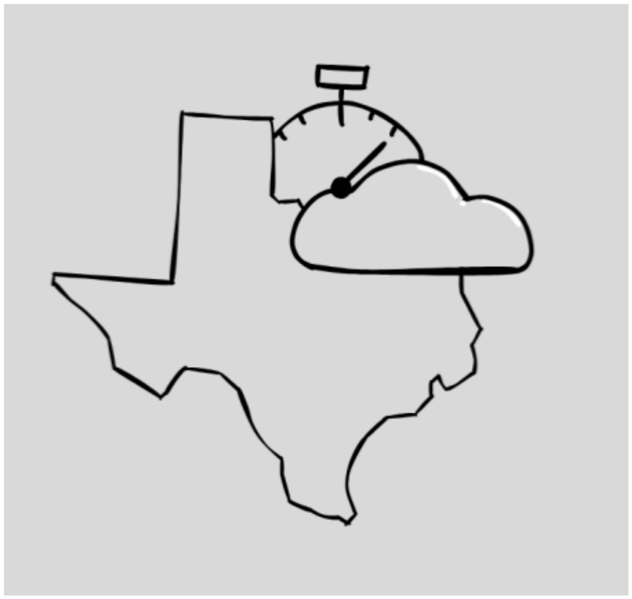
Department of Computer Science & Engineering
The University of Texas at Arlington

Architectural Design Specification
CSE 4316: Senior Design I
Fall 2025



Forecast Tx
Severe Weather Prediction Model

Noe Sanchez
Kevin Simbakwira
Kristal Phommalay
Kenil Patel
Mason Berry

Revision History

Version	Date	Author(s)	Description
0.1	04.07.2025	MB, KP, KPh, KS, NS	document creation
1.0	7.22.2025	MB, KP, KPh, KS, NS	Updated for Closeout

Contents

List of Figures

List of Tables

1 INTRODUCTION

The Severe Weather Prediction Model is composed of a Prediction Model that outputs a prediction of the likelihood of severe weather occurring in Texas in the next 5 to 10 years and a Web Application that displays the data given from the model. The Severe Weather Prediction Model is designed to be used by State Farm's risk management and claims team to predict if the severe weather events will occur in the future to help mitigate some of the losses.

The users of the Web Application will be either a general user or an advanced user. General users will be able to view the predictions that the model produces through graphs, maps, and interactive displays. They can filter and select certain types of weather or perils they wish to view. Advanced users will be able to view not only the model predictions, but also the data inputted, chosen features, machine learning descriptions, and supply or update data.

The prediction model will utilize machine learning algorithms such as neural networks, linear regression, SARIMA, and LSTM for time series forecasting. The model will use data collected from data sources such as NOAA and ERA5 that is stored in PostgreSQL and cloud data buckets.

The output from the algorithms will be displayed on a web application that utilizes React.js. This ensures the web application is easily viewable on different devices. The users will be able to interact with the displays and input parameters to get a prediction. The entire system will be maintained in Google Cloud.

2 System Overview

This section describes the overall structure of the software system. The system has 5 layers: Data Storage Layer, Data ETL Layer, Cloud Layer, Machine Learning Layer, User Interface Layer.

System Overview

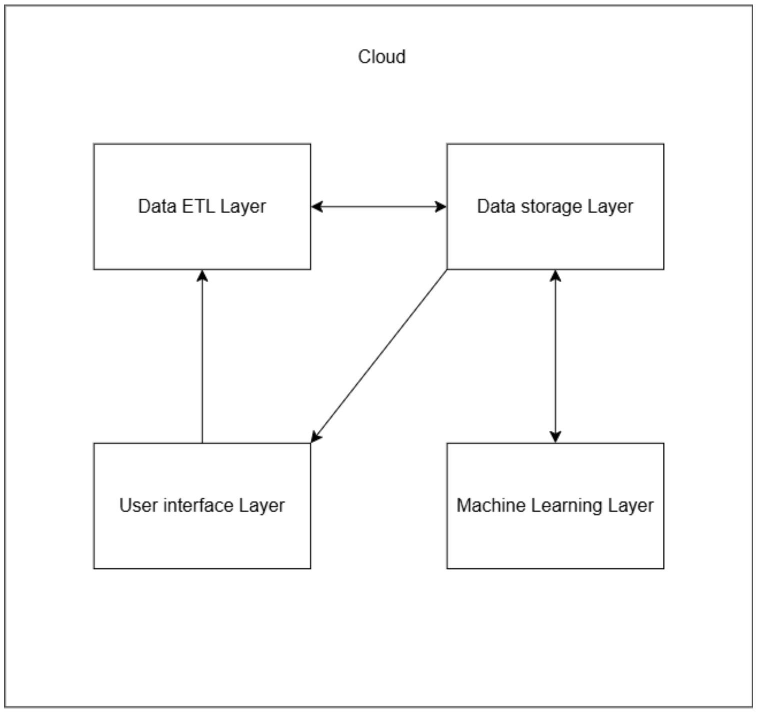


Figure 1: Architectural Layer Diagram

2.1 Data Storage Layer

The Data Storage Layer is responsible for maintaining all structured datasets required throughout the machine learning pipeline. It provides secure, persistent, and scalable storage for both pre-processed input data and model outputs. This layer ensures data integrity, traceability, and accessibility for other components of the system, particularly the Machine Learning and Analysis layers.

2.2 Data ETL Layer

The Data ETL (Extract, Transform, Load) Layer is responsible for ingesting raw environmental and climate data from external sources, transforming it into a structured and enriched format, and loading it into persistent cloud storage. This layer ensures data consistency, cleanliness, and readiness for downstream machine learning tasks and analytics.

2.3 Machine Learning Layer

The Machine Learning Layer is responsible for executing model training workflows, validating model performance, and storing prediction outputs. This layer orchestrates the retrieval of training data, application of machine learning algorithms, and evaluation of model accuracy using cross-validation techniques. it serves as the computation cor of the system's intelligence

2.4 User Interface Layer

The User Interface (UI) Layer serves as the primary interaction point between users and the system. It provides a visually intuitive and role-sensitive platform for interacting with data outputs, initiating model updates, and retrieving information. Designed with both functionality and usability in mind, the UI supports a range of user roles, customizable data views, and data export options.

