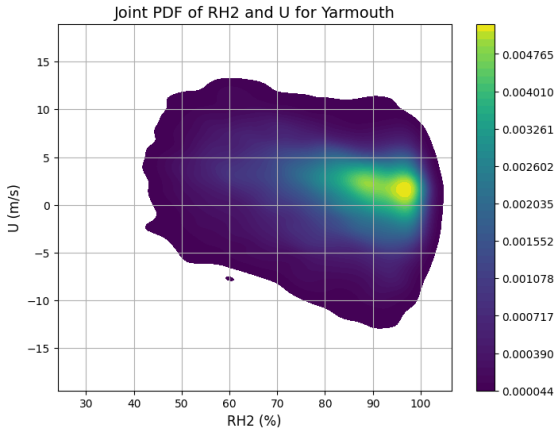
Figure 2.5: Hourly fog occurrence classified monthly for each location. In Fig. 2.5(a), we have the fog count for St John's, with the peak month being April. In Fig. 2.5(b), we have the fog count for Yarmouth, with the peak month being July.



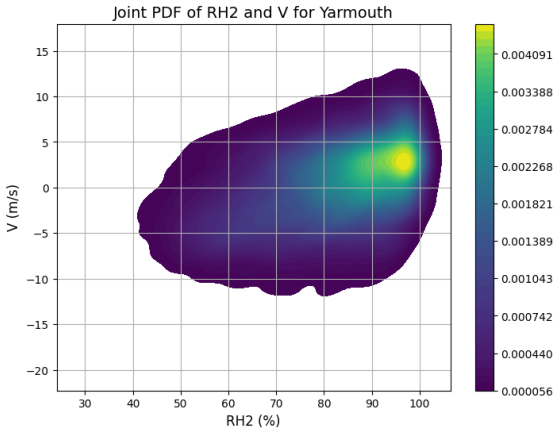
(a) Joint probability density between zonal wind speed U at 10 m and relative humidity at 2 m (RH2) for St John's.



(b) Joint probability density between meridional wind speed V at 10 m and relative humidity at 2 m (RH2) for St John's.

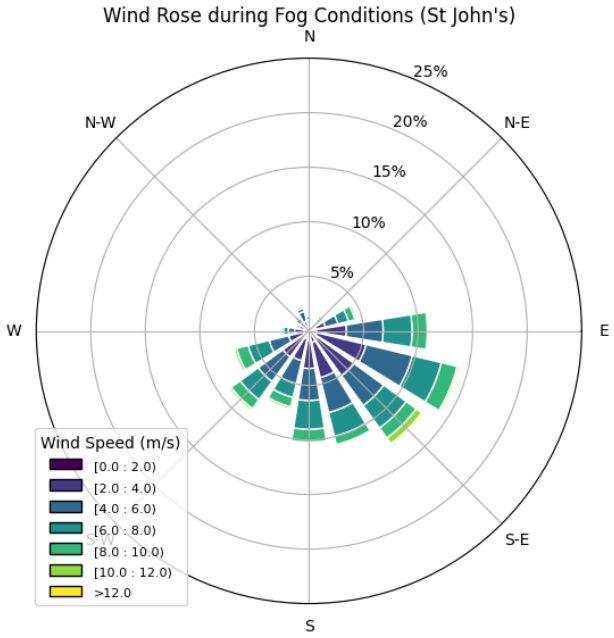


(c) Joint probability density between zonal wind speed U at 10 m and relative humidity at 2 m (RH2) for Yarmouth.

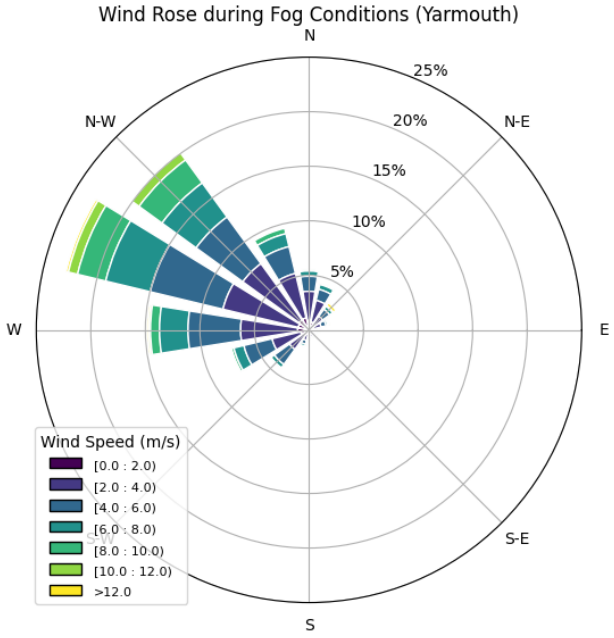


(d) Joint probability density between meridional wind speed V at 10 m and relative humidity at 2 m (RH2) for Yarmouth.

Figure 2.6: During foggy hours, we plotted joint probability distribution of zonal and meridional winds U and V at 10m with relative humidity at 2 m for both locations, St John's and Yarmouth, using April to August ERA5 data from 2012 to 2023. This gives insight into which type of fog is occurring.



(a) Wind rose plot during foggy hours for St John's, Canada.



(b) Wind rose plot during foggy hours for Yarmouth, Canada.

Figure 2.7: During foggy hours, we plotted wind rose plot with wind speed and its direction at 10 m on a polar plot with the radial axis indicating the percentage of wind data in a direction for St John's and Yarmouth, from 2012 to 2023.

distribution of random variables, which means that it shows the distribution of the random variables and how they depend on each other. For fog types, we looked at the zonal and meridional winds at 10 m co-occurring with relative humidity at 2 m. As described above, the high moisture level, together with the magnitude and direction of each wind component, will give us information on the fog that is occurring. We used kernel density estimation (KDE Wikipedia contributors (2025b)) to get a smoothed-out probability density estimation for the variables. It is a function from the scipy.stats library in Python and the equation is given below:

$$\widehat{f}_h(x,y) = \frac{1}{nh_xh_y}\sum_{i=1}^{n} K_x\Big(\frac{x-x_i}{h_x}\Big)K_y\Big(\frac{y-y_i}{h_y}\Big) \tag{2.5}$$

, where

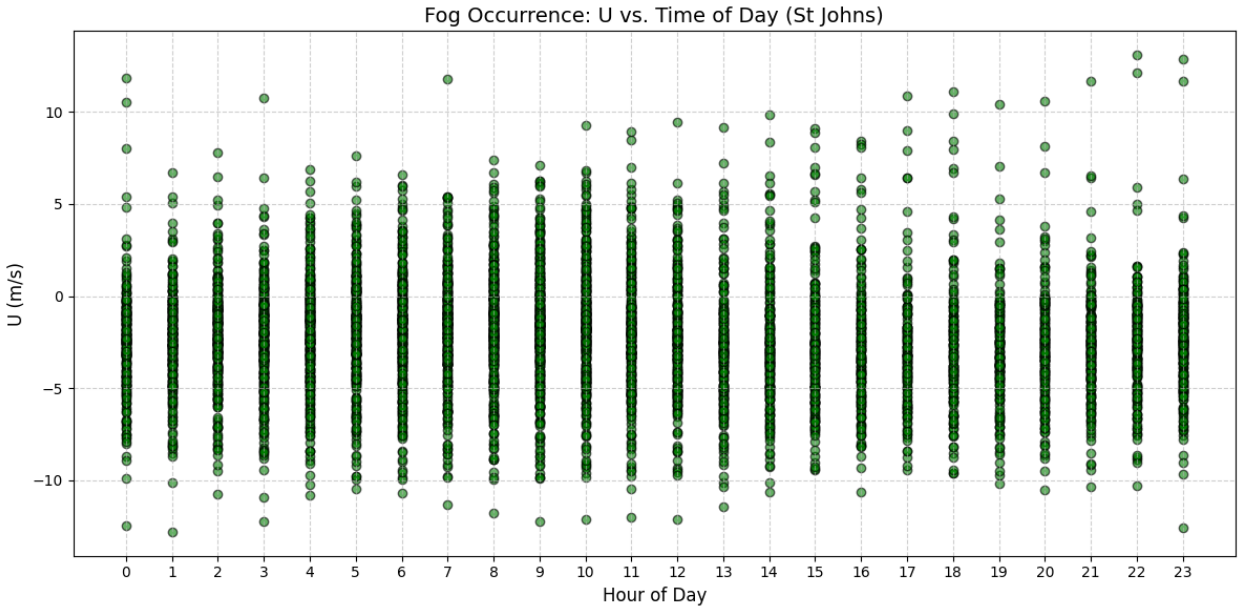
$$K(u) = \frac{1}{2\sqrt{\pi}}\exp^{\frac{-u^2}{2}} \tag{2.6}$$

The function $\hat{f}_h$ gives the Gaussian KDE, n is the number of data points, x_i and y_i are the data points for each variable, K_x and K_y are the Gaussian kernels to estimate the joint PDFs given in equation 2.6, and h is the smoothness parameter. In Fig. 2.6, we plotted the joint PDFs of the zonal and meridional wind components at 10 m with relative humidity at 2 m for St John's and Yarmouth. We also plotted in Fig. 2.7, the wind rose plots showing the wind direction and wind speed distributions at each locations. The wind speed bins represent the magnitude of the wind with the size representing the amount of wind data present. The wind direction is where the wind is coming from and is given by the polar plot which is divided in 8 segments (north, north-east, east, south-east, south, south-west, west, north-west). On the radial axis, we have the percentage of wind data present at that particular direction. To get a more in-depth analysis, we plotted the fog occurrence count in

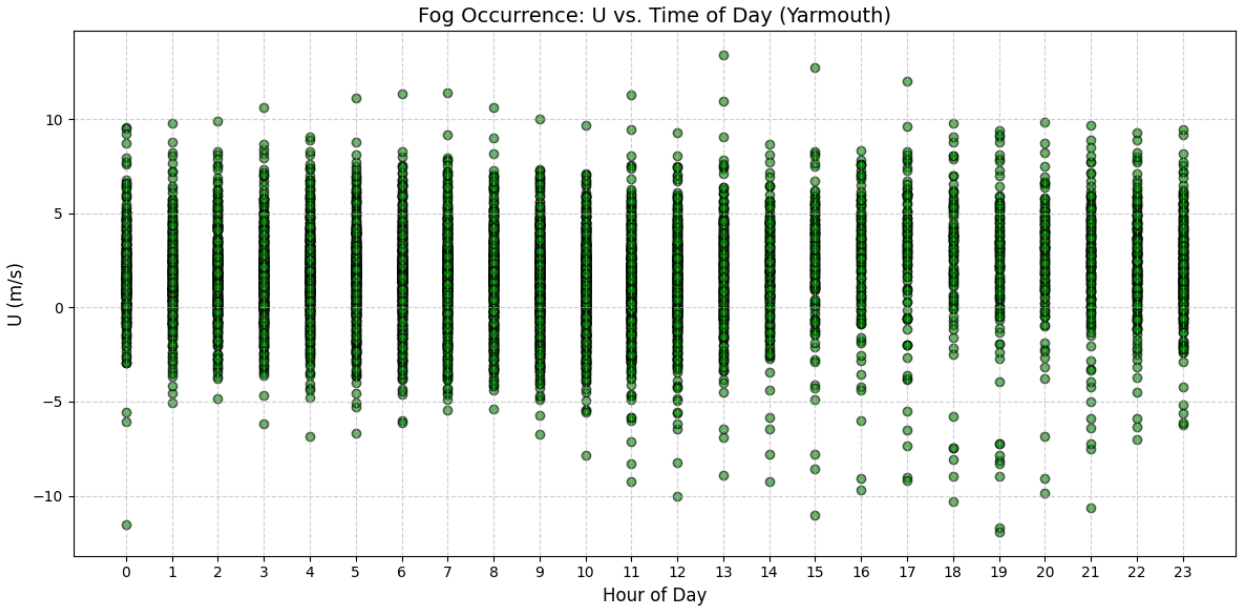
a day (a line plot) to determine the peak hours of fog during the day in Fig. 2.9 and also a scatter plot of zonal wind, U at 10 m, during fog hours in a day to see the wind distribution when fog is happening in Fig. 2.8.

According to National Oceanic and Atmospheric Administration (2025), moderate wind is from 13-18 mph. Wind speeds of 4 m/s are equivalent to approximately 9 mph, representing weak to slightly moderate winds. Winds around 8 m/s (18 mph) represents moderate winds. For St John's, when the U-component at 10 m is positive, the wind is blowing from the west, and using the weather map in Fig. 2.1 (b), the winds are from the land, and when it is negative, the winds are coming from the ocean. Furthermore, when the meridional winds are positive, the winds are coming from the south and the negative means coming from the north. This implies for sea advection, the meridional winds should be positive. Looking at Fig. 2.7 (a), we see that most of the winds are coming from the south, with around 10% of the winds coming from south-east and 14% coming from east-southeast. The wind speeds are mainly 2-6 m/s, representing weak to moderate winds coming from the ocean. From Fig. 2.6, we have some weak to slightly moderate U-compoment wind, and there are two splits. This suggests that there was both inland and oceanic winds occurring at the location, with wind from the ocean being more prominent at high relative humidity. As seen in the V-component at 10 m plot in Fig. 2.6 (b), there are moderate winds (around 5 m/s), coming from the south when relative humidity at 2 m is high, suggesting that moist air is advected inland.

Therefore, based on the weather map, during the foggy hours, there are relatively moderate winds coming from the ocean, bringing moist air as seen in the previous plot, which promotes the formation of advection fog. Looking at Fig. 2.8(a), we see the distribution of the zonal wind, U at 10 m, when fog happens at each hour of the day from 2012 to 2023. Fig. 2.9(a) shows fog occurrence per hour in a day for St John's in UTC, from 2012 to 2023,

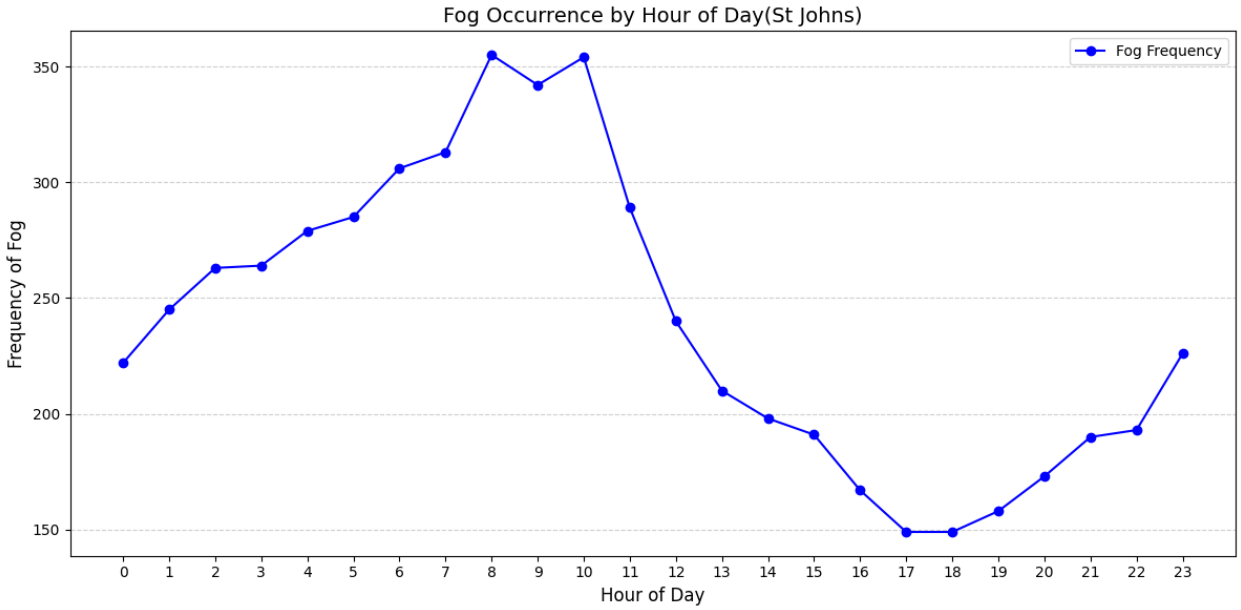


(a) Scatter plot of U at 10 m hourly during fog at St John's, Canada.

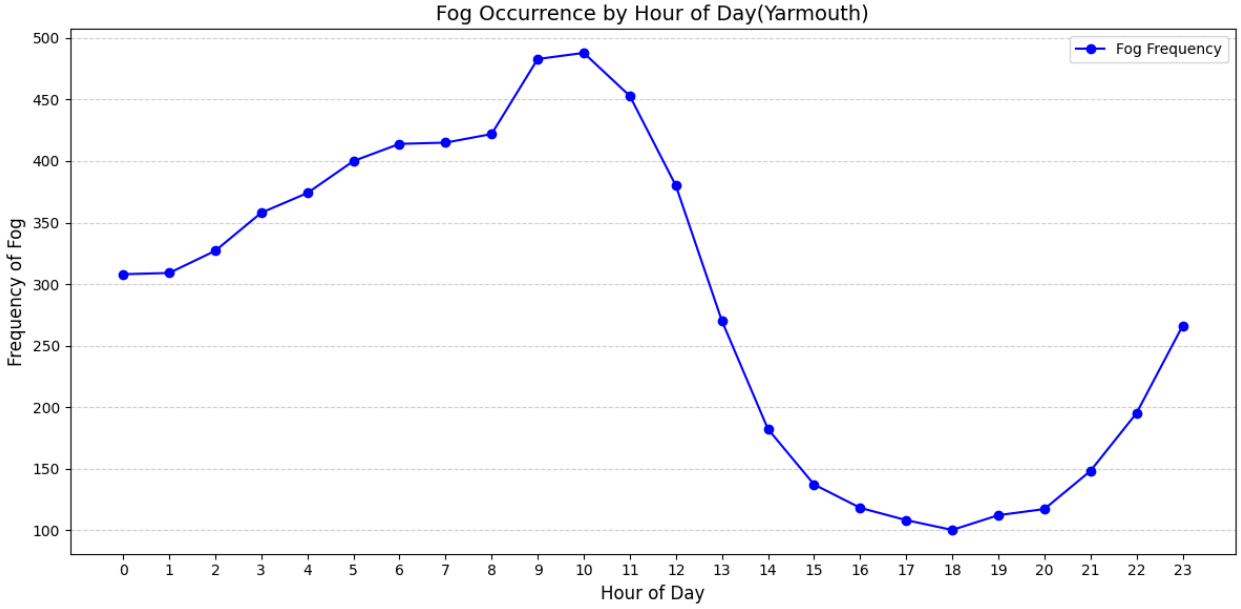


(b) Scatter plot of U at 10 m hourly during fog at Yarmouth, Canada.

Figure 2.8: Scatter plot of the zonal wind component, U at 10 m, at each hour of the day (in UTC) when fog is happening to show wind distribution during foggy hours for St John's and Yarmouth from 2012 to 2023.



(a) Line plot of hourly fog occurrences count for each day for St John's, Canada.



(b) Line plot of hourly fog occurrences count for each day for Yarmouth, Canada.

Figure 2.9: Line plots of hourly fog occurrences in a day during March to August, from 2012 to 2023 for both St John's and Yarmouth. Since the time is in UTC, this indicates higher fog during nighttime and early morning.

and shows peak fog occurrence happening between 07:00 to 10:00 UTC, which local time (NST) is 04:30 to 07:30, representing very early mornings. During that time, in the U at 10 m scatter plot, the winds were concentrated around -1 to -2 m/s with a range of 5 to -10 m/s, meaning that fog occurrence was mainly accompanied by light winds coming from the ocean. The long range of the U-wind could suggest that even in less stable conditions, fog persisted. Therefore, for St John's, advection fog could be the main reason during the peak hours of fog occurrence due to moist air being advected from the ocean, as seen by the joint PDFs of wind speed and relative humidity and the wind rose plots. However, radiation fog could also play a role during calm and stable conditions in the early morning hours, particularly when the winds are very light.

For Yarmouth, the analysis of wind components and joint PDFs also provided insights into the dominant fog type. From the zonal wind (U-component) at 10 m and using the weather map as a reference, when U is positive, the wind is blowing from the west (ocean), and when negative, it is coming from the east (land). The meridional wind (V-component) at 10 m shows southerly winds bring air from the ocean and for northerly winds, it is inland. We noticed that the density of the joint PDFs for Yarmouth is higher than St John's, indicating a higher frequency at the clusters. The joint PDF of RH2 and U in Fig. 2.6 (c) reveals that high relative humidity often corresponds to very light westerly winds (positive U), suggesting very light breezes coming from the ocean. Similarly, the joint PDF of RH2 and V in Fig. 2.6 (d) indicates that high RH2 is observed with moderate southerly winds (around 5 m/s), suggesting there is advection of moist air from the ocean from both wind components. Looking at the wind rose plots during fog conditions, we have a better representation of the winds. A majority of the wind data comes from the northwest and west-northwest directions (around 43%) with wind speeds ranging from 4-8 m/s. This indicates moderate winds coming from the ocean during the foggy hours. This aligns with the advection of moist oceanic air. The distribution of wind speeds in this range is typical for advection fog formation as moist

air moves over cooler surfaces, leading to condensation.

Fig. 2.8 (b) highlights the distribution of the U-component during foggy hours shows less scattering than St John's and is concentrated around 3 to 5 m/s, indicating weak moderate westerly winds. The occurrence of fog per hour in Fig. 2.9 (b) shows a peak between 08:00 to 11:00 UTC (local time is 04:00 to 07:00 ADT), corresponding to the early morning hours. The fog occurrence per hour in a day follows the same trend as in St John's, with peak times in the early mornings and nights and very little fog during the day, suggesting the sun to be a main factor for stopping fog formation. The peak fog hours were much higher than at St John's as well, with around 500 counts compared to 350 counts. During this time, the U-component suggests moderate winds from the ocean, which, combined with high relative humidity, supports fog formation. Based on the plots and analysis, advection fog appears to dominate in Yarmouth due to the advection of moist air from the ocean during foggy periods, as shown by the consistent westerly and southerly winds and high RH2.

2.4.4 Imbalanced dataset

In addition, one important aspect of classification is figuring out how the categorical labels are distributed. Google For Developers (2024c) describes two types of datasets, balanced and imbalanced. For binary classification, a balanced dataset will have both labels nearly the same amount, while an imbalanced dataset will have one label that appears much more than the other. Those are called the majority label, and the one that shows less often is called the minority class. From Fig. 2.10, we can see that there are many more clear hours than fog hours, and therefore the fog label is considered to be a minority label. To get a good sense of the degree of imbalance in our dataset, we refer to Table 2.3. For both locations, the percentage of fog data is around 10-12%. Therefore, we have a moderate case for imbalanced data. Fig. 2.10, thus, enables us to understand our visibility data better and allows us to

take measures when preparing our data to run our model properly.

Percentage of data belonging to minority class	Degree of imbalance
20-40% of the dataset	Mild
1-20% of the dataset	Moderate
<1% of the dataset	Extreme

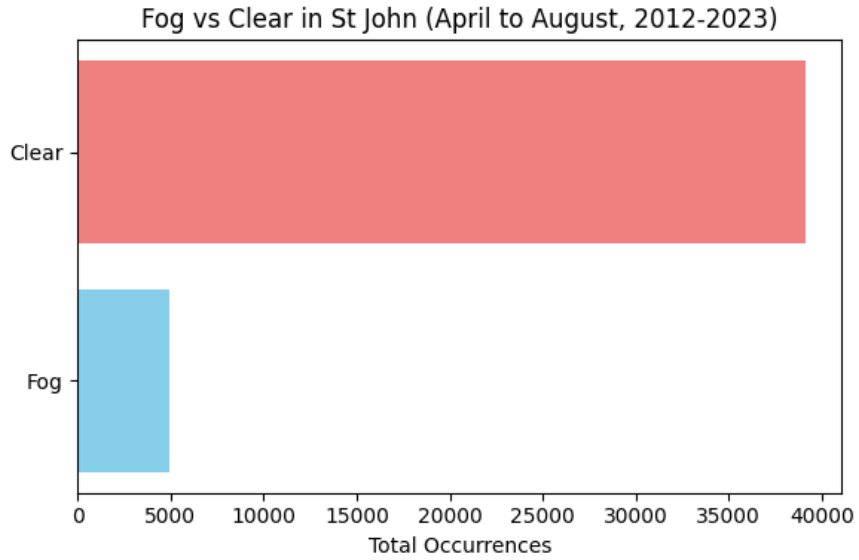
Table 2.3: Degree of imbalance based on minority class percentage (Google For Developers (2024c))

2.5 Data Preparation

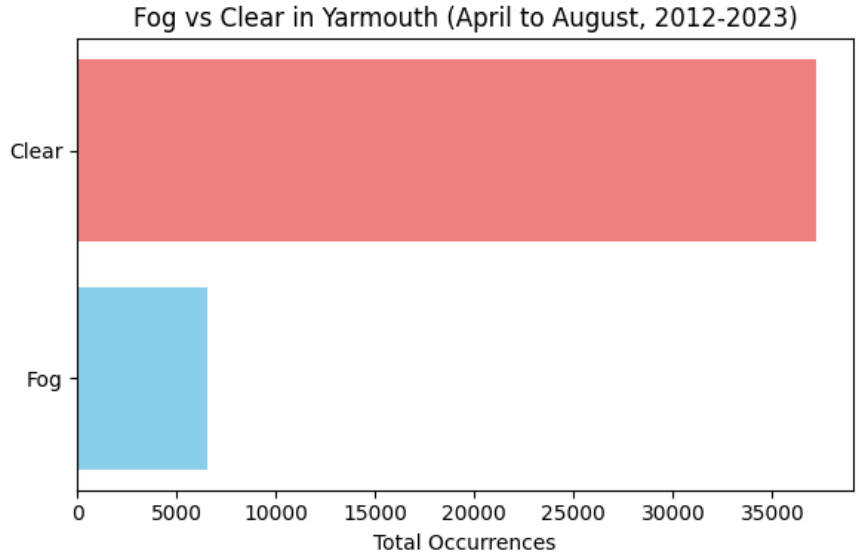
After cleaning and exploring the data we collected, we need to prepare it so that it is properly formatted and can be used in the ML models we used. Malik et al. (2015) describes data transformation as taking the cleansed data and transforming them into a structure or changing the values to meet the requirements of the ML model. On top of transforming the data for the models, data preparation helps minimize errors in the data. It makes it easier to monitor the data so that any future problems or errors in the data can be identified and corrected. As a result, this process helps improve the quality of the data, ensuring that the machine learning model receives accurate data to produce better results (Njeri (2022)).

2.5.1 Time-Based features

The first step was to prepare the time feature of our data. Since we are dealing with a time series situation, we deemed it effective to format time into a useful feature. After saving our data in a .csv file, we uploaded it to a Panda data frame for easier use in Python using the Pandas library (The pandas development team (2020)). We parse our time column into a DateTime format, YYYY-MM-DD_HH. YYYY is the year, MM is the month, DD is the date, HH is the hour. Then, we extracted each specific component, creating time-based



(a) Hourly fog occurrence count from 2012 to 2023 compared to clear hours for St John's, Canada.



(b) Hourly fog occurrence count from 2012 to 2023 compared to clear hours for Yarmouth, Canada.

Figure 2.10: Comparison of hourly fog count to clear count for both locations, from 2012 to 2023, showing an evident case of imbalanced data of a moderate degree. This allows us to take measures for imbalanced data when we are preparing it for model use.

features. By extracting these features, we were able to analyze the data based on different time components, such as looking at trends over specific hours of the day or different days. These features are useful in training ML models that need to capture time-based patterns, especially weather patterns that can last for hours, like fog. We have learned in the EDA section that monthly fog occurrence and the hour at which fog happens are valuable to learn the behavior of fog. Therefore, by having those time-based features, the ML model can identify any useful traits to make better prediction.

Another important aspect when we have time-based features is that the other features are also time series features. Since we collected our data hourly, the features we used are serially dependent and will have different types of trends (Holbrook and Cook (2023)). The first trend we established at the start was seasonality and we chose only summer data, and there are hourly trends (diurnal cycle) and monthly trends. One way to try and capture this dependency is to use lagged features. Lagged features are essential for time series forecasting because they can capture historical dependencies, trends, and seasonal patterns that influence future values. Lagging a time series feature means we are shifting the current value forward in time by a desired amount of steps. For our case, this means having the previous hour or hours with our features and this helps capture either persistence or autocorrelation. From American Meteorogical Society (2024), persistence in weather is that the condition happening right now will remain the same through the next time steps. For our case, we tried to use various lagged timesteps and from the results, lagging by one hour was the most effective so we used lagged features by one hour as well as the features to train the ML models. For example, we will have relative humidity (RH_t) and relative humidity 1 hour ago (RH_{t-1}) which can help capture any changes in the variable and effectively make better predictions.

2.5.2 Format the Features and Target

Since we saved all our data into a .csv file, the features and the target are in the same dataset. After having classified the visibility data as fog or clear (mist when multi-class), we introduced a new column of data named 'class_vis' for either binary or multi-class. We then separated the features from the target (class_vis) and drop the unwanted columns such as visibility and time. After parsing the Time column, we dropped it because we were going to use the decomposed features that were more meaningful. In the next part of data preprocessing, we use the Scikit-Learn package, which is an open-source library that has simple and efficient tools for predictive data analysis and is also ideal for traditional ML with structured data (Pedregosa et al. (2011)). Now, based on the EDA, we can approach feature scaling with greater confidence.

Feature scaling is a method used to transform the values of independent variables so that they have common characteristics, such as being within a certain range of values, having an average value of 0, and having the same standard deviation. This is done in ML techniques because if the features are on different scales, the weight they carry will differ, irrespective of their importance to the outcome, which leads to poor results (Scikit-Learn (2024e)). Since some of our data follow a normal distribution closely (except V and RH2), we decided to standardize using StandardScaler (Scikit-Learn (2024h)) which uses the Z-score formula given below:

$$\text{Z - score} = \frac{x - \mu}{\sigma} \tag{2.7}$$

,where x is the data points, μ is the mean and σ is the standard deviation. We chose this scaler because the U-component and V-component of winds at 10 m and surface pressure closely follow a normal distribution and have a few outliers, which we do not want to lose

because they might contain valuable weather information. So, StandardScaler is the most appropriate to use as it is suitable for Gaussian distributions and is sensitive to outliers (Scikit-Learn (2024a)). In Fig. 2.11, we can see the normalized features and how they are scaled the same, with mean = 0 and unit standard deviation, and they are now ready to implement in the ML models.

Now that we have prepared our features, the target needs to be carefully processed so that the ML models can use it. When doing binary classification, we cannot use string fog and clear for the ML model. Instead, we mapped them to binary classes 0 and 1. By mapping clear as 0 and fog as 1, we represent the target variable as a numeric value to train the ML model. When doing the multi-class, we have three classes: fog, mist, and clear. We use the LabelEncoder function (Scikit-Learn (2024f)) which is a helpful tool that converts labels into a numerical format, assigning them values between 0 and the number of unique classes minus one (0 to n_classes-1). In our case, it is used to turn the non-numeric labels (fog, mist, and clear) into numeric ones (0, 1, 2) (Scikit-Learn (2024j)). One issue that will arise when doing multi-classification is that, by further categorizing the visibility, it may be harder for the ML model to learn about accurately predicting fog due to imbalanced data.

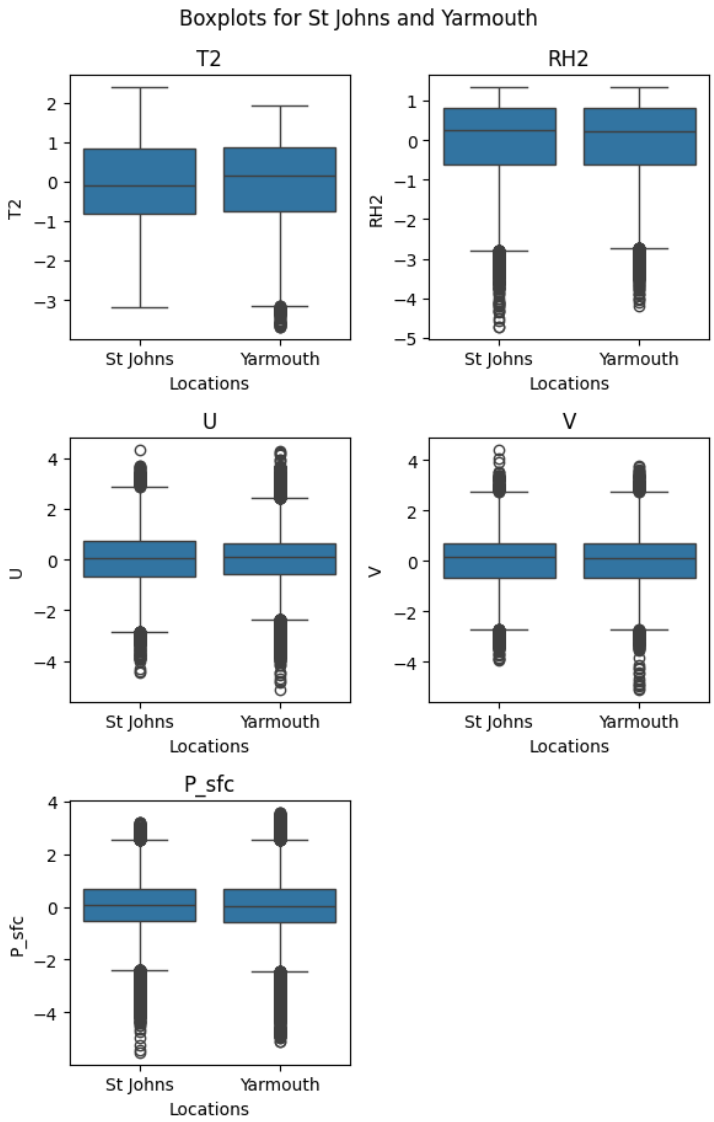


Figure 2.11: Normalized boxplots for T at 2 m, RH at 2 m, V and U at 10 m, and surface pressure for St John's and Yarmouth, Canada. This shows that all the features are now in the same order and ready to use.

2.5.3 Training-Test data

Now that we have prepared our data to train the ML models, we must split the data into training and testing carefully. Weather data is considered to be a time series type of data, meaning that there is a serial correlation between the data points. The persistence; how neighboring data are correlated, is strongly present in weather data. If there is rain in one hour, the probability that it rains in the next hour increases (Hsieh (2023)). Our data comprise a set of observations that are measured sequentially between equally spaced intervals of time (hourly). This introduces the concept of time lags (Ludlow and Perez (2018)) which was explored more in the previous section. So, when we divide our data into training and testing, we have to preserve this sequence of data so that the autocorrelation is captured and can be used to infer whether there is fog. There are many ways to split your data, and one of the more popular ways is to randomize your dataset and then perform an 80-20 split with training and testing. This is an issue with time series weather data, as they are autocorrelated. By mixing your dataset, you perform data leakage, meaning that your model is not using the features to predict, but rather using the closest data points which in our case would be closest temporally. This will lead to higher model performance, but in reality, it will not do well with new data (Rao (2021)). We ensured that the data was split sequentially, placing the test data at the end of the time frame to preserve the time order. We decided to split the data into 3 sets. We did an 80-20 split between the training set and the testing set. We perform this splitting because the model needs a decent amount of data to train, while the remaining 20% provides enough unseen data to evaluate how well the model generalizes to new, unseen examples. This is a typical split; however, a better split now includes a further 20% of the training set to the validation set. The validation set is used as an initial testing of the trained model and provides a means to optimize the model by tuning the hyperparameters while avoiding overfitting (Google For Developers (2024b)).

