FloodWatch: Building real-time geospatial visualizations for flood detection

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Chris Santamaria

Spring, 2023

Technical Project Team Members

Ankit Gupta

Abhir Karande

Shiva Manandhar

Shuo Yan

N. Rich Nguyen, Department of Computer Science

FloodWatch: Building real-time geospatial visualizations for flood detection

Chris Santamaria
University of Virginia
Charlottesville, Virginia, USA

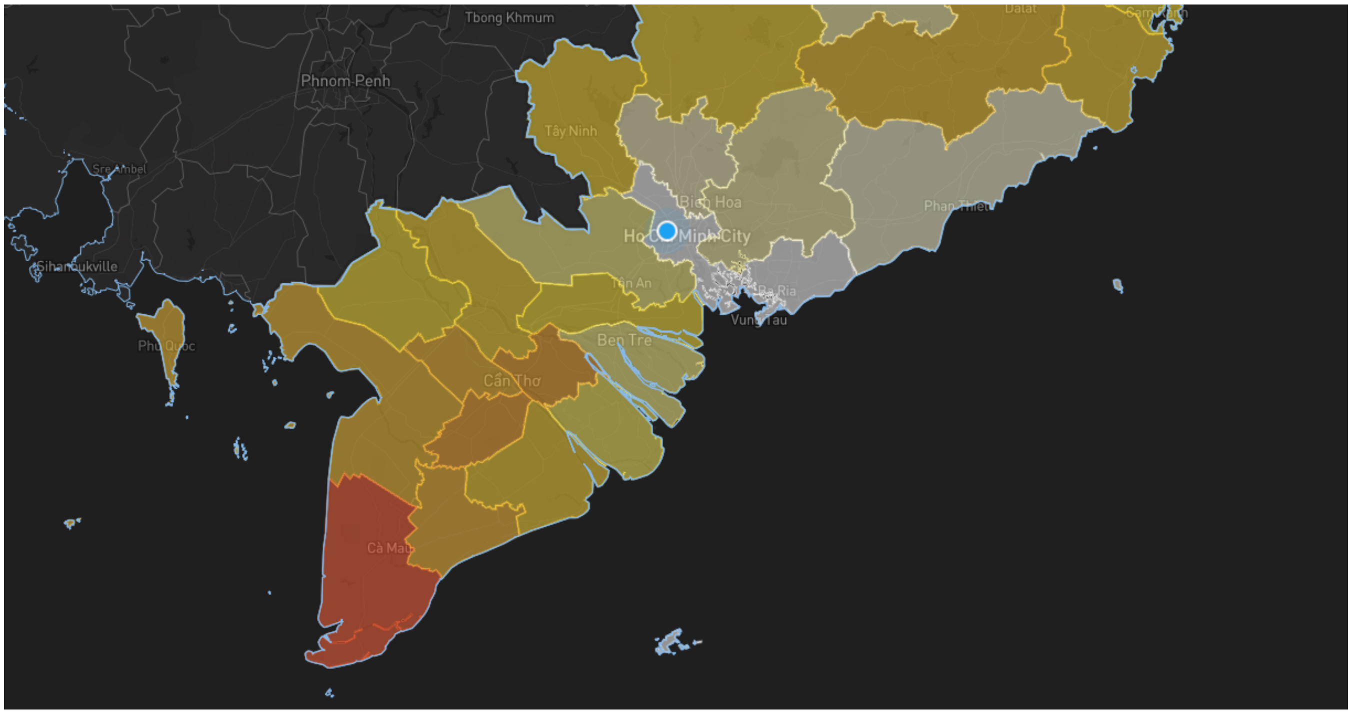


Figure 1. Region tileset layer displaying flood risk in southern Vietnam

Abstract

FloodWatch is a progressive web application, offering real-time flood forecasting and a crowd-sourced reporting platform. The app primarily seeks to aid citizens in Vietnam, especially those in regions susceptible to coastal flooding like Ho Chi Minh City. As part of the platform, effectively informing users of relative flood risk in their region has become a significant area of focus. Our team has implemented this through "region tileset layers", allowing users to gauge day-by-day flood risk at a glance through a visual map overlay.

1 Introduction

Recent models of global tide, storm surge and wave trends have suggested that 52% of the global population will be at risk of flooding by the year 2100 [6]. While this is becoming an increasingly relevant topic in regions worldwide, areas such as Ho Chi Minh City are already greatly susceptible to coastal flooding with the issue worsening due to factors such as growing urban expansion [3]. As a result, there have been ongoing efforts to mitigate flood risk, both to ensure safety of those affected and minimize associated damages. While these sorts of endeavors may be promising long-term, they aren't without their own challenges; efforts such as a $2.6 billion USD ring dike have unclear timelines and may ultimately displace citizens living in construction zones [11]. Furthermore, newly emerging and rapidly changing data can conflict with original projections, resulting in inadequate short-term solutions [7].

