TECHNISCHE UNIVERSITÄT ILMENAU
Institute for Practical Computer Science and Media Informatics
Faculty of Computer Science and Automation
Department of Databases and Information Systems

Research Project

In-Browser Raster Data Processing

submitted by

Vishnu Raj Cherukudi Mattuvayil
Matriculation Number 65429

supervisor:

Prof. Dr.–Ing. Kai-Uwe Sattler
Dr.–Ing. Marcus Paradies

Ilmenau, den September 1, 2024

Abstract

As the volume of geospatial data continues to grow exponentially, there is an increasing need for innovative methods that allow for efficient and real-time processing of this data within accessible platforms. Web browsers, being widely used and universally accessible, are emerging as a promising environment for such data-intensive tasks. This project explores the implementation of raster data processing directly within a web browser environment, focusing on client-side operations using modern web technologies. By leveraging WebAssembly (Wasm) and Pyodide, which is a Python runtime compiled to Wasm, this project demonstrates how large-scale climate datasets can be efficiently processed and visualized without leaving the browser. The report emphasizes the development of data loading mechanisms, a user-friendly interface, and the challenges encountered in optimizing memory management and performance within the browser context. Additionally, it references WebTensor, a similar high-performance raster data analysis framework, highlighting the strengths and limitations of each approach. The findings suggest that, with careful optimization, client-side processing can significantly enhance latency, interactivity, and scalability in climate data analysis applications.

Contents

1 Introduction 1

2 Background 3
- 2.1 Zarr . . . 3
- 2.2 Object Store and Minio . . . 3
- 2.3 Xarray . . . 4
- 2.4 WebAssembly (Wasm) Overview . . . 4
- 2.5 Pyodide Environment and Its Implications . . . 4
  - 2.5.1 Isolation of Pyodide Instances . . . 4
  - 2.5.2 Package Installation on Each Load . . . 5
  - 2.5.3 No Persistent Caching . . . 5
  - 2.5.4 Implications . . . 5

3 Literature Survey: DataLoader for Raster Data Processing 6
- 3.1 Analysis of Geospatial Data Loading . . . 6
- 3.2 WebTensor: Towards high-performance raster data analysis in the browser . . . 6

4 Comparison of Client-Side and Server-Side Processing 8
- 4.1 Client-Side Processing with Pyodide . . . 8
- 4.2 Server-Side Processing . . . 8

5 Implementation 9
- 5.1 Overview . . . 9
- 5.2 Data Acquisition and Storage . . . 9
- 5.3 Data Loading and Processing . . . 10
  - 5.3.1 Python for Raster Data Processing . . . 10
  - 5.3.2 Zarr Dataset and Programs . . . 10
  - 5.3.3 Implementation of DataLoader in Pyodide . . . 11
  - 5.3.4 Evaluation of the DataLoader . . . 12
  - 5.3.5 Memory Usage Considerations . . . 12
- 5.4 User Interface Implementation . . . 13

6 Result and Evaluation 14
- 6.1 Dataloader Memory Usage and Management . . . 14
- 6.2 Responsiveness and User Interaction . . . 14

7 Conclusion and Future Work 18

1 Introduction

The rapid advancement of satellite-based remote sensing technology has resulted in a significant surge in the volume and diversity of raster data over the past decade. These satellites incessantly observe Earth's geographical features, producing vast quantities of spatial data indispensable for environmental research, urban planning, disaster response, and environmental monitoring. Spatial data is generally classified into two main types: raster data and vector data. Raster data, exemplified by satellite imagery, is represented as a multi-dimensional array that encodes geographical information. Each cell or pixel within this array corresponds to a specific geographic attribute, such as temperature, elevation, or land cover. This grid-based format is particularly adept at capturing continuous phenomena, rendering it exceedingly valuable for studies related to environmental and Earth observation[SED+21].

In contrast, vector data represents discrete spatial features using geometric shapes such as points, lines, and polygons. This form of data is typically used to illustrate clearly defined entities, such as GPS coordinates, political boundaries, transportation networks, and other infrastructural elements. The distinction between raster and vector data is pivotal in geospatial science, as each necessitates unique processing techniques and analytical approaches. The synthesis of both data types is often essential to derive comprehensive insights into intricate spatial phenomena, enabling a deeper and more refined understanding of the geographic landscape[MMC21].

The processing of raster data has traditionally relied heavily on robust server-side infrastructures. This approach involves processing the data at its storage location or transferring it to a dedicated processing server. Subsequently, the processed data is typically delivered to end-users through web interfaces using technologies such as JavaScript and HTML. While effective for managing large-scale computations, this method presents challenges. Transferring large datasets between servers and clients can introduce significant latency, particularly in real-time applications. Furthermore, relying on centralized servers can result in performance bottlenecks and scalability issues, especially with the exponential growth in data volume [LZZ21].

The adoption of WebAssembly (Wasm) for client-side processing of raster data has led to recent advancements in web technology. Wasm, a low-level binary instruction format, enables code to execute in web browsers at near-native speeds, effectively addressing the performance drawbacks traditionally associated with JavaScript. By minimizing the necessity for extended data transfers between client and server, Wasm reduces latency, enhances user experience, and allows computationally intensive tasks to be completed directly in the browser. This transition from server-side to client-side processing could revolutionize the handling and visualization of raster data in web-based applications, opening up possibilities for more interactive and real-time data analytics[HRS+17].

The research effort is propelled by the availability of large-scale raster datasets, often sourced from international satellite networks such as the Copernicus program. These datasets play a crucial role in meeting the growing demands of climate research. Their immense size and

complexity necessitate the development of advanced tools capable of real-time data processing and visualization with utmost efficiency. Traditional server-side technologies often need to meet modern applications' high performance and interaction requirements. To address this, the project aims to leverage WebAssembly to process raster data on the client side, offering a scalable solution and near-instantaneous user feedback. The potential impact of this project on web-based applications, particularly in the field of climate research, is significant[LZZ21].

