CS 6242 Final Report: TRiBX: Interactive Real-Time Activity Finder with Event Clustering

Luis A Tupac **Ritesh Vadodaria** **Benjamin Underwood** **Xukun Zhang**
Georgia Institute of Technology
ltupac3@gatech.edu rvadodaria6@gatech.edu bunderwood31@gatech.edu xzhang3151@gatech.edu

Abstract

Our project, Activity Finder, is an interactive tool that helps users discover nearby events based on interests, location, and weather conditions. Our app automates event categorization and delivers real-time information, integrating factors such as time of day and weather. By clustering events by type and proximity and providing dynamic visualizations on an interactive map, the app caters to locals, travelers, and spontaneous users

1. Problem Introduction

Cities are becoming more dynamic, making it harder to find relevant activities and events in real-time. Current apps allow users to find current events only by searching popular venues, then require them to visit individual websites to receive information about what is happening around them. Furthermore, their showings typically only include the most popular, ticketed events. Activity Finder is an interactive tool that helps users discover nearby events based on their interests, location, and the weather. Users can filter events by type, time, or distance, while the app uses historical weather data and forecasts to alert them to possible disruptions. It introduces a range of events, from popular concerts to local recreational meetups.

2. Project Github Link

Project GitHub

3. Literature Survey

Local Event Detection Scheme by Analyzing Relevant Documents in Social Networks The paper presents methodology for detecting local events by leveraging geographical data and social network documents like posts, comments and threads. It gave us insight on how to validate our model, and replicate the keyword graph with our data. [1]

Intelligent Event Finder and Management System The paper aims to develop a recommendation system for finding events and managing logistics, whether users are hosting or joining. The event registration system automates processes such as ticketing, registration, posting, and certificate issuance. We referenced the front-end design from the paper and improve the user experience for event recommendations. [2]

Geospatial Keyword Graph for Extracting Spatial Information from Social Media Data The paper proposes a method for extracting and visualizing geo-information from social media. A keyword based graph to build a structure that connects places, events, and users. We gained insight from the detection and spatial analysis framework from there improved it by including more advanced visualization. [3]

Atmosfy: Strategic Guide on How to Build a Great Product Roadmap Atmosfy is an app that allows users to discover new places of interest (mainly dining) by video shares of past customers to each of the businesses. Their social media style of location sharing is user friendly, but their map only shows established businesses. Our website offers similar, including live events. [4] *A GIS methodology for the analysis of weather radar precipitation data.* This paper used ARCGis to process and analyze weather data. The interface gave better and more user-friendly rainfall predictions. Our performs similarly, considering rainfall patterns to re-suggest events depending on how extreme outdoor weather will impact an experience. [5] *Dynamic and accurate location college bus tracking System using Google API.* This system gave accurate real-time location information for college buses, ensuring students would have the most information to reach their classes on time. The college bus tracking system used Google API to accurately track bus movements and provide precise location updates. It also added a feature that allowed students to track where the buses were manually, which was then broadcast to other tracking system users, improving accuracy. [6]

Aurigo, TripBuilder Aurigo and TripBuilder both provide innovative approaches to creating personalized, preplanned itineraries. Aurigo uses POI (Points of Interest) data from sources like Yelp and Google Maps to help users explore and optimize tour paths based on geographic proximity, with visualization tools like the ”Pop Radius.” [7] Similarly, TripBuilder leverages crowd-sourced geo-tagged photos from Flickr to optimize tourist itineraries, using the Generalized Maximum Coverage (GMC) model to maximize user satisfaction by covering popular points of interest. Both papers offer valuable insights into how user preferences and location-based services can enhance event discovery, which we applied to improve our app’s real-time clustering and event recommendation algorithms. [8] *Event Aware* Focuses on conferences, using a tag-based algorithm to recommend sessions and exhibitors based on real-time contextual factors like location and time. While this system personalizes agendas for conference attendees, it remains limited to fixed venues and specific types of events. [9] Use of real-time data and contextual filtering provided intuition to our approach to event recommendations based on dynamic factors such as weather and time windows, helping us make the user experience more adaptive.