Given inevitable constraints of developing physical flood mitigation strategies (be it cost, time or even political decision making), FloodWatch presents a practical and accessible alternative; the app serves at a data distribution platform for citizens, enabling them to make informed choices about their personal safety through both collected weather data and user-submitted flood reports. The notion of live, crowd-sourced flood reports also presents a new perspective when considering the idea of "technological citizenship"; rather than individuals needing to rely solely on relatively opaque government entities, they can instead empower and benefit from other citizens through a new form of civic involvement [1].

Though user-submitted reports provide significant value in the event of a flood, FloodWatch additionally aims to offer objective data through real-time, historical and forecasted weather sampling. All readings are supplemented with a generated "flood risk" score, conveying relevant information to users at a glance. Weather data is currently collected from a variety of third-party providers; in the future, we aim to enable individuals across Vietnam to contribute crowd-sourced weather data directly to the FloodWatch platform.

This paper will describe the design and implementation of FloodWatch's "region tileset" feature, which visualizes approximately 20,000 daily collected weather readings through an intuitive map-based user interface. With this, we aim to provide a meaningful tool to citizens in Vietnam by enabling them to respond to changing flood trends in their particular region.

2 Design

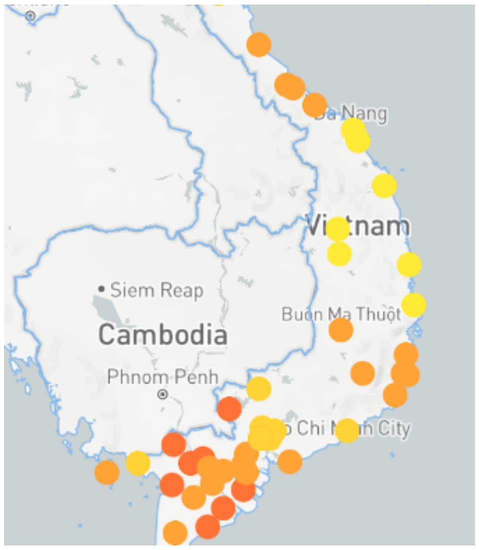


Figure 2. Initial prototype of per-city flood risk markers

As an initial prototype, we approached this problem on a per-city basis with a focus on the most populated cities across Vietnam. Daily average weather data was collected for each city, with readings stored in the app's database alongside its associated collection date. Each location could then be visualized in the FloodWatch app via a marker on the app's main map display, with colors representing the location's relative flood risk. In practice, we found individual markers to be a severe limitation; while major regions like Ho Chi Minh City often had several nearby markers, they failed to offer the level of granularity we sought to provide to users. Similarly, lower population regions were often completely unrepresented, providing no value to residing citizens.

Though our initial thought was simply to increase the number of sampled cities, we quickly realized this was an incomplete solution which would only further clutter the map interface. Instead, we opted to rethink our approach to visualization; while it was clear that we needed to collect data across a larger representation of Vietnam, individual point-based markers proved to be both cumbersome for users and detrimental for app performance.

Ultimately, we landed on a region-based visualization approach. Rather than displaying individual weather collection points, we instead show shaded boundaries representing regions across all of Vietnam. The fill color visualizes the average flood risk score across all sampled points within the boundary, providing a more holistic representation of a region than a single point. Regions are based on "administrative divisions" as defined by the Database of Global Administrative Areas [5]. Notably, our implementation supports multiple levels of granularity: administrative level one (ADM1) including provinces and municipalities, ADM2 including cities and districts, and ADM3 including local communes and wards. By collecting data for multiple levels, we enable visualization of flood risk from high-level provinces to low-level neighborhoods.