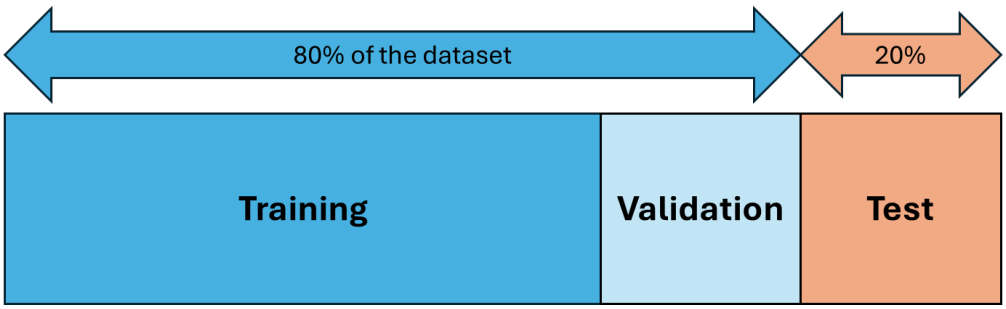


Figure 2.12: Optimal split of the dataset into training, validation, and test set to train the model.

In Fig. 2.12, the splitting of the whole dataset is illustrated and 64% of the whole dataset is used to train the model, 16% is used as validation and finally, 20% is used for testing. This ensures that we have a large amount of data to build an accurate model and enough data to evaluate the model. By including a validation set, we allow the model to generalize better. This can be done by tuning the hyperparameters, such as learning rate, and batch size, based on the result of the validation set. Also, it is an early step to prevent overfitting as we are training and evaluating with the validation set. If the model is doing well with the training data but poorly on the validation, it indicates that the model might be overfitting to the training data. The validation set helps detect this issue early. The workflow of optimizing and tuning the model with the validation set is given in Fig. 2.13.

2.5.4 Imbalanced Dataset

As we noticed in Fig. 2.10, we have an imbalanced dataset where the majority of our data is clear compared to fog. This is an issue as our model will train mainly on clear occurrence and as a result, will give poor predictions for fog. This happens because minimizing the overall error makes it easier for the model to predict the majority class correctly. This will lead to a poor generalization of the model as it may not have learned enough from the minority class, leading to poor performance when it encounters the minority class during testing. In

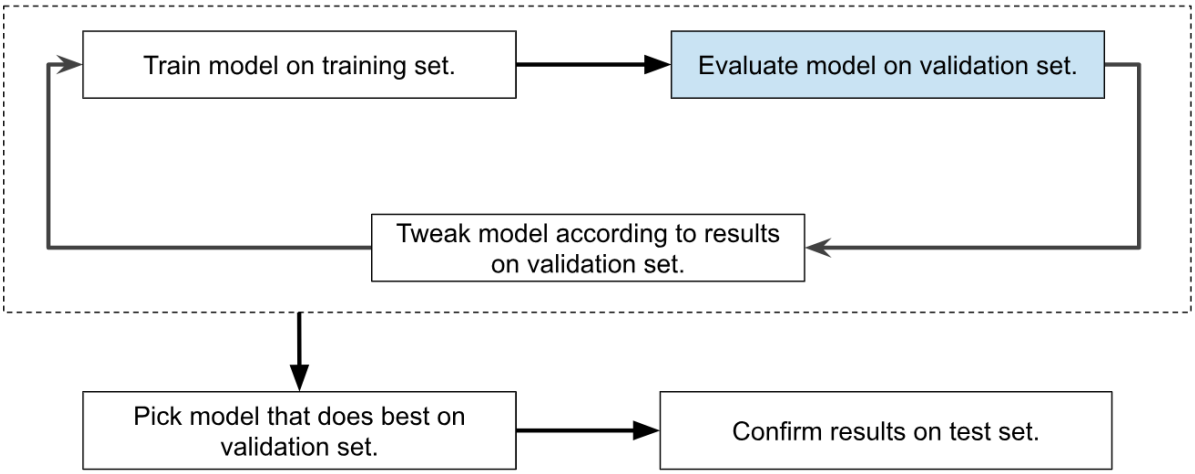


Figure 2.13: Workflow of splitting the data into training, validation, and test sets, using the training set to train the model, the validation set to fine-tune and optimize it, and the test set for final evaluation to assess its performance on unseen data (Google For Developers (2024b)).

addition, performance metrics, such as accuracy, will be skewed. If the model predicts the majority class very well, the accuracy will be very high, but it does not mean that the model can accurately predict the minority class. For the fog classification case, this means that we are misrepresenting the important class as it is more crucial to predict fog than clear (Jafarigol and Trafalis (2023)). There are many ways to deal with an imbalanced dataset, and we chose a combination of two methods during data preparation and a few techniques when optimizing our models. Both techniques we chose were resampling methods. The first method is downsampling your majority class, that is, training the model on a lower subset of the majority class. This balances the data more and allows the model to train on the fog data (Google For Developers (2024d)). The second method is oversampling the minority class. Mohammed et al. (2020) defined oversampling as a method to increase the number of minority labels in your dataset by either producing new minority labels or duplicating the labels.

We use the imbalanced-learn package, which is an open-source library that relies on Scikit-

Learn to provide tools for classification and imbalanced data (Lemaître et al. (2017)). The downsampling method used is the Edited Nearest Neighbor (ENN) (Wilson (1972)) and the oversampling method used is SMOTE (Synthetic Minority Over-sampling Technique (Chawla et al. (2002))). Both concepts are based on the K-Nearest Neighbor (k-NN) principle. k-NN is a non-parametric supervised learning method designed to classify objects by inferring the neighboring points. From Hsieh (2023), a parametric model has a finite number of parameters, while a non-parametric model's parameters grow with the amount of training data and this makes it better at forming decision boundaries for complicated cases. The neighbors are determined by measuring the distance between them, usually,non-parametric the Euclidean distance given in equation 2.8. The Euclidean distance d(p,q) is calculated between two points, p and q, on a real line in equation 2.8 (Wikipedia contributors (2024a)). An example of how k-NNN is given in Fig. 2.14.

$$d(p, q) = \sqrt{(p - q)^2} \tag{2.8}$$

The ENN method is an undersampling method that cleans the dataset by using the k-NN methodology to identify the surrounding classes around the target, then removes the target if any or most of the neighbors are of a different class (Wilson (1972)). First, ENN will train k-NN through the entire dataset and determine k, the number of nearest neighbors. Then, during the editing process, for every data point, the ENN method checks whether most of its k-nearest neighbors belong to the same class as the data point itself. If the class label of the data point disagrees with the majority class label of its neighbors, the data point is considered misclassified, and therefore the algorithm will remove this misclassified data point from the training set. The steps are then repeated until the desired proportions are obtained (Scikit-Learn (2024k)). This process can also be considered as cleaning the data and it will help the model, and since we are filtering potential misclassified data points,

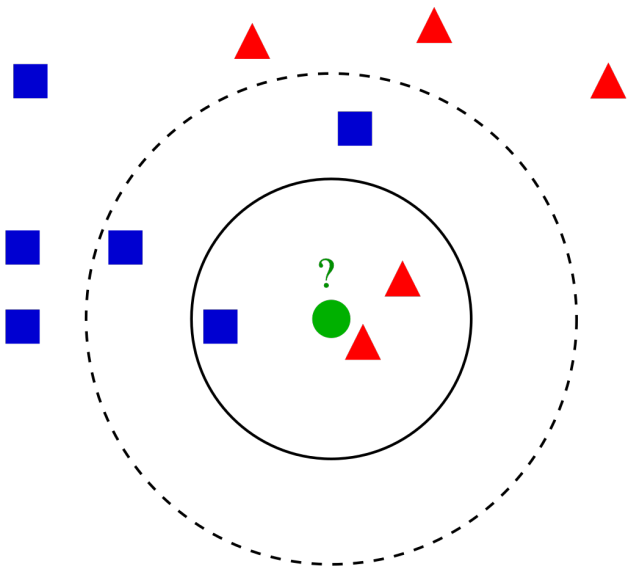


Figure 2.14: Example of k-NN classification. The green dot is the test which will either be assigned to the blue square or the red triangle. If k = 3, the circle will be the boundary and the green dot will be classified as red as there are two red and one blue. If k=5, the boundary is the dashed circle and now there are three blue and two red and the green will be assigned to blue. Image by Ajanki (2010).

the decision boundaries are improved, which can lead to better decision-making. Based on Aurelius and Viadinugroho (2021), in comparison to other undersampling methods such as TOMEK Links (Tomek (1976)), it provides a more in-depth data cleaning where it is more careful at removing the labels.

The second method to deal with imbalanced data is the Synthetic Minority Over-sampling Technique (SMOTE), which is an oversampling method. This method will look at the minority class label. Chawla et al. (2002) proposed this method as a way to generate synthetic data points for the minority class rather than simply duplicate the original ones. It does so by interpolating between existing data points and creating a new one between them. This method also uses the k-NN methodology to determine the minority class and create synthetic labels. When the fog label is identified as a minority class, SMOTE will randomly take a sample of it and find its k-nearest neighbors. The default number of neighbors, k, is five.

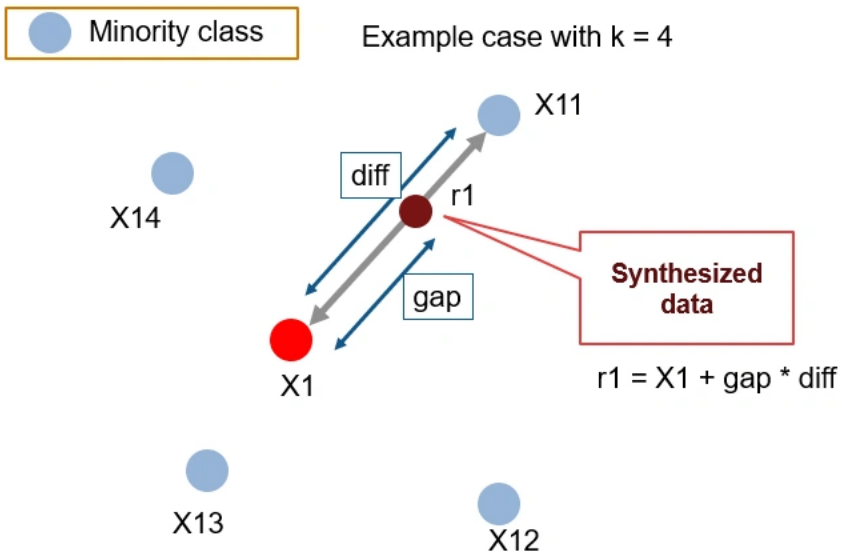


Figure 2.15: Oversampling method, SMOTE - Synthetic Minority Over-sampling Technique, that creates new data for the minority class using the k-NNN methodology. Image by Swastik (2024).

So, depending on the amount of oversampling needed, the amount of neighbors used will be adjusted. For example, if an oversampling of 200% is needed, then only 2 nearest neighbors will be used. SMOTE will generate synthetic data points by choosing one of the k-nearest neighbors and creating a new synthetic point along the line segment between the original data point and the neighbor. This will affect the features as well such that each feature of the new sample is synthesized by interpolating between the feature of the original data sample and its neighbor. This interpolation is done by taking a random value between 0 and 1 and multiplying it by the difference between the original sample and the neighbor and this is given by equation 2.9 below. The process is then repeated until there is a balance between the majority (clear) and minority (fog) labels.

$$x_{syn} = x_i + (x_{nn} - x_i) \times random(0, 1) \tag{2.9}$$

, where x_{syn} is the new synthetic data, x_i is the original minority sample and x_{nn} is a

randomly chosen neighbor. Fig 2.15 illustrates how new synthetic data is created from the original sample and the neighboring data with the neighboring number k = 4.

Now that we understand how both processes work, we chose to use a combination of both, SMOTEENN, which means SMOTE plus ENN. Batista et al. (2004) introduced this method which involves both oversampling the minority label and undersampling both classes. ENN, while remains a method to counter imbalanced data, is often not enough to train the model to accurately predict fog labels. SMOTE also is sometimes not enough as this method does not have knowledge of the underlying distribution and hence can introduce noisy data at the boundaries (The Imbalanced-Learn Developers (2024)). SMOTEENN tackles this issue by first generating synthetic data for the minority samples till the desired proportion is obtained, and then uses ENN to clean the data of noisy samples so that the decision boundaries remain clear. This combination of methods has been proven to work well in cases of moderate to extreme imbalance, which is our case.

After applying SMOTEENN, we have to look at how much new data were generated from the original. We showed results for St John's, Canada only as they are going to be similar for Yarmouth, Canada. A quick comparison between Fig. 2.10 and Fig. 2.16 shows that the issue of having imbalanced data has been taken care of using SMOTEENN. At first for St John's, there was a total of 46, 729 clear labels compared to 5761 fog labels. This means that fog comprised around 11% of the entire dataset and after SMOTENN, there were 34, 277 clear labels and 36, 016 fog labels. This results in a fog distribution of around 51%, making a more even binary classification between clear and fog.

Having seen that the SMOTEENN successfully increased the number of fog labels such that the model can learn both clear and fog equally, we need to know how were they produced. It is important to see if the new synthetic data still made sense in an atmospheric sense so we plotted a 2D-scatter plot with temperature at 2 m on the y-axis and relative humidity

Figure 2.16: Bar chart of the class distribution between clear and fog label after applying SMOTEENN (0 = clear and 1 = fog) at St John's, Canada. The imbalanced dataset is now even and suitable for the model to learn for both cases.

at 2 m on the x-axis with the two labels, clear and fog. In Fig. 2.17, we have the scatter plot for St John's before applying the SMOTEENN and we can see that fog, as expected, mostly occurs when relative humidity is close to 100% and some cases around 90%. The two features were chosen randomly just to show how the SMOTEENN method works. After applying SMOTEENN, we now have a balanced amount of clear and fog labels as shown in Fig. 2.16 and we observe that the synthetic data points do make sense as most of them were generated above 90% relative humidity. This ensures that the correlation between the features and the target remains almost the same, and the model will learn the relationships correctly.

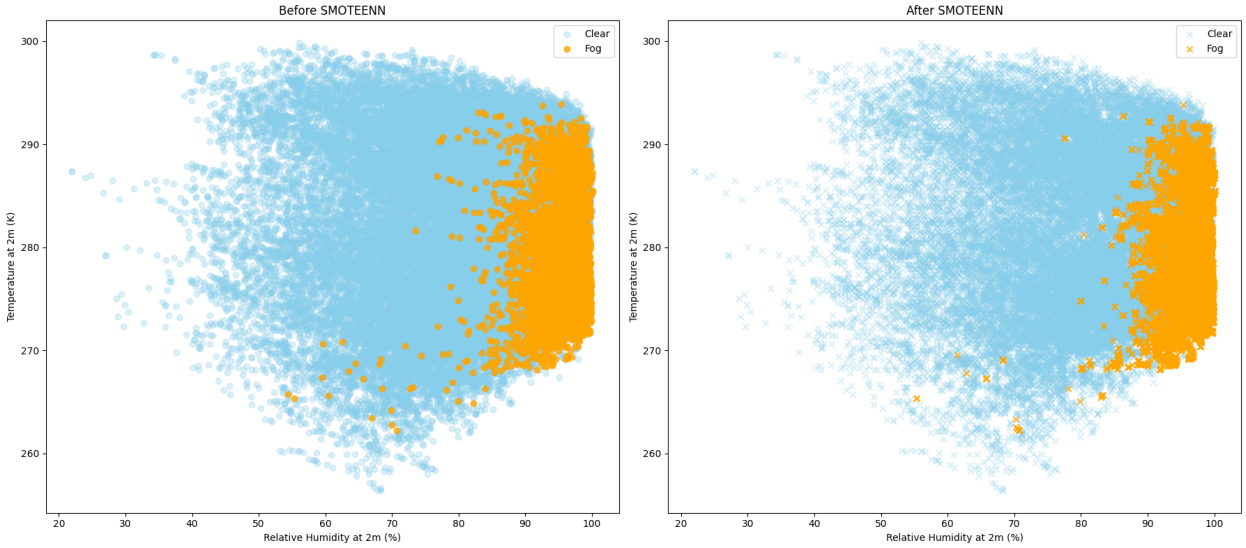


Figure 2.17: 2D-scatter plot with temperature at 2 m and relative humidity at 2 m with clear as the blue label and fog as the red label for St John's, Canada. The figure on the left is before SMOTEENN and the figure on the right is after.

Chapter 3

Building the ML Models

Having carefully cleaned and prepared the data, we proceeded to use them on ML models. For this research, we decided to use two very popular ML techniques that are suitable for weather forecasts and prediction. We chose to use Random Forest as it is one of the five most popular ML models and also Long Short-Term Memory (LSTM), which is also in the top five of deep learning (DL) models based on Kumar (2020). Kumar (2023) ran an experiment to predict the weather using various ML models and found that ExtraTreesClassifier performed the best which is a type of extremely randomized Random Forest so we decided to test the performance of this technique as well.

3.1 Decision Trees

Decision trees are a type of supervised machine-learning model used for classification and regression tasks. They create a tree-like structure that resembles a flowchart, breaking down a decision-making process into a series of binary choices based on feature values. They are a simple way of classifying examples into labels or categories. Paraphrasing Hsieh (2023)

description, we utilize the tree concept described below.

The tree will start at the root node, which represents the very first split of the whole dataset. The splitting depends on the importance of the feature and the threshold that will best divide the data into other classes. After the root node, the data will continue to divide into more classes, each representing a subset of the whole dataset. Finally, the last decision of the tree is the leaf node, and the internal nodes found in between are called splits. Each internal node in the tree represents a question about one of the features in our dataset, and based on the answer, the data moves down a specific path to the next question or to the leaf. Then, each leaf in the tree will show the predicted class or the probability of each class, indicating the category the data likely belongs. The most common decision tree method is the classification and regression tree (CART) introduced by Friedman (1984). Classification tree analysis is done when the desired output is classes to which the data belong and regression tree is done when the predicted outcome is continuous.

Our main focus is to classify fog, so we used classification decision trees. The main differences are how to predict the outcome and how to tune the decision tree model. In a node, the decision will have to decide where to split or not. The decision node contains a sample size, n, and the decision tree will calculate the purity of that node and decide to split to get a more homogeneous sample. The impurity measures how mixed a node is and the goal of each split is to reduce the impurity, making the nodes purer. There are two methods to determine impurity and thus the splitting. There are three available criteria to calculate the impurity, and the appropriate criterion is determined by tuning the model (Scikit-Learn (2024b)). For our situation, after optimizing the model, we found that for St John's, the Gini criterion worked better, and for Yarmouth, it was the Shannon entropy. First, the proportion of observations of class k in a decision node is given by:

$$p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k) \tag{3.1}$$

, where Q_m is the data at node m, n_m is the sample size and m is the node. This determines what fraction of the node belongs to class k. I(y=k) is the indication function, which returns 1 if the condition inside it is true and 0 otherwise. The criterion Gini then uses this fraction to measure the impurity of the node:

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk}) \tag{3.2}$$

This calculates the probability of incorrect classification at a node. For binary classification, when p = 0 or 1, one class dominates and Gini will show a value of 0 (Hsieh (2023)). The other method uses the same principle but measures the disorder at a node, aiming to reduce the entropy with each split. The Shannon entropy calculates the probability by considering the class frequencies of all the training data points that reached a certain leaf, m. The equation is given below:

$$\text{LL}(D, T) = \sum_{m \in T} \frac{n_m}{n} H(Q_m) \tag{3.3}$$

, where D is the training dataset, T is the tree model, data at node m is denoted by Q_m, and $\frac{n_m}{n}$ is the fraction of training data points that reached that leaf. LL(D,T) is the sum of Shannon entropies computed at a leaf. Those two methods measure the impurity in each node, and then the decision tree uses the information gained to decide how to split to the next nodes using the following equation:

$$\text{Information Gain} = (\text{Impurity of Parent Node}) - (\text{Weighted Impurity of Child Nodes}) \tag{3.4}$$

Information gain aims to minimize the impurity at each split, as it evaluates it before and after a split. Bhuvaneswari (2020) describes information gain as how much information the training points give us about the class and the parent node with the highest gain is chosen to construct the child nodes. For classification, the highest gain is a measure of how homogeneous a node is for the classes. After the initial split, using the same technique, the tree then takes each subgroup and repeats the splitting process within each one. Each split creates new subgroups, and the process is applied to each until a stopping criterion is met. This is known as recursive partitioning (Cook and Goldman (1984)).

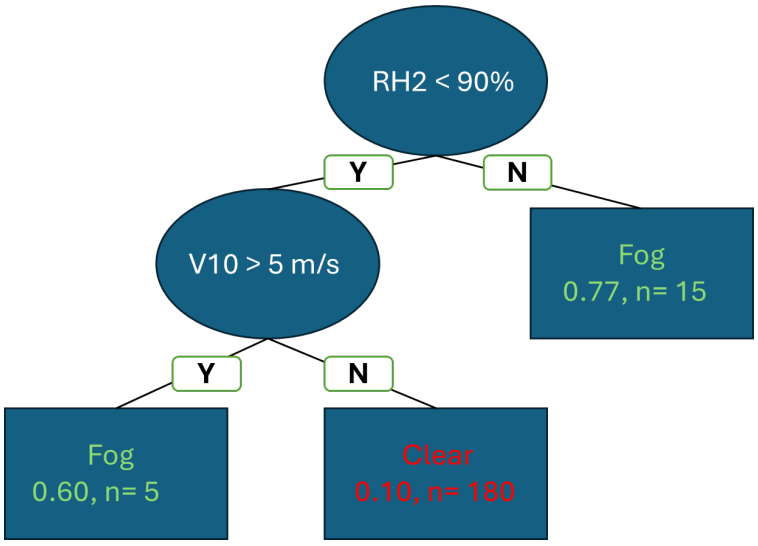


Figure 3.1: A simple decision tree that was made that showed how a decision tree will classify fog from the features we chose. The tree is plotted upside down, that is, the leaves (the boxes) are at the bottom and the internal nodes (ellipses) are above. The numbers in the leaves show the probability of fog occurrence and how many samples were in the leaf. In summary, the greatest chance of fog is when relative humidity is above 90% or there is a strong wind coming from the South.

In Fig. 3.1, an example of a simple decision is given. The decision tree is fictitious, but

it gives an idea of how the tree will classify between fog and no fog. RH2 is the relative humidity at 2 m and V10 is the meridional velocity at 10 m. We have a sample size of 200 cases. The initial split is done on RH2 as it has a higher feature importance and the splitting process of the node and leaf is determined by the loss function (e.g. Gini Impurity). When RH2 is higher than 90%, the tree gives a 77% probability that it is fog and the sample size at the node. The rest of the sample goes to the below 90% decision and therefore needs further splitting. V10 is the next feature, and when there is a strong wind coming from the South, it brings more moisture over land which increases the chances of fog occurrence. So, the leaf node with V10 greater than 5 m/s is true has a 60% chance of fog happening, and has 5 samples. The rest of the sample occurs when the conditions are drier and with weak winds and the probability of fog occurring is 10% the tree classifies the left sample size as "clear". This case reflects more an advection fog case.

3.2 Random Forest and ExtraTreesClassifier

Simple decision trees are very fast and have low computational costs, however, they have a few disadvantages. CART is prone to overfitting as it is very easy for the tree to grow too deep, and the model becomes too tailored to the training data and fails to generalize to new data. Decision trees are also sensitive to the data, so having outliers will result in different splits changing the whole outcome of the prediction (Aaboub et al. (2023)). Moreover, the model is susceptible to data with small variations or is noisy, and all those criteria fit a weather dataset as we discussed before.

So, we used a more robust model called the ExtraTreesClassifier (ETClassifier), which is short for extremely randomized trees. Since ETs use the same principles as random forests (RFs), we will first describe how random forests work. Random forest is an ensemble

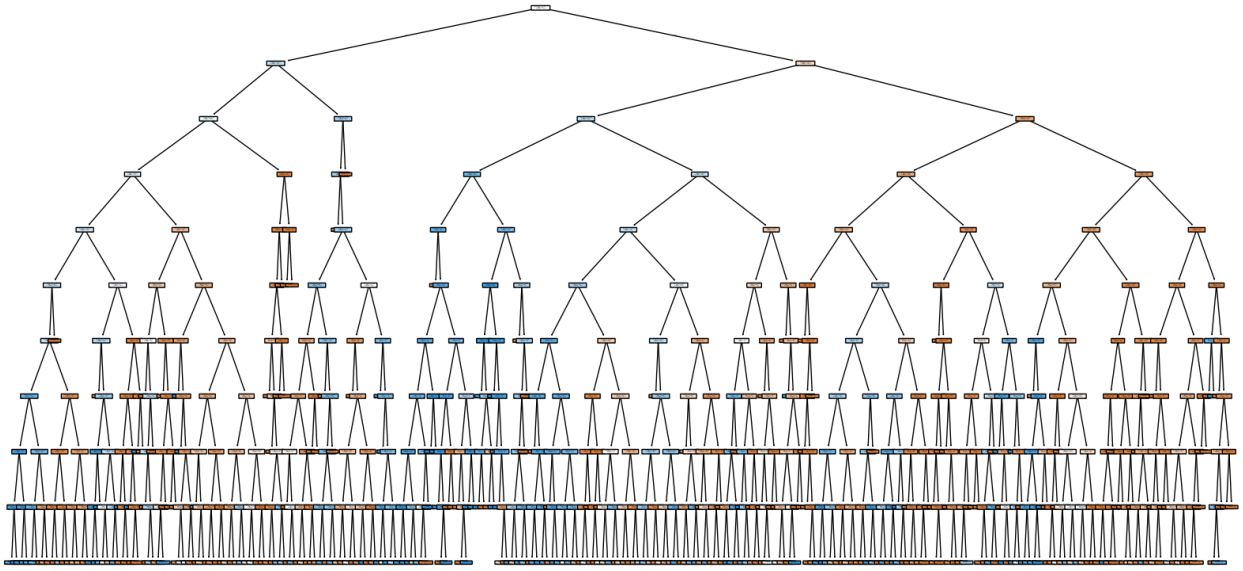


Figure 3.2: One decision tree structure from an ensemble of trees showing the complexity of the splitting from the chosen features and how the classification of fog is made in our ETClassifier model which was trained and optimized for St John's, Canada.

learning ML technique that combines the result from multiple decision trees (weak learners) to get a more accurate and robust prediction and it can be used for both regression and classification. From Breiman (2001), each tree is trained on a randomly sampled subset of the data, with replacement (bootstrap sampling), and at each split, only a random subset of features is considered. This introduces diversity among the trees and reduces the correlation between them, which helps to minimize overfitting. For classification, the trees vote for the majority class, while for regression, they average their predictions. Random Forests inherently provide an internal error estimate through out-of-bag (OOB) samples, eliminating the need for a separate validation set. This method is highly accurate, resilient to noise, and effectively handles large datasets with high-dimensional features. It also ranks feature importance, offering interpretability in complex models. By averaging predictions across diverse trees, Random Forest reduces variance, maintains generalization, and is versatile for both classification and regression tasks.

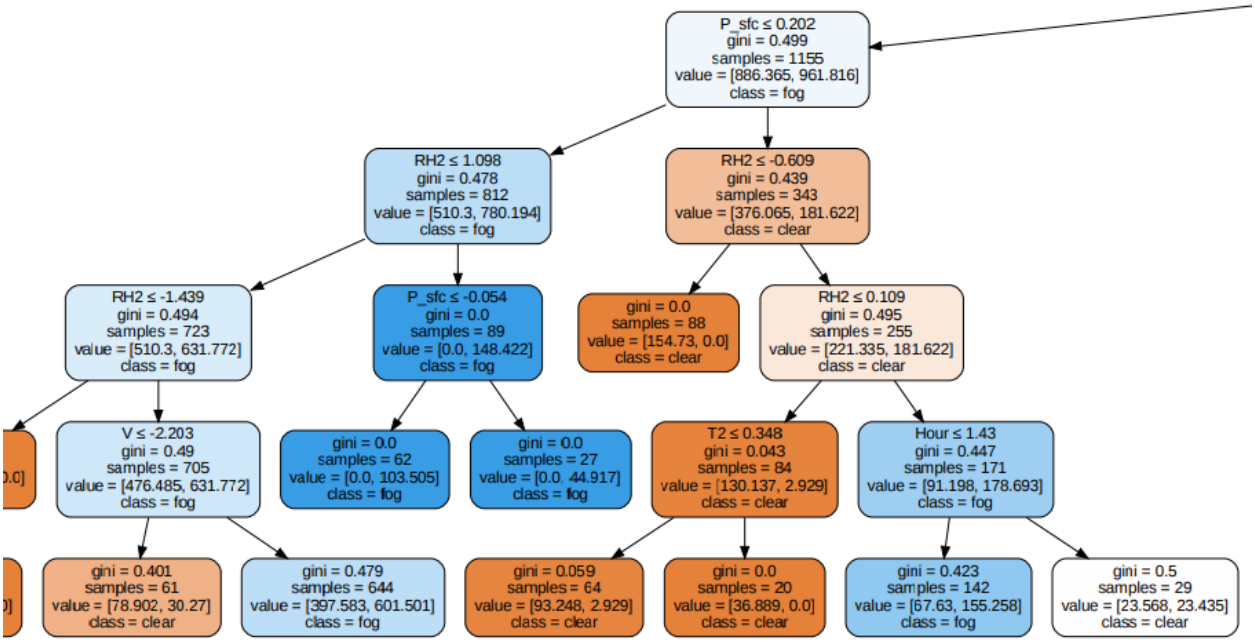


Figure 3.3: Close up of the decision tree from our ETClassifier model to see how it reaches a decision using the features and parameters.

The Extremely Randomized Trees (ExtraTrees) ML method is another ensemble method proposed by Geurts Pierre (2006) that constructs multiple decision or regression trees, using more randomness in its technique to improve robustness and computational efficiency. Like Random Forest, ExtraTrees combines predictions from multiple unpruned (not optimized) trees for classification or regression but introduces additional randomness in the splitting process. Moreover, ensemble methods such as ExtraTrees and Random Forest often have to balance their bias-variance trade-off (Wikipedia contributors (2025a)). Bias refers to the error introduced by approximating a real-world problem with a simplified model. It indicates how far the model's predictions are from the true values on average. Variance measures how much the model's predictions change when the training data changes and indicates sensitivity to small fluctuations in the data. In ExtraTrees, both the feature and the cut-point for splitting at each node are chosen at random, as opposed to Random Forest,

which selects a random subset of features but optimizes the split threshold for maximum impurity reduction. Even if the split threshold is chosen at random, the ExtraTrees will still try to get the best score. By randomizing both the features and the splitting threshold, ExtraTrees reduces the dependency of individual trees on specific data patterns, making the trees less correlated. The ensemble combines multiple such "diverse" trees, averaging their predictions, which cancels out random noise, and thus reduces variance.

Additionally, ExtraTrees uses the full training dataset to build each tree, whereas Random Forest relies on bootstrap sampling (with replacement). This design choice reduces bias in ExtraTrees by providing more data to train individual trees, while the increased randomness reduces variance more effectively than Random Forest. By skipping the optimization step for splits, ExtraTrees is computationally faster and is well-suited for high-dimensional or noisy datasets. However, this added randomness slightly increases bias which can decrease the model's accuracy. Compared to Random Forest, ExtraTrees is a better fit for weather problems where atmospheric features are usually noisier and large amounts of data are available that are rich in information. They are mostly 4D variables; for example, T2 is the temperature at 2 m at a specific location (lat-lon) at a given time. We ran both the Random Forest and ExtraTrees models for fog prediction and observed a better performance on the ExtraTrees model, aligning with its advantages. We will show results for ExtraTrees only as both models are quite similar and ExtraTrees performed better.

The ExtraTrees can be used for classification (ETClassifier) and work in the same way as described in the Decision Tree section. In Fig. 3.2, we extracted a tree from the ensemble of trees used for the binary classification of fog at St John's, and in Fig. 3.3, we see a close-up portion of that tree. From Fig. 3.3, we can see how the ETClassifier worked for the classification. At the root node (not really the root node, but in this example it is), the first split is based on surface pressure (P_{sfc} 0.202), with a Gini impurity of 0.499, indicating that

the samples are fairly mixed between fog and clear. The value = [868.365, 961.816] suggests that there are roughly 868.365 clear samples and 961.816 fog samples at this node, with the fog class being slightly more dominant. We have decimal places in our classification because we use average weights, and at that node, the label was fog. It then repeats the process with other features until some threshold is obtained and leaf nodes are reached.

To run the model, we need to optimize it using the training data. To optimize the ExtraTreesClassifier model, we will use RandomizedSearchCV (Scikit-Learn (2024g)) with a TimeSeriesSplit cross-validation strategy tailored for time-series data. This approach ensures that the hyperparameter (hyperparameter is a configurable setting that can be set to tune the machine learning model) tuning respects the temporal order of the data, preventing data leakage from future observations that influence the training process as shown in Fig. 3.4.

The hyperparameter search will explore the following distribution (param_dist): the number of trees in the forest (n_estimators) will be randomly chosen between 100 and 500, while the criterion for measuring split quality can be one of 'gini', 'log_loss', or 'entropy'. The maximum number of features considered for splits (max_features) will include options like 'auto', 'sqrt', and 'log2', and the maximum depth of each tree (max_depth) will vary between fixed values of 10, 20, 30, 40, or be unlimited (None). To ensure robust splits, the minimum number of samples required to split a node (min_samples_split) will be drawn from a random range between 2 and 15, and the minimum number of samples required to form a leaf node (min_samples_leaf) will be sampled between 1 and 5. To address potential class imbalances, the parameter for class_weight will include 'balanced' and 'balanced_subsample' options. Additionally, we will explore both the True and False values for the bootstrap parameter, which determines whether bootstrap sampling is used when building trees. This comprehensive randomized search, combined with cross-validation based on time series, will allow us to identify the best combination of hyperparameters for the ETClassifier. (Scikit-Learn

(2024d)).

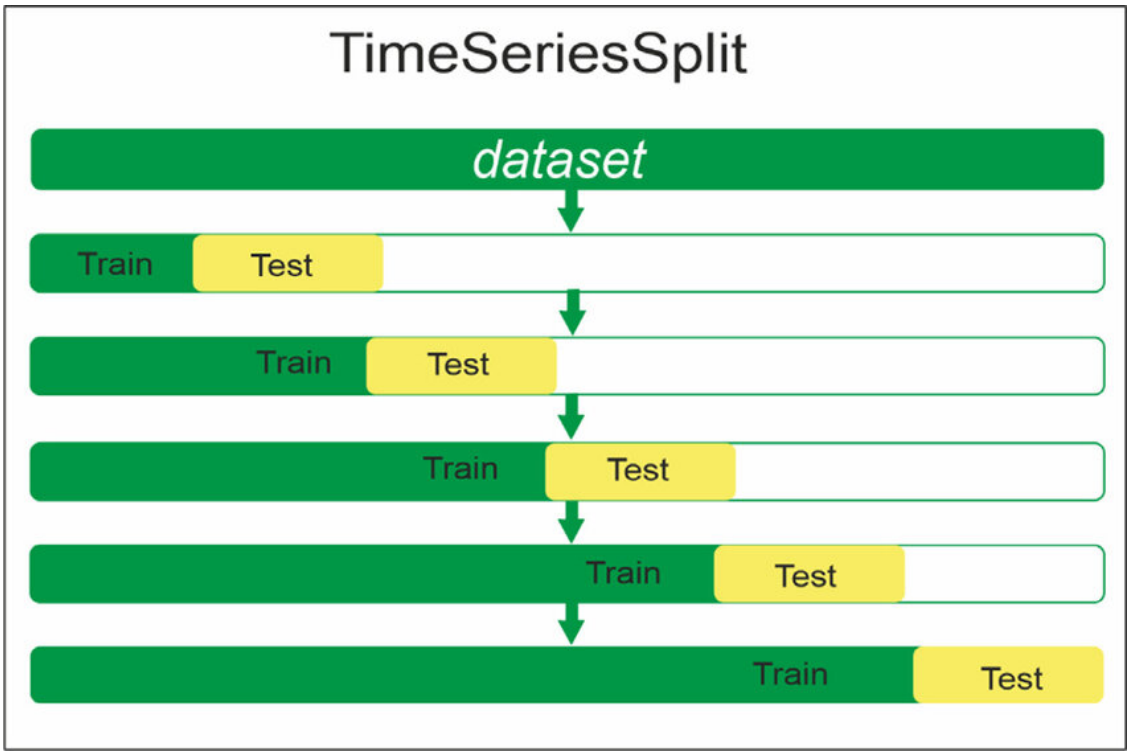


Figure 3.4: Time series split cross-validation from RandomSearchCV (Scikit-Learn (2024i)) to optimize the ETClassifier model to get the most accurate results without data leakage and overfitting. In this diagram, the dataset is the training data. Image by da Silva et al. (2022).

