Vector Rendering of Biomarker Topomaps

A Comparison of Direct Vector Visualization Pipelines Against Raster Visualization Pipelines for Rendering Topomaps of EEG Biomarkers

Guney Bayindir

Responsible Professor: Ricardo Marroquim
Supervisor: Arthur-Ervin Avramiea

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Guney Bayindir
Final project course: CSE3000 Research Project
Thesis committee: Ricardo Marroquim, Arthur-Ervin Avramiea, Thomas Abeel

Abstract

Electroencephalography (EEG) often relies on topographic scalp maps (topomaps) to visualize how biomarkers vary across the scalp. These visualizations can be generated many times during biomarker research, making higher efficiency pipelines important for reduced server load and latency. This research compares a raster based topomap rendering pipeline with a vector/SVG based pipeline in terms of server-side and client-side latency, and memory usage for EEG biomarker visualization.

A benchmark was implemented to measure the server-side cost of both pipelines over 100 iterations across 3 montage sizes and 5 interpolation resolutions using randomized EEG-like data. Within the tests, the vector pipeline was found to have server-side speedups between $2.88\times$ and $5.29\times$ while reducing memory usage between $1.24\times$ and $1.42\times$ dependent on the resolution. Stage-level analysis of the two pipelines showed that the raster pipeline was dominated by the rendering stage, whereas the vector pipeline distributed its cost mainly across interpolation and figure construction. However, improvements in server rendering came at the cost of client-side performance, with latency ratios ranging from $1.03\times$ to $0.50\times$ and memory-reduction ratios ranging from $0.96\times$ to $0.14\times$ depending on the resolution.

The results indicate that direct vector SVG rendering can significantly reduce server-side latency and memory usage. However, since the vector approach shifts part of the rendering work to the browser, the client impact remains an important concern. Overall, the proposed vector-based pipeline is a promising approach for modernizing EEG biomarker topomap visualization, but its benefits are dependent on its usage and the consideration of the client-side performance costs.

1 Introduction

"EEG is a non-invasive neuroimaging technique used to record the electrical activity of the brain via electrodes placed on the scalp" [17]. EEG is often used because of its non-invasive, relatively inexpensive, portable nature as well as its high temporal resolution compared to alternative neuroimaging techniques [17][12]. However, a potential issue with modern EEG scans is the post-processing step possibly requiring a large number of biomarker computations across the brain per subject, which can slow down research when large batches are requested. One post processing tool used for neurophysiological biomarker analysis is the Biomarker Toolbox (NBT), a tool originally written in MATLAB, that aims to provide the computational basis needed for biomarker processing, and analysis [9][20][19]. This project will investigate the visualization of biomarkers in a python implementation of NBT.

A central visualization in NBT is the *topomap* (topographic scalp map), which projects per-electrode biomarker values onto a 2D representation of the scalp and interpolates between electrodes to produce a continuous field which signifies the weight of the biomarker at a given (x, y) position over the head [25]. The current NBT topomap pipeline relies on Matplotlib [11] and MNE-Python [8], to rasterize the image before converting it into an interactive image. This rasterization pipeline can cause latency issues with larger inputs, and also cause artifacts when viewing the image in different resolutions [10].

An alternative to the rasterization pipeline is to render the topomaps as direct vector graphics (SVG) where the elements are described by scalable vectors instead of a bitmap [29]. SVGs can be less computationally expensive server side since the client side handles the rendering and the server only needs to calculate a function that generates the image instead of the image itself. This can potentially decrease the server load, while also solving the image artifacts since the client can re-render the image, or parts of an image, for different resolutions. As such an SVG rendering pipeline solution would have a different set of performance profiles and computational complexity both client and server side compared to the rasterized rendering pipeline with different drawbacks and advantages.

This paper aims to answer the research question **"How do raster-based and direct vector/SVG topomap pipelines compare in computational complexity, latency scaling, and visual/numerical fidelity across montage sizes and frequency-band grids?"**. To answer this question, the performance profiles of the current visualization pipeline will be investigated, a production ready SVG pipeline will be created and profiled, and both the implementations will be compared in terms of server, and client performance. The paper is structured such that section 2 discusses the background, and the literature surrounding the topic, section 3 discusses the methodology of the research, and the implementation details, section 4 discusses the results gathered from the experiment setup, section 5 discusses the successes and failures of the research, section 6 discusses the conclusions and recommendations for further investigation, and section 7 discusses the reproducibility of the research.

2 Background

Electroencephalography (EEG) is a brain imaging technique that places an arrangement of electrodes on the scalp and measures electric potential fluctuations caused by the activity of neurons [22]. Compared to other prominent brain imaging techniques, EEG is relatively inexpensive and portable [17].

EEGs use pre-determined electrode layouts, and channel frequencies for different purposes such as studying seizures, and identifying different biomarkers. A common layout is the International 10-20 layout, which has the electrodes spaced from each other in distances of 10% and 20% of the scalp, recommended by the International Federation of Societies for Electroencephalography and Clinical Neuro-physiology (IFSECN) [13][1].

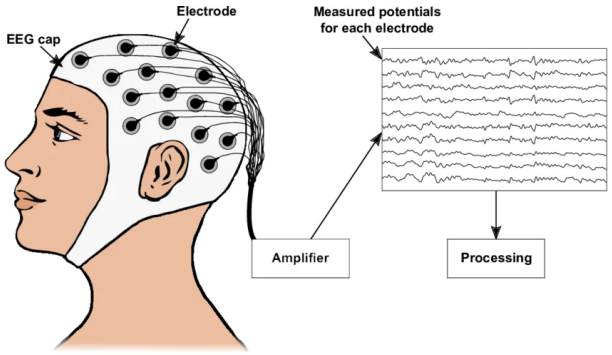


Figure 1: A sketch of EEG equipment (reproduced from [18]).

A sketch of EEG recording can be found in Figure 1. Although the 10-20 setup mentioned above typically has 21 electrodes, EEG electrode arrangements can also have denser arrangements of up to 256 electrodes [23]. Furthermore, each of the channels is recommended to have a minimum sampling rate of 256Hz and with high definition EEGs going upwards to several kHz [24][7], thus making processing even more complicated.

2.1 EEG Data Processing and Biomarkers

EEG scan processing involves pre-processing for removing artifacts and post-processing to compute biomarkers.

2.1.1 Pre-Processing

Raw EEG data contains noise and artifacts from many different sources such as the equipment itself, the scalp, and brain activity [6] [12]. Examples of these artifacts are muscle twitching, eye blinking, head movements, poor electrode connections, line noise from power systems connected to the electrode etc. So before computation of biomarkers, these artifacts are removed via filtering, artifact detection, artifact correction, channel rejection or interpolation [12][6]. There can also be manual removal of certain time boundaries if a major artifact is detected after the automatic pre-processing steps [6].

2.1.2 Post-Processing and Biomarkers

After pre-processing is completed, the data is analyzed for biomarkers. Biomarkers are quantitative features that attempt to predict harder to assess health conditions [2]. Some examples of common biomarkers include blood pressure and blood sugar level, which are used to measure overall health, and diabetes level respectively. Biomarkers are important as they can provide numerical values to diagnose potential underlying conditions. Although there are different biomarkers that can be used, this research focuses on a set of biomarkers based on the spectral power of the signal. EEG frequency ranges are often grouped into bands, such as delta, theta, alpha, beta, and gamma. These bands can be shown as follows [3]:

$$\delta : 1\text{–}4\,\text{Hz}, \quad \theta : 4\text{–}8\,\text{Hz}, \quad \alpha : 8\text{–}13\,\text{Hz},$$
$$\beta : 13\text{–}30\,\text{Hz}, \quad \gamma :> 30\,\text{Hz}.$$

After extracting and computing the absolute power of each electrode, the areas in between the electrodes are interpolated using the found values [25]. The resulting scalp can then be displayed through various means such as, the main focus of this research, the topographic scalp map.

2.2 EEG Scalp Topomaps

A topographic scalp map is a topographic map, or topomap, representing the scalp. Topomaps display three-dimensional information on a two-dimensional grid by using color to represent elevation.

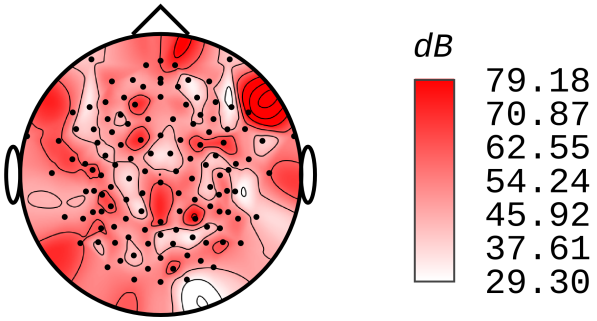


Figure 2: An example plot of EEG Scalp Topomaps

An example plot can be found in Figure 2. Within the figure, varying shades of red are used to represent points where a certain biomarker is greater than a reference point, and varying shades of blue if a biomarker is less pronounced than a reference point [16]. This color variation creates a terrain which is then separated by contour lines to show the difference in biomarker weights.

2.2.1 MNE Topomap Creation

Topomaps are generated by the aforementioned interpolation between the electrodes. More specifically, the points are made into a mesh consisting of triangles via Delaunay Triangulation, which is a technique that connects points across a plane such that no point lies in the circumcircle of any triangle [5]. Once a mesh over the scalp is created, the points within these triangles are then interpolated using Clough-Tocher interpolation [4]. One issue that can be found with this setup, is that the mesh created from the electrodes does not cover the entire scalp, meaning there are areas between the boundaries of the scalp and the outer electrodes that are not included in the mesh, and not interpolated. To fix this, ghost points are added outside of the scalp by assigning them the extrapolated weights of the boundary electrodes and added to the mesh [16].

2.2.2 MNE Complexity Analysis

The computational complexity of the MNE topomap pipeline can be written in terms of 2 variables: the number of electrodes C and the interpolation grid resolution R, where the grid is $R \times R$ pixels. Both the vector and the raster pipelines use MNE for interpolation and contour generation causing some of the complexity to be shared.

Interpolation Step. The main interpolation step has two main contributors to complexity with Delaunay Triangulation and Grid Evaluation. The electrodes are triangulated using the Qhull implementation of Delaunay Triangulation [5] which runs in $O(C \log C)$ time. A small number of ghost points are also added for boundary calculations however they do not affect the cost since their amount is bounded.

Then the interpolant is constructed using the Clough-Tocher scheme, which builds a continuously differentiable surface over the triangulation [4]. This requires gradient estimates at each vertex, computed using the global algorithm described in Nielson [21] and Renka and Cline [27] as implemented in SciPy [28]. The algorithm runs k many times each doing $O(C)$ work where k is bounded by a tolerance and a maximum iteration count leading to a cost of $O(C)$.

Then the R^2 many coordinates in the grid are evaluated using the interpolant. The ghost points are also assigned points, however due to their limited number, they do not contribute to the complexity. This leads to R^2 calls doing constant work per call which gives a time complexity of $O(R^2)$. In total, the interpolation stage has a combined complexity of $O(R^2 + C \log C)$

Contour Generation Step. The contour lines are generated using the marching-squares algorithm [14], which does constant work on each of the $(R-1)^2$ grid cells. Thus leading to a complexity of $O(R^2)$ for the contour generation.

Raster-Only Passes. In the raster pipeline, the interpolated grid is rendered by Matplotlib's Agg backend. The figure DPI is set to R, with the figure size set to (2,2) causing the output to be $4 \times O(R^2) = O(R^2)$ pixels. This rendering causes extra $O(R^2)$ passes that do not exist in the vector pipeline.

Vector Pipeline. In the vector pipeline, the interpolation grid is produced via the same $O(R^2 + C \log C)$ evaluation. However, instead of rasterizing, the $R \times R$ array is passed to a `go.Heatmap`. The browser performs the colormap mapping and clipping at display time. This eliminates the raster-only $O(R^2)$ passes listed above.

