Weather-triggered Wireless Telemetry System

Design Document

sdmay25-18

Client: Daji Qiao

Advisor: Sarath Babu

Alex Chambers: Individual Component Designer

Alexander Christie: Client Interaction

Adam Fields: Data Formatting

Nisha Raj: Team Lead

Aidan Gull: Component Integration

Colin Kempf: Documentation

Team Email: sdmay25-18@iastate.edu

Team Website: https://sdmay25-18.sd.ece.iastate.edu/

Executive Summary

ARA is an advanced wireless research platform covering Iowa State University, Ames, and nearby rural areas. The team is tasked with creating a weather-triggered telemetry system that measures the performance of wireless technologies developed on ARA before, during, and after weather events of interest. The team will be utilizing multiple external forecast API's in order to automatically have the ARA equipment recognize when a weather event is going to occur. This system will eventually allow researchers to determine how the performance from the ARA framework differs during different weather events.

The current problem our team looked to solve was determining how weather events affect experiments conducted on the ARA testbed. Researchers want to know how the results of their experiments will be affected due to weather events occurring. The requirements of our design include utilizing forecast API's to predict future weather events, triggering our software to collect and store weather and wireless data 30 minutes before, during and after weather events. This data must then be stored in a ZIP file hierarchy, which then ARA researchers would be able to query for this data using a user-friendly interactive UI.

Our team created a design that includes using three unique forecast APIs that would predict when a weather event would occur, and then based on that information, we would trigger the wireless signal data collection and store that data in the ZIP hierarchy. Our team initially developed a Python program that queried these three APIs, pulling their forecasts for a future predicted time every hour. The team then visualized this data using graphs, comparing the different APIs against true ARA data. The graphs gave the team, along with the client, a visualization to see the predicted weather from the API compared to the actual weather measured at the ARA weather stations. This served as an accuracy test to see how well the weather prediction model would work in terms of predicting a weather event and then begin collecting wireless signal data.

From there, our group implemented our designs, using the prediction program which queried the three APIs to inform a second script which would gather the ARA weather and wireless data. A third script we created receives this data and formats it together. Lastly, we created a UI for a user to be able to get visualization of the collected data.

Learning Summary

Development Standards & Practices Used

- IEEE 1413-2010: IEEE Standard Framework for Reliability Prediction of Hardware
- IEEE 1063-2001: IEEE Standard for Software User Documentation
- IEEE 1012-2016: IEEE Standard for Software Verification and Validation
- ISO/IEC/IEEE 14764-2021: Standard for Software Engineering Maintenance
- IEEE 1448a-1996: Standard for Information Technology Life Cycle
- Software version control utilizing Git

Summary of Requirements

Functional

- Must trigger collection based on abnormal weather conditions
- Must trigger data collection when rain is detected within desired lead-in data collection time.
- Must trigger data collection when snow is detected within desired lead-in data collection time.
- Must trigger data collection when winds above 15 mph are detected within desired lead-in data collection time.
- The software shall have a specified lead-in time where data is being collected a specified amount of time before the weather event.
- The software shall have a specified lead-out time where data is being collected a specified amount of time after the weather event.
- The software shall use three weather forecasting APIs to predict when a weather event will occur.
- The software shall incorporate a UI that allows users to query weather events on specific days.
- Must utilize local weather data for validation.
- While the software detects there is a weather event it shall keep collecting data until the weather event or the lead-out time is complete.
- When the software first starts it shall query the ARA weather API and the weather forecasting APIs.
- When the software detects a weather event it shall begin lead-in data collection before a specified amount of time before the weather event.

- When data collection for a weather event is complete the software shall send data to be processed.
- If there is no weather event then the software shall wait for the next weather event prediction.
- If there is a weather event but not within the specified time to start lead-in time data collection then the software shall wait to begin collection until the specified lead-in time.
- The software shall store weather collection data in a ZIP file hierarchy.

Resource

- Uses the ARA framework to collect, store, and provide access to weather data.
- Shall have access to external weather forecasting APIs
- Weather data shall be stored in the storage space provided by the ARA framework
- Shall run on the server space provided by the ARA framework

Physical

- Must use ARA's Disdrometer when collecting live weather data from the ARA framework

Aesthetic

- The software shall utilize data visualization tools such as graphs and histograms
- The software shall graph the predicted weather at specified intervals and plot the difference between the predicted and actual weather.
- The graph shall utilize multiple colors to see the difference between actual and predicted weather.

User Experiential

- GUI shall be accessible from the ARA systems
- Shall be easy to use for someone who has experience running scripts using the command line.
- The GUI shall be intuitive to query data for users.
- The GUI shall be easy to navigate for new users on the first attempt of using it.

UI

- The UI shall execute queries in a timely manner such as depending on the size of the dataset under 2 seconds.
- The UI shall allow users to write queries or select pre-determined queries.

Applicable Courses from Iowa State University Curriculum

- COMS 2270 (Introduction to Object-Oriented Programming)
- COMS 2280 (Introduction to Data Structures)
- COMS 3090 (Software Development Practices)
- COMS 3110 (Introduction to Algorithms)
- CPRE 2810 (Digital Logic)
- CPRE 3080 (Operating Systems: Principles and Practice)
- SE 3170 (Introduction to Software Testing)

New Skills/Knowledge acquired that was not taught in courses

- Python scripting development
- Linux server experience
- Data parsing and formatting
- Model predictions

Table of Contents

List of figures/tables/symbols/definitions

Figures and Tables

Important Definitions and Terms

ARA: Acronym for Agriculture and Rural Communities is an at-scale platform for advanced wireless research across Iowa State University and the city of Ames spanning a rural diameter over 60km.

Disdrometer: Optical device situated on ground station platforms that measure precipitation.

GUI and UI: Graphical user interface and user interface which are both used interchangeably through the document. These are visual ways for users to interact with digital backend devices.

API: Acronym for application programming interface which in our project is used for external weather sources. This is a software intermediary that allows applications to communicate with each other.

ZIP File Structure: ARA dataset guidelines for organizing the data generated from weather and wireless signal experiments. Allow for a standard naming convention for experimental dataset across ARA.

1 Introduction

1.1 Problem Statement

Our clients are researchers for ARA (Agriculture and Rural Communities), an advanced wireless research platform in Ames, Iowa and the surrounding area. Part of their research involves collecting weather data from disdrometers set up at points throughout the region. These disdrometers can record data from all types of weather to be used for research. However, recording all data at all times is inefficient, taking up more space than needed, gathering data when weather conditions are normal. But how can the disdrometers know when a weather event is occurring to begin recording data?

This problem is what our group aims to solve. Our project is tasked with creating a system that will recognize and predict when a weather event is occurring. This trigger signals data collection before a given weather event has begun and allows us to continue collecting data until the weather event has passed. The collected weather data will eventually allow researchers to determine how the performance from the ARA framework differs during different weather events. The weather data will be able to be queried once gathered, allowing for easy visualization and analysis.

As our group works through this problem, we want to intelligently collect data on a wide range of network data during a variety of weather events. We also want to use forecast data to predict future weather events to gather data only when weather events we want to record are going to occur. Lastly, we need to store collected data and allow for user queries to access and format selected data.

Allowing for this data to be gathered and analyzed will allow for better research into ARA's primary mission; creating a wireless lab for the research and development of wireless technology that is both affordable and connective for rural communities and industries such as agriculture. Both internal researchers at ARA and external researchers from around the world will be able to utilize the data our project collects to further their studies and create new devices.

In order to predict the weather events, our project looks to weather forecasting APIs (Application Programming Interfaces). These will give our project data on weather events that are forecasted in the near future. But this can create an issue for us as we need access to other programs not owned by ARA. This is an important issue because without these forecast APIs, our predictions won't be as accurate. We want to have access to multiple forecast APIs so that we can rely on many predictions rather than just one. In order to address this issue we are requesting access to APIs we find that suit our data needs through keys.

The data from these external APIs can also prove problematic even once we have access to them. Just because they have the data we need doesn't mean that all of it is useful or formatted in a usable way. This is another important issue as we need to be able to read the data to inform our disdrometers the predicted time to begin recording the live weather data. To fix this, we are working to clean the data from the APIs as it is received, only keeping the important parts for prediction, and formatting it to a uniform standard. (Appendix 1)

1.2 Intended Users

1. Internal Users for ARA typically from Iowa State (All have same needs)

Examples: Professors, Graduate students, Undergraduate students

2. External Users of ARA platform (All have same needs)

   Examples: Outside Universities, Industry Professional, Individual Users

From speaking with our client we determined that there are only two users for our product. Our client described to us that users using our product will either be internal users or external users. They described that the only difference between the external and internal users would be that internal users would be from Iowa State institutions, however, these users will not have different needs from the external client. Our client described that the internal users will only have admin access to the product. The external users will be outside of Iowa State but this can include other universities, industry and even individual users. Through this process we determined that our product only has two categories of users: internal and external.

The product that we create will be used by internal ARA users which will include groups of people from Iowa State such as professors, graduate students, and undergraduate students. The product will also be used by external users who will have the same needs as the internal users. Some of the external users include outside university faculty, industry professionals, and individual users.

The people who will benefit from our product are the internal and external users since they will be able to learn when weather events are occurring and how those weather events affect the experiments they are running of the ARA framework. The internal users are local to Iowa State and some of their characteristics is they want to query for certain weather data that occurred on a specific day. They will want to do this in order to see how their experiments running on the ARA framework might have been affected by weather events. The external users will want to query for certain weather data that occurred on a specific day. They will want to do this in order to see how their experiments running on the ARA framework might have been affected by weather events. These characteristics are the same for both internal and external users.

The user needs for the internal users are the ARA researcher needs weather data to be collected when a weather event occurs because they want to collect, store, and publish this information.(Appendix 3) Another need is the researcher needs weather data because they want to analyze how the ARA equipment was affected by weather events. For external users their needs are similar; The researchers need a way to know when weather events occurred because they will analyze how the weather affected their test results on the ARA framework. (Appendix 2) Another need is the researchers want an easy way to query specific weather events data because they want to access weather event data efficiently.

The characteristics of each of our user groups are described in the empathy maps in the appendix below. However, a brief description and characteristics of our internal and external users are as follows. The internal users are any users that are internal to Iowa State University. These can include undergraduate and graduate students, professors, postdoc researchers as well. These researchers have various goals, some can include trying to advance 5G or massive MIMO technology. For example, our persona David is a 25 year old PhD student who is working on experimenting with ARA containers to provide high-speed connectivity in rural areas. Our external user persona, Bob, is a long-time researcher at Bell Laboratories. He recently started working on exploring 5G connectivity and he hopes to find large datasets online that will help his investigation on how weather events can affect connectivity.

The users will benefit from our product because they will have an easy and intuitive way to query weather data so they can easily identify when a weather event occurred so they know if their experiments were affected by weather events. Our problem statement includes storing and collecting data for users to query and access in a formatted way. The proposed benefits we have directly connect to our problem statement as users will have an easy and intuitive way to query data.

2 Requirements, Constraints, And Standards

2.1 Requirements and Constraints

Functional

- Must trigger data collection when rain is detected within desired lead-in data collection time.
- If average predicted precipitation probability is 80% or greater, begin gathering lead in data 30 minutes ahead of the predicted time.
- If average predicted precipitation probability is 25% or greater, begin gathering lead in data 30 * (Average Predicted Precipitation Probability / 100) minutes ahead of the predicted time.
- The software shall have a specified lead-in time where data is being collected a specified amount of time before the weather event.
- The software shall have a specified lead-out time where data is being collected a specified amount of time after the weather event.
- The software shall use three weather forecasting APIs to predict when a weather event will occur.
- The software shall incorporate a UI that allows users to query weather events on specific days.
- Must utilize local weather data for validation.
- While the software detects there is a weather event it shall keep collecting data until the weather event or the lead-out time is complete.

- When the software first starts it shall query the ARA weather API and the weather forecasting APIs.
- When the software detects a weather event it shall begin lead-in data collection before a specified amount of time before the weather event.
- When data collection for a weather event is complete the software shall send data to be processed.
- If there is no weather event then the software shall wait for the next weather event prediction.
- If there is a weather event but not within the specified time to start lead-in time data collection then the software shall wait to begin collection until the specified lead-in time.
- The software shall store weather collection data in a ZIP file hierarchy.

Resource

- Uses the ARA framework to collect, store, and provide access to weather data.
- Shall have access to external weather forecasting APIs
- Weather data shall be stored in the storage space provided by the ARA framework
- Shall run on the server space provided by the ARA framework

Physical

- Must use ARA's Disdrometer when collecting live weather data from the ARA framework

Aesthetic

- The software shall utilize data visualization tools such as graphs and histograms
- The software shall graph the predicted weather at specified intervals and plot the difference between the predicted and actual weather.
- The graph shall utilize multiple colors to see the difference between actual and predicted weather.

User Experiential

- GUI shall be accessible from the ARA systems
- Shall be easy to use for someone who has experience running scripts using the command line.
- The GUI shall be intuitive to query data for users.
- The GUI shall be easy to navigate for new users on the first attempt of using it.

UI

- The UI shall execute queries in a timely manner such as depending on the size of the dataset under 2 seconds.

- The UI shall allow users to write queries or select pre-determined queries.

2.2 Engineering Standards

Engineering standards are important since they often provide a guideline for different processes and components of a project. These guidelines are commonly used in a variety of projects which makes every project that follows these standards easier to understand. It also makes the process of creating a project easier since anyone making a project can find various standards that have been set by other projects and apply them to their own.

The first engineering standard that we had picked was standard 1413 IEEE Standard Framework for Reliability Prediction of Hardware. This standard goes over a framework for a reliability prediction of hardware. It specifies any required elements needed to form a reliability prediction as well as information needed to determine if collected data meets requirements. Each reliability prediction needs to have a specified list of inputs, assumptions, data sources, methodologies, and uncertainties. All of these are used when calculating risks when we decide to use the results gathered from our hardware. This standard does not provide any way to evaluate various methodologies the group might pick when deciding which one is best for our reliability prediction. The standard also does not provide any instructions on how to perform the reliability prediction. The standard's goal is to allow an ease of defining and creating reliability predictions for hardware so that development teams and users of the application can be assured that the data gathered by hardware components meets the acceptable ranges set in the requirements.