We ran and trained the ETClassifier model for each location individually and performed a time series cross-validation to ensure that the model is robust and generalizes well and we are able to track if the model is correctly learning by looking at the cross-validation accuracy score across the five-fold given in the table below:

We can observe a good generalization for both St John's and Yarmouth, showing that the ETClassifier can get good predictive performance and is doing well as new data is being introduced. The mean accuracy of the cross-validation for St John's is 0.92 and we can see that the accuracy for each fold is around the same and similar for Yarmouth, the mean CV accuracy is 0.88 and the accuracy of each fold is close.

Parameter	**St John's**	**Yarmouth**
bootstrap	False	False
class_weight	balanced	balanced_subsample
criterion	gini	entropy
max_depth	40	None
max_features	log2	sqrt
min_samples_leaf	2	1
min_samples_split	11	12
n_estimators	450	150

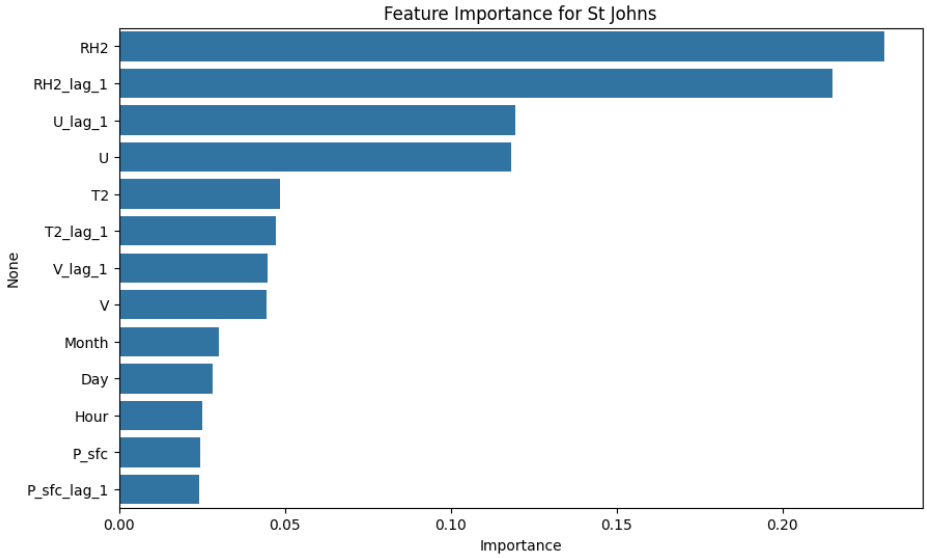
Table 3.1: Best Parameters for the ETClassifier model for both locations using RandomSearchCV.

St John's	Values
Cross-Validation Scores	[0.917, 0.912, 0.921, 0.924, 0.931]
Mean CV Accuracy	0.921

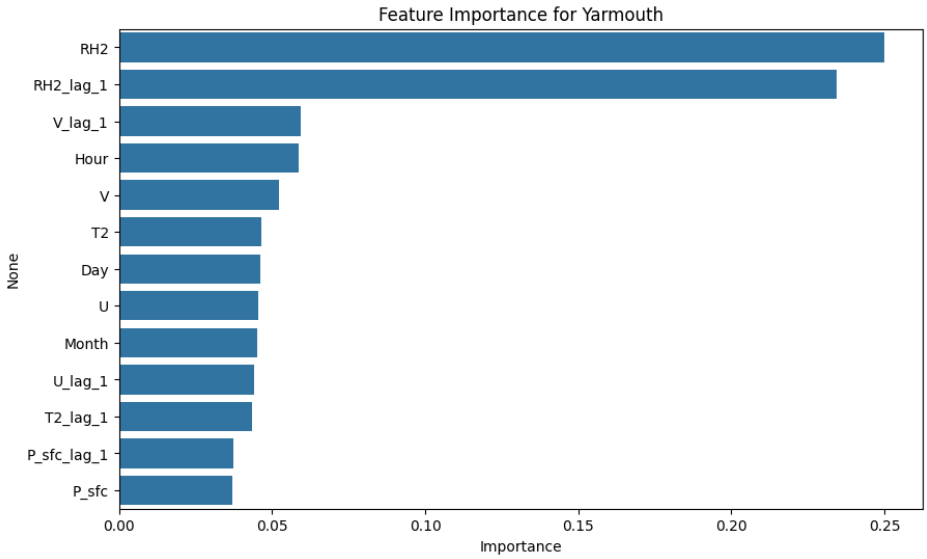
Table 3.2: Cross-Validation Scores and Mean CV Accuracy for St John's, Canada

As highlighted in the Data Exploration section, there are differences in weather conditions between St John's and Yarmouth, so we opted to build and train two different models, one for each location. Running the RandomSearchCV took around 18 minutes on 15 GB CPU for both cases and we iterated the five-fold time series cross-validation 50 times. RandomSearchCV works differently from GridSearchCV. GridSearch will go through every parameter listed in Table 3.1 and optimize it by cross-validation to find the best. While you will get the best parameters, it can be very costly to run so we ran the RandomSearchCV that will fit a fixed number of parameters from the specified distribution. In our case, we chose n_iter to be 50 and since it is a 5-fold cross-validation, it will fit through 250 different combinations. RandomSearchCV will find the best hyperparameters through a scoring method, and for binary classification, the f1 score with average weighted in case of imbalanced data is most suitable for better results.

As expected, the optimal hyperparameters for St John's and Yarmouth are quite different. The hyperparameters for both locations are listed in Table 3.1. The number of estimators



(a) Feature Importance for ETClassifier for St John's, Canada.



(b) Feature Importance for ETClassifier for Yarmouth, Canada.

Figure 3.5: Feature Importance of the ETClassifier model for both locations after optimizing, showing the differences in weather patterns and their importance to fog prediction.

Yarmouth	Values
Cross-Validation Scores	[0.895 0.844 0.887 0.906 0.866]
Mean CV Accuracy	0.880

Table 3.3: Cross-Validation Scores and Mean CV Accuracy for Yarmouth, Canada

(ensemble of trees built) is 450 for St John's and 150 for Yarmouth, suggesting a more complex relationship between the training data for St John's. The criterion used to split the nodes is gini for St John's and entropy for Yarmouth. We do not need the bootstrap method for both, meaning that all the training data was used for each tree. The class weight was balanced and balanced_subsample which work similarly by using the value of the target y that adjusts the weight inversely proportional to the frequency of the classes, given below:

$$class\_weight = \frac{n\_samples}{(n\_classes * np.bincount(y))} \tag{3.5}$$

, where np.bincount calculates the frequency of classes in y. The maximum depth for the trees is 40 for St John's and None for Yarmouth, meaning that for St John's they will expand to 40 nodes, and for Yarmouth, they will continue to grow until the threshold of the minimum sample split is met for a leaf node. This condition is 11 samples for St John's and 12 in Yarmouth. The minimum samples to determine a leaf is 2 for St John's and 1 for Yarmouth and finally, the maximum features selected for splitting was log2 of the number of features we had for St John's and sqrt for Yarmouth.

The features do not contribute equally to the splitting decision in the trees and usually, it can be easily visualized for an individual tree. However, since we have an ensemble of trees, we plot the feature importance which is calculated by averaging the impurity-based feature importance of each tree (Scikit-Learn (2024c)). The feature importance plot after training the model for each location is given in Fig. 3.5. The ETClassifier models ranked the importance of each feature as expected in the correlation heatmap. For both St John's

and Yarmouth, RH at 2m has the highest predictive power, and RH at 2m lagged is second which makes sense as relative humidity persists through. In the heatmap (Fig. 2.2), we saw U at 10 m, which has a higher correlation to visibility for St John's and V at 10 m for Yarmouth. This is also shown in the feature importance, as we see the lagged winds at 10 m being 3rd most important. This indicates that the wind speed at 10 m one hour before has a good predictive ability for fog prediction, however, V at 10 m and lagged V at 10 m for Yarmouth is much less dominant as 3rd feature for Yarmouth compared with U at 10 m for St John's. As seen in Fig. 2.5 (the monthly fog occurrence plot), the time-based features have some predictive ability as well. The time of the day is important (hour) and which month it is. From Fig. 2.5, we have a higher chance of fog in April to June for St John's, and July and August for Yarmouth.

3.3 Long Short-Term Memory (LSTM)

3.3.1 Artificial Neural Network (ANN)

The neural network (NN) is a machine learning technique inspired by the way biological neural networks in the human brain work. McCulloch Warren S (1943) proposed the first NN of historical importance which consisted of a single neuron with a binary step function, where the weighted sum of the inputs and the threshold of the neuron gives a binary output of 0 or 1. This mimics how biological neurons work in a sense as the neuron will receive signals from its neighbor, and if the stimulus exceeds the threshold of the neuron, it will become activated. When interconnected, those neurons can perform any calculations a computer can do. The output of the neuron is given in equation 3.6, where y is the output, H is the step function, w_i is the weight of the stimulus x_i and b is the bias parameter.

$$y = H\Big(\sum_i w_i x_i + b\Big) \tag{3.6}$$

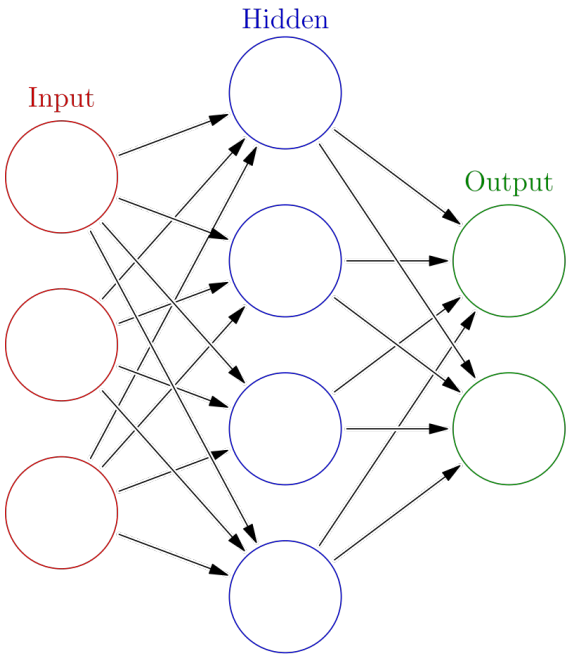


Figure 3.6: A simple neural network model where each node represents an artificial neuron, consisting of an input layer, hidden layer, and an output layer, all interconnected. Image by Glosser.ca (2013).

The next big advancement is the perceptron introduced by Rosenblatt (1958) and Rosenblatt (1962). It was the first simple neural network capable of learning weights to classify inputs. However, it was limited to linear problems and could not solve tasks like XOR (Hsieh (2023)). The evolution of neural networks stagnated until Werbos (1974) introduced the backpropagation algorithm for perceptrons to handle multi-layers. Backpropagation helps neural networks learn by efficiently adjusting their weights based on errors. It works in two main steps: first, the forward pass computes the output by passing inputs through the network layer by layer, and the loss function calculates the error between the predicted and actual values. Then, in the backward pass, the algorithm uses the chain rule to compute the loss gradients for each weight, allowing the optimizer (e.g., Adam by Kingma and Ba

(2017)) to update them and minimize the error. The chain rule states that for a composite function $f(g(x))$, the derivative is given by:

$$\frac{d}{dx}f(g(x)) = f'(g(x)) \cdot g'(x) \tag{3.7}$$

In neural networks, if the loss L depends on the activation a, which depends on the weight w, the gradient is computed as:

$$\frac{dL}{dw} = \frac{dL}{da} \cdot \frac{da}{dw} \tag{3.8}$$

This recursive application of the chain rule propagates errors backward through the layers, ensuring that each weight is updated in proportion to its contribution to the error. Backpropagation makes deep learning feasible by automating weight adjustments, allowing networks to learn complex patterns from data efficiently. In Fig. 3.6, we see a simple example of how a multi-layer NN works. The input layer will accept the raw data which is the selected features in our case, and the hidden layer will process the data to extract patterns and learn from the training data. Then, the output layer will produce the final results, either classification or regression.

The activation function also plays a major role in how NN works and the results you will obtain. Activation functions introduce non-linearity into neural networks, allowing them to learn complex patterns by calculating the output of the neuron using the input it receives and its individual weight. For our case, we used four different types of activation functions with advantages and disadvantages. For the input layer, we used the hyperbolic tangent (tanh) defined as,

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.9}$$

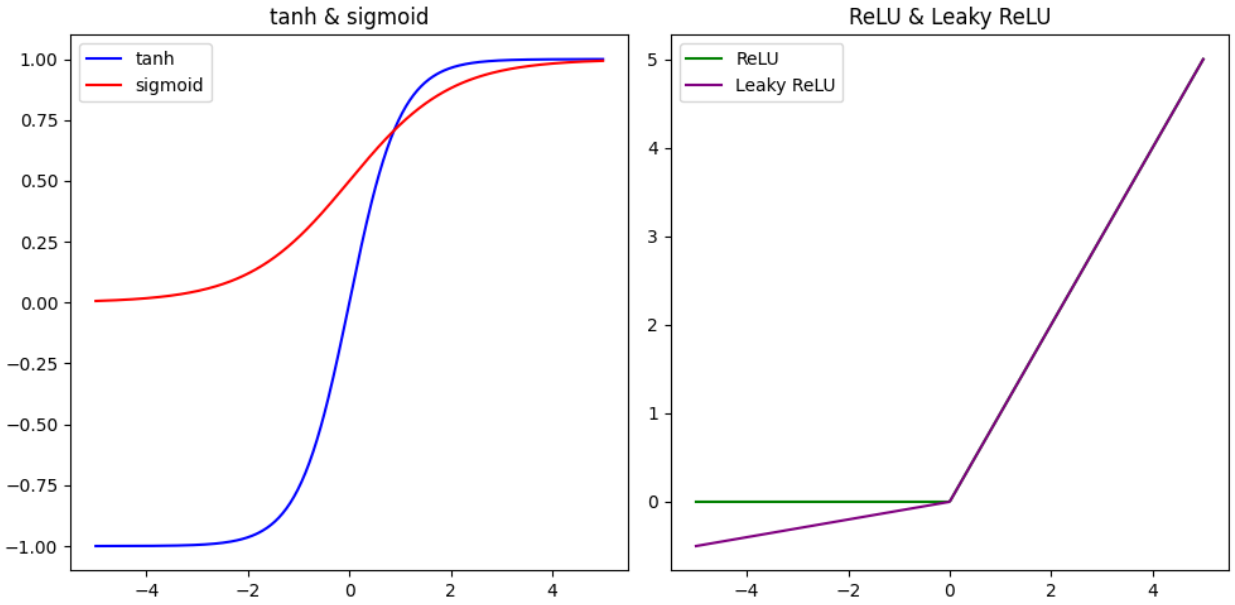


Figure 3.7: Activation functions used for the LSTM model for St John's and Yarmouth, which helps learn the non-linear patterns in data. On the left, we have the Sigmoid function in red, and Tanh in blue. On the right, we have ReLU in green and Leaky ReLU (with $\alpha = 0.1$) in purple.

which maps input values to the range $(-1, 1)$. The advantage of tanh is that it is zero-centered, as seen in Fig. 3.7 (left plot, blue line). This helps optimization, but it suffers from the vanishing gradient problem for very large or small inputs (Namin et al. (2009)). The vanishing gradient can be seen in the equation. Another widely used function is the sigmoid, given by

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{3.10}$$

which restricts outputs to $(0, 1)$ which is shown in Fig. 3.7 as well (left plot, red line). This makes it useful for binary classification as it converts very large values to 1 and very small values to 0. However, this also means that the function suffers from vanishing gradients and is not zero-centered, which can slow down the training process (Hinton et al. (2012)).

One way to overcome the vanishing problem during training and is more computationally efficient is to use the Rectified Linear Units (ReLU) activation function,

$$\text{ReLU}(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \tag{3.11}$$

However, it suffers from the dying ReLU problem, where neurons output zero for all negative inputs, effectively preventing them from learning (Kunihiko (1975)). This is shown in the right plot in green in Fig. 3.7. A modification, Leaky ReLU, solves this by allowing small negative values:

$$\text{Leaky ReLU}(x) = \begin{cases} \alpha x, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \tag{3.12}$$

where α is a small constant (e.g., 0.01). This prevents neurons from becoming inactive, although it introduces an extra hyperparameter to tune (Nair and Hinton (2010)).

3.3.2 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a class of ANNs designed for sequential data processing. Unlike traditional feed-forward neural networks (FNNs), where the input flows in a single direction from input to output, RNNs contain feedback loops. These loops allow information from previous steps in the sequence to be passed on to the current step, which makes RNNs well-suited for tasks involving time-dependent or sequential data.

As shown in Fig. 3.8, RNNs have a similar structure to FNNs. There is an input layer, hidden layers, and an output layer. The building blocks of RNNs are called recurrent units.

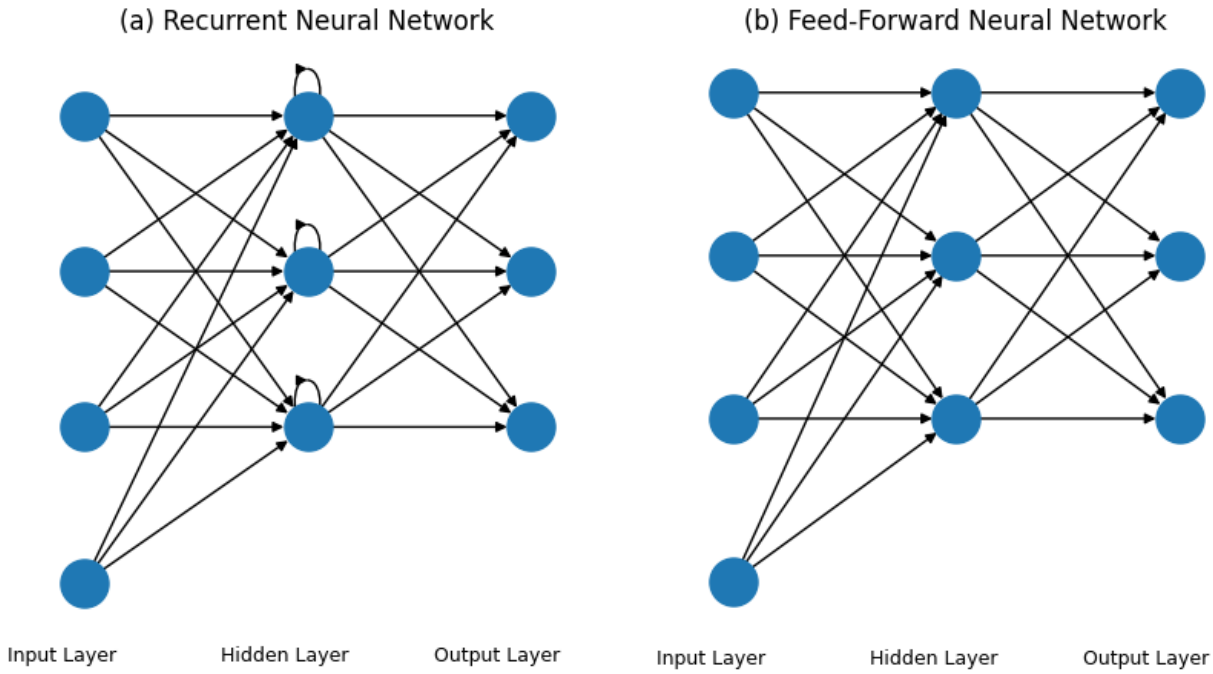


Figure 3.8: Comparison between Recurrent Neural Networks (RNNs) and Feed-Forward Neural Networks (FNNs). FNNs (ANNs and CNNs) only go in one direction, forward, while RNNs have feedback loops as seen in the hidden layer (aishwarya.27 (2024)).

They have hidden states which acts as internal memory that stores information from the previous timestep. At each time step, the hidden state of the network depends not only on the current input but also on the previous hidden state. This feedback loop allows the network to retain information over time, which is crucial for tasks like time-series forecasting. Essentially, the RNN has a function f_θ which will map $(x_t, h_t) \mapsto (y_t, h_{t+1})$, where x_t is the input vector, h_t is the hidden function, y_t is the output vector and θ is the NN parameters. Therefore, the hidden state at x_t is given by:

$$h_t = f(W_h h_{t-1} + W_x x_t + b) \tag{3.13}$$

The NN will map x_t to y_t using the hidden state given in equation 3.13, which acts as a memory. W_h, W_x, and b are the learned weights and parameters. h_{t-1} is the hidden state at

the previous time step, and f is the nonlinear activation function. So, the hidden state at time t, h_t is computed recurrently, enabling RNNs to handle sequential datasets. As data flow sequentially in RNNs, it is handled by updating its hidden state at each time step based on the previous state and the current input. To train an RNN, Gruslys et al. (2016) introduced a version of backpropagation called Backpropagation Through Time (BPTT), which accounts for the sequential nature of the data. The first step is RNN unfolding, that is, "unrolling" the RNN across all the time steps in the sequence, treating each time step in the sequence as a layer (aishwarya.27 (2024)). Then, the steps are the same as standard NN. The RNN computes the hidden state based on the input and the previous hidden state, the forward pass process. The loss is calculated and the error is propagated backward through time with the key difference that the gradients must also account for the dependencies between time steps. However, the main challenge when training RNNs with BPTT is the vanishing gradient problem. This occurs during the backward pass when the gradients of the loss function with respect to the weights are propagated backward through time. As we also used the previous timesteps, the gradients can decrease very rapidly as they propagate through time (Bengio et al. (1994)) or can become excessively large, that is, exploding gradients (Pascanu et al. (2012)). Therefore, standard RNNs are effective for short-term dependencies but tend to forget earlier information as the sequence length increases. This limitation is a direct result of the vanishing gradient problem, which impedes the model's ability to learn from long sequences.

3.3.3 LSTM

Hochreiter and Schmidhuber (1997) introduced a new model, the Long Short-Term Memory (LSTM) neural network which is a type of RNN. LSTMs were specifically designed to address the shortcomings of standard RNNs. They introduce a special kind of unit called a memory

cell (C_t), which allows the network to store information for long periods of time. This design helps overcome the vanishing gradient problem and allows the model to capture long-term dependencies.

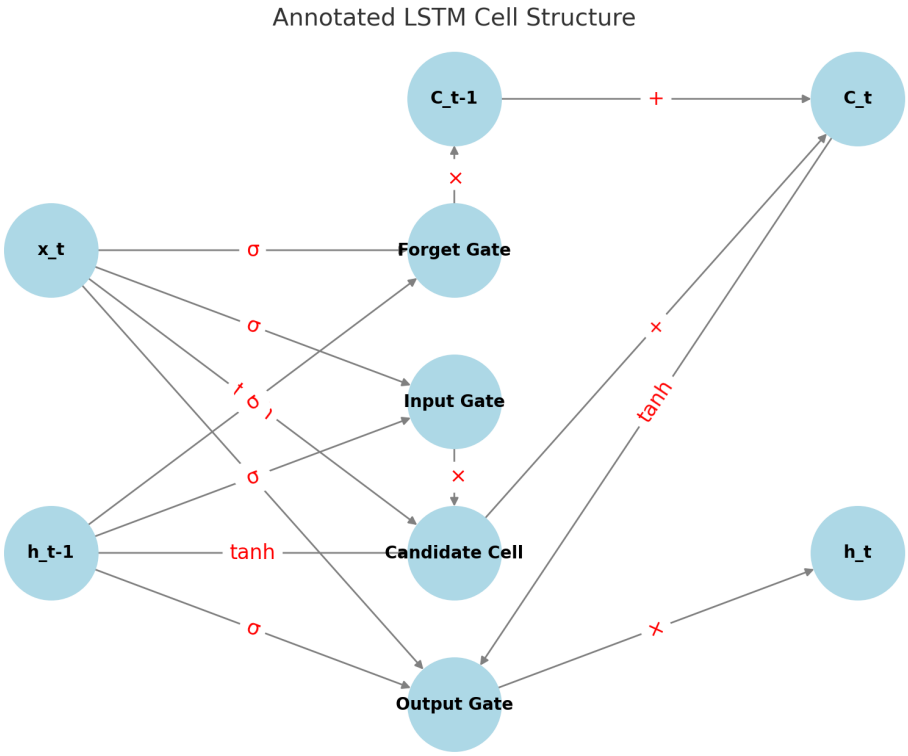


Figure 3.9: LSTM cell structure showing all the gates: input, output, and forget, together with the hidden states and cell state (Chug (2024)).

The cell structure of an LSTM model is given in Fig. 3.9. The core innovation in LSTMs lies in their use of gates that regulate the flow of information into and out of the memory cell. The 3 new key gates are the forget gate, f_t, the input gate, i_t, and the output gate, o_t. These gates determine which information should be remembered, which should be forgotten, and what new information should be added to the cell. The forget gate decides what information from the previous cell state should be discarded. It outputs a value between 0 and 1 (via a sigmoid function) for each value in the memory cell. If the value is close to 0, the information is forgotten and if it is close to 1, the information is retained:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{3.14}$$

, where W_f and U_f are the weights, h_{t-1} is the previous hidden state, x_t is the input and b_f is the bias. In Fig. 3.9, this is shown when the forget gate is multiplied by the previous cell state, C_{t-1}.

The input gate determines what new information should be stored in the cell state. This gate combines the output of a tanh function and the sigmoid function to add new information to the memory.

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{3.15}$$

$$\tilde{C}_t = \tanh(W_C x_t + U_C h_{t-1} + b_C) \tag{3.16}$$

In the figure, the input gate output and the candidate cell output are multiplied ($\times$) before being added ($+$) to the updated cell state. Therefore, the input gate controls how much new information should be stored in the cell state (Equation 3.15), and the candidate cell ($\tilde{C}_t$) computes potential new information using the tanh activation function (Equation 3.16). So, the forget gate determines how much of the previous state is kept, and the input gate adds any new important information. The new cell update is then given as the sum of the gates:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{3.17}$$

, where the $\odot$ represents the Hadamard product (Davis (1962)) which takes 2 matrices

of the same dimension and multiplies the elements at the same position. Finally, the output gate decides which part of the memory cell should be output as the hidden state for the next step or layer. It also uses a sigmoid function to control the amount of information passed forward.

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{3.18}$$

$$h_t = o_t \odot \tanh(C_t) \tag{3.19}$$

We can see how it works in Fig. 3.9. The output gate determines how much of the cell state should be used to compute the hidden state, h_t (Equation 3.18) and the new hidden state is the filtered version of the cell state passed through a tanh function and multiplied by the output gate activation (Equation 3.19). The combination of these gates allows LSTMs to selectively remember or forget information across long sequences, making them much better at capturing long-range dependencies than standard RNNs (Graves (2013)). For our fog forecasting case, we used the Bidirectional LSTM, proposed by Schuster and Paliwal (1997). Unlike a standard LSTM, which only maintains context from past inputs to future outputs, a BiLSTM captures information from both past and future contexts by utilizing two LSTM layers, one processing the sequence left to right and another processing it right to left. This leads to a better understanding of the weather data and better handling of long-term dependencies.

To build the biLSTM model, you have to follow a structured approach to sequential data processing, making it well-suited for forecasting and binary classification of the fog occurrence using the features we prepared for St. John's and Yarmouth. The full model layout is summarized in Table 3.4. The biLSTM model consists of an input layer, followed

Layer (Type)	Output Shape	Param #
Bidirectional LSTM (64)	(None, 1, 64)	12,032
BatchNormalization	(None, 1, 64)	256
Dropout (0.2)	(None, 1, 64)	0
Bidirectional LSTM (32)	(None, 64)	24,832
BatchNormalization	(None, 64)	256
Dropout (0.2)	(None, 64)	0
Dense (16)	(None, 16)	1,040
BatchNormalization	(None, 16)	64
LeakyReLU	(None, 16)	0
Dropout (0.2)	(None, 16)	0
Dense (16)	(None, 16)	272
BatchNormalization	(None, 16)	64
LeakyReLU	(None, 16)	0
Dropout (0.2)	(None, 16)	0
Dense (1, Sigmoid)	(None, 1)	17
Total Params:	115,861 (452.59 KB)	
Trainable Params:	38,513 (150.44 KB)	
Non-trainable Params:	320 (1.25 KB)	

Table 3.4: biLSTM Model Summary for both St John's and Yarmouth, Canada.

by two bidirectional LSTM layers, and two fully connected (dense) layers, before reaching the final output layer.

To be able to use biLSTM, we need to reshape our features dataset into [batch_size, time_steps, num_features]. For our case, the time_step is 1, features were 14 (features + lagged features) and batch_size is our input data. The input layer then takes in the time-series data. The first hidden LSTM layer has 64 units and returns sequences to feed into the second hidden LSTM layer, which has 32 units and outputs a fixed-length representation. Both LSTM layers use the tanh activation function and are wrapped in Bidirectional wrappers, allowing the model to learn from both past and future time steps. To improve stability, Batch Normalization (BatchNorm Ioffe and Szegedy (2015)) is applied after each LSTM layer, normalizing activations across the mini-batch, which helps reduce internal covariate

shift and speeds up training. Additionally, Dropout layers (0.2 probability Srivastava et al. (2014)) are introduced after each LSTM and dense layer to prevent overfitting by randomly deactivating a fraction of neurons during training.

After using the biLSTM model, we have 2 dense hidden layers. Dense layers are used to compress and transform these learned features into a meaningful representation. This allows the model to focus on the most relevant aspects of the time-series data before making a prediction. The fully connected layers use LeakyReLU activation, which helps to avoid dead neurons and allows better gradient flow. From Singh et al. (2023), the final output layer is a single neuron with a sigmoid activation function, making it suitable for binary classification. The model has a total of 115,861 parameters, with 38,513 trainable parameters, which is close to your dataset size of 33,000 samples, ensuring a good balance between model complexity and generalization.

To run the model, we need to compile it with various configurations. For binary classification, we use the binary cross entropy loss function which calculates the log loss between the true labels and the predicted probabilities which is the same as the Shannon entropy described earlier. We used the Adam optimizer(Kingma and Ba (2017)) with a learning rate of 0.0001 to update the weights based on the gradients calculated from the loss function. A learning rate of 0.0001 ensured stability in the training process and it was not too small that the model took too long to converge. In addition, to track the performance of the loss function, we used a set of metrics suitable for binary classification given below which was used by Tensorflow (2024) for dealing with imbalanced data.

```
METRICS = [
keras.metrics.BinaryCrossentropy(name='cross_entropy'),
keras.metrics.MeanSquaredError(name='Brier_score'),
keras.metrics.TruePositives(name='tp'),
```

```
    keras.metrics.FalsePositives(name='fp'),
    keras.metrics.TrueNegatives(name='tn'),
    keras.metrics.FalseNegatives(name='fn'),
    keras.metrics.BinaryAccuracy(name='accuracy'),
    keras.metrics.Precision(name='precision'),
    keras.metrics.Recall(name='recall'),
    keras.metrics.AUC(name='auc'),
    keras.metrics.AUC(name='prc', curve='PR'),  # Precision-recall curve
    F1Score()]
```

We used a batch size of 128 samples and set the epochs to 200. Setting the epoch to 200 allowed the model to run through the entire training dataset enough to learn more effectively and converge to a better solution. Since processing individual data sample is very expensive computationally, we use a batch size of 128 which refers to the number of training samples that are processed before the model's internal parameters (such as weights and biases) are updated (McCandlish et al. (2018)).

To enhance the training process, two callbacks are used: EarlyStopping and ReduceLROnPlateau. The EarlyStopping callback monitors the validation F1 score ('val_f1_score') and halts the training if there is no improvement for 20 epochs. This prevents overfitting by ensuring that training stops when the model performs on the validation set plateaus. The model's best weights, achieved during training, are restored when this callback is triggered. On the other hand, the ReduceLROnPlateau callback dynamically adjusts the learning rate. If the validation F1 score does not improve for 5 epochs, the learning rate is reduced by a factor of 0.5, with a lower bound of 1e-7. This helps the model escape local minima and potentially improve its convergence behavior, particularly when performance plateaus.

Figure 3.10 shows the training and validation F1-score and loss curves over the 200 epochs for St John's, Canada. The blue line is the when the model is learning from the training

Figure 3.10: BiLSTM model performance using the f1-score metric, Fig. 3.10(a), and the loss function value, Fig. 3.10(b) over epochs between the training set and validation set for St John's, Canada (blue line- training dataset and orange line- validation set).

dataset and orange line is when the model is being evaluated with the validation set. We can see that early stopping actually stop the run to around 50 epochs. The model exhibited rapid learning in the initial epochs, with the F1-score increasing sharply and the loss decreasing significantly. After approximately 20 epochs, the performance reached a plateau, indicating convergence. Additionally, the validation F1-score and loss closely tracked the training metrics throughout the training process, suggesting that the model generalized well and did not overfit to the training data. The final F1-score on the validation set was 0.50, demonstrating the model's ability to predict fog occurrences. The stable performance in later epochs indicates that further training beyond the 60 epochs is unlikely to yield significant improvements.

In Fig. 3.11, we plotted the same graphs for Yarmouth, Canada. The model ran much longer for Yarmouth, around 100 epochs, suggesting a longer learning process for the latter. We noticed that the validation performs better than the training and Tensorflow (2024) ex-

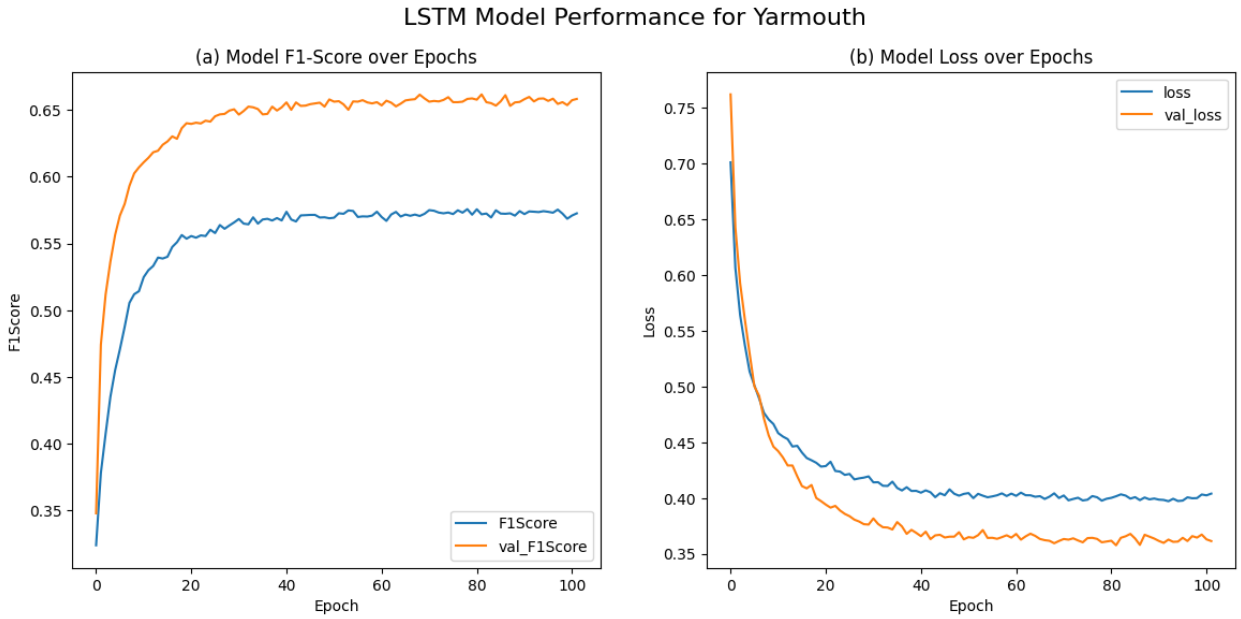


Figure 3.11: BiLSTM model performance using the f1-score metric, Fig. 3.10(a) and the loss function value, Fig. 3.10(b) over epochs between the training set and validation set for Yarmouth, Canada (blue line- training dataset and orange line- validation set).

plains that not having a dropout layer when evaluating the model with the validation set can make it behave better. Similarly, the model demonstrated strong initial learning with rapid improvements in the F1-score and a decline in loss. The training F1-score steadily increased before stabilizing around 0.56, while the validation F1-score exhibited a higher peak, reaching approximately 0.66. The model's ability to maintain a stable validation F1-score across epochs without significant fluctuations suggests that it generalized well to unseen data. The loss curves followed a smooth downward trend, confirming effective model optimization and we observe again that the validation loss was consistently lower than the training loss. Therefore, the better generalization and F1-score observed in Yarmouth compared to St. John's may suggest regional differences as illustrated in the data exploration section.