3 Subsystem Definitions & Data Flow

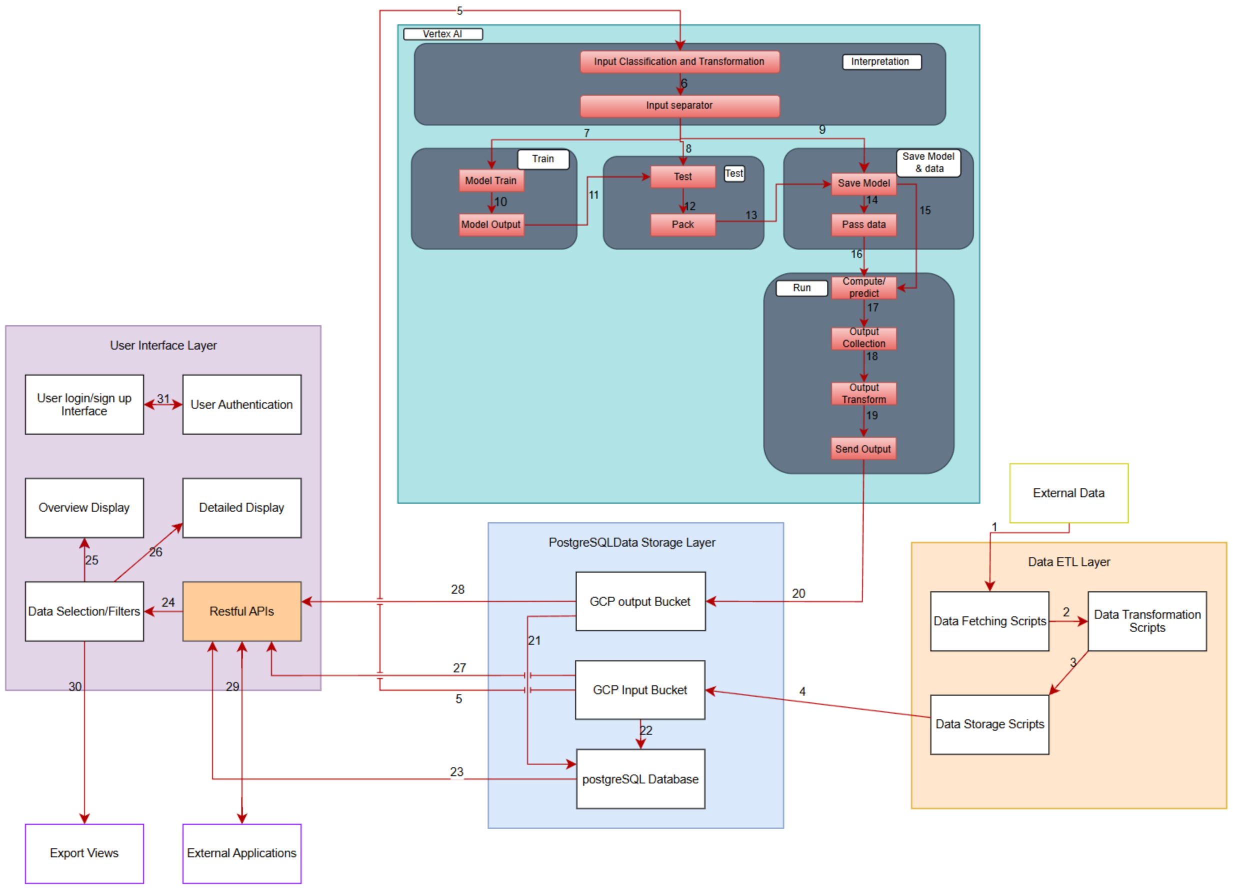


Figure 2: Data Flow Diagram

4 Data ETL Layer Subsystems

The Data ETL Layer handles the extraction, transformation, and loading of data from open source weather databases. The data such as the severe weather event (hail, thunderstorm, and tornado), the coordinates the event took place in, and the days the event occurred is fetched from open source databases. The raw data that is fetched will then be transformed using Python scripts to make the data compatible with our Machine Learning Layer. Once the raw data is transformed, it can be loaded into our Data Storage Layer.

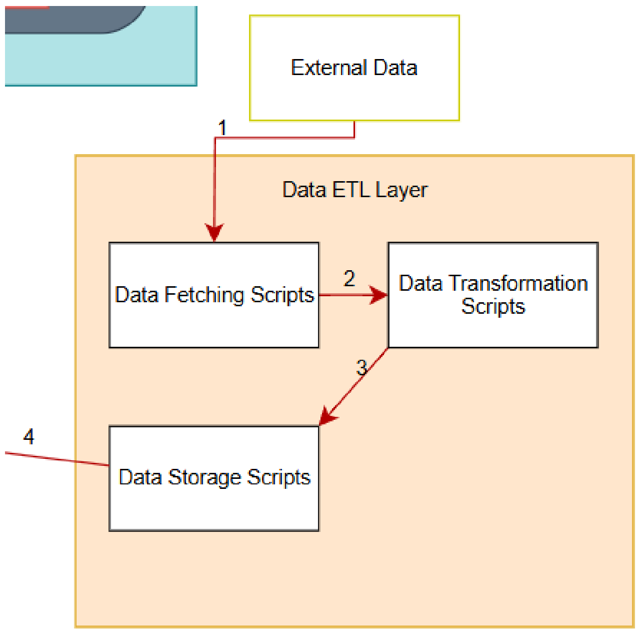


Figure 3: ETL Layer Diagram

4.1 Data Fetching Scripts

The Data Fetching Scripts Subsystem is responsible for obtaining weather data from open source databases such as NOAA and ERA5. It will fetch raw data from the databases from previous years on when the severe weather events, hail, thunderstorm, and tornadoes, have occurred.

4.1.1 Assumptions

- The subsystem assumes that the External Data will contain the coordinates of the severe weather event, the wind speeds, and the exact calendar days that the event occurred.
- It is assumed that the subsystem is able to connect to the data transformation scripts.

4.1.2 Responsibilities

The Data Fetching Scripts subsystem is responsible for ingesting the raw environmental and climate data from external sources. The external sources will provide data from previous years. The detailed responsibilities are:

- **Obtaining raw data from external sources as CSV files:** The subsystem will take in data from external sources as a CSV file by exporting the data directly from the source.

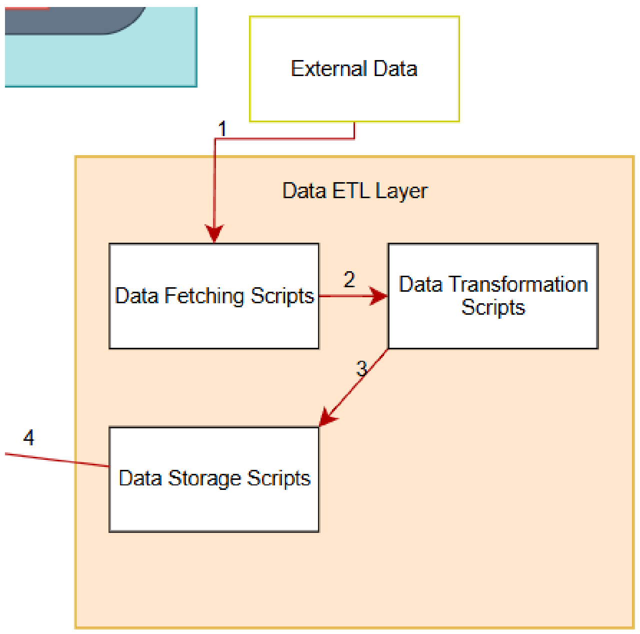


Figure 4: Data Fetching Scripts Subsystem Diagram

- **Supply CSV files to Data Transformation Scripts subsystem :** Once the subsystem is able to obtain the CSV files from the external sources, it will give them to the Data Transformation subsystem using Python scripts to make the data digestible for our model.

4.1.3 Subsystem Interfaces

Table 1: Data Fetching Scripts Interfaces

ID	Description	Inputs	Outputs
#1	Python Scripts For Fetching Data from External Sources	Raw Environmental and Climate Data from External Sources (GRIB or CSV)	CSV file given to Data Transformation subsystem

4.2 Data Transformation Scripts

The Data Transformation Scripts Subsystem handles the process of converting the raw data into a digestible format to be stored and used by the machine learning model.

4.2.1 Assumptions

- The subsystem assumes that the Data Fetching Scripts subsystem will provide raw data in a CSV file.
- The CSV file is assumed to have all the necessary data (weather event coordinates, wind speeds, and calendar date of event) to be stored in the Data Storage Scripts Layer.

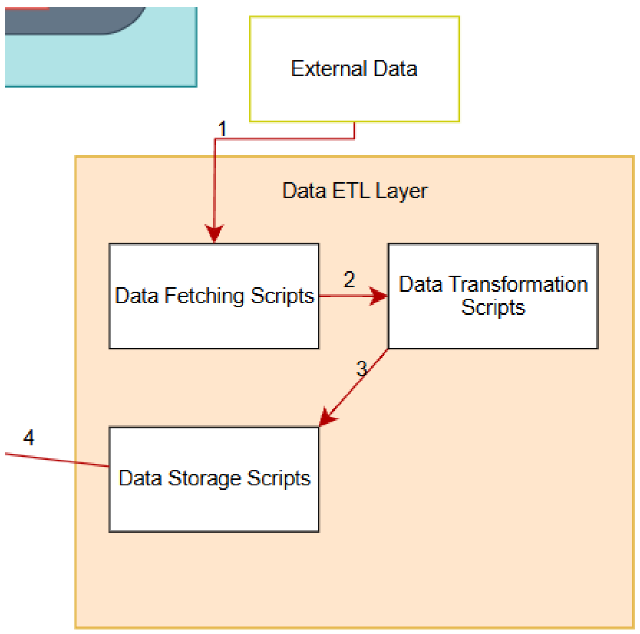


Figure 5: Data Transformation Scripts Subsystem Diagram

4.2.2 RESPONSIBILITIES

The Data Transformation Scripts subsystem is responsible for ensuring that the raw data gets transformed into a structured and enriched format that can be used by the machine learning model. In detail, the subsystem is responsible for:

- **Obtaining CSV files containing raw data from external sources from Data Fetching Scripts subsystem:** The subsystem will receive CSV files with the raw data, and it will transform that data using Python scripts to format the data to be structured in a manner that is consumable by the machine learning model. It will ensure the only data included in the file is necessary for the model to create accurate predictions.