In an extension of WebTensor, a chunked tensor implementation for WebAssembly, this approach aims to enhance in-browser raster data processing. By utilizing a chunked tensor model to enable rapid data access and aggregation within the browser, WebTensor is designed to handle large raster datasets effectively. This project introduces a novel approach to client-side processing by incorporating Pyodide, a Python runtime compiled into WebAssembly. By leveraging Pyodide, the project seeks to merge Python's versatile toolkit with the performance capabilities of Wasm. Python is renowned for its extensive collection of scientific libraries [Nau23], while Wasm is designed to operate seamlessly within the browser environment.

The integration of Pyodide with current web technologies, such as HTML and JavaScript, presents both advantages and disadvantages. On the one hand, it enables Python-based analytical workflows to seamlessly meld with the dynamic features of modern web interfaces. However, ensuring responsiveness, particularly when handling large datasets, demands careful attention to memory management, data flow, and performance optimization. To optimize resource utilization while maintaining a high level of interaction for end-users, this project addresses these challenges by employing a modular approach to data loading, processing, and visualization[RHW+18].

This project focuses on implementing raster data processing within a web browser using WebAssembly (Wasm). Traditionally, data processing tasks have been handled on a server or transferred between servers, with the final output delivered to the client-side through JavaScript and HTML. This project shifts the processing burden to the client-side using WebAssembly, which enables near-native execution speeds by integrating Wasm files with JavaScript [Nau23].

The project builds on the concepts introduced by WebTensor—a framework for high-performance raster data analysis in the browser—and explores the use of Pyodide (a Python runtime compiled to Wasm) to enhance the processing capabilities of web-based applications. By running Python code directly in the browser, this approach seeks to combine the performance benefits of WebAssembly with the extensive data processing capabilities offered by Python libraries, ultimately providing a more interactive and scalable tool for climate data analysis.

2 Background

Raster data processing is essential for climate data analysis, which often requires handling complex, multi-dimensional datasets. Traditionally, these computationally intensive operations have been performed server-side, which can introduce latency and scalability issues, especially when processing large datasets or serving multiple users. Using WebAssembly we can reduce these performance issues by using client-side approach with near native execution speed. The terminologies used in this projcet are detailed below:

2.1 Zarr

Zarr is a powerful tool in the geospatial domain that helps manage large raster datasets efficiently. It uses a chunked storage approach enabling easy access to specific data subsets, making it ideal for massive geospatial imagery. Zarr is designed to handle growing data volumes without sacrificing performance, making it perfect for collaborative workflows and large-scale geospatial analysis on cloud platforms. It also offers faster loading times and improved responsiveness, especially with large datasets, thanks to its efficient data access mechanisms. Zarr supports various geospatial libraries and tools, promoting interoperability and enabling seamless data exchange across different software environments. However, Zarr, like NetCDF, lacks built-in support for geospatial referencing systems, which requires additional effort to define the spatial context of the data. Moreover, since it is relatively new compared to established formats like GeoTIFF, there are fewer readily available tools and resources specifically tailored for working with Zarr data, particularly within the geospatial domain.[GDA24]

Zarr is a versatile file format designed for efficiently storing large, multidimensional arrays, widely used by Python users due to its foundation in NumPy and an API similar to h5py. Like HDF5 and netCDF4, Zarr is a hierarchical, self-describing format with groups containing datasets, representing homogeneous data types. Its flexibility allows data to be stored in memory, on local file systems, or in cloud storage solutions like Amazon S3. In this project, Zarr is used to store raster data and is accessed through MinIO, making it an effective tool for handling large datasets in web-based climate data analysis[GHJK22].

2.2 Object Store and Minio

Object storage is a data management approach that stores information as objects, each composed of the data, metadata, and a unique identifier. Unlike traditional file systems that organize data hierarchically, object storage uses a flat structure, enabling it to efficiently manage large, unstructured datasets typical in cloud environments[GK21].

In this project, Minio is employed, an open-source object storage platform compatible with Amazon S3. Minio is designed for high-performance data management, offering scalability, fault tolerance, and seamless integration in distributed, cloud-native environments, making it ideal for handling extensive geospatial data[GK21].

2.3 Xarray

Xarray is a powerful Python library designed for handling and analyzing multi-dimensional labeled arrays, drawing strong inspiration from pandas' intuitive data structures but extending them to support arbitrary dimensions beyond the two-dimensional DataFrame. Built on Unidata's self-describing Common Data Model, which underpins the netCDF format, xarray is tailored for scientific data analysis, providing a well-defined framework for working with N-dimensional data. It integrates seamlessly with core Python scientific packages like NumPy, SciPy, Matplotlib, and pandas, while also offering support for various serialization and IO backends, including zarr, netCDF, HDF, and GRIB[HH17].

2.4 WebAssembly (Wasm) Overview

WebAssembly or Wasm is an efficient, low-level language that can be run on all major web browsers. It is designed for speed, safety, and portability, Wasm delivers performance close to that of native applications. It supports numerous programming languages, including C/C++, Rust, and Haskell, acting as a compilation target. This versatility allows developers to handle intensive computing tasks in Wasm while using JavaScript for manipulating the DOM, making it an effective tool for modern web development [DMAPS22].