The second engineering standard that we had picked was standard 1063 IEEE Standard for Software User Documentation. This standard details the minimum requirements for each format, structure, and content for various pieces of user documentation. It states that each of these minimum requirements are important for user documentation since many software applications can be made far more difficult to use if the documentation surrounding them are unclear, disorganized, and difficult to read. The standard breaks down each of the minimum requirements into their own sections while stating that the order of each section does not imply a correct step by step process to follow. For the format of each document should remain consistent throughout not only one document but all associated with the project. Each document should also have a file format that is available for most modern computers and have the ability to be modified for users with special accommodations. For the structure, each document should be broken down and divided into topics. Each section should only contain relevant information regarding the main topic with some reference to other topics covered by the document. Lastly the standard covers how the content in the user documents should be laid out in a way that facilitates an easy understanding of topics covered in user documents. It goes into more detail about how the content should be as accurate as possible without being too full of technical definitions that would make the content more difficult to understand. This standard's goal is to provide a baseline for how any documents created for a project should be made. This baseline can be used to evaluate different documents created for this project.

The third engineering standard that we had chosen was standard 1012 IEEE Standard for Software Verification and Validation. This standard goes over the software validation and verification process. These processes help determine whether a piece of software satisfies requirements and user needs. The standard lists four different levels called software integrity levels that help quantify if the software meets requirements. These levels are in decoding order, high, major, moderate, and low. These processes may have different steps such as analysis, evaluation, testing, review, and assessment. The standard also states that any evaluations or assessments of software must be done within the context of the system which includes any hardware, users, environment, and interface software associated with the tested software. The goal of this standard is to provide a common framework for validation and verification of pieces of software. It also wants to define what the validation and verification process is and the various parts that go into the validation and verification process.

The first standard we had picked is relevant to our project. Since our project uses many different hardware components to measure and record data points we need to ensure that each one of these components is measuring and recording data points correctly. The standard goes over a process on each item that is picked for the reliability prediction. The first part of this process is to define any elements needed for the reliability prediction. This includes a description of the item, intended prediction results, methodologies used for the prediction, inputs for the prediction, metrics used, and any uncertainties that might be in the prediction. Each one of these steps can be done on the hardware components we are going to use. These pieces of hardware are WS100 Radar Precipitation Sensor/Smart Disdrometer which is located at ARA's Agronomy Hall site. The other site has a different distometer, OTT Parsivel² Disdrometer that is located at Wilson Hall. Each of these sites also has a weather station that provides other measuring tools.

The second standard covers user documentation standards. ARA already has various user manuals and documents that would need to be updated with our new features. We would need to be confident that our changes to ARA's user documents match with previously set formatting and structural design decisions. Currently our plan is to create an API that can predict weather events so we can begin collecting data points during those weather events. The details of this API will need to be included in ARA's user manual under the ARA API section. In this section instructions on how to use the API will be included as well as details such as the formatting of data, parameters, and outputs for the API. All of this will be structured similarly to that of ARA's current API documents. Based on the many changes that will need to be made to the ARA user documentation this standard is very relevant to our project since it details how user documentation should be structured, formatted, and how the content is presented in each document.

The third standard details the process for validation and verification. Our project has many requirements that are specified above in the requirements section and our API needs to satisfy them. In order to determine if our developed pieces of software meet specific requirements we need some process that can be used as an evaluation tool. This standard provides us with such a process. It helps us figure out how and where to test our API's components and to what degree do those components meet requirements. Due to all of the helpful planning and evaluation information in the standard it is important for our project.

The standards picked out will only make small changes to our project. They are mostly providing some structure and guidance on how we should go about certain parts of our project. Each of the standards is adding something new that we need to the project but they are not necessarily changing our current approaches to developing the API. The first standard will require us to add some kind of hardware testing to the project to ensure that the hardware components are gathering data points correctly. The second standard will give us a guideline on how we should go about updating the user documentation on the ARA user manual. The third will be adding a verification and validation process to the end of our development cycles for each piece of software to make sure that the software satisfies the requirements for the project.

3 Project Plan

3.1 Project Management/Tracking Procedures

For this project we decided to adopt an agile project management style. The reason our team picked this project management style is that it followed our structure for weekly meetings. Every week (when possible) we met with our client and advisor to discuss what we had accomplished and what our plans were for the coming week. Due to this meeting structure it made sense to use an iterative development style like agile. Our team's progress was tracked using Github where we list out tasks that need to be completed by the end of the next week. Our team also used Discord for communication and a shared Google Drive for our documentation.

3.2 Task Decomposition

First Semester

The tasks for the first semester focus more so on the planning of the project which is reflected in the tasks that were completed.

- Project Charter

- Conceptual Design
- Initial Research
- ARA Experiments
- Formulating Users
- Creating Requirements
- Finalize Requirements and Users
- Task Decomposition
- Detailed Design
- Weather Forecast API Prototype
- Analyze Weather API Data
- Forecast Data Visualization
- Weather Prediction Prototype

Second Semester

For the second semester we took our component design and divided the individual actions into tasks that could be worked on congruently. From this we developed a Task Decomposition Diagram to visualize the individual tasks that we needed to complete. From our diagram (shown below) we determined there were three major tasks that needed to be completed; API Prediction Metrics, ARA Live Data Collection, and Data Formatting. The latter already serves as a fairly disconnected component to the overall system, making it a clear candidate for a task. The other two were determined by analyzing the *Prediction and Data Gathering* component of our design and recognizing the unique logic that separates Prediction (outlined in Red) and Data Gathering (outlined in Purple). As a team we decided to dedicate these tasks to specific teams composed of two members to ensure project development occurs concurrently. Additionally, a final task was determined that falls outside the scope of the previous three tasks. This final task's focus is on documentation, testing, and GUI development. Due to the breadth of its scope this task was treated as a team task that was handled after the implementation of the individual components.

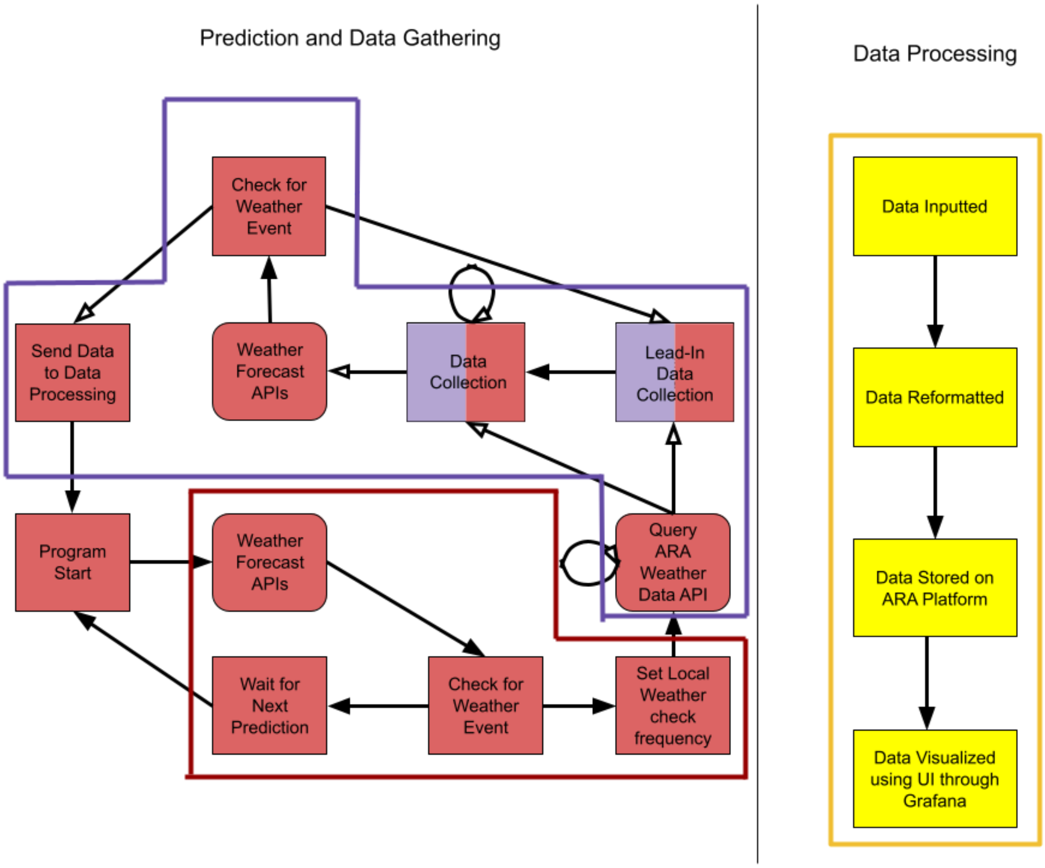


Figure 3.2.1: Task Decomposition Diagram

API Prediction Metrics:

- This task involved utilizing the three APIs we selected (OpenMeteo, Tomorrow Weather, and Weather.gov) to predict the precipitation probability and when certain weather events will occur. This functionality could be broken down into three minor subtasks; Gather Prediction Data, Calculate Predicted Weather Event, and Send Trigger to Collection component.

  Gathering the Prediction Data required the development of a python script that would query each API once an hour for the predicted precipitation within the next hour. Each request is unique to the individual APIs. Once the percentage of precipitation is collected, the script should calculate the overall odds of precipitation based on the collective information pulled from the APIs. From this, lead-in time is calculated to ensure that higher chances of rain have longer lead-in times. Finally, the system should send this information to our Data Collection component.

ARA Live Data Collection:

- The focus of this task is to utilize data collection devices (such as disdrometer and COTS) to collect live data as a weather event occurs. Much of the time spent on this task was working out how the communication channels between our script and the data collection devices should be organized. Additionally, the script was specifically designed to begin collecting lead-in data based on the API prediction while also starting collection independently if a local weather event was detected live. From this we determined four minor subtasks needed to complete this work; Collect Live ARA Data, Trigger if local event detected, Lead-in recording based on Prediction trigger, Validate Predictions.

  The Live Data Collection section of our script would be triggered either from the API predictions or by detecting independently that a weather event (that hadn't been predicted) was actively occurring. This functionality was developed hand-in-hand with the actual collection of the live data pulled from the disdrometer and COTS. Lead-in time based on the trigger from the API predictions was then added to the script. Finally, with all of this developed effort was placed in ensuring that the API predictions were accurate in predicting future weather events.

Data Formatting:

- Data Formatting exists as more of a standalone task than the other two. While API Predictions and Live Data Collection are fairly intertwined in their functionality, Data Formatting runs almost independently of the two. The main task is to format and store the collected live data in a .Zip hierarchy while updating a local database with the file names and paths. This task was subdivided into three minor subtasks; Access the ARA data from the interval collections, Format Wireless and Weather data in .Zip hierarchy, and Update database with new data information.

  The first subtask required the team to access the data created by the Live Data Collection script. This task innately requires intense communication and a focus of vision to ensure that data is presented in an agreed upon format. Once the data is accessed, it is formatted into two CSVs (one for disdrometer data and one for COTS) before being compressed into a .Zip file along with a readme. The next task was to update the server's MariaDB with the proper information including the relative path of the .Zip, location of the data collected (such as Wilson Hall), and the start date.

Concluding Subtasks:

- Our concluding subtasks include all additional tasks that don't fit within the scope of the previously listed tasks. These include documentation of our process, testing, validation, and GUI development.

  Documentation is a formality, one that the functionality of the project itself does not depend on. The testing and validation tasks feel as though they are complementary to one another, and were done concurrently at the end of the semester. Our GUI development relied heavily on the implementation of the Data Formatting task, taking the .Zip files and visualizing the data using a local instance of Grafana. The GUI subtask saw us creating a web application to visualize the selection of a given dataset before graphically displaying it on the ARA server's local instance of Grafana.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Predict Weather Data

- **Milestone:** Script that uses multiple APIs to predict precipitation probability and then find the average over the probabilities and send a trigger to begin ARA data collection based on the precipitation probability values.

- Metric 1: The system should collect API Forecast Data from each API consistently.
- Metric 2: The system should consistently send a trigger to data collection whose value is dependent on the average forecast precipitation prediction.

Data Collection

- **Milestone:** Script to inform ARA network to begin live data collecting when an event is detected from predict weather data script.

- Metric 1: The system should be able to collect data from the ARA network, both weather collection devices and wireless collection devices, at a specific time.
- Metric 2: The system should be able to identify when the API prediction script misses a weather event and begin to predict on its own if needed.
- Metric 3: The system should be able to determine the end of an occurring weather event in order to stop collecting data for the event.
- Metric 4: Upon completion of a weather event, the system should be able to send the collected data to the data formatting script

Data Formatting

- **Milestone:** Format and store collected live data utilizing a ZIP hierarchy.

- Metric 1: The system should take collected weather and wireless data from intervals of collecting time and format them for storage
- Metric 2: The system should utilize a ZIP hierarchy

User Interface

- **Milestone:** User Interface which allows a user to be able to access and display gathered data.

- Metric 1: The UI should be usable to new users after reading a user manual/ guide.
- Metric 2: The UI displays the gathered data to the user.
- Metric 3: The accuracy of the UI is 100% for the data that should be displayed.

Data Processing

- **Milestone:** Initial Prototype to feed in some sample test datasets.

- Metric: The program formats the sample datasets correctly.

ARA Platform Querying

- **Milestone:** Initial Prototype to query sample datasets.

- Metric 1: We are able to query the data from the backend correctly.
- Metric 2: Any user is able to query the data using our GUI.

Program Overhead

- **Milestone:** Full Prototype with Collection Loop and Data Parsing.

- Metric: The program runs continuously with no issue.

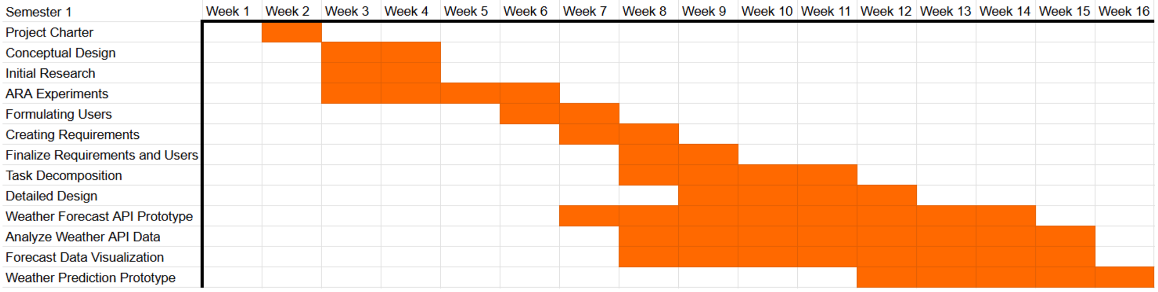
Final Deliverable

- **Milestone:** Integrating all parts of the code to predict, gather and store data.