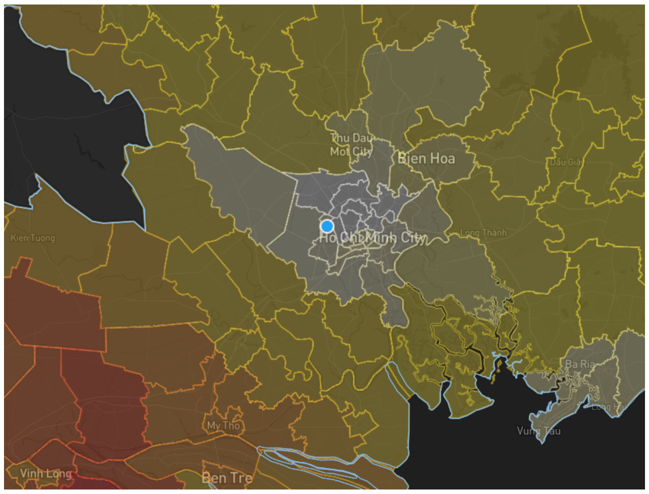


Figure 3. Administrative level 2 region tileset layer

3 Generating collection points

As opposed to the previous per-city approach, sufficiently representing the weather within an entire region requires collecting multiple weather readings which can be aggregated into a final flood risk score. However, determining the quantity and location of these points presents an interesting challenge: while obtaining a boundary's centroid or center of mass is relatively straightforward, generating multiple points within a region is a far more subjective process.

In order to accurately reflect the weather condition across an entire region, generated points must span the entire area;

more specifically, the distance from an arbitrary point in the region from its closest collection point should be minimized. Similarly, there should be enough generated points within the region to keep the shortest distance between collection points low. Fortunately, our use case is relatively flexible on these constraints; given that sampled data will ultimately be averaged, minor deviations from an ideal point distribution are likely negligible for end users.

To solve this problem, we created an internal tool which handles computing and visualizing these points. Given an input data set, the tool generates a list of collection points for each region. Input data is defined according to the GeoJSON specification [2]: regions are represented as a `Feature` objects, with each containing either a `Polygon` (single boundary) or `MultiPolygon` (multiple boundaries) describing its shape. Additionally, each `Feature` contains metadata `properties`; though our input data includes basic properties like region name and administrative level, our tool also adds the generated collection points as an additional property.

For each input `Feature`, collection points are generated with the following algorithm:

1. Calculate a bounding box around the region (including all boundaries if the feature is a `MultiPolygon`)
2. Generate random points within the bounding box, based on a customizable "Area per random point" parameter
3. Filter out generated points which are not contained by the feature's boundary
4. Perform K-means clustering on the points, with the number of clusters based on a customizable "Area per cluster" parameter
5. For each cluster, extract its centroid point
6. Store resulting centroids in the feature's `properties`

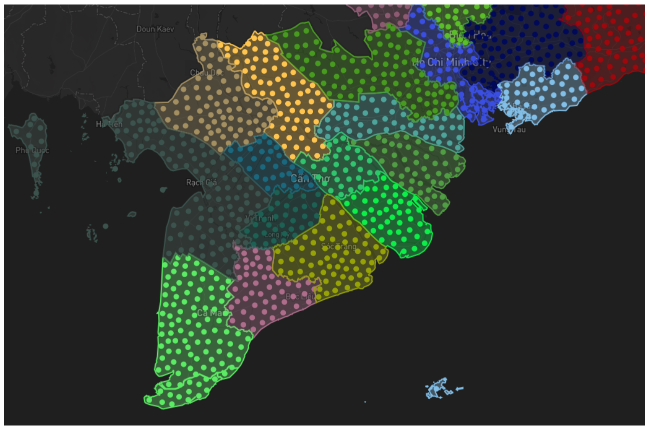


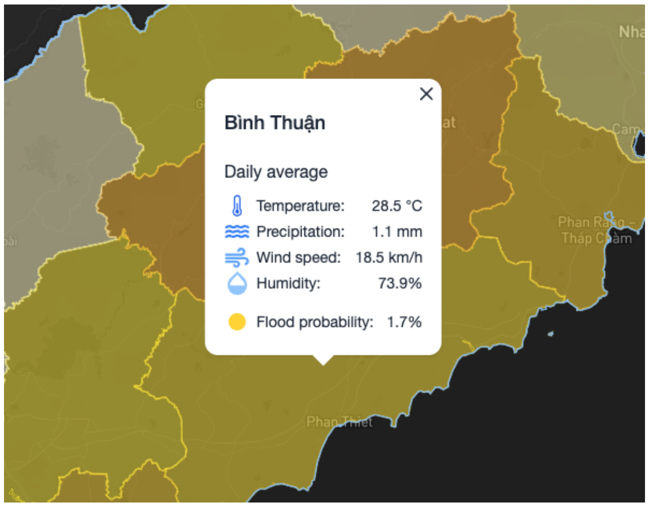
Figure 4. Generated collection points from internal tool

After completion, the tool saves the resulting GeoJSON data set into a new "output" file which can be consumed later in the region tileset pipeline. This process is performed for the ADM1, ADM2 and ADM3 administrative levels — as boundaries become smaller towards the ADM3 level, the number of generated points per feature similarly decrease. However, higher-granularity levels also contain significantly more total points; while the ADM1 data set generates 6,640 points with the tool's default parameters, ADM3 generates more than double with 13,938 points.

