Revista Eletrônica de Iniciação Científica em Computação, 2026, 24:1
ISSN: 3085-8461 • doi: 10.5753/reic.2026.7493
This work is licensed under a Creative Commons Attribution 4.0 International License.

Research Paper

Evaluation of Python Libraries for Visualization of Weather Fields

Lucas Adati de Paula [Faculdade de Tecnologia de Cruzeiro (FATEC) / Instituto Nacional de Pesquisas Espaciais (INPE) | *lucas.adati.p@gmail.com*]
Ivo Kenji Koga [Instituto Nacional de Pesquisas Espaciais (INPE) | *ivo.koga@inpe.br*]
Eugênio Sper de Almeida [Instituto Nacional de Pesquisas Espaciais (INPE) | *eugenio.almeida@inpe.br*]

Faculdade de Tecnologia de Cruzeiro (FATEC), Avenida Rotary, 383, Vila Paulista, Cruzeiro, SP, 12701-170, Brasil

Abstract. BRAMS forecasting workflows generate multidimensional atmospheric fields that are post-processed with legacy GrADS scripts, which limits integration with current web-oriented dissemination pipelines. This paper investigates a Python-based alternative for the final visualization stage of this workflow. We implemented a controlled qualitative comparative study in which Cartopy, Plotly, Bokeh, and Folium were evaluated under the same preprocessing conditions, using BRAMS outputs and equivalent geographic and variable selections. The assessment followed an explicit framework based on functional coverage, interactivity, integration effort, and output suitability. Results showed a consistent trade-off between projection/cartographic quality and interactive delivery. In the evaluated scenarios, the combination of Folium, Matplotlib, and geojsoncontour provided the best balance for interactive contour-based map dissemination. As an application contribution, we integrated this pipeline into a Streamlit interface that enables non-programmer users to explore variables and pressure levels with real-time visual feedback. These findings provide a reproducible basis for updating BRAMS visualization workflows while preserving operational usefulness.

Keywords: Graphics libraries evaluation, Visualization, Python language, Atmospheric models, BRAMS

Received: 28 January 2026 • **Accepted:** 10 July 2026 • **Published:** 11 September 2026

1 Introduction

Atmospheric and environmental forecasting involves determining the future state of the atmosphere based on a known state at a given time Corte-Real [2015]. The output of these forecasts, often generated by numerical models, consists of multi-dimensional matrices representing atmospheric variables at various grid points Lynch [2008].

The Brazilian Developments on the Regional Atmospheric Modeling System (BRAMS) is a model used for atmospheric and environmental forecasting. Its workflow collects environmental information, adjusts spatial resolution, and runs the model on supercomputers or High-Performance Computing (HPC) clusters.

The BRAMS outputs generate files in GrADS (Grid Analysis and Display System) format. In the final stage of the workflow, meteorological maps and graphs are generated to allow analysis by meteorologists and for visualization on web pages.

Historically, BRAMS outputs were visualized using specialized, monolithic software for atmospheric sciences, such as GrADS (Grid Analysis and Display System), NCAR Graphics, and Vis5D. These tools are highly efficient for handling formats inherent to BRAMS, especially GrADS and NetCDF Drach and Caron [2013].

However, their closed ecosystems make integration with modern machine learning workflows and custom Web APIs difficult. In addition, they usually prioritize cartographic accuracy over user interactivity.

In this context, the shift toward programmatic workflows has decoupled data manipulation from rendering, allowing finer control of BRAMS datasets with modern geoscience libraries such as Xarray, MetPy, and Cartopy.

To address this challenge, it is essential to find alternatives that enable a simple and efficient execution of the meteorological workflow.

This work's objective is to improve the final stage of the BRAMS atmospheric model workflow. It focuses on analyzing and evaluating Python libraries as an alternative to the GrADS system for manipulating and visualizing data produced by the model. The evaluation was based on three practical criteria: functionality, user interactivity, and long-term maintainability. To support this process, we used Python libraries to process data from different atmospheric layers. The specific objectives include:

- Choosing a framework that enables the agile development of a web application;
- Defining, analyzing, and evaluating suitable libraries for manipulating and visualizing atmospheric data;
- Preparation and automation of the work environment for reading and processing these data.

The following sections discuss the related work, methodology, results and discussions, and conclusions, respectively.

2 Background

BRAMS is a regional-scale multifunctional numerical weather prediction model. It was designed to simulate atmospheric circulations ranging from hemispheric scales down to Large Eddy Simulations (LES) of the planetary boundary layer Freitas *et al.* [2017].