- Metric 1: The accuracy of the model to identify actual weather events. The accuracy of such should identify all actual weather events.
- Metric 2: The accuracy of the model to collect and timestamp all relative data. The model shall accurately collect and store data 100% of the time.
- Metric 3: The program runs continuously with no issue.
- Metric 4: The program formats data 100% correctly.
- Metric 5: A user is able to query the data using our GUI.
- Metric 6: The final product is completely ready (meets all metrics) by the demo.

3.4 Project Timeline/Schedule

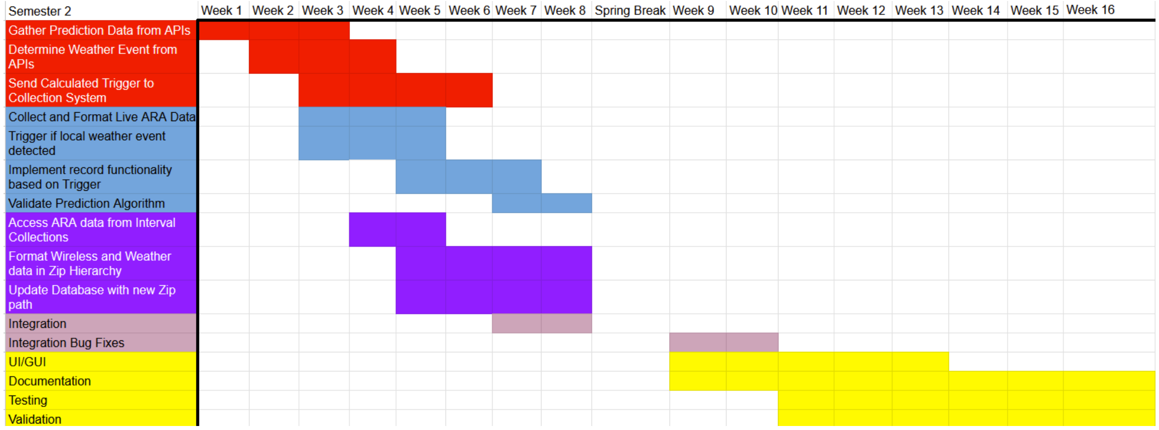
First Semester



Semester 1	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16
Project Charter		■														
Conceptual Design			■	■												
Initial Research			■	■												
ARA Experiments			■	■	■	■										
Formulating Users						■	■									
Creating Requirements							■	■								
Finalize Requirements and Users								■	■							
Task Decomposition								■	■	■	■					
Detailed Design									■	■	■	■				
Weather Forecast API Prototype							■	■	■	■	■	■	■	■		
Analyze Weather API Data								■	■	■	■	■	■	■	■	
Forecast Data Visualization								■	■	■	■	■	■	■	■	
Weather Prediction Prototype												■	■	■	■	■

Figure 3.4.1: First Semester Schedule

Second Semester



Semester 2	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Spring Break	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16
Gather Prediction Data from APIs	■	■	■														
Determine Weather Event from APIs		■	■	■													
Send Calculated Trigger to Collection System			■	■	■	■											
Collect and Format Live ARA Data			■	■	■												
Trigger if local weather event detected			■	■	■												
Implement record functionality based on Trigger					■	■	■										
Validate Prediction Algorithm							■	■									
Access ARA data from Interval Collections				■	■												
Format Wireless and Weather data in Zip Hierarchy					■	■	■	■									
Update Database with new Zip path					■	■	■	■									
Integration							■	■									
Integration Bug Fixes										■	■						
UI/GUI										■	■	■	■	■			
Documentation										■	■	■	■	■	■	■	■
Testing												■	■	■	■	■	■
Validation												■	■	■	■	■	■

Figure 3.4.2: Second Semester Schedule

3.5 Risks And Risk Management/Mitigation

Consolidate Weather Data

When it comes to consolidating weather data, there are 3 main risks that we have considered:

The first risk we considered was if the external API would go down for extended time. We estimate that there is a probability of 0.10, but the severity of risk is high. We are mitigating this risk by utilizing multiple APIs. We mitigate the severity of this risk as a single API becoming unavailable will not break the system.

The second risk we considered was if an external API call fails on rare occasions. We estimate that there is a probability of 0.05 and the severity of risk is moderate. We are mitigating this risk by utilizing multiple APIs. We use separate calls in try-catch blocks to ensure that there are no problems even if an API call fails on rare occasions.

The third risk we considered was if an external API changes data output format. We estimate that there is a probability of 0.2, but the severity risk is low. We are mitigating this risk by monitoring the API websites that we are using and staying up-to-date on any updates from those sites regarding their data formatting and any changes they make.

Consume and Record Active Weather Data

When it comes to consuming and recording active weather data, there are 2 main risks that we considered:

The first risk we considered was if we were unable to collect data from an ARA Weather Station. We estimate that it has a probability of 0.1 and the severity of risk is moderate. Because this project is intended to be on the ARA Framework, any issues with the weather stations not outputting data would be an issue across most of the ARA Framework. This issue would be mediated by the ARA admins and requires no mitigation plan from our team.

The second risk we considered was if ARA changes their data formatting. We estimate that this risk has a probability of 0.2 and the severity of risk is low. Since we are directly working with the researchers who control the ARA equipment, our plan is to make them aware of how we are using the data currently and inform them how a change in formatting can affect our model.

Program Overhead

When it comes to our program overhead, we considered one main risk. Specifically, if the ARA server goes down. We estimate that this risk has a probability of 0.1 and a high severity. In terms of our code, there will always be a copy of the code we are running on our GitHub. In terms of the program itself going down and not being able to predict weather events, we will have to temporarily pause weather event data collection in order to get the server up and running again. Another solution could be to request an additional server and have scripts running in parallel on both the servers.

Data Processing

When it comes to data processing, we considered one main risk. Specifically, if the data corrupts or the data produced is false. We estimate a probability of 0.3 and the severity is moderate. We have a check in the data processing script that will check that the data has not been corrupted and is in the correct expected format.

ARA Platform Querying

When it comes to ARA platform querying, we considered two main risks:

The first risk we considered was if the GUI fails to visualize the data. We estimate that it has a probability of 0.3 and a moderate severity. The team will need to get user validation on if the data was correctly visualized in graphical form. If the data was not correctly visualized then there will be a button for the user to regenerate the graphical visualizations.

The second risk we considered was if the GUI fails to pull correct data. We estimate that the probability of this is 0.1 and a low severity. The team plans to implement multiple tests in order to prevent this failure from occurring. However, if the GUI did fail to pull correct data then the user can have the option to report this issue to the developers which is the team.

Data Collection, Storage and Hardware

When it comes to Data Collection and Storage, we considered three main risks:

The first risk we considered was taking up too much storage space. We estimate that it has a probability of 0.9 with a low severity. This will almost always be a concern, we can mitigate this by decreasing the lead-in time and preventing the system from gathering unnecessary data, generally by having accurate predictions.

The second risk we considered was collecting inaccurate data. We estimate that it has a probability of 0.2 with a moderate severity. The team mitigates this by using local devices to measure weather data and incorporating methods for false positives and negatives.

The third risk we considered was creating hardware malfunctions. We estimate that it has a probability of 0.01 with a high severity. The team mitigates this by ensuring our program has been thoroughly tested before merging it with the ARA servers and currently existing applications on their hardware.

-

3.6 Personnel Effort Requirements

Tasks	Description	Hours
1st Semester		
Project Charter	Begin learning about our project and exploring ideas.	2
Conceptual Design	Start brainstorming initial ideas based on client description.	6
Initial Research	Research ideas on how to begin implementing initial parts of our project.	4
ARA Experiments	Start experimenting with the ARA framework and get familiar with the	4

	reserving resources.	
Formulating Users	Work with client to identify potential future users of our project.	2
Creating Requirements	Based on the users of the project identify requirements needed for the project.	4
Finalize Requirements and Users	Confirm with the client the requirements needed to meet the user's needs.	2
Task Decomposition	Breakdown the different tasks needed to complete the project and how all the components fit together.	6
Detailed Design	Create component design of predicting and gathering weather data and breaking down the chart based on different phases.	10
Weather Forecast API Prototype	Begin creating a prototype that uses external APIs to predict and gather weather event data.	12
Forecast Data Visualization	Use the weather forecast data output and create graphs to visualize the difference in predicted weather forecast at a certain time out.	6
Weather Prediction Prototype	Create a prototype that will use external APIs to predict when a weather event will	20

	occur.	
2nd Semester		
Gather Prediction Data from APIs	Call APIs to gather data to determine when a weather event will next occur	2
Determine Weather Event from APIs	Decide based on the prediction data whether or not there is a weather event. Determine a time to start collecting if there is one.	6
Send Calculated Trigger to Collection System	When the designated time arrives, send a trigger to start collecting data.	9
Collect and Format Live ARA Data	Collect data and send it to the formatting module when it is done.	5
Trigger if Local Weather Event Detected	This is a backup system. If we fail to determine a weather event based on prediction data, but a weather event is happening we can start data collection immediately.	2
Implement Record Functionality Based on Trigger	This is the system for collecting data.	5
Validate Prediction Algorithm	Making sure that the prediction algorithm triggers collection at the correct time.	8
Access ARA Data from	Take the collected data and	4

Interval Collections	be able to manipulate the files.	
Format Weather and Wireless Data in Zip Hierarchy	Take the files and rename them to match ARA standards. Then place them in folders whose names also match ARA standards. Lastly, zip it and place it in a designated folder on the ARA server.	8
Update Database with New Zip Paths	Use MariaDB to store the file paths for the created zip files.	6
Integration	Combine prediction and collection scripts and eventually the formatting scripts.	8
Integration Bug Fixing	Takes place over time to find any remaining bugs.	20
UI/GUI	Create the GUI that users will use to query the data on the ARA platform.	6
Documentation	This. Literally this. Like what I am doing right now.	15
Testing	Making sure all outstanding bugs are squashed.	20
Validation	Verifying that the final product meets all requirements and client expectations.	17

Table 1: Personnel Hours Breakdown

Tasks	**Hours**	**Significant Mismatch?**
1st Semester		
Project Charter	2	No
Conceptual Design	6	No
Initial Research	4	No
ARA Experiments	4	No
Formulating Users	2	No
Creating Requirements	4	No
Finalize Requirements and Users	2	No
Task Decomposition	6	No
Detailed Design	10	No
Weather Forecast API Prototype	12	No
Forecast Data Visualization	6	No
Weather Prediction Prototype	20	No
2nd Semester		
Gather Prediction Data from APIs	4	No
Determine Weather Event from APIs	9	No
Send Calculated Trigger to Collection System	12	No
Collect and Format Live ARA Data	9	No

Trigger if Local Weather Event Detected	6	No
Implement Record Functionality Based on Trigger	9	No
Validate Prediction Algorithm	6	No
Access ARA Data from Interval Collections	6	No
Format Weather and Wireless Data in Zip Hierarchy	12	No
Update Database with New Zip Paths	12	Yes, we ended up having a lot of issues related to creating and filling the database.
Integration	8	No
Integration Bug Fixing	6	Yes, the transition ended up being smoother than we expected.
UI/GUI	15	Yes, Grafana ended up being tedious to work with.
Documentation	16	No
Testing	21	No
Validation	21	No

Table 2: Personnel Hours Actual Breakdown

4 Design

4.1 Design Context

4.1.1 Broader Context

The broader context of our design problem involves the main focus of ARA's goals. ARA focuses on research for communicating in rural communities. Our project is specifically involved with research into wireless signal strength. Our designs will help the ARA researchers better understand what weather conditions could worsen the wireless communication to rural areas. Not only does this affect the ARA researchers or external researchers who use our project, but it could also affect people living in rural areas as researchers use this data to improve communication for their communities. Through this work, our project addresses the societal need for better, faster communication, even in areas of the United States where technology is less prevalent.

Relevant considerations in areas of importance, related to our project:

Area	Description
Public Health, safety, and welfare	The project research will lead to a better understanding of how weather affects wireless data. In turn, ARA can discover ways to help keep internet uptime at a maximum.
Global, cultural, and social	Our project reflects the ideals, values, goals, and practices of the ARA community and workspace. Our project helps further their research and achieve their goals. Our project also affects rural communities through ARA. Our project looks to reflect their needs by assisting in the creation of better wireless communication for those communities.
Environmental	The software we are developing should have essentially zero impact on the environment.
Economic	The only costs would be for ARA to maintain and manage the data storage after we have completed the project. Otherwise, it is of no cost to us or other stakeholders.

Table 3: Areas of Ethical Concern

4.1.2 Prior Work/Solutions

According to our thorough product research, there is weather forecasting software out there. In fact, in the previous iteration of our design, we are using several APIs to predict when weather events would be happening. Specifically, we are using Tomorrow.io API [1], Open-Meteo API [2], and National Weather Service API [3] for this purpose. We are using a feedback loop [4] of data to help determine weather events.

4.1.3 Technical Complexity

Our project is of sufficient technical complexity since the design consists of multiple components and subsystems that each utilize distinct scientific and engineering principles. Our design requires the use of external APIs to predict weather events and the use of weather stations to collect data. The design process has been quite challenging and requires scientific accuracy to determine what is considered inclement weather and not simply a muggy day. We also need to be accurate in our software to make sure that it is precise and not buggy.

Our project also matches and exceeds the technical complexity of other projects in the market. Our design features the use of weather stations to determine when inclement weather occurs and collects wireless signal data during those weather events. In our thorough product research, there has not been anything that matches our project in terms of what we are trying to do. Overall, our design meets the technical complexity requirements.

4.2 Design Exploration

4.2.1 Design Decisions

Key Design Decisions:

1. We will use multiple forecasting API's to predict future weather conditions in Ames, Iowa. This is important to the success of our project because by using these data points, we can determined when to start collecting data in advance in order to collect data during any performance changes

2. We will only be predicting when the next weather event occurs at any point in time. This is important because it determines how far ahead with the forecast data we need to look. Additionally, it simplifies the logic for when to start and stop data collection

3. We will use ARA operated weather devices to measure current weather conditions. This is important to the success of our project as it will be how we

validate our forecasting algorithm to determine when we should collect wireless data.