- **Supply transformed CSV files to the Data Storage Scripts subsystem:** Once the raw data has been transformed, it will be given to the Data Storage Scripts subsystem to be stored in the Pre-Training Data subsystem. It is responsible for giving data to train the machine learning model by storing the data in a PostgreSQL database that will be accessed by the model.

4.2.3 Subsystem Interfaces

Table 2: Data Transformation Scripts Interfaces

ID	Description	Inputs	Outputs
#2	Python Scripts For Transforming Data from Data Fetching Scripts	CSV File with Raw Environmental and Climate Data	Transformed Data in CSV File
#3	Python Scripts to give Transformed Data to Data Storage Scripts Subsystem	Transformed CSV File	Prepare to Store CSV File in Database

4.3 Data Storage Scripts

The Data Storage Scripts Subsystem is where the process of putting the transformed data into the PostgreSQL Database. It closely interacts with the PostgreSQLData Storage Layer to ensure the data is being stored efficiently and can be accessed easily.

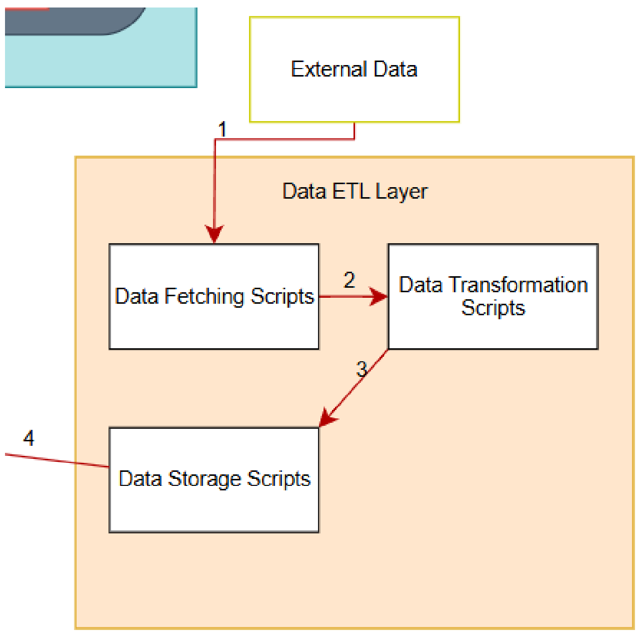


Figure 6: Data Storage Scripts Subsystem Diagram

4.3.1 Assumptions

- The subsystem assumes that the Data Transformation Scripts subsystem will provide correctly transformed CSV files that will contain sufficient data to train the machine learning model.
- The subsystem assumes it is able to connect to the database seamlessly.

4.3.2 Responsibilities

The Data Storage Scripts subsystem is responsible for storing the transformed data into the PostgreSQL database. It is meant to ensure the data is inputted into the database smoothly. More specifically, the subsystem is responsible for:

- **Obtaining Transformed CSV files Data Transformation Scripts subsystem:** The subsystem will take the transformed CSV files from the Data Transformation Scripts subsystem by using Python scripts to obtain them.
- **Input transformed data into PostgreSQl database:** The transformed data will be inputted to the PostgreSQL database using Python scripts. The data will be stored in a table to be easily accessible by the machine learning model.

4.3.3 Subsystem Interfaces

Table 3: Data Storage Scripts Interfaces

ID	Description	Inputs	Outputs
#3	Python Scripts to give Transformed Data to Data Storage Scripts Subsystem	Transformed CSV File	Prepare to Store CSV File in Database
#4	Python Scripts to store data in PostgreSQL database	Formatted Data to fill PostgreSQL Table	Store CSV File in Database

5 Data Storage Layer Subsystems

The Data Storage Layer is responsible for maintaining all structured datasets required throughout the machine learning pipeline. It consists of two GCP buckets-one for pre-processed inputs and one for model outputs-and a PostgreSQL database for relational tables and views. This layer ensures secure, persistent, and scalable file storage in the buckets, plus fast querying and indexing in the database. It provides data integrity, traceability, and accessibility for the ETL, Machine Learning, and User Interface components.

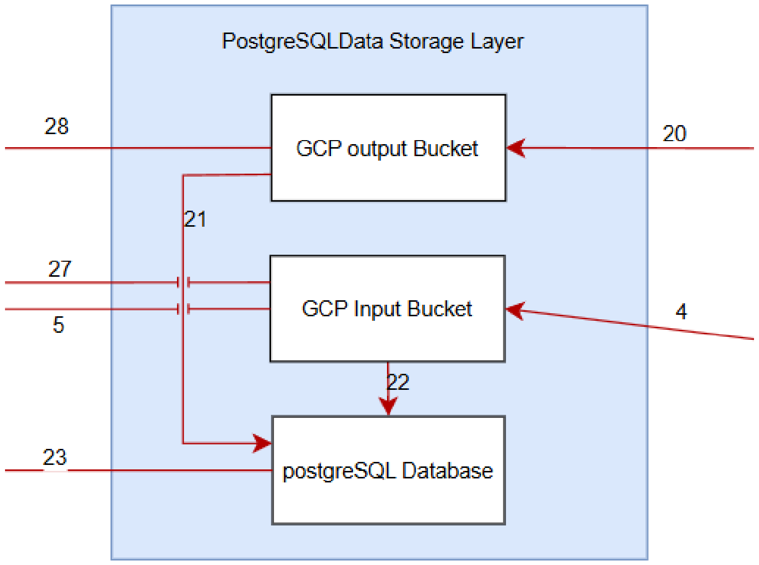


Figure 7: Data Storage Layer Diagram

5.1 GCP Input Bucket

The Input Bucket Subsystem is responsible for storing pre-processed datasets (CSV, Parquet, etc.) produced by the Data ETL Layer and consumed by the Model Training Subsystem. It provides a durable landing zone for training data files, and periodically migrates these files into PostgreSQL for optimized querying by downstream components.

5.1.1 Assumptions

- Pre-processed data files are consistently produced by the Data ETL Layer and uploaded to the bucket.
- Standardized schemas and naming conventions are enforced for all input files.
- Sufficient GCP storage and network throughput are available for large dataset transfers.

5.1.2 Responsibilities

- **Data Ingestion**: Receive and store pre-processed training datasets from the ETL stage.
- **File Durability & Access**: Ensure files are versioned and access-controlled; provide read URLs or IAM roles to the Model Training Subsystem.
- **Data Migration**: Sync or load files into PostgreSQL tables and views for fast analytical queries.

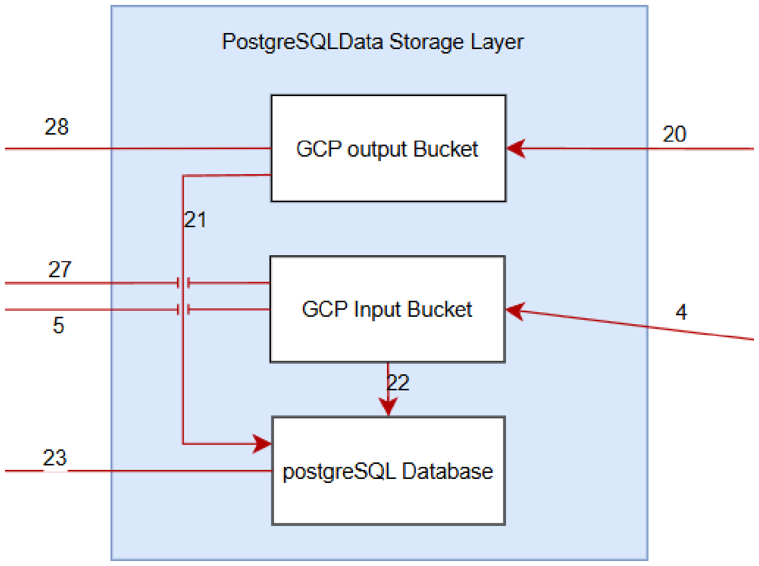


Figure 8: GCP Input Bucket Subsystem Diagram

5.1.3 Subsystem Interfaces

Table 4: Subsystem Interfaces

ID	Description	Inputs	Outputs
4	Data ETL to Input Bucket: upload of pre-processed files	Transformed data files (CSV, Parquet)	Stored files in GCS
5	Input Bucket to Model Training: data consumption	GCS file URIs	Training job input streams
22	Input Bucket to PostgreSQL: batch or streaming ingestion	GCS file URIs	Populated relational tables

5.2 GCP Output Bucket

The Output Bucket Subsystem serves as the landing zone for model prediction outputs. After each run in the Machine Learning Layer, result files (e.g., JSON risk scores, CSV hazard predictions) are written here, then migrated into PostgreSQL for fast front-end querying and filtering. It ensures consistency in file format and labeling for downstream visualization.