Atmospheric and environmental data outputs can be accessed, manipulated, and visualized using GrADS in an interactive and simplified manner Doty *et al.* [1997]. In addition

to being implemented on various operating systems, it is also freely distributed over the Internet.

In traditional BRAMS workflows, map generation in GrADS relies on its own scripting language, which couples data processing and visualization in a single environment. As an example, Figure 1 presents a Mineral Dust Optical Thickness field from the Global Forecast System (GFS) rendered in GrADS.

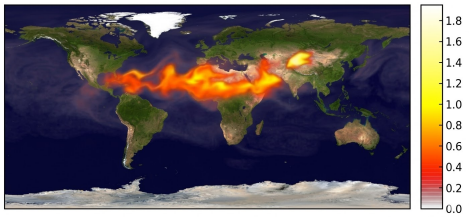

Figure 1. Visualization of the Mineral Dust Optical Thickness field from the GFS model. FSource: https://a.fsdn.com/con/app/proj/opengrads/screenshots/153722.jpg/max/max/1.

Although this approach is effective for operational routines, it makes it harder to separate data orchestration, scientific analysis, and user-facing dissemination as independent components of the workflow. To clarify this broader perspective, Figure 2 presents a multi-tiered Meteorological Visualization System architecture Rautenhaus *et al*. [2018] organized into three functional layers, from initial data acquisition to final user interaction.

The Data Orchestration Layer is the system foundation and manages raw information, including data ingestion from satellites, radar systems, and ground weather stations, followed by cleaning, standardization, and storage in high-performance databases and server infrastructure.

The Static Geo-analysis Layer transforms these inputs into scientifically meaningful spatial information through meteorological modeling (e.g., BRAMS), spatial-temporal analysis, and GIS-based geoprocessing with proper coordinate systems and map projections.

The Interactive Dissemination Layer, where this study is positioned, focuses on end-user access through interactive web maps, real-time dashboards, and user applications across devices, while also supporting alerts and data export for external analysis.

Given this three-layer architecture, implementing the Interactive Dissemination Layer requires technologies that connect scientific processing to accessible web delivery. In this context, Python has become increasingly common in Earth Sciences because it combines a broad scientific ecosystem with reproducible workflows Kadiyala and Kumar [2017]; Arms *et al*. [2020]. To make this review clearer, we group the work into three categories: data manipulation and interpretation, static and interactive map rendering, and web application framework.

In BRAMS workflows, Python is often adopted as an alternative to shell scripts and GrADS Lin [2012]. This shift is also influenced by limitations in legacy tools, including lower maintenance activity and weaker integration with modern software stacks.

Data manipulation and interpretation (DMI)

The xgrads package Qian *et al*. [2025] reads and parses `.ctl` descriptor files and converts BRAMS outputs to structures compatible with the scientific Python stack. NetCDF support Rew and Davis [1990] provides metadata-rich access to multidimensional atmospheric variables.

Xarray and NumPy form the core in-memory data model of this workflow Hoyer and Hamman [2017]; Gupta and Bagchi [2024]. In practice, their main strengths are labeled dimensions, coordinate-aware indexing, and smooth integration with other analysis libraries.

For BRAMS outputs, these features help reduce indexing mistakes across atmospheric levels and variables. A Dataset example is shown in Figure 3, and a four-dimensional reading workflow is illustrated in Figure 4.

MetPy complements these libraries by providing meteorology-oriented calculations and CF-compliant coordinate handling, helping preserve geospatial consistency during transformations May and Bruick [2019]. However, because its role is analytical rather than presentation, it does not provide a complete interactive visualization layer on its own.

Static and interactive map rendering (SIMR)

Rather than treating these libraries as interchangeable, our analysis used three criteria aligned with the Interactive Dissemination Layer: (i) cartographic fidelity, (ii) interactive usability, and (iii) integration effort in the BRAMS workflow. Under this lens, no single library dominates all dimensions.

Cartopy performs best for cartographic fidelity because its projection support integrates naturally with Matplotlib and yields publication-quality static products May and Bruick [2019]; Shreemathi *et al*. [2024]. Its main limitation, however, is in interactive usability, since native outputs are primarily static.

Plotly and Bokeh improve interactive usability in browser environments Frants and Ohman [2024]; Jolly [2018], but in our context they require additional adaptation when combining meteorological contours, base maps, and layered composition. This raises integration effort for operational BRAMS pipelines.