4.2.2 Ideation

When it comes to the first design decision we made it made sense to have multiple forecast API's. This was done due to the variety of the data samples that each API provided. We also found that many of the API's used recorded some data points with better accuracy compared to other APIs. The end goal is to use all of the APIs to have a more accurate prediction model for when we want to start the collection process of performance data for wireless data. Here are 5 API's that we had looked at,

- Open-Meteo
- Tomorrow Weather API
- National Weather Service
- WeatherBit
- AccuWeather

Currently we are using 3 API's in our final project, Open-Meteo, Tomorrow Weather API, and National Weather Service. How we determined these 3 is that we had prioritized free API's and ones that we can get data frequently. Accuracy was also a large part of determining which API's that we wanted in our project. Below is a list of pros and cons for each one of these API's.

4.2.3 Decision-Making and Trade-Off

We discussed as a group what the best decisions would benefit us the most throughout our project. We were primarily concerned about gathering accurate forecast data from forecast APIs, so we can predict weather events. We looked at many different APIs and weighed the options about what benefits and drawbacks they would each have. There were 5 primary APIs that we considered. Through our analysis and testing, we narrowed the list down to 3 APIs (Open-Meteo, National Weather Service, and Tomorrow). After making a decision as a group, we presented our findings with our client and advisor, and they approved our decisions.

API	**Pros**	**Cons**
Open-Meteo	● No API key ● 10000 requests per day ● Pulling weather data from NOAA ● Hourly weather data for a 7 day forecast ● Weather models are updated every hour	● When tried the API for Ames it didn't seem to be that accurate but could also be user error

	● Incorporate real time measurements from airplane data, buoys, rain radar and satellite observations ● Historical weather data from the past 80 years ● Provide predictions for Temperature, Clouds, Rain, Solar radiation, Winds at higher altitudes, Transpiration, and Air quality ● Geocoding API → find coordinates of locations ● Seems relatively easy to use and integrate ● API can respond with a chart of whatever data points user wants plotted	● Not as intuitive as the other APIs ● Outputs a lot of information that will need to be parsed into more readable format to make decisions from
Tomorrow Weather API	● Free version ● Provides hyper localized weather data ○ Can deliver weather data that's very localized, down to specific city blocks ● Provides real time updates ● Advanced weather layers ○ Can choose between different parameters to include in your forecast such as precipitation, temperature, humidity, wind speed, cloud cover ● Integrates multiple weather models ○ Uses weather prediction models from NOAA ● Can get access to historical weather data ○ Can analyze the trends and tailor our application to specific weather events ● Well documented on how to integrate the API with projects ● Has 60 different weather fields to choose from	● Output of the API is a lot of information ○ Looks like it needs some work to be parsed and formatted correctly ● Allows for 25 requests an hour ○ Returns too many calls error until next hour ● Requires an API key ○ Need to generate a new key every hour

	● Easy to personalize the location to take weather from	
National Weather Service	● Completely free and open source ○ Everything found on github ● Retrieve data for a location using latitude and longitude ● Data available ○ Temperature, Dewpoint, maxTemp, minTemp, Relative humidity, Feel like temperature, Heat index, Wind chill, Sky cover, Wind direction, Wind speed, Wind gust, Weather conditions array, Hazards, Probability of precipitation, and Quantitative precipitation ● Has even more weather data layers such as snowfall amount and ice accumulation ● grid points endpoint fives access to raw numerical data ○ Formatted in JSON document ○ Each data point has valid time which is the interval that the value applies to ○ 2019-07-04T18:00:00+00:00/PT3H ■ Interval starting on 7/4/19 at 1800 and going for 3 hours ○ Value which is the actual value that applies in the valid Time interval ○ Include last-modified header ● Caching the result of the points requests to avoid having to do same request multiple time	● The JSON format seems to be a little nicer to read and parse ● Need to do 2 API requests to get the weather data ○ 1 for location ○ Another for the actual data user wants ● User's have said the cache part is a little complicated ● The model grid point also seems a little complicated, need to first get the grid point then will create the table of weather data ● Not super reliable, goes down often ○ Was down at the beginning of this year in January ● Integration is not as smooth

WeatherBit	● Has multiple API endpoints ○ Current weather ○ Severe weather ○ Forecasts ○ Historical weather ○ Ag-weather ○ Air quality ● Has many methods to look up weather ○ latitude/longitude ○ City name ○ Weather station ID ○ Zip code ○ City id ● Seems to have good documentation for integration ● Uses swagger UI ● Precipitation forecast returns 1 minutes interval forecast for precipitation rate and snowfall rate	● Doesn’t provide an individual endpoint for wind speed ○ Combines it with cooling and heating degree day information ● Requires an API ● Need a subscription ● Only 50 requests per day with free version ● Lighting data, climate normals, air quality, degree days not included in free version ● Can only have 1 API key with free version ● Standard plan is $40 a month ● Accuracy depends on location ● Semmes to be pulling weather from nearest weather station
AccuWeather	● There is a key for specifically getting forecast data every hour ● Has great documentation with a lot of information about what the requests will give and how we can access the information	● Needs an account. Free accounts can only have one API key ● Free accounts only get 50 calls a day, which could be a problem eventually ● Keys get the weather forecast for a specific area. If we want to get the forecast data

		for somewhere outside of Ames down the line, we would need additional API keys for those areas requiring a paid account or multiple accounts.

Table 4: Analysis of Implementation

After discussing our options with our client and advisor, they were interested in us pursuing using APIs to predict upcoming events or gathering all data for periods at a time and deleting the unneeded data. We first did research into possible APIs we could use that would gather the forecast data we would need and would be free for us to use. The APIs we decided to pursue were Mateo, Weather.gov, and Tomorrow API. We create a basic implementation (described further in section 6) that would gather the forecast predictions from those APIs. We then compared those predictions to actual historical data for those predicted times. Overall, we found that these forecast predictions could never meet the desired accuracy. Additionally, the APIs couldn't gather specific enough forecast data for the locations where the weather would be recorded, instead encompassing a larger area. This led us to decide to go with the other option, gathering all data for set periods of time and deleting the unneeded data.

4.3 Final Design

4.3.1 Overview

Our final design focuses on using APIs to gather forecast data, and use this data to inform when to begin gathering live weather data during weather events. The state diagram below gives a clear visualization of the phase our design will go through as it gathers, predicts, and collects data. It also takes into account certain outlier cases such as back to back weather events.

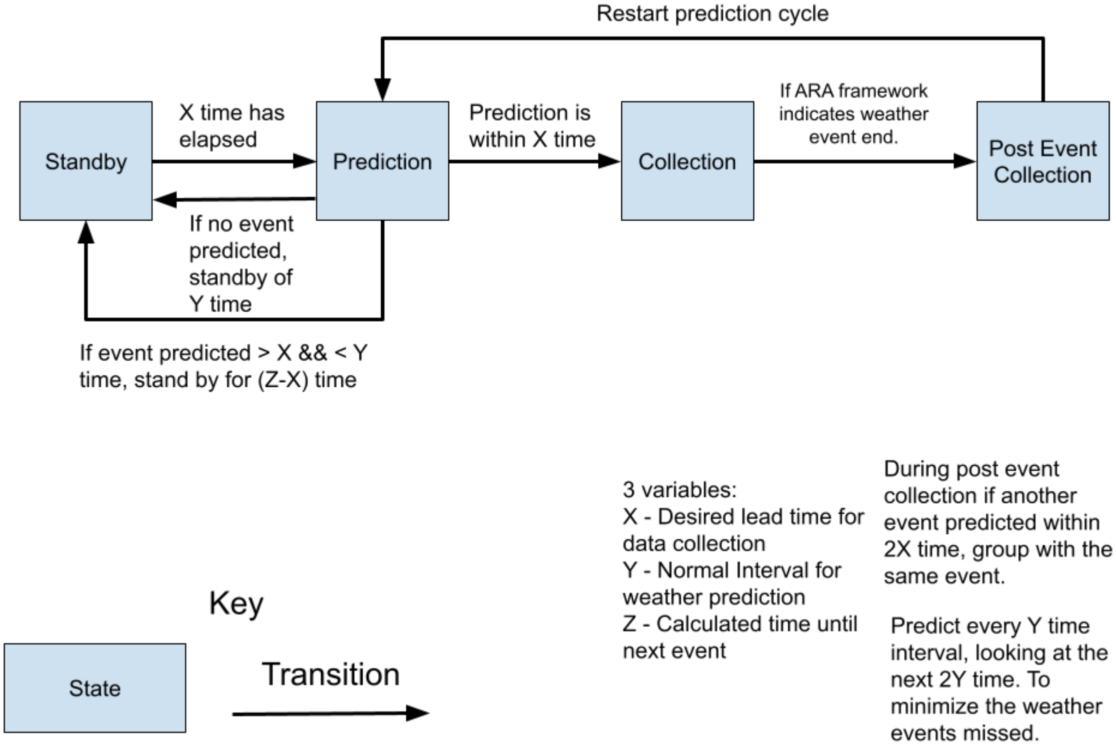

Restart prediction cycle
Standby
X time has elapsed
Prediction
Prediction is within X time
Collection
If ARA framework indicates weather event end.
Post Event Collection
If no event predicted, standby of Y time
If event predicted > X && < Y time, stand by for (Z-X) time
3 variables:
X - Desired lead time for data collection
Y - Normal Interval for weather prediction
Z - Calculated time until next event
During post event collection if another event predicted within 2X time, group with the same event.
Predict every Y time interval, looking at the next 2Y time. To minimize the weather events missed.
Key
State
Transition

Figure 4.3.1: State Diagram

The state diagram shows the different states of our system relative to our scheduling system. The four main states of our software are:

1. Standby: The passive state of the software.
2. Prediction: Where the software uses forecast API to look for future weather events.
3. Collection: The state the software is in when it gets the live weather data from the ARA Framework for the lead-in time before and during a weather event.
4. Post Event Collection: Where the software gathers data for the lead-out time, configures the collected data from a weather event and stores it in a designated location.

There are also three different variables which are used when transitioning from one state to another. These are:

1. X: The desired time before a weather event (lead-in time) when we wish to begin gathering data from the event. For example if our desired lead-in time is 1 hour, we want to begin gathering data 1 hour before an upcoming weather event. Based on the average predicted precipitation probability of the 3 APIs, we adjust the lead in time for how likely rain is to occur. If it is 80% or above, we begin gathering lead in time 30 minutes before the predicted time. If it is 25% or above, we begin

gathering lead in time 30 * (Average Predicted Precipitation Probability / 100) minutes before the predicted time. If it is less than 25% we will not have any lead in time, but as normal the system will continue to run in case there is rain we have missed.

2. Y: The normal interval between weather predictions. This is how much time our program waits between making predictions.
3. Z: How much time there is before the next predicted event.
4. ST: Standby time, how long the program will wait before predicting. This is inferred in the diagram but is elaborated on here. ST is initially set equal to X when the program starts.

These variables allow the program to switch between the states of the system creating the following transitions:

1. When the program is on standby, when ST time has elapsed, it transitions to the prediction state.
2. When the program is on prediction, if there is no event predicted, it transitions to the standby state, and sets ST equal to Y.
3. When the program is on prediction, and it predicts an event, if the time until the next predicted event is further than X but less than Y, it transitions to the standby state and sets ST equal to Z - X.
4. When the program is on prediction, and it predicts an event, if the time until the next predicted event is within X time, it transitions to the collection state.
5. When the program is on collection, and the ARA Framework data indicates that the weather event is over, it transitions to the post event collection state.
6. When the program is on post event collection, it will transition back to the prediction state without a trigger.

These variables also indicate how far to be looking ahead when predicting and how the states should react to predicted events. The two instances this design highlights are:

1. Based on our prediction time interval Y, we should look ahead 2Y when predicting to minimize missed events.
2. If another event is predicted within 2X time when transitioning from post event collection to prediction, the data from both events should be grouped as one event.

4.3.2 Detailed Design and Visual(s)

To help ourselves and others better grasp our design, we created a component diagram, and separated different components into small groups. Ideal phases for us to create the programming of the design.

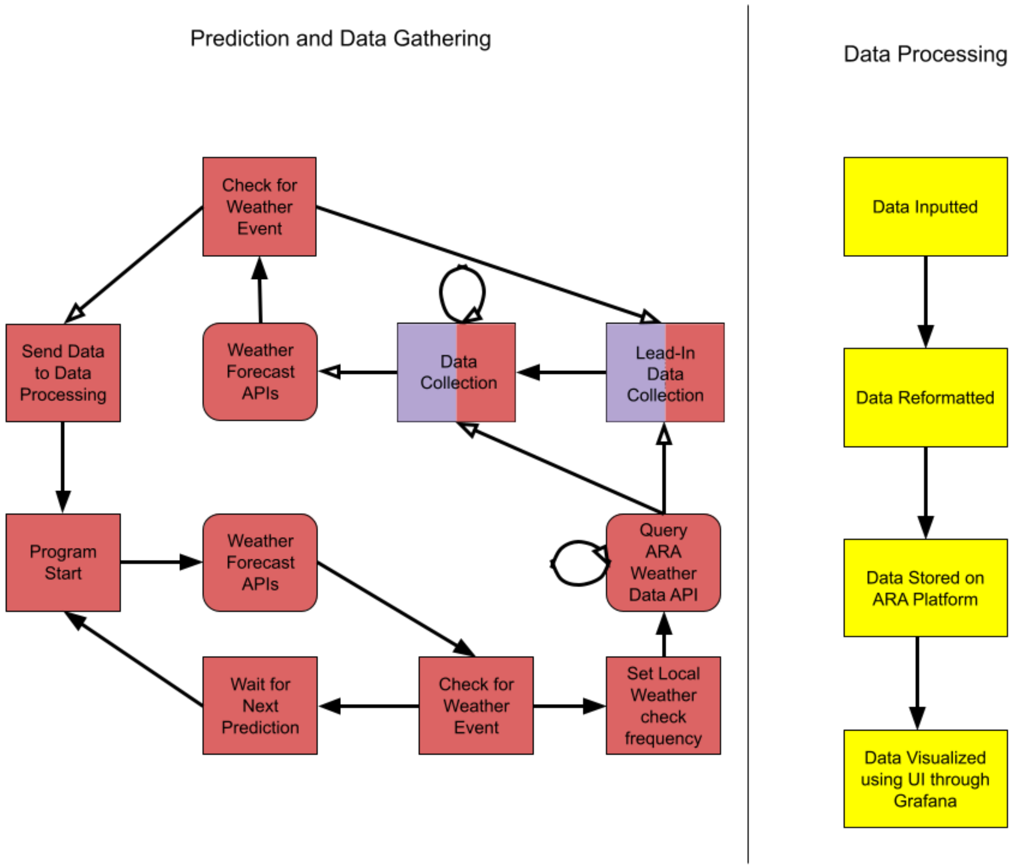


Figure 4.3.2: Component Diagram

The diagram above breaks down the overall system into 3 logic blocks:

1. Weather Prediction and Collection (Red)
2. Wireless Data Collection (Purple)
3. Data Storage and Display(Yellow)

Weather Prediction and Collection (Red):

This logic block handles the querying and formatting of weather data from various API's, as well as the system overhead. First we query various forecasting API's and process the data to determine the probability of the next weather event occurring in an hour. Depending on that probability, we set the frequency in which we check the current weather at local ARA Disdrometers. When a local ARA Disdrometer detects weather, we begin the weather data storage, and start the Wireless Data Collection subsystem. If we have just finished collecting data, and we determine another event is likely to occur soon, we concatenate the two events and keep collecting data, otherwise, we stop collections. Once the weather event is determined to be over, we send the collected data to the Data Storage and Display subsystem.