5.2.1 Assumptions

- ML outputs are generated in a standardized, self-describing file format (JSON, CSV).
- Outputs are validated by the ML Layer before being stored.
- Reliable network connectivity exists for bucket-to-database migrations.

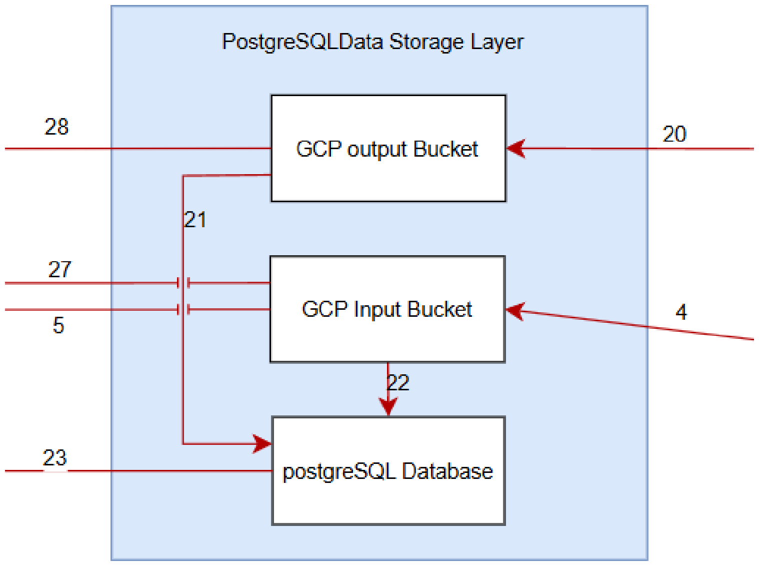


Figure 9: GCP Output Bucket Subsystem Diagram

5.2.2 Responsibilities

- **Output Collection**: Receive and persist model run results as files in GCS.
- **Formatting & Labeling**: Embed metadata (timestamps, model version, region) in each output file.
- **Versioning & Archival**: Retain historical outputs for traceability and rollback.
- **Data Migration**: Sync output files into PostgreSQL tables and views for UI consumption.

5.2.3 Subsystem Interfaces

Table 5: Subsystem Interfaces

ID	Description	Inputs	Outputs
20	ML Layer to Output Bucket: store prediction files	JSON/CSV result files	Stored files in GCS
28	Output Bucket to UI Layer: file access for display	GCS URLs of result files	Front-end file streams
21	Output Bucket to PostgreSQL: ingestion of result data	GCS file URIs	Populated result tables/views

5.3 PostgreSQL Database Subsystem

The PostgreSQL Database Subsystem provides relational tables and materialized views for fast querying and analytics. It consolidates both input and output bucket files into structured schemas, supports complex joins and filters, and serves data to the ML Layer and UI Layer with low-latency SQL.

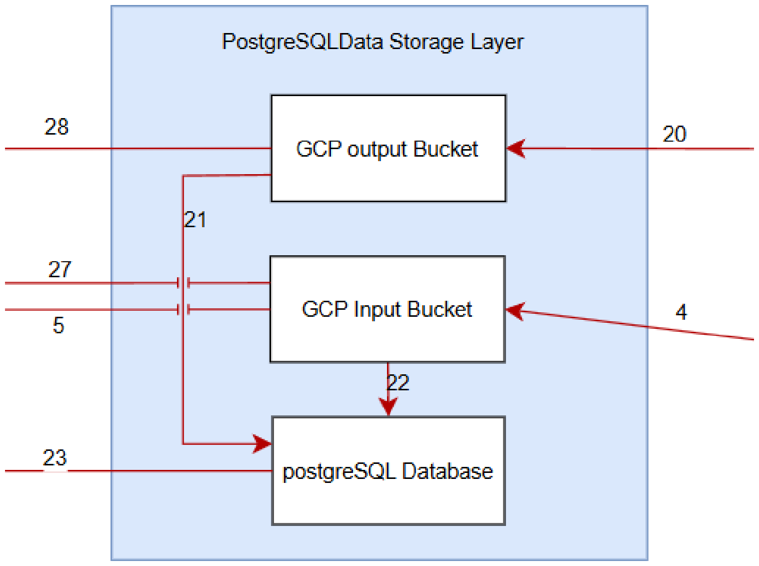


Figure 10: PostgreSQL Database Subsystem Diagram

5.3.1 Assumptions

- Bucket-to-database ingestion jobs run reliably on schedule or triggered by file events.
- Database schemas and indexes are tuned for expected query patterns.
- Sufficient compute and storage are provisioned in Cloud SQL (or self-managed) for peak workloads.

5.3.2 Responsibilities

- **Data Ingestion**: Load and transform files from both GCP buckets into normalized tables.
- **Indexing & Optimization**: Create indexes, materialized views, and partitions to accelerate complex queries.
- **Query Serving**: Respond to SQL queries from the Machine Learning layer (for feature extraction) and the UI layer (for filters, maps, charts).
- **Data Governance**: Enforce access controls, backups, and retention policies on stored tables and views.

5.3.3 Subsystem Interfaces

Table 6: Subsystem Interfaces

ID	Description	Inputs	Outputs
22	Input Bucket to Postgres: ingest training data files	Pre-processed files in GCS	Training tables
21	Output Bucket to Postgres: ingest prediction files	Model output files in GCS	Result tables/views
23	Postgres to UI Layer: filter/query data	UI-driven SQL queries	UI result sets

6 Machine Learning Layer Subsystems

In the ML layer, we will use Google Cloud's Vertex AI to interpret, Train, Test, save, and Run the model. It will either output or train the model depending on the input from the UI. It will also save multiple models, and depending on the input, it will choose the right model for the job and output to the SQL after line per line from the CSV

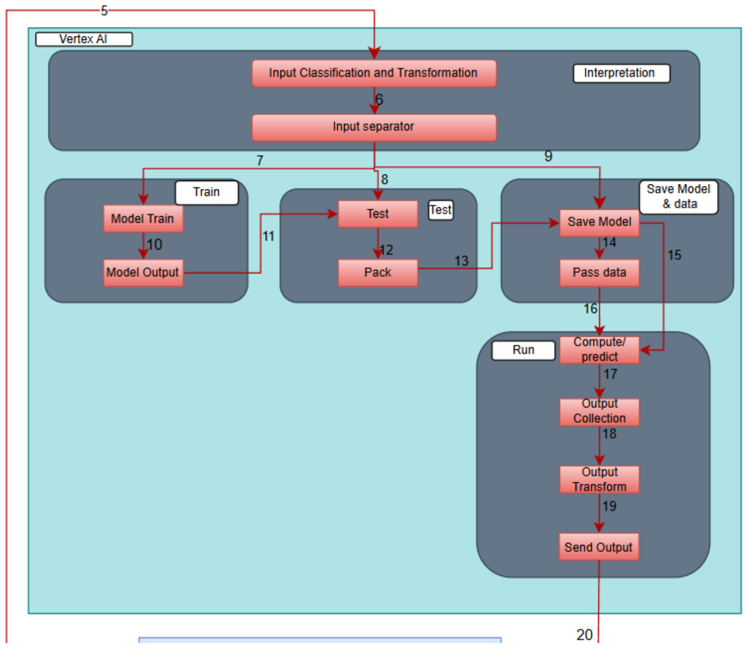


Figure 11: Machine Learning Layer

6.1 Interpretation

Interpretation is responsible for getting the input from the PostgreSQL storage layer (Pre-training module) and feeding it either to the Training module or the "Save Model" based on user input (Train model or run model).