Chapter 4

Results and Discussion

4.1 Results

We trained both models using the ERA5 reanalysis data from March to August (2012 to 2023) which was down scaled using the WPS, and the visibility reports. We divided the training and validation dataset as 64% and 16% with the test set as 20%. We ensured with the training and validation data sets that both models were generalizing well and not overfitting. Since we split the data considering the nature of the time component, the 20% test data was approximately the year 2023. This means that we tested our trained models with the 2023 dataset for both locations and saw how they performed.

As mentioned earlier, fog is a rare occurrence, accounting for only 11% to 12% of our dataset. Due to this class imbalance, we had to adjust our data preparation and model-building approach. Now, the evaluation of the test dataset will also require a different strategy to ensure an accurate assessment of fog. To be able to visualize the performance of the model, we plotted four variables in a matrix. It is important to know how much the model accurately predicted "clear" which we will denote as true positive and how much of

the "clear" was predicted wrongly, false positive. Then, more importantly in our case, how much fog is actually predicted, true negative, and how much fog is predicted wrongly, false negative (Google For Developers (2024f)). This denotation is interchangeable, that is, true positive can mean fog as well, and "clear" means true negative.

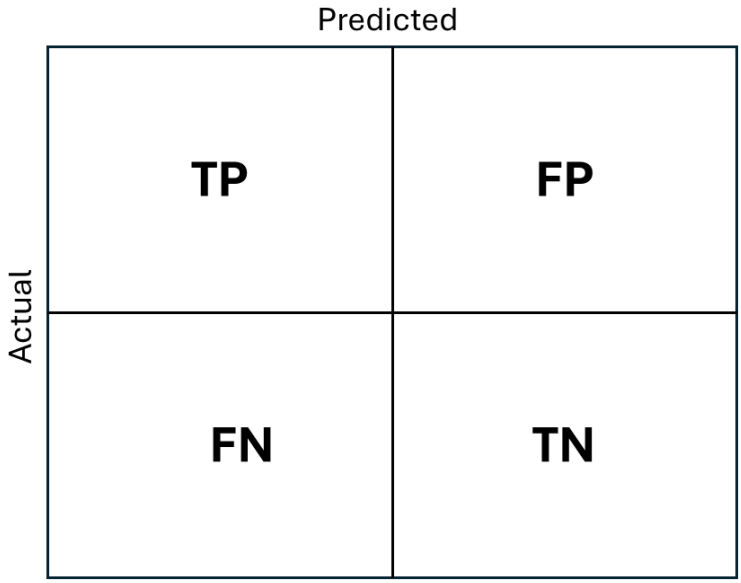


Figure 4.1: Example of confusion matrix showing true positive (TP), false positive (FP), false negative (FN), and true negative (FN) between actual events and the predicted events from the ML model.

In Fig. 4.1, an example of a confusion matrix is shown which helps us visualize how much fog we predicted correctly and how many "clear" hours we got as well. Usually, to evaluate the performance of the model, we calculate the accuracy between the predicted labels and the actual with the equation given:

$$Accuracy = \frac{\text{correct classifications}}{\text{total classifications}} = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.1}$$

However, if the true positive is very large, which is common for imbalanced dataset (our case), the accuracy of the model is very high, even if we are not able to accurately predict fog. This leads to a false sense of how good the model is performing (Google For Developers

(2024a)) and therefore other metrics should be used. We evaluated the recall and precision of the test dataset. The recall looks at the proportion of true positives that are correctly classified by the model:

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

For fog occurrence, recall is how many fog classes the model got correctly out of the total number of fog cases. The false negative, in this case, is the amount of fog our model falsely labeled as "clear". The next important metric is precision given by:

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

Precision is the proportion of actual fog that our model got accurately from all the fog predictions it did. This helps us to understand whether our model is just predicting a lot of fog and bumping up the performance or if it can differentiate between clear and fog well. Precision and recall are closely related, as seen in Equations 4.2 and 4.3, where they show an inverse relationship. Precision is high when false positives are low, meaning the model rarely misclassifies clear conditions as fog. In contrast, recall is high when false negatives are low, indicating that the model successfully captures most fog occurrences. As both false positives and false negatives are related, an inverse relationship between precision and recall is established. An important aspect of this relationship is adjusting the threshold of the model to get a good balance between the two metrics. This helps in identifying fog classes without overpredicting and underpredicting them.

In cases of class imbalance, where fog is significantly underrepresented, precision and recall alone may not provide a comprehensive measure of model performance. Instead, the F1-score, which is the harmonic mean of precision and recall, offers a more balanced

evaluation. It ensures that both metrics are considered together, making it particularly useful when dealing with imbalanced datasets where optimizing for just one metric can lead to misleading conclusions. The equation is given by:

$$F_1 \text{ score} = 2\frac{precision \cdot recall}{precision + recall} = \frac{2TP}{2TP + FN + FP} \tag{4.4}$$

Below, we will show all the results we obtained from both ExtraTreesClassifier and biLSTM using a classification report showing the precision, recall, and F1-score for both clear and fog classes for St John's and Yarmouth for the 2023 dataset and their confusion matrices.

4.1.1 ExtraTreesClassifier Results

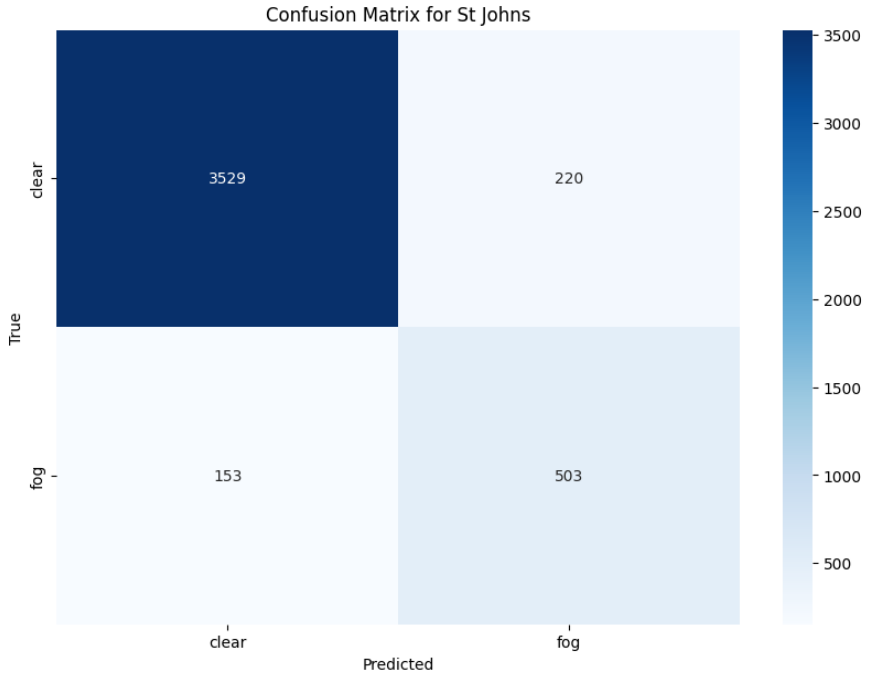
Earlier, we saw that the ETClassifier model is generalizing well by looking at the time series cross-validation accuracy scores during training, so the next step is to see how it does when predicting with new data, the test dataset which is the year 2023.

Class	Precision	Recall	F1-Score	Support
clear	0.96	0.94	0.95	3749
fog	0.70	0.77	0.73	656
Accuracy		0.92		4405

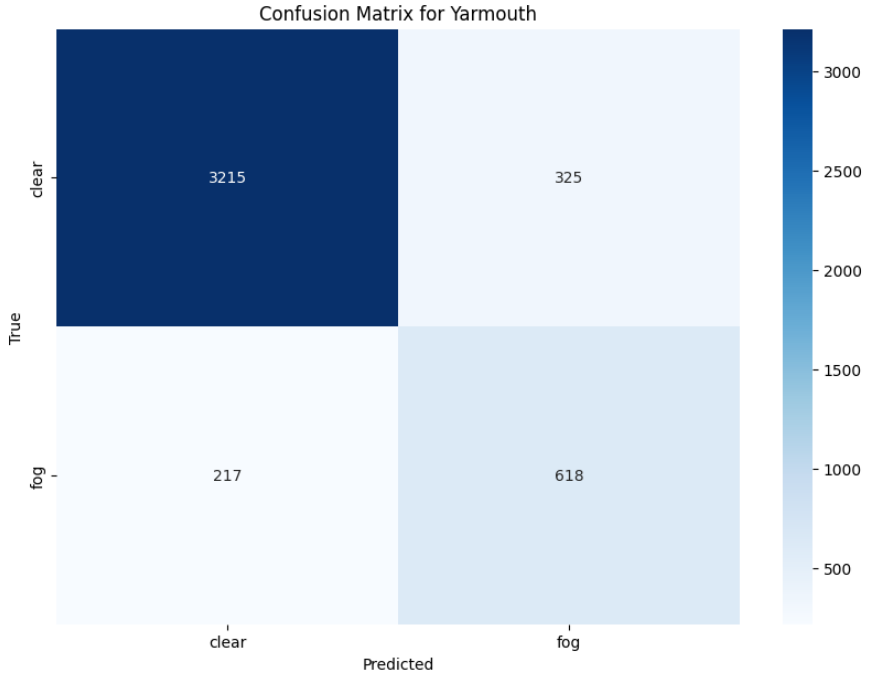
Table 4.1: Classification Report for St John's, Canada on the 2023 test data using ETClassifier.

Class	Precision	Recall	F1-Score	Support
clear	0.94	0.91	0.92	3540
fog	0.66	0.74	0.70	835
Accuracy		0.88		4375

Table 4.2: Classification Report for Yarmouth, Canada on the 2023 test data using ETClassifier.



(a) Confusion matrix for St John's, Canada.



(b) Confusion matrix for Yarmouth, Canada.

Figure 4.2: Confusion matrix for both St John's and Yarmouth, Canada on the test dataset (year 2023) for the binary classification of fog using the ETClassifier model.

Above we have the classification report for St John's in Table 4.1 and its confusion matrix in Fig. 4.2(a) and for Yarmouth in Table 4.2 and the confusion matrix in Fig. 4.2(b).

For St John's, we have a good recall of 0.77 and an F1-score of 0.73. We can observe how using the F1-score as our evaluation metric is better than accuracy. The accuracy of the model is 0.92, while the F1-score of fog is 0.73. This shows a more precise performance of our model. While the report gives us insight into how well the model is doing, the confusion matrix shows us how much fog our model captured. Out of the 656 hours of fog, it was able to predict 503 hours and misclassified 220 clear hours out of 3749. The precision for fog (0.70) suggests that 30% of predicted fog cases are clear conditions (false positives), while the recall (0.77) indicates that 23% of actual fog cases are missed (false negatives). The F1-score (0.73) provides a balanced measure, showing room for improvement in distinguishing fog from clear conditions.

For Yarmouth, the accuracy of the model is around 88%. This is close to the accuracy during the training phase which shows good generalization of our model. The precision for fog (0.66) indicates that about 34% of predicted fog cases are clear (false positives), while the recall (0.74) shows that 26% of actual fog cases are missed (false negatives). This gives out an F1-score of 0.70. The confusion matrix shows 325 false positives, meaning the model tends to overpredict fog, which could lead to unnecessary alerts. Additionally, the 217 false negatives out of the 835 suggest that a notable number of fog events are missed, which is critical. Compared to St. John's, Yarmouth's model has slightly lower precision and recall for fog, which may be influenced by the weather characteristics differences.

4.1.2 biLSTM Results

We also tried to use neural networks for the binary classification of fog occurrence and take advantage of the ability of neural networks to capture nonlinear complexities very well. By

using a complex model such as the biLSTM that can capture long-term temporal dependencies, we wanted to see a comparison with a random forests model such as ETClassifier and also which one is more appropriate for our case. Again, we trained the model, but this time we used a simple time series split for the validation set and plotted the loss and F1-score plots of the model which showed how the model was learning and generalizing during the epochs.

Class	Precision	Recall	F1-Score	Support
clear	0.96	0.94	0.95	3749
fog	0.70	0.75	0.72	656
Accuracy		0.91		4405

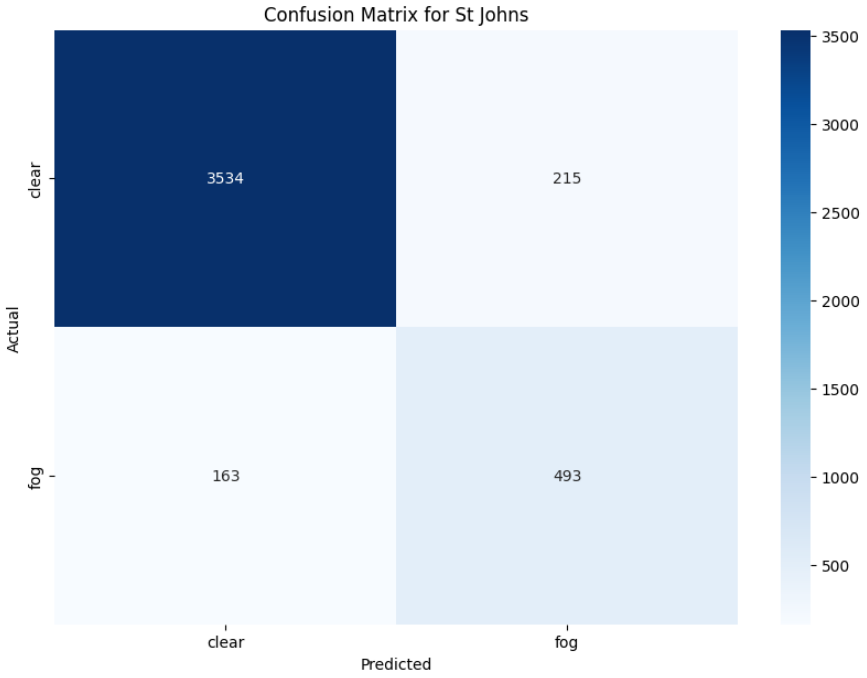
Table 4.3: Classification Report for St John's, Canada on the 2023 test data using biLSTM.

Class	Precision	Recall	F1-Score	Support
clear	0.93	0.92	0.92	3540
fog	0.66	0.69	0.68	835
Accuracy		0.87		4375

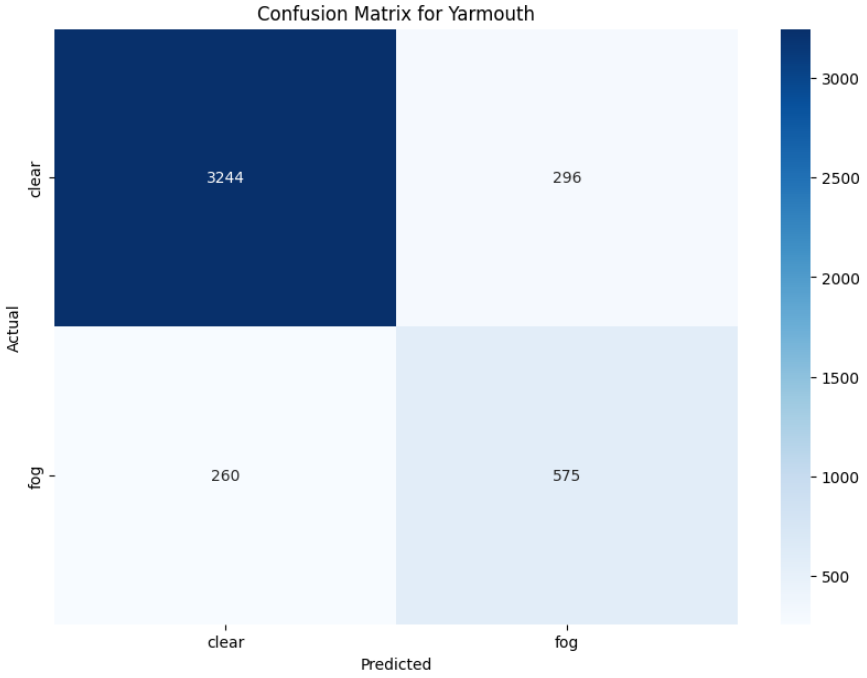
Table 4.4: Classification Report for Yarmouth, Canada on the 2023 test data using biLSTM.

We have the classification reports using biLSTM for St John's in Table 4.3 with its confusion matrix in Fig. 4.3(a) and for Yarmouth in Table 4.4 with its confusion matrix in Fig. 4.3 (b) for the test set (the year 2023).

With the biLSTM model for St. John's, the classification performance remains similar to the previous ETClassifier model but shows some differences in recall and F1-score for fog detection. The overall accuracy remains high (91%), and the precision for fog (0.70) is unchanged, indicating that the false positive rate has not significantly improved. However, the recall has slightly decreased from 0.77 to 0.75, meaning the model is now missing slightly more actual fog cases. The F1-score for fog (0.72) has also dropped slightly from 0.73, reflecting the trade-off between precision and recall. The confusion matrix confirms this:



(a) Confusion matrix for St John's, Canada using biLSTM.



(b) Confusion matrix for Yarmouth, Canada using biLSTM.

Figure 4.3: Confusion matrix for both St John's and Yarmouth, Canada on the test dataset (year 2023) for the binary classification of fog using the biLSTM model.

493 fog cases were correctly identified, while 163 were missed. Compared to the ETClassifier model, this suggests that while the biLSTM model may capture bidirectional dependencies in the data, it does not necessarily lead to a substantial improvement in fog classification for St John's.

For Yarmouth using the biLSTM model, the overall performance remains similar to the ETClassifier model, with a slight shift in recall and F1-score for fog detection which is the same pattern as for St John's. The accuracy decreased by 1% (87%), and the precision for fog (0.66) is unchanged, indicating that the model's false positive rate has not improved. However, the recall for fog has dropped from 0.74 to 0.69, meaning the model is now missing more actual fog cases. From the confusion matrix, the biLSTM model correctly identified 575 fog cases, but it also misclassified 260 actual fog cases as clear- an increase in false negatives compared to the ETClassifier model. Meanwhile, 296 clear cases were mistakenly predicted as fog, suggesting that the model still struggles to differentiate between the two classes effectively. The F1-score for fog (0.67) has decreased slightly from 0.70, indicating that the model's ability to balance precision and recall has worsened.

We have seen that there is similar difference with both ML models between St John's and Yarmouth. Both ML models give higher accuracy, recall, precision and F1-score at St John's compared to Yarmouth. When looking at the 2024 dataset, the difference is even more apparent. This likely stems from the differences in how the key meteorological features influence fog formation at each location. We have seen from the feature importance plot in Fig. 3.5 that the most dominant feature is RH2, followed by RH2 lagged by 1 hour. The major difference is in the winds that influence fog at both locations. For St John's, U at 10 m is the second most important features (U10 and lagged U10) while for Yarmouth it is only the lagged V at 10 which is the third most contributing factor. In addition, at St John's, the U at 10 and its lag show relatively higher importance (0.13) compared to Yarmouth, where

the V at 10 (0.06) is more influential. Since RH2 is the highest contributor for Yarmouth, this can explain the difference as the model is relying highly on it. It is also possible that at St John's, fog is more sensitive to zonal (east-west) moisture transport and wind-driven advection effects, which are more consistently captured by the model. In contrast, Yarmouth's fog events may involve more complex or localized influences, potentially making the classification task noisier and reducing model performance. Additionally, Yarmouth may experience more transient or mixed visibility conditions that blur the distinction between fog and no-fog events, contributing to a lower F1-score.

Based on both locations, we see the same patterns when comparing biLSTM and ET-Classifier with ETClassifier slightly outperforming biLSTM. This may be due to ETClassifier having a higher tolerance to noisy data such as weather data and using an ensemble of trees to reduce the variance in prediction. Therefore, for our case of fog prediction, the ETClassifier appears to be more suitable. ETClassifier is also a decision tree based model which can be more suitable for solving weather tasks which often relies on the condition or threshold of a meteorological feature. For example, RH2 needs to be high enough to have enough moisture for fog formation. Furthermore, since ETClassifier is less complex than biLSTM which is used for deep learning models, the computational time is significantly less. ETClassifier took around 30-45 secs to train and give results while biLSTM took around 6 min to run, making ETClassifier more than 6× quicker and more efficient.

4.1.3 24-hour forecast 2024 dataset

Now that we looked at the performance of our models with the test dataset and saw that the models are generalizing well to unknown data, we tried to forecast fog occurrence in the next 24 hours using our ML models to predict. We have been using features (RH2, T2, P_sfc, U at 10 m and V at 10 m and the time-based features) to predict if there is fog or

not (binary classification). This allows our model to learn from those features and predict fog, but in terms of forecasting, we are not extrapolating through time but rather predicting at that specific time. Forecasting involves using past or present data to predict the future while predicting using ML learns patterns from input features and provides an output, but it does not necessarily account for the sequence of events (cherriebllxk7 (2024)).

There are many challenges in ML forecasting. Forecasting needs a lot of time series data availability that may contain missing values, have irregular sampling, or have abrupt changes due to external factors. Additionally, preparing the data is more complex due to more advanced feature engineering such as creating lag variables, rolling averages, or trend components. This means that the model complexity needs to be much higher so that it can handle sequence data, often with recurrent architectures like LSTMs, or attention-based models like Transformers, which demand more computational resources and tuning (Lam et al. (2023) and Matthias (2018)). Bi et al. (2022) also suggests that forecasts over multiple time steps suffer from compounding errors, that is, small errors in early predictions snowball into larger deviations over time. Therefore, ML models struggle when extrapolating beyond training data, which makes long-term forecasting challenging as Gultepe et al. (2024a) noticed when trying to forecast marine fog. They were able to get decent results by forecasting a one-hour lead time of fog.

So, we decided to approach the issue differently. The features we used for our ML models are the main meteorological variables and WRF does a good job of forecasting them, especially in summer with the largest error for wind components (Bughici et al. (2019) and Pappa et al. (2023)). We decided to use the forecast values of those variables which is essentially "extrapolated" through time and then use our ML models to predict fog occurrence from them. However, instead of the GFS Final Analysis data, we used GFS forecast values, available every three hours at a 0.25-degree resolution (National Centers for Environmental

Prediction, National Weather Service, NOAA, U.S. Department of Commerce (2015b)). Instead of doing long simulations of 6 months , we ran WRF pseudo-operationally for 36 hours with identical physical parameterization settings, extracting variables after a 12-hour spin-up period. These forecasted variables were then processed using the same data preparation pipeline and fed into our trained ML models to evaluate their performance on new, unseen data. This approach allowed us to assess whether the models could be used operationally to forecast fog on an hourly basis for the next 24 hours. The results are given below where a direct comparison between ETClassifier and biLSTM are made:

St John's

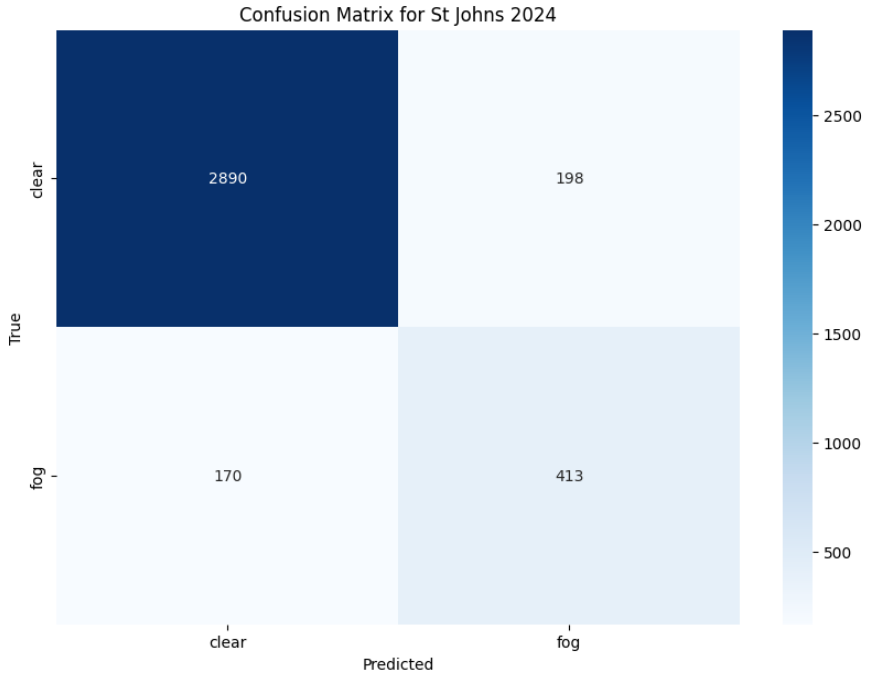
Class	Precision	Recall	F1-Score	Support
clear	0.94	0.94	0.94	3088
fog	0.68	0.71	0.69	583
Accuracy		0.90		3671

Table 4.5: Classification Report for St John's, Canada on the 2024 forecast data using ETClassifier.

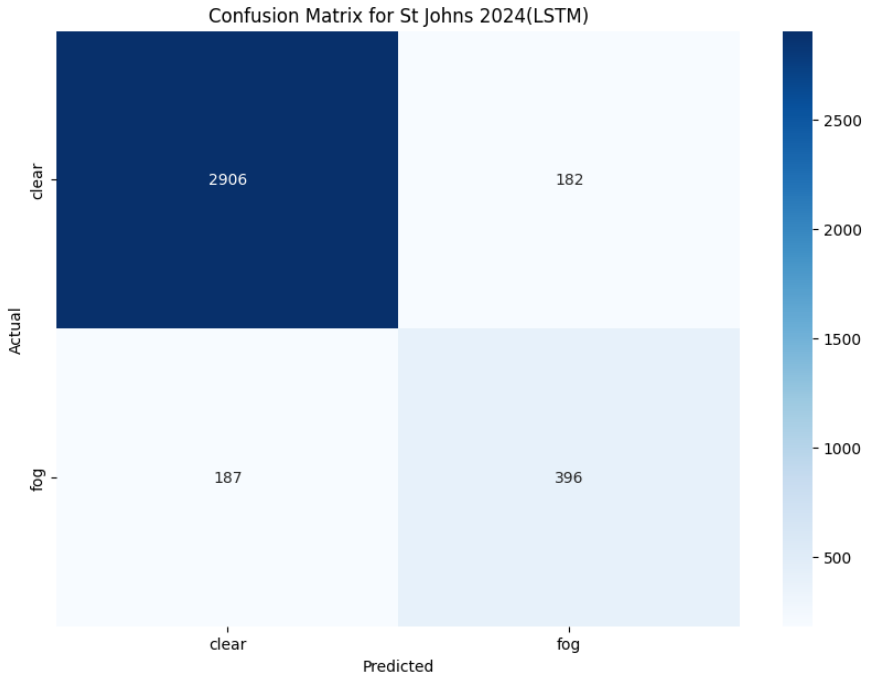
Class	Precision	Recall	F1-Score	Support
clear	0.94	0.94	0.94	3088
fog	0.69	0.68	0.68	583
Accuracy		0.90		3671

Table 4.6: Classification Report for St John's, Canada on the 2024 forecast data using biLSTM.

For both the ETClassifier and biLSTM models, the overall accuracy decreased by 1%, which is expected when testing with new data. Both models performed equally well in classifying clear conditions, achieving a precision, recall, and F1-score of 0.94. However, there were slight differences in classifying fog. Recall, the most critical metric for our ML approach, was higher for the ETClassifier (0.71) compared to biLSTM (0.68). While biLSTM



(a) Confusion matrix for St John's, Canada using ETClassifier.



(b) Confusion matrix for Yarmouth, Canada using biLSTM.

Figure 4.4: Comparison between ETClassifier and biLSTM using the confusion matrix for St John's, Canada on the forecast dataset (year 2024) for the binary fog classification.

had a marginally better precision by 0.01, the overall F1-score was slightly higher for the ETClassifier (0.69 vs. 0.68), indicating better overall performance in distinguishing between clear and fog conditions. The confusion matrix further supports this: the ETClassifier misclassified 170 fog instances, whereas biLSTM missed 187. However, biLSTM demonstrated a slight advantage in correctly identifying fog cases, misclassifying 182 instances as clear compared to 198 for the ETClassifier. Ultimately, the ETClassifier correctly predicted 413 fog labels, while biLSTM correctly identified 396, contributing to the small difference in their F1-scores.

Yarmouth

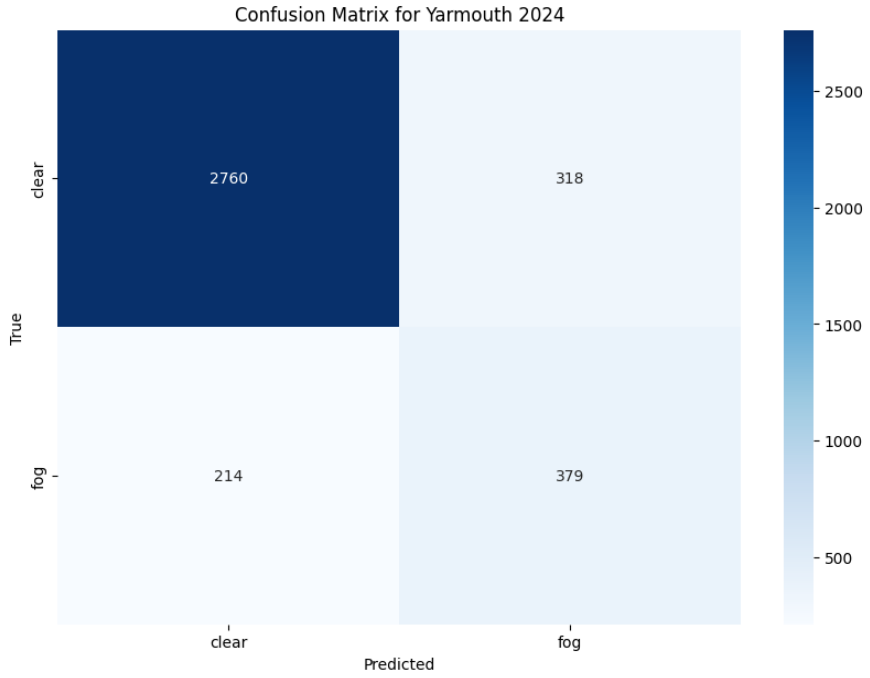
Class	Precision	Recall	F1-Score	Support
clear	0.93	0.90	0.91	3078
fog	0.54	0.64	0.59	593
Accuracy		0.86		3671

Table 4.7: Classification Report for Yarmouth, Canada on the 2024 forecast data using ETClassifier.

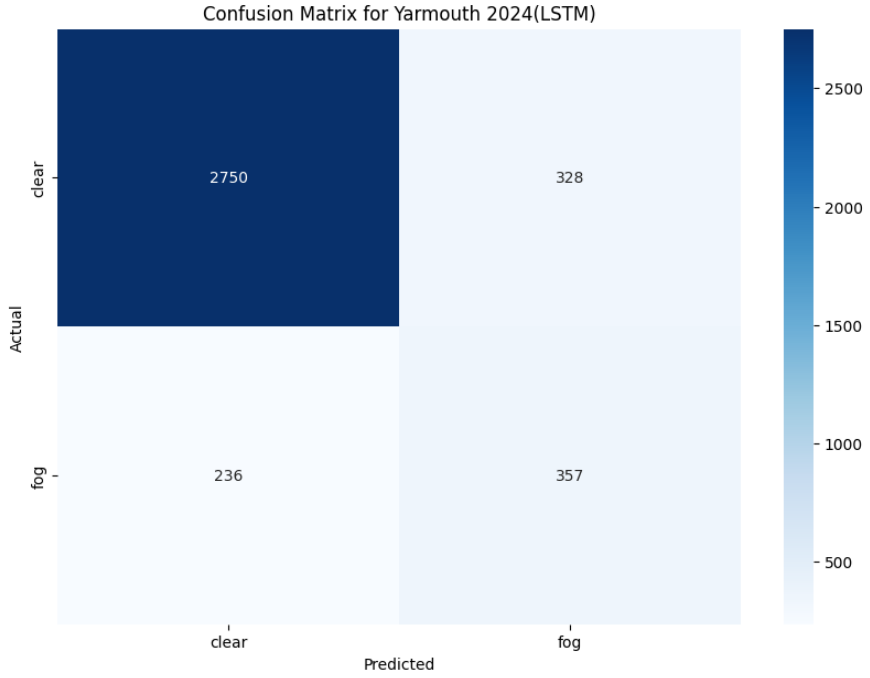
Class	Precision	Recall	F1-Score	Support
clear	0.92	0.89	0.91	3078
fog	0.52	0.60	0.56	593
Accuracy		0.85		3671

Table 4.8: Classification Report for Yarmouth, Canada on the 2023 forecast data using biLSTM.

For Yarmouth, there is a more noticeable difference in the performance of our ML models with the forecasted features from WRF. The overall accuracy dropped by 1% for ETClassifier (0.86) and by 2% for biLSTM (0.85), which is expected again for new data. Notably, both models experienced a significant decline in performance, likely due to data quality as we seen



(a) Confusion matrix for Yarmouth, Canada using ETClassifier.



(b) Confusion matrix for Yarmouth, Canada using biLSTM.

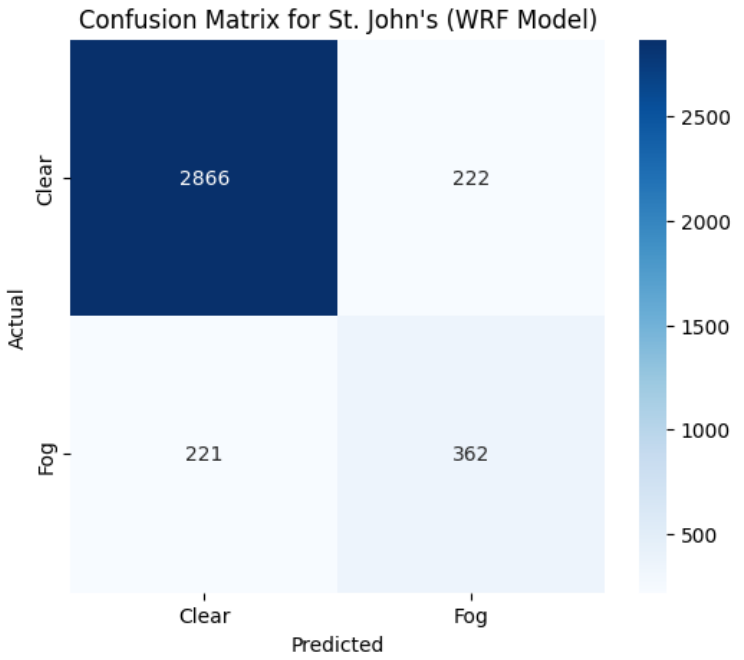
Figure 4.5: Comparison between ETClassifier and biLSTM using the confusion matrix for Yarmouth, Canada on the forecast dataset (year 2024) for the binary fog classification.

that the RH2 for Yarmouth was 100% while the weather reports indicated much drier air which led to a higher fog occurrence predicted by the ML models as RH2 is an important feature with high predictive ability. The F1-score dropped by approximately 10%, with ETClassifier achieving 59% and biLSTM 56%. However, when directly comparing the two, ETClassifier consistently outperformed biLSTM in fog classification across all metrics. As shown in the confusion matrix (Fig. 4.5), ETClassifier correctly identified more actual fog cases (379 vs. 357), produced fewer false positives (318 vs. 328), and had fewer false negatives (214 vs. 236), demonstrating its superior ability to distinguish between fog and clear conditions.