Wasm also facilitates web software portability, allowing applications to run seamlessly across different environments with improved performance. It is intended to be the standard compilation target for various programming languages, enhancing the versatility and efficiency of web applications. Additionally, several compilers, such as Emscripten, Cheerp, AssemblyScript, and Asterius, convert code from different languages into Wasm, enabling developers to preserve and migrate legacy applications and write new web applications using their preferred programming languages[DMAPS22].

By enabling applications written in multiple programming languages to run efficiently on the web, Wasm represents a significant evolution in web development, reducing load times, increasing performance, and broadening the scope of what can be achieved within the browser[DMAPS22].

2.5 Pyodide Environment and Its Implications

2.5.1 Isolation of Pyodide Instances

Pyodide runs within a WebAssembly (Wasm) environment embedded in our browser's JavaScript engine. When a web page using Pyodide is loaded or refreshed, a new Pyodide instance is

started from scratch, meaning it doesn't save any data or state from previous sessions. This means that each session begins with a clean slate, and users must manage Python environment settings and data accordingly for every new session.

2.5.2 Package Installation on Each Load

Since each page load initializes a new Python runtime, any packages installed or variables defined during a previous session are not preserved. This means that all necessary dependencies, must be reloaded or reinstalled every time the page is refreshed. This process can be a little time-consuming, especially for larger packages, as it adds to the overall page load time.

2.5.3 No Persistent Caching

Due to the isolated nature of the Pyodide environment, even attempts to cache packages do not prevent the need for these packages to be reloaded into the new Wasm VM with each session. The lack of persistent caching means that developers need to be mindful of the performance impact of repeatedly loading and initializing large packages.

2.5.4 Implications

- **Performance**: The repeated loading and initialization of packages can lead to slower performance, particularly on pages that require several large packages but if the initial loading is not a problem for the application then it works smoothly afterwards
- **Persistence**: The inability to persist the Python environment across page reloads or between different pages limits the efficiency of long-running or state-dependent applications. While serialization and deserialization of state are possible, they do not mitigate the need for package reinstallation.

This project focuses on using Pyodide (a port of CPython to WebAssembly/Emscripten) to enable Python-based raster data processing within the browser. Running Python in the browser via Wasm allows us to merge the performance advantages of WebAssembly with the vast ecosystem of Python libraries.

3 Literature Survey: DataLoader for Raster Data Processing

Processing raster data, particularly in climate science and geospatial analysis, often involves working with large and complex datasets that need efficient management for loading, manipulation, and analysis. The data loader component is crucial in this workflow, ensuring that raster data is fetched, managed, and prepared efficiently for subsequent processing stages. This literature survey examines various approaches and technologies in data loading for raster processing, focusing on advancements that enhance performance, scalability, and ease of use.

3.1 Analysis of Geospatial Data Loading

With the rise in geospatial data collected from diverse sources like GPS devices, simulations, and social media, processing this data has become increasingly important. The efficiency of geospatial data analysis is primarily determined by the data loading process, where the choice of file formats and processing systems significantly impacts performance. Recent studies have explored the impact of different geospatial file formats and libraries on data loading performance. For instance, GeoParquet has been highlighted as providing the highest loading throughput, particularly when paired with systems like DuckDB, which can efficiently leverage parallelism. The correlation between data density—defined as the number of features per byte—and loading speed has been established, with denser files loading more rapidly. Additionally, microarchitectural analysis reveals that geospatial data loading generally exhibits high instructions per cycle (IPC), with instruction cache misses dominating in most cases, except for GeoParquet, where data misses prevail[WTZ24]. Despite these advancements, this method offers unique advantages by optimizing the loading process in scenarios involving massive raster datasets, where traditional vector-focused benchmarks may fall short. By addressing the challenges of file format selection and system compatibility, our approach ensures efficient, scalable loading of complex geospatial data, thus enhancing overall pipeline performance.

3.2 WebTensor: Towards high-performance raster data analysis in the browser

Raster data analysis has evolved with the development of various tools and technologies to meet the growing demand for processing large geospatial datasets. Recent efforts in optimizing data loading performance, such as GeoParquet, demonstrate a significant leap in loading throughput across different libraries, focusing on file formats' impact on data processing efficiency. Meanwhile, WebTensor, a new approach to raster data analysis in browsers, leverages WebAssem-

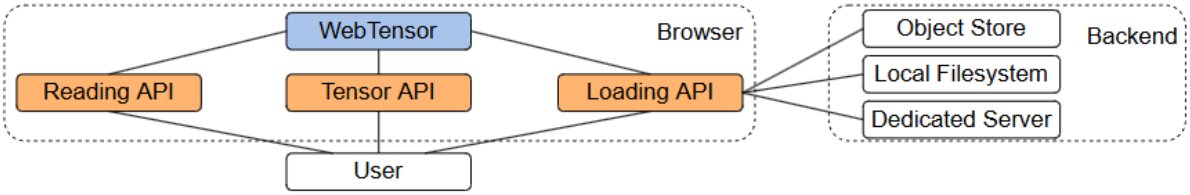

Figure 3.1: Architecture of WebTensor (orange: JavaScript components, blue: Wasm components)[Nau23].

bly to perform high-performance operations directly in the browser, overcoming JavaScript's limitations. WebTensor supports chunked data processing, enhancing its ability to efficiently handle large, multidimensional raster datasets. Compared to traditional methods, WebTensor's integration of tensor processing with WebAssembly offers substantial performance benefits in interactive and web-based applications[Nau23].

This project aligns with these advancements by further optimizing data loading and analysis. Lazy loading techniques minimize memory usage and reduce load times, particularly for large raster datasets, offering a more efficient approach to handling geospatial data. This approach benefits applications requiring high interactivity and precision, thus contributing to the ongoing efforts to improve raster data analysis performance in modern computing environments.