Folium, especially when combined with GeoJSONContour, offered the most balanced trade-off for our dissemination objective: acceptable geospatial expressiveness, strong web interactivity, and lower integration friction for serving contour-based products in a web map interface Suma *et al*. [2024]; Sujithra and Shanmugapriyaa [2024].

Web application framework (WAF)

Consistent with the same evaluation criteria used above (interactive usability and integration effort), Streamlit was adopted as the application layer because it enables rapid development of interactive scientific interfaces with low front-end overhead Moscato [2024]. In our workflow, this choice reduced implementation complexity and accelerated delivery of the Interactive Dissemination Layer, with the trade-off of less fine-grained UI control than fully custom web frameworks.

A comparative synthesis of the libraries discussed in this section is presented in Table 1.

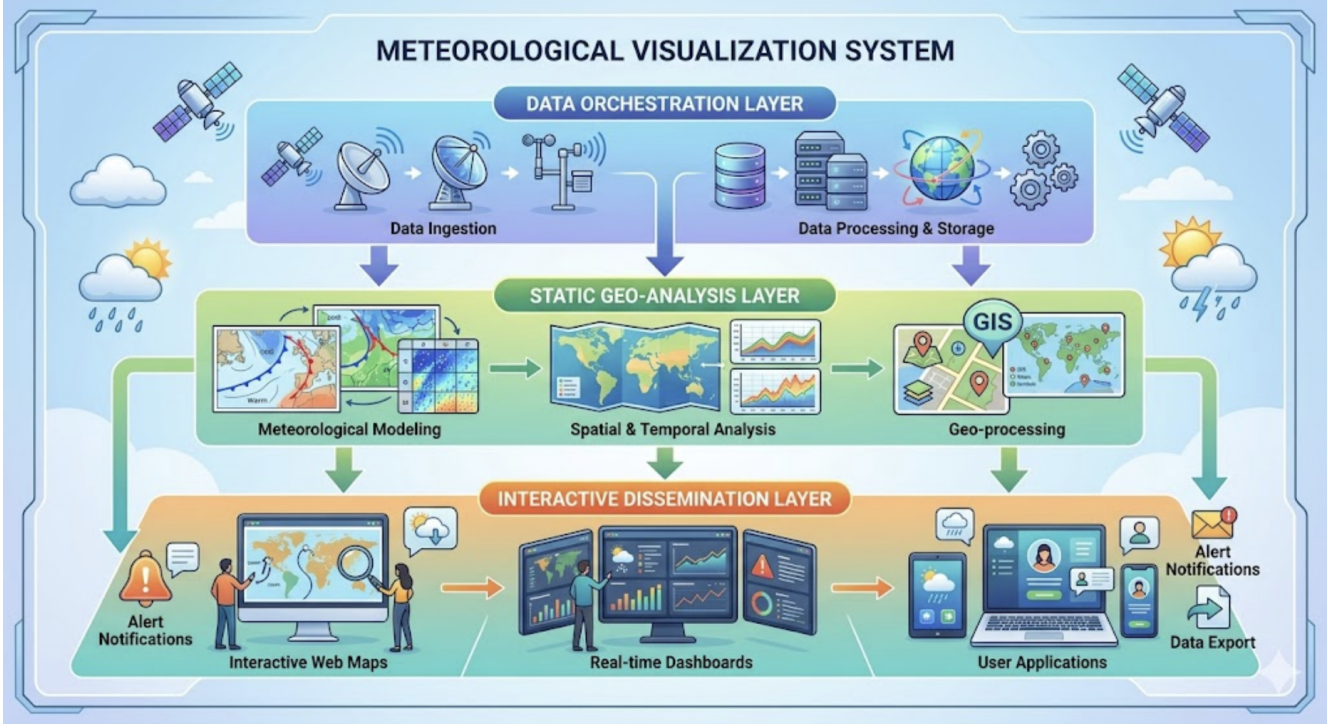

Figure 2. Meteorological Visualization System (Image). Source: Generated by Gemini AI (Google), 2026, prompt: "A meteorological visualization system is composed of Data Orchestration Layer, Static Geo-analysis Layer, Interactive Dissemination Layer. Generate a figure of these layers".

Table 1. Comparative synthesis of libraries considered for BRAMS outputs.