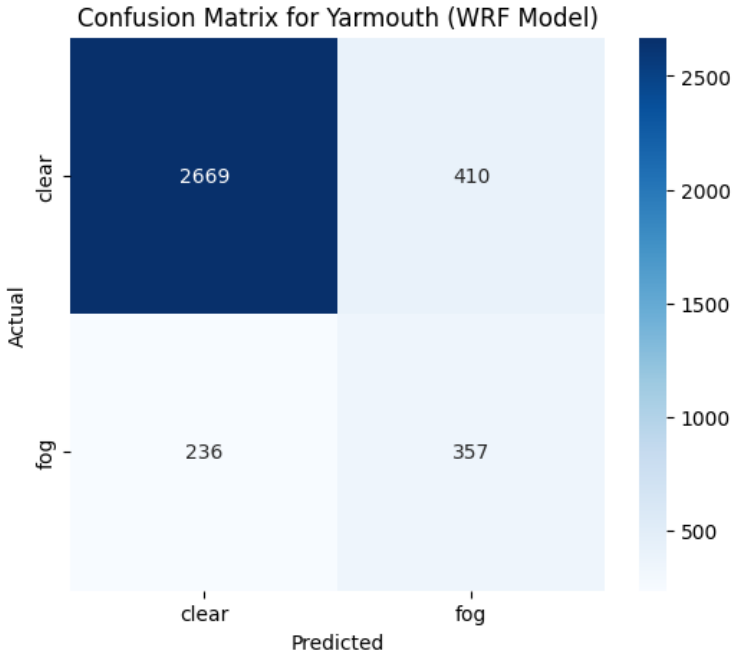
Direct forecast using the LWC in the WRF Model

We were able to use the ML models pseudo-operationally and showed that they were getting good results with new, unforeseen data twice, with the test data and 2024 dataset. The next crucial step is to compare their performance with other forecasting methods, specifically the WRF model, which is commonly used for operational marine fog prediction based on surface liquid water content. For this comparison, we ran the WRF model on the 2024 dataset and extracted the cloud mixing ratio at 2 meters. To classify fog in a binary manner, we labeled any instance where the cloud mixing ratio was nonzero (indicating the presence of liquid water) as fog, and instances where it was zero as clear. This approach aligns with WRF's tendency to predict high liquid water content which always dropped the visibility level in the fog zone. The results are summarized in the confusion matrix (Fig. 4.6), and a direct comparison between ETClassifier (the better-performing ML model) and WRF for both locations is presented in Table 4.9.

The comparison in Table 4.9 highlights that the ETClassifier+WRF approach outperforms the WRF operational model in fog classification for both St. John's and Yarmouth.



(a) Confusion matrix for St John's, Canada for the WRF model.



(b) Confusion matrix for Yarmouth, Canada for the WRF model.

Figure 4.6: Confusion matrix for St John's and Yarmouth, Canada on the forecast dataset (year 2024) for the binary fog classification using WRF pseudo-operationally.

	St. John's			Yarmouth		
Model	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
ETClassifier	0.68	0.71	0.69	0.54	0.64	0.59
WRF	0.62	0.62	0.62	0.47	0.60	0.53

Table 4.9: Performance comparison of ETClassifier+WRF and WRF Pseudo-operational for St. John's and Yarmouth for fog classification.

For St. John's, ETClassifier achieves higher precision (0.68 vs. 0.62), recall (0.71 vs. 0.62), and F1-score (0.69 vs. 0.62), indicating a more balanced and effective classification of fog events. In Yarmouth, the performance gap is even more noticeable, with ETClassifier showing improvements in all metrics: precision (0.54 vs. 0.47), recall (0.64 vs. 0.60), and F1-score (0.59 vs. 0.53). The higher recall values suggest that ETClassifier is better at detecting fog events compared to WRF, which is crucial for operational forecasting. In general, these results indicate that the integration of machine learning with WRF improves fog prediction, particularly by reducing missed fog events while maintaining reasonable precision.

This difference is significant. As shown in Fig. 4.6, we see that WRF correctly identified 362 fog labels while ETClassifier identified 413 and also misclassified 170 times compared to WRF's 221 for St John's. WRF also overestimated fog more than ETClassifier (222 vs. 198), which is more false positives. Additionally, WRF does a much poorer job in distinguishing fog and clear labels for Yarmouth as it got 357 fog correct and labeling fog wrong at a higher percentage, 410 times. ETClassifier, while underperforming compared to the 2023 dataset, still got 379 fog labels and 318 wrong. Moreover, WRF not only overestimated fog more frequently than ETClassifier but also missed more actual fog events. For St John's, it missed 51 more fog events than ETClassifer and for Yarmouth, it missed 22 fog events more than ETClassifier. This consistent underperformance of WRF in both locations highlights that ETClassifier is more effective at accurately distinguishing between fog and clear conditions, making it a more reliable model for operational fog prediction.

By having a post-processing approach with WRF and machine learning algorithms, we were able to outperform the WRF model for the 24-hour coastal fog prediction with an hourly resolution. While direct quantitative comparisons with prior studies are constrained by differences in data sources and forecast settings, the relative performance of our approach aligns well with or in many cases exceeds the results reported in similar classification-based ML studies, despite those often focusing on shorter lead times. This highlights the strength of combining high-resolution numerical weather predictions with machine learning to enhance site-specific fog forecasting over longer time horizons. Miao et al. (2020) used LSTM for short-term fog forecasting using weather stations in Anhui, China and provided the F1-score for hourly forecasts up to 4 hours. They were able to obtained an F1-score of 0.758 in the first hour, 0.66 in the next hour, 0.53 on hour 3 and 0.474 on the final hour. These scores reflect a declining trend over time because they separated the data in batch for the hours respectively and trained their model for each forecast hour. In our case, we had the F1-score for the 24 hour prediction at an hourly resolution. For St John's, we obtained an F1-score of 0.69 and for Yarmouth we got 0.59. While Miao et al. (2020) achieved higher performance in the very short term (first hour for St John's and first two hours for Yarmouth), our method maintained a more stable F1-score across the entire 24-hour period, offering a longer and more operationally useful forecast horizon.

Kamangir et al. (2021) proposed FogNet which gave the forecast of fog over the coast of Texas. They also used a post-processing approach with NWP and sea surface temperature satellite observations to give fog forecast at the Mustang Beach Airport in Port Aransas and chose three different lead times (6 hours, 12 hours and 24 hours). For the evaluation of their approach, they used different metrics such as the Probability of Detection (POD) which measures the likelihood that the ML model correctly classifies the intended target and False Alarm Ratio (FAR) which is the fraction of forecasted non-occurrence of the event over the occurrence of the event. Both equations are given below:

$$POD = \frac{TP}{TP + FN} \tag{4.5}$$

$$FAR = \frac{FP}{TP + FP} \tag{4.6}$$

The value for both metrics is between 0 and 1. For POD, a value closer to 1 is better and for the FAR, a value closer to 0 is better. They categorized the visibility in three parts and we will only focus of the below 1.6 km visibility category, where there are fog events. For the 6-hour forecasting window, they obtained a POD of 0.61 and a FAR of 0.40. For the 12-hour window, they had a POD of 0.60 and FAR of 0.56 and for the 24-hour window they had a POD of 0.54 and FAR of 0.50. For St John's, we had a POD of 0.71 and a FAR of 0.32 for the 24-hourly forecast and for Yarmouth, we had a POD of 0.64 and a FAR of 0.46. We were able to have a higher time resolution (1-hour compared to 6-hour) for the 24 hour prediction and for both locations we obtained a higher POD. We obtained a higher FAR as well, except when comparing the 6-hour window with Yarmouth which was 0.06 higher.

Castillo-Botn et al. (2022) conducted a comprehensive comparison of supervised machine learning approaches for fog prediction at the Mondoñedo weather station in Galicia, Spain. Their work focused on a very short-term nowcasting task, with a lead time of 30 minutes, and evaluated both classification and regression methods. For classification, they assessed model performance using accuracy and the weighted F1-score, with Gradient Boosting yielding the best results-an accuracy of 0.826 and a weighted F1-score of 0.828. In comparison, our post-processing approach using WRF output and the ETclassifier achieved higher performance at both sites studied: an accuracy of 0.900 and weighted F1-score of 0.901 at St John's, and 0.858 for both metrics at Yarmouth. Beyond surpassing these short-term results in terms of classification performance, our method also extends the forecast horizon significantly by

providing 24-hour predictions at hourly resolution. Pelez-Rodrguez et al. (2023) did further research in the same area and improved the fog results by 17.3% using post-processing methods with the ERA5 dataset. Their regression approach will be compared in the discussion section.

More closely related to our research, Chen et al. (2024) proposed a sea-fog forecast area over the Yellow Sea using a deep learning approach. They were able to give hourly forecasts up to 7 hours, showing the spatial changes with promising results. They used precision, recall and Critical Success Index (CSI) as metrics to access the performance. CSI is the amount of correct predictions over total number of the predicted events and actual events. It is very similar to F1-score and is given by:

$$CSI = \frac{TP}{TP + FP + FN} \tag{4.7}$$

Chen et al. (2024) performed extensive testing on their dataset, and their best-performing model achieved a Critical Success Index (CSI) of 0.475, a recall of 0.660, and a precision of 0.628 for a 7-hour fog forecast. These results demonstrate strong short-term predictive capabilities. For our approach, for St John's, we had a CSI of 0.529, a precision of 0.69 and a recall of 0.68. For Yarmouth, we had a CSI of 0.349, a precision of 0.54 and a recall of 0.64. From this comparison, we observe that the results are very dependent on the location, as our approach was better on all metrics for St John's but lower on all metrics for Yarmouth. Chen et al. (2024) acknowledged that extending the forecast lead time would degrade performance too much. In contrast, our method maintained relatively stable scores over a 24-hour forecast horizon, offering better long-range utility. That said, their approach was more effective at capturing the spatial extent of fog, a limitation of our station-based method. Since we tried regression techniques in the discussion section, a relative comparison is also done for that approach.

4.2 Discussion

We were able to see an improvement by using the ML models to predict fog instead of using traditional NWP models such as WRF in the section above. To make sure we got the best possible trained ML model, we had to ensure that we prepared the data correctly, optimized the model carefully, and tested it on different types of data. Besides, we implemented specific strategies to refine our approach, ensuring that our model was well-suited for the task and capable of delivering the most accurate fog predictions.

4.2.1 Input data evaluation

The first important thing we did was to decide to proceed with ERA5 reanalysis data for our training features compared to using WRF initialized with GFS FNL data. There are a few reasons, including it was computationally more efficient and took less time to gather, ERA5 is considered to be the standard input data for many meteorological tasks and we were able to use all of Navigation Canada's weather stations data for visibility.

	St. John's			**Yarmouth**		
Input Data	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
WRF+GFS FNL	0.68	0.72	0.70	0.60	0.64	0.62
WPS+ERA5	0.70	0.77	0.73	0.66	0.74	0.70

Table 4.10: Comparison between GFS Final Analysis and ERA5 input data to use as features for ETClassifier in St John's and Yarmouth, Canada

We used data from 2016 to 2023 to train with the GFS FNL and 2012 to 2023 for the ERA5 dataset. A comparison between using the two input data to predict fog in the 2023 test data is given in Table 4.10. The results shown were from ETClassifer since it was the better-performing ML model. As expected, for both St John's and Yarmouth, the performance when using features from the ERA5 data was better on all metrics. For St

John's, the precision was 2% better with the ERA5 (0.70 vs. 0.68), meaning that it was better at predicting actual fog, less false positives. The recall was 5% better than when initialized with GFS FNL, therefore, our ML model was able to distinguish fog from clear labels better and the overall F1-score was higher at 0.73, compared to 0.70. The difference was more significant in Yarmouth, where the precision was 6% better (0.66 vs. 0.60) and the recall was 10% better (0.74 vs. 0.64) which resulted in a better F1-score of 0.70. This overall improvement in results between the input data can be mainly attributed to having more data collected with ERA5, approximately 4 more years (2012 to 2023 compared to 2016 to 2023).

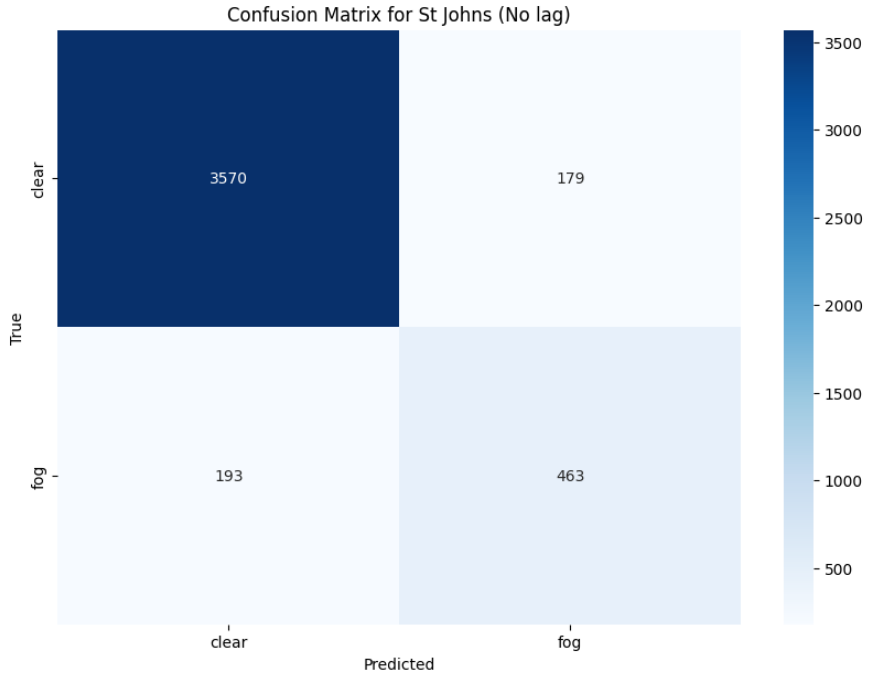
4.2.2 Lagged features evaluation

Furthermore, when we prepared our data, we introduced lagged variables for all features and we saw that it helped to have the features one hour before to predict fog from the feature importance figures. So, below we have a comparison of the ETClassifier trained with lagged features as well as ETClassifier trained with lagged features. Again, we are only showing results for the 2023 test data and ETClassifier since we determined it was our better ML model.

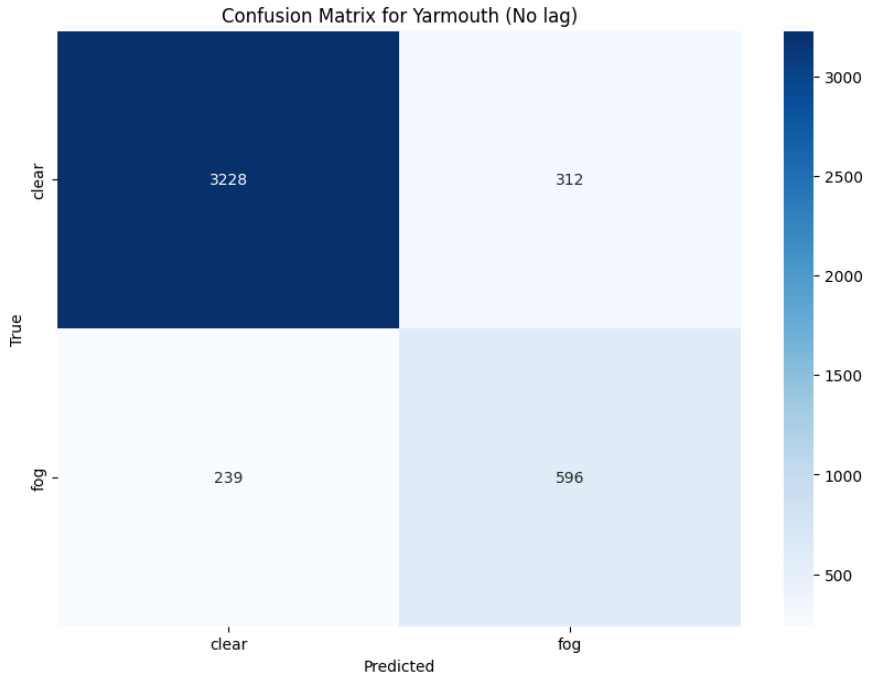
	St. John's			**Yarmouth**		
Features	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
No lag	0.72	0.71	0.71	0.66	0.71	0.68
Lagged features	0.70	0.77	0.73	0.66	0.74	0.70

Table 4.11: Comparison between GFS Final Analysis and ERA5 input data to use as features for ETClassifier in St John's and Yarmouth, Canada

As expected, the ETClassifier trained with lagged features (lagged ETClassifier) did better than with only the features (no lag ETClassifier). When optimizing the model using RandomSearchCV, we observed that the hyperparameters were different as well, with the ML



(a) Confusion matrix for St John's, Canada with no lagged features.



(b) Confusion matrix for Yarmouth, Canada with no lagged features.

Figure 4.7: Confusion matrix for St John's and Yarmouth, Canada on the 2023 test dataset using only the features selected (no lagged features).

model trained with lagged features needing to be more complex. For example, the optimized Yarmouth ETClassifier had 375 estimators with lagged features included while the no lag model needed only 120 estimators. Lagged ETClassifier did better on most of the metrics used and had a 2% increase in F1-score for both locations, from 71% to 73% in St John's and from 68% to 70% for Yarmouth.

For St. John's, the precision of the lagged ETClassifier decreased slightly (0.70 vs. 0.72), indicating fewer false positives and a better ability to correctly predict fog occurrences for the no-lag ETClassifier. More notably, the recall increased by 6%, meaning the model with lagged features was better at distinguishing between fog and clear conditions, reducing false negatives. This is evident in the confusion matrix, where the lagged ETClassifier correctly identified more fog cases (503 vs. 463). Although it also misclassified fog more frequently (220 vs. 179), it ultimately had fewer false negatives (153 vs. 193) due to the higher recall. This led to capturing more fog events with lagged features (503 vs. 463).

For Yarmouth, while the precision remained the same (0.66), the recall improved with the lagged ETClassifier (0.74 vs. 0.71). This led to more correctly identified fog instances (618 vs. 596) and better differentiation between fog and clear conditions (217 vs. 239 false negatives). Although the overall improvement was not drastic, incorporating a one-hour lag to the features helped the model to distinguish more accurately between fog and clear cases, which is crucial for operational forecasting.

4.2.3 Combining Locations

We analyzed the features we had for each location thoroughly in the Data Exploration section, where we concluded that the weather patterns were different enough to train the ML models individually instead of combining them. However, we have not tested the difference in performance in training together or separately. One can argue that by having more data

to train, the ML model may learn more about the patterns to classify fog or will generalize better.

Therefore, in this section, we combined both locations together and trained a single ET-Classifier ML model. We had to prepare our data differently. We loaded both datasets for St John's and Yarmouth in the Python environment and before we combined them into one dataset, we added a new column as a location identifier for each place. After concatenating the datasets, we used OneHotEncoder (Scikit-Learn (2025b)) to transform the names of the locations into a binary format without any ordinality. Then, we prepared the data the same as before, splitting the data while maintaining the time series, normalizing the features, using lagged features, and BorderlineSMOTE to counter imbalanced data. Finally, we used RandomSearchCV to get the optimal parameters which are given in the table below (Table 4.12):

Parameter	**Combined St John's & Yarmouth**
bootstrap	False
class_weight	balanced
criterion	entropy
max_depth	40
max_features	log2
min_samples_leaf	3
min_samples_split	5
n_estimators	475

Table 4.12: Best Parameters for the ETClassifier model for the combined locations using RandomSearchCV.

The optimal hyperparameters for the combined location ETClassifier are given in Table 4.12. Since we have more data, it makes sense that the ensemble model is the largest, needing 475 estimators. No bootstrap was needed, meaning that each tree used the whole training set, and the class weight was balanced to account for fog imbalanced. The other hyperparameters are standard, with maximum depth being 40, and maximum features considered for a split

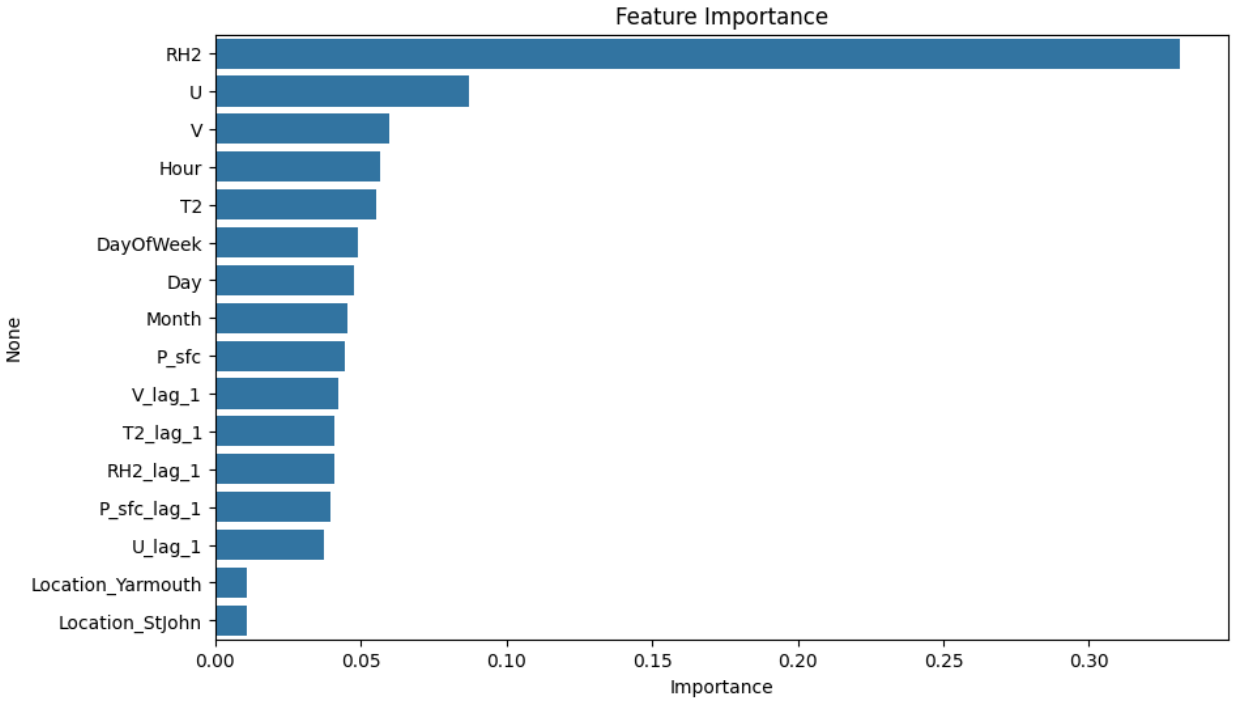


Figure 4.8: Feature importance after combining both locations using ETClassifier ranking the importance of each feature for predicting fog.

being log 2. The minimum samples for a leaf node was 3 and for a split was 5.

After training the model, we tested it with the 2023 dataset for both St John's and Yarmouth. In Fig. 4.8, we see the ranking of all the features we used to train the ET-Classifier. Surprisingly, the lagged features were not as important as when we individually trained the model for each location and we can see that the model is not learning much from identifying the locations during the training. Despite having more data, the feature importance decreased compared to training individually, for example, relative humidity at 2 m is at 0.3 while when training one location, it is around 0.5. The classification report for the whole test dataset (St John's + Yarmouth) is given in Table 4.13.

From Table 4.13, we see that the overall accuracy of the model is 88% which shows that it can do a good job at classifying fog and clear labels. For St John's, it has a lower accuracy (ETClassifier with St John's only is 92%) but for Yarmouth, it has the same accuracy. The

Class	Precision	Recall	F1-Score	Support
clear	0.93	0.93	0.93	7283
fog	0.66	0.65	0.65	1496
Accuracy		0.88		8779

Table 4.13: Classification Report for St John's and Yarmouth, Canada combined on the test data using ETClassifier.

precision, recall, and F1-score metrics for fog are lower than when we trained individually but to get a good sense of how well the model did for each location, we had to extract the results for them from the combined test dataset. By separating the test data with the location columns, we can easily see how the model did for each location. The classification reports for each location are given below:

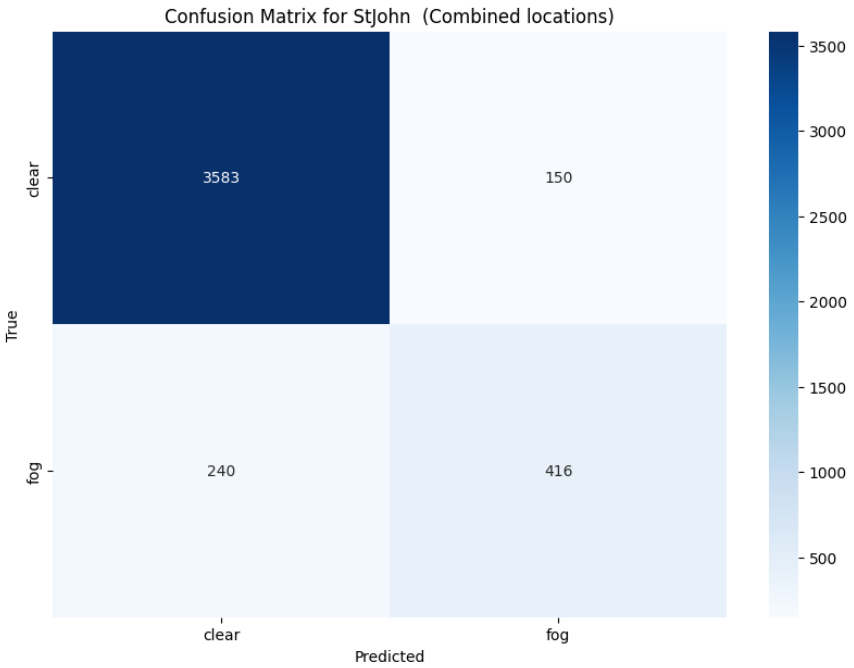
Class	Precision	Recall	F1-Score	Support
clear	0.94	0.96	0.95	3733
fog	0.73	0.63	0.68	656
Accuracy		0.91		4389

Table 4.14: Classification Report for St John's extracted from the combined test dataset using ETClassifier.

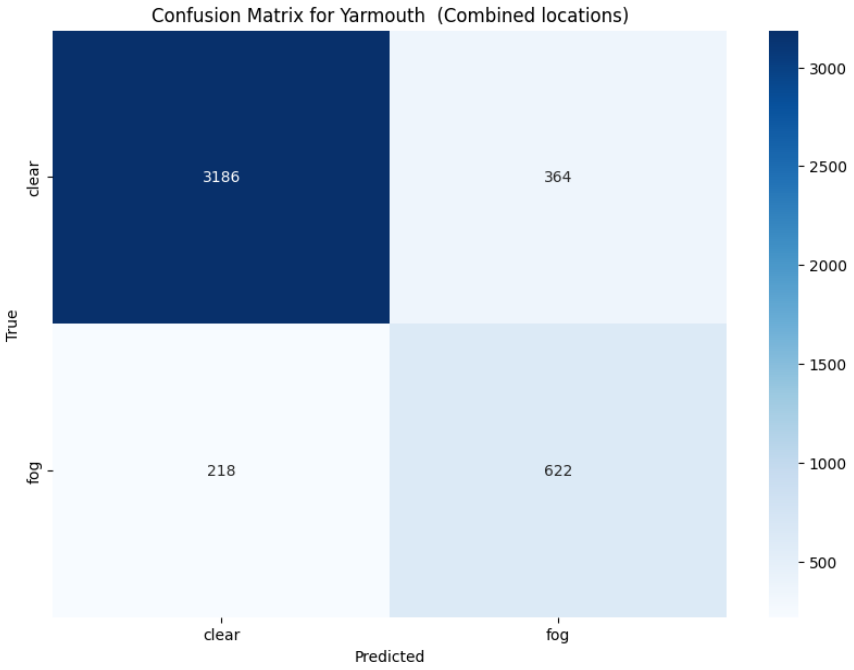
Class	Precision	Recall	F1-Score	Support
clear	0.94	0.92	0.92	3550
fog	0.63	0.74	0.68	840
Accuracy		0.88		4390

Table 4.15: Classification Report for Yarmouth, Canada extracted from the combined test dataset using ETClassifier.

By looking at the performance of the model based on locations, we have a better idea of how it is performing. For St John's, we can do a direct comparison with Table 4.1 and Fig. 4.2(a). We had a higher precision using the combined locations ETClassifier with 0.73 compared to 0.70, therefore fewer false positives. However, there is a big difference in recall,



(a) Confusion matrix for St John's, Canada extracted from the combined test set.



(b) Confusion matrix for Yarmouth, Canada extracted from the combined test set.

Figure 4.9: Confusion matrix for St John's and Yarmouth, Canada on the combined test dataset using only one model, ETClassifier.

14% (0.63 vs. 0.74), which brings down to F1-score to 0.68 compared to 0.73. From the confusion matrix, we do observe less actual fog captured by the model (416 vs. 503) but since the precision was higher, the model overestimated fewer fog events (150 vs. 220). With a much lower recall, it mislabeled and missed 240 fog labels compared to 153 with St John's only ETClassifier.

For Yarmouth, the combined ETClassifier did a better job and had a performance much closer to the Yarmouth-only ETClassifier. It was able to capture 622 fog events compared to 618 with Yarmouth-only ETClassifier and mislabeled clear events as fog at the same rate (218 vs. 217). This is because both models had the same recall of 74%. The difference in performance comes in the precision, where it was 63% effective compared to the Yarmouth-only model with 66%. This means that it overestimated fog labels more than the Yarmouth model, and classified 39 fog events wrong (364 vs. 325). The overall performance of this model compared to the Yarmouth-only ETClassifier is given by the F1-score, where it got 68% compared to 70%. Even if it captured slightly more fog labels, the performance of the Yarmouth-only model was better and generalized better.

Therefore, now we are able to confidently say that choosing a model for each location was the correct approach to get the best results as the model was able to learn the weather patterns of that location better, thus learning and generalizing better.

4.2.4 Multi-Classification

Another major decision was to either do binary classification (either fog or no fog) or do multi-classification for the dataset we have since we collected visibility from the weather reports. To do multi-classification, we had 3 labels: clear (> 5 km), mist ($5 \leq$ m $<$ 1.2 km), and fog (≤ 1.2 km). A key advantage of multi-class classification is that if the model predicts mist instead of fog, it still serves as an early warning, bringing us closer to

identifying fog conditions. Additionally, having mist as a separate category provides more nuanced predictions and better reflects real-world conditions.

The first step was to classify visibility in our data as such, then we cleaned and prepared the data using the same approach as before with the exception of how we converted the 'class_vis' column containing our categorical features (clear, mist, fog). We used LabelEncoder (Scikit-Learn (2024j)) which encodes the target values with values between 0 and n_classes-1. Therefore, it mapped the 'class_vis' column from categorical label to numerical classes, that is, "clear" got mapped to 0, fog to 1, and mist to 2.

We prepared the data exactly as before and used RandomSearchCV to get the best hyperparameters for this task. We only prepared data at St John's and trained the ETClassifier because the results for the two locations align and we can just look at St John's results to evaluate the performance of our model. The table is given below:

Parameter	**Combined St John's & Yarmouth**
bootstrap	True
class_weight	balanced_subsample
criterion	gini
max_depth	None
max_features	sqrt
min_samples_leaf	2
min_samples_split	14
n_estimators	300

Table 4.16: Best Parameters for the ETClassifier model for multi-classification using RandomSearchCV.

The model used the bootstrap method to train, meaning it used a smaller part of the dataset to train each tree and the ensemble had 300 trees. The class weight was balanced for each sample the tree had (balanced_subsample) and they were allowed to grow deep without restrictions. The criterion was Gini and the maximum number of features chosen in a node

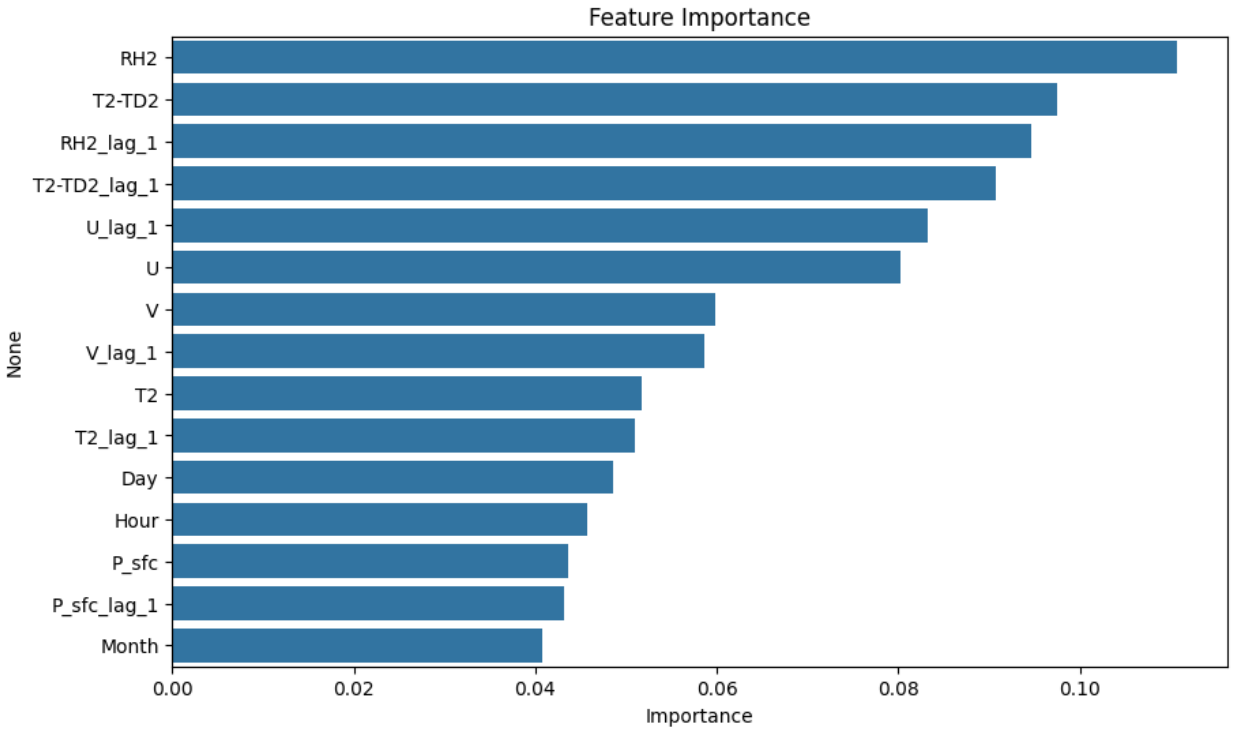


Figure 4.10: Feature importance for ETClassifier ranking the importance of each feature for multi-classification of fog and mist.

was the square root of the latter. Therefore, for a split, the model needed a minimum of 14 samples and for a leaf node, it was 2.