4 Comparison of Client-Side and Server-Side Processing

4.1 Client-Side Processing with Pyodide

In this project, client-side processing is implemented using Pyodide, allowing the entire data processing workflow to be executed within the user's browser. This is akin to the approach used by WebTensor, which also leverages Wasm for in-browser processing. By retaining data processing within the browser, this approach reduces latency, enhances interactivity, and offloads computational work from central servers, thereby increasing scalability.

However, client-side processing is not without its challenges. The performance is heavily dependent on the user's hardware capabilities. Furthermore, browsers are not inherently designed for large-scale data processing tasks, which can result in performance degradation or crashes when their computational limits are pushed. To mitigate these issues, this project utilizes chunked data loading strategies, such as those employed by WebTensor and Xarray, which optimize memory usage by loading only the necessary data slices, ensuring smoother performance. Additionally, ensuring consistent performance across different browsers and devices requires careful optimization and testing. Browser compatibility, memory management, and the increased energy consumption on mobile devices due to intensive processing tasks are also considerations that must be managed effectively.

4.2 Server-Side Processing

Server-side processing is a well-established method for handling computationally intensive tasks, offering numerous advantages. This approach centralizes all data processing activities on powerful servers, ensuring consistent performance across various client devices. The server's robust computational resources manage complex operations, making server-side processing particularly beneficial for applications dealing with large datasets, big data analytics, or other resource-intensive tasks.

However, server-side processing has its challenges. One major issue is the significant delay caused by transferring large datasets between the client and server. This delay can be particularly problematic for real-time applications requiring frequent updates. Additionally, slow or intermittent internet connections can greatly impact user experience, as server-side processing performance is heavily reliant on network quality. As the user base expands, the server may struggle to keep up with rising demand, leading to potential scalability issues and costly infrastructure upgrades for maintaining peak performance. Despite these challenges, server-side processing holds promise for cost savings, especially in resource management.

5 Implementation

5.1 Overview

The project uses a raster dataset stored in the Zarr format and hosted on MinIO, which acts as a cloud storage solution. These storage systems can be object stores, local file systems, or dedicated servers, but we are using object stores for efficiency. The core components, including the DataLoader, Raster Data Processing Library, and Reading API, are all written in Python and executed within the browser using Pyodide; refer to Figure 5.1. The user interface is built with basic HTML, CSS, and JavaScript, allowing for interaction with the data and visualizations directly in the browser environment.

5.2 Data Acquisition and Storage

The dataset utilized in this project was sourced from the Copernicus Climate Data Store (CDS) and covers 20 days, amounting to 2.6 GB. The data is stored in `.zarr` format, which is optimized for handling large, multi-dimensional arrays. The raster dataset contains various meteorological variables such as t2m, lsm, u10, u100, v10, and v100, which are commonly used in climate and weather data analysis:

- **t2m**: Temperature at 2 meters above the ground.
- **lsm**: Land-sea mask indicates whether a given location is land or sea.
- **u10**: U-component of wind at 10 meters, representing the east-west wind speed.
- **u100**: U-component of wind at 100 meters, also indicating the east-west wind speed but at a higher altitude.
- **v10**: V-component of wind at 10 meters, representing the north-south wind speed.
- **v100**: V-component of wind at 100 meters, indicating the north-south wind speed at 100 meters.

These variables are often used in weather forecasting models and climate research.

A significant aspect of raster data handling in this project is the adoption of chunked data storage formats, particularly Zarr. This format allows large datasets to be split into smaller, more manageable chunks that can be loaded into memory on demand. Zarr is especially well-suited for cloud storage systems and supports parallel I/O operations, reducing memory usage and improving performance.

The dataset was uploaded to a MinIO bucket, an object storage solution compatible with the Amazon S3 API for storage and access. This setup provides scalable storage and efficient data

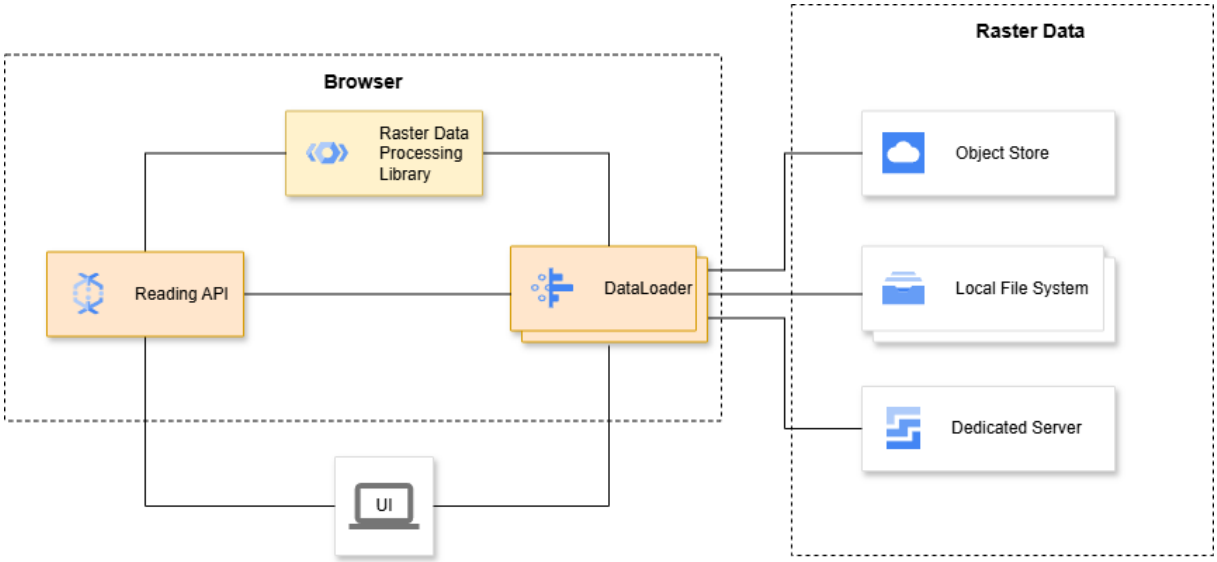

Figure 5.1: Architecture

retrieval through HTTP requests, similar to the backend storage methods used by WebTensor. MinIO ensures that large geospatial datasets are handled efficiently, leveraging cloud-based resources while maintaining data accessibility.