DuckDB:An embeddable, lightweight analytical database, optimized for read-heavy workloads with columnar support performance optimal for tasks like feature extraction. The engine currently supports parallel execution for inter-query workloads. [10] *Density-Based Spatial Clustering of Applications with Noise (DBSCAN)* is a location based clustering method, which helps in identifying clusters of arbitrarily shape. We plan to cluster events by location to create a richer experience and avoid overwhelming users within a set radius. A graph based DBSCAN is reliable as we can set constraints not confined to typical geometric shapes. [11] *Efficient Estimation of Word Representations in Vector Space*The paper presents neural network architectures for efficient word representation as N-dimensional vectors, enabling semantic comparison. Project plans to tag event using similarity matches of vector representation. High dim vectors are more accurate but adds to computation cost.

4. Architecture Diagram

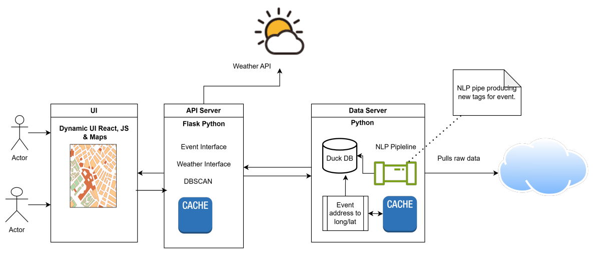


Figure 1. Architecture and Tech Stack

5. Design Details of Tiers

5.1. UI & Map Framework

We leverage **Mapbox** and **React** to create a responsive, interactive, and filterable map visualization. The main components include:

5.1.1 Tile Layer and OpenStreetMap Map

A tile layer from OpenStreetMap is added to the map, rendering the base map with elements such as streets, rivers, buildings, etc., providing essential geographical context. To enhance the user experience, a loading screen animation was implemented to inform the user while the map is loading and during DBSCAN clustering operations.

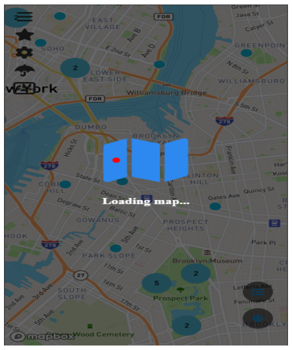


Figure 2. Loading animation on Map loads.

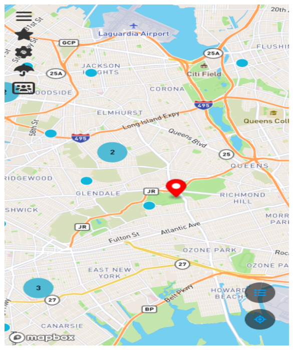


Figure 3. Current Events Map showing activities in Queens are of New York.

5.1.2 Marker Clustering

When multiple event markers are positioned close to each other, they appear as a single clustered marker based on the zoom level. As users zoom in, the clustered markers separate into individual markers, allowing users to view each event independently. This interaction enhances readability and performance in areas with dense event locations. Users can select a cluster zone within their area of interest before exploring individual events in detail.

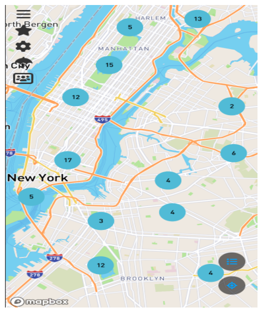
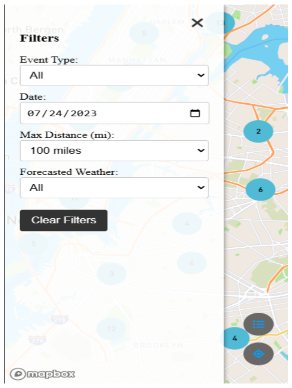


Figure 4. Current Events Map showing clustered markers in Brooklyn and Manhattan areas.

5.1.3 Dynamic Data Rendering and Filtering

A JSON file retrieved from a server call to the database dynamically updates the displayed markers without reloading the map. This approach enables users to stay up-to-date on events near their point of interest. The *Haversine formula* is used to calculate distances and filter events by proximity, displaying only those events within the specified maximum distance.