In addition to representing flood risk through a fill color on the region map overlay, the app also supports tapping on individual regions to open a popup containing specific weather information. However, rendering this requires determining a specific point within the region to visually "anchor" the popup to. For most single-boundary regions, this is relatively straightforward — calculations like center of mass provide a visually intuitive location. However, our data sets contain additional cases which must be supported:

- If a geometry data is a a `MultiPolygon`, find the center of mass of the largest sub-polygon
- If the calculated center of mass does not fall within the polygon, snap it to the nearest point on its boundary

This point is calculated as part of the previously mentioned collection point algorithm and saved as an additional feature property.



Bình Thuận

Daily average

Temperature:	28.5 °C
Precipitation:	1.1 mm
Wind speed:	18.5 km/h
Humidity:	73.9%
● Flood probability:	1.7%

Figure 5. Region popup anchored at its visual center

4 Collecting weather data

With the collection points generated, weather data can be sampled. To provide per-day visualizations, our team utilities a nightly Python script managed via GitHub Actions [4] which fetches data for a specified query date. For each GeoJSON data set emitted by the previous tool, centroids contained within its feature objects are used to query daily average weather data at each point. Once data is collected for all points within a region, readings are averaged to produce a single set of weather properties displayed to the user. Additionally, a flood risk score is computed based on the obtained weather data — though our team is independently developing a unsupervised learning prediction model, our current generation process uses a relatively simple prediction based on measured precipitation.

Unlike the previously mentioned point generation tool, performance of this weather data collection process is a major priority. In early testing with a initial implementation, we found that querying for just the initial ADM1 data set took nearly five minutes to complete. Our first optimization attempt was to implement multi-threading, spreading the query workload across 8 threads. While this did improve performance, we observed that CPU usage remained relatively low throughout querying, leading us to believe that computation was not the bottleneck. As an alternative, we explored concurrently performing network requests through an asynchronous task pool. By implementing this approach in Python with asyncio [10], we observed a significant reduction to approximately 90 seconds for the ADM1 data set.

While these speedups were promising, they were still insufficient for the total amount of data we would be querying with the ADM2 and ADM3 data sets. Fortunately, we noticed that our chosen weather provider supports batched requests of up to 50 queries at once. Utilizing this, we observed dramatic improvements from the minimized number of round-trip network requests: our previous ADM1-only benchmark was reduced to less than 5 seconds, while running all data sets simultaneously takes approximately 16 seconds.

5 Visualization

With the weather data collected, we shifted our focus towards visualization in the FloodWatch web app. We've previously relied on Mapbox for map rendering via their mapbox-gl-js library [8], so we opted to implement this feature within our existing map UI. Mapbox supports loading GeoJSON data as "sources", enabling rendered "layers" to selectively display portions of the retrieved data; for example, rendering a partially-transparent layer with boundary fill colors dependent on flood risk score.

However, the manner in which data is loaded is fairly customizable. Our initial approach was to host generated GeoJSON data sets from our nightly weather collection jobs, allowing clients to download and visualize them in-app. Unfortunately, we quickly realized this had a significant upfront data bottleneck — all three data sets were approximately 130 megabytes, with ADM3 alone being nearly 90 megabytes. Until the data sets were loaded, we were unable to visualize any portion of the data. Given that many target users in Vietnam may not have high-speed or stable internet connections, this approach was insufficient.

Many modern mapping solutions approach this problem via streaming "tiles" of data as needed, allowing clients to only download map data for the regions visible or nearby their current viewport. Tiles are often composed of raster (pixel-based) or vector (GeoJSON-based) data, with metadata such as zoom bounds being defined by a parent "tileset". Using a vector tile-based approach seemed ideal for our use case, though we quickly encountered another obstacle: in order to support tile streaming, clients must interact with a tileset server configured to send each generated data set. Attempting to manually implement, configure and manage such a server proved to be a significant challenge.

Fortunately, Mapbox provides their own Mapbox Tiling Service [9], abstracting away most of the implementation complexity of tilesets. At a high level, generated GeoJSON data sets containing daily weather information are sent to Mapbox as "tileset sources" during our nightly job. With these uploaded, we can compose a single tileset with each data set configured as a source layer. Once completed, clients can load the tileset via a predetermined URL and progressively stream map data; on average, rendering an initial map view downloads less than 800 kilobytes of tile data.