Wireless Data Collection (Purple):

The purple sections in the block diagram denote when we are collecting and storing ARA Wireless Data. This data needs to be collected on a separate device than all other processes, so this subsystem needs to be able to handle communication between these different devices. When this subsystem is told to begin collecting data, it will generate an id and collect data in association with that id. Following id generation, that id is returned to the caller. When a caller tells the subsystem to stop collecting data with a specific id, it will stop collecting wireless data in association with that id, and return the associated data to the caller.

Data Storage and Display (Yellow):

The yellow blocks handle the output data and formats it in a queryable format. This module will put the output data into a hierarchical zip file as well. This zip file structure is the format the client would like so that users would be able to access these data files. When the weather event is finished and the data from the event is all gathered, it will be reformatted and stored in the ARA platform's database. This data can be queried by users from the ARA platform which can be used for visualization and analysis.

4.3.3 Functionality

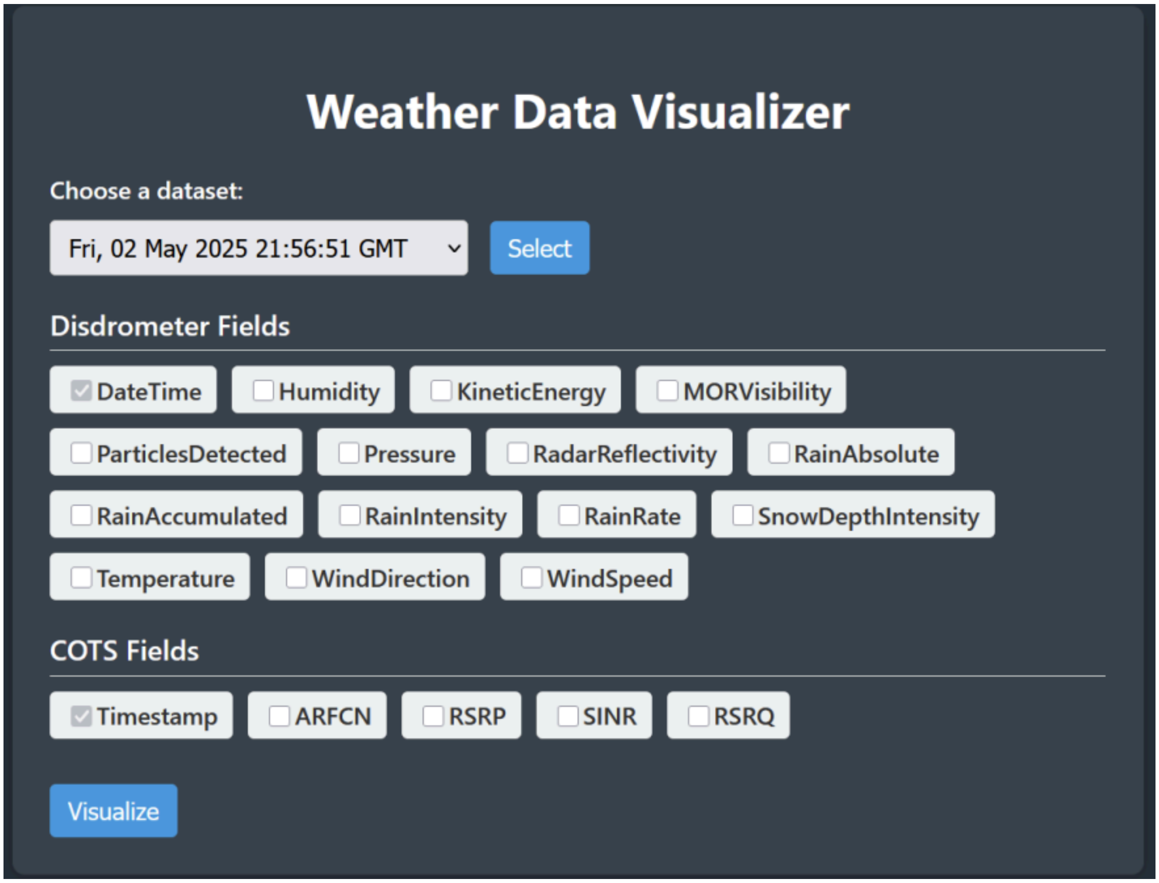
In terms of functionality we see the datasets that have been generated from our system to be visualized in a graphical format generated through a UI. The data can be used to view the collected weather and wireless data in a graphical format. In order to visualize this data we need to first discuss the functionality of the full system and how the datasets are generated. In addition all the scripts for the system and the UI will be run on the ARA server. These scripts should also typically be run as a service on the ARA server such that a user would only have to launch the UI in order for the datasets to be visualized.

First, the weather events will need to be predicted using three different weather APIs. The average probability of a weather event occurring will be taken from the three weather APIs. Based on the predicted probability of a certain weather event occurring certain lead-in and lead-out data collection times will be set. Additionally, when a weather event is predicted multiple different scripts will be triggered.

When the weather APIs predict a high probability of a weather event occurring then lead-in time will be set and the ARA weather APIs will be checked to ensure that a weather event is actually occurring in the specified wireless data collection location. If a weather event is occurring then ARA wireless and weather data will begin being collected. Next, the ARA weather API will be continuously checked every five seconds to see if the specified weather event is still occurring. If the API determines that the weather event has stopped then lead-out time data collection will begin. The lead-out data collection time will be the same as the lead-in data collection time.

Once the data collection has been completed. The ARA wireless datasets and weather datasets will be paired as specifying the weather event in which the dataset has been collected. These datasets will be stored in a hierarchical ZIP file structure. This structure has been specified by the client as all the datasets produced by the client are in this specific structure. The path of the ZIP file of the dataset will be stored in the MariaDB database on the server. The server will host the MariaDB database such that all ZIP files can be stored in this database and these files can be accessed through the UI when requested by the user.

In order to visualize the datasets we created a UI such that all the datasets that have been collected and stored in the MariaDB database can be visualized. The way the UI works is that it is launched on the server. It accesses the MariaDB database which then populates the different dates on the datasets the user can choose from to visualize. The datasets will be unzipped such that they can be used to visualize the data. The user then picks the dataset date they would like to visualize. Once the dataset has been chosen then there will be options of which fields that can be visualized in a graphical format. There will be options for wireless data fields and weather data fields. For each field that is selected a separate graph will be created. All visualizations of the graphs will be done through the Grafana visualization tool. This tool is an open-source interactive visualization application that can produce graphs and charts based on selected data. Once the visualization button is selected the Grafana application will launch. There will be a different graph created for each of the fields selected. Through this application users will be able to visualize the graphs of the weather and wireless data that has been collected.



Weather Data Visualizer

Choose a dataset:

Fri, 02 May 2025 21:56:51 GMT

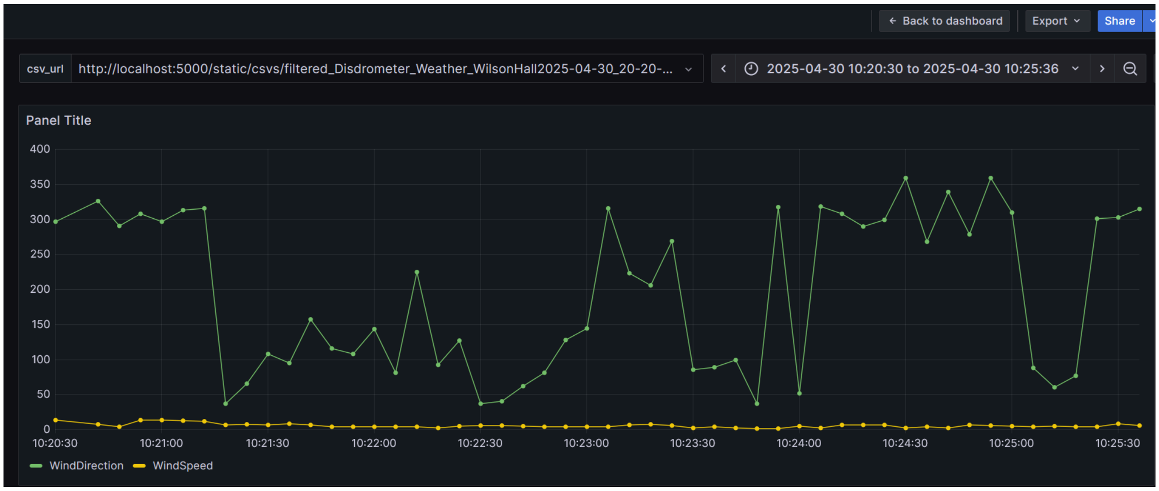
Disdrometer Fields

☑ DateTime ☐ Humidity ☐ KineticEnergy ☐ MORVisibility ☐ ParticlesDetected ☐ Pressure ☐ RadarReflectivity ☐ RainAbsolute ☐ RainAccumulated ☐ RainIntensity ☐ RainRate ☐ SnowDepthIntensity ☐ Temperature ☐ WindDirection ☐ WindSpeed

COTS Fields

☑ Timestamp ☐ ARFCN ☐ RSRP ☐ SINR ☐ RSRQ

Figure 4.3.3.1: Query GUI



csv_url http://localhost:5000/static/csvs/filtered_Disdrometer_Weather_WilsonHall2025-04-30_20-20-...

2025-04-30 10:20:30 to 2025-04-30 10:25:36

Figure 4.3.3.2: Grafana Graph Visualization

4.3.4 Areas of Challenge

Our solution addresses the needs of the internal and external users of the ARA framework users. As a team, we initially had to identify the users and then figure out what exactly they needed from our project. Through this process we derived many requirements that we plan on implementing in our design. Our solution addresses the user needs well since we have directly derived our requirements from the needs the users wanted from the solution. We have implemented functionality that will allow the users to query for weather data given a certain date that would allow the ARA users to determine what the weather was like during that time. This can be visualized through the UI we have created.

There were a few challenges we faced during the first semester of our project. One of these challenges involved determining which lead-in time metrics will work best for our prediction calculations. This included making sure the amount of lead-in time we have allocated is satisfactory for our design. We found a range based on the predicted values to assist us in getting the correct amount of lead-in time, however we feel that this could still use further testing over a longer period of time and more weather events. Another challenge we faced during the first semester was needing to make sure that the metrics that we were taking during that lead-in time accurately gave us the predictions of when a weather event will occur. For example, how well does wind-speed or temperature as a lead-in time metric work for predicting a weather event. In terms of addressing this concern we had different trial runs to see which metrics work best for predicting weather events and chose to focus on precipitation.

The primary challenges we faced during the second semester came from integrating the different elements of the project. We ran into a couple roadblocks that we had to debug when we integrated the ARA data collection scripts and the API weather data section. We ran into different code issues where the wrong methods were being called or some methods were not correctly entering.

4.4 Technology Considerations

When designing our project, we had to consider the abilities and limitations of the technology at our disposal:

ARA Framework & Servers

Strength: Client owned and created, with through documentation and resources.
Weakness: Must fit within the bounds of the existing ARA framework, both for gathering live data and storing data on their servers. Need to consider the amount of storage on the ARA devices.
Trade-offs: We get inside access to the ARA backend, but are limited by how much storage space we have. Our program must also fit in with the existing ARA systems without causing problems

Third-party Forecast APIs

Strength: Are pre existing, professional resources that allow us to easily get the data we need.
Weakness: Must find free, public, APIs. The data collected from APIs will have a range of accuracy.
Trade-offs: We have no control over the code, but we get a much wider range of data for prediction than we could otherwise.
Solutions: Use multiple APIs to get multiple opinions on predicted weather events.

5 Testing

5.1 Unit Testing

We wrote several unit tests covering both our API predictions and the systems on the whole. The tools we used for testing were Py Test and Monkey Patch, a component of Py Test specifically used for mocking a number of calls. A vast majority of our functions directly interact with either APIs, direct data collection from devices (i.e. wireless data collection), and I/O writing and reading of data from files. Monkey Patch allowed us to bypass many of these needed calls during tests. The need to mock a large number of these features when testing does make our tests more “brittle”, but was necessary for testing purposes. For the most part, our unit tests are parameterized tests which test different inputs and expected outputs of several functions allowing for significant coverage of different paths.

5.2 Interface & Integration Testing

Our goals for the interface and integration tests were to verify that our individual components within our system could properly communicate with one another. We did this by covering our critical paths. Our system runs under two primary paths, either we properly predict an event and begin collecting data based on that, or we fail to predict and begin collecting when our system detects a weather event is occurring. Because of this, we have a dedicated test to ensure that we can create weather events from predictions, that we can collect data when predictions begin, and that we can collect data when we miss a weather event. All of these integrate numerous subsystems within our program.