Library	Category	Main strength	Main limitation	Fit for BRAMS workflow
xgrads	DMI	Reads GrADS `.ctl`/binary structure	Depends on GrADS descriptor quality	Essential for BRAMS ingestion
Xarray + NumPy	DMI	Labeled multidimensional processing	Memory-sensitive with large datasets	Core analysis layer
MetPy	DMI	CF-aware meteorological operations	Not a full visualization stack	Strong domain support
Cartopy + Matplotlib	SIMR	Accurate projections and publication-ready maps	Low end-user interactivity	Strong for static outputs
Plotly / Bokeh	SIMR	Browser-based interaction and rapid exploration	Extra work for contour-map composition	Moderate, use-case dependent
Folium + GeoJSONContour	SIMR	Interactive maps with contour overlays	Requires conversion pipeline	Strong for interactive map delivery
Streamlit	WAF	Fast interface prototyping	Limited fine-grained UI control	Strong for rapid deployment

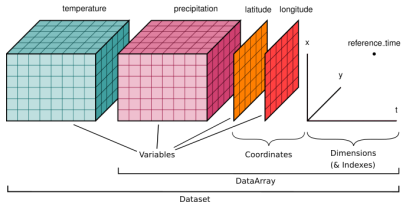

Figure 3. Representation of an Xarray Dataset. Source: Xarray developers (2023).

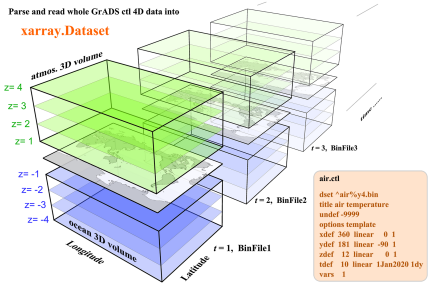

Figure 4. Reading and analyzing four-dimensional GrADS ctl data. Source: xgrads (2023).

3 Methodology

This section describes the experimental setup used to compare visualization libraries for BRAMS outputs, including the dataset, processing pipeline, evaluation criteria, and comparison protocol. All tests were carried out within the same Python-based workflow to ensure a fair comparison across libraries. The data ingestion and preprocessing steps were kept fixed, and only the visualization layer was changed during the evaluation.

The comparison was designed as a controlled library-substitution experiment: each candidate library received equivalent atmospheric inputs, geographic extents, and plotting tasks, and was assessed under the same workflow conditions. The objective was to evaluate suitability for BRAMS dissemination in practice, not to establish hardware-dependent performance benchmarks.

As summarized in Figure 5, the pipeline is organized into three complementary categories: data manipulation and interpretation libraries (xgrads, Xarray, NumPy, and MetPy) for ingesting and processing multidimensional BRAMS fields, static and interactive map rendering tools (Cartopy/Matplotlib, Plotly, Bokeh, and Folium) for geospatial rendering and exploratory outputs, and a web application framework (Streamlit) for delivering user-facing interactive products without dedicated front-end development.

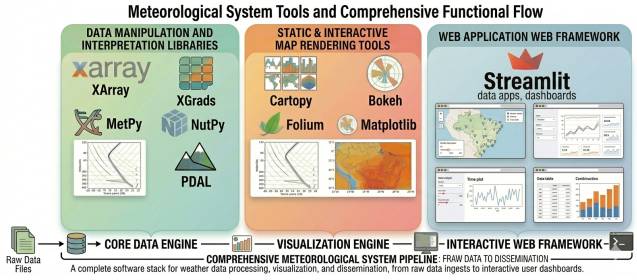

Figure 5. Categories of meteorological data tools used in the workflow context.

The data manipulation pipeline was defined as follows:

- read BRAMS output descriptor files (`.ctl`) and binary data with xgrads (v0.2.3);
- convert data to `xarray.Dataset` structures;
- process atmospheric variables in memory with Xarray (v0.20.2) and NumPy (v1.21.6);
- parse coordinates and metadata with MetPy (v1.2.0), using `parse_cf()`.

The dataset consists of BRAMS atmospheric model outputs exported in GrADS-compatible format. The experiments used multidimensional fields (latitude, longitude, level, and time), with emphasis on variables and pressure levels commonly used in operational workflows. To ensure consistency, each visualization library received the same preprocessed slices of atmospheric fields and equivalent geographic extents.

Four visualization alternatives were tested as candidates to replace or complement the traditional GrADS rendering stage:

- Cartopy (v0.20.3), combined with Matplotlib for static geospatial maps;
- Plotly (v5.8.0), for interactive web-oriented maps and contours;
- Bokeh (v2.4.3), for browser-based interactive visualization;
- Folium (v0.14.0), with GeoJSON overlays for interactive map dissemination.