Summary In summary, the shared pipeline complexity is $O(C \log C + R^2)$. The raster pipeline adds additional $O(R^2)$ passes, giving it a higher constant factor on the R^2 term compared to the vector pipeline. The topomap is generated via the following process:

- A 2-dimensional rendition of the scalp is generated and the electrodes are projected onto said scalp.
- Ghost points are added outside of the scalp by extrapolating the electrode values so the interpolation mesh covers the entire scalp.
- Delaunay Triangulation is used to create a mesh over the electrode coordinates.
- The insides of the triangles are interpolated, commonly with Clough-Tocher interpolation using the values from the electrodes in the edges of the triangles.
- Figures such as contour lines, and labels are added, and the mesh outside of the scalp is masked.
- The created image is then rendered using a rendering technique such as rasterization or vectorization.

2.3 Rendering

This research mainly focuses on the rendering stage of this pipeline, more specifically a comparison between rasterization, and direct vector generation techniques.

2.3.1 Raster Rendering

Raster rendering represents an image as a rectangular grid of pixels [10]. Each pixel stores a color value. In the case of rasterized rendering of an EEG topomap, the interpolated field is sampled over a fixed resolution grid.

The main advantage of raster rendering is that it is widely supported and easy to display. Libraries such as Matplotlib [11] and Plotly [26] can generate, display, and export raster images efficiently. The main issue with raster rendering is its resolution-dependence. Since the image is stored as a fixed grid of pixels, rescaling the image at a different resolution than its intended one can introduce artifacts [10] such as the one seen in Figure 3. For example, if the image is zoomed, or resized, blurry visuals, and pixelated images can be spotted. Increasing the resolution of the source image reduces these artifacts but it also introduces a proportional computational, memory, and file size cost since each of the pixels within the grid needs to be calculated by the image generator. This creates a further problem as topomaps are generally requested from the image server many times per biomarker, further increasing the computational demand the image server needs to cover.

2.3.2 Vector and SVG Rendering

Vector rendering represents an image using geometric objects rather than pixels. Instead of storing the color of every pixel, a vector image stores shapes, vectors, polygons, gradients, etc. and style attributes. Scalable Vector Graphics (SVG) [29] is a common XML-based format for vector graphics on the web.

In an SVG-based topomap rendering pipeline, the visualization can be represented using vector primitives, and interpolation fields. The scalp outline can be drawn as a circle, the nose and ears as simple polygons such as triangles, and the electrodes as circles. And since the biomarker field is generated via a set of mathematical functions, the interpolated biomarkers can be represented as an interpolation field, and the contours as a set of points that generate a gradient.

The main advantage of SVGs over rasterized images is that the image itself is resolution independent. Since the browser renders the shapes at display time, the image can be scaled for different resolutions, or a part of the image can be zoomed in without the artifacts found in rasterized images which can be seen in Figure 3.

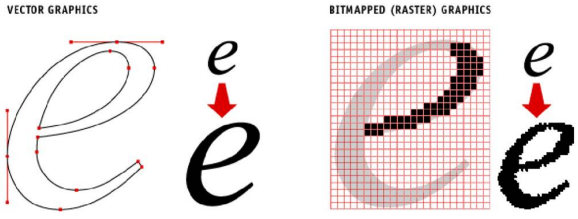


Figure 3: Comparison of raster and vector graphics artifacts (reproduced from [30], CC BY).

Another potential benefit is the reduced server load. Since the image server calculates a set of mathematical equations that generate the image and not the image itself, like in rasterization, the server load per image can be reduced for less complex images, and is also resolution independent. However, while the server load is decreased per image, it is offloaded into the client side since the client's browser needs to render the image from the SVG. This can cause issues if the images are complex, too many images are rendered at the same time, or the client has a computationally limited device, which wouldn't have been an issue with server-side rendering. The core trade-off can therefore be summarized as follows:

$$\text{Raster cost} \approx \text{interpolation grid size} + \text{image encoding cost},$$

$$\text{Vector cost} \approx \text{geometric complexity} + \text{browser rendering cost}.$$

This distinction motivates the central comparison of this research: whether a direct vector/SVG topomap pipeline can provide better latency scaling, computational efficiency, and visual fidelity than a traditional raster-based pipeline across different montage sizes and topomap grids.

2.4 Relevance to This Research

The Neurophysiological Biomarker Toolbox (NBT) is a scientific platform specializing in computing and visualizing EEG biomarkers for neurophysiological analysis [9][20][19]. A central visualization method that NBT uses is the topomap, because it allows researchers to inspect how biomarkers are distributed along the scalp. And since topomaps are generated repeatedly, and in batches for many subjects, biomarkers, frequency bands, or conditions, the performance of the topomap visualization pipeline becomes an important concern to decrease latency.

This research focuses on comparing raster-based and direct vector/SVG topomap pipelines. The comparison requires an understanding of topomap generation pipelines including:

- The generation of the biomarker field, over the topomaps
- The rendering of said biomarker field via Rasterization or Direct Vector SVGs.
- The finalization of the topomap by adding contours, figures etc.

The performance profile of the individual steps in the topomap creation pipeline is unknown, as such the possible latency, and memory impact of rasterization versus direct vector rendering, within the pipeline are also unknown. Furthermore the server side and the client side impacts of the two rendering methods have not been investigated. And while there is a core trade-off between rasterized rendering and vector rendering, it is currently not possible to tell whether one should be preferred over the other, or in which cases one should be preferred over the other, due to this lack of knowledge.

3 Methodology

This study is structured into two sections to meet the two prominent gaps in knowledge, profiling and comparative analysis of rasterized and vectorized topomap implementations in terms of computational profiles and the resulting visuals. The two sections were then addressed by the following four sub-questions:

- **SQ1 — What is the performance profile of the rasterization pipeline?** What is the per-stage breakdown of end-to-end latency in NBT's current topomap implementation.
  Success criterion: Profiles and attributions of major computational time for the pipeline stages.
- **SQ2 — What is the performance profile of the vectorized pipeline?** What is the per-stage breakdown of end-to-end latency in the vectorized topomap implementation, using the same benchmark as SQ1.
  Success criterion: Profiles and attributions of major computational time for the pipeline stages.
- **SQ3 — How does the server-side performance of rasterized and the vectorized topomap visualisation pipelines compare for different input sizes?** How do the different topomap visualization implementations compare in terms of server performance for different set of inputs, and resolutions.
  Success criterion: A benchmark of the server latency for both pipelines across different input parameters.
- **SQ4 — How does the client-side performance of rasterized and the vectorized topomap visualisation pipelines compare for different input sizes?** How do the different topomap visualization implementations compare in terms of client/browser performance for different set of inputs, and resolutions.
  Success criterion: A benchmark of the client latency for both pipelines across different input parameters.

To answer these sub-questions the relevant code within the NBT implementation was found to be in nbt/visualization/plots/topography.py. Additional folders were also made to house the various scripts and results related to the project such as: project-scripts/ that contains python scripts related to the project, such as the benchmark itself, and the sub-questions, and a results/ to contain the results of the conducted tests. A pyproject.toml, was also included for python package management.

3.1 Benchmark

To compare the raster-based and vector-based topomap rendering pipelines, a benchmark suite was implemented. The suite was designed to be used universally with different implementations, and different pipelines, to allow for flexibility and further research.

The benchmark takes 3 arguments: the number of runs the benchmark should run, the number of warm-up runs so the initial uncached runs do not stand out as outliers, and a series of FunctionHolders. This FunctionHolder abstraction works as a way to pass individual functions and stages to the benchmark, so that the per stage performance can be analyzed. Each FunctionHolder stores the name of the stage, the function itself, a list of args for the function, a dictionary made up of names, and arguments for named arguments the

function can take, the names of the outputs the function produces and a dictionary that matches the names of the outputs to the outputs themselves, which is filled after the function is run. For stages that depend on the outputs of earlier stages, the arguments can also be set to a reference from a previous function to allow the benchmark to account for dependence on previous functions.

The benchmark runs the FunctionHolders in order with the arguments provided, and collects the duration, and the memory usage of the function runtime via Python's time.perf_counter, and tracemalloc.get_traced_memory functions respectively. The python garbage collector is run and disabled then re-enabled for each function run as the garbage collector was found to interfere with results. Memory usage and duration are collected separately since memory measurements impact the time measurements. The benchmark repeats this process for twice the amount specified, once for time measurements and once for memory measurements. Before collecting these measurements, the benchmark also performs warm-up iterations, which are not collected, since the first few runs were found to be outliers compared to the rest of the data, most likely due to matplotlib libraries not being cached during the first few runs. The data generated for the benchmarks are random as to not use confidential medical data.

3.2 SQ1

Within the SQ1 branch, the current rasterized implementation of topomap was refactored and divided into 6 main stages to allow for each stage to be benchmarked:

- Interpolation and rendering of the data into a rasterized topomap
- Normalization of the channel coordinates
- Building the plotly figure using the render
- Adding electrodes to the plot
- Adding Colorbar traces to the plot
- Applying Topomap Layout

3.3 SQ2

A similar process was made within the SQ2 branch, with the vector image rendering being implemented and divided into five stages:

- Rendering of the data into a vectorized topomap
- Building the plotly figure using the render
- Adding electrode to the plot
- Adding Colorbar traces to the plot
- Applying Topomap Layout

The first two stages were built specifically for the vector rendering pipeline, and the rest were reused from SQ1 since the implementations already produced vector images, or didn't affect the rendering at all.

During the implementation of the first two stages, different interpolation and contour algorithms were also tested. However, they were not used since during the implementation of SQ4, they were found to give inaccurate/inconsistent results compared to the rasterized pipeline. As such, MNE was used to interpolate and draw contour lines, without directly rendering the result.

3.4 SQ3

Both pipelines were designed to accommodate different resolutions and montage sizes. Here resolution refers to the interpolation resolution of the grid, while montage sizes refers to different arrangement of electrodes. Different resolutions were tested by passing different values to the resolution parameter in MNE.viz.plot_topomap other than the default value of 300. The different montage sizes were acquired from the function: mne.channels.make_standard_montage by passing names of different montage configurations. The latency and memory was tested and reported, and the html of the output was generated per resolution.

3.5 SQ4

The html outputs mentioned in SQ3 were then loaded and further experimented. The latency of loading the page, interacting with the plot, and the memory of the page were recorded and compared using Playwright [15].

4 Experimental Setup and Results

Data was collected using a Lenovo Legion Slim 5, 16APH8, with 32 GB of RAM, an RTX 4070M, and a Ryzen 7 7840HS. A Linux system, running Arch by the way, with KDE Plasma was used to conduct the tests. The tests were run using Pycharm, and the CPU mode set to performance. No other significant application was open or used during the runtime to ensure as little interference as possible with the results.

The benchmark was conducted with 100 iterations and 3 warm-up iterations. Three montage sizes were used: easycap-M1 with 74 electrodes, GSN-HydroCel-128 with 128 electrodes, biosemi-256 with 256 electrodes. The montages were used to generate random data per electrode to avoid the use of confidential EEG recordings while preserving the same per-channel data structure required by the topomap rendering pipelines. Although the default image resolution used for the topomap is 300x300 pixels, image resolutions of 100x100, 300x300, 500x500, 1000x1000, 2000x2000 pixels were tested to observe the effects of resolution on the performance of the two pipelines. Playwright with chromium was used for client-side testing.

Side	Ch.	100	300	500	1k	2k
Srv.	74	5.29	4.62	3.85	3.20	2.88
	128	5.14	4.60	3.87	3.22	2.96
	256	4.17	3.86	3.87	3.21	2.91
Cl.	74	1.01	0.96	0.93	0.78	0.53
	128	0.96	0.98	0.92	0.79	0.53
	256	0.96	1.03	0.93	0.79	0.50

Table 1: Ratios of the raster pipeline latency over the vector pipeline latency across resolutions and channel counts.