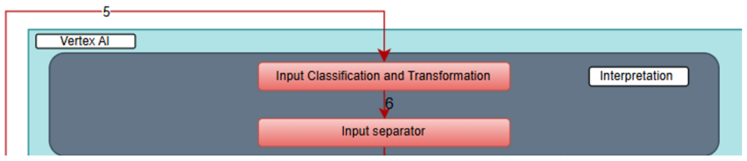


Figure 12: Interpretation Sublayer

6.1.1 Assumptions

- user will decide what job the model will run (Training or Running)
- it also assumes that the necessary data is available to use in the PostgreSQL Storage layer

6.1.2 Responsibilities

- Input Classification: Determining the type of input (Training or Running), Correcting input from PostgreSQL and Adding input specific flags on input for separator and send input

- Input Separator: provide data to the training module and Save Model module, for training it divides the data for training and Testing and sends data to the train module and test module for training. For prediction, sending all data to the "Save Module"

6.1.3 Subsystem Interfaces

Table 7: Subsystem interfaces

ID	Description	Inputs	Outputs
#5	Data coming from the GCP input bucket to Interpretation's Input classification module	Data in JSON to input classification	Output in CSV after collecting and setting up flags for separator
#6	Flagged data from Input Classification to Separator unit	Flagged CSV	Train and Test gets fixed amount training and Testing data, Save model unit gets the whole data
#7	Fixed amount of training data in CSV	Input separator flagged csv	Train Module, model train unit
#8	Flagged Test data in CSV	Input separator flags test data	Test module's Test unit gets the CSV to Testing
#9	Flagged Input data for save model and data module's save model unit in CSV format.	Input data in CSV from Input Separator	CSV Input data to save model unit in save module.

6.2 Train

The Train subsystem's task is to train the model using the input from the Interpretation module and send the model to "Model Output." The model output is then sent to Testing.

6.2.1 Assumptions

- Model Train: Assumes that the Interpretation module gives the correct data so the model can learn features

- Model Output: Expects a model that is trained and ready to be tested

6.2.2 Responsibilities

- Model Train: Responsible for giving off trained model along with metadata, such as input dimensions, number of the epoch, layers, units, etc to Model Output

- Model Output: Responsible for adding the metadata given by Model Train, appropriately naming model file name and handing it to Testing

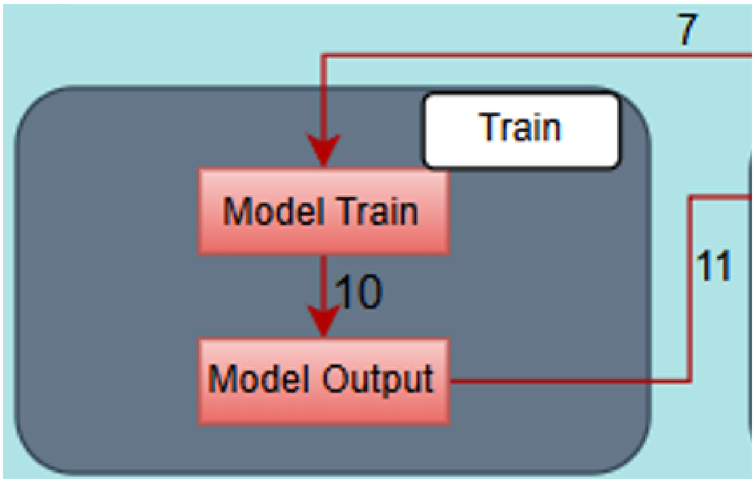


Figure 13: Training Sublayer

6.2.3 Subsystem Interfaces

Table 8: Subsystem interfaces

ID	Description	Inputs	Outputs
#7	Training data from Interpretation's Separator unit for training model	Train data from separator module	Metadata to Model output for adding additional Metadata
#10	Model file and metadata CSV	Model file and metadata	packed metadata and changed file name
#11	Changed filename and additional metadata for test module	changed file name and metadata to Test module's Test unit	To Test for Testing

6.3 Test

Test Subsystem tests the model for accuracy and correct outputs; it then sends the model to Pack, which adds all the details like accuracy, number of test inputs, test outputs, etc.

6.3.1 Assumptions

- Test: Assumes that the model name and model compilation are correct for Testing using input from Interpretation
- Pack: Assumes that all the metadata from model and Test is provided for packing everything in a file

6.3.2 Responsibilities

- Test: Responsible for testing the model using the input from the Interpretation module, sending additional metadata along with model metadata data to Pack module

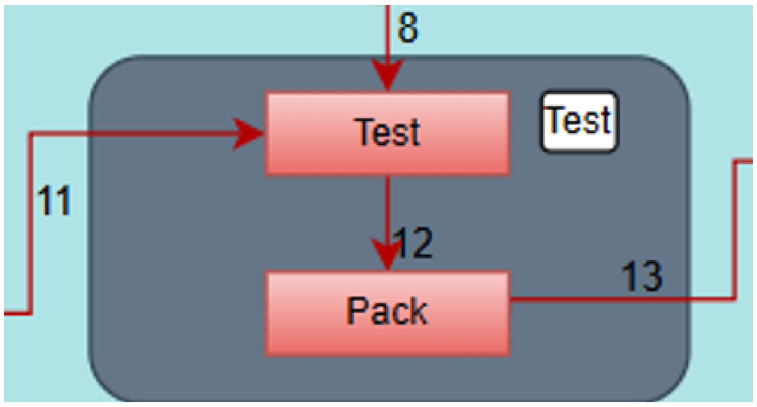


Figure 14: Testing Sublayer

- Pack: Responsible for adding the metadata given by Test and sending it over to Save Model and data

6.3.3 Subsystem Interfaces

Table 9: Subsystem interfaces

ID	Description	Inputs	Outputs
#8	Flagged Test data in CSV	Input separator flags test data	Test module's Test unit gets the CSV to Testing
#11	Changed filename and additional metadata for test module	changed file name and metadata to Test module's Test unit	To Test for Testing
#12	Test data and model metadata	from Test unit it test module	to Pack Unit in test module for packing it all in one package
#13	model file to save model and additional test data	From pack unit of test	To Save Model and data

6.4 Save Model and Data

Save Model saves the model file and passes input from the Interpretation module and model file to the Run module. This way, all the trained models can be saved for later use.

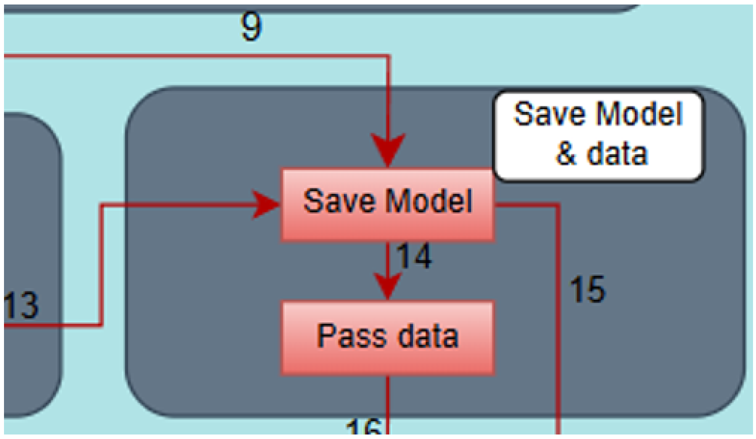


Figure 15: Save Model and Data Sublayer

6.4.1 Assumptions

- Save Model: Assumes that Interpretation gives correct and relevant input data, and the model has no errors.
- Pass data: assumes that Save module gives data for it to log input metadata correctly

6.4.2 Responsibilities

- Save Model: saves the Model given by the Train module for later use and passes the model to the Run module to run it; it also passes on the input received from the Interpretation module to Pass Data
- Pass data: responsible for passing model input data to Run module and logging metadata of input

6.4.3 Subsystem Interfaces

Table 10: Subsystem interfaces

ID	Description	Inputs	Outputs
#9	Flagged Input data for save model and data module's save model unit in CSV format.	Input data in CSV from Input Separator	CSV Input data to save model unit in save module.
#13	model file to save model and additional test data	From pack unit of test	To Save Model and data
#14	Pass data from Interpretation module	Original Input data	Original Input data
#15	Model File for running the model	file with Keras extension model file given to Run module's Compute unit	Outputs the prediction from Compute unit
#16	Passes the input data after logging the input metadata (i.e., dimensions)	adds dimensions data and sends to compute	sends to compute Unit in run module

6.5 Run

Run Module runs the model using input data given by Save Module and transforms the data to put it back in the PostgreSQL data storage layer.