Filters

Event Type: All

Date: 07/24/2023

Max Distance (mi): 100 miles

Forecasted Weather: All

Figure 5. Filtering options for event type, date, distance, and weather conditions

5.1.4 DBSCAN Clustering

DBSCAN clustering is triggered by the DBSCAN button in the UI (below the umbrella icon). When activated, the application queries the dedicated API DBSCAN endpoint to generate new clustered data with a cluster ID field. This dataset is returned to the UI, where it dynamically updates the map markers, changing their colors based on their assigned cluster IDs. This visual differentiation makes it easier for users to identify clusters and explore related events within each group.

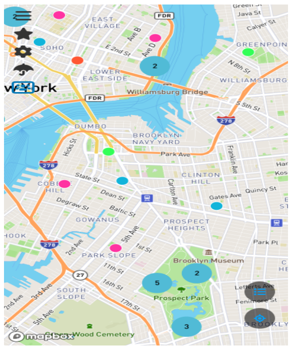
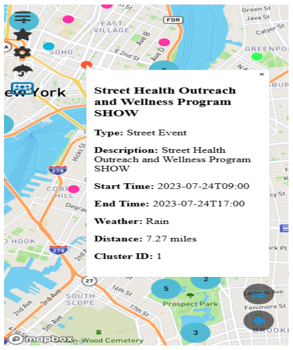


Figure 6. Markers grouped by color after DBSCAN.


Street Health Outreach and Wellness Program SHOW

Type: Street Event

Description: Street Health Outreach and Wellness Program SHOW

Start Time: 2023-07-24T09:00

End Time: 2023-07-24T17:00

Weather: Rain

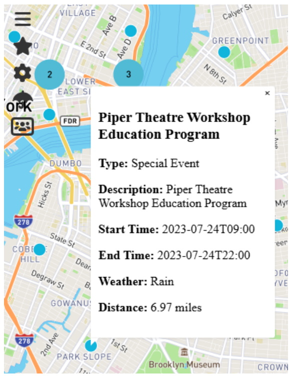
Distance: 7.27 miles

Cluster ID: 1

Figure 7. Cluster ID being displayed

5.1.5 User Location, List View, and Custom Buttons

The user's current location or selected point of interest is represented by a custom icon. Each event is displayed as a Leaflet marker, and we are customizing markers based on event type. Unique marker icons increase the map's visual appeal and facilitate easier event selection. Each marker includes popup bindings to display event information (e.g., **name, type, description, start/end times, weather, and distance**). When users click on a marker, a popup appears, providing quick access to event details without navigating away from the map. To improve event organization, we implemented a list view option that displays events in a structured, scrollable format. Users can toggle between the list view and the map view using the custom list button on the bottom right. This toggle is seamless and does not require any re-rendering or any downtime of the application. To help users navigate the map and quickly find their location, we implemented a "Locate Me" button represented by the target icon on the bottom right. This button provides a seamless and smooth transition by zooming in on the user's current location, regardless of where they are on the map. This feature enhances usability and ensures users can reorient themselves effortlessly while exploring events.


Piper Theatre Workshop Education Program

Type: Special Event

Description: Piper Theatre Workshop Education Program

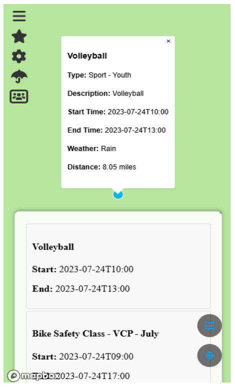
Start Time: 2023-07-24T09:00

End Time: 2023-07-24T22:00

Weather: Rain

Distance: 6.97 miles

Figure 8. Event details and icon design.