Table 1 shows the runtime speedups, calculated by taking the mean of the total results: $TotalResult_{Raster}/TotalResult_{Vector}$. From the table, it can be seen that the vector pipeline has a substantially lower average server cost compared to the raster pipeline across resolutions and montage sizes. The runtime speedup fluctuates between $5.29\times$ and $2.88\times$, and the larger speedups occur in smaller resolutions. Although resolution seems to be correlated with server performance, montage sizes do not seem to be correlated.

The client-side measurements reveal the server-client tradeoff discussed with the vector approach. The client-side results show the opposite trend from the server-side results. As resolution increases, the vector output becomes slower relative to the raster output, with speedup ratios ranging from $1.03\times$ to $0.50\times$ depending on the resolution. The montage sizes do not seem to correlate with the latency, suggesting that the resolution of the topomap is the main deciding factor of the latency.

Side	Ch.	100	300	500	1k	2k
Srv.	74	1.42	1.38	1.36	1.38	1.37
	128	1.41	1.40	1.36	1.37	1.38
	256	1.24	1.30	1.37	1.38	1.38
Cl.	74	0.90	0.71	0.51	0.33	0.14
	128	0.93	0.78	0.62	0.33	0.17
	256	0.96	0.75	0.65	0.36	0.15

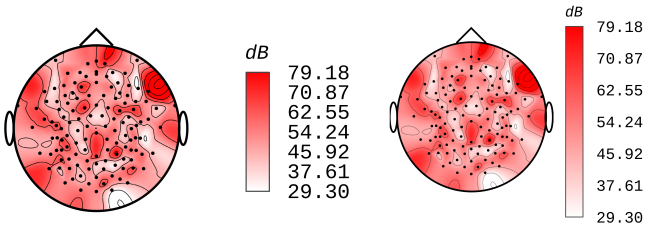
Table 2: Ratios of the raster pipeline memory usage over the vector pipeline memory usage across resolutions and channel counts.

Table 2 shows the mean memory reduction between the raster and the vector pipeline, calculated by $TotalResult_{Raster}/TotalResult_{Vector}$. The server side measurements show a moderate improvement over the raster pipeline, ranging from $1.24\times$ to $1.42\times$ less memory usage. Unlike the latency measurements, the memory usage is consistent across resolutions and montage sizes. In contrast, client side memory usage is below 1 for all of the measurements, more specifically between $0.96\times$ and $0.14\times$ and decreases with higher resolution. This further illustrates the server-client trade-off of the vector rendering pipeline.

Side	Stage	Raster (ms)	Vector (ms)	Speedup
Server	Interp./render	129.90	13.06	9.95×
	Norm. coords	3.37	–	–
	Build fig.	25.00	18.72	1.34×
	Electrodes	0.50	0.47	0.99×
	Colorbar	0.53	0.52	1.07×
	Layout	2.40	2.43	1.02×
	Total	161.70	35.20	4.59×
Client	Load page	662.46	697.37	0.95×
	Zoom in	11.1	29.52	0.38×

Table 3: Per-stage runtime breakdown for raster and vector pipelines using GSN-HydroCel-128 and 300×300 resolution.

Table 3 shows the mean per stage costs of raster and vector rendering pipelines. From the table, it can be gathered that the raster pipeline is dominated by the `render_mne_topomap` stage with a 129.90 ms runtime. In contrast, the most dominant stage in the vector pipeline is `build_vector_figure` at 18.72 ms, followed by `interpolate_topomap` at 13.06 ms. The remaining shared stages, such as adding electrode traces, adding the colorbar, and applying the layout, contribute only a negligible part of the total runtime in both pipelines. The vector pipeline shows improvement in both of these stages as the rendering stage finishes $9.95\times$ faster, and building the figure finishes $1.34\times$ faster in the vector pipeline. For the client side impact, the load page metrics have similar results in both pipelines. The main difference is the zoom in stage where there is a slowdown for vector rendering. The server-side total speedup is $4.59\times$.



(a) Raster Topomap **(b)** Vector Topomap

Figure 4: Raster and vector topomap output for GSN-HydroCel-128 at 300×300 resolution.

The visual output of the vector and raster renders, an example of which can be seen in Figure 4, were also tested for each of the results, however since the interpolation algorithms used are the same, no difference was found between the results.

5 Discussion

The results show a variety of improvements and drawbacks between raster and vector rendering as well as confirmation of theoretical analysis.

5.1 Server Side Performance Analysis

The results show a significant performance difference between raster and vector topomap rendering pipelines on the server side.

5.1.1 Latency Comparison

All tested montages and resolutions were found to be faster with the speedup ranging from $2.88\times$ to $5.29\times$ depending on the resolution. One of the reasons for this seems to be the complexity of the interpolation not being high enough to dominate the rendering stage. This indicates that the server side rasterization of the image is the dominant contributor to the server side rendering cost, which doesn't exist in the vectorized rendering pipeline.

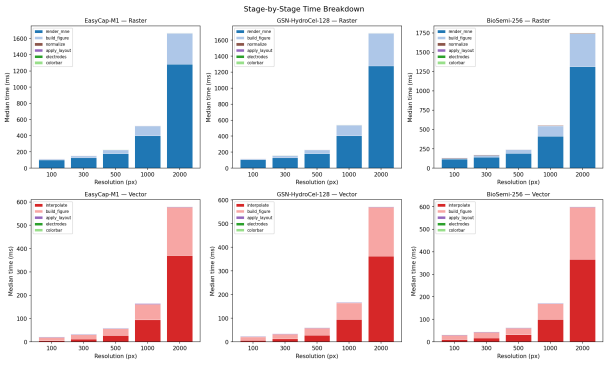


Figure 5: Stage latency per resolution for raster and vector rendering across montage sizes

As it can be seen from Figure 5, the advantage of the vector pipeline diminishes with greater resolutions, hence showcasing that the vector pipeline still contains stages that are resolution-dependent. This can be attributed to the interpolation costs, and contour generation costs. The interpolation depends on the resolution, which dominates at larger resolutions. The `build_figure` stage also becomes more significant for both pipelines with increasing resolutions. This is also consistent with the complexity analysis as both pipelines have $O(R^2+C\log C)$ complexity due to shared stages. However, the rasterization process still has to render the image server side suggesting that although the advantage diminishes with increasing resolution, there would still be an advantage for larger resolutions.

5.1.2 Memory Comparison

The server side memory results show a consistent but lesser improvement compared to server side latency.

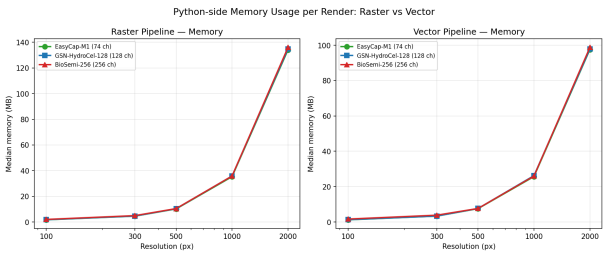


Figure 6: Memory usage per resolution for raster and vector rendering across montage sizes

The raster pipeline uses $1.24\times$ to $1.42\times$ more memory independent of montage sizes and resolution as seen in Figure 6. The improvement in memory usage can be attributed to the omission of the rasterization step. The two major memory usages in the raster pipeline are the storage of the interpolation grid and the rasterized image which are both $O(R^2)$. Thus the complexity ratios between the two implementations is constant and can be estimated to be 1.36 times each other.

5.2 Client Side Performance Analysis

The client-side measurements show the main trade-off of the vector approach. Since the vector rendering off-loads some of the rendering costs to the browser, the client-side performance of the vector rendering is equal to or worse than that of raster rendering. The load times and the zoom times of the vector render are similar in lesser resolutions and become worse with greater resolution.

It is also seen in the memory measurements, where the vector output uses more memory than the raster output in every tested case. This shows that the vector pipeline is not universally faster, and its further implementation, and integration into NBT should be done while considering the trade-off between client side and server side rendering costs.

5.3 Effect of Montage Sizes

The different montages have a less significant effect on the performance profile compared to resolution, especially at higher resolutions. This is expected as the different montage sizes and distributions mostly affect the initial Delaunay Triangulation, and the interpolant construction steps of the interpolation which have computational complexity $O(C\log C)$ and $O(C)$ respectively. Since the impact of channel size on computational complexity is less than the impact of resolution, which is $O(R^2)$ it causes channel size to be less impactful at higher resolutions. The range of the channels is also smaller than the resolution, with the channel count going up to 256 with a complexity of $O(C\log C)\approx 2048$ while the resolution reaches 2000×2000 which has a complexity of $O(R^2)\approx 4000000$. For smaller resolutions and larger channels, channel count seem to have a non-negligible impact on the runtime.

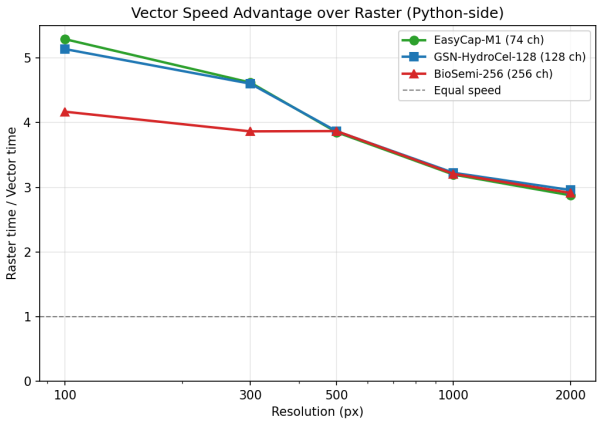


Figure 7: Latency ratios per resolution for different montage sizes

As it can be seen in Figure 7, the larger channels have a lesser ratio which converges as the resolution becomes dominant, with GSN-HydroCel-128 converging with EasyCap-M1 at 300 pixels, and BioSemi-256 converging with the other two at 500 pixels. Since the vector rendering step eliminates the rasterization steps but doesn't change the channel dependent steps, they represent a larger fraction of the computation for smaller resolutions, thus having a lower ratio.

5.4 General Discussion of Results

Overall, the results show a clear improvement in server-side rendering and an extra cost in client-side rendering for the

vectorized pipeline. It can also be seen in Appendix A that the results are unlikely to be from noise as the standard deviations of data are low. For further development of the vector image rendering, and NBT this relationship should also be considered. If topomaps are generated repeatedly per session, using vector rendering will decrease the load on the server, however complex SVGs may create a load too great for older/weaker hardware. The resolution of the rendered topomaps are also a factor as lower resolution renders are shown to be similar in terms of client-side performance and vastly superior in terms of server-side performance. One proposed solution is using a hybrid approach where the majority of the topomaps are generated via vectorization and rasterized for clients where browser rendering cost can be a concern.

The results also identify points to further optimize. Since the raster rendering is removed from the pipeline, the main costs can be found in interpolation, SVG generation and contour generation. These can be further investigated by exploring different algorithms in terms of accuracy and latency. Investigating interpolation is difficult as it has the highest risk of altering and misinterpreting experimental results if implemented poorly. Alternative contour generation algorithms can be more efficient but it too risks misrepresenting results as the interpolation has, as earlier contour implementations had lesser latency but difference in the lines drawn. Investigating different SVG implementations is the least risky, and possibly the highest yield, approach as it only displays data and doesn't generate it, and can yield benefit both server-load and client-load.

Several limitations should also be considered when discussing the results. Firstly, the benchmark uses randomly generated EEG-like data rather than real EEG recordings. This was done to avoid privacy concerns, and create more varied data, but it limited the research of frequency-band grids since they are extremely reliant on input data. Secondly, the client-side results depend on the browser, device, and rendering environment, so they shouldn't be treated as universal results and should be further investigated. Finally, the benchmark measures individual topomap rendering operations rather than full research workflows with many parallel users or large batch requests. The impact of the topomap pipeline in a realistic workflow can and should be tested further after integration into the NBT software.