The feature importance ranking is given in Fig. 4.10, where we see a similar trend as when we did binary classification as expected. However, the importance of the features was around 0.10 which suggests that they did not have good predictive ability. This is directly illustrated in the classification report in Table 4.17 and the confusion matrix in Fig. 4.11.

The classification report shows good performance in identifying clear and fog, with an F1-score of 0.92 and 0.70, respectively. However, the model cannot identify the extra label mist at all. The overall accuracy dropped from 0.92 to 0.87 and the metrics for identifying fog, our main task, dropped as well. The precision in identifying fog is better by 1% but

Class	Precision	Recall	F1-Score	Support
clear	0.94	0.92	0.92	3465
fog	0.71	0.70	0.70	656
mist	0.00	0.00	0.00	284
Accuracy		0.87		4405

Table 4.17: Classification Report for St John's, Canada using ETClassifier for multi-classification of fog and mist.

the recall is worse by 7% which leads to an overall decrease in performance of 3% with the F1-score. This is shown in the confusion matrix where the actual fog was predicted less correctly (462 vs. 503) and the amount of false positives is slightly better (189 vs. 203) where in this case, the false positives included both mislabeled 'clear' and 'mist'.

Moreover, the model did worse at differentiating between fog and clear (194 vs. 153) but the main issue is identifying mist. Even with a lot of optimizing and changing the class weight of mist to be 10× that of clear label so that the model would place more importance on correctly identifying mist, it could not predict mist at all. Instead, it predicted clear 192 times and fog 92 times. We even tried various undersampling methods, where we removed clear cases so that the data was less imbalanced and the results were the same. There are two main reasons why this could happen.

The mist class is severely underrepresented in our data, for example, since it is comprised of only 284 counts in the test dataset, it is around 6% of the data we have. The second reason is that mist has similar characteristics to fog (class 1) or clear (class 0), so the model might not be able to distinguish it properly. This leads to the model predicting clear 192 times and fog 92 times instead of mist as the features are too similar. We would need more features that are more indicative of mist to be able to differentiate it. Therefore, choosing binary classification not only helped us identify fog better as seen above but also reduced the error of misidentifying it for another label.

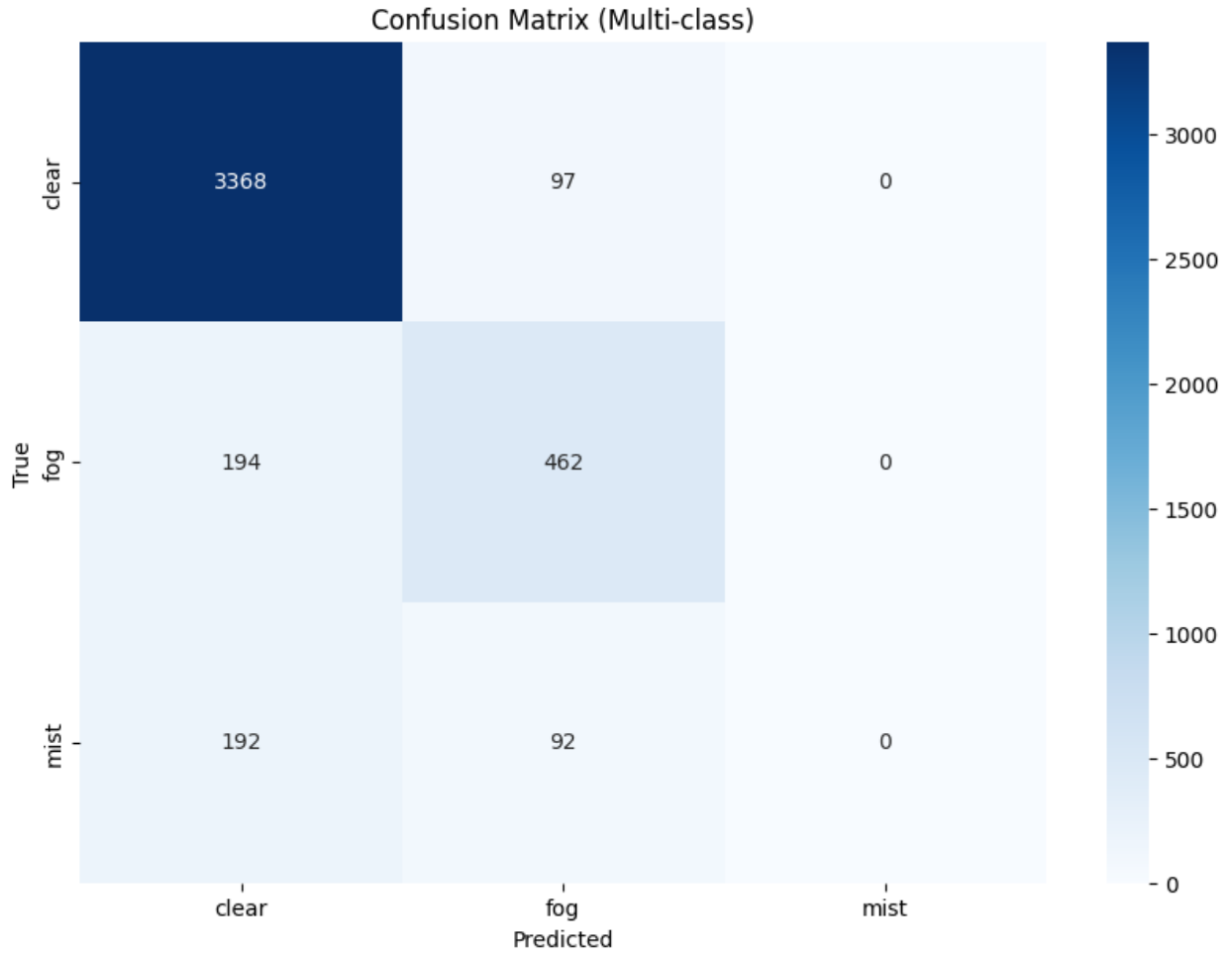


Figure 4.11: Feature importance for ETClassifier ranking the importance of each feature for multi-classification of fog and mist.

4.2.5 Regression Technique

Regression is another supervised ML technique that models the relationship between a dependent variable (target) and one or more independent variables (features), similar to classification, but the model predicts continuous values instead of discrete labels (Sagar (2025)). This approach does make sense since we have visibility reports that range from 0 km to around 24 km.

ExtraTreesRegressor

There are various types of regression, with the simplest being linear regression (Altman and Krzywinski (2015)), which assumes a linear relationship between the input variables and the target. In this study, we started with the ExtraTrees Regressor (ETRegressor) from Scikit-Learn (Scikit-Learn (2025a)), an ensemble-based model that captures nonlinear relationships between features. While ETRegressor operates similarly to ETClassifier, it requires different parameters and hyperparameter tuning for optimal performance. Additionally, the model's evaluation differs, as regression tasks use distinct performance metrics. For this analysis, we focus on the St. John's dataset, as the performance is expected to be similar across locations, and our goal is to gain a general understanding of this approach's effectiveness.

We loaded the data for St John's and cleaned it for null visibility values. Then, we prepared the data similarly to ETClassifier, we used the time-based features and lagged features but we also dropped the 'class_vis' column which was used for binary classification. Since our features are on different scales, we standardized them as before. In addition, the range of visibility is quite broad (0-24 km), where we are interested in less than 1.2 km which is fog. So, to make sure that the model can learn the small values well and reduce the impact of the larger values, we converted the 'Vis' column to a logarithmic scale, as given in Equation 4.8:

$$log1p('Vis') = log(1 + \text{ 'Vis'}) \tag{4.8}$$

This ensured our data were all within the same range, and the $log(1+x)$ helps in handling 0 values of visibility. For instance, a visibility of 1 km is transformed to approximately 0.693, while 20 km becomes 3.04. By compressing larger values and reducing the disparity between them, this transformation helps the model better capture fog-related patterns and improves

its ability to learn visibility characteristics. We proceeded to separate the features (RH2, T2, U10, V10, P_sfc, T2-TD2, their corresponding lagged features, and time-based features) and the target which was visibility on the log scale.

Similar to ETClassifier, to find the best hyperparameters we used RandomSearchCV, but instead of using F1-score as scoring, we used R^2 which is the coefficient of determination to evaluate the hyperparameters and get the best model. It is the default metric used in RandomSearchCV for regression. R^2 is a statistical measure that indicates how well the regression models predictions match the actual data. The formula is given below:

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y})^2}{\sum_i (y_i - \overline{y})^2} \tag{4.9}$$

, where y_i is the actual value, $\hat{y}$ is the predicted value and $\overline{y}$ is the mean of actual values. As seen from the equation, R^2 measures the variability, that is, it tells you how much of the variance in the target variable is explained by the model. The numerator contains the sum of square residuals (SS_{res}), which measures how much the predictions deviate from the actual values. The denominator is the total sum of squares (SS_{tot}), which measures how much the actual values deviate from their mean. When R^2 is equal to 1, the model perfectly predicts the data and when it is 0, the model performs no better than simply predicting the mean of the target values. If the R^2 value is negative, this means that the model performs worse than simply predicting the mean, which can happen if the model fits the data poorly. So, we used RandomSearchCV with the time series split cross-validation 5-fold for different hyperparameters, and the optimal tuning is given in Table 4.18.

A lot of the hyperparameters are the same as binary classification; the model did not need bootstrap to train, and the maximum depth of the trees was up to 50 splits. The maximum features were set to be log2(number of features) which are randomly selected for

Parameter	ETRegressor
bootstrap	False
criterion	squared_error
max_depth	50
max_features	log2
min_samples_leaf	3
min_samples_split	2
n_estimators	450

Table 4.18: Best Parameters for the ETRegressor model to predict visibility using RandomSearchCV.

ETRegressor and for splitting a node, the minimum samples were 2. To establish a leaf node, the minimum amount of samples was 3, and the ensemble of trees to train the model was 450. We have two different hyperparameters, the criterion used for regression and we do not use class_weight which was for classification.

To deal with imbalanced data, we used a different approach, where we changed the sample weight. Sample weighting is a common technique used to handle imbalanced datasets by assigning different importance levels to training instances (Kuhn and Johnson (2013)). We implemented sample weighting with visibility. Since we changed the visibility to log(1+Vis), we established a threshold for fog at log(2.2). This threshold is then used to classify target values (y_train), where cases with lower visibility (i.e., y_train $<$ log_fog_threshold) receive a weight of 5, while other cases receive a weight of 1. By assigning higher importance to low-visibility conditions, the model prioritizes minimizing errors in these critical scenarios, which is particularly useful in imbalanced datasets where fog events occur infrequently.

The other difference from the classification approach is the criterion used for splitting in the tree. The function used for this task was squared_error as seen in Table 4.18 which derives from mean squared error (MSE). MSE is the average squared difference between the predicted and actual values and is usually used as an evaluation metric, given by:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.10}$$

y_i is the actual value and $\hat{y}_i$ is the predicted value over n amount of predictions. It averages the error across all predictions. However, the squared_error criterion differs slightly to adapt for splitting in the trees. It determines the best split by minimizing variance within nodes. It is defined as the variance reduction after the split, which is equivalent to minimizing the sum of squared errors (SSE) within each node (Yadolah (2008)). At each split, the algorithm selects the feature and threshold that minimize the following criterion:

$$SSE = \sum_{i \in L}(y_i - \bar{y}_L)^2 + \sum_{i \in R}(y_i - \bar{y}_R)^2 \tag{4.11}$$

Here, L represents the left node, and R represents the right node. It measures the total variance, and the criterion will attempt to minimize SSE to get as close as possible to the actual value. After training with the optimized model that also accounts for fog, we evaluated it with the 2023 test dataset. We used 3 different metrics to evaluate the performance of our model, the R^2 score as described before, the MSE, and the mean absolute error (MAE). Wikipedia contributors (2025c) defines MAE as the measure of errors between predicted and actual values and is calculated as the sum of absolute errors over the sample size n:

$$MAE = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n} \tag{4.12}$$

The R^2 measures the variability between the target and predicted values, which helps us to understand how well your model explains the relationship between input features and the target. The MSE computes the average squared difference between actual and predicted values and penalizes larger errors more heavily due to squaring. Since it emphasizes large

errors, it is useful when big mistakes are more costly. Finally, MAE gives a balanced measure of overall model performance without over-penalizing large errors since it has the same units as the target. The table below gives the performance evaluation of the ETRegressor:

Metrics	**ETRegressor**
R^2	0.3297
MAE/ km	5.650
MSE/ km^2	63.43

Table 4.19: Performance evaluation of the ETRegressor for St John's, Canada for the 2023 dataset

The model's performance on the test set indicates that while it captures some patterns, there is significant room for improvement. The Mean Absolute Error (MAE) of 5.65 suggests that, on average, predictions deviate by about 5.65 km from the actual values. However, the Mean Squared Error (MSE) of 63.43 km^2, which is much larger than the MAE, indicates that some predictions have significantly large errors, contributing heavily to the overall loss. Additionally, the R score of 0.33 shows that the model explains only 33% of the variance in the target variable, meaning a substantial portion of the variability remains unaccounted for.

So, to get more insight, we plotted the predicted visibility for 72 hours where there were fog events and compared it with the visibility reports to see how well the model could capture the events in Fig. 4.12. The fog threshold at 1.2 km was marked by the red dashed line. The actual visibility (blue line with dots) shows sharp fluctuations, frequently dropping to near zero, indicating foggy conditions. The predicted visibility (orange line with crosses) follows the general trend of the actual data but often fails to capture sharp drops and underestimates peak values. The model does reasonably well in identifying periods of lower visibility but appears to over-smooth rapid changes, leading to errors in predicting the exact timing and

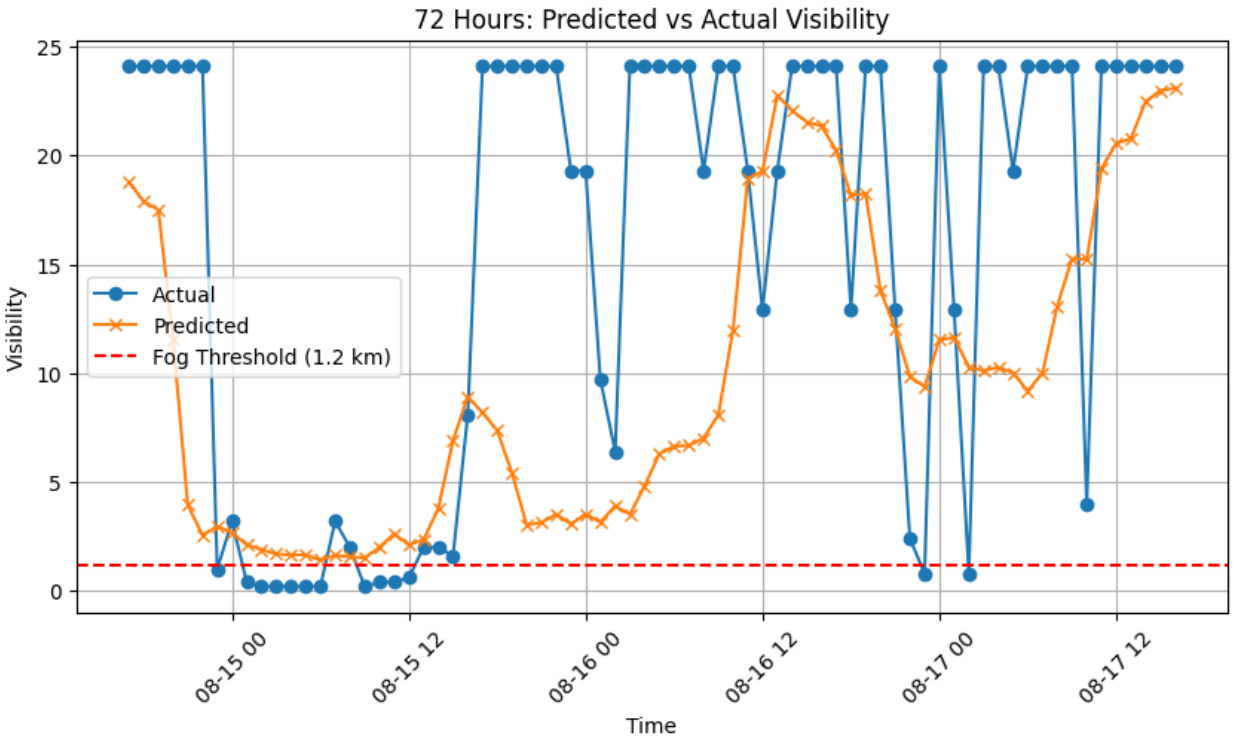


Figure 4.12: Line plot of Visibility in km vs Time in hours for 72 hours from August 14th at 18 UTC to 17th at 18 UTC for the ETRegressor compared to visibility reports.

severity of fog events. In doing so, the model actually misses all fog events in that 72-hour period. This aligns with the previously observed high MSE, suggesting that large prediction errors occur in those cases.

Quantile Regression

The performance of the model was not bad, but in our case, it was not useful at all to be able to predict visibility trends because it missed all fog events. So, we decided to try a more robust approach, the quantile regression. This approach was also used in hurricane tracking by numerous AI-based models, first proposed by Pathak et al. (2022).

From Tofalli (2015), quantile regression is a type of regression that estimates the conditional quantiles of a response variable rather than predicting just the mean. Instead of

minimizing the squared errors, it minimizes the absolute errors weighted by the quantile level. ETRegressor used the mean value of all the estimators to produce the trend but quantile regression shows the model distribution as well. Therefore, the model can capture the uncertainty of the prediction made, that is, the estimation of different quantiles (e.g., 10th, 50th, 90th percentiles) to model the upper and lower bounds of predictions.

Another advantageous aspect of quantile regression is that it can also handle heteroscedasticity (Scikit-Learn (2025c)). It refers to a situation where the variance of errors (residuals) is not constant across different values of the predictor variables. In our case, when predicting visibility, heteroscedasticity can appear as the fog has higher uncertainty; some fog clears quickly, and some persist. However, high-visibility periods may have lower uncertainty, meaning predictions are more stable. We did observe this condition when looking at the 72-hour prediction in Fig. 4.12, with the sharp drop and jump in visibility and how sometimes the low visibility persists. By using quantile regression, we can estimate the range of possible visibility values rather than a single average prediction, providing better confidence intervals for fog forecasting and also making it more robust to outliers.

For this task, we used quantile regression with Random Forest as it is not supported by the ExtraTrees model. The approach is very similar to ETRegressor with the exception that we defined the 5th, 50th, and 95th percentile. The 5th and 95th percentile will represent the 90% confidence interval of our model while the median (50th percentile) represents the best prediction of our model. We used RandomSearchCV to get the best hyperparameters given in Table 4.20.

The criterion is the same as before and bootstrap is always used in Random Forest. The number of estimators was 200 and they were allowed to grow to 10 splits with the minimum samples split to be 10 and for a leaf node 4. The model used all the features in a node to make a decision. The median is used as the best prediction for the model and will be used

Parameter	Quantile Regression
criterion	squared_error
max_depth	10
max_features	None
min_samples_leaf	4
min_samples_split	10
n_estimators	200

Table 4.20: Best Parameters for the Quantile Random Forest Regressor model to predict visibility using RandomSearchCV.

to evaluate the performance of the model using the same metrics as before. The evaluation is given in Table 4.21 below:

Metrics	Quantile RFRegressor
R^2	0.5094
MAE/ km	4.710
MSE/ km^2	46.43

Table 4.21: Performance evaluation of the Quantile RandomForest Regressor for St John's, Canada for the 2023 dataset

We can see an overall improvement in Table 4.21 from the ETRegressor on all the metrics. The MAE is lower than with the ETRegressor, suggesting that the RF Regressor is predicting closer. Furthermore, MSE is smaller, meaning that there are fewer large deviations between predicted and actual values, and R^2 is higher, showing better explanatory power and fit to the data by around 20%. However, we need to see how much change this represents, so we plotted the visibility line plot over the same 72-hour interval in Fig. 4.13 to see how well it predicts the foggy events.

The model demonstrates a reasonable ability to predict visibility trends over the 72-hour period, with the predicted median closely following the actual visibility in many instances. The inclusion of a 90% prediction interval provides insight into uncertainty, with wider

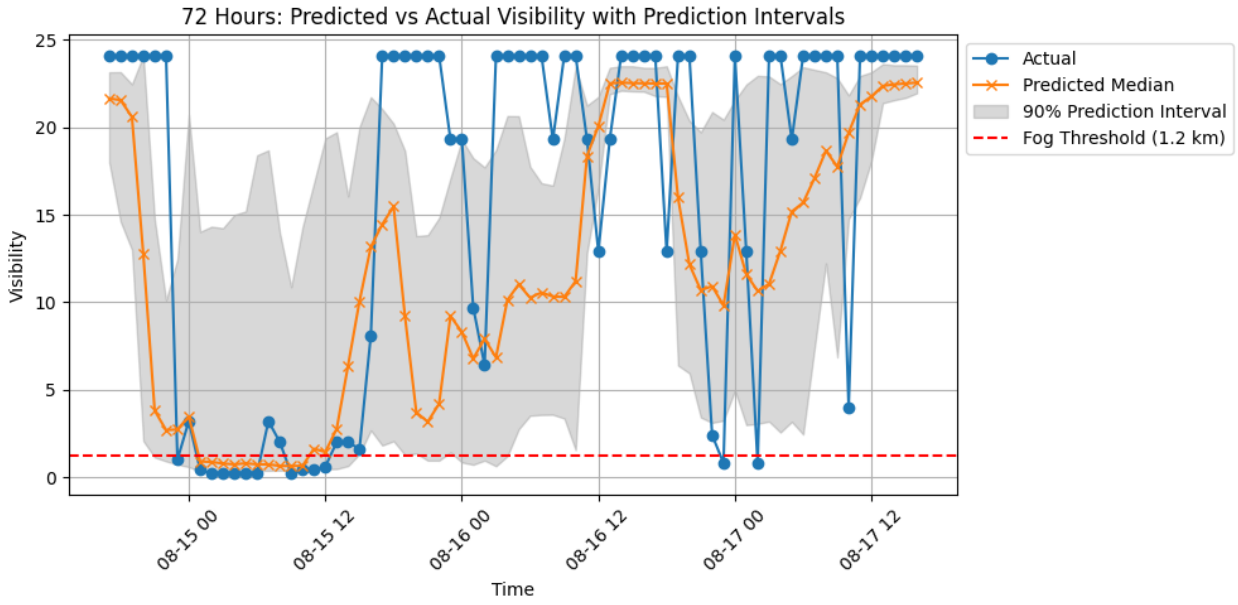


Figure 4.13: Line plot of Visibility in km vs Time in hours for 72 hours from August 14th at 18 UTC to 17th at 18 UTC for the Quantile RF Regressor with 90% confidence interval compared to visibility reports.

intervals observed during rapid visibility transitions, indicating increased model uncertainty in these cases. This increase in uncertainty can be due to many reasons, and in our case, it is more probable to be the sudden changes in visibility.

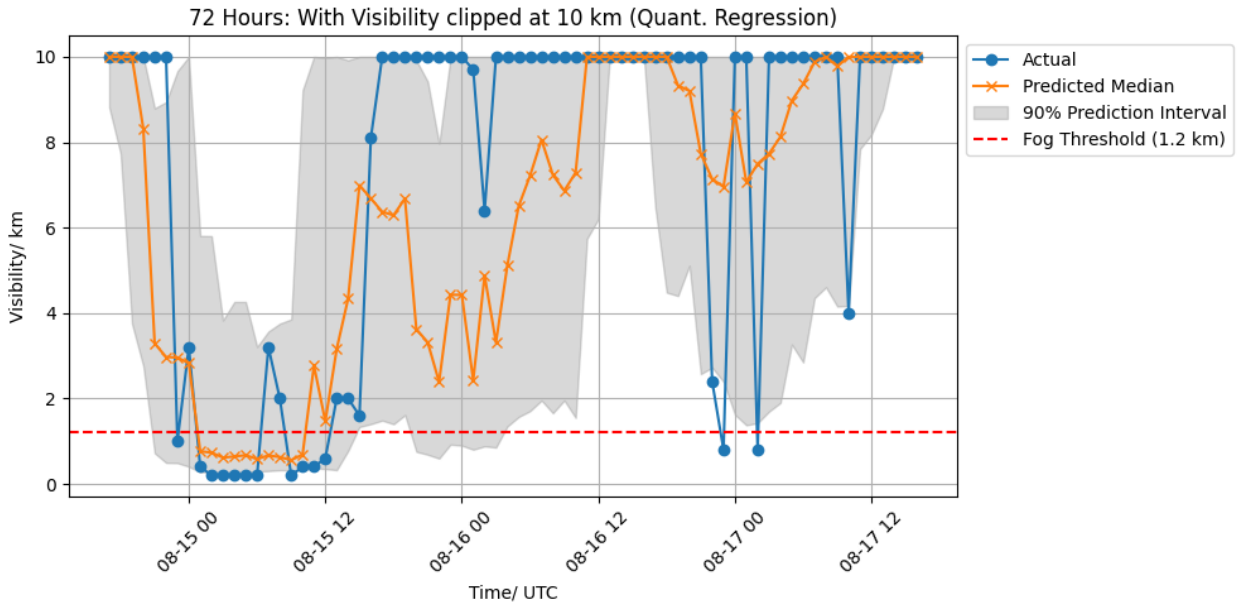
The model effectively captures low-visibility events below the 1.2 km fog threshold, though some instances show a slight underestimation of extreme fog conditions (from 08-15: 00 to 08-15:12). Additionally, while the model generally follows fluctuations in visibility, it occasionally struggles with sharp transitions, leading to minor prediction delays. Overall, the model performs well in identifying fog events but its responsiveness to sudden changes in visibility can lead to fog events being completely missed, even with the confidence interval (from 08-16:20 to 08-17:00). To have an idea of how well the regression approach was performing, we had a relative comparison with a few papers. Close to our research, Gultepe et al. (2024b) used generative nowcasting of fog near Sable Island and used regression techniques to nowcast marine fog up to 1 hour. Their best regression model, the Least-squares

gradient boosting (LSB), had a RMSE of 2.92 km. Our regression approach had a RMSE of 6.81 km which is almost twice as much. However, we were doing 24 hour predictions and used the whole range of visibility (around 24 km), whereas their approach was only for 1 hour prediction and the model was predicting for visibility below 10 km. Having a smaller RMSE means better prediction, but we believe that if we clipped our visibility data to 10 km, we would have gotten comparative results.

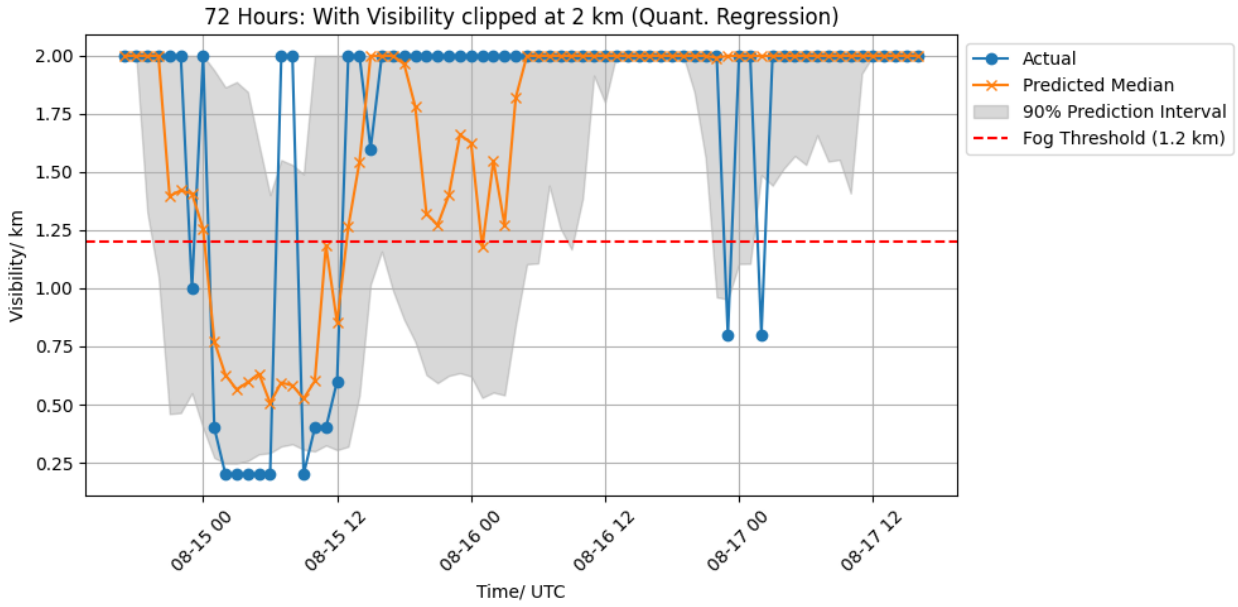
Metrics	**Quantile RFRegressor (10 km)**	**Quantile RFRegressor (2 km)**
R^2	0.6082	0.6161
MAE/ km	1.166	0.1486
MSE/ km^2	5.456	0.1153

Table 4.22: Performance evaluation of the Quantile RandomForest Regressor for St John's with visibility clipped at 10 km and at 2 km.

In Fig. 4.14, we have the same 72 hours prediction we used at St John's but we have clipped the visibility at 10 km. In Table 4.22, we see the statistical performance of the ML model when we clipped the visibility at 10 km. As expected, we saw an improvement in the R^2, MAE and MSE. With the same visibility range, we had a RMSE of 2.34 km. This new RMSE value is slightly better to what Gultepe et al. (2024a) obtained with their best models, suggesting that the range of visibility plays a big role in the performance. Therefore, we decided to look deeper at other research regarding the regression approach and the statistical metrics used. Castillo-Botn et al. (2022) used a regression approach as well and the sensors at the weather stations had a range of 2 km. Therefore, we clipped the visibility data at 2 km to match the research paper. In Fig. 4.14b, we see the results for this method and the metrics are given in Table 4.22. We see the same trend in the metrics as when we clipped to 10 km, that is, a higher performing model with better R^2, MAE and MSE. For their 30 min regression approach, they found that the largest multi-layered perceptron that they used gave the best results (gradient boosting for classification as mentioned before). They had an R^2 value of 0.8417, MAE of 164.30 and a RMSE of 327.56. We only used the regression approach



(a) Visibililty prediction for regression approach with Vis clipped at 10 km



(b) Visibililty prediction for regression approach with Vis clipped at 2 km

Figure 4.14: Line plot of Visibility in km vs Time in hours for 72 hours from August 14th at 18 UTC to 17th at 18 UTC for the Quantile RF Regressor with visibility clipped to match other research.

at St John's, and from Table 4.22, we had an R^2 of 0.6161, MAE of 148.6 and a RMSE of 339.5. We had a lower R^2 value which indicates how much variability in your predictions is explained by the model (0.84 v/s 0.61) and since it does not give much indication on the predictive performance of the model, we focussed more on the MAE and RMSE score. For both metrics, a lower value is better. The RMSE value we obtained was slightly higher than what they had but our MAE was lower. Given that our approach produced 24-hour forecasts instead of short-term nowcasts, the performance is relatively strong and demonstrates the feasibility of extending fog prediction to longer lead times.

Pelez-Rodrguez et al. (2023) did further experiments on the same weather station and they used only regression techniques but were able to get 1-hour forecast, 3-hour forecast and 6-hour forecast. They used MAE as a metric and for the 1-hour forecast, they got 159.98, for the 3-hour forecast they got 316.96 and the 6-hour forecast they had 402.07. They obtained the best results by nowcasting 1-hour ahead and the range of visibility was 2 km. Our approach clipped at 2 km was 148.6 which i s lower than their 1-hour forecast which further shows that our post-processing approach had a solid performance. Another thing that we noticed was the metrics used for regression approach. We have plotted the same 72hr prediction of visbility in Fig. 4.14 for both visibility clipped at 10 km and 2 km. We showed that decreasing the visibility to match the other research gave us a better comparison but we were not sure if the results were more accurate. By looking at both clipped prediction, we noticed that the predictions did not improve but was the same as when we looking at the whole range of visibility. This was pretty interesting since we re-trained and optimized the models for the new ranges of visibility but it did not capture more fog events, even if the metrics showed a significant improvement. Therefore, we believe that the metrics are strongly influenced to the visibility range and the actual prediction did not change as shown in Fig. 14.4.

While the regression approach does give us an indication of how intense the fog is, it missed a lot fog events and the binary approach is more appropriate. The variability in visibility suggests that predicting visibility as a continuous variable may not always be reliable, especially for operational decision-making. Instead, our binary classification approach, where the model simply predicts whether foggy conditions (e.g., visibility < 1.2 km) will occur, is more appropriate. By distinguishing between fog and no-fog conditions, we reduced the impact of the high variance from the visibility. We have also seen that the performance of our regression approach is comparable to other research done in predicting fog and we concluded that our binary classification approach is more accurate than the regression approach. In addition, with binary classification, it is also possible to give a percentage of the chance of fog since it is done by a popularity vote with the ensemble methods like ETClassifier which makes it even more advantageous.

Chapter 5

Conclusion

In this research project, we explored various methods to understand marine fog. We used the WRF model to operationally predict marine fog for the FATIMA project over the Yellow Sea, which led to trying to understand how the model treats cloud and fog during simulations. The main goal was to understand the microphysics scheme in WRF, more specifically the Thompson microphysics scheme, which was used in the FATIMA campaign and the "Surface deposition of marine fog and its treatment in WRF model" by Taylor et al. (2021). We looked at the cloud concentration number, N_c in the scheme and found out that it was a constant in the Thompson single moment scheme. We change the number from 100 cm^{-3} to 75 cm^{-3} to represent cleaner air which is suitable for maritime cases, in our case, marine fog. We also used the double-moment Thompson Aerosol-Aware scheme which varied N_c with time. By tuning N_c and changing the scheme, we were able to reduce the LWC content produced by WRF which is often overpredicted. With the Thompson Aerosol-Aware scheme, the LWC decreased by 55%, resulting in an increase in visibility by 300 m.

We were able to have a good understanding of the microphysics scheme in WRF and acknowledged that WRF, along with many NWP models, struggles with accurately predicting

the LWC and this impacts the cloud processes, including fog formation. So, we decided to use a different approach, using ML techniques. We used binary classification, a supervised ML technique, to classify between fog and no-fog events. We tried various models and ET-Classifier performed the best consistently, improving the prediction of marine fog from the WRF model over St John's and Yarmouth, Canada.

We used the visibility reports from NAV Canada at those locations and labeled fog when visibility dropped below 1.2 km. After training the models adequately with the prepared data, we used the F1-score as an evaluation metric, which offered a comprehensive understanding of how well the model predicted fog events. We obtained a 11% increase in performance for both St John's and Yarmouth from the WRF model. Furthermore, we ensured that binary classification was the optimal method by trying other ML techniques such as multi-classification (fog, mist, clear) and regression. The binary classification was also the method adopted by Lam et al. (2023) to capture extreme weather events and showed that this approach was most suitable in Appendix 8.3 of the paper, where they looked at extreme temperatures. This approach enabled us to do medium-range forecasting (1-15 days) of marine fog. We were able to identify fog events hourly for the next 24 hours compared to other approaches which ranged from 1 to 7 hours, and also obtained a higher performance in terms of classifying fog.

5.1 Future Work

Generally, there is a lot of work being done to understand how cloud processes are being modeled in NWP models. The first thing we can do is try to have a deeper understanding of the microphysics scheme and why the Thompson Aerosol-Aware scheme produces much less cloud concentration number (around 20 cm^{-3} compared to 100 cm^{-3}). The simple

changes we made can be evaluated by running the real-case WRF and comparing it with observations, for example, running the Yellow Sea simulations and comparing it with the data collected from the campaign. We can also look at the cloud distribution, as N_c has a generalized gamma distribution, and many observations taken of fog suggest a bimodal distribution (Nelli et al. (2024), Wagh et al. (2021)). Having a more accurate distribution can help the NWP models in producing fog more accurately.