Volleyball

Type: Sport - Youth

Description: Volleyball

Start Time: 2023-07-24T10:00

End Time: 2023-07-24T13:00

Weather: Rain

Distance: 8.05 miles

Volleyball

Start: 2023-07-24T10:00

End: 2023-07-24T13:00

Bike Safety Class - VCP - July

Start: 2023-07-24T09:00

2023-07-24T17:00

Figure 9. Events organized in scrollable list

5.2. API Server Framework

We are hosting a local server to manage the connection between the frontend visualization and the "gold layer" table in the database. Two main API endpoints have been defined as follows:

5.2.1 Filter Data Endpoint

- **Resource Path**: `<host>/api/run-dbscan`
- **Method**: POST
- **Description**: Runs DBSCAN on events within a specified datetime and distance from user location.

- **Response**: A JSON object with two sections:
  - **Events**: Array of filtered events within the boundary and datetime, including:
    - `event_name`
    - `type/tag`
    - `description`
    - `start`
    - `end`
    - `cluster_id`
  - **Weather Information**: Weather details for the user's location, such as temperature, rain, date time.

5.2.2 Get Filtered Data Endpoint

- **Resource Path**: `<host>/api/get-events`
- **Method**: GET
- **Description**: This endpoint provides a predefined set of event data, useful for quickly populating the frontend UI with markers to display.
- **Response**: A JSON object containing:
  - **Sample Events**: Array of filtered events within the boundary, including:
    - `event_name`
    - `type/tag`
    - `description`
    - `start`
    - `end`
  - **Weather Information**: Weather details for the user's location, such as temperature, rain, date time.

5.2.3 Weather API

We used the Weatherstack API to collect real-time and historical weather data, retrieved one day at a time. Our Python script organizes the data into columns like Location, Country, Region, Coordinates, Timezone, Time, UTC Offset, Temperature, and Weather Code. This enables precise weather insights to enhance user event experiences. Weatherstack offers a free API reporting current weather conditions for any latitude and longitude combination in the world. We classify three-number weather code into four categories: Sunny, Windy, Rainy, and Snowy. Our code calls the lat/long combination from the API key and sections out the weather code, classified into one of the four groups. The front-end server takes the weather condition and applies the appropriate overlay to the UI, showing users if their outdoor events may be rained out. K-Means event type clustering is also affected, as the algorithm will begin to suggest events indoors that a user may prefer at the current moment.

5.2.4 DBSCAN(Density-Based Spatial Clustering of Applications with Noise) on selected locations

Once the points of interest are retrieved by querying a database with specified parameters like event type, geographic coordinates, and a distance radius, we applied DBSCAN clustering to create a more intuitive and interactive user experience on the map. Our implementation of DBScan uses the Haversine distance formula, which is used to calculate the great-circle distance between two points on a sphere. DBScan identifies clusters based on: the maximum haversine distance allowed for two points to be considered in the same cluster, and the minimum points required to form a dense cluster. Cluster neighborhoods are defined by the equation: $N_\varepsilon(p) = \{q \in D \mid \text{dist}(p, q) \leq \varepsilon\}$ then a new point is density-reachable from point p if q is within the neighborhood of p, shown here: $q \in N_\varepsilon(p)$.

Why use DBScan rather than regular radius-based filtering?

1. DBScan identifies dense clusters to suggest events outside of a typical range of a predefined shape (such as a circle for radius). It can group clusters based on the varying density, which is very helpful in high-population areas where major event density is not uniform. DBScan also captures natural groupings, such as events close to each other by being located along a street or in a park. 2. DBScan is also able to handle noise effectively. It identifies points outside clusters, helping eliminate isolated, unpopular events. This will allow users to travel from events effectively, keeping them out of dangerous or unfamiliar areas. 3. Lastly, for large datasets, DBScan does not have to recalculate the distance from the user each time they move to select and suggest events. By using spatial indexing, it can cluster events without having to calculate individual distances to the user for every point.