Despite the stated limitations, the results provided give a clear answer to the main performance investigation. Direct vector rendering substantially reduces server-side latency and moderately reduces server-side memory usage, and preserves the visual output, while creating client-side drawbacks.

6 Conclusions and Future Work

This research investigated the question: **"How do raster based and direct vector/SVG based topomap pipelines compare in computational complexity, latency scaling, and visual/numerical fidelity across montage sizes and frequency-band grids?"**. To answer this question, a Python implementation of NBT was investigated, the existing raster-based topomap pipeline was refactored for benchmarking, a benchmark suite to measure the time of different stages was developed, a new vector-based SVG pipeline was developed, and both of these stages were profiled and compared using the benchmark suite.

The results show that the vector based pipeline vastly outperforms the raster based pipeline during server-side rendering. On the server-side, the vector pipeline is between $2.88\times$ and $5.29\times$ faster than the raster pipeline with the largest improvement occurring in lower resolutions. An improvement in server-side memory usage was also observed to be between $1.24\times$ and $1.42\times$. However, a significant client-side performance impact was also seen as the client-side latency ratio was found to be between $1.03\times$ and $0.50\times$ and the memory usage ratio was found to be between $0.96\times$ and $0.14\times$.

The stage-level analysis for 300×300 resolution, GSN-HydroCel-128 shows that the raster pipeline is dominated by the interpolation and rendering stage with it taking 129.90 ms, while the vector pipeline is mainly dominated by the figure construction and interpolation stages taking 18.72 ms and 13.06 ms respectively.

In terms of visual fidelity and information, the vector pipeline was found to be identical to the raster image thus causing no loss in visual information. SVG graphics being able to scale to different resolutions without the major artifacts seen in rasterized images also makes it an ideal format for web based clients.

There are several ways this research can be extended in the future. First, the data can be gathered from real life EEG examples to test the two implementations using real-life data instead of randomly generated data. Secondly, the client side impact of the raster and vector implementations can be further expanded by including different browsers, hardware, operating systems and stages to understand the main limitations of browser-side vector rendering. Third, different vector and raster algorithms or implementations can be investigated to further optimize stages in both implementations. Finally, usage of the topomaps in NBT by users can be investigated to identify the realistic impact of the implementations.

In conclusion, the research suggests that vector based topomap rendering pipelines can be a promising alternative for raster rendering pipelines for NBT. The lower server-side latency and memory usage, combined with the reduced artifacts makes it well suited for EEG topomaps. However, implementations need to be careful in balancing the client-side downsides of the pipeline with the visual and server-side advantages that it brings.

7 Responsible Research

This research doesn't contain a direct ethical risk related to human subjects or patient data. As although the investigated systems such as: NBT, and the topomap representation are used to analyze confidential medical data, medical data have not been used in this study, instead dummy data such as randomly generated data have been used for the investigation.

The work is partially reproducible as the newly created implementations such as the benchmark and the vector renderer have been thoroughly described in the methodology. The full reproducibility of this study is reliant on the source code as although it is currently closed source, it is planned to be released as open-source software via GPL licensing for scientific use in the future.

Large Language Models (LLMs) were used to assist in development and writing during this research. For development purposes LLMs have been used to generate boilerplate code, find possible bugs in created functionality, and help in understanding already existing code bases. For writing purposes, LLMs have been used to check for grammatical/spelling errors, help in finding literature, and help in creating better explanations.

In each of these cases, the output generated by the LLM was thoroughly reviewed and verified by the author, in order to confirm that the data generated is correct and was not caused by a hallucination. Especially when searching, and citing literature as to not accidentally plagiarize someone's work, and respect intellectual property.

A Recorded Data

A.1 Server Side Results — Totals

eeg	resolution	iterations	render	stage	mean time (ms)	time std (ms)	time median (ms)	time P95 (ms)	memory mean (mb)	memory std (mb)	memory median (mb)	memory P95 (mb)	Time Mean Ratios
GSN-HydroCel-128	100	100	raster	Total	124.34	3.73	123.44	127.41	1.80	0.01	1.80	1.80	5.17
GSN-HydroCel-128	100	100	raster	add_colorbar_trace	0.53	0.03	0.52	0.56	0.01	0.00	0.01	0.01	1.02
GSN-HydroCel-128	100	100	raster	add_electrode_traces	0.49	0.04	0.48	0.52	0.01	0.00	0.01	0.01	1.05
GSN-HydroCel-128	100	100	raster	apply_topomap_layout	2.40	0.05	2.38	2.47	0.12	0.00	0.12	0.12	0.99
GSN-HydroCel-128	100	100	raster	build_topomap_figure	12.33	0.28	12.26	12.70	0.69	0.01	0.69	0.69	0.84
GSN-HydroCel-128	100	100	raster	normalize_channel_coords	3.39	0.09	3.36	3.51	0.22	0.00	0.22	0.22	
GSN-HydroCel-128	100	100	raster	render_mne_topomap	105.22	3.59	104.41	108.06	0.75	0.00	0.75	0.76	17.75
GSN-HydroCel-128	100	100	vector	Total	24.05	0.17	24.03	24.37	1.28	0.00	1.28	1.28	0.19
GSN-HydroCel-128	100	100	vector	add_colorbar_trace	0.52	0.01	0.52	0.53	0.01	0.00	0.01	0.01	0.98
GSN-HydroCel-128	100	100	vector	add_electrode_traces	0.46	0.01	0.46	0.48	0.01	0.00	0.01	0.01	0.95
GSN-HydroCel-128	100	100	vector	apply_topomap_layout	2.42	0.04	2.41	2.45	0.12	0.00	0.12	0.12	1.01
GSN-HydroCel-128	100	100	vector	build_vector_figure	14.73	0.11	14.71	14.91	0.91	0.00	0.91	0.92	1.19
GSN-HydroCel-128	100	100	vector	interpolate_topomap	5.93	0.11	5.91	5.98	0.22	0.00	0.22	0.22	0.06
GSN-HydroCel-128	300	100	raster	Total	161.70	0.50	161.77	162.47	4.67	0.00	4.67	4.67	4.59
GSN-HydroCel-128	300	100	raster	add_colorbar_trace	0.53	0.01	0.53	0.55	0.01	0.00	0.01	0.01	1.02
GSN-HydroCel-128	300	100	raster	add_electrode_traces	0.50	0.01	0.50	0.51	0.01	0.00	0.01	0.01	1.07
GSN-HydroCel-128	300	100	raster	apply_topomap_layout	2.40	0.02	2.40	2.42	0.12	0.00	0.12	0.12	0.99
GSN-HydroCel-128	300	100	raster	build_topomap_figure	25.00	0.13	24.99	25.26	0.90	0.00	0.90	0.90	1.34
GSN-HydroCel-128	300	100	raster	normalize_channel_coords	3.37	0.03	3.37	3.43	0.22	0.00	0.22	0.22	
GSN-HydroCel-128	300	100	raster	render_mne_topomap	129.90	0.42	129.96	130.62	3.41	0.00	3.41	3.41	9.95
GSN-HydroCel-128	300	100	vector	Total	35.20	0.17	35.15	35.53	3.34	0.00	3.34	3.34	0.22
GSN-HydroCel-128	300	100	vector	add_colorbar_trace	0.52	0.01	0.52	0.53	0.01	0.00	0.01	0.01	0.98
GSN-HydroCel-128	300	100	vector	add_electrode_traces	0.47	0.01	0.46	0.49	0.01	0.00	0.01	0.01	0.93
GSN-HydroCel-128	300	100	vector	apply_topomap_layout	2.43	0.05	2.43	2.47	0.12	0.00	0.12	0.12	1.02
GSN-HydroCel-128	300	100	vector	build_vector_figure	18.72	0.14	18.68	18.99	2.98	0.00	2.98	2.98	0.75
GSN-HydroCel-128	300	100	vector	interpolate_topomap	13.06	0.07	13.05	13.16	0.22	0.00	0.22	0.22	0.10
GSN-HydroCel-128	500	100	raster	Total	236.46	1.58	236.29	238.61	10.37	0.01	10.37	10.37	3.86
GSN-HydroCel-128	500	100	raster	add_colorbar_trace	0.54	0.01	0.53	0.55	0.01	0.00	0.01	0.01	1.02
GSN-HydroCel-128	500	100	raster	add_electrode_traces	0.51	0.01	0.51	0.53	0.01	0.00	0.01	0.01	1.09
GSN-HydroCel-128	500	100	raster	apply_topomap_layout	2.41	0.09	2.40	2.45	0.12	0.00	0.12	0.12	1.00
GSN-HydroCel-128	500	100	raster	build_topomap_figure	47.90	0.30	47.88	48.24	1.28	0.01	1.28	1.28	1.57
GSN-HydroCel-128	500	100	raster	normalize_channel_coords	3.38	0.05	3.37	3.48	0.22	0.00	0.22	0.22	
GSN-HydroCel-128	500	100	raster	render_mne_topomap	181.72	1.49	181.54	183.46	8.73	0.00	8.73	8.73	6.63
GSN-HydroCel-128	500	100	vector	Total	61.23	0.62	61.10	61.63	7.64	0.00	7.63	7.64	0.26
GSN-HydroCel-128	500	100	vector	add_colorbar_trace	0.52	0.01	0.52	0.54	0.01	0.00	0.01	0.01	0.98
GSN-HydroCel-128	500	100	vector	add_electrode_traces	0.47	0.01	0.47	0.49	0.01	0.00	0.01	0.01	0.91
GSN-HydroCel-128	500	100	vector	apply_topomap_layout	2.42	0.03	2.41	2.45	0.12	0.00	0.12	0.12	1.00
GSN-HydroCel-128	500	100	vector	build_vector_figure	30.43	0.53	30.33	30.73	7.27	0.00	7.27	7.28	0.64
GSN-HydroCel-128	500	100	vector	interpolate_topomap	27.40	0.14	27.36	27.76	0.22	0.00	0.22	0.22	0.15
GSN-HydroCel-128	1000	100	raster	Total	543.48	3.73	543.41	550.13	35.85	0.00	35.85	35.85	3.22
GSN-HydroCel-128	1000	100	raster	add_colorbar_trace	0.54	0.01	0.53	0.56	0.01	0.00	0.01	0.01	1.01
GSN-HydroCel-128	1000	100	raster	add_electrode_traces	0.51	0.01	0.51	0.52	0.01	0.00	0.01	0.01	1.08
GSN-HydroCel-128	1000	100	raster	apply_topomap_layout	2.41	0.02	2.41	2.46	0.12	0.00	0.12	0.12	0.99
GSN-HydroCel-128	1000	100	raster	build_topomap_figure	132.09	0.66	132.02	133.21	2.20	0.00	2.20	2.20	1.89
GSN-HydroCel-128	1000	100	raster	normalize_channel_coords	3.37	0.04	3.36	3.43	0.22	0.00	0.22	0.22	
GSN-HydroCel-128	1000	100	raster	render_mne_topomap	404.57	3.54	404.37	410.63	33.30	0.00	33.30	33.30	4.25
GSN-HydroCel-128	1000	100	vector	Total	168.70	0.96	168.55	169.95	26.23	0.00	26.23	26.23	0.31
GSN-HydroCel-128	1000	100	vector	add_colorbar_trace	0.53	0.01	0.53	0.55	0.01	0.00	0.01	0.01	0.99
GSN-HydroCel-128	1000	100	vector	add_electrode_traces	0.48	0.02	0.47	0.49	0.01	0.00	0.01	0.01	0.92