5.3 Data Loading and Processing

5.3.1 Python for Raster Data Processing

For using Python for the implementation, several factors were taken into account:

- **Ease of Use**: The learning curve for Python is less, and the language itself is more human-readable. It has strong library support for handling raster data, such as Xarray and NumPy.
- **Ecosystem**: Python's extensive collection of scientific libraries is widely used in climate research, providing ready-to-use tools for data analysis directly in the browser.
- **Compatibility**: WebAssembly enables Python to run efficiently in browsers. Python's flexibility, ease of use, and integration with scientific libraries make it well-suited for rapidly developing web-based applications.

5.3.2 Zarr Dataset and Programs

The project initially loaded data from the Copernicus Climate Data Store (CDS in Figure 5.2) and stored it in a MinIO bucket using Python. However, there were issues when attempting to load the dataset in Pyodide due to package loading problems, likely caused by incompatible wheel packages. Consequently, loading and slicing the dataset were attempted in JavaScript, though the primary focus remained on Python for its robust data processing capabilities.

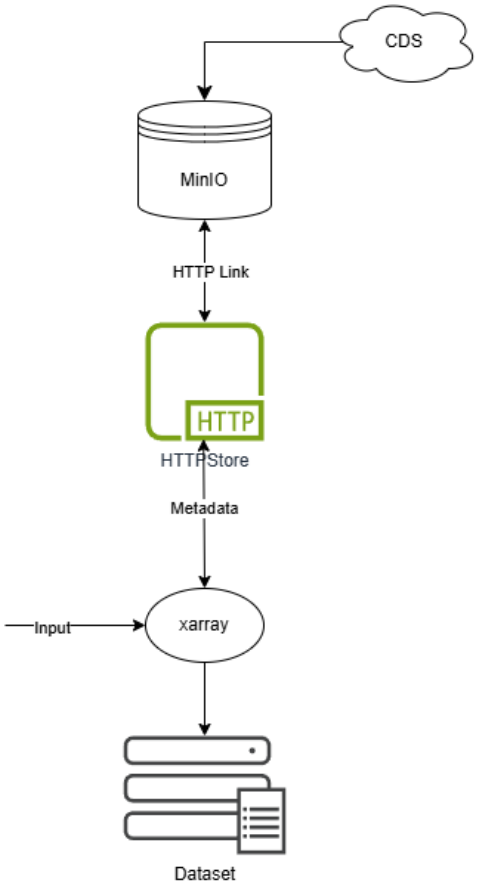

Figure 5.2: Dataloader

Lazy loading techniques were employed to address the challenges posed by large datasets. Rather than loading the entire dataset into memory, only the portions of data required for specific operations were loaded. This approach minimized memory consumption and reduced the time required to load data, enhancing the responsiveness of the application.

5.3.3 Implementation of DataLoader in Pyodide

To facilitate data loading, a loader program is tested in Python and adapted for use with Pyodide. The implementation involved loading the `.zarr` file from the MinIO bucket where it is hosted. In Zarr, HTTPStore is a type of storage backend that is developed on top of BaseStore, which is an interface for implementing different types of storage backends.

HTTPStore: This is used for reading Zarr arrays directly from an HTTP server. It allows users to access data stored in remote locations via URLs without needing to download the entire dataset. This is particularly useful for cloud-based data access, enabling efficient, on-demand loading of array chunks over the network.

MinIO, HTTPStore and xarray shown in Figure 5.2 provide flexibility in how Zarr datasets are stored and accessed, making it easier to manage and interact with large, multi-dimensional arrays across different environments. HTTPStore keeps a link to the MinIO bucket and transfers some metadata to xarray for efficient on-demand data loading in the form of labelled multi-

dimensional arrays. Using xarray, we can slice the data according to user-specified parameters such as time range, latitude, and longitude. These parameters are available from the metadata of the zarr dataset as labeled dimensions, since it is in the labeled form we can easily slice or select based on these dimensions. Finally, the output of xarray will be a sliced dataset according to the program needs that can be used for further processing.

5.3.4 Evaluation of the DataLoader

The DataLoader is designed to efficiently read and process large raster datasets stored in Zarr format, utilizing MinIO cloud storage. The evaluation of the DataLoader focused on several key performance metrics:

- **Execution Time:** The DataLoader's performance was assessed based on the time taken to load and process data over varying time ranges.
- **Scalability:** The DataLoader demonstrated the capability to handle large-scale datasets effectively, showing scalability in processing extensive spatial and temporal ranges. This is critical for applications that require high-resolution data analysis.
- **Slicing and Dicing Operations:** The DataLoader effectively performed slicing and dicing operations, which allowed for selective data extraction based on specific dimensions (e.g., time, latitude, longitude). This functionality enabled more granular data analysis and reduced the amount of data loaded into memory, further optimizing performance.
- **Resource Utilization:** By leveraging Pyodide for in-browser execution, the DataLoader minimized server-side computation, transferring processing tasks to the client. This strategy reduced server load and allowed for more efficient use of client resources.
- **Ease of Use:** The integration of Zarr and Xarray libraries simplified data access and manipulation, providing a user-friendly interface for developers working with complex raster data. This ease of use facilitated rapid development and integration into web applications.