5.2.5 K-Means for Event Type Clustering

K-means reduces the intracluster variance, or the sum of squared differences between data points and a defined cluster centroid. The cluster assignment is given by minimizing the squared Euclidean distance between data point and centroid. The value of each centroid must then be updated as each data point is included in its cluster. The following equation shows centroid computation as the mean of all points assigned to the cluster.

$$c_j = \frac{1}{|s_j|} \sum_{x \in s_j} x$$

Because event types are categorical, one-hot encoding is used to give numerical values to their assigned event type categories. Events are then grouped into clusters by on their event type vectors provided by the one-hot encoding. Events in the same cluster share common attributes, such as being outdoors, music-related, etc. Using k-means, users can receive recommendations for events similar to their preferences. For example, if a user selects "street-event", our equation will classify and recommend other events with the same event type. However, the equation becomes increasingly useful by implementing enhanced recommendations. For example, if "Central High School Baseball game" is classified under "amateur sporting events", but there is no other relevant sporting events in the area, a user may be recommended "NYC stick ball" in a category type of "outdoor recreation". By minimizing the intracluster variance, K-means ensures that the recommended events are highly similar and tailored to previous interest, increasing satisfaction of the end user.

5.3. Data Server

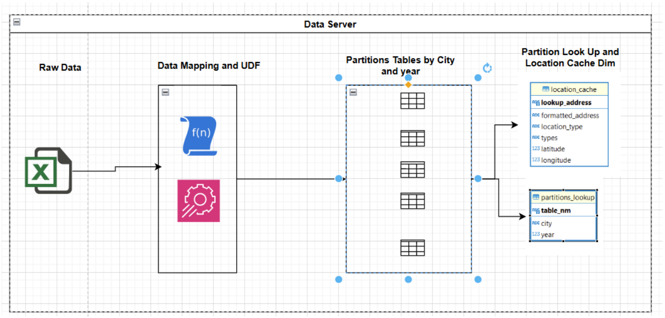


Figure 10. DataServer Layer

Data Server would host main tables for APIs to fetch data relevant for the API. It primary function are described below.

5.3.1 Sample Data

25+ million New york city public events records from 2009-2025. NYC Permitted Event Data

5.3.2 Data Server Pipeline

The data server process begins by loading raw CSV files, which may be mounted or stored in a bucket. This design leverages few innovations to optimize data ingestion and preparation.

Dynamic CSV Ingestion with Custom Header Mapping and UDF Support This feature allows ingestion of different CSV files with varied headers. Consumers can

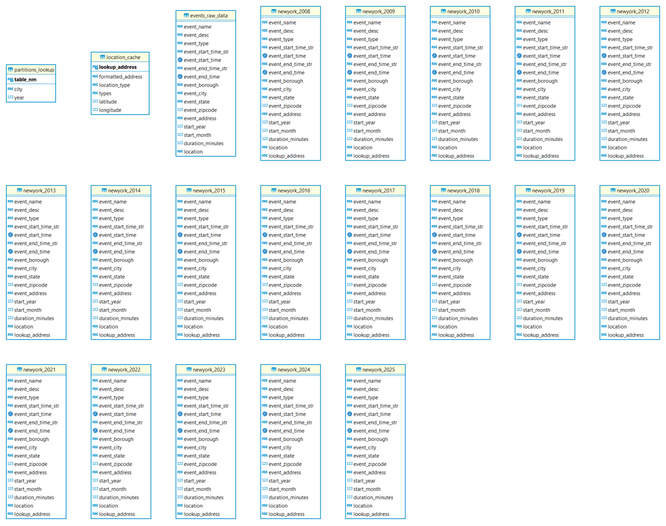


Figure 11. Database Schema

provide a configurable mapping of headers, enabling ingestion of dynamic data without relying on a fixed CSV format. Consumers can also provide ***User Defined Function*** (UDF).