Additionally, while our ML models improved marine fog predictions, further enhancements are possible. We trained our models using ERA5 reanalysis data and used GFS forecasts as input to run 24-hour simulations in WRF. To potentially improve the 24-hour forecast, two approaches can be explored. First, we could use the ECMWF forecast, which is produced by their Integrated Forecast System (IFS; ECMWF (2024)), as input for WRF. Second, we could directly use the forecasted data and process it through WPS to generate the same domain and hourly resolution used in our study. However, since we primarily used surface variables (T2, RH2, U10, V10, and P_sfc), which are well-constrained by observations from weather stations, buoys, and satellites, both ERA5 and GFS tend to be highly accurate and exhibit minimal differences.

Furthermore, AI-based models can be used independently to generate 24-hour forecasts, reducing the time required for data collection. According to documentation from various AI models, their forecasts often outperform traditional NWP models in accuracy. By leveraging more accurate inputs, the ML models we trained (e.g., ETClassifier) may yield better results. ECMWF has already implemented this approach by running its own AI-based model, the Artificial Intelligence Forecasting System (AIFS), alongside the IFS to provide an additional forecast dataset. Operational since February 25, 2025, AIFS is a deterministic model trained on ERA5 data, producing forecasts every six hours for up to 15 days at a $0.25^o \times 0.25^o$ resolution (Lang et al. (2024)). This output can be readily used to improve marine fog

predictions using our trained ML models.

Another critical factor influencing forecast accuracy is the reliability of the ERA5 dataset. Velikou et al. (2022) evaluated ERA5 temperature data and found a tendency to underestimate temperatures, which they attributed to the discrepancies between grid points and actual station locations. Haiden et al. (2021) evaluated all the ERA5 data variables and did show some difference between observations and the forecast values as well. Increasing the availability of observation-based data for ML training could enhance predictive accuracy. Field campaigns such as FATIMA provide valuable datasets that can support ML training, as demonstrated by Gultepe et al. (2024a) in their fog nowcasting approach. Nowcasting fog can be another method we can explore to increase the accuracy of fog prediction. We have not used visibility as one of our features compared to all the other research, mainly to be able to do longer forecasts. We briefly tried nowcasting using our approach (1-hour to 6-hour), meaning that we used visibility as a feature as well and obtained significant improvements on the performance until the 6 hour lead time (F1-score for fog was around 0.85 for 1-hour nowcast).

Finally, it is important to consider the broader context in which marine fog is evolving. A growing body of research suggests that climate change is contributing to a decline in fog frequency in many regions. Johnstone and Dawson (2010) looked at the summer marine fog events from 1985 to 2008 in the coast Redwood region in California. They noticed a 33% decrease in fog occurrence since the 21st century. In Southern Iraq, Taha and Abduljabbar (2024) showed a similar trend at various of the weather stations with very few showing a slight increase in fog occurrence. This trend is largely driven by rising sea surface and air temperatures, which reduce the air-sea temperature contrast essential for advection fog formation, particularly for coastal regions such as St. John's and Yarmouth. (Torregrosa et al. (2014)). In addition, changes in wind patterns, the weakening of cold ocean currents

such as the Labrador Current, and reductions in aerosol concentrations all play a role in altering fog dynamics. These shifts may result in fewer fog events or a redistribution of when and where they occur. As such, understanding and predicting marine fog in a warming climate requires not only accurate models but also adaptive frameworks such as ML models that consider these evolving environmental conditions more efficiently.

References

Aaboub, F., H. Chamlal, and T. Ouaderhman (2023, 10). Statistical analysis of various splitting criteria for decision trees. *Journal of Algorithms & Computational Technology 17*. DOI: 10.1177/17483026231198181.

aishwarya.27 (2024). Introduction to recurrent neural networks. https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/. [Online; accessed 31-January-2025].

Ajanki, A. (2010). Own work, cc by-sa 3.0,. https://commons.wikimedia.org/w/index.php?curid=2170282.

Altman, N. and M. Krzywinski (2015). Simple linear regression. https://doi.org/10.1038/nmeth.3627.

American Meteorogical Society (2024). Persistence forecast. https://glossary.ametsoc.org/wiki/Persistence_forecast.

Aurelius, R. and A. Viadinugroho (2021). Imbalanced classification in python: Smote-enn method. https://towardsdatascience.com/imbalanced-classification-in-python-smote-enn-method-db5db06b8d50. Accessed: 2024-10-17.

Bartokov, I., A. Bott, J. Bartok, and M. Gera (2015). Fog prediction for road traffic safety in a coastal desert region: Improvement of nowcasting skills by the machine-learning

approach. https://journals.ametsoc.org/view/journals/wefo/22/2/waf980_1.xml. DOI: 10.1007/s10546-015-0069-x.

Batista, G. E. A. P. A., R. C. Prati, and M. C. Monard (2004, June). A study of the behavior of several methods for balancing machine learning training data. https://doi.org/10.1145/1007730.1007735. DOI:10.1145/1007730.1007735.

Bengio, Y., P. Simard, and P. Frasconi (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks 5*(2), 157–166. DOI: 10.1109/72.279181.

Beyer, H. (1981). Tukey, john w.: Exploratory data analysis. addison-wesley publishing company reading, mass. menlo park, cal., london, amsterdam, don mills, ontario, sydney 1977, xvi, 688 s. *Biometrical Journal 23*(4), 413–414. DOI: https://doi.org/10.1002/bimj.4710230408.

Bhuvaneswari, G. (2020). What is entropy and information gain? how are they used to construct decision trees? https://www.numpyninja.com/post/what-is-entropy-and-information-gain-how-are-they-used-to-construct-decision-trees. [Online; accessed 05-November-2024].

Bi, K., L. Xie, H. Zhang, X. Chen, X. Gu, and Q. Tian (2022). Pangu-weather: A 3d high-resolution model for fast and accurate global weather forecast. https://arxiv.org/abs/2211.02556.

Boneh, T., G. T. Weymouth, P. Newham, R. Potts, J. Bally, A. E. Nicholson, and K. B. Korb (2015). Fog forecasting for melbourne airport using a bayesian decision network. https://journals.ametsoc.org/view/journals/wefo/30/5/waf-d-15-0005_1.xml. DOI: 10.1175/WAF-D-15-0005.1.

Bothma, J. (2024). Seaborn heatmaps: A guide to data visualization. https://www.datacamp.com/tutorial/seaborn-heatmaps. [Online; accessed 15-January-2025].

Breiman, L. (2001). Random forests. https://doi.org/10.1023/A:1010933404324. DOI: 10.1023/A:1010933404324.

Bughici, T., N. Lazarovitch, E. Fredj, and E. Tas (2019). Evaluation and bias correction in wrf model forecasting of precipitation and potential evapotranspiration. https://journals.ametsoc.org/view/journals/hydr/20/5/jhm-d-18-0160_1.xml. DOI: 10.1175/JHM-D-18-0160.1.

Bullock, T., G. A. Isaac, J. Beale, and T. Hauser (2016, 10). Improvement of visibility and severe sea state forecasting on the grand banks of newfoundland and labrador. https://doi.org/10.4043/27406-MS.

Canavan, T. and W. Sanford (2007). Fog climatology near the atlantic coast of nova scotia. https://ams.confex.com/ams/87ANNUAL/techprogram/paper_116589.html.

Castillo-Botn, C., D. Casillas-Prez, C. Casanova-Mateo, S. Ghimire, E. Cerro-Prada, P. Gutierrez, R. Deo, and S. Salcedo-Sanz (2022). Machine learning regression and classification methods for fog events prediction. https://www.sciencedirect.com/science/article/pii/S0169809522001430. DOI: https://doi.org/10.1016/j.atmosres.2022.106157.

Chawla, N. V., K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer (2002, June). Smote: Synthetic minority over-sampling technique. http://dx.doi.org/10.1613/jair.953. DOI: 10.1613/jair.953.

Chen, C., M. Zhang, W. Perrie, R. Chang, X. Chen, P. Duplessis, and M. Wheeler (2020). Boundary layer parameterizations to simulate fog over atlantic canada waters. https://

agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2019EA000703. e2019EA000703 2019EA000703. DOI: https://doi.org/10.1029/2019EA000703.

Chen, K., Y. Zhou, T. Ren, and X. Li (2024). Short-term sea fog area forecast: A new data set and deep learning approach. https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2024JH000230. DOI: https://doi.org/10.1029/2024JH000230.

Chen, L., Z. Ma, and Z. Xu (2021, 07). Evaluation of the wrf physics ensemble using a multivariable integrated evaluation approach over the haihe river basin in northern china. 10.1007/s00382-021-05723-x.

cherriebllxk7 (2024). Introduction to recurrent neural networks. https://www.geeksforgeeks.org/difference-between-forecasting-and-prediction. [Online; accessed 13-February-2025].

Chug, A. (2024). What is lstm long short term memory? https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/. [Online; accessed 03-March-2025].

Colabone, R. d. O., A. L. Ferrari, F. A. d. S. Vecchia, and A. R. B. Tech (2015). Application of artificial neural networks for fog forecast. *Journal of Aerospace Technology and Management 7*, 240–246.

Cook, E. and L. Goldman (1984). Empiric comparison of multivariate analytic techniques: Advantages and disadvantages of recursive partitioning analysis. https://www.sciencedirect.com/science/article/pii/0021968184900419. DOI: https://doi.org/10.1016/0021-9681(84)90041-9.

Cornejo-Bueno, L., C. Casanova-Mateo, J. Sanz-Justo, E. Cerro-Prada, and S. Salcedo-Sanz (2017). Efficient prediction of low-visibility events at airports using machine-learning regression. *Boundary-layer meteorology 165*(2), 349–370.

da Silva, D., M. Geller, M. Moura, and A. Meneses (2022, 01). Performance evaluation of lstm neural networks for consumption prediction. *e-Prime - Advances in Electrical Engineering, Electronics and Energy 2*, 100030. DOI: 10.1016/j.prime.2022.100030.

Davis, C. (1962). The norm of the schur product operation. *Numerische Mathematik*. DOI: 10.1007/BF01386329.

Developmental Testbed Centre (2009). This is a single column model test case. https://github.com/yyr/wrf/blob/master/test/em_scm_xy/README.scm. [Online; accessed 26-December-2024].

Dorman, C. E., D. R. Koracin, and R. Dimitrova (2022, December). A Fall Fog Bank Along Nova Scotia Generated by Coastal Sea and Air Turbulent Processes. https://ui.adsabs.harvard.edu/abs/2022AGUFM.A34D..06D. Provided by the SAO/NASA Astrophysics Data System.

Durán-Rosal, A. M., J. C. Fernández, C. Casanova-Mateo, J. Sanz-Justo, S. Salcedo-Sanz, and C. Hervás-Martínez (2018). Efficient fog prediction with multi-objective evolutionary neural networks. *Applied Soft Computing 70*, 347–358.

ECMWF (2024). IFS Documentation Cycle 49r1. https://www.ecmwf.int/en/forecasts/documentation-and-support/changes-ecmwf-model.

Environment and Natural Resource (2024). Historical data. https://climate.weather.gc.ca/historical_data/search_historic_data_e.html. [Online; accessed 13-September-2024].

Fabbian, D., R. de Dear, and S. Lellyett (2007). Application of artificial neural network forecasts to predict fog at canberra international airport. https://journals.ametsoc.org/view/journals/wefo/22/2/waf980_1.xml. DOI: 10.1175/WAF980.1.

Flynn, C. (2018). The physics of fog. https://blog.metservice.com/Physics-of-Fog. [Online; accessed 25-December-2024].

Friedman, B. (1984). *Classification and Regression Trees (1st ed.)*. Chapman and Hall/CRC. DOI: https://doi.org/10.1201/9781315139470.

Gao, X., X. Bao, S. Ma, Q. Chen, and B. Wang (2024). An online assimilation method to improve the numerical forecast of sea fog using microwave radiometer-retrieved cloud water path. https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2023JD040229. DOI: https://doi.org/10.1029/2023JD040229.

Geurts Pierre, Ernst Damien, W. L. (2006). Extremely randomized trees. https://doi.org/10.1007/s10994-006-6226-1. DOI: 10.1007/s10994-006-6226-1.

Glossary of Meteorology (2024). Glossary of meteorology- fog. https://glossary.ametsoc.org/wiki/Fog.

Glosser.ca (2013). Own work, derivative of file:artificial neural network.svg, cc by-sa 3.0. https://commons.wikimedia.org/w/index.php?curid=24913461.

Gonzalez, E., A. Dowling, H. J. S. Fernando, E. D. Creegan, H.-S. Lee, K.-I. Chang, I. Gultepe, Q. Wang, D. G. Ortiz-Suslow, and J. Ruiz-Plancarte (2023, December). Marine Fog Observations During the 2023 FATIMA Yellow Sea Field Campaign. https://ui.adsabs.harvard.edu/abs/2023AGUFM.A51S2233G.

Google (2024). Google maps. https://maps.google.com.

Google For Developers (2024a). Classification: Accuracy, recall, precision, and related metrics. https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall. Accessed: 2025-02-10.

Google For Developers (2024b). Datasets: Dividing the original dataset. https://developers.google.com/machine-learning/crash-course/overfitting/dividing-datasets?hl=en. Accessed: 2024-10-14.

Google For Developers (2024c). Datasets: Imbalanced datasets. https://developers.google.com/machine-learning/crash-course/overfitting/imbalanced-datasets. Accessed: 2024-10-07.

Google For Developers (2024d). Datasets: Imbalanced datasets- downsampling and upweighting. https://developers.google.com/machine-learning/crash-course/overfitting/imbalanced-datasets. Accessed: 2024-10-17.

Google For Developers (2024e). Supervised learning. https://developers.google.com/machine-learning/intro-to-ml/supervised. Accessed: 2024-08-27.

Google For Developers (2024f). Thresholds and the confusion matrix. https://developers.google.com/machine-learning/crash-course/classification/thresholding. Accessed: 2025-02-10.

Google For Developers (2024g). Working with numerical data. https://developers.google.com/machine-learning/crash-course/numerical-data. Accessed: 2024-08-27.

Graves, A. (2013). Generating sequences with recurrent neural networks. http://arxiv.org/abs/1308.0850.

Gruslys, A., R. Munos, I. Danihelka, M. Lanctot, and A. Graves (2016). Memory-efficient backpropagation through time. http://arxiv.org/abs/1606.03401.

Guijo-Rubio, D., P. Gutiérrez, C. Casanova-Mateo, J. Sanz-Justo, S. Salcedo-Sanz, and

C. Hervás-Martínez (2018). Prediction of low-visibility events due to fog using ordinal classification. *Atmospheric Research 214*, 64–73.

Gultepe, E., S. Wang, B. Blomquist, H. J. S. Fernando, O. P. Kreidl, D. J. Delene, and I. Gultepe (2024a). Generative nowcasting of marine fog visibility in the grand banks area and sable island in canada. https://arxiv.org/abs/2402.06800.

Gultepe, E., S. Wang, B. Blomquist, H. J. S. Fernando, O. P. Kreidl, D. J. Delene, and I. Gultepe (2024b). Machine learning analysis and nowcasting of marine fog visibility using fatima grand banks campaign measurements. https://www.frontiersin.org/journals/earth-science/articles/10.3389/feart.2023.1321422. DOI: 10.3389/feart.2023.1321422.

Haby, J. (2024). More in depth - radiation/advection fog. https://www.weather.gov/source/zhu/ZHU_Training_Page/fog_stuff/forecasting_fog/RADIATION_ADVECTION_FOG.html. [Online; accessed 22-January-2025].

Haiden, T., M. Janousek, F. Vitart, Z. Ben-Bouallegue, L. Ferranti, and F. Prates (2021, 09/2021). Evaluation of ecmwf forecasts, including the 2021 upgrade. https://www.ecmwf.int/node/20142. DOI: 10.21957/90pgicjk4.

Hersbach, H., B. Bell, P. Berrisford, S. Hirahara, A. Hornyi, J. Muoz-Sabater, J. Nicolas, C. Peubey, R. Radu, D. Schepers, A. Simmons, C. Soci, S. Abdalla, X. Abellan, G. Balsamo, P. Bechtold, G. Biavati, J. Bidlot, M. Bonavita, G. De Chiara, P. Dahlgren, D. Dee, M. Diamantakis, R. Dragani, J. Flemming, R. Forbes, M. Fuentes, A. Geer, L. Haimberger, S. Healy, R. J. Hogan, E. Hlm, M. Janiskov, S. Keeley, P. Laloyaux, P. Lopez, C. Lupu, G. Radnoti, P. de Rosnay, I. Rozum, F. Vamborg, S. Villaume, and J.-N. Thpaut (2020). The era5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society 146*(730), 1999–2049. DOI: https://doi.org/10.1002/qj.3803.

Hinton, G., L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine 29*(6), 82–97. DOI: 10.1109/MSP.2012.2205597.

Hochreiter, S. and J. Schmidhuber (1997, November). Long short-term memory. https://doi.org/10.1162/neco.1997.9.8.1735. DOI: 10.1162/neco.1997.9.8.1735.

Holbrook, R. and A. Cook (2023). Time series as features. https://www.kaggle.com/code/ryanholbrook/time-series-as-features. [Online; accessed 22-January-2025].

Hsieh, W. W. (2023). *Introduction to Environmental Data Science*. Cambridge University Press.

Ibebuchi, C. C., C. C. Lee, A. Silva, and S. C. Sheridan (2024). Evaluating apparent temperature in the contiguous united states from four reanalysis products using artificial neural networks. *Journal of Geophysical Research: Machine Learning and Computation 1*(2), e2023JH000102. e2023JH000102 2023JH000102. DOI: https://doi.org/10.1029/2023JH000102.

Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. http://arxiv.org/abs/1502.03167.

Iowa Department of Transportation (2008). Airport manager/operators guide to awos operations. https://iowadot.gov/aviation/managersandsponsors/Resources/2-AWOS%20Guide.pdf.

Isaac, G. A., T. Bullock, J. Beale, and S. Beale (2020). Characterizing and predicting marine fog offshore newfoundland and labrador. https://journals.ametsoc.org/view/journals/wefo/35/2/waf-d-19-0085.1.xml. DOI: 10.1175/WAF-D-19-0085.1.

Jafarigol, E. and T. Trafalis (2023). A review of machine learning techniques in imbalanced data and future trends. https://arxiv.org/abs/2310.07917.

Johnstone, J. A. and T. E. Dawson (2010). Climatic context and ecological implications of summer fog decline in the coast redwood region. https://www.pnas.org/doi/abs/10.1073/pnas.0915062107. DOI: 10.1073/pnas.0915062107.

Josh, H. (2009). This is a single column model test case. https://github.com/yyr/wrf/blob/master/test/em_scm_xy/README.scm. [Online; accessed 26-December-2024].

Kamangir, H., W. Collins, P. Tissot, S. A. King, H. T. H. Dinh, N. Durham, and J. Rizzo (2021). Fognet: A multiscale 3d cnn with double-branch dense block and attention mechanism for fog prediction. https://www.sciencedirect.com/science/article/pii/S2666827021000190. DOI: https://doi.org/10.1016/j.mlwa.2021.100038.

Kingma, D. P. and J. Ba (2017). Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980.

Korain, D. and C. E. Dorman (2017). *Marine Fog: Challenges and Advancements in Observations, Modeling, and Forecasting.* Springer Atmospheric Sciences. Springer.

Korain, D., C. E. Dorman, J. M. Lewis, J. G. Hudson, E. M. Wilcox, and A. Torregrosa (2014). Marine fog: A review. https://www.sciencedirect.com/science/article/pii/S016980951300361X. DOI: https://doi.org/10.1016/j.atmosres.2013.12.012.

Koziara, M. C., R. J. Renard, and W. J. Thompson (1983). Estimating marine fog probability using a model output statistics scheme. https://journals.ametsoc.org/view/journals/mwre/111/12/1520-0493_1983_111_2333_emfpua_2_0_co_2.xml. DOI: 10.1175/1520-0493(1983)111¡2333:EMFPUA¿2.0.CO;2.

Kuhn, M. and K. Johnson (2013, 01). Applied predictive modeling. DOI: 10.1007/978-1-4614-6849-3.

Kumar, P. (2020). Top 10 categories in ml and dl. https://www.kaggle.com/discussions/general/230511?focusReplyOnRender=true. [Online; accessed 24-October-2024].

Kumar, S. (2023). Weather prediction. https://www.kaggle.com/code/sayamkumar/weather-prediction. [Online; accessed 24-October-2024].

Kunihiko, F. (1975). Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics 20*, 121–136. DOI: 10.1007/BF00342633.

Lam, R., A. Sanchez-Gonzalez, M. Willson, P. W. Wirnsberger, M. Fortunato, F. Alet, S. Ravuri, T. Ewalds, Z. Eaton-Rosen, W. Hu, A. Merose, S. Hoyer, G. Holland, O. Vinyals, J. Stott, A. Pritzel, S. Mohamed, and P. Battaglia (2023). Graphcast: Learning skillful medium-range global weather forecasting. https://arxiv.org/abs/2212.12794.

Lang, S., M. Alexe, M. Chantry, J. Dramsch, F. Pinault, B. Raoult, M. C. A. Clare, C. Lessig, M. Maier-Gerber, L. Magnusson, Z. B. Bouallgue, A. P. Nemesio, P. D. Dueben, A. Brown, F. Pappenberger, and F. Rabier (2024). Aifs – ecmwf's data-driven forecasting system. https://arxiv.org/abs/2406.01465.

Lee, J.-H., K.-I. Chang, J. W. Cha, and H. J. S. Fernando (2023, December). Highlights of FATIMA Yellow Sea 2023 Campaign. https://ui.adsabs.harvard.edu/abs/2023AGUFM.A44F..02L.

Lemaître, G., F. Nogueira, and C. K. Aridas (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. http://jmlr.org/papers/v18/16-365.html.

Liu, Y., L. Zhuo, and D. Han (2023). Developing spin-up time framework for wrf extreme precipitation simulations. https://www.sciencedirect.com/science/article/pii/S0022169423003852. DOI: https://doi.org/10.1016/j.jhydrol.2023.129443.

Ludlow, L. and S. Perez (2018, 01). Addressing autocorrelation in time series data: A comparison of four analytic methods using data from college course evaluations. *General Linear Model Journal 44*, 11–23. DOI: 10.31523/glmj.044001.002.

Malakar, P., A. Kesarkar, J. Bhate, V. Singh, and A. Deshamukhya (2020). Comparison of reanalysis data sets to comprehend the evolution of tropical cyclones over north indian ocean. *Earth and Space Science 7*(2), e2019EA000978. e2019EA000978 2019EA000978. DOI: https://doi.org/10.1029/2019EA000978.

Malik, K., T. Ahmad, M. Farhan, A. Muhammad, S. Jabbar, S. Khalid, and M. Kim (2015, 09). Big-data: Transformation from heterogeneous data to semantically-enriched simplified data. *Multimedia Tools and Applications 75*. DOI: 10.1007/s11042-015-2918-5.

Matthias, D. (2018). Prediction vs forecasting. https://www.datascienceblog.net/post/machine-learning/forecasting_vs_prediction/. [Online; accessed 13-February-2025].

McCandlish, S., J. Kaplan, D. Amodei, and O. D. Team (2018). An empirical model of large-batch training. http://arxiv.org/abs/1812.06162.

McCulloch Warren S, P. W. (1943, 12). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics 5*, 99–115. DOI: 10.1007/BF02478259.

Miao, K.-c., T.-t. Han, Y.-q. Yao, H. Lu, P. Chen, B. Wang, and J. Zhang (2020). Application of lstm for short term fog forecasting based on meteorological elements. *Neurocomputing 408*, 285–291.

Mohammed, R., J. Rawashdeh, and M. Abdullah (2020). Machine learning with oversampling and undersampling techniques: Overview study and experimental results. In *2020 11th International Conference on Information and Communication Systems (ICICS)*, pp. 243–248. DOI: 10.1109/ICICS49469.2020.239556.

Nair, V. and G. E. Hinton (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, Madison, WI, USA, pp. 807814. Omnipress.

Namin, A. H., K. Leboeuf, R. Muscedere, H. Wu, and M. Ahmadi (2009). Efficient hardware implementation of the hyperbolic tangent sigmoid function. In *2009 IEEE International Symposium on Circuits and Systems*, pp. 2117–2120. DOI: 10.1109/ISCAS.2009.5118213.

National Center for Atmospheric Research (2022). Fog forecasting to avoid delays and accidents. https://ral.ucar.edu/pressroom/features/fog-forecasting-to-avoid-delays-accidents. [Online; accessed 25-December-2024].

National Centers for Environmental Prediction, National Weather Service, NOAA, U.S. Department of Commerce (2015a). Ncep gdas/fnl 0.25 degree global tropospheric analyses and forecast grids. https://rda.ucar.edu/datasets/dsd083003/.

National Centers for Environmental Prediction, National Weather Service, NOAA, U.S. Department of Commerce (2015b). Ncep gfs 0.25 degree global forecast grids historical archive. https://rda.ucar.edu/datasets/dsd084001/.

National Oceanic and Atmospheric Administration (2025). Estimating wind speed. https://www.weather.gov/pqr/wind. [Online; accessed 24-January-2025].

NavCanada (2017). Aviation weather services guide for aviation users. https://www.navcanada.ca/en/aviation-weather-services-guide.pdf.

NavCanada (2024). Advetion fog. https://avmet.navcanada.ca/en/advection-fog.aspx.

Nelli, N., D. Francis, R. Abida, R. Fonseca, O. Masson, and E. Bosc (2024). In-situ measurements of fog microphysics: Visibility parameterization and estimation of fog droplet sedimentation velocity. https://www.sciencedirect.com/science/article/pii/S0169809524003521. DOI: https://doi.org/10.1016/j.atmosres.2024.107570.

Njeri, N. R. (2022). Data preparation for machine learning modelling. *International Journal of Computer Applications Technology and Research 11*, 231–235. DOI: 10.7753/IJCATR1106.1008.

Pappa, A., E. Siouti, S. N. Pandis, and I. Kioutsioukis (2023). High-resolution wrf forecasts in the smartaq system: Evaluation of the meteorological forcing used for pmcamx predictions in an urban area. https://www.sciencedirect.com/science/article/pii/S0169809523004386. DOI: https://doi.org/10.1016/j.atmosres.2023.107041.

Pascanu, R., T. Mikolov, and Y. Bengio (2012). Understanding the exploding gradient problem. http://arxiv.org/abs/1211.5063.

Pathak, J., S. Subramanian, P. Harrington, S. Raja, A. Chattopadhyay, M. Mardani, T. Kurth, D. Hall, Z. Li, K. Azizzadenesheli, P. Hassanzadeh, K. Kashinath, and A. Anandkumar (2022). Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. https://arxiv.org/abs/2202.11214.

Pearson, K. and O. M. F. E. Henrici (1895). X. contributions to the mathematical theory of evolution.ii. skew variation in homogeneous material. https://royalsocietypublishing.org/doi/abs/10.1098/rsta.1895.0010. DOI: 10.1098/rsta.1895.0010.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12*, 2825–2830.

Pelez-Rodrguez, C., J. Prez-Aracil, C. Casanova-Mateo, and S. Salcedo-Sanz (2023). Efficient prediction of fog-related low-visibility events with machine learning and evolutionary algorithms. *Atmospheric Research 295*, 106991.

Rao, K. (2021). Splitting data randomly can ruin your model. https://datascience.stanford.edu/news/splitting-data-randomly-can-ruin-your-model. Accessed: 2024-10-15.

Ratner, B. (2009). The correlation coefficient: Its values range between +1/1, or do they? *Journal of Targeting, Measurement and Analysis for Marketing*. DOI: https://doi.org/10.1057/jt.2009.5.

Reen, B. (2016). A brief guide to observation nudging in wrf. https://www2.mmm.ucar.edu/wrf/users/docs/ObsNudgingGuide.pdf.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. https://doi.org/10.1037/h0042519.

Rosenblatt, F. (1962). Principles of neurodynamics; perceptrons and the theory of brain mechanisms. https://catalog.hathitrust.org/Record/000203591.

Sagar, S. (2025). Regression in machine learning. https://www.geeksforgeeks.org/regression-in-machine-learning/. [Online; accessed 19-February-2025].

Schuster, M. and K. Paliwal (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing 45*(11), 2673–2681. DOI: 10.1109/78.650093.

Scikit-Learn (2024a). Compare the effect of different scalers on data with outliers. https://scikit-learn.org/dev/auto_examples/preprocessing/plot_all_scaling.html#plot-all-scaling-standard-scaler-section. [Online; accessed 12-October-2024].

Scikit-Learn (2024b). Decision trees. https://scikit-learn.org/1.5/modules/tree.html#mathematical-formulation. [Online; accessed 05-November-2024].

Scikit-Learn (2024c). Ensembles: Gradient boosting, random forests, bagging, voting, stacking. https://scikit-learn.org/stable/modules/ensemble.html. [Online; accessed 21-January-2025].

Scikit-Learn (2024d). Extratreesclassifier. https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html. [Online; accessed 13-December-2024].

Scikit-Learn (2024e). Importance of feature scaling. https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html. [Online; accessed 12-October-2024].

Scikit-Learn (2024f). Labelencoder. https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.LabelEncoder.html. [Online; accessed 12-October-2024].

Scikit-Learn (2024g). Randomsearchcv. https://scikit-learn.org/1.5/modules/generated/sklearn.model_selection.RandomizedSearchCV.html. [Online; accessed 15-December-2024].

Scikit-Learn (2024h). Standardscaler. https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.StandardScaler.html. [Online; accessed 12-October-2024].

Scikit-Learn (2024i). Timeseriessplit. https://scikit-learn.org/1.6/modules/generated/sklearn.model_selection.TimeSeriesSplit.html. [Online; accessed 15-December-2024].

Scikit-Learn (2024j). Transforming the prediction target (y). https://scikit-learn.org/dev/modules/preprocessing_targets.html#preprocessing-targets. [Online; accessed 12-October-2024].

Scikit-Learn (2024k). Undersample. https://imbalanced-learn.org/stable/under_sampling.html#edited-nearest-neighbors. [Online; accessed 23-October-2024].

Scikit-Learn (2025a). Extratreesregressor. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesRegressor.html. [Online; accessed 20-February-2025].

Scikit-Learn (2025b). Onehotencoder. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html. [Online; accessed 17-February-2025].

Scikit-Learn (2025c). Quantile regression. https://scikit-learn.org/stable/auto_examples/linear_model/plot_quantile_regression.html. [Online; accessed 15-December-2024].

Singh, Y., M. Saini, and Savita (2023). Impact and performance analysis of various activation functions for classification problems. DOI: 10.1109/InC457730.2023.10263129.

Skamarock, W. C., J. B. Klemp, J. Dudhia, D. O. Gill, D. M. Barker, M. G. Duda, X. Y. Huang, W. Wang, and J. G. Powers (2019). A Description of the Advanced Research WRF Version 4. https://www2.mmm.ucar.edu/wrf/users/docs/arw_v4.pdf.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: A simple way to prevent neural networks from overfitting. http://jmlr.org/papers/v15/srivastava14a.html.

Stensrud, D. J. (2007). *Why study parameterization schemes?*, pp. 111. Cambridge University Press.

Swastik (2024). Smote for imbalanced classification with python. https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/. [Online; accessed 18-October-2024].

Taha, N. W. and H. S. Abduljabbar (2024). Coastal fog, climate change, and the environment. https://www.e3s-conferences.org/articles/e3sconf/abs/2024/113/e3sconf_itese2024_02014/e3sconf_itese2024_02014.html. DOI: https://doi.org/10.1051/e3sconf/202458302014.

Taylor, P. A., Z. Chen, L. Cheng, S. Afsharian, W. Weng, G. A. Isaac, T. W. Bullock, and Y. Chen (2021). Surface deposition of marine fog and its treatment in the weather research and forecasting (wrf) model. https://acp.copernicus.org/articles/21/14687/2021/. DOI: 10.5194/acp-21-14687-2021.

Tensorflow (2024). Classification on imbalanced data. https://www.tensorflow.org/learn.

The Imbalanced-Learn Developers (2024). Compare sampler combining over- and under-sampling. https://imbalanced-learn.org/stable/auto_examples/combine/plot_comparison_combine.html#sphx-glr-auto-examples-combine-plot-comparison-combine-py. [Online; accessed 18-October-2024].

The pandas development team (2020, February). pandas-dev/pandas: Pandas. https://doi.org/10.5281/zenodo.3509134. DOI:10.5281/zenodo.3509134.

Thompson, G. and T. Eidhammer (2014). A study of aerosol impacts on clouds and precipitation development in a large winter cyclone. https://journals.ametsoc.org/view/journals/atsc/71/10/jas-d-13-0305.1.xml. DOI: 10.1175/JAS-D-13-0305.1.

Thompson, G., P. R. Field, R. M. Rasmussen, and W. D. Hall (2008). Explicit forecasts of winter precipitation using an improved bulk microphysics scheme. part ii: Implementation of a new snow parameterization. https://journals.ametsoc.org/view/journals/mwre/136/12/2008mwr2387.1.xml.

Tofalli, C. (2015). A better measure of relative prediction accuracy for model selection and model estimation. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2635088#.

Tomek, I. (1976). Two modifications of cnn. *IEEE Transactions on Systems, Man, and Cybernetics SMC-6*(11), 769–772. DOI: 10.1109/TSMC.1976.4309452.

Torregrosa, A., T. A. Obrien, and I. C. Faloona (2014). Coastal fog, climate change, and the environment. https://eos.org/features/coastal-fog-climate-change-environment.

Tukey, J. W. (1977). *Exploratory Data Analysis*. Reading, Massachusetts: Addison-Wesley.

Tuychiev, B. (2023). Understanding skewness and kurtosis and how to plot them. https://www.datacamp.com/tutorial/understanding-skewness-and-kurtosis.

University of Notre Dame (2021-2026). Fatima- project overview. https://efmlab.nd.edu/research/fatima/project-overview/. [Online; accessed 26-December-2024].

VandenBoer, T. C. (2022). Fog and turbulence interactions in the marine atmosphere (fatima) research cruise. https://www.tcvandenboer.ca/group-blog/2022/10/10/fog-

and-turbulence-interactions-in-the-marine-atmosphere-fatima-research-cruise. [Online; accessed 26-December-2024].

Velikou, K., G. Lazoglou, K. Tolika, and C. Anagnostopoulou (2022). Reliability of the era5 in replicating mean and extreme temperatures across europe. https://www.mdpi.com/2073-4441/14/4/543. DOI: 10.3390/w14040543.

Wagh, S., R. Krishnamurthy, C. Wainwright, S. Wang, C. Dorman, H. Fernando, and I. Gultepe (2021, 12). Study of stratus-lowering marine-fog events observed during c-fog. *Boundary-Layer Meteorology 181*, 1–28. DOI: 10.1007/s10546-021-00670-w.

Werbos, P. (1974, 01). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science. Thesis (Ph. D.). Appl. Math. Harvard University.* Ph. D. thesis, Harvard University, Cambridge.

Wikipedia contributors (2024a). Euclidean distance — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Euclidean_distance&oldid=1241824394. [Online; accessed 23-October-2024].

Wikipedia contributors (2024b). Joint probability distribution — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Joint_probability_distribution&oldid=1263530803. [Online; accessed 23-January-2025].

Wikipedia contributors (2024c). Pearson correlation coefficient — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Pearson_correlation_coefficient&oldid=1246912561. [Online; accessed 18-October-2024].

Wikipedia contributors (2025a). Biasvariance tradeoff — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Bias%E2%80%93variance_tradeoff&oldid=1268873330. [Online; accessed 13-January-2025].