5.3 System Testing

Because our system runs real time and on multiple threads, full system testing can be extremely difficult. Therefore our goal was to have sufficient coverage, both through unit tests and integration tests focusing on either the entire prediction to event creation pipeline, or the data collection pipeline. Additionally, we created dedicated tests for the COTS API. Due to the unknowable nature of real time data collection, we relied on metamorphic testing to test its functionality. All of these tests intandem we believe provide sufficient coverage of our system as a whole. We have also made sure to do our own manual testing and logging to further guarantee correctness within our system.

5.4 Regression Testing

Because we have a large number of automated tests which are able to be run to test multiple aspects of our system, we continue to run these automated tests over time. We will continue to add more of these tests as the development cycle continues, making the test group set more diverse and better for regression testing. Additionally when updating the system, we did comprehensive testing, both manual and automated, to ensure that nothing broke and the system continued to work as intended.

5.5 Acceptance Testing

Our team was somewhat limited with regards to our acceptance testing. Without historical weather data from ARA, our team was unable to simulate our system running over long periods of time, checking for expected accuracy. The best way we found to perform our own acceptance testing was to manually check that the system was running as intended. Once all pieces of our system were functional and connected, we allowed the program to constantly run. With the expectation that the system performed as expected, it would collect data when a weather event occurred. Once the event was completed, we manually performed testing, checking the accuracy of the collected data ourselves to what the weather was at the collected time. We also checked the API predictions accuracy manually, looking for times in which the system did not collect data when it should have. We did this multiple times, refining our code when we found inaccuracies or errors, and restarting the program.

In addition to this testing, we made sure to keep in close contact with our client and advisor about making sure that the system met the requirements that they had and was acceptable to their standards. If we had more time and the system was able to run for a much longer time, we would likely have collected our own complete weather datasets to use as historical data. As complete records of what our system should be expecting, we could perform acceptance testing far easier, looking for our desired accuracy to an exact degree.

5.6 User Testing

The part of our system that Users interact with is our UI which visualizes a given data file with a specific interval of time. The user testing for our UI involved using both manually created test data and real weather data collected from the rest of our system. We first made sure that our UI worked on its own, properly communicating the desired features and data to Grafana. We made sure that Grafana correctly visualized the requested graph when we knew the data was accurate to ensure any issues that occurred would come from the UI and not from the rest of the system. We then used the real weather data collected from our system. These tests allowed us to make sure our system correctly formats our data and that our UI is expecting the right results from the rest of our system. It also allowed us to test that communication between all aspects of the system worked and that the datasets that were being collected could actually be accessed by our UI.

5.7 Results

Our 40+ existing tests currently all pass. Additionally our system is up and running and in the short time it has been running it has already caught 9 separate weather events. If future bugs occur we believe that these tests should help to quickly identify the issues and get our system back up and running. Further tests could be made with more time and with potential accurate historical data. Our existing tests show that our project meets the requirements sent forth to us by our client and advisor and help to guarantee our system works as intended.

6 Implementation

6.1 Overview

Our main implementation that we had worked on was focused on the usage of external API's for our prediction model for when to start collecting data. We chose to use external API's due to concerns about costs for gathering and storing data on ARA's servers. The external API's can provide us with weather data for our prediction model without needing to be stored and gathered on ARA's own systems. We created a state diagram to go over the main states of the implementation. The state diagram has four states, the first being a standby where we wait for the prediction model. The second is the prediction state where we evaluate if we need to begin collecting. Then we have the collection state where we begin collecting data. Finally we have the post event collection where we process the data.

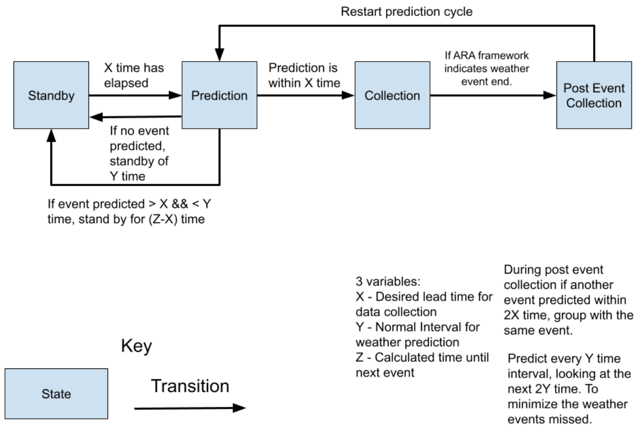

Restart prediction cycle
Standby
X time has elapsed
Prediction
Prediction is within X time
Collection
If ARA framework indicates weather event end.
Post Event Collection
If no event predicted, standby of Y time
If event predicted > X && < Y time, stand by for (Z-X) time
3 variables:
X - Desired lead time for data collection
Y - Normal Interval for weather prediction
Z - Calculated time until next event
During post event collection if another event predicted within 2X time, group with the same event.
Predict every Y time interval, looking at the next 2Y time. To minimize the weather events missed.
Key
State
Transition

Figure 6.1: State Diagram

We also created a task description of the different components of the initial design. The first section of components are the red components which The red sections handle the querying and formatting of weather data. We first query the ARA weather data API then process that data for use in determining if we need to begin collecting the wireless data. The next section is purple which records and formats the data for use in the prediction model for if we need to keep collecting both weather and wireless data. The next section is green which acts as the gatekeepers that start or end various processes for the program such as when to start collecting weather data or if we need to wait. The last section is yellow which processes the data into readable formats or graphs to show correlation between weather events and wireless data performance.

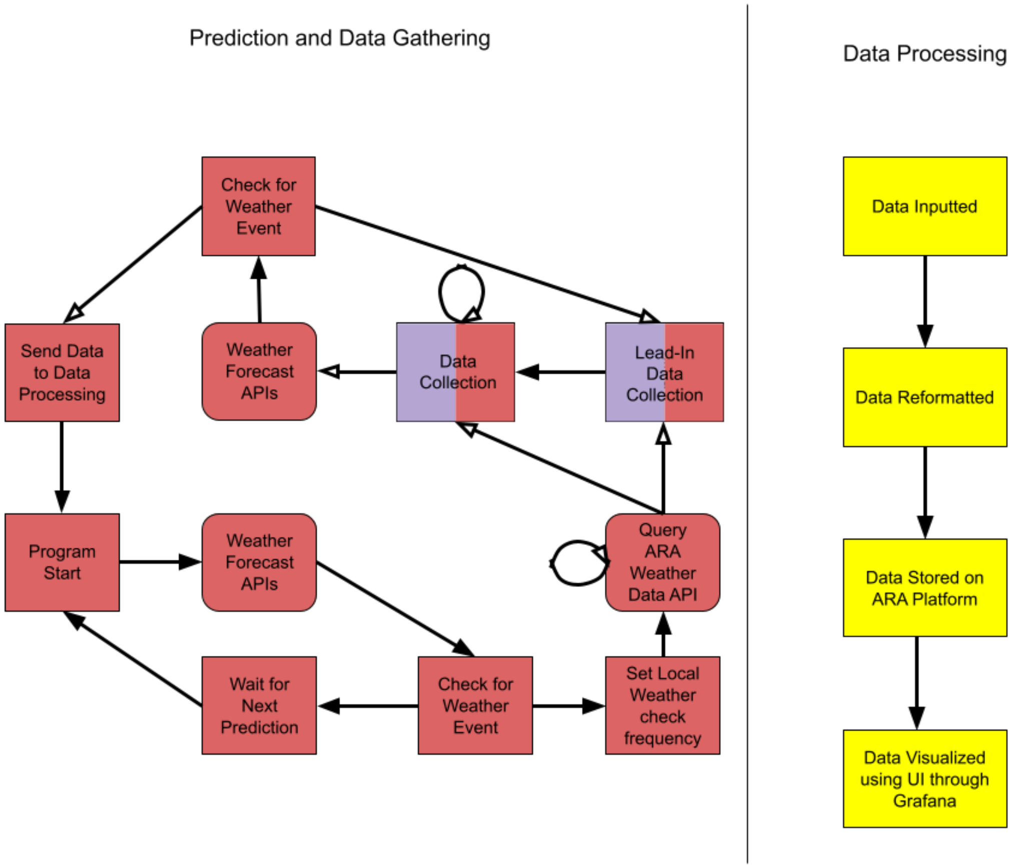


Figure 6.2: Task Decomposition

For each of the API's that we wanted to use in the final project we created scripts that gathered different data values such as temperature, wind speed, humidity, and wind direction. Each of these data points were stored away into text files that would be sent to the prediction model for use in that system. These text files were then used to create graphs that showed absolute error of the graph.

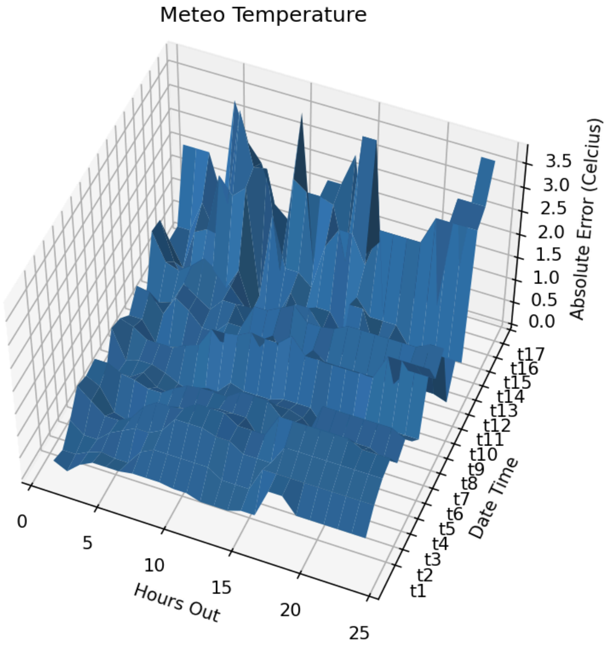


Figure 6.3: 3D Meteo Temperature Graph

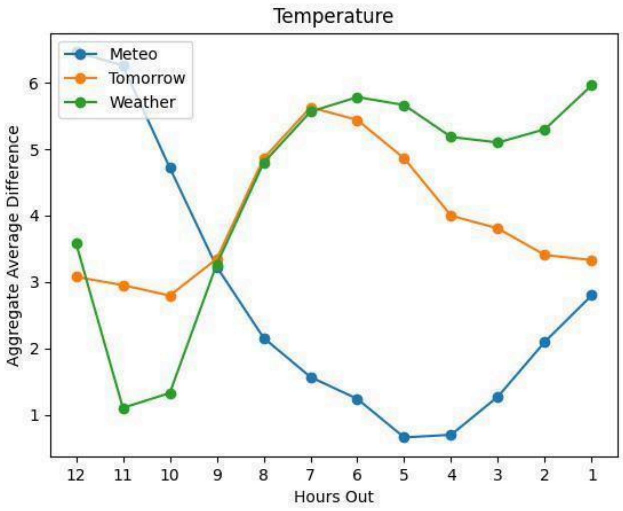


Figure 6.4: Temperature Graph for 3 Different APIs

When analyzing the data from the graphs our team had noticed that there were going to be moments of high inaccuracy. Our team also had other issues with the APIs such as them not being specific enough to the locations of ARA base stations. The external weather APIs could be gathering their data miles away from the ARA base station that we want to turn on. This can lead to moments where the base station is being hit with rain so we want to collect data but the external API weather station is not being hit with rain so the system never kicks on. From this we recognize that a reassessment of the API's implementation is needed to improve the accuracy of prediction.

6.1.1 Weather Prediction and Collection

The implementation of the weather prediction scripts include first creating a script that will predict when weather events will occur. The team does this by using three weather prediction APIs. As listed previously the ones we chose were MeteoWeather, Tomorrow Weather and the National Weather Service API. The team accesses the weather predictions by using each weather services' APIs and then sending that hour's specific weather prediction to the activation script. There the average of the three APIs weather predictions is taken. Each of the three APIs are checked and then every hour the average of the three are taken. That average value is then taken and used to determine the lead-in time of when we want to start the ARA weather collection before a weather event occurs.

Based on the average probability of a certain weather event occurring then we will determine the lead-in time for ARA weather and wireless data collection. When the activate script determines that average value is greater than or equal to 80% then it will begin collecting ARA wireless and weather data with a lead-in time of 30 minutes before the weather event is predicted to occur. Otherwise, if the average value is greater than or equal to 25% then the lead-in time is set to the percentage of the average value multiplied by 30 minutes. This will set the lead in time for the next predicted weather event the APIs see. Additionally, the lead-out time will also be determined by the lead-in time. For example, if the lead-in time is 30 minutes then the lead-out time for that collection event will also be 30 minutes. Then continue normal weather prediction API checking until the next weather event is predicted.

The ARA weather API is also utilized to check if there are any weather events happening that the weather prediction scripts missed. The ARA weather API gets triggered every five seconds because the API updates every six seconds. When there is an event predicted by the weather prediction APIs then the ARA weather API will check if there is still the weather event happening. The data will continue to be collected and if the ARA weather API determines if there is still a weather event ongoing then data will continue to be collected. However, if the ARA weather API determines there is no longer a weather event ongoing for 15 minutes then stop collecting.

6.1.2 Wireless Data Collection

This section of the system handles the collection of COTS Wireless data within ARA Server, as well as the Flask Application that communicates with the rest of our system. The subsystem runs on a separate server than all other subsystems, specifically, it runs on the same server where the wireless data collection occurs. This subsystem integrates with the other subsystems via a Flask Application. This application has two API endpoints, start_collection and end_collection, these endpoints allow the Weather

Prediction and Collection subsystem to control when we are collecting Wireless Data. When the start_collection endpoint is hit, Wireless Data Collection is started and the caller is given an associated id to the collection. When the end_collection endpoint is hit with a valid id, Wireless Data Collection is ended and the wireless data collected with the associated id is sent to the caller.

This current implementation works very well as it allows us to easily start and stop wireless data collection on demand from the Weather Prediction and Collection subsystem without impacting other end users. Additionally, this solution is particularly effective as it allows other users to hit this endpoint and get the same data for themselves. This allows this particular subsystem to be useful outside of the context of our problem statement. We have proven this by having multiple users simultaneously collect data from the same endpoint without impacting the others. One current downside of the system is that each ids data is written to a separate file on the server, thus, if multiple users are collecting simultaneously, a large amount of redundant data is being stored. Currently, we are looking into a rewrite that will stop the writing of redundant data.