Overall, the DataLoader's performance in this project underscores its suitability for modern web applications that require efficient, scalable, and versatile data processing capabilities, including advanced data slicing operations.

5.3.5 Memory Usage Considerations

WebAssembly (Wasm), the technology behind Pyodide, operates within the JavaScript engine of the browser rather than creating a separate virtual machine. As such, memory usage and performance are tracked alongside JavaScript in the browser's developer tools:

- **Memory Tab**: WebAssembly's memory usage appears within the JavaScript VM's memory. WebAssembly typically uses `ArrayBuffer` or `TypedArray` objects to manage memory, which can be observed in heap snapshots.
- **Performance Tab**: WebAssembly functions are often intermingled with JavaScript in call trees and flame graphs, sometimes labelled as `.wasm` or "native" code.

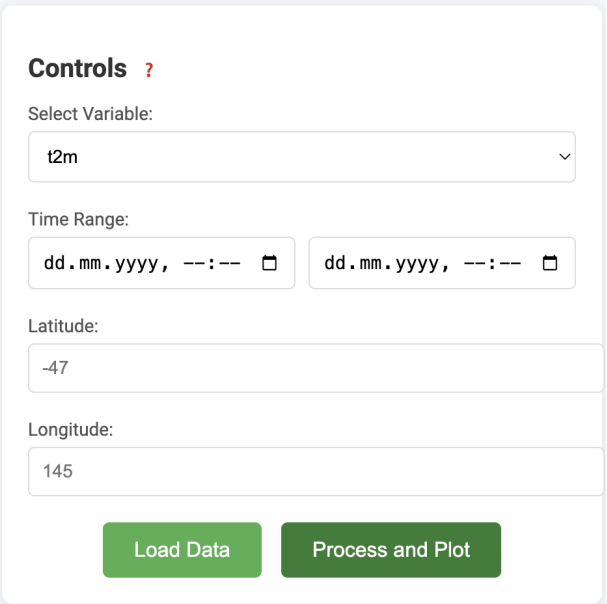

Figure 5.3: Dataloader Controls

- **Sources Tab**: The sources panel in developer tools may list `.wasm` files, where the compiled WebAssembly code can be inspected.

These considerations are important for optimizing the memory footprint and performance of the application, especially when handling large datasets. Integrating chunked storage and lazy loading techniques further contributes to managing the application's memory usage effectively.

5.4 User Interface Implementation

The user interface (UI) is designed for intuitive interaction with climate data, allowing users to select climate data variables, time ranges, latitude, and longitude for data retrieval, as shown in Figure 5.3. Instead of relying on JavaScript libraries for visualization, the UI utilizes Python's Matplotlib for generating plots. These plots are rendered in the browser via Pyodide, which allows Python code to run in the web environment. Updates to the visualizations are managed through the Document Object Model (DOM), with Pyodide dynamically updating Matplotlib charts based on user inputs. Similar to the plots, the output of the raster data processing library is also displayed in the UI, along with the time taken for each part of the module. This approach ensures seamless integration and responsive data exploration.

6 Result and Evaluation

6.1 Dataloader Memory Usage and Management

One of the main challenges encountered during the implementation was managing the memory efficiently, given that browsers have inherent limitations compared to dedicated server environments. The memory usage was closely monitored through the browser's developer tools, and it was observed that Pyodide, operating within the WebAssembly memory model, utilized memory in a similar manner to JavaScript as illustrated in Figure 6.1. The memory consumption for loading a data for a range of 5 days and 13 days as shown in Figures 6.2 and 6.3.

By leveraging chunked data formats such as Zarr, the project was able to handle large datasets more effectively. This approach minimized memory consumption by loading only the necessary data slices into memory. However, there is a trade-off in terms of the overhead associated with initializing the Pyodide environment and reloading packages during each session, which could lead to higher memory usage in some scenarios. The package reloading occurs whenever the page is refreshed not with every interaction of the page. Once the page is loaded, the page can be used like a single page application.

6.2 Responsiveness and User Interaction

The interactivity of the web application was another crucial aspect evaluated in this project as shown in Figures 6.4 and 6.5. Thanks to the client-side processing capabilities provided by WebAssembly, users experienced a high level of responsiveness when interacting with the data. For instance, adjusting the time range, latitude, and longitude inputs resulted in immediate updates to the data visualizations, providing a seamless user experience.

However, the responsiveness was somewhat dependent on the user's hardware capabilities. In cases where users operated the application on devices with limited computational resources, there was a noticeable lag, particularly during the initial data load and when manipulating large datasets. This finding underscores the importance of optimizing both the data loading strategy and the Pyodide environment to ensure consistent performance across different devices as shown in Figures 6.6 and 6.7.