6 Future work

Though our final implementation has served us well, we're continuing to iterate on both the tileset generation pipeline and user-facing experience. Notably, one area of focus is the frequency of tileset generation. Our current approach creates entirely new tilesets (and associated tileset sources) for each day processed; given the static nature of our administrative boundary data, we'd ideally prefer to only update the weather data encoded in `Feature` properties while keeping `geometry` data constant. However, due to the way Mapbox Tiling Service handles tiles, we are unable to make any changes to an existing tileset without reprocessing all tiles, effectively recreating the entire tileset.

One possible alternative could be to decouple the weather data from the tileset itself. Rather than generating a complete tileset in our nightly job, we could instead create a minimal JSON-based file containing only weather data, with each entry keyed by a unique region ID. When a client loads the map, we could download both a static tileset (containing only administrative boundary data) as well as well as dynamic weather data (from a file storage provider like AWS S3). Once both have loaded, we could inject weather data into the client-side tileset properties for visualization. While this may work in theory, it effectively eliminates the streaming capabilities of our current tileset approach, requiring clients to download weather data for all of Vietnam upfront to render any portion of the region tileset.

More broadly, our team is beginning to shift focus from research-oriented feature experimentation towards polishing the app for end users. Though we believe the platform can provide immense value to citizens in Vietnam, refining the new user experience and promoting a sense of civic involvement will become crucial aspects to preparing FloodWatch for real-world usage. Early user testing has highlighted a few notable areas of improvement, with the previously mentioned flood risk score being a common area of confusion. To help improve this, other FloodWatch team members have been independently developing a clustering-based unsupervised learning model trained on historic flood data in Vietnam. Once completed, the accuracy (and in turn, utility) of our predictions should significantly improve. Similarly, we're continuing to iterate on our visualization approach to intuitively convey risk level — for example, reserving red-shaded regions for areas which have a high risk of flooding.

Acknowledgments

This project has been funded by grants won by Professor Nguyen, Department of Computer Science. He has helped with defining the vision of the project, gaining support from our collaborators at HCMUS, and overseeing the project's progress.

References

[1] C.J. Andrews. 2006. Practicing technological citizenship. *IEEE Technology and Society Magazine* 25, 1 (2006), 4–5. https://doi.org/10.1109/MTAS.2006.1607713

[2] H. Butler, M. Daly, A. Doyle, Sean Gillies, T. Schaub, and Stefan Hagen. 2016. *The GeoJSON Format.* Request for Comments RFC 7946. Internet Engineering Task Force. https://doi.org/10.17487/RFC7946 Num Pages: 28.

[3] Phan N. Duy, Lee Chapman, Miles Tight, Phan N. Linh, and Le V. Thuong. 2017. Increasing vulnerability to floods in new development areas: evidence from Ho Chi Minh City. *International Journal of Climate Change Strategies and Management* 10, 1 (Jan. 2017), 197–212. https://doi.org/10.1108/IJCCSM-12-2016-0169 Publisher: Emerald Publishing Limited.

[4] GitHub. 2023. GitHub Actions. https://github.com/features/actions

[5] Robert J. Hijmans. 2009. Global Administrative Areas (GADM). https://gadm.org/

[6] Ebru Kirezci, Ian R. Young, Roshanka Ranasinghe, Sanne Muis, Robert J. Nicholls, Daniel Lincke, and Jochen Hinkel. 2020. Projections of global-scale extreme sea levels and resulting episodic coastal flooding over the 21st Century. *Scientific Reports* 10, 1 (July 2020), 11629. https://doi.org/10.1038/s41598-020-67736-6 Number: 1 Publisher: Nature Publishing Group.

[7] Robert Lempert, Nidhi Kalra, Suzanne Peyraud, Zhimin Mao, Sinh Bach Tan, Dean Cira, and Alexander Lotsch. 2013. *Ensuring Robust Flood Risk Management in Ho Chi Minh City.* The World Bank. https://doi.org/10.1596/1813-9450-6465

[8] Mapbox. 2023. mapbox-gl-js. https://github.com/mapbox/mapbox-gl-js original-date: 2013-03-07T14:45:24Z.

[9] Mapbox. 2023. Mapbox Tiling Service. https://docs.mapbox.com/mapbox-tiling-service/guides/

[10] Python Software Foundation. 2023. asyncio — Asynchronous I/O. https://docs.python.org/3/library/asyncio.html

[11] Lizzie Yarina. 2018. Your Sea Wall Won't Save You. *Places Journal* (March 2018). https://doi.org/10.22269/180327