The UDF approach allows consumers to supply a function that supports the necessary formatting or manipulation of data. This adds flexibility for ingesting differently structured information. In our case, this was particularly useful for extracting the location from an event address. Below is an example of data mapping for eventAddress using a UDF for location extraction.

```
{
"rawDataCSV": "data/NYC_events.csv",
"city": "NewYork",
"state": "NewYork",
"csvMapping": [
{"eventName": "Event Name"},
{"eventDesc": "Event Name"},
{"eventType": "Event Type"},
{"eventStartTime": "Start Date/Time",
"udf": null},
{"eventEndTime": "End Date/Time"},
{"eventBorough": "Event Borough"},
{"eventCity": null, "udf": null},
{"eventState": null, "udf": null},
{"eventZipCode": null, "udf": null},
{"eventAddress": "Event Location" ,"udf":
"SPLIT_PART(\"Event Location\",':',1)"}
]}
```

The mapping gets turn into dynamic SQL to ingest data. **Dynamic Data Partition for Performance** As part of this feature we split the large raw data into respective partitions where city and year combination makes total partitions (see figure 10).User queries include year and city, narrowing the search space for better performance. We avoid hash-based partitioning, as our current strategy improves query routing. Partition details are stored in a dedicated dimension table.

Location-Latitude & Longitude Cache Database dimension create_location_cache_dim_table is used as the cache for latitude and longitude. Event address is first looked inside cache before we call API. We use Google Geocode APIs to extract lat/long from partial addresses (e.g., "108th Street between Northern Blvd and 34th Avenue"). To ensure reliability and avoid rate-limiting, we implemented an *exponential back-off algorithm*.

Find nearby locations Implemented Haversine distance function using SQL in DB so that we can use SQL query to find nearby points of interest. Doing this in python was computationally expensive.

For full design details and data pipeline architecture please refer to notebook ***setup_dataserver_db_interface .ipynb***

5.3.3 NLP Pipeline and Multi Label Classification

The NLP pipeline supports tagging events with more enhanced tags.

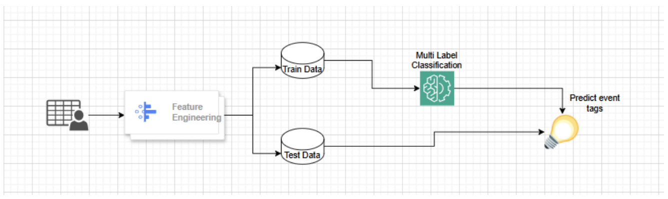


Figure 12. NLP Layer

Feature Engineering for NLP Workflow The feature engineering process (`DataPrep.py`) converts raw event descriptions into numerical representations and labels for model training. Raw data is multi_label_class_event_input_label_data.csv

- *Text Pre-processing:* Convert text to lowercase, remove punctuation, and eliminate stop words.
- *Embedding Generation:* A pre-trained BERT base model (uncased) is used to extract semantic embedding vectors from text
- *Multi-hot Label Encoding:* Represent each event's tags as a multi-hot vector, with 18 unique tags in total.
- *Optimization:* BERT's forward pass to get embedding vector is computation heavy so we have pre-computed and stored feature and label matrices as PyTorch tensors.

Multi Label Classification Model Training & Result Metrics We are using a Deep Neural Network model using pytorch consist of following layers:

```
MultiLabelClass(
(input_layer): Linear(in_features=768,
out_features=512, bias=True)
(hidden_layer): Linear(in_features=512,
out_features=256, bias=True)
(hidden_layer_1): Linear(in_features=256,
out_features=128, bias=True)
(output): Linear(in_features=128,
out_features=18, bias=True)
(relu): ReLU()
(dropout): Dropout(p=0.3, inplace=False)
(sigmoid): Sigmoid()
)
```

Binary Cross-Entropy (BCE) is used as loss function since each output is independent and represents a binary choice: either the tag is present (1) or absent (0). ***Sigmoid*** is used in the output layer so that each output can have full probability from 0 to 1. ***ReLU*** is used as the activation function for capturing any non linearity. ***Finding and evaluation*** The training results were captured by running 20-30 epochs. The classification results demonstrate significant class imbalance, with the model performing well on majority classes but poorly on minority classes.

Class	Precision	Recall	F1-Score	Support
0	0.00	0.00	0.00	1
1	0.00	0.00	0.00	610
2	0.00	0.00	0.00	5
...	...	...	...	...
8	0.57	1.00	0.72	1522
10	0.59	1.00	0.74	1581
14	0.00	0.00	0.00	57

Table 1. Classification Metrics for Each Class

Assessment: The model shows strong performance on majority classes (e.g., class 8 and 10) but fails to classify minority classes due to significant class imbalance. For full design and model refer notebook **`nlp_multilabel_class_model .ipynb`**

6. Summary of Key Innovations

Here is a summary of the innovations discussed in detail above.