GSN-HydroCel-128	1000	100	vector	add_electrode_traces	0.48	0.02	0.47	0.49	0.01	0.00	0.01	0.01	0.92
GSN-HydroCel-128	1000	100	vector	apply_topomap_layout	2.44	0.03	2.44	2.50	0.12	0.00	0.12	0.12	1.01
GSN-HydroCel-128	1000	100	vector	build_vector_figure	69.97	0.35	69.95	70.36	25.87	0.00	25.87	25.87	0.53
GSN-HydroCel-128	1000	100	vector	interpolate_topomap	95.29	0.86	94.83	96.43	0.22	0.00	0.22	0.22	0.24
GSN-HydroCel-128	2000	100	raster	Total	1692.51	8.60	1691.08	1702.44	134.77	0.01	134.77	134.77	2.95
GSN-HydroCel-128	2000	100	raster	add_colorbar_trace	0.54	0.01	0.53	0.55	0.01	0.00	0.01	0.01	1.02
GSN-HydroCel-128	2000	100	raster	add_electrode_traces	0.52	0.01	0.51	0.53	0.01	0.00	0.01	0.01	1.08
GSN-HydroCel-128	2000	100	raster	apply_topomap_layout	2.41	0.09	2.39	2.44	0.12	0.00	0.12	0.12	0.99
GSN-HydroCel-128	2000	100	raster	build_topomap_figure	404.97	3.03	404.84	408.02	3.79	0.01	3.79	3.79	1.94
GSN-HydroCel-128	2000	100	raster	normalize_channel_coords	3.36	0.04	3.36	3.41	0.22	0.00	0.22	0.22	
GSN-HydroCel-128	2000	100	raster	render_mne_topomap	1280.72	7.52	1279.78	1291.05	130.62	0.00	130.62	130.62	3.54
GSN-HydroCel-128	2000	100	vector	Total	574.13	6.60	571.83	584.70	97.82	0.00	97.82	97.82	0.34
GSN-HydroCel-128	2000	100	vector	add_colorbar_trace	0.53	0.01	0.52	0.54	0.01	0.00	0.01	0.01	0.98
GSN-HydroCel-128	2000	100	vector	add_electrode_traces	0.48	0.02	0.47	0.49	0.01	0.00	0.01	0.01	0.92
GSN-HydroCel-128	2000	100	vector	apply_topomap_layout	2.43	0.04	2.42	2.50	0.12	0.00	0.12	0.12	1.01
GSN-HydroCel-128	2000	100	vector	build_vector_figure	208.50	6.26	206.76	219.27	97.46	0.00	97.46	97.46	0.51
GSN-HydroCel-128	2000	100	vector	interpolate_topomap	362.20	1.55	362.24	364.81	0.22	0.00	0.22	0.22	0.28
biosemi256	100	100	raster	Total	137.35	0.37	137.26	137.92	2.06	0.00	2.06	2.06	4.17
biosemi256	100	100	raster	add_colorbar_trace	0.52	0.01	0.52	0.54	0.01	0.00	0.01	0.01	1.01
biosemi256	100	100	raster	add_electrode_traces	0.63	0.01	0.63	0.65	0.02	0.00	0.02	0.02	1.02
biosemi256	100	100	raster	apply_topomap_layout	2.38	0.02	2.37	2.43	0.12	0.00	0.12	0.12	0.99
biosemi256	100	100	raster	build_topomap_figure	12.56	0.12	12.53	12.80	0.70	0.00	0.70	0.70	0.64
biosemi256	100	100	raster	normalize_channel_coords	6.22	0.06	6.21	6.31	0.43	0.00	0.43	0.43	
biosemi256	100	100	raster	render_mne_topomap	115.05	0.36	114.98	115.54	0.78	0.00	0.78	0.78	11.79
biosemi256	100	100	vector	Total	32.95	0.17	32.93	33.21	1.66	0.00	1.66	1.67	0.24
biosemi256	100	100	vector	add_colorbar_trace	0.52	0.01	0.52	0.53	0.01	0.00	0.01	0.01	0.99
biosemi256	100	100	vector	add_electrode_traces	0.62	0.01	0.61	0.63	0.02	0.00	0.02	0.02	0.98
biosemi256	100	100	vector	apply_topomap_layout	2.41	0.02	2.41	2.45	0.12	0.00	0.12	0.12	1.02
biosemi256	100	100	vector	build_vector_figure	19.64	0.17	19.63	19.88	1.08	0.00	1.08	1.08	1.56
biosemi256	100	100	vector	interpolate_topomap	9.76	0.07	9.75	9.89	0.43	0.00	0.43	0.43	0.08
biosemi256	300	100	raster	Total	176.73	0.53	176.62	177.79	5.02	0.01	5.02	5.02	3.86
biosemi256	300	100	raster	add_colorbar_trace	0.53	0.02	0.53	0.55	0.01	0.00	0.01	0.01	1.02
biosemi256	300	100	raster	add_electrode_traces	0.65	0.01	0.65	0.67	0.02	0.00	0.02	0.02	1.05
biosemi256	300	100	raster	apply_topomap_layout	2.40	0.02	2.40	2.44	0.12	0.00	0.12	0.12	0.99
biosemi256	300	100	raster	build_topomap_figure	26.79	0.11	26.77	26.96	0.99	0.01	0.98	0.98	1.07
biosemi256	300	100	raster	normalize_channel_coords	6.26	0.07	6.25	6.37	0.43	0.00	0.43	0.43	
biosemi256	300	100	raster	render_mne_topomap	140.10	0.50	139.96	141.22	3.46	0.00	3.46	3.46	8.20
biosemi256	300	100	vector	Total	45.74	0.19	45.70	46.10	3.86	0.00	3.86	3.86	0.26
biosemi256	300	100	vector	add_colorbar_trace	0.52	0.01	0.52	0.53	0.01	0.00	0.01	0.01	0.98
biosemi256	300	100	vector	add_electrode_traces	0.62	0.01	0.62	0.65	0.02	0.00	0.02	0.02	0.95
biosemi256	300	100	vector	apply_topomap_layout	2.42	0.03	2.42	2.46	0.12	0.00	0.12	0.12	1.01
biosemi256	300	100	vector	build_vector_figure	25.10	0.17	25.06	25.44	3.27	0.00	3.27	3.28	0.94
biosemi256	300	100	vector	interpolate_topomap	17.08	0.09	17.07	17.23	0.43	0.00	0.43	0.43	0.12
biosemi256	500	100	raster	Total	248.33	1.39	248.10	250.17	10.49	0.00	10.49	10.49	3.87
biosemi256	500	100	raster	add_colorbar_trace	0.54	0.01	0.53	0.56	0.01	0.00	0.01	0.01	1.02
biosemi256	500	100	raster	add_electrode_traces	0.67	0.01	0.67	0.68	0.02	0.00	0.02	0.02	1.07
biosemi256	500	100	raster	apply_topomap_layout	2.42	0.03	2.41	2.47	0.12	0.00	0.12	0.12	0.99
biosemi256	500	100	raster	build_topomap_figure	48.01	0.26	47.96	48.44	1.24	0.00	1.24	1.24	1.66
biosemi256	500	100	raster	normalize_channel_coords	6.24	0.05	6.23	6.32	0.43	0.00	0.43	0.43	
biosemi256	500	100	raster	render_mne_topomap	190.46	1.23	190.27	192.22	8.67	0.00	8.67	8.67	6.01

biosemi256	500	100	vector	Total	64.13	0.19	64.12	64.50	7.65	0.00	7.65	7.66	0.26
biosemi256	500	100	vector	add_colorbar_trace	0.53	0.01	0.53	0.54	0.01	0.00	0.01	0.01	0.98
biosemi256	500	100	vector	add_electrode_traces	0.62	0.01	0.62	0.64	0.02	0.00	0.02	0.02	0.93
biosemi256	500	100	vector	apply_topomap_layout	2.44	0.04	2.42	2.54	0.12	0.00	0.12	0.12	1.01
biosemi256	500	100	vector	build_vector_figure	28.85	0.15	28.82	29.08	7.07	0.00	7.07	7.07	0.60
biosemi256	500	100	vector	interpolate_topomap	31.70	0.12	31.69	31.89	0.43	0.00	0.43	0.43	0.17
biosemi256	1000	100	raster	Total	555.92	2.69	555.58	560.50	35.84	0.01	35.84	35.85	3.21
biosemi256	1000	100	raster	add_colorbar_trace	0.54	0.01	0.53	0.55	0.01	0.00	0.01	0.01	1.02
biosemi256	1000	100	raster	add_electrode_traces	0.67	0.01	0.67	0.68	0.02	0.00	0.02	0.02	1.07
biosemi256	1000	100	raster	apply_topomap_layout	2.42	0.02	2.42	2.45	0.12	0.00	0.12	0.12	1.00
biosemi256	1000	100	raster	build_topomap_figure	135.02	0.65	134.95	136.03	2.14	0.01	2.14	2.14	1.91
biosemi256	1000	100	raster	normalize_channel_coords	6.24	0.07	6.23	6.36	0.43	0.00	0.43	0.43	
biosemi256	1000	100	raster	render_mne_topomap	411.04	2.54	410.81	415.58	33.13	0.00	33.13	33.13	4.15
biosemi256	1000	100	vector	Total	173.32	0.73	173.19	174.68	25.98	0.00	25.98	25.98	0.31
biosemi256	1000	100	vector	add_colorbar_trace	0.53	0.01	0.53	0.54	0.01	0.00	0.01	0.01	0.99
biosemi256	1000	100	vector	add_electrode_traces	0.62	0.01	0.62	0.64	0.02	0.00	0.02	0.02	0.93
biosemi256	1000	100	vector	apply_topomap_layout	2.42	0.03	2.41	2.47	0.12	0.00	0.12	0.12	1.00
biosemi256	1000	100	vector	build_vector_figure	70.59	0.60	70.46	71.83	25.39	0.00	25.39	25.40	0.52
biosemi256	1000	100	vector	interpolate_topomap	99.16	0.31	99.10	99.68	0.43	0.00	0.43	0.43	0.24
biosemi256	2000	100	raster	Total	1751.28	6.22	1750.54	1759.99	136.02	0.00	136.02	136.02	2.90
biosemi256	2000	100	raster	add_colorbar_trace	0.53	0.01	0.53	0.54	0.01	0.00	0.01	0.01	1.01
biosemi256	2000	100	raster	add_electrode_traces	0.67	0.01	0.67	0.69	0.02	0.00	0.02	0.02	1.07
biosemi256	2000	100	raster	apply_topomap_layout	2.41	0.04	2.40	2.46	0.12	0.00	0.12	0.12	0.99
biosemi256	2000	100	raster	build_topomap_figure	427.55	2.50	427.80	431.25	4.60	0.00	4.60	4.60	1.83
biosemi256	2000	100	raster	normalize_channel_coords	6.26	0.09	6.24	6.38	0.43	0.00	0.43	0.43	
biosemi256	2000	100	raster	render_mne_topomap	1313.87	5.65	1313.14	1323.86	130.84	0.00	130.84	130.84	3.59
biosemi256	2000	100	vector	Total	603.72	6.19	601.08	614.10	98.80	0.00	98.80	98.81	0.34
biosemi256	2000	100	vector	add_colorbar_trace	0.53	0.01	0.53	0.55	0.01	0.00	0.01	0.01	0.99
biosemi256	2000	100	vector	add_electrode_traces	0.62	0.01	0.62	0.64	0.02	0.00	0.02	0.02	0.94
biosemi256	2000	100	vector	apply_topomap_layout	2.43	0.07	2.42	2.46	0.12	0.00	0.12	0.12	1.01
biosemi256	2000	100	vector	build_vector_figure	233.89	5.78	231.09	242.36	98.22	0.00	98.22	98.22	0.55
biosemi256	2000	100	vector	interpolate_topomap	366.25	1.35	366.11	368.87	0.43	0.00	0.43	0.43	0.28
easycap-M1	100	100	raster	Total	120.38	5.76	118.30	133.02	1.72	0.00	1.72	1.72	5.37
easycap-M1	100	100	raster	add_colorbar_trace	0.54	0.03	0.53	0.58	0.01	0.00	0.01	0.01	1.04
easycap-M1	100	100	raster	add_electrode_traces	0.44	0.04	0.42	0.50	0.01	0.00	0.01	0.01	1.10
easycap-M1	100	100	raster	apply_topomap_layout	2.43	0.07	2.40	2.56	0.12	0.00	0.12	0.12	1.00
easycap-M1	100	100	raster	build_topomap_figure	12.54	0.39	12.42	13.35	0.69	0.00	0.69	0.69	0.85
easycap-M1	100	100	raster	normalize_channel_coords	2.26	0.11	2.23	2.41	0.13	0.00	0.13	0.13	
easycap-M1	100	100	raster	render_mne_topomap	102.19	5.32	100.27	113.81	0.76	0.00	0.76	0.76	23.40
easycap-M1	100	100	vector	Total	22.40	0.19	22.36	22.72	1.21	0.00	1.21	1.21	0.19
easycap-M1	100	100	vector	add_colorbar_trace	0.52	0.01	0.52	0.54	0.01	0.00	0.01	0.01	0.96
easycap-M1	100	100	vector	add_electrode_traces	0.40	0.01	0.40	0.43	0.01	0.00	0.01	0.01	0.91
easycap-M1	100	100	vector	apply_topomap_layout	2.42	0.03	2.41	2.47	0.12	0.00	0.12	0.12	1.00
easycap-M1	100	100	vector	build_vector_figure	14.70	0.15	14.66	14.88	0.94	0.00	0.94	0.94	1.17
easycap-M1	100	100	vector	interpolate_topomap	4.37	0.08	4.36	4.45	0.13	0.00	0.13	0.13	0.04
easycap-M1	300	100	raster	Total	157.73	0.83	157.79	159.19	4.61	0.01	4.61	4.61	4.61
easycap-M1	300	100	raster	add_colorbar_trace	0.53	0.01	0.53	0.55	0.01	0.00	0.01	0.01	1.02
easycap-M1	300	100	raster	add_electrode_traces	0.44	0.02	0.44	0.46	0.01	0.00	0.01	0.01	1.08
easycap-M1	300	100	raster	apply_topomap_layout	2.41	0.04	2.41	2.48	0.12	0.00	0.12	0.12	0.99
easycap-M1	300	100	raster	build_topomap_figure	24.99	0.15	24.96	25.20	0.91	0.01	0.91	0.91	1.31