To deliver the resulting products to end users, Streamlit served as the application framework. It enables rapid conversion of Python scripts into fully functional web applications and supports interactive dashboards in which users can select forecast dates with sliders, choose atmospheric variables (e.g., wind speed), and visualize map updates in real time without requiring HTML or JavaScript development.

The comparative analysis was based on four criteria defined before the experiments:

- **C1 – Functional coverage:** ability to render meteorological contours, overlays, and projected maps from BRAMS fields;
- **C2 – Interactivity:** support for zoom, pan, layer toggling, and user exploration in a web interface;
- **C3 – Integration effort:** complexity of integrating each library into the Streamlit-based application;
- **C4 – Output suitability:** adequacy for the target use cases, including publication-style static outputs and interactive web dissemination.

Each library was evaluated qualitatively against these criteria using the same input data and comparable map products. The assessment procedure prioritized alternatives that simultaneously satisfied contour-rendering requirements (C1), interactive web delivery (C2), and manageable integration effort (C3), while still producing outputs suitable for dissemination and reporting (C4). Under this framework, the selected combination was derived from explicit cross-criterion trade-offs rather than from isolated visual preference.

4 Results and Discussion

Python was adopted as an alternative for visualizing the meteorological fields of the BRAMS numerical model. Its usage, widely recognized in data science projects due to its diverse libraries, significantly facilitated the learning curve for any developer joining the project.

During the project's development, we analyzed and selected appropriate libraries for manipulating and visualizing atmospheric data. This process included defining the necessary library adaptations and automating the work for reading the BRAMS data.

Developing a web application for manipulating and visualizing meteorological data became simpler and more agile with the adoption of the Python-based Streamlit framework, which allows easy integration of various processing and visualization libraries. We tested and evaluated various graphical libraries to determine the optimal visualization method for the meteorological fields.

The selection of appropriate software tools is a critical step in the development of a Meteorological Visualization System. This selection process is governed by the specific requirements of each architectural layer: the Static Geo-analysis Layer, which demands high cartographic precision, and the

Interactive Dissemination Layer, which prioritizes user engagement and web accessibility.

The comparative analysis showed distinct trade-offs among Cartopy, Plotly, Bokeh, and Folium when applied to BRAMS outputs. The main differences emerged in projection fidelity, contour-map integration, and level of interactive web usability.

Table 2 provides a technical summary of this evaluation, highlighting the specific advantages and limitations encountered during the prototyping of atmospheric data visualizations. The comparison specifically focuses on the libraries' capacity to render meteorological contours and their compatibility with standard geographic projections.

Figure 6 illustrates the Streamlit-based user interface for accessing and configuring meteorological variables. It allows users to select a variable, define its atmospheric level, and adjust its opacity through interactive selectors and sliders. The figure shows an example configuration in which the relative humidity (rh) variable and the 550 millibar (mb) atmospheric level are selected.

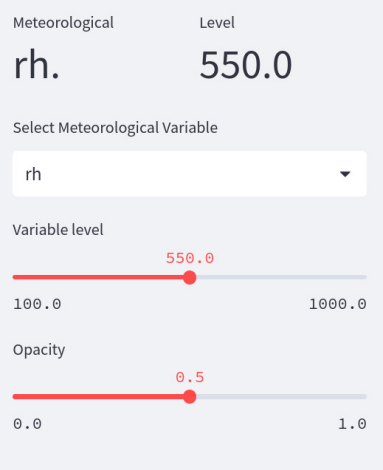

Figure 6. Data control menu.

For this variable, atmospheric levels are plotted from 100 to 1000 mb at 50 mb intervals. Figure 7 illustrates the command-line interface (CLI) output after selecting the variable and its atmospheric level.

```
########################################################################
***                  TIME: 2023-09-28 16:45:00                       ***
########################################################################


>>> Selection:  rh 550.0
>>>  <class 'xarray.core.dataarray.DataArray'>
----------------------------------------
>>> var_interval:  [ 0 10 20 30 40 50 60 70 80 90]
>>> var_selected:  rh
<class 'cartopy.mpl.geoaxes.GeoAxes'>
```

Figure 7. Command line interface output.

Out of the four visualization libraries initially analyzed, we opted to use Cartopy in conjunction with Matplotlib to integrate and plot the collected data. This initial test successfully enabled a visualization of a map featuring a contour plot of the generated data.

However, a key limitation of this approach is that the resulting plot is static, which prevents user interaction. Figure 8 illustrates the output produced with the Cartopy and Matplotlib combination.