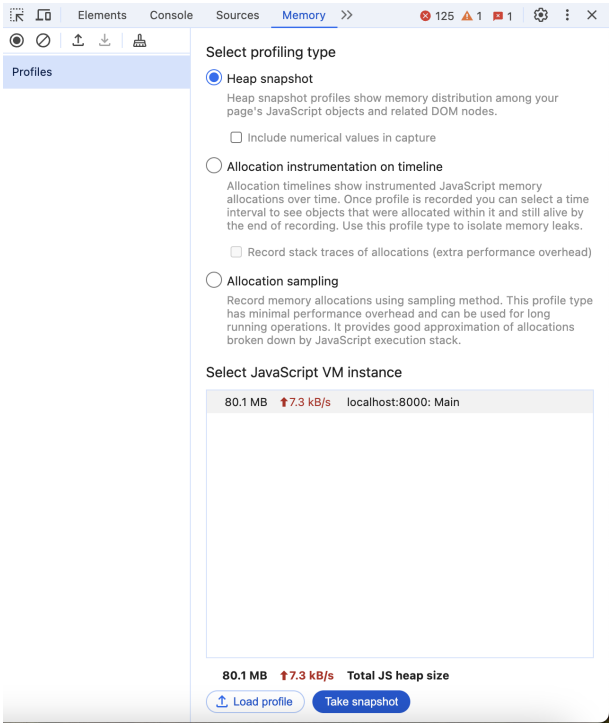

Figure 6.1: JVM Instance

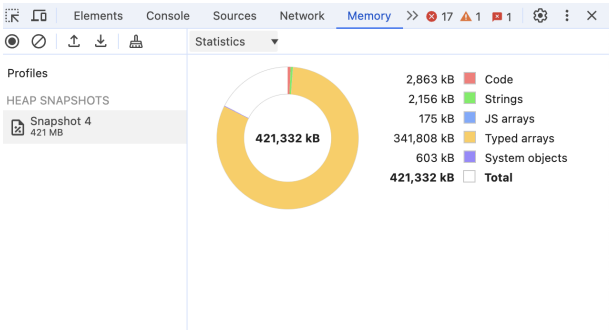

Figure 6.2: Memory Statistics for a 13-Day Period

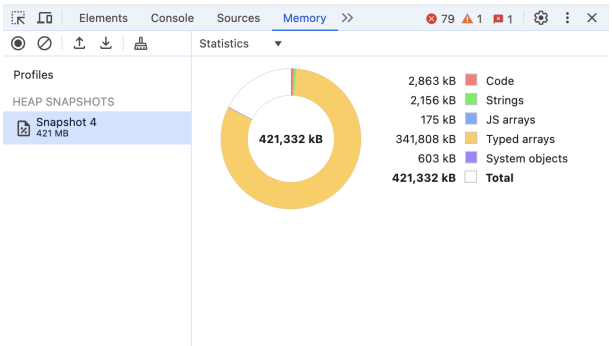

Figure 6.3: Memory Statistics for a 13-Day Period

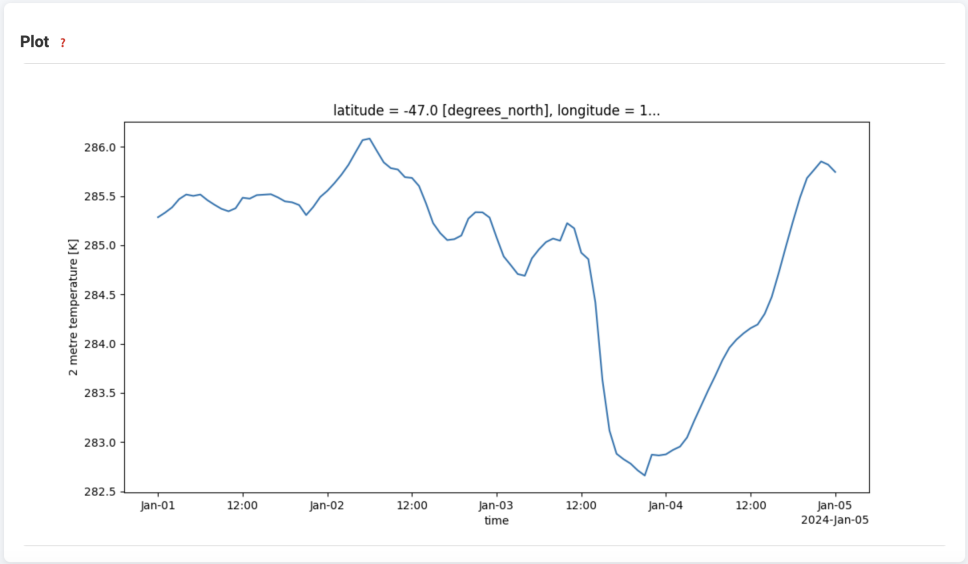

Figure 6.4: Data Plot for a 13-Day Period

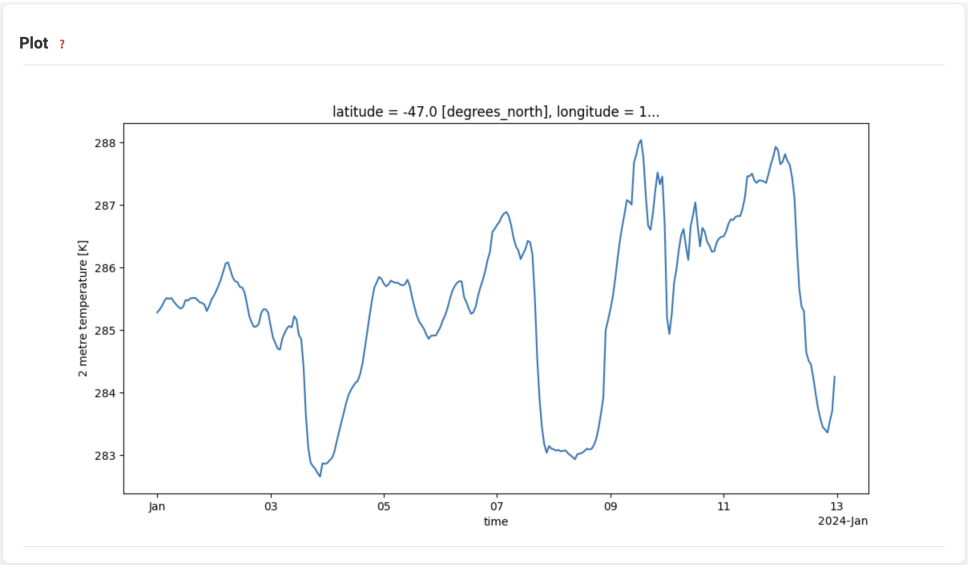

Figure 6.5: Data Plot for a 13-Day Period

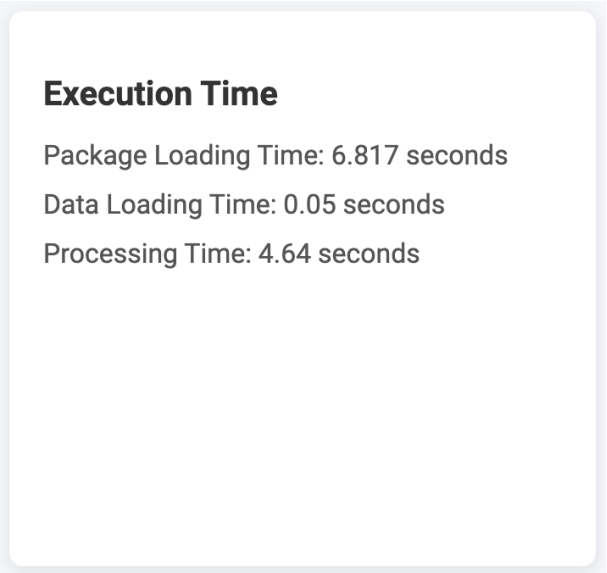

Figure 6.6: Execution Time for a 13-Day Period

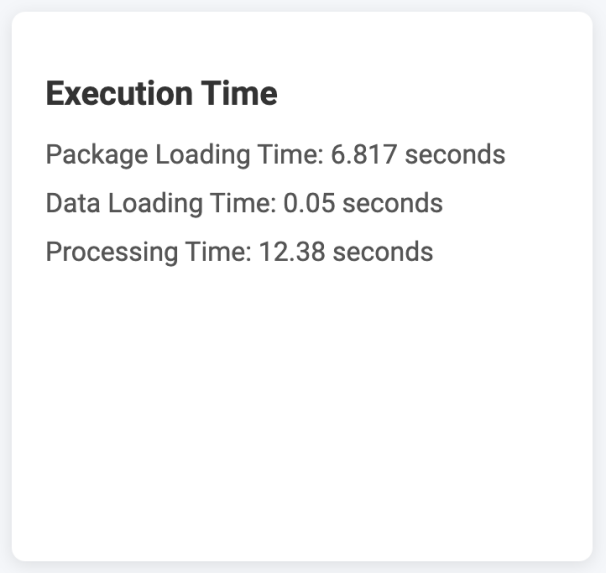

Figure 6.7: Execution Time for a 13-Day Period

7 Conclusion and Future Work

This project successfully demonstrates the feasibility of performing massive raster data processing directly within a web browser. By leveraging client-side processing with Python and WebAssembly, the project offers a scalable and responsive tool for climate data analysis. With reference to WebTensor[Nau23] this project highlights the strengths and potential improvement areas, particularly in handling large datasets with lazy loading and more straightforward computation of complex operations with Python. Finally, the user interface (UI) that reflects the backend functionalities.

Future work could optimize data loading and memory management to handle larger datasets better. Additionally, expanding the user interface to accommodate more complex analysis tasks and further enhancing the interactivity of visualizations could significantly improve the user experience. Lastly, implementing robust error-handling mechanisms would help the application maintain stability and prevent crashes due to unexpected errors.

Bibliography

[DMAPS22] João De Macedo, Rui Abreu, Rui Pereira, and João Saraiva. Webassembly versus javascript: Energy and runtime performance. In *2022 International Conference on ICT for Sustainability (ICT4S)*, pages 24–34, 2022.