easycap-M1	300	100	raster	normalize_channel_coords	2.22	0.03	2.22	2.28	0.13	0.00	0.13	0.13	
easycap-M1	300	100	raster	render_mne_topomap	127.14	0.80	127.22	128.53	3.43	0.00	3.43	3.44	10.80
easycap-M1	300	100	vector	Total	34.19	0.21	34.16	34.52	3.34	0.00	3.34	3.34	0.22
easycap-M1	300	100	vector	add_colorbar_trace	0.52	0.01	0.52	0.54	0.01	0.00	0.01	0.01	0.98
easycap-M1	300	100	vector	add_electrode_traces	0.40	0.01	0.40	0.42	0.01	0.00	0.01	0.01	0.92
easycap-M1	300	100	vector	apply_topomap_layout	2.43	0.03	2.43	2.48	0.12	0.00	0.12	0.12	1.01
easycap-M1	300	100	vector	build_vector_figure	19.06	0.15	19.04	19.33	3.07	0.00	3.07	3.08	0.76
easycap-M1	300	100	vector	interpolate_topomap	11.77	0.12	11.76	11.90	0.13	0.00	0.13	0.13	0.09
easycap-M1	500	100	raster	Total	231.06	1.58	230.96	233.52	10.21	0.00	10.21	10.21	3.83
easycap-M1	500	100	raster	add_colorbar_trace	0.53	0.01	0.53	0.55	0.01	0.00	0.01	0.01	1.02
easycap-M1	500	100	raster	add_electrode_traces	0.45	0.01	0.45	0.46	0.01	0.00	0.01	0.01	1.11
easycap-M1	500	100	raster	apply_topomap_layout	2.42	0.07	2.41	2.45	0.12	0.00	0.12	0.12	0.99
easycap-M1	500	100	raster	build_topomap_figure	46.19	0.31	46.15	46.49	1.21	0.00	1.21	1.21	1.51
easycap-M1	500	100	raster	normalize_channel_coords	2.23	0.03	2.22	2.27	0.13	0.00	0.13	0.13	
easycap-M1	500	100	raster	render_mne_topomap	179.25	1.54	179.18	181.54	8.74	0.00	8.74	8.74	6.78
easycap-M1	500	100	vector	Total	60.31	1.36	59.96	62.22	7.52	0.00	7.52	7.53	0.26
easycap-M1	500	100	vector	add_colorbar_trace	0.53	0.01	0.52	0.54	0.01	0.00	0.01	0.01	0.99
easycap-M1	500	100	vector	add_electrode_traces	0.40	0.01	0.40	0.42	0.01	0.00	0.01	0.01	0.90
easycap-M1	500	100	vector	apply_topomap_layout	2.44	0.04	2.43	2.50	0.12	0.00	0.12	0.12	1.01
easycap-M1	500	100	vector	build_vector_figure	30.49	1.33	30.14	32.45	7.26	0.00	7.26	7.26	0.66
easycap-M1	500	100	vector	interpolate_topomap	26.45	0.14	26.42	26.57	0.13	0.00	0.13	0.13	0.15
easycap-M1	1000	100	raster	Total	527.76	2.80	527.41	533.03	35.32	0.01	35.31	35.32	3.20
easycap-M1	1000	100	raster	add_colorbar_trace	0.54	0.01	0.54	0.56	0.01	0.00	0.01	0.01	1.02
easycap-M1	1000	100	raster	add_electrode_traces	0.45	0.01	0.45	0.48	0.01	0.00	0.01	0.01	1.11
easycap-M1	1000	100	raster	apply_topomap_layout	2.41	0.03	2.41	2.47	0.12	0.00	0.12	0.12	0.99
easycap-M1	1000	100	raster	build_topomap_figure	122.69	0.85	122.68	124.41	1.89	0.01	1.88	1.88	1.84
easycap-M1	1000	100	raster	normalize_channel_coords	2.22	0.02	2.21	2.25	0.13	0.00	0.13	0.13	
easycap-M1	1000	100	raster	render_mne_topomap	399.46	2.57	399.11	404.05	33.17	0.00	33.17	33.17	4.20
easycap-M1	1000	100	vector	Total	165.10	0.76	165.03	166.24	25.68	0.00	25.68	25.69	0.31
easycap-M1	1000	100	vector	add_colorbar_trace	0.53	0.01	0.52	0.54	0.01	0.00	0.01	0.01	0.98
easycap-M1	1000	100	vector	add_electrode_traces	0.41	0.01	0.41	0.43	0.01	0.00	0.01	0.01	0.90
easycap-M1	1000	100	vector	apply_topomap_layout	2.43	0.04	2.42	2.51	0.12	0.00	0.12	0.12	1.01
easycap-M1	1000	100	vector	build_vector_figure	66.54	0.60	66.45	67.71	25.41	0.00	25.41	25.42	0.54
easycap-M1	1000	100	vector	interpolate_topomap	95.20	0.30	95.14	95.62	0.13	0.00	0.13	0.13	0.24
easycap-M1	2000	100	raster	Total	1672.10	6.53	1670.46	1683.50	134.00	0.00	134.00	134.00	2.87
easycap-M1	2000	100	raster	add_colorbar_trace	0.53	0.01	0.53	0.55	0.01	0.00	0.01	0.01	1.00
easycap-M1	2000	100	raster	add_electrode_traces	0.45	0.01	0.45	0.46	0.01	0.00	0.01	0.01	1.10
easycap-M1	2000	100	raster	apply_topomap_layout	2.41	0.03	2.41	2.45	0.12	0.00	0.12	0.12	0.99
easycap-M1	2000	100	raster	build_topomap_figure	383.12	2.42	383.26	386.64	3.17	0.00	3.17	3.17	1.83
easycap-M1	2000	100	raster	normalize_channel_coords	2.23	0.05	2.22	2.27	0.13	0.00	0.13	0.13	
easycap-M1	2000	100	raster	render_mne_topomap	1283.36	5.94	1281.59	1291.18	130.56	0.00	130.56	130.56	3.47
easycap-M1	2000	100	vector	Total	583.14	7.27	580.88	593.07	97.54	0.00	97.54	97.54	0.35
easycap-M1	2000	100	vector	add_colorbar_trace	0.53	0.01	0.53	0.55	0.01	0.00	0.01	0.01	1.00
easycap-M1	2000	100	vector	add_electrode_traces	0.41	0.01	0.41	0.43	0.01	0.00	0.01	0.01	0.91
easycap-M1	2000	100	vector	apply_topomap_layout	2.44	0.02	2.44	2.48	0.12	0.00	0.12	0.12	1.01
easycap-M1	2000	100	vector	build_vector_figure	209.46	7.12	207.07	217.90	97.27	0.00	97.27	97.27	0.55
easycap-M1	2000	100	vector	interpolate_topomap	370.30	2.04	369.99	373.56	0.13	0.00	0.13	0.13	0.29

A.2 Server Side Results — Per Stage

eeg	resolution	iterations	Raster time mean (ms)	Vector time mean (ms)	Raster time std (ms)	Vector time std(ms)	Raster time median (ms)	Vector time median (ms)	Raster time P95 (ms)	Vector time P95 (ms)
Easycap-M1_100	100	100	120.383	22.397	5.755	0.194	118.296	22.36	133.016	22.717
Easycap-M1_300	300	100	157.734	34.188	0.826	0.205	157.787	34.156	159.191	34.523
Easycap-M1_500	500	100	231.062	60.31	1.575	1.357	230.96	59.956	233.518	62.22
easycap-M1_1000	1000	100	527.762	165.099	2.801	0.755	527.411	165.03	533.026	166.244
Easycap-M1_2000	2000	100	1672.102	583.143	6.53	7.274	1670.456	580.877	1683.499	593.072
GSN-HydroCel-128_100	100	100	124.343	24.048	3.726	0.167	123.437	24.028	127.41	24.374
GSN-HydroCel-128_300	300	100	161.703	35.196	0.501	0.168	161.774	35.153	162.466	35.53
GSN-HydroCel-128_500	500	100	236.462	61.23	1.576	0.618	236.293	61.101	238.613	61.629
GSN-HydroCel-128_1000	1000	100	543.484	168.704	3.726	0.962	543.407	168.545	550.127	169.954
GSN-HydroCel-128_2000	2000	100	1692.508	574.126	8.599	6.595	1691.082	571.826	1702.435	584.699
biosemi256_100	100	100	137.351	32.95	0.368	0.17	137.257	32.925	137.916	33.21
biosemi256_300	300	100	176.727	45.737	0.531	0.194	176.62	45.701	177.792	46.097
biosemi256_500	500	100	248.326	64.128	1.388	0.19	248.096	64.12	250.168	64.496
biosemi256_1000	1000	100	555.917	173.32	2.694	0.726	555.578	173.188	560.499	174.675
biosemi256_2000	2000	100	1751.282	603.719	6.215	6.185	1750.536	601.075	1759.986	614.101