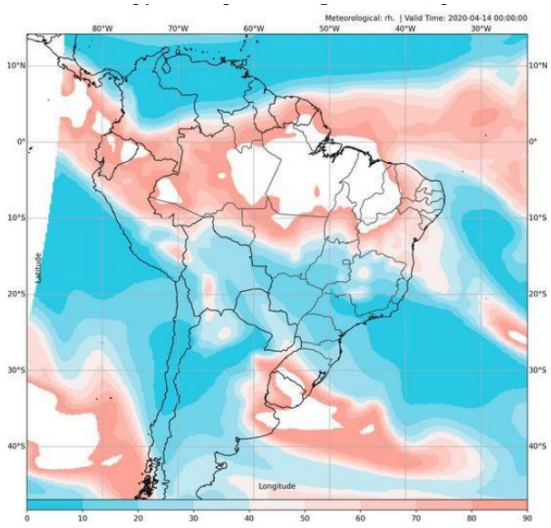

Figure 8. Visualization with Cartopy and Matplotlib.

We tested the Plotly library, which enabled the creation of interactive maps and contour plots. However, integrating both elements was not feasible, resulting in the map being overlaid on the contour plot (Figure 9).

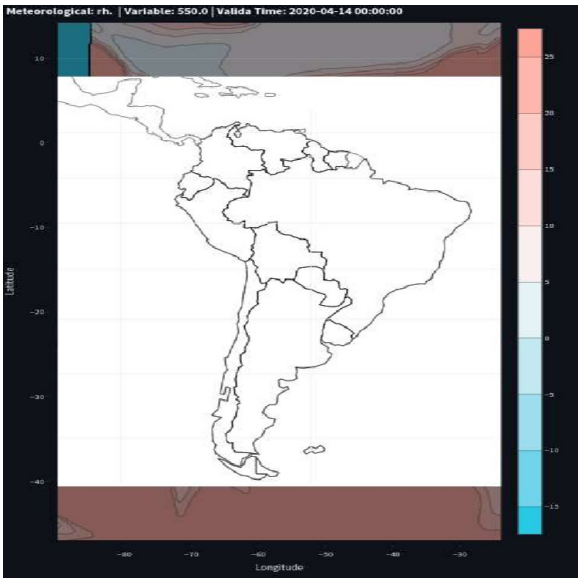

Figure 9. Visualization with Plotly.

The Bokeh library provides the functionality to plot the map. However, in the version used in this study, it did not include a native function for contour plotting. As a result, only the base map was produced, as shown in Figure 10.

Map creation for visualizing spatial data was optimized using Folium with OpenStreetMap as the base layer. Data exploration was enhanced by adding zoom and full-screen controls, and a grid overlay was integrated into the main visualization.

Contour plots were generated with Matplotlib and then converted to GeoJSON using the geojsoncontour library. The resulting GeoJSON layer was overlaid on the base map, producing an interactive visualization that displays the generated contour field (Figure 11).

In addition, PNG (Portable Network Graphics) export was implemented to support use on web-based platforms.

Table 2. Comparison of visualization libraries evaluated in this study.

Library	Version	Core Advantages	Technical Limitations
Cartopy	0.20.3	Robust support for geospatial projections (e.g., Plate-Carrée); seamless integration with Matplotlib for publication-quality static maps.	Lacks native dynamic interactivity; limited to static output formats (PNG, PDF, SVG).
Plotly	5.8.0	High-level interactive API; supports complex hover effects and web-ready animations using Scattergeo and Contour traces.	Significant challenges in layering diverse data types; contour plots often obscure underlying base maps rather than blending.
Bokeh	2.4.3	Optimized for high-performance web interactivity; supports various tile providers like CartoDB Positron.	No native high-level function for meteorological contouring; requires custom patches or external pre-processing.
Folium	0.14.0	Seamless integration with Leaflet.js; supports OpenStreetMap tiles and GeoJSON overlays for lightweight web dissemination.	Does not natively calculate contours; requires a multi-step pipeline (Matplotlib → geojsoncontour → Folium) for spatial analysis.

