Interactive-Explanatory Geospatial Data Visualizations on the Web

Tammy Zhang[1], Nihanth Cherukuru[2], David John Gagne[2], Pritam Das[3], Negin Sobhani[2]

[1] Cornell University, [2] NSF National Center for Atmospheric Research, [3] University of Washington

Introduction

Interactive web visualizations make datasets directly viewable in a browser, allowing users to explore the data through scrolling, panning, filtering, etc.

- Researchers can quickly get **general insights** from their data
- Makes datasets and the stories they convey **easily shareable to broad audiences**

Challenges

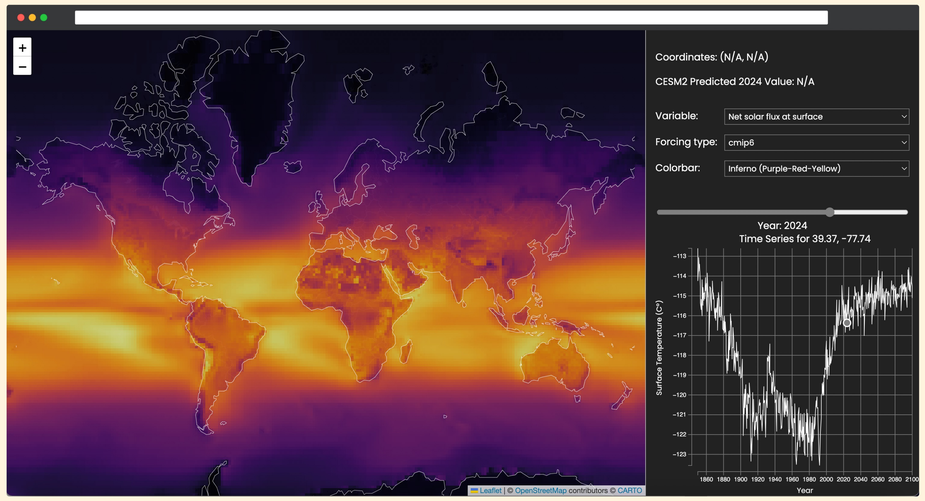
Climate/weather datasets often:

- Are **large** in file size, challenging browser storage limits
- Come in **varied formats** that don't naturally play well with JavaScript, the core language of the web
- Are **rasterized** in map visualizations, which limits their interactivity

Objectives

- Explore a range of popular **modern open source libraries and approaches** for creating interactive climate/weather data visualizations on the web
- Evaluate **tradeoffs between two visualization projects** that each used a variety of different techniques

Case 1: CESM2 LENS2 Dashboard



A dashboard for the NSF NCAR Community Earth System Model (CESM2) Large Ensemble Community Project (LENS2), a set of climate model simulations.

Main tools used:

- D3.js
- Leaflet
- Flask (Python)

Workflow:

1.) Data acquired in NetCDF (.nc) format and stored locally

2.) Backend Python server uses the xarray library to index the dataset and return chunks by map tile

3.) In the frontend, D3.js constructs a rectangle for each data point to cover tiles continuously

Comparison

Python Server vs. client-side JavaScript

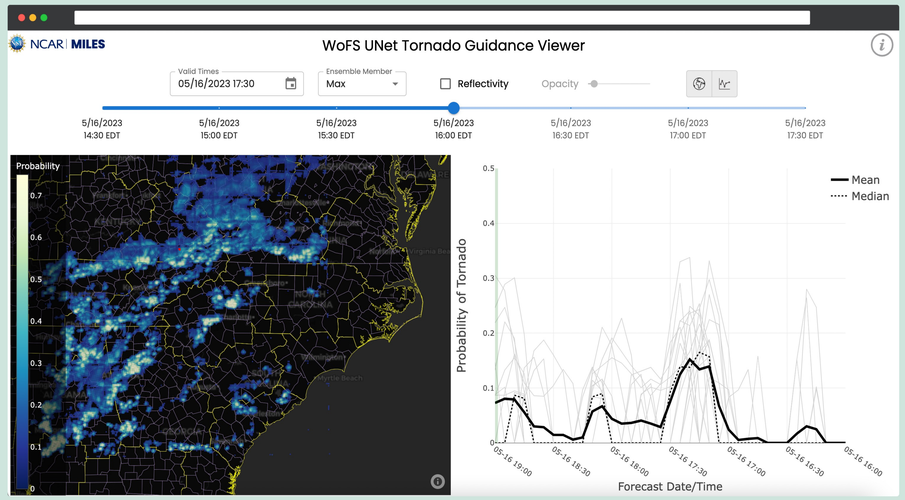
More customized vs. more built-in map tiling

Native JavaScript vs. JavaScript frameworks

Conclusions

- Python **eliminates intermediary formats**
- However, solely client-side JS is much **easier to host and share**
- More customized workflows allow for more room for creativity -> good for **communicating narratives**
- More built-in workflows have faster dev time -> good for **exploring data**
- JavaScript frameworks like React have higher dev time but allow access to highly **optimized data fetching libraries** that can take advantage of datasets **stored in the cloud**

Case 2: WoFS UNet Tornado Guidance Viewer



A readaptation of a visualizer for Warn-on-Forecast tornado predictions generated through machine learning at NSF NCAR's Research Applications Laboratory (RAL).

Main tools used:

- React
- React Plotly
- TanStack Query
- Microsoft Azure

Workflow:

1.) Data stored by model and init run time in a cloud object storage bucket

2.) TanStack Query makes parallel async requests for all needed datasets

3.) Requested data transformed from MessagePack format into standard JSON

4.) Passed to React Plotly for visualization