Raster Memory mean (mb)	Vector Memory mean (mb)	Raster Memory std (mb)	Vector Memory std (mb)	Raster Memory median (mb)	Vector Memory median (mb)	Raster Memory P95 (mb)	Vector Memory P95 (mb)	Time Speedup Raster Over Vector	Memory Ratio Raster Over Vector
1.717	1.209	0.003	0.002	1.716	1.208	1.721	1.212	5.29	1.42
4.607	3.34	0.014	0.002	4.605	3.34	4.609	3.344	4.62	1.38
10.208	7.524	0.001	0.002	10.208	7.524	10.211	7.528	3.85	1.36
35.316	25.682	0.014	0.002	35.314	25.682	35.318	25.685	3.20	1.38
133.995	97.538	0.001	0.002	133.995	97.538	133.998	97.541	2.88	1.37
1.802	1.276	0.014	0.002	1.8	1.275	1.803	1.279	5.14	1.41
4.666	3.34	0.002	0.002	4.666	3.339	4.669	3.343	4.60	1.40
10.371	7.635	0.014	0.003	10.369	7.634	10.373	7.64	3.87	1.36
35.85	26.227	0.002	0.002	35.85	26.227	35.853	26.232	3.22	1.37
134.772	97.819	0.014	0.002	134.771	97.819	134.773	97.822	2.96	1.38
2.06	1.66	0.002	0.003	2.06	1.66	2.063	1.665	4.17	1.24
5.021	3.858	0.014	0.003	5.019	3.857	5.023	3.863	3.86	1.30
10.489	7.653	0.002	0.002	10.489	7.653	10.492	7.657	3.87	1.37
35.844	25.976	0.014	0.003	35.843	25.976	35.845	25.981	3.21	1.38
136.017	98.8	0.002	0.002	136.017	98.8	136.021	98.805	2.91	1.38

A.3 Client Side Results

eeg	iterations	resolution	render	metric	mean	std	median	p95
easycap-M1	100	100	raster	DOMContentLoaded	639.56	2.19	639.65	642.81
easycap-M1	100	100	raster	JS heap	15.67	0.02	15.67	15.7
easycap-M1	100	100	raster	drag zoom	224.86	6.45	223.95	233.11
easycap-M1	100	100	raster	first Plotly ready	652.4	2.08	652.4	655.4
easycap-M1	100	100	raster	load	640.39	2.19	640.5	643.6
easycap-M1	100	100	raster	relayout zoom	8.38	0.37	8.3	9.1
easycap-M1	100	100	vector	DOMContentLoaded	658.18	3.02	657.95	663.81
easycap-M1	100	100	vector	JS heap	17.32	0.02	17.32	17.35
easycap-M1	100	100	vector	drag zoom	234.84	4.82	233.6	244.52
easycap-M1	100	100	vector	first Plotly ready	670.64	3.09	670.5	676.91
easycap-M1	100	100	vector	load	659.03	3.03	658.8	664.61
easycap-M1	100	100	vector	relayout zoom	20.87	0.31	20.8	21.5
easycap-M1	100	300	raster	DOMContentLoaded	647.07	2.48	646.85	651.41
easycap-M1	100	300	raster	JS heap	16.82	0.02	16.81	16.84
easycap-M1	100	300	raster	drag zoom	224.63	4.89	222.95	234.81
easycap-M1	100	300	raster	first Plotly ready	659.78	2.53	659.75	663.71
easycap-M1	100	300	raster	load	647.94	2.49	647.7	652.31
easycap-M1	100	300	raster	relayout zoom	10.97	1	10.7	13.81
easycap-M1	100	300	vector	DOMContentLoaded	683.03	2.65	682.9	686.81
easycap-M1	100	300	vector	JS heap	23.63	0.28	23.74	23.84
easycap-M1	100	300	vector	drag zoom	249.7	4.56	249.65	258.7
easycap-M1	100	300	vector	first Plotly ready	695.97	2.77	695.7	700.21
easycap-M1	100	300	vector	load	683.94	2.66	683.8	687.8
easycap-M1	100	300	vector	relayout zoom	32.59	0.55	32.5	33.7
easycap-M1	100	500	raster	DOMContentLoaded	659.86	3.16	659.4	664.32
easycap-M1	100	500	raster	JS heap	18.4	0.02	18.4	18.43
easycap-M1	100	500	raster	drag zoom	228.61	6.01	226.45	237.4
easycap-M1	100	500	raster	first Plotly ready	672.71	3.2	672.25	677.21
easycap-M1	100	500	raster	load	660.78	3.17	660.4	665.21
easycap-M1	100	500	raster	relayout zoom	12.34	0.45	12.3	13
easycap-M1	100	500	vector	DOMContentLoaded	722	2.5	722.05	725.71
easycap-M1	100	500	vector	JS heap	35.84	0.34	35.89	36.23
easycap-M1	100	500	vector	drag zoom	269.61	3.85	269.25	277.44
easycap-M1	100	500	vector	first Plotly ready	734.7	2.63	735.1	738.71
easycap-M1	100	500	vector	load	723.17	2.51	723.25	726.91
easycap-M1	100	500	vector	relayout zoom	63.86	0.92	63.9	65.01
easycap-M1	100	1000	raster	DOMContentLoaded	688.8	2.61	688.6	693.82
easycap-M1	100	1000	raster	JS heap	22.03	0.02	22.03	22.06
easycap-M1	100	1000	raster	drag zoom	233.79	5.42	231.85	243.31
easycap-M1	100	1000	raster	first Plotly ready	702.43	2.64	702.05	707.81
easycap-M1	100	1000	raster	load	689.91	2.61	689.7	695.02
easycap-M1	100	1000	raster	relayout zoom	19.02	1.27	18.8	20.11
easycap-M1	100	1000	vector	DOMContentLoaded	881.86	3.19	882.05	886.32
easycap-M1	100	1000	vector	JS heap	66.84	0.35	66.66	67.62
easycap-M1	100	1000	vector	drag zoom	339.1	3.94	339.25	345.53
easycap-M1	100	1000	vector	first Plotly ready	894.9	3.27	894.95	899.4
easycap-M1	100	1000	vector	load	883.91	3.19	884.2	888.42
easycap-M1	100	1000	vector	relayout zoom	110.74	1.35	110.65	112.9
easycap-M1	100	2000	raster	DOMContentLoaded	758.15	2.35	758.35	761.82

easycap-M1	100	2000	raster	JS heap	28.91	0.02	28.92	28.94
easycap-M1	100	2000	raster	drag zoom	254.21	9.82	252	270.5
easycap-M1	100	2000	raster	first Plotly ready	772.68	2.56	772.75	777.2
easycap-M1	100	2000	raster	load	759.41	2.4	759.55	763.11
easycap-M1	100	2000	raster	relayout zoom	30.93	0.72	30.8	31.9
easycap-M1	100	2000	vector	DOMContentLoaded	1495.97	5.25	1495.75	1505.74
easycap-M1	100	2000	vector	JS heap	208.36	0.13	208.32	208.56
easycap-M1	100	2000	vector	drag zoom	519.68	17.15	513.5	555.43
easycap-M1	100	2000	vector	first Plotly ready	1510.47	5.35	1510.4	1519.66
easycap-M1	100	2000	vector	load	1502.7	5.29	1502.3	1512.15
easycap-M1	100	2000	vector	relayout zoom	335.7	4.16	334.9	342.92
GSN-HydroCel-128	100	100	raster	DOMContentLoaded	647.58	9.56	645	666.09
GSN-HydroCel-128	100	100	raster	JS heap	16.05	0.03	16.05	16.09
GSN-HydroCel-128	100	100	raster	drag zoom	226.07	3.9	225.85	233.71
GSN-HydroCel-128	100	100	raster	first Plotly ready	660.79	9.62	658.55	679.27
GSN-HydroCel-128	100	100	raster	load	648.43	9.59	645.9	666.98
GSN-HydroCel-128	100	100	raster	relayout zoom	8.83	0.45	8.7	9.62
GSN-HydroCel-128	100	100	vector	DOMContentLoaded	665.08	9.89	663.3	670.87
GSN-HydroCel-128	100	100	vector	JS heap	17.18	0.02	17.18	17.21
GSN-HydroCel-128	100	100	vector	drag zoom	240.69	7.72	240.7	245.11
GSN-HydroCel-128	100	100	vector	first Plotly ready	677.89	9.99	676.3	683.5
GSN-HydroCel-128	100	100	vector	load	665.93	9.91	664.25	671.68
GSN-HydroCel-128	100	100	vector	relayout zoom	21.35	4.73	20.8	22.02
GSN-HydroCel-128	100	300	raster	DOMContentLoaded	649.79	2.93	649.5	654.7
GSN-HydroCel-128	100	300	raster	JS heap	17.05	0.03	17.05	17.09
GSN-HydroCel-128	100	300	raster	drag zoom	230.51	4.84	231.9	236
GSN-HydroCel-128	100	300	raster	first Plotly ready	662.46	2.91	662.1	667.51
GSN-HydroCel-128	100	300	raster	load	650.7	2.95	650.35	655.61
GSN-HydroCel-128	100	300	raster	relayout zoom	11.1	1.18	10.7	13.91
GSN-HydroCel-128	100	300	vector	DOMContentLoaded	683.44	3.48	683	688.62
GSN-HydroCel-128	100	300	vector	JS heap	21.83	0.21	21.93	22.15
GSN-HydroCel-128	100	300	vector	drag zoom	252.77	4.07	252.8	260
GSN-HydroCel-128	100	300	vector	first Plotly ready	697.37	12.38	695.95	702.32
GSN-HydroCel-128	100	300	vector	load	684.37	3.47	684	689.52
GSN-HydroCel-128	100	300	vector	relayout zoom	29.52	0.72	29.4	30.7
GSN-HydroCel-128	100	500	raster	DOMContentLoaded	663.57	2.95	663.4	667.61
GSN-HydroCel-128	100	500	raster	JS heap	19.08	0.03	19.08	19.12
GSN-HydroCel-128	100	500	raster	drag zoom	231.88	3.41	231.95	237.41
GSN-HydroCel-128	100	500	raster	first Plotly ready	676.99	3.1	676.95	681.2
GSN-HydroCel-128	100	500	raster	load	664.49	2.96	664.35	668.61
GSN-HydroCel-128	100	500	raster	relayout zoom	12.49	0.42	12.4	13.3
GSN-HydroCel-128	100	500	vector	DOMContentLoaded	727.45	3.47	726.95	732.46
GSN-HydroCel-128	100	500	vector	JS heap	30.64	0.15	30.61	30.85
GSN-HydroCel-128	100	500	vector	drag zoom	274.29	4.14	274.85	280.61
GSN-HydroCel-128	100	500	vector	first Plotly ready	740.23	3.56	739.55	745.39
GSN-HydroCel-128	100	500	vector	load	728.62	3.48	728.1	733.58
GSN-HydroCel-128	100	500	vector	relayout zoom	66.37	1.89	66.85	68.3
GSN-HydroCel-128	100	1000	raster	DOMContentLoaded	708.27	19.69	704	719.28
GSN-HydroCel-128	100	1000	raster	JS heap	23.98	0.03	23.98	24.03
GSN-HydroCel-128	100	1000	raster	drag zoom	240.01	3.45	239.9	245.03