6.1. Visualization

- *Pseudo Real-Time Dynamic Data Updating*: Seamlessly updates data as it arrives.

- *Marker Clustering*: Efficiently groups nearby markers to declutter maps.
- *Proximity-Based Event Filtering*: Filters events based on user proximity for relevance.
- *Customizable and Distinctive Marker Icons*: Offers flexibility with custom icons.

6.2. API Framework

- *Standardized API Services*: Unified API support for UI components.
- *Real-Time Weather API*: Integrates live weather updates using Weatherstack API.
- *Ticketmaster API*:Gives historical and future events from ticketmaster.
- *DBSCAN Clustering*: Implements DBSCAN for improved user experience.

6.3. Data Server Framework

- *Data Ingestion and Mapping*: Configurable data ingestion with ability to provide mappings and user defined functions.
- *Data Partitioning*: Speeds up APIs through optimized data partitioning.
- *Cache Management*: Efficient cache for latitude and longitude data. Usage of Google APIs with exponential back-off.
- *Event Tagging with Neural Networks*: Tags events using multi label classification model.

7. Limitations and Future Extension

The current limitation is the absence of a live source system for event data, which could enable real-time updates. As part of POC we have used Ticketmaster API but not fully integrated. This API pulls data from every main event offered by Ticketmaster and organizes it into the required columns. Introducing real-time capabilities would require a more distributed system with asynchronous event loading and the addition of data ingestion APIs. The existing multi-label classification model needs improvement through the collection of more diverse and representative data. Addressing class imbalance is another necessary enhancement. For scalability, DuckDB will need to be shard-ed to handle larger datasets effectively. We can think of data being local to location can be more efficient and reliable. DuckDB being light weight may allow us to distributed the DB itself close to location for which the system is serving.

8. Team Contributions

All team members have contributed a similar amount of effort.

References

[1] Park S. Ham D. Lim H. Bok K. Yoo J. Choi, D. Local event detection scheme by analyzing relevant documents in social networks. applied sciences. 2021.

[2] Faizudheen M. Anikkadan M. J. Rashad M. P. & Shabeer M. U. Afsar, P. Intelligent event finder and management system. proceedings of the fifth international conference on i-smac (iot in social, mobile, analytics and cloud). 2021.

[3] Liu A. & Zhang C. Lu, W. Research and implementation of big data visualization based on webgis. proceedings of the international cartographic association. 2019.

[4] D. Okaragba. Atmosfy: Strategic guide on how to build a great product roadmap. 2022.

[5] M. A. Gad. A gis methodology for the analysis of weather radar precipitation data. 2003.

[6] M. Apputhurai. Dynamic and accurate location college bus tracking system using google api. 2024.

[7] Alexandre Yahi, Antoine Chassang, Louis Raynaud, Hugo Duthil, and Duen Horng (Polo) Chau. Aurigo: An interactive tour planner for personalized itineraries. In *Proceedings of the 20th International Conference on Intelligent User Interfaces (IUI)*, pages 329–334, Atlanta, GA, USA, 2015. ACM.

[8] Igo Brilhante, Jose Macedo, Franco Maria Nardini, Raffaele Perego, and Chiara Renso. Where shall we go today? planning touristic tours with tripbuilder. *International Conference on Information and Knowledge Management, Proceedings*, 10 2013.

[9] Daniel Horowitz, David Contreras, and Maria Salamó. Eventaware: A mobile recommender system for events. *Pattern Recognition Letters*, 105:121–134, 2018. Machine Learning and Applications in Artificial Intelligence.

[10] Mark Raasveldt and Hannes Mühleisen. Duckdb: an embeddable analytical database. 2019.

[11] Jiirg Sander Xiaowei Xu Martin Ester, Hans-Peter Kriegel. A density-based algorithm for discovering clusters in large spatial databases with noise. 1996.