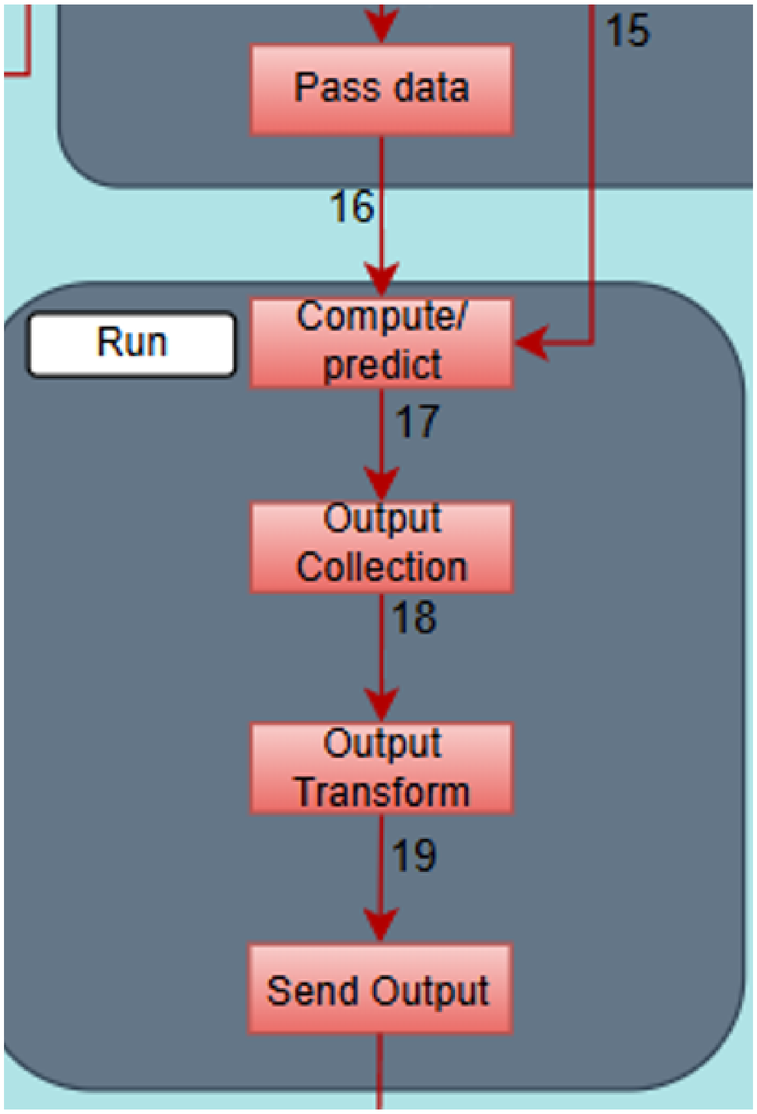


Figure 16: Model Run Sublayer

6.5.1 Assumptions

- Compute: Compute assumes that the correct model file and data is given for it to compute the output; it also assumes that necessary computing power is given
- Output Collection: all the data output from computing is in a format that can be sent to transform. it also assumes that no data is skipped while computing
- Output Transform: Assumes that when the data is fetched from the collection module, it includes correct data for the input start and end, along with other metadata such as runtime of each data vector

- Send Output: assumes that the transform spits out a file that has all the data, including metadata such as date and time needed for sending JSON to PostgreSQL

6.5.2 Responsibilities

- Compute: responsible for running the model and outputting each line of predicted data
- Output Collection: Responsible for collecting all the output from compute and putting it in a file along with runtime metadata
- Output Transform: Responsible for parsing and adding necessary columns for the database stripped in the collection.
- Send Output: Responsible for sending chunks (rows) of data in a JSON format to PostgreSQL module

6.5.3 Subsystem Interfaces

Table 11: Subsystem interfaces

ID	Description	Inputs	Outputs
#15	Model File for running the model	file with Keras extension model file given to Run module's Compute unit	Outputs the prediction from Compute unit
#16	Passes the input data after logging the input metadata (i.e., dimensions)	adds dimensions data and sends to compute	sends to compute Unit in run module
#17	Output of the model per line	To output collection for collecting all the inputs	Collect all and output in a csv
#18	CSV of model output	to output transform for change format that matches PostgreSQL module	Output transform changes the output and puts it in CSV and passes it to send output
#19	CSV with model output data that is transformed for front end requirements such as removing 0 flags or creating JSON objects for UI components	CSV from Output Transform	Send output gets CSV or JSON for sending output to output bucket
#20	Contains JSON or CSV	JSON or CSV preprocessed for front end components	JSON or CSV preprocessed for front end components

7 User Interface Layer Subsystems

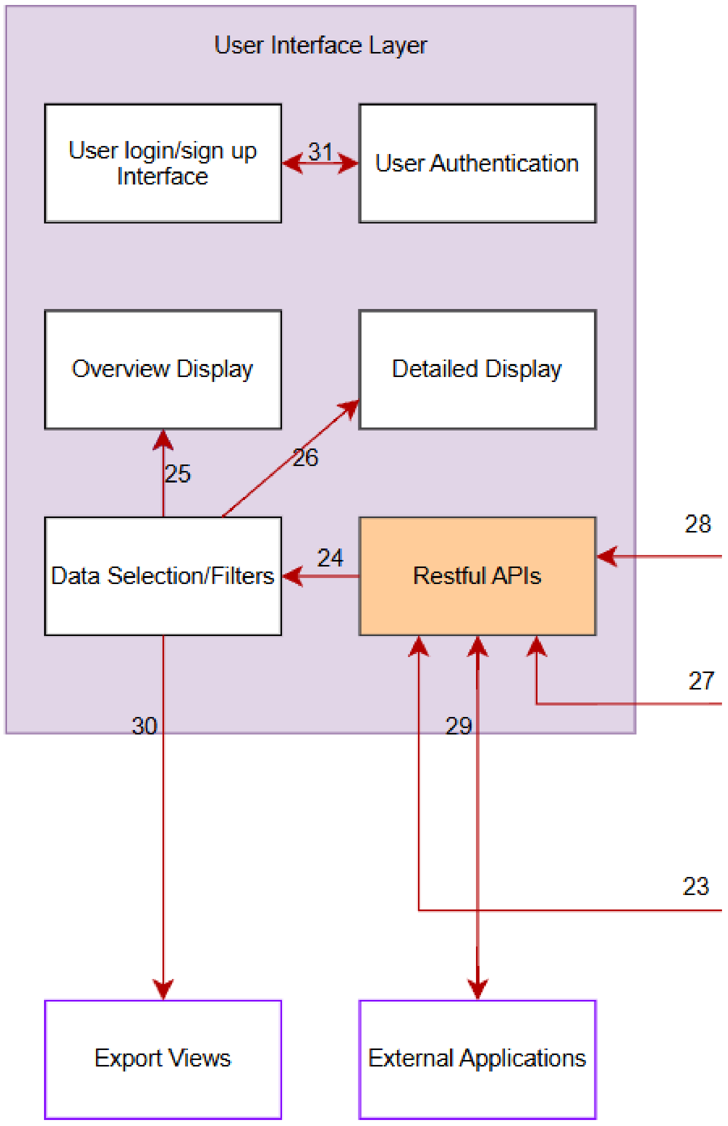


Figure 17: User Interface Layer Diagram

7.1 Subsystem 1: User login/sign up Interface

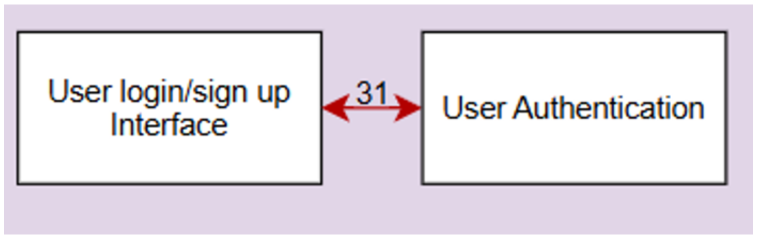


Figure 18: User login/sign up Sublayer Diagram

7.1.1 Assumptions

- Assumes valid input is entered by the user.
- Communicates directly with the User Authentication subsystem.
- Data is sent securely using encrypted channels.