6.1.3 Data Storage and Display

This part of the system handles the renaming and storage of weather and wireless data files that are collected by the system. This part takes inputs of the weather data file path, the COTS data file path, the location code (that we have designated), the start time of the collection and optionally some keywords that are used to name the folder and files to meet ARA Standards.

The current implementation starts by renaming the weather and wireless data files and then are placed in a specifically named folder. It is compressed into a zip file as per the client's instructions. The zip file is stored in a folder on the ARA server. The system also stores the file path of the zip file into a MariaDB database.

To allow our users to visualize the data graphically, we also developed a UI in the form of a web application hosted on the ARA server. This web application allows the user to select a dataset (the list of optional datasets are pulled from MariaDB and are listed based on date) and select specific fields from the dataset to visualize. These fields can include disdrometer data such as temperature and rain rate along with COTS data such as SINR. Once the user has selected the fields they wish to view, a new page is opened showing the generated Grafana dashboard with individual visualizations for each selected field. The user will need to log into the server's local instance of Grafana using the AraResearcher login credentials included in the User Guide. After three hours of inactivity, the web application will mark the generated dashboards to be deleted.

6.2 Design Analysis

Our design overall works well. The weather prediction and collection component is able to get values from external API's and average their precipitation values to send off to the weather and wireless data collection component. This component works well due to it being thoroughly tested to ensure that data values from the external APIs are being collected. As for the wireless data collection component, it is properly triggered by the weather collection and is paired with the weather data collected. This component works well because it connects to ARA devices to gather and record that data points that will be packaged together Lastly for the data storage component it works well in that it is properly able to receive paired data files, zip them together and store them, and store their file paths to later be referenced by the UI for displaying.

One part of the project that could use some improvement is possibly the detection of other types of weather events. Further testing can be done to ensure that the system works. Another part of the project that does not work as well as expected is the weather prediction component of the project. There are times where a weather event is very likely happening or happened but the prediction might not send the signal to start gathering data leading to no lead in data being collected. This is due to the various external API's sometimes being accurate. For example, the Tomorrow Weather API will not send out a percentage chance of precipitation until the precipitation event is likely already happening. What we could have done was more testing of the API's to see what aspects of them are accurate and can be used in our project. More testing could be done with the API's to provide some weight to certain aspects of them to better increase the accuracy of our own prediction model.

7 Ethics and Professional Responsibility

Through the course of this project our group has thought about engineering ethics and professional responsibility. Initially we had come up with possible ethical concerns that are applicable to our project. We used the exercises provided in the first half of senior design to help generate ethical codes to follow and also plans on how to abide by the ethical codes. Throughout the second semester our ethical concerns have not changed due to us already adhering to our code of ethics. While working on the project there were no new ethical concerns we had as a team therefore the sections below have not changed.

7.1 Areas of Professional Responsibility/Codes of Ethics

Area of Responsibility	**Definition**	**Code of Ethics**	**Adherence to the Code**

Public	Create products and services that benefit the public.	1.01. Accept full responsibility for their own work.	Our team has adhered to this code throughout our project. Everyone is assigned tasks and are responsible for completing that task at a reasonable time and quality.
Client and Employer	Perform work that aligns with the interests of the client and employers.	2.03. Use the property of a client or employer only in ways properly authorized, and with the client's or employer's knowledge and consent.	Our team has contacted and signed up for various accounts for third party APIs to use them for parts of the project. We have also talked to our client to ensure that these third party APIs are alright to use. We have also contacted our client to gain access to their own systems so we can run our project's prototypes on them.
Product	Products created meet specified requirements and standards.	3.07. Strive to fully understand the specifications for software on which they work.	Our team has contacted our clients many times to better understand their systems so our project doesn't interfere with the other processes that are on the system. We wanted to

			ensure that our code would not prevent any other users from gathering weather data and wireless performance data
Judgement	Maintain integrity about any assumptions and judgments made in the project.	4.03. Maintain professional objectivity with respect to any software or related documents they are asked to evaluate.	Our team has ensured that our documents have been as accurate as possible when describing our progress on our project.
Management	Managers and leaders ensure an ethical approach to the management of software.	5.02. Ensure that software engineers are informed of standards before being held to them.	While our team doesn't necessarily have a manager we as a team still define the standards set out for each other when completing tasks.
Profession	Advance the integrity and reputation of the profession with the interests of the public.	6.08. Take responsibility for detecting, correcting, and reporting errors in software and associated documents on which they work.	Our team reviews documents and code that are created before we finalize them. If there are any inaccuracies or parts that need to be changed then requests for change are created and carried out to ensure that the product meets the

			end requirements.
Colleagues	Be fair and supportive towards colleagues.	7.04. Review the work of others in an objective, candid, and properly-documented way.	Whenever our team reviews another member's work, if there are any changes that need to be made we state them in an objective and constructive way. Oftentimes we offer to help or give some suggestions.
Self	Participate in learning more about the profession of software engineering and an ethical approach to the profession.	8.03. Improve their ability to produce accurate, informative, and well-written documentation.	Our team has used senior design I as a learning tool to create and design well-written documentation. These skills are invaluable in the professional work force and are always needed for any job so being able to make mistakes and learn from them without too much consequence is very important.

Table 5: Areas of Professional Responsibility

Areas Done Well

One area that our team is doing well in is the product area. Our team has been able to produce high quality products for our client in a reasonable amount of time. Any suggestions the client has in regards to our current deliverables is incorporated in by the next weekly meeting. We understand what needs to be done for our client and when it needs to be done. This shows that our team is motivated when it comes to doing work and is also diligent when it comes to ensuring that our deliverables meet our clients needs.

Areas for Improvement

One area that our team can improve on is the client and employer area. There were several times where there was miscommunication between our team and the client as to the scope and limitations of our project. Our team could have done a better job in reaching out to our client more often to ensure that we fully understood the scope and limitations of the project. In the future we plan to have more meetings and go to our clients office hours if we have questions about certain system requirements.

7.2 Four Principles

	Beneficence	**Nonmaleficence**	**Respect for Autonomy**	**Justice**
Public Health, Safety, and welfare	Design helps researchers gather and analyze both weather data as well as wireless device data.	Design works on existing ARA systems and can be easily expanded with ARA's deployment of other base stations.	Design allows for users to provide feedback. Users can also tailor the project to fit their local area.	Design promotes access to all users with permissions.
Global, cultural, and social	Design helps gather data to improve transmission of wireless data in rural areas.	Design allows a variety of users to use our project. Different cultures and groups of people can use the project.	Design allows users to use the project for specific purposes. They can gather data for their local area.	Implementation is focused on access for educational purposes.
Environmental	Design will help research in rural environments.	Implementation will slightly disrupt the environment. The only impact would be the base stations that need to be set up in that local area.	Design can be tailored to the individual environment where the base station is set up.	Design does not disturb one type of habit over others.

Economic	Design could help to speed up research regarding solutions to rural broadband connections.	Design has not a large impact on the economy. It could affect certain companies that offer products for broadband connection.	Design has some costs for the client currently. These costs are mostly around computation costs for gathering and parsing wireless data.	Design would be usable for all. As long as users follow ARA's user agreements they are allowed access to ARA's system.

Table 6: Broader Context and Four Principles

One context principle pair that is important to our project is the global, cultural, and social beneficence pair. This pair is important because our project will be used by a variety of other users. For now the users in our current scope are internal users in ARAs system and external users. When it comes to the principle pair we are more focused on the external users since they will have different needs and uses for our project. We are working hard to ensure that our project will be useful for these different needs and uses by making sure that our project is easy to use and provides reliable and accurate information.

One context pair that our project is lacking is economic respect for autonomy. Our current project has some cost for our current client. The costs of our project comes from the computations needed to gather and parse the wireless data, and the storage of some of that data to use in our prediction models. These costs can quickly grow as more users are on ARA's system as there will be many calls to not only our project but other experiments on the system. To mitigate these costs our team is only storing enough data to reliably predict weather events and are ensuring that we are only gathering wireless data whenever we are certain that a weather event is occurring.

7.3 Virtues

Team Virtues

1. **Commitment to quality:** This virtue is all about creating quality work which is something that our team strives for. We all value our work and want to ensure that it meets the requirements set out by our clients as well as our own personal standards. We as a team review and critique each other's work to find any areas for improvement.

2. **Responsibility:** This virtue is about each team member taking responsibility for the work that they have done. As a team we make sure that work is divided evenly

and each person knows what work they need to do. If one person is struggling with their work we offer help to them.

3. **Integrity:** This virtue is about being honest with the work that each person has done. Each person in the group knows what work they must do and if they are having issues completing their work then we appreciate honesty about how they are struggling with the work. This integrity is important since we want to ensure that the work still gets done at an acceptable quality.

Individual Virtues

Aidan Gull

I have demonstrated friendliness. It is important to me because I want to have a solid team dynamic as well as a friendly work environment. I have demonstrated this by offering assistance whenever needed.

I have not demonstrated industriousness. It is important to me because I would like to be more focused and get more work done for the team. I can demonstrate this by removing any possible distractions from my work areas to help me stay on task.

Adam Fields

I have demonstrated responsiveness. It is important to me because the team contract requires it of me. I have demonstrated this by paying close attention to the team discord and helping clarify things as soon as possible.

I have not demonstrated resolution in a few cases. It is important to me because the team expects me to do what I have said that I would do. I can demonstrate this by doing what I have said I’m going to do without fail.

Alex Chambers

I have demonstrated resolution. It is important to me because when someone commits to working on something, there is a certain expectation that they will complete that work. I have demonstrated this by generally working to completion on various code components of our project.

I have not demonstrated timeliness. It is important to me because when working on a task, it is important to complete it at a convenient time so as not to interfere with others' progress. I can demonstrate this by striving to be more productive during normal working hours rather than doing a majority of my work later into the night.

Alexander Christie

I have demonstrated communication. It is important to me because communication is the key to success in a project like this. With so many moving parts and pieces that need to be completed, strong communication with our team, client, and advisor is necessary. I have demonstrated this by responding promptly to team messages and announcements, alerting my team members to progress I've made on different tasks, and serving as the head of client communications.

I have not demonstrated competence. It is important to me because being able to complete work at an adequate pace is important, especially if I've previously made assurances a deadline could be met. I can demonstrate this by being more aware of my limitations and working to improve my skills.

Colin Kempf

I have demonstrated order. It is important to me because I want to keep things organized and easy to find so the team won't get caught up on where things are or problems arriving from clarity. I have demonstrated this by keeping our documentations structured and by making sure that we follow similar formats throughout our work.

I have not demonstrated silence. It is important to me because our team often needs to focus and remain on task to get work done. I can demonstrate this by talking less with others in distracting conversations and staying on task.

Nisha Raj

I have demonstrated cooperativeness. It is important to me because in order to work well within a team all members need to cooperate with each other. I have demonstrated this by making sure all voices within our team are heard and by ensuring that all team members have a chance to express their opinions and views on team matters.

I have not demonstrated resourcefulness. This virtue is important to me and the team because we need to be able to find innovative solutions for any problems that might arise. I can demonstrate this by working to think of more out of the box and unique solutions to problems that come up while we are working on our project.

8 Conclusions

8.1 Summary of Progress

Through our work, our team has successfully addressed the project's overall goals that were set out by our client and advisor. With our project we have designed, developed, and tested an application that can use external API's to forecast predictions that are used to inform ARA's systems to start collecting both weather and wireless data. Our team found 3 different APIs to use for forecasting data which address our needs and requirements for predicting weather events. Our group has tested our script for accuracy and have ensured that our project is able to successfully complete its given task. We feel our testing of our project has been through, but it would benefit from additional testing overtime as it requires weather events outside our control.

8.2 Value Provided

The design our group has created sets out to meet all of our users' needs. It accomplishes this task fairly well within the bounds of the project's requirements. The design takes priority using forecast predictions to inform the collector, which means that other problems such as lead-in time our design attempts to address are sometimes less accurate. However our design does take this into account, attempting to make up for errors of false positives and negatives. In the broader context, our design fits in with the expectations of our client and advisor.

There are several ways we can see the value our design provides. For starters, it includes false positive and negative checks for predictions. This allows us to more accurately collect data according to our users needs, ensuring that even when API predictions are wrong, we can collect some of the data we might have otherwise missed. Additionally, our design is able to recognize closely timed weather events as one event. This allows for more seamless data which can be more easily understood and visualized by users after collection. Lastly, our design handles formatting the data into a UI for the user to be able to visualize the gathered data. This addresses how the user will be able to interact with our design with ease.

8.3 Next Steps

There are several next steps ARA could take to further our project after our group's completion of this course. The first step would be to continue testing. Because our script monitors live weather data, our script will need to be tested in all kinds of weather events. It also needs to be tested for extended periods of time to ensure its continued functionality. The second step would be to integrate our scripts with the existing ARA experiments system. This would allow for more seamless access to running our scripts and would centralize it with the rest of the ARA systems. The third step would be to improve the accuracy of the API predictions. This could be done either by looking at the accuracy of the current three APIs, looking at any outliers which bring down the accuracy, or by adding new APIs to get a better averaged prediction. The last major next step would be to add our scripts to additional ARA location weather and wireless data collection sites. This would involve slight adjustments for the data we expect to collect but should be mostly a smooth transition.

9 References

[1] Tomorrow.io APIs, Tomorrow, https://www.tomorrow.io/weather-api/ (Accessed Sept. 25, 2024).

[2] Open-Meteo.com Free Open-Source Weather API, Open-Meteo, https://open-meteo.com/ (Accessed Sept. 25, 2024).

[3] API Web Service, National Weather Service, https://weather-gov.github.io/api/ (Accessed Sept. 25, 2024).

[4] "How to Create a Feedback Loop: Step-By-Step Guide With Best Practices", Userpilot, https://userpilot.com/blog/how-to-create-a-feedback-loop/ (Accessed Dec. 2, 2024).

[5] "The Ultimate Guide to Building a Functioning Feedback Loop Model", Fibery, https://fibery.io/blog/product-management/feedback-loop-model-guide/ (Accessed Dec. 2, 2024).

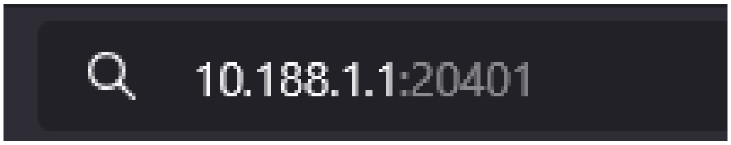
10 Appendices

Appendix 1 – Operation Manual

In order for a user to use our project they would need to get access to ARA servers that host our code. To get access to the ARA servers the user would have to contact ARA and create a user account with them. Once the account is created the user can reach out to an ARA representative to get specific access to the server that contains the code. This access will be accompanied with guides on how to use the server and access its contents.