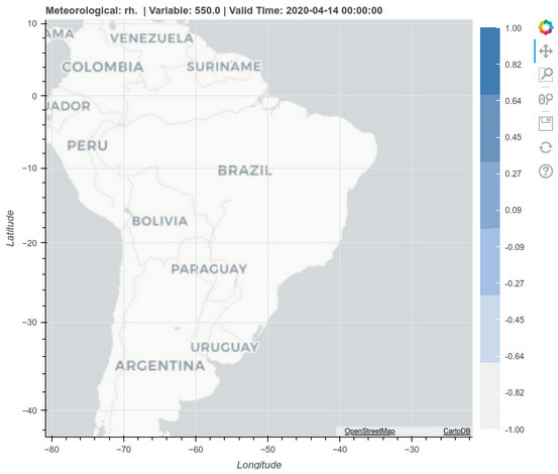

Figure 10. Visualization with Bokeh.

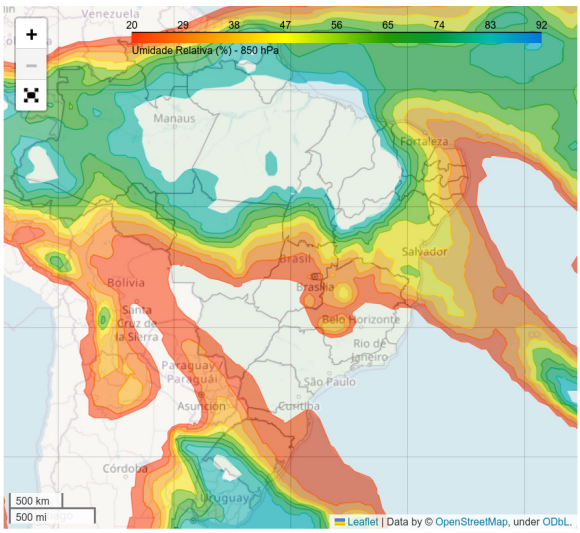

Figure 11. Visualization with Folium, geojsoncontour and Matplotlib.

5 Conclusion

During development, we observed that several widely used libraries, including Cartopy, Plotly, and Bokeh, did not fully satisfy the project's objectives for map generation. Cartopy, for example, did not provide the required level of interactivity, while Plotly and Bokeh could not effectively display meteorological field data alongside the map interface.

In contrast, the combination of Folium, Matplotlib, and Geojsoncontour proved highly effective for generating maps that incorporated the required meteorological field data. In addition, the Streamlit framework enabled the development of a simple, intuitive web interface for displaying and accessing the maps.

This study has some limitations that should be acknowledged. First, the evaluation was primarily qualitative and focused on functional behavior observed during prototyping rather than on a formal quantitative benchmark. Second, the experiments were conducted on a limited set of case studies and visualization scenarios, which may not fully represent all BRAMS operational contexts, variable types, and performance constraints.

Future work can extend this contribution along three directions. One direction is to add quantitative evaluation criteria, such as rendering time, memory consumption, interaction latency, and reproducibility across datasets. Another is to integrate the selected pipeline into automated operational workflows, including scheduled processing and continuous delivery of updated products. A third direction is to expand deployment to richer web dashboards and cloud-based environments, improving scalability, multi-user access, and long-term maintainability.

Overall, the project delivered both a simplified interactive web application for map visualization and a detailed comparison of Python libraries for the BRAMS model. The selected approach makes data manipulation and analysis significantly more accessible, reducing the need for prior programming knowledge among end users.

Declarations

Acknowledgements

We wish to express our gratitude to the Brazilian National Institute for Space Research (INPE) and the Brazilian National Council for Scientific and Technological Development (CNPq) for the financial support received.

Funding

This research was funded by the Brazilian National Council for Scientific and Technological Development (CNPq) under the Scientific Initiation scholarship (Process No. 163.139/2022-9).

Authors' Contributions

LAP is the main contributor ot this manuscript, and contributed tothe software development and manuscript writing. ESA and IKK contributed to the conception, research supervision, manuscript writing and re-vision. All authors read and approved the final manuscript

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

The software generated and/or analysed during the current study are available at: https://github.com/eugeniofatec/BRAMS and the datasets are available at: https://ftp1.cptec.inpe.br/pesquisa/sysadmin/eugenio/grads_G/

References

Arms, S., Chastang, J., Grover, M., Thielen, J., Wilson, M., and Dirks, D. (2020). Introducing students to scientific python for atmospheric science. *Bulletin of the American Meteorological Society*, 101(9):E1492–E1496. DOI: 10.1175/BAMS-D-20-0069.1.

Corte-Real, J. (2015). A importância da previsão do tempo na prevenção de riscos meteorológicos. *Finisterra*, 50(100). DOI: 10.18055/Finis7867.