[GDA24] GDAL contributors. Zarr, 2024. Accessed on March 7, 2024.

[GHJK22] Taylor A. Gowan, John D. Horel, Alexander A. Jacques, and Adair Kovac. Using cloud computing to analyze model output archived in zarr format. *Journal of Atmospheric and Oceanic Technology*, 39(4):449 – 462, 2022.

[GK21] Frank Gadban and Julian Kunkel. Analyzing the performance of the s3 object storage api for hpc workloads. *Applied Sciences*, 11(18), 2021.

[HH17] Stephan Hoyer and Joe Hamman. xarray: N-d labeled arrays and datasets in python. *Journal of Open Research Software*, Apr 2017.

[HRS+17] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. F. Bastien. Bringing the web up to speed with webassembly. *SIGPLAN Not.*, 52(6):185–200, 2017.

[LZZ21] Z. Liu, J. Zhang, and W. Zhang. Real-time big data analytics: Challenges and techniques. *IEEE Access*, 9:12215–12235, 2021.

[MMC21] J. Marmol, D. Morin, and J. Chou. Geospatial data processing and analysis. *IEEE Transactions on Geoscience and Remote Sensing*, 59(5):3394–3406, May 2021.

[Nau23] L. F. Naumann. Webtensor: Towards high-performance raster data analysis in the browser. In *Datenbanksysteme für Business, Technologie und Web (BTW 2023)*, pages 1083–1089, 2023.

[RHW+18] A. Rossberg, D. Herman, J. Wagner, B. Nelson, and L. Zakai. Webassembly: High-performance, safe, and portable code for the web. *Communications of the ACM*, 61(10):107–115, Oct. 2018.

[SED+21] S. Singla, A. Eldawy, T. Diao, A. Mukhopadhyay, and E. Scudiero. Experimental study of big raster and vector database systems. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2243–2248, 2021.

[WTZ24] Aske Wachs and Eleni Tzirita Zacharatou. Analysis of geospatial data loading. 06 2024.