7.1.2 Responsibilities

- Provide a UI for users to log in or register.
- Collect and validate user input before submission.
- Forward credentials to the authentication module.
- Display feedback messages (errors, confirmations).

7.1.3 Subsystem Interfaces

Table 12: Subsystem interfaces

ID	Description	Inputs	Outputs
#31	Interface with User Authentication for credential validation	Username, password, registration data	Auth token or error/success status

7.2 Subsystem 2: User Authentication (via Firebase)

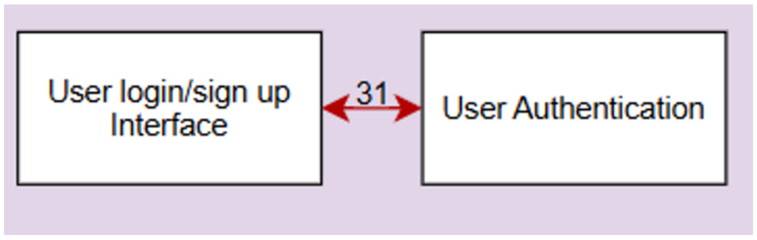


Figure 19: User Authentication Sublayer Diagram

7.2.1 ASSUMPTIONS

- Firebase Authentication will be used for handling all login and registration logic.
- Frontend connects securely to Firebase using official SDKs.
- Firebase handles credential storage, password hashing, and session management.

7.2.2 RESPONSIBILITIES

- Leverage Firebase Authentication to sign in, register, and manage user sessions.
- Provide secure login/signup functionality using email/password and optionally third-party providers (Google, etc.).
- Return user ID tokens to frontend for session tracking and authorization.
- Support error feedback for failed login attempts or registration issues.

7.2.3 SUBSYSTEM INTERFACES

Table 13: Subsystem interfaces

ID	Description	Inputs	Outputs
#31	Firebase Auth API via SDK	Email/password credentials	Firebase ID token or error message
–	Session management through Firebase	Refresh tokens	Auto-login or logout behavior

7.3 Subsystem 3: Overview Display

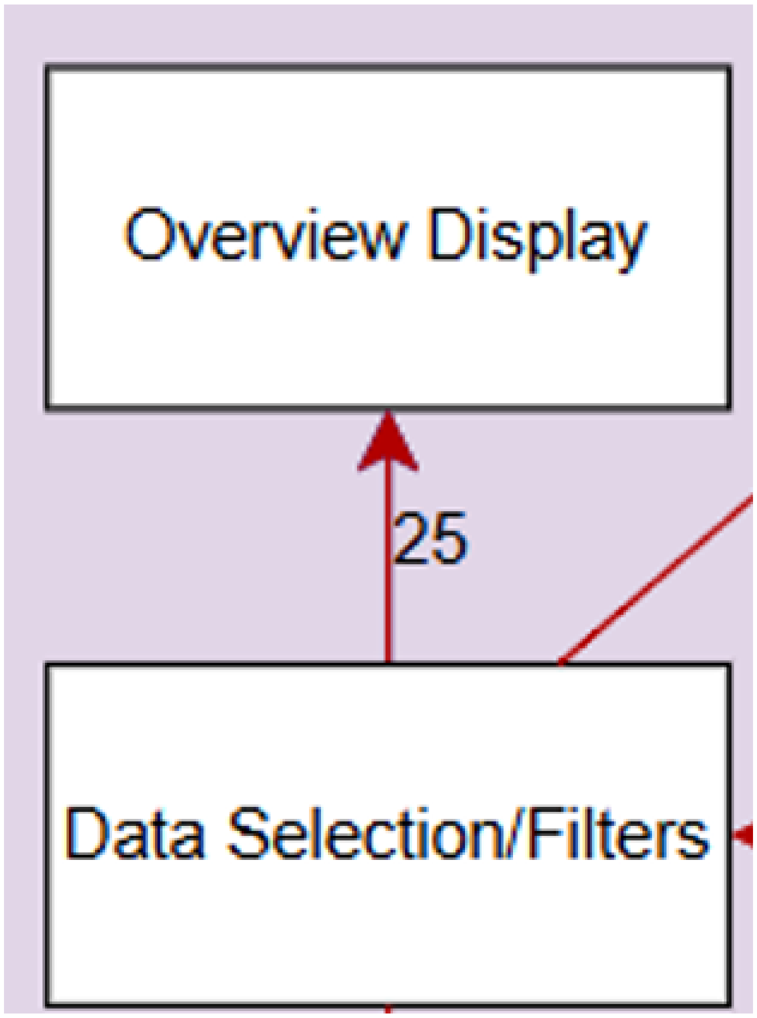


Figure 20: Overview Display Sublayer Diagram

7.3.1 Assumptions

- Depends on input from data filtering subsystem.
- Geospatial data is structured and compatible with mapping library.

7.3.2 Responsibilities

- Render visual representation of data on the map.
- Respond dynamically to changes in filters or selections.
- Support interactive exploration of map elements.

7.3.3 Subsystem Interfaces

Table 14: Subsystem interfaces

ID	Description	Inputs	Outputs
#25	Filtered data passed from Data Selection/Filters	Filter parameters, dataset	Rendered map visuals
–	UI controls for zoom/pan/layers	Mouse/keyboard events	Updated map display

7.4 Subsystem 4: Detailed Display Interface

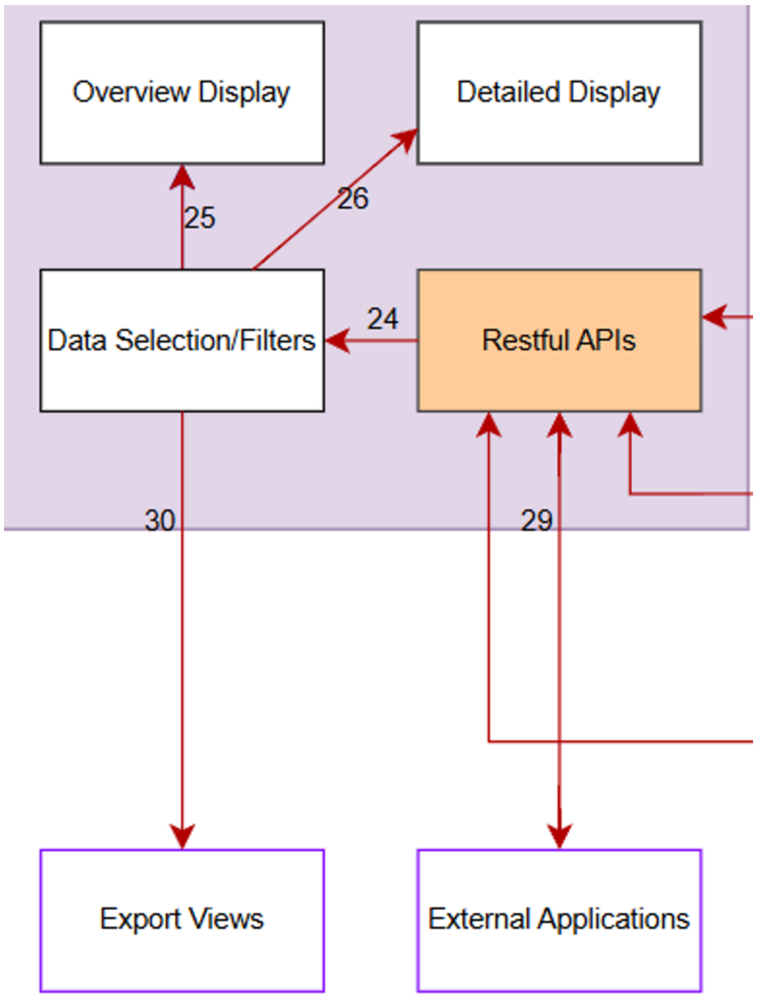


Figure 21: Detailed Display Sublayer Diagram

7.4.1 Assumptions

- Depends on input from data filtering subsystem.