Doty, B. E., Kinter III, J. L., Fiorino, M., Hooper, D., Budich, R., Winger, K., Schulzweida, U., Calori, L., Holt, T., and Meier, K. (1997). The grid analysis and display system (grads): An update for 1997. In *13th International Conference on IIPS for Meteorology, Oceanography and Hydrology, Amer. Meteor. Soc., Longbeach, CA*, page 117. American Meteorological Society.

Drach, R. and Caron, J. (2013). *Data Representation*, pages 25–37. Springer Berlin Heidelberg, Berlin, Heidelberg. DOI: 10.1007/978-3-642-36464-8_5.

Frants, M. and Ohman, M. D. (2024). Online visualization of geospatial cruise data with plotly python. In *American Geophysical Union, Ocean Sciences Meeting*. American Geophysical Union. ID AGU: DO14B-2493.

Freitas, S. R., Panetta, J., Longo, K. M., Rodrigues, L. F., Moreira, D. S., Rosario, N. E., Silva Dias, P. L., Silva Dias, M. A., Souza, E. P., Freitas, E. D., *et al.* (2017). The brazilian developments on the regional atmospheric modeling system (brams 5.2): an integrated environmental model tuned for tropical areas. *Geoscientific Model Development*, 10(1):189–222. DOI: 10.5194/gmd-10-189-2017.

Gupta, P. and Bagchi, A. (2024). Introduction to numpy. In *Essentials of Python for Artificial Intelligence and Machine Learning*, pages 127–159. Springer. DOI: 10.1007/978-3-031-43725-0_4.

Hoyer, S. and Hamman, J. (2017). xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, 5(1). DOI: 10.5334/jors.148.

Jolly, K. (2018). *Hands-on data visualization with Bokeh: Interactive web plotting for Python using Bokeh*. Packt Publishing Ltd.

Kadiyala, A. and Kumar, A. (2017). Applications of python to evaluate environmental data science problems. *Environmental Progress & Sustainable Energy*, 36(6):1580–1586. DOI: 10.1002/ep.12786.

Lin, J. W.-B. (2012). Why python is the next wave in earth sciences computing. *Bulletin of the American Meteorological Society*, 93(12):1823–1824. DOI: 10.1175/BAMS-D-12-00148.1.

Lynch, P. (2008). The origins of computer weather prediction and climate modeling. *Journal of computational physics*, 227(7):3431–3444. DOI: 10.1016/j.jcp.2007.02.034.

May, R. and Bruick, Z. (2019). Metpy: An community-driven, open-source python toolkit for meteorology. In *AGU Fall Meeting Abstracts*, volume 2019, pages NS21A–16. ID AGU: NS21A-16.

Moscato, R. (2024). *Web App Development Made Simple with Streamlit: A web developer's guide to effortless web app development, deployment, and scalability*. Packt Publishing Ltd.

Qian, Y.-K., Yuyang, H., Dapeng, W., Blanchard, A., and Jiaxi, L. (2025). miniufo/xgrads: v0.2.7.

Rautenhaus, M., Böttinger, M., Siemen, S., Hoffman, R., Kirby, R. M., Mirzargar, M., Röber, N., and Westermann, R. (2018). Visualization in meteorology—a survey of techniques and tools for data analysis tasks. *IEEE Transactions on Visualization and Computer Graphics*, 24(12):3268–3296. DOI: 10.1109/TVCG.2017.2779501.

Rew, R. and Davis, G. (1990). Netcdf: an interface for scientific data access. *IEEE computer graphics and applications*, 10(4):76–82. DOI: 10.1109/38.56302.

Shreemathi, M., Senthilkumar, B., Sujithra, S. M., Praisoodi, A., and Rithika, S. (2024). Mastering geospatial analysis with python: Understanding geopandas, gdal, fiona, matplotlib, data integration, and gis tools. In *Ethics, Machine Learning, and Python in Geospatial Analysis*, pages 120–149. IGI Global Scientific Publishing. DOI: 10.4018/979-8-3693-6381-2.ch006.

Sujithra, M. and Shanmugapriyaa, D. (2024). Spatial data visualization with python: Techniques, tools, and real. *Geospatial Application Development Using Python Programming*, page 123. DOI: 10.4018/979-8-3693-1754-9.ch005.

Suma, K., Sunitha, G., Avanija, J., Galety, M. G., and Varna, C. P. (2024). Geospatial data visualization with folium. In *Geospatial Application Development Using Python Programming*, pages 187–208. IGI Global Scientific Publishing. DOI: 10.4018/979-8-3693-1754-9.ch007.