GSN-HydroCel-128	100	1000	raster	first Plotly ready	723.87	22.96	718.85	736.43
GSN-HydroCel-128	100	1000	raster	load	709.44	19.67	705.25	720.39
GSN-HydroCel-128	100	1000	raster	relayout zoom	21.61	1.02	21.4	22.7
GSN-HydroCel-128	100	1000	vector	DOMContentLoaded	894.67	3.41	894.6	899.32
GSN-HydroCel-128	100	1000	vector	JS heap	73.37	0.38	73.26	74.14
GSN-HydroCel-128	100	1000	vector	drag zoom	352.99	4.78	353.9	359.81
GSN-HydroCel-128	100	1000	vector	first Plotly ready	908.05	3.5	908.05	913.41
GSN-HydroCel-128	100	1000	vector	load	896.85	3.44	896.9	901.62
GSN-HydroCel-128	100	1000	vector	relayout zoom	121.62	1.27	121.3	123.7
GSN-HydroCel-128	100	2000	raster	DOMContentLoaded	787.63	2.59	787.5	791.92
GSN-HydroCel-128	100	2000	raster	JS heap	35.22	0.02	35.22	35.25
GSN-HydroCel-128	100	2000	raster	drag zoom	253.47	10.1	252.45	270.32
GSN-HydroCel-128	100	2000	raster	first Plotly ready	802.56	2.82	802.55	807.71
GSN-HydroCel-128	100	2000	raster	load	788.96	2.57	788.9	793.31
GSN-HydroCel-128	100	2000	raster	relayout zoom	32.42	0.42	32.4	33.21
GSN-HydroCel-128	100	2000	vector	DOMContentLoaded	1512.38	6.16	1512.2	1521.84
GSN-HydroCel-128	100	2000	vector	JS heap	211.35	0.24	211.25	212.12
GSN-HydroCel-128	100	2000	vector	drag zoom	515.02	12.36	512.6	548.31
GSN-HydroCel-128	100	2000	vector	first Plotly ready	1527.3	6.37	1526.85	1536.71
GSN-HydroCel-128	100	2000	vector	load	1519.31	6.27	1519.3	1528.43
GSN-HydroCel-128	100	2000	vector	relayout zoom	337.39	3.48	336.8	343.37
biosemi256	100	100	raster	DOMContentLoaded	645	2.76	645.05	649.82
biosemi256	100	100	raster	JS heap	17.14	0.03	17.14	17.19
biosemi256	100	100	raster	drag zoom	225.41	4.62	224.05	235.31
biosemi256	100	100	raster	first Plotly ready	658.05	2.65	657.85	662.5
biosemi256	100	100	raster	load	645.84	2.76	645.85	650.71
biosemi256	100	100	raster	relayout zoom	10.19	0.44	10.1	10.71
biosemi256	100	100	vector	DOMContentLoaded	672.2	2.72	672.4	676.2
biosemi256	100	100	vector	JS heap	17.83	0.02	17.83	17.85
biosemi256	100	100	vector	drag zoom	247.68	3.42	247.35	254.04
biosemi256	100	100	vector	first Plotly ready	684.94	2.7	685.1	689.01
biosemi256	100	100	vector	load	673.04	2.72	673.3	677.01
biosemi256	100	100	vector	relayout zoom	29.24	0.72	29.3	30.52
biosemi256	100	300	raster	DOMContentLoaded	655.4	4.73	655.3	659.21
biosemi256	100	300	raster	JS heap	18.48	0.03	18.48	18.52
biosemi256	100	300	raster	drag zoom	230.5	14.44	227	239.03
biosemi256	100	300	raster	first Plotly ready	668.78	4.94	668.5	672.61
biosemi256	100	300	raster	load	656.32	4.74	656.2	660.11
biosemi256	100	300	raster	relayout zoom	12.03	0.35	12	12.61
biosemi256	100	300	vector	DOMContentLoaded	700.09	2.75	699.9	704.12
biosemi256	100	300	vector	JS heap	24.76	1.68	26.09	26.23
biosemi256	100	300	vector	drag zoom	262.03	5.56	263.8	269.02
biosemi256	100	300	vector	first Plotly ready	713.12	2.81	713.2	717.21
biosemi256	100	300	vector	load	701.05	2.76	700.85	705.21
biosemi256	100	300	vector	relayout zoom	52.96	6.84	48.45	63.81
biosemi256	100	500	raster	DOMContentLoaded	672.4	18.34	665.8	710.69
biosemi256	100	500	raster	JS heap	19.86	0.03	19.85	19.9
biosemi256	100	500	raster	drag zoom	231.01	6.81	229.5	239.73
biosemi256	100	500	raster	first Plotly ready	685.46	18.42	679	724.96
biosemi256	100	500	raster	load	673.35	18.37	666.75	711.69

biosemi256	100	500	raster	relayout zoom	14.25	2.56	13.9	15.2
biosemi256	100	500	vector	DOMContentLoaded	726.97	2.86	727	731.44
biosemi256	100	500	vector	JS heap	30.35	2.11	29.51	35.62
biosemi256	100	500	vector	drag zoom	275.07	3.45	275	280.7
biosemi256	100	500	vector	first Plotly ready	739.93	2.81	739.95	744.22
biosemi256	100	500	vector	load	728.14	2.86	728.1	732.44
biosemi256	100	500	vector	relayout zoom	65.45	0.99	65.6	66.51
biosemi256	100	1000	raster	DOMContentLoaded	704.73	2.92	704.8	710.41
biosemi256	100	1000	raster	JS heap	24.65	0.03	24.64	24.69
biosemi256	100	1000	raster	drag zoom	239.98	3.57	240.3	245.53
biosemi256	100	1000	raster	first Plotly ready	718.75	3.03	718.85	724.81
biosemi256	100	1000	raster	load	705.91	2.96	705.9	711.7
biosemi256	100	1000	raster	relayout zoom	20.63	0.41	20.6	21.3
biosemi256	100	1000	vector	DOMContentLoaded	891.4	3.8	891.1	897.94
biosemi256	100	1000	vector	JS heap	68.67	0.68	68.76	69.31
biosemi256	100	1000	vector	drag zoom	347.76	4.93	347.25	356.5
biosemi256	100	1000	vector	first Plotly ready	904.71	3.87	904.25	911.44
biosemi256	100	1000	vector	load	893.52	3.83	893.1	900.42
biosemi256	100	1000	vector	relayout zoom	118.87	1.65	118.6	122.41
biosemi256	100	2000	raster	DOMContentLoaded	813.53	2.66	813.2	818.21
biosemi256	100	2000	raster	JS heap	33.68	0.02	33.68	33.72
biosemi256	100	2000	raster	drag zoom	264.73	12.24	260.4	280.83
biosemi256	100	2000	raster	first Plotly ready	829.42	11.96	827.95	834.02
biosemi256	100	2000	raster	load	814.97	2.64	814.65	819.71
biosemi256	100	2000	raster	relayout zoom	39.51	0.49	39.4	40.51
biosemi256	100	2000	vector	DOMContentLoaded	1532.06	6.13	1530.9	1542.21
biosemi256	100	2000	vector	JS heap	220.34	0.04	220.34	220.42
biosemi256	100	2000	vector	drag zoom	532.97	8.96	531.55	547.87
biosemi256	100	2000	vector	first Plotly ready	1547.24	6.33	1546.5	1557.14
biosemi256	100	2000	vector	load	1539.16	6.28	1538.1	1549.05
biosemi256	100	2000	vector	relayout zoom	357.11	2.97	356.6	362.47

References

[1] Turky Alotaiby, Fathi E. Abd El-Samie, Saleh A. Alshebeili, and Ishtiaq Ahmad. A review of channel selection algorithms for eeg signal processing. *EURASIP J. Adv. Signal Process.*, 2015.

[2] Jeffrey K. Aronson and Robin E. Ferner. Biomarkers—a general review. *Current Protocols in Pharmacology*, 76(1):9.23.1–9.23.17, 2017.

[3] Anusha Baskaran, Roumen Milev, and Roger S. McIntyre. The neurobiology of the eeg biomarker as a predictor of treatment response in depression. *Neuropharmacology*, 63(4):507–513, 2012.

[4] R. W. Clough and J. L. Tocher. Finite element stiffness matrices for analysis of plate bending. In *Proceedings of Conference on Matrix Methods in Structural Analysis*, pages 515–546, 1965.

[5] Boris Delaunay. Sur la sphère vide. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et naturelles*, 6:793–800, 1934.

[6] Arnaud Delorme and Scott Makeig. Eeglab: an open source toolbox for analysis of single-trial eeg dynamics including independent component analysis. *Journal of Neuroscience Methods*, 134(1):9–21, 2004.

[7] Stephen V. Gliske, Zachary T. Irwin, Cynthia Chestek, and William C. Stacey. Effect of sampling rate and filter settings on high frequency oscillation detections. *Clinical Neurophysiology*, 127(9):3042–3050, September 2016.

[8] Alexandre Gramfort, Martin Luessi, Eric Larson, Denis A. Engemann, Daniel Strohmeier, Christian Brodbeck, Roman Goj, Mainak Jas, Teon Brooks, Lauri Parkkonen, and Matti Hämäläinen. Meg and eeg data analysis with mne-python. *Frontiers in Neuroscience*, 7, 2013.

[9] R. Hardstone et al. Detrended fluctuation analysis: a scale-free view on neuronal oscillations. *Frontiers in Physiology*, 3:450, 2012.

[10] John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, and Kurt Akeley. *Computer Graphics: Principles and Practice*. Addison-Wesley, 3 edition, 2013.

[11] John D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[12] Md Kafiul Islam, Amir Rastegarnia, and Zhi Yang. Methods for artifact detection and removal from scalp eeg: A review. *Neurophysiologie Clinique/Clinical Neurophysiology*, 46(4):287–305, 2016.

[13] George H. Klem, Hans Lüders, Herbert H. Jasper, and Christian Erich Elger. The ten-twenty electrode system of the international federation. the international federation of clinical neurophysiology. *Electroencephalography and clinical neurophysiology. Supplement*, 52:3–6, 1999.

[14] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, page 163–169, New York, NY, USA, 1987. Association for Computing Machinery.

[15] Microsoft. Playwright: Fast and reliable end-to-end testing for modern web apps. https://playwright.dev, 2020. Accessed: 2026-06-21.

[16] MNE-Python Developers. mne.viz.plot_topomap. https://mne.tools/stable/generated/mne.viz.plot_topomap.html, 2026. Accessed: 2026-06-02.

[17] Faisal Mushtaq et al. One hundred years of EEG for brain and behaviour research. *Nature Human Behaviour*, 8(8):1437–1443, 2024.

[18] Sebastian Nagel. *Towards a home-use BCI: fast asynchronous control and robust non-control state detection*. PhD thesis, Tübingen, Universität Tübingen, 2019.

[19] NBT-Analytics. NBTpublic: The Neurophysiological Biomarker Toolbox. https://github.com/NBT-Analytics/NBTpublic. Accessed: 2026-06-02.

[20] Neurophysiological Biomarker Toolbox. The Neurophysiological Biomarker Toolbox. https://www.nitrc.org/projects/nbt/. Accessed: 2026-06-02.

[21] Gregory M. Nielson. A method for interpolating scattered data based upon a minimum norm network. *Mathematics of Computation*, 40(161):253–271, 1983.

[22] Paul L. Nunez and Ramesh Srinivasan. *Electric Fields of the Brain: The Neurophysics of EEG*. Oxford University Press, 2006.

[23] Robert Oostenveld and Peter Praamstra. The five percent electrode system for high-resolution eeg and erp measurements. *Clinical Neurophysiology*, 112(4):713–719, 2001.

[24] Maria E. Peltola, Markus Leitinger, Jonathan J. Halford, Kollencheri Puthenveettil Vinayan, Katsuhiro Kobayashi, Ronit M. Pressler, Ioana Mindruta, Luis Carlos Mayor, Leena Lauronen, and Sándor Beniczky. Routine and sleep eeg: Minimum recording standards of the international federation of clinical neurophysiology and the international league against epilepsy. *Clinical Neurophysiology*, 147:108–120, 2023.

[25] F. Perrin, J. Pernier, O. Bertrand, and J.F. Echallier. Spherical splines for scalp potential and current density mapping. *Electroencephalography and Clinical Neurophysiology*, 72(2):184–187, 1989.

[26] Plotly Technologies Inc. Collaborative data science. https://plotly.com, 2015. Montreal, QC. Accessed: 2026-06-21.

[27] R. J. Renka and A. K. Cline. A triangle-based C1 interpolation method. *Rocky Mountain Journal of Mathematics*, 14(1):223–237, 1984.

[28] Pauli Virtanen et al. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17:261–272, 2020.

[29] World Wide Web Consortium. Scalable vector graphics (svg) 2. https://www.w3.org/TR/SVG2/, 2018. W3C Candidate Recommendation. Accessed: 2026-06-02.

[30] Joseph Zoulikian. Week 3: Computer-aided design CAD. https://fabacademy.org/2019/labs/berytech/students/joseph-zoulikian/week_3.html, 2019. Fab Academy 2019. Accessed: 2026-06-21.