Wikipedia contributors (2025b). Kernel density estimation — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Kernel_density_estimation&oldid=1268389204. [Online; accessed 23-January-2025].

Wikipedia contributors (2025c). Mean absolute error — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Mean_absolute_error&oldid=1276071917. [Online; accessed 21-February-2025].

Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. https://api.semanticscholar.org/CorpusID:6699477.

WRF-ARW Online Tutorial (2024). Input sst data into the model. https://www2.mmm.ucar.edu/wrf/OnLineTutorial/CASES/SST/index.php. Accessed: 09/13/2024.

WRF Users Page (2024). Wrf model physics options and references. https://www2.mmm.ucar.edu/wrf/users/physics/phys_references.html.

Yadolah, D. (2008). Criterion of total mean squared error. https://doi.org/10.1007/978-0-387-32833-1_92. DOI: 10.1007/978-0-387-32833-1_92.

Yarker Consulting (2015). An introduction to parameterization options in wrf. https://yarkerconsulting.com/2015/11/21/intro-to-parameterizations-in-wrf/. [Online; accessed 26-December-2024].

Appendix A

Data Exploration code

All of the coding for ML and data exploration were done in Python. Below is the code sample for Fig. 2.17 which investigated how SMOTEENN worked.

```
# @title Cleaning the Data
# Load the CSV file into a DataFrame
# Define the path and pattern for the CSV files
path = 'Era5_Yarmouth.csv'

# Read all CSV files into a Dask DataFrame
dask_df = dd.read_csv(path)

# Compute the Dask DataFrame to get a Pandas DataFrame
combined_df = dask_df.compute()
# Remove rows where the first column is blank
df_cleaned = combined_df[combined_df['Vis'].notnull()]

# Save the cleaned DataFrame back to a CSV file
df_cleaned.to_csv('cleaned_file1.csv', index=False)
```

```
# Define the path and pattern for the CSV files
path2 = 'Era5_Yarmouth.csv'

# Read all CSV files into a Dask DataFrame
dask_df2 = dd.read_csv(path2)

# Compute the Dask DataFrame to get a Pandas DataFrame
combined_df2 = dask_df2.compute()
# Remove rows where the first column is blank
df_cleaned2 = combined_df2[combined_df2['Vis'].notnull()]

# Save the cleaned DataFrame back to a CSV file
df_cleaned2.to_csv('cleaned_file2.csv', index=False)
# Display the first 10 entries
#print(df.head(10))

# @title Data Preparation
# load the data
data = pd.read_csv('cleaned_file1.csv')
data1 = pd.read_csv('cleaned_file2.csv')
# Parse the Time column
data['Time'] = pd.to_datetime(data['Time'], format='%Y-%m-%d_%H:%M:%S')

# Extract time-based features
data['Hour'] = data['Time'].dt.hour
data['Day'] = data['Time'].dt.day
data['Month'] = data['Time'].dt.month
data['DayOfWeek'] = data['Time'].dt.dayofweek

# Encode the target variable
label_encoder = LabelEncoder()
```

```
data['class_vis'] = label_encoder.fit_transform(data['class_vis'])

# Split data into features and target
X = data.drop(['Time', 'class_vis', 'Vis','TD2','T2-TD2'], axis=1)  #
                                    Dropping 'Time' as it is now
                                    decomposed into useful features
y = data['class_vis']
X2 = data1.drop(['Time', 'class_vis', 'Vis','TD2','T2-TD2'], axis=1)  #
                                    Dropping 'Time' as it is now
                                    decomposed into useful features
# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X2_scaled = scaler.fit_transform(X2)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size
                                    =0.2, random_state=42, stratify=y)

# Apply SMOTE for data balancing
from imblearn.combine import SMOTEENN
from imblearn.over_sampling import BorderlineSMOTE
#smote = BorderlineSMOTE()
smote = SMOTEENN()
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE

#two features are chosen, 'RH2' and 'T2' and a binary target 'class'
```

```
X = data[['RH2', 'T2']]  # Choose 2 features for the 2D scatter plot
y = data['class_vis']  # Binary target

# Apply SMOTE
smote = SMOTEENN()
X_resampled, y_resampled = smote.fit_resample(X, y)

# Combine the resampled data for easier plotting
resampled_data = pd.DataFrame(X_resampled, columns=['RH2', 'T2'])
resampled_data['class'] = y_resampled

# Plot 1: Before SMOTE
plt.figure(figsize=(18, 10))

plt.subplot(1, 2, 1)  # 1 row, 2 columns, first plot

# Access the columns using their names to avoid the error
plt.scatter(X['RH2'], X['T2'], c=y, cmap="coolwarm", marker="o", alpha=0.7
                                    )

plt.title("Before SMOTE")
plt.xlabel("Relative Humidity at 2m/ %")
plt.ylabel("Temperature at 2m/K")
plt.colorbar(label='Class: clear=0, fog=1')

# Plot 2: After SMOTE
plt.subplot(1, 2, 2)  # 1 row, 2 columns, second plot

# Access the columns using their names for X_resampled as well
plt.scatter(X_resampled['RH2'], X_resampled['T2'], c=y_resampled, cmap="
                                    coolwarm", marker="x", alpha=0.7)
```

```
plt.title("After SMOTE")
plt.xlabel("Relative Humidity at 2m/ %")
plt.ylabel("Temperature at 2m/K")
plt.colorbar(label='Class: clear=0, fog=1')

plt.tight_layout()
plt.show()
```

The next code block was to look at the probability distributions of every feature for both St John's and Yarmouth, Canada which was Fig. 2.3.

```
# Load the data
data = pd.read_csv('cleaned_file1.csv')
data1 = pd.read_csv('cleaned_file2.csv')

# Assign data for clarity
X = data.drop(['Time', 'class_vis', 'Vis','TD2','T2-TD2'], axis=1)  #
                                    Dropping 'Time' as it is now
                                    decomposed into useful features
X2 = data1.drop(['Time', 'class_vis', 'Vis','TD2','T2-TD2'], axis=1)  #
                                    Dropping 'Time' as it is now
                                    decomposed into useful features

# Rescale 'P_sfc' by dividing by 1000 for both datasets
X['P_sfc'] = X['P_sfc'] / 1000
X2['P_sfc'] = X2['P_sfc'] / 1000

# Create a dictionary for units
unit_dict = {
    'T2': 'K',    # Temperature
    'RH2': '%',    # Relative Humidity
```

```
    'U': 'm/s',    # Wind Speed
    'V': 'm/s',    # Wind Speed (or direction, specify if needed)
    'P_sfc': 'kPa' # Surface Pressure (after rescaling)
}

# Rename the columns to include units
X.rename(columns={col: f"{col} ({unit_dict[col]})" for col in X.columns},
                                        inplace=True)
X2.rename(columns={col: f"{col} ({unit_dict[col]})" for col in X2.columns}
                                        , inplace=True)

# List of selected columns (now including units in the names)
selected_cols = [f"{col} ({unit_dict[col]})" for col in ['T2', 'RH2', 'U',
                                         'V', 'P_sfc']]

# Plot distributions
rows, cols = (len(selected_cols) + 1) // 2, 2
plt.figure(figsize=(3 * cols, 3 * rows))

for i, col in enumerate(selected_cols, 1):
    plt.subplot(rows, cols, i)
    # Plot for both datasets
    sns.histplot(X[col], color='blue', label='St John', kde=True, stat="
                                            density", common_norm=False)
    sns.histplot(X2[col], color='orange', label='Yarmouth', kde=True, stat
                                            ="density", common_norm=False)

    # Update the title and labels with units
    plt.title(f"{col}")
    plt.xlabel(f"{col}")
    plt.ylabel('Density')
```

```
    plt.legend()

plt.suptitle('Distribution Plots for St John and Yarmouth', y=1.02)
plt.tight_layout()
plt.show()
```

Then, we plotted the boxplots for the same features at both locations in Fig. 2.4 by first calculating the skewness and kurtosis:

```
#calculating skewness and kurtosis for Yarmouth and St John's
import pandas as pd

# Load the cleaned datasets
df1 = pd.read_csv('cleaned_file1.csv')
df2 = pd.read_csv('cleaned_file2.csv')

selected_cols = ['T2', 'RH2', 'U', 'V', 'P_sfc']

# Function to calculate skewness and kurtosis
def calculate_stats(df, col_name):
    skewness = df[col_name].skew()
    kurtosis = df[col_name].kurtosis()
    return skewness, kurtosis

# Print skewness and kurtosis for each selected column in both datasets
for col in selected_cols:
    skew1, kurt1 = calculate_stats(df1, col)
    skew2, kurt2 = calculate_stats(df2, col)
    print(f"Column: {col}")
    print(f"  Dataset 1 (cleaned_file1.csv): Skewness = {skew1:.2f},
                                        Kurtosis = {kurt1:.2f}")
    print(f"  Dataset 2 (cleaned_file2.csv): Skewness = {skew2:.4f},
```

```
                                        Kurtosis = {kurt2:.4f}")


import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# List of specific columns to plot (common columns in both X and X2)
selected_cols = ['T2', 'RH2', 'U', 'V', 'P_sfc']  # Columns of relevant
                                        features

# --- Plot: Boxplots for Both Datasets in Rows ---
plt.figure(figsize=(6, 3 * ((len(selected_cols) + 1) // 2)))  # Adjust
                                        figure size (width for each column,
                                        fixed height)

for i, col in enumerate(selected_cols, 1):
    plt.subplot((len(selected_cols) + 1) // 2, 2, i)

    # Use the column index instead of the column name to access data in
                                            the NumPy array
    col_index = X.columns.get_loc(col)  # Get the column index for 'col'
                                            in X

    # Get the minimum number of rows between the two arrays
    min_rows = min(X_scaled.shape[0], X2_scaled.shape[0])

    # Create a DataFrame to merge the data from both X and X2 for each
                                            column,
    # using only the first 'min_rows' rows of each array
    data_to_plot = pd.DataFrame({
```

```
        'St John': X_scaled[:min_rows, col_index],
        'Yarmouth': X2_scaled[:min_rows, col_index]
    })

    # Melt the data for seaborn's boxplot (long-form DataFrame)
    data_melted = data_to_plot.melt(var_name='Locations', value_name=col)

    # Plot the boxplot, with data from both datasets
    sns.boxplot(x='Locations', y=col, data=data_melted)

    plt.title(f"{col}")
    plt.xlabel('Locations')
    plt.ylabel(col)

plt.tight_layout()
plt.suptitle('Boxplots for St Johns and Yarmouth', y=1.02)
plt.show()
```

We also looked at the 2D scatter plots of two variables trying to understand which type of fog dominates the locations we chose.

```
# V and RH2 joint pdf scatter plots

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Load the datasets
df1 = pd.read_csv('cleaned_file1.csv')
df2 = pd.read_csv('cleaned_file2.csv')

# Select relevant columns
```

```
selected_cols = ['T2', 'RH2', 'U', 'V', 'P_sfc', 'Vis']
data = df2[selected_cols]

# Calculate wind speed and add it as a new column
data['Wind_Speed'] = (data['U']**2 + data['V']**2)**0.5

# Extract 'RH2' and 'V' for the joint PDF
x = data['RH2']
y = data['V']
# Create a joint PDF plot for RH2 and V
plt.figure(figsize=(8, 6))
# Assign the result of sns.kdeplot to a variable
contour_plot = sns.kdeplot(x=x, y=y, cmap="viridis", fill=True, levels=50,
                                    thresh=0.01, cbar=True)  # Add cbar=
                                   True

# Customize the plot
plt.title('Joint PDF of RH2 and V for Yarmouth', fontsize=14)
plt.xlabel('RH2 (%)', fontsize=12)
plt.ylabel('V (m/s)', fontsize=12)
plt.grid(True)
plt.show()

# joint pdf for wind speed and wind direction
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import gaussian_kde

# Load the dataset
file_path = "cleaned_file2.csv"
```

```
cleaned_data = pd.read_csv(file_path)

# Filter for foggy conditions
fog_data = cleaned_data[cleaned_data['class_vis'] == 'fog']

# Compute wind speed and meteorological wind direction
fog_data['wind_speed'] = np.sqrt(fog_data['U']**2 + fog_data['V']**2)
fog_data['wind_direction'] = (np.arctan2(fog_data['V'], fog_data['U']) *
                                    180 / np.pi) % 360
fog_data['wind_direction_met'] = (360 - fog_data['wind_direction']) % 360
                                     # Meteorological convention

# Convert wind direction to radians for plotting
wind_speed = fog_data['wind_speed']
wind_direction_met_radians = np.deg2rad(fog_data['wind_direction_met'])

# Perform 2D KDE for meteorological wind direction and wind speed
values_met = np.vstack([wind_direction_met_radians, wind_speed])
kde_met = gaussian_kde(values_met)

# Create a grid for the PDF
theta_met, r = np.meshgrid(
    np.linspace(0, 2 * np.pi, 100),  # Full circle
    np.linspace(0, wind_speed.max(), 100)  # Wind speed range
)
grid_coords_met = np.vstack([theta_met.ravel(), r.ravel()])

# Evaluate the KDE on the grid
pdf_met = kde_met(grid_coords_met).reshape(theta_met.shape)

# Plot the joint PDF using meteorological conventions
```

```
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, polar=True)
c = ax.contourf(theta_met, r, pdf_met, levels=50, cmap='viridis')
fig.colorbar(c, ax=ax, label='PDF Value')

# Set the angular labels for meteorological conventions (clockwise, 0 = North)
ax.set_theta_direction(-1)  # Clockwise
ax.set_theta_offset(np.pi / 2)  # 0 at the top (North)

ax.set_title('Joint PDF of Wind Speed and Direction (Yarmouth)', pad=30)
plt.show()
```

We looked at line plots as well for fog occurrence (Fig. 2.6 - Fig. 2.9):

```
# looking at the fog days
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('cleaned_file1.csv')

# Convert 'Time' column to datetime and extract the hour of the day
df['Time'] = pd.to_datetime(df['Time'], format='%Y-%m-%d_%H:%M:%S')
df['Hour'] = df['Time'].dt.hour

# Filter fog occurrences
fog_data = df[df['class_vis'] == 'fog']

# Calculate wind speed (based on U and V components)
fog_wind_speed = (fog_data['U']**2 + fog_data['V']**2)**0.5
U = fog_data['U']
```

```
V = fog_data['V']
# Plot: Fog occurrence by hour (frequency)
plt.figure(figsize=(12, 6))
fog_count_by_hour = fog_data['Hour'].value_counts().sort_index()
plt.plot(fog_count_by_hour.index, fog_count_by_hour.values, marker='o',
                                        color='blue', label='Fog Frequency')

plt.title('Fog Occurrence by Hour of Day(Yarmouth)', fontsize=14)
plt.xlabel('Hour of Day', fontsize=12)
plt.ylabel('Frequency of Fog', fontsize=12)
plt.xticks(range(0, 24))
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.legend()
plt.tight_layout()
plt.show()

# Plot: Fog wind speed by hour (scatter plot)
plt.figure(figsize=(12, 6))
plt.scatter(fog_data['Hour'], U, alpha=0.6, color='green', edgecolor='k')

plt.title('Fog Occurrence: U vs. Time of Day (St Johns)', fontsize=14)
plt.xlabel('Hour of Day', fontsize=12)
plt.ylabel('U (m/s)', fontsize=12)
plt.xticks(range(0, 24))
plt.grid(axis='both', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

Appendix B

ExtraTrees code

This appendix contains all the coding done for all the different approaches we did using ExtraTrees ML model. The first one is the code block for the RandomSearchCV to find the best parameters for St John's and Yarmouth.

```
# @title Find the parameters for Yarmouth

# Define the parameter distribution for RandomizedSearchCV
param_dist = {
    'n_estimators': randint(100, 500),
    'criterion': ['gini', 'log_loss', 'entropy'],
    'max_features': ['sqrt', 'log2', None],
    'max_depth': [10, 20, 30, 40, None],
    'min_samples_split': randint(2, 15),
    'min_samples_leaf': randint(1, 5),
    'class_weight': ['balanced', 'balanced_subsample'],
    'bootstrap': [True, False]
}

# Initialize the Extra Trees classifier
```

```
et_classifier = ExtraTreesClassifier()

# Initialize RandomizedSearchCV
tscv = TimeSeriesSplit(n_splits=5)
random_search = RandomizedSearchCV(estimator=et_classifier,
                                   param_distributions=param_dist,
                                   scoring= make_scorer(f1_score,
                                   average='weighted'), n_iter=50, cv=
                                   tscv, n_jobs=-1, verbose=2)

# Fit the model
random_search.fit(X_train_res, y_train_res)

# Print the best parameters
print("Best Parameters:", random_search.best_params_)
```

Then, we build and ran the ETClassifier for binary classification of fog at each location respectively:

```
# @title Building the Model (Yarmouth)
# Initialize and train ExtraTreeClassifier
et_classifier = ExtraTreesClassifier(
    n_estimators= 120,    # Number of trees 375
    criterion = 'log_loss', # criteria to split gini
    max_depth= None,       # Maximum depth of each tree 20
    max_features=None,  # Number of features to consider when looking for
                                    the best split sqrt
    min_samples_split=11,  # Minimum number of samples required to split
                                    an internal node 5
    min_samples_leaf=3,    # Minimum number of samples required to be at a
                                    leaf node 2
    bootstrap= False,     # Whether bootstrap samples are used when
```

```
                                        building trees True
    class_weight='balanced' # class weight for imbalanced adjusting
                                        balanced
)
# Fit the model
et_classifier.fit(X_train_res, y_train_res)

# Predictions
y_train_pred = et_classifier.predict(X_train_scaled)
y_test_prob = et_classifier.predict_proba(X_test_scaled)
y_test_pred = (y_test_prob[:, 1] >= 0.75).astype(int)

# Evaluation on Training Set
train_accuracy = accuracy_score(y_train, y_train_pred)
train_precision = precision_score(y_train, y_train_pred, average='weighted
                                        ')
train_recall = recall_score(y_train, y_train_pred, average='weighted')
train_f1 = f1_score(y_train, y_train_pred, average='weighted')

print(f"Training Set Evaluation:\nAccuracy: {train_accuracy:.4f}")
print(f"Precision: {train_precision:.4f}")
print(f"Recall: {train_recall:.4f}")
print(f"F1-Score: {train_f1:.4f}")

# Evaluation on Test Set
test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred, average='weighted')
test_recall = recall_score(y_test, y_test_pred, average='weighted')
test_f1 = f1_score(y_test, y_test_pred, average='weighted')

print(f"Test Set Evaluation:\nAccuracy: {test_accuracy:.4f}")
```

```
print(f"Precision: {test_precision:.4f}")
print(f"Recall: {test_recall:.4f}")
print(f"F1-Score: {test_f1:.4f}")
print("\nClassification Report:\n", classification_report(y_test,
                                    y_test_pred))

# Cross-Validation
cv_scores = cross_val_score(et_classifier, X, y, cv=5)
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Accuracy: {cv_scores.mean():.4f}")

# Feature Importance
importances = et_classifier.feature_importances_
feature_names = X.columns
feature_importance_df = pd.DataFrame(importances, index=feature_names,
                                    columns=['Importance']).sort_values('
                                    Importance', ascending=False)
print("\nFeature Importance:\n", feature_importance_df)

# Visualization of Feature Importance
plt.figure(figsize=(10, 6))
sns.barplot(x=feature_importance_df['Importance'], y=feature_importance_df
                                    .index)
plt.title('Feature Importance for Yarmouth')
plt.show()

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['
                                    clear', 'fog'], yticklabels=['clear',
```

```
                                        'fog'])
plt.title('Confusion Matrix for Yarmouth (No lag)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Then, in the discussion section, we tried different approaches to see if binary classification was the best option for predicting fog. The first one was combining all locations:

```
# @title Cleaning the Data
# Load the CSV file into a DataFrame
# Load and clean data
def load_and_clean_data(path):
    df = pd.read_csv(path)
    df = df[df['Vis'].notnull()]
    return df



location1_data = load_and_clean_data('Era5_Stjohn.csv')
location2_data = load_and_clean_data('Era5_Yarmouth.csv')

# Ensure the time columns are in datetime format
location1_data['Time'] = pd.to_datetime(location1_data['Time'], format='%Y
                                        -%m-%d_%H:%M:%S')
location2_data['Time'] = pd.to_datetime(location2_data['Time'], format='%Y
                                        -%m-%d_%H:%M:%S')


# Add location identifiers
location1_data['Location'] = 'StJohn'
location2_data['Location'] = 'Yarmouth'

# Combine the datasets
```

```
combined_data = pd.concat([location1_data, location2_data])
combined_data.to_csv('combined_data.csv', index=False)

# One-hot encode the location column
combined_data = pd.get_dummies(combined_data, columns=['Location'],
                                    drop_first=False)

# Binary classification: Map 'class_vis' to binary classes (assuming '
                                    clear' and 'fog' are the classes)
combined_data['class_vis'] = combined_data['class_vis'].map({'clear': 0, '
                                    fog': 1})

# Extract time-based features
combined_data['Hour'] = combined_data['Time'].dt.hour
combined_data['Day'] = combined_data['Time'].dt.day
combined_data['Month'] = combined_data['Time'].dt.month
combined_data['DayOfWeek'] = combined_data['Time'].dt.dayofweek

#Sort the data by time
combined_data = combined_data.sort_values(by='Time')
combined_data = combined_data[combined_data['Month'] > 3]

# Drop the original time column
combined_data = combined_data.drop(['Time'], axis=1)

# Create lag features
lag_features = [col for col in combined_data.columns if col not in ['
                                    class_vis','T2-TD2','TD2','Vis','Hour
                                    ','Day','Month','DayOfWeek','
                                    Location_StJohn','Location_Yarmouth']
                                    ]
```

```
for feature in lag_features:
    for lag in range(1, 2):  # Creating lag features for the past 2 time
                                                steps
        combined_data[f'{feature}_lag_{lag}'] = combined_data[feature].
                                                    shift(lag)

# Drop rows with missing values created by lag features
combined_data = combined_data.dropna()
# Preserve the locations data
location_columns = combined_data[['Location_StJohn', 'Location_Yarmouth']]

# Define features and target
X = combined_data.drop(['class_vis','Vis', 'T2-TD2','TD2'], axis=1)
y = combined_data['class_vis']

# Train-test split based on time
train_size = int(len(combined_data) * 0.9)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Split the locations into train and test sets
locations_train = location_columns[:train_size]
locations_test = location_columns[train_size:]

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply SMOTE for data balancing
smote = BorderlineSMOTE()
```

```
X_train_res, y_train_res = smote.fit_resample(X_train_scaled, y_train)
```

The second was to look at the multi-classification of clear, fog, and mist. The dataset for St John's was modified for multi-class as seen in the code block and the specifics to ensure we get the best results.

```
# @title Cleaning the Data
# Load the CSV file into a DataFrame
# Define the path and pattern for the CSV files
path = 'Era5multi_Stjohn.csv'

# Compute the Dask DataFrame to get a Pandas DataFrame
combined_df = pd.read_csv(path)
# Remove rows where the first column is blank
df_cleaned = combined_df[combined_df['Vis'].notnull()]

# Save the cleaned DataFrame back to a CSV file
df_cleaned.to_csv('cleaned_file.csv', index=False)

# @title Data Preparation
# Convert 'Time' column to datetime with the correct format
data = df_cleaned.copy()
data['Time'] = pd.to_datetime(data['Time'], format='%Y-%m-%d_%H:%M:%S',
                                        errors='coerce')

# Extract time-based features
data['Hour'] = data['Time'].dt.hour
data['Day'] = data['Time'].dt.day
data['Month'] = data['Time'].dt.month

# Sort the data by time
```

```
data = data.sort_values(by='Time')

data = data[data['Month'] > 3]
#data['Wind_Speed'] = (data['U']**2 + data['V']**2) ** 0.5

# Multi classification: Map 'class_vis' to multi classes
label_encoder = LabelEncoder()
data['class_vis'] = label_encoder.fit_transform(data['class_vis'])

# Create lag features
lag_features = ['T2','U','V','RH2','P_sfc', 'T2-TD2']  # feature names
for feature in lag_features:
    for lag in range(1, 2):  # Creating lag features for the past 2 time
                                        steps
        data[f'{feature}_lag_{lag}'] = data[feature].shift(lag)

# Drop rows with missing values created by lag features
data = data.dropna()

# Split data into features and target
X = data.drop(['Time', 'class_vis', 'Vis','TD2'], axis=1)  # Dropping '
                                    Time' as it is now decomposed into
                                    useful features
y = data['class_vis']

# Train-test split based on time
train_size = int(len(data) * 0.9)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Feature scaling
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply SMOTE for data balancing
from imblearn.combine import SMOTEENN
from imblearn.over_sampling import BorderlineSMOTE
smote= SMOTEENN()
X_train_res, y_train_res = smote.fit_resample(X_train_scaled, y_train)
from imblearn.under_sampling import RandomUnderSampler
# Counting the labels
unique, counts = np.unique(y_train, return_counts=True)
print(dict(zip(unique, counts)))
# Define undersampling strategy
undersample = RandomUnderSampler(sampling_strategy={0: 10000, 1: 4259, 2:
                                    2414}, random_state=42)

# Apply undersampling
#X_train_res, y_train_res = undersample.fit_resample(X_train, y_train)

# Check new class distribution
unique_resampled, counts_resampled = np.unique(y_train_res, return_counts=
                                    True)
print(dict(zip(unique_resampled, counts_resampled)))
```

Finally, we tested for Regression models as well. First, we used the ETRegressor and the way we prepared the data and trained the model are given below:

```
# @title Preparing the data
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,
                                    r2_score
```

```
# Load the dataset
df = pd.read_csv("cleaned_file.csv")

# Convert Time column to datetime and extract features
df['Time'] = pd.to_datetime(df['Time'], format='%Y-%m-%d_%H:%M:%S', errors
                                        ='coerce')
df['hour'] = df['Time'].dt.hour
df['dayofyear'] = df['Time'].dt.dayofyear
df['month'] = df['Time'].dt.month

# Log transform the target variable Vis
df['Vis'] = np.log1p(df['Vis'])  # log(Vis + 1) to handle zero cases

# Create lagged features except for Vis
lag_features = ['T2', 'TD2', 'T2-TD2', 'U', 'V', 'RH2', 'P_sfc']
for feature in lag_features:
    df[f'{feature}_lag1'] = df[feature].shift(1)

df.dropna(inplace=True)  # Remove rows with NaN values from lagging

# Drop unnecessary columns
df_reg = df.drop(columns=["class_vis"])

# Split into train (2012-2022) and test (2023) sets
df_train = df_reg[df_reg['Time'].dt.year < 2023]
df_test = df_reg[df_reg['Time'].dt.year == 2023]

# Separate features and target
X_train = df_train.drop(columns=["Time", "Vis"])
y_train = df_train["Vis"]
```

```
X_test = df_test.drop(columns=["Time", "Vis"])
y_test = df_test["Vis"]

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# @title Train the best model
# Train the best model with sample weights
log_fog_threshold = np.log1p(1.2)  # log(1 + x)
sample_weights = np.where(y_train < log_fog_threshold, 5, 1) # Higher
                                    weight for low visibility cases
best_et_regressor = ExtraTreesRegressor(
    n_estimators= 450,  # 450
    criterion= 'squared_error', # squared_error
    max_depth= 50,  # 50
    min_samples_split= 2, # 2
    min_samples_leaf= 3, # 3
    bootstrap= False, # False
    max_features= 'log2', #log2
    random_state=42)

best_et_regressor.fit(X_train, y_train, sample_weight= sample_weights)

# Make predictions
y_pred = best_et_regressor.predict(X_test)

# Looking at the results for ETRegressor
# Inverse log transform predictions
y_pred1 = np.expm1(y_pred)
```

```
y_test1 = np.expm1(y_test)

# Evaluate model performance
mae = mean_absolute_error(y_test1, y_pred1)
mse = mean_squared_error(y_test1, y_pred1)
r2 = r2_score(y_test1, y_pred1)

print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"R2 Score: {r2}")

# Plot first 24 hours of predictions vs actual
plt.figure(figsize=(10, 5))
plt.plot(df_test['Time'].iloc[4000:4072], y_test1.iloc[4000:4072], label='
                                    Actual', marker='o')
plt.plot(df_test['Time'].iloc[4000:4072], y_pred1[4000:4072], label='
                                    Predicted', marker='x')

# Add a horizontal red line at y = 1.2 km to indicate fog threshold
plt.axhline(y=1.2, color='r', linestyle='--', label='Fog Threshold (1.2 km
                                    )')

plt.xlabel('Time')
plt.ylabel('Visibility')
plt.title('72 Hours: Predicted vs Actual Visibility')
plt.legend()
plt.xticks(rotation=45)
plt.grid()
plt.show()
```

Then, we also tried a more robust approach, the quantile regression technique using

Random Forests, and the code block is given below:

```
# @title Quantile Regression Forest
from sklearn.ensemble import RandomForestRegressor
# Initialize Quantile Regression Forest (RandomForestRegressor with
                                        quantile support)

best_qrf = RandomForestRegressor(
    n_estimators= 200,
    max_depth= 10,
    min_samples_split= 10,
    min_samples_leaf= 4,
    random_state=42)

best_qrf.fit(X_train, y_train)

# Predict quantiles
def predict_quantiles(model, X, quantiles=[0.05, 0.5, 0.95]):
    preds = np.array([tree.predict(X) for tree in model.estimators_])
    return {q: np.percentile(preds, q*100, axis=0) for q in quantiles}

quantiles = predict_quantiles(best_qrf, X_test)

# Inverse log transform predictions
y_pred_median = np.expm1(quantiles[0.5])
y_test = np.expm1(y_test)

# Evaluate model performance
mae = mean_absolute_error(y_test, y_pred_median)
mse = mean_squared_error(y_test, y_pred_median)
r2 = r2_score(y_test, y_pred_median)
```

Appendix C

LSTM code

We also used neural networks for this study. We used LSTM which is a type of RNN that can capture long-term temporal dependencies. The code blocks for binary classification are given below.

The data preparation for LSTM is very similar with the exception of shaping the training data to sequential data for LSTM to process it and also setting up an initial bias for the first epoch.

```
# Apply SMOTE for data balancing
from imblearn.combine import SMOTEENN
from imblearn.over_sampling import BorderlineSMOTE
smote = SMOTEENN()
#X_train_scaled, y_train = smote.fit_resample(X_train_scaled, y_train)
#X_val_scaled, y_val = smote.fit_resample(X_val_scaled, y_val)
# Reshape data for LSTM
X_train_resampled = X_train_scaled.reshape((X_train_scaled.shape[0], 1,
                                            X_train_scaled.shape[1]))
X_test_resampled = X_test_scaled.reshape((X_test_scaled.shape[0], 1,
                                          X_test_scaled.shape[1]))
```

```
X_val_resampled = X_val_scaled.reshape((X_val_scaled.shape[0], 1,
                                        X_val_scaled.shape[1]))

# Define class labels (0 and 1 for binary classification)
class_labels = np.unique(y_train)

# Compute class weights
class_weights = compute_class_weight(class_weight="balanced", classes=
                                        class_labels, y=y_train)

# Convert to dictionary format
class_weights_dict = {i: weight for i, weight in enumerate(class_weights)}

from sklearn.utils.class_weight import compute_sample_weight

# Compute sample weights
sample_weights = compute_sample_weight(class_weight='balanced', y=y_train)
print(class_weights_dict)
print(sample_weights)
print(X_val_scaled.shape)
print(X_train_scaled.shape)



# Setting the initial bias
pos = np.sum(y_train == 1)  # Count of positive samples
neg = np.sum(y_train == 0)  # Count of negative samples

#initial_bias = np.log([pos / neg])
initial_bias = 0
print(initial_bias)
```

After preparing the data, we build the LSTM model as described above with all the evaluation metrics and the plots for the loss function, and the F1 score are also given in the code block.

```
# @title Building the Model St John's

from tensorflow.keras.optimizers import RMSprop, Nadam

# Define custom F1 Score metric
class F1Score(tf.keras.metrics.Metric):
    def __init__(self, name="f1_score", **kwargs):
        super(F1Score, self).__init__(name=name, **kwargs)
        self.precision = Precision()
        self.recall = Recall()

    def update_state(self, y_true, y_pred, sample_weight=None):
        self.precision.update_state(y_true, y_pred, sample_weight)
        self.recall.update_state(y_true, y_pred, sample_weight)

    def result(self):
        p = self.precision.result()
        r = self.recall.result()
        return 2 * (p * r) / (p + r + tf.keras.backend.epsilon())

    def reset_state(self):
        self.precision.reset_state()
        self.recall.reset_state()

# List of metrics including custom F1 score
METRICS = [
    keras.metrics.BinaryCrossentropy(name='cross_entropy'),
    keras.metrics.MeanSquaredError(name='Brier_score'),
```

```
    keras.metrics.TruePositives(name='tp'),
    keras.metrics.FalsePositives(name='fp'),
    keras.metrics.TrueNegatives(name='tn'),
    keras.metrics.FalseNegatives(name='fn'),
    keras.metrics.BinaryAccuracy(name='accuracy'),
    keras.metrics.Precision(name='precision'),
    keras.metrics.Recall(name='recall'),
    keras.metrics.AUC(name='auc'),
    keras.metrics.AUC(name='prc', curve='PR'),  # Precision-recall curve
    F1Score()
]



# Define LSTM layers
model = Sequential([
    Input(shape=(X_train_resampled.shape[1], X_train_resampled.shape[2])),
                                            # Input shape

    # First LSTM Layer
    Bidirectional(LSTM(64, activation='tanh', return_sequences=True)),  #
                                          tanh is default lstm
    BatchNormalization(),
    Dropout(0.2),

    # Second LSTM Layer
    Bidirectional(LSTM(32, activation='tanh', return_sequences=False)),
    BatchNormalization(),
    Dropout(0.2),

    # Fully Connected Layers
```

```
    Dense(16, activation=None),  # No activation, since we use LeakyReLU
    LeakyReLU(negative_slope=0.01),
    BatchNormalization(),
    Dropout(0.2),

    Dense(16, activation=None),
    LeakyReLU(negative_slope=0.01),
    BatchNormalization(),

    Dropout(0.2),

    # Output Layer
    Dense(1, activation='sigmoid', bias_initializer=Constant(initial_bias)
                                        )  # Binary classification
])

# Set the initial learning rate
learning_rate = 0.0001
optimizer = Adam(learning_rate=learning_rate)
#optimizer = RMSprop(learning_rate=learning_rate, rho=0.9, epsilon=1e-06)
#optimizer = Nadam(learning_rate=learning_rate)
# Compile the model
model.compile(optimizer= optimizer, loss='binary_crossentropy', metrics=
                                    METRICS) # [Precision(), Recall(),
                                    F1Score() ]

# Train the model
early_stopping = EarlyStopping(monitor='val_f1_score', patience=20,
                                    restore_best_weights=True, mode='max'
                                    )
reduce_lr = ReduceLROnPlateau(monitor='val_f1_score', factor=0.5, patience
```

```
                                        =5, min_lr=1e-7, mode='max')

history = model.fit(X_train_resampled, y_train, epochs=200, batch_size=
                                        128, validation_data= (
                                        X_val_resampled, y_val), callbacks=[
                                        early_stopping, reduce_lr],
                                        class_weight= class_weights_dict)  #,
                                         sample_weight = sample_weights

# Save the model
#model.save('lstm_onebinary.keras')

# Plotting the results
plt.figure(figsize=(14, 6))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['f1_score'], label='F1Score')
plt.plot(history.history['val_f1_score'], label='val_F1Score')
plt.xlabel('Epoch')
plt.ylabel('F1Score')
plt.legend()
plt.title('(a) Model F1-Score over Epochs')

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

```
plt.title('(b) Model Loss over Epochs')
plt.suptitle('LSTM Model Performance for St Johns', fontsize=16)
plt.show()


# amount of parameters
model.summary()
```