- Geospatial data is structured and compatible with mapping library.

7.4.2 Responsibilities

- Render visual representation of data on the map.
- Respond dynamically to changes in filters or selections.
- Support interactive exploration of map elements.

7.4.3 Subsystem Interfaces

Table 15: Subsystem interfaces

ID	Description	Inputs	Outputs
#26	Receives refined filter selections to internal UI components	User data selections/filters	Filtered query output displayed on front end components using JSON objecct

7.5 Subsystem 5: Data Selection/Filters

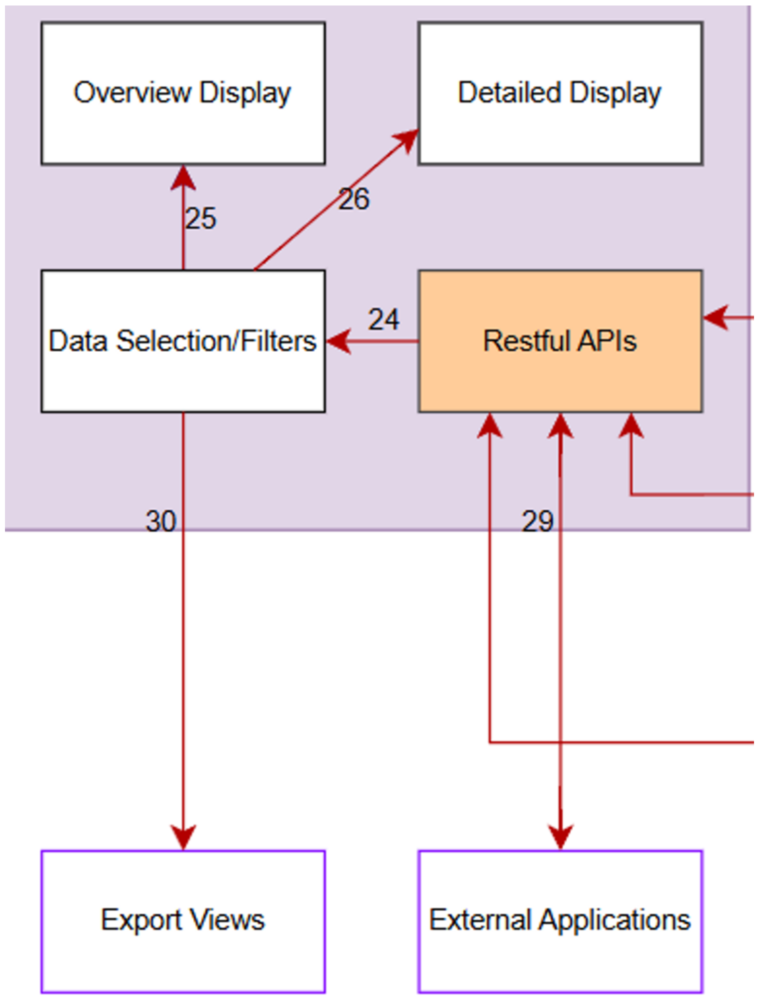


Figure 22: Data Selection/Filter Sublayer Diagram

7.5.1 Assumptions

- Dataset is available and up to date.
- Filter options reflect user needs and dataset structure.

7.5.2 Responsibilities

- Provide dropdowns, checkboxes, sliders to refine dataset.
- Process and send filter criteria to Map Display.
- Allow real-time updates to filtered results.

7.5.3 SUBSYSTEM INTERFACES

Table 16: Subsystem interfaces

ID	Description	Inputs	Outputs
#24	Receives data filters from External Applications	Filter config/metadata	Transformed filtering options
#26	Sends refined filter selections to internal UI components	User interactions	Filter query
#25	Filtered dataset interface to Map Display	User-selected filters	Refined dataset
–	User interactions with UI elements	Clicks, selections	Filter parameters

7.6 Subsystem 6: RESTful APIs

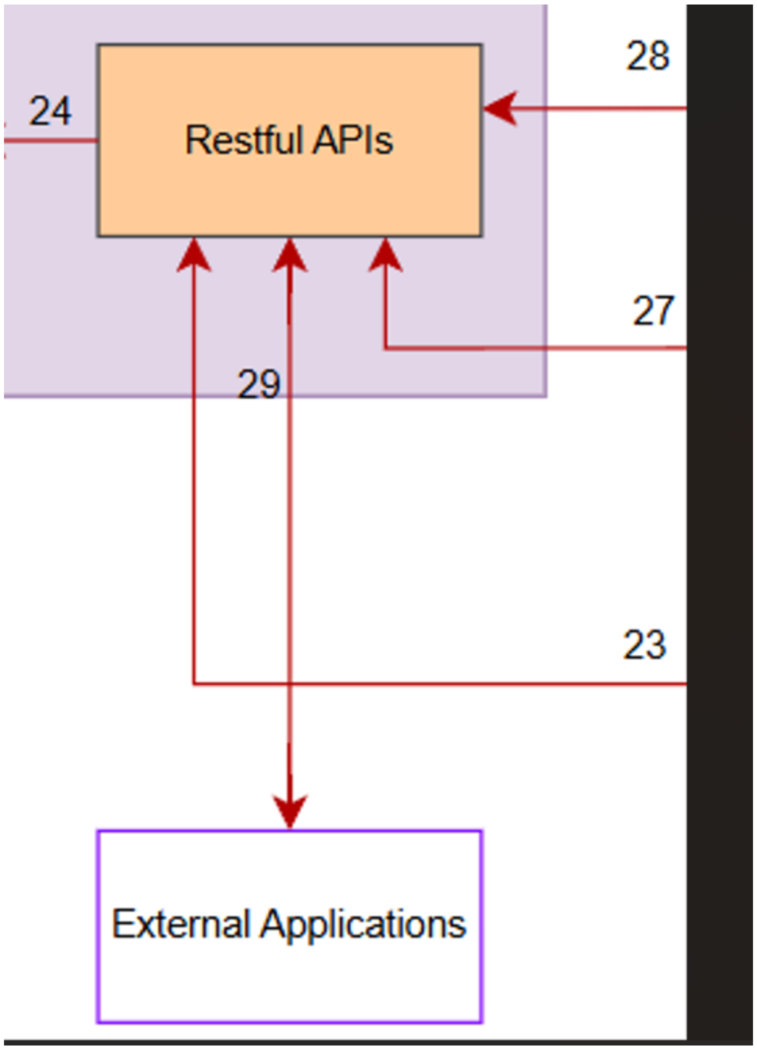


Figure 23: RESTful API Sublayer Diagram

7.6.1 Assumptions

- All APIs are compliant with REST standards.
- Backend services are reachable and authenticated.

7.6.2 Responsibilities

- Provide endpoints for registration, data retrieval, and model update.
- Accept requests from front-end subsystems and return appropriate responses.
- Handle data validation, error reporting, and access control.

7.6.3 Subsystem Interfaces

Table 17: Subsystem interfaces

ID	Description	Inputs	Outputs
#24	Interfaces with UI components for API calls	RESTful requests (GET, POST, PUT)	JSON responses
#29	Handles API calls from external applications	Authenticated HTTP requests	Processed output or error message
#27	Retrieves Data for GCP output Bucket for UI components	CSV or JSON	JSON
#28	Retrieves Data for GCP input Bucket for UI components	CSV or JSON	JSON
#23	Retrieves Data for GCP PostgreSQL instance for UI components	SQL	JSON

References