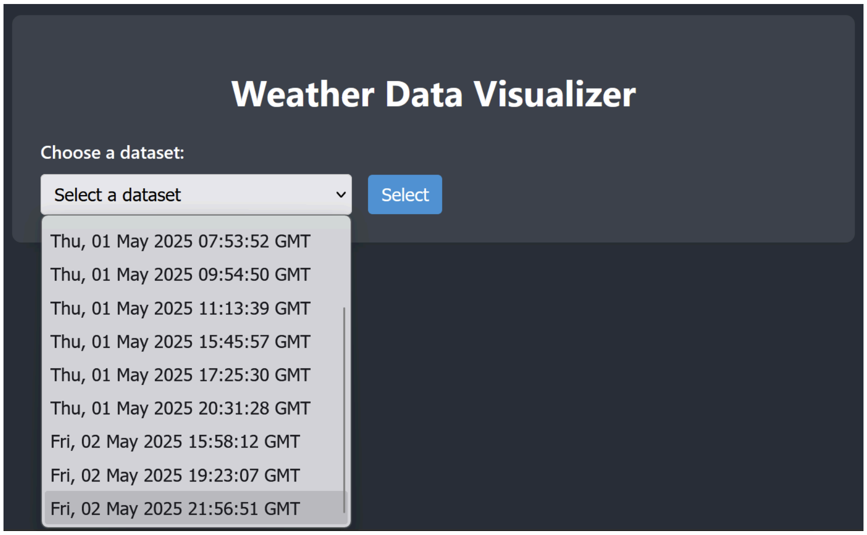
Now that the user has access to the server that our code is running on, the user can view and utilize the Weather Data Visualization UI. The user does not have to manually start the web application (unless it is down) and doesn't even need a connection established to the ARA server itself. All the user needs to access the UI is a local machine with a web browser installed and a VPN connection to the Iowa State network.

From the user’s local machine, they can open a browser and access http://10.188.1.1:20401/ which hosts the UI. Once the user has access to the local host UI, they can select what data they want to visualize. Within the UI, the user first can select the gathered dataset from a given weather event. Then there are several different weather features the user can select to have visualized on Grafana. Multiple features can be selected at a time. Then the user can select as many of these different features as they want. Once satisfied with their selections, the user clicks the Visualize button at the bottom of the page. This will automatically take the user to the Grafana graph generated from their data and feature selections. The user may need to login utilizing the AraResearcher account (username: AraResearcher ; password: Password).



10.188.1.1:20401

Step 1: Navigate to http://localhost:5000/

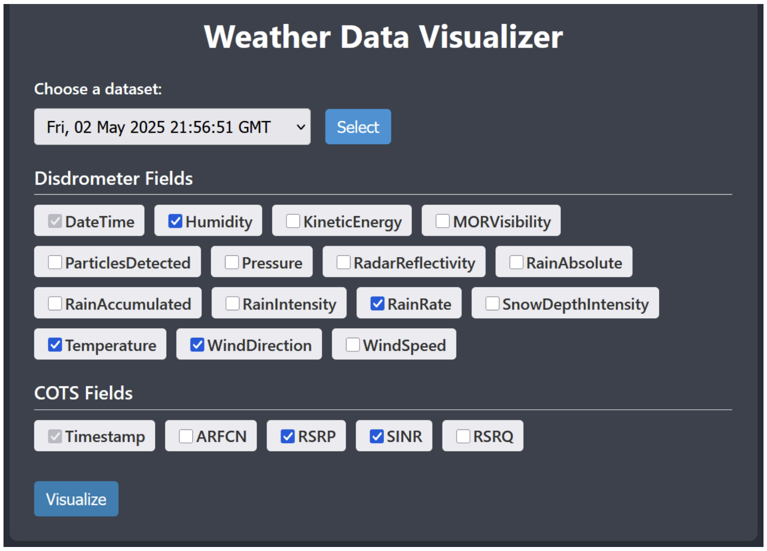


Weather Data Visualizer

Choose a dataset:

Thu, 01 May 2025 07:53:52 GMT
Thu, 01 May 2025 09:54:50 GMT
Thu, 01 May 2025 11:13:39 GMT
Thu, 01 May 2025 15:45:57 GMT
Thu, 01 May 2025 17:25:30 GMT
Thu, 01 May 2025 20:31:28 GMT
Fri, 02 May 2025 15:58:12 GMT
Fri, 02 May 2025 19:23:07 GMT
Fri, 02 May 2025 21:56:51 GMT

Step 2: Pick dataset to visualize and click Select



Weather Data Visualizer

Choose a dataset: Fri, 02 May 2025 21:56:51 GMT

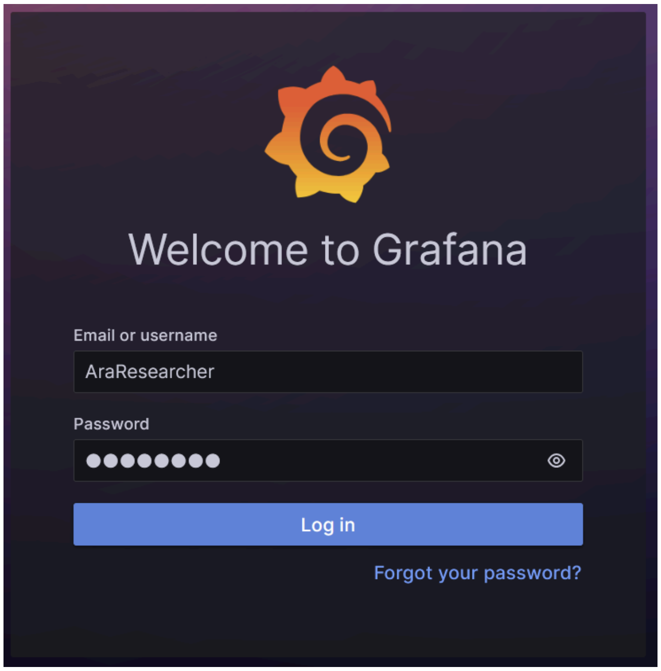
Disdrometer Fields

☑ DateTime ☑ Humidity □ KineticEnergy □ MORVisibility
□ ParticlesDetected □ Pressure □ RadarReflectivity □ RainAbsolute
□ RainAccumulated □ RainIntensity ☑ RainRate □ SnowDepthIntensity
☑ Temperature ☑ WindDirection □ WindSpeed

COTS Fields

☑ Timestamp □ ARFCN ☑ RSRP ☑ SINR □ RSRQ

Step 3: Select fields from Disdrometer and COTS and click Visualize



Welcome to Grafana

Email or username: AraResearcher

Password: ●●●●●●●●

Step 4: Login with Grafana AraResearcher

Features Available for Visualization:

1. Average Particle Speed
2. Date Time - Selected by default

3. Humidity
4. Kinetic Energy
5. MOR Visibility
6. Particles Detected
7. Pressure
8. Radar Reflectivity
9. Rain Absolute
10. Rain Accumulated
11. Rain Intensity
12. Rain Rate
13. Raw Volume Equivalent Diameter
14. Snow Depth Intensity
15. Temperature
16. Volume Equivalent Diameter
17. Wind Direction
18. Wind Speed

Appendix 2 – Alternative/initial version of design

One design we first considered after talking to our was to always collect and determine afterwards if there was a weather event that took place. We scrapped this idea because it did not align with client specifications.

Another design we considered used a feedback loop to more accurately predict weather. We would use collected ARA data to then be used as a predictor for determining future weather events by helping to define what, for example, “rain” looks like. The concept was scrapped for being way too complicated for the time and skill constraints of the project.

Appendix 3 – Code

https://git.ece.iastate.edu/sd/sdmay25-18

Appendix 4 – Team Contract

Team Members

1) Aidan Gull

2) Colin Kempf

3) Adam Fields

4) Nisha Raj

5) Alexander Christie

6) Alexander Chambers

Required Skill Sets for Your Project

Front-End development: Developing the front end so that users can query weather data and the wireless signal data.

Linux Experience: Running processes in the background and piping output into a file that can be parsed.

Python Development: The scripts that we will be creating in order to correlate weather data and wireless signal data will be in Python. Many ARA APIs already utilize Python for scripting so this was the preferred language.

Experience with ARA: Familiarity, with the ARA platform and different experiments that can be run on the testbed.

Data Analysis: Understanding how to assess and manipulate gathered data to inform our development, create an algorithmic feedback loop, and format for the final deliverable to the end user.

Skill Sets covered by the Team

Front-End Development: Adam, Colin, Alex Cha., Aidan

Linux Experience: Adam, Nisha, Alex Chr., Alex Cha., Aidan

Python Development: Nisha, Alex Chr., Colin, Aidan

Experience with ARA: Nisha, Alex Chr.

Data Analysis: Colin, Adam

Project Management Style Adopted by the team

Our team uses an Agile methodology since we have team meetings weekly where we collaborate as a team. We also speak with our advisors on a weekly basis where we work and discuss breaking down tasks into smaller components. The weekly meetings help facilitate consistent feedback from our client and help us stay on track with our project.

Team Procedures

1) Day, time, and location (face-to-face or virtual) for regular team meetings: Tuesdays Wendsdays, and Thursdays from 2-4pm at the library (face-to-face)

2) Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face): Discord
3) Decision-making policy (e.g., consensus, majority vote): Majority Vote
4) Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived): Team Recorder: Colin Kempf, Archived: Stored in documents in a team shared Google Drive.

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings: We expect all members to attend all meetings unless some unforeseen circumstances arise or they communicate a reasonable reason for not being able to attend. We also expect all members to contribute equally to the project and try to split up tasks depending on the member's strengths.
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines: Everyone should roughly have equal responsibilities. Do what you commit to within the timeline specified, if issues arise, communicate with team members and ask for help.
3. Expected level of communication with other team members: We expect steady communication of schedule conflicts, development progress, and ideas among the team through our discord.
4. Expected level of commitment to team decisions and tasks: Strong commitment to the team, decisions, and tasks. Communicate with the team during the decision making process and talk about any disagreements and differences in opinions.
5. Strategies for supporting and guiding the work of all team members: We will use consistent communication to ensure that the team is organized and support is extended to individuals who need additional help on their tasks.
6. Strategies for recognizing the contributions of all team members: Give positive feedback when team members do a good job on a project. Give team members the correct credit for their contributions.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):

Alex Chambers: Individual Component Designer

Alexander Christie: Client Interaction

Adam Fields: Data Formatting

Nisha Raj: Team Lead

Aidan Gull: Component Integration

Colin Kempf: Documentation

2. Strategies for supporting and guiding the work of all team members:

Our core strategy for supporting and guiding the work of all team members is having frequent team meetings. During the meetings, everyone is able to update their progress and share their work. This will help everyone to be on the same track and keep up to date with where others are at.

3. Strategies for recognizing the contributions of all team members:

Our strategy for recognizing the contributions of all team members is making sure that each person receives credit for their individual parts. We want to ensure that others outside of our group know of our team members' contributions so that they can get validation from others. This includes recognizing them during meetings with our client and advisor, the industry review panel, and the poster presentation.

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.

Alexander Christie: I have industry experience utilizing Python scripts to automate complex tasks. Also, through my experience with acquiring a Technical Communications minor, I’ve gained valuable skills in technical document writing and professional communication.

Alexander Chambers: I have 3 semesters of research experience utilizing python scripts. Additionally, I have had 3 summer internships giving me a lot of experience working on and producing commercially viable code.

Nisha Raj: I have done previous research with the ARA team so I have some background experience in their framework and their ARA portal. I have industry experience working in wireless communications. I also have a minor in cybersecurity so I have experience in Linux and security measures the team could take to keep the data we collect secure.

Aidan Gull: I have some industry experience working on the backend of various systems. I have also worked with python and frontend development that would be useful for this project.

Colin Kempf: I have industry experience with gps devices from my time doing an internship with Samasung. I also am getting a Data Science minor along with my major so I have experience handling and analyzing data.

Adam Fields: I have industry experience with monitoring wireless connections. I am also minoring in Cyber Security.

2. Strategies for encouraging and support contributions and ideas from all team members:

Give positive feedback when team members do a good job on a project. Give team members the correct credit for their contributions.

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)

The best way to inform the team of this issue is to communicate in person with the team during our meetings on Fridays. Members can also bring it up in the Discord chat. The team can address the issue and analyze what is causing it and how to go about resolving the issue. The goal of the team is to make sure every team member is heard and has an equal say in all the decisions and projects.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:

Complete deliverables as they arise, meet expectations of our client, set ourselves up for success next semester. Make sure we are making progress in terms of implementing the software components that we had planned out from the previous semester.

2. Strategies for planning and assigning individual and team work:

Assign tasks based on individual skills and team members schedules. Work together to determine who is best suited to approach what and who may need more help based on difficulty and time. Additionally, try to work together on larger tasks that may be more involved and need full team collaboration.

3. Strategies for keeping on task:

Make sure the team doesn’t get off track by talking about unrelated topics. This can be achieved by keeping each other accountable when someone goes off track. The team needs to work together to bring everyone back to working on the related task at hand.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

   First, try to determine the cause of the issue and resolve things internally by communicating. Speak with team members and see what is going on and why the infractions are happening. Make it clear with team members that this is a team project and everyone needs to evenly contribute in order to achieve the end goal, and let them know that by participating in these infractions they are letting the team down and even jeopardizing the project.

2. What will your team do if the infractions continue?

   If the infractions continue then the team will speak with the advisors and the professors about the team members' continued infraction and how it is negatively affecting the productivity of the team.

****

a) *I participated in formulating the standards, roles, and procedures as stated in this contract.*
b) *I understand that I am obligated to abide by these terms and conditions.*
c) *I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.*

1) Aidan Gull	DATE 4/3/2025
2) Colin Kempf	DATE 4/3/2025
3) Adam Fields	DATE 4/3/2025
4) Alexander Christie	DATE 4/3/2025
5) Alexander Chambers	DATE 4/3/2025
6) Nisha Raj	DATE 4/3/2025