Boostlet.js: Medical image processing plugins for the web via JavaScript injection

Edward Gaibor*
University of Massachusetts Boston

Shruti Varade†
University of Massachusetts Boston

Rohini Deshmukh‡
University of Massachusetts Boston

Tim Meyer§
University of the Bundeswehr Munich

Mahsa Geshvadi¶
University of Massachusetts Boston

SangHyuk Kim‖
University of Massachusetts Boston

Vidhya Sree Narayanappa**
University of Massachusetts Boston

Daniel Haehn††
University of Massachusetts Boston

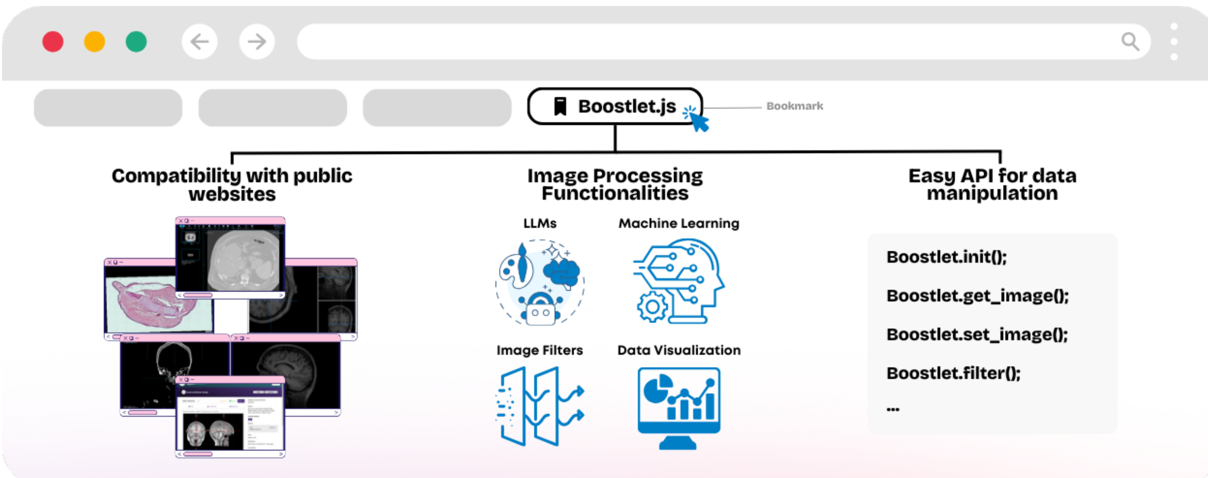

Figure 1: The Boostlet.js library provides a suite of image processing plugins for the web that integrates within various available visualization frameworks using JavaScript injection. This library enables direct access to image data via a simple and easy-to-use API and can be deployed as bookmarklets. Examples, code, and data are available at https://boostlet.org.

Abstract

Can web-based image processing and visualization tools easily integrate into existing websites without significant time and effort? Our Boostlet.js library addresses this challenge by providing an open-source, JavaScript-based web framework to enable additional image processing functionalities. Boostlet examples include kernel filtering, image captioning, data visualization, segmentation, and web-optimized machine-learning models. To achieve this, Boostlet.js uses a browser bookmark to inject a user-friendly plugin selection tool called *PowerBoost* into any host website. Boostlet also provides on-site access to a standard API independent of any visualization framework for pixel data and scene manipulation. Web-based Boostlets provide a modular architecture and client-side processing capabilities to apply advanced image-processing techniques using consumer-level hardware. The code is open-source and available.

*e-mail: edward.gaibor001@umb.edu
†e-mail: s.varade001@umb.edu
‡e-mail: r.deshmukh001@umb.edu
§e-mail: tim.meyer@unibw.de
¶e-mail: mahsa.geshvadi001@umb.edu
‖e-mail: sanghyuk.kim001@umb.edu
**e-mail: v.narayanappa001@umb.edu
††e-mail: daniel.haehn@umb.edu

Index Terms: Web-based image processing, JavaScript injection, Biomedical

1 Introduction

Medical image processing is a common approach to visualizing and comprehending medical data. Different web-based software frameworks exist to visualize the images. However, these frameworks are often limited in processing capabilities and focus mainly on pure visualization tasks. In addition, adding processing algorithms to these frameworks requires targeted software development with a specific programming interface.

To address this, we present Boostlet.js, a tool designed to expand existing web-based visualization frameworks with image processing capabilities (Tab. 1). REMOVE THIS SENTENCE Boostlet offers a variety of pre-built example functionalities like large language models (LLMs), image filters, and other more advanced algorithms.

To address this, we present Boostlet.js, a tool that contributes:

- An unified API that allows the development and integration of image-processing plugins to bridge existing visualization frameworks such as Cornerstone2D [18], NiiVue [2], OpenSeaDragon [1], Xtk [4], and Papaya [13]. With a default fallback in case no framework is detected.
- Boostlet simplifies the development of computational algorithms, making it easy for members of the Medical Imaging Community to create their image processing plugins and share them with others on the client side.

- We offer an open-source implementation and comprehensive documentation to facilitate adoption and further development by the visualization community. Boostlet offers a variety of pre-built example functionalities like large language models (LLMs), image filters, and other more advanced algorithms as reference.

REMOVE THIS PARAGRAPHBoostlet also simplifies the development of computational algorithms, making it easy for members of the Medical Imaging Community to create their image processing plugins and share them with others. Boostlet.js integrates seamlessly with popular web-based visualization frameworks such as Cornerstone2D [18], NiiVue [2], OpenSeaDragon [1], Xtk [4], and Papaya [13].

The user only needs to drag and drop the *PowerBoost* as bookmark into the web bookmarks bar. *PowerBoost* is a user-friendly plugin selection tool that once clicked on a compatible website, a menu with all available functionalities will be displayed through JavaScript injection allowing the user to inject any functionality needed.

	Boostlet	Static Frameworks
Cross-compatibility?	✓	✗
Easy to share?	✓	✗
Quick development?	✓	✗

Table 1: Boostlet allows for easy cross-framework compatibility, sharing, and prototyping with its simple API, while current state-of-the-art frameworks are static, and the development of new functionalities takes time to propagate amongst other popular frameworks.

GET RID OF THE TABLE ENTIRELY WE CAN KEEP THE CAPTION AS TEXT, WHICH COMMUNICATES THE SAME THING

Example usage: a user wants to segment brain-related images on a public data repository like OpenNeuro.org [8] that visualizes data with NiiVue. Without Boostlet, the user would have to download the data, import it into another compatible tool or model, obtain the segmentation output, and then focus on a single region of interest (ROI). This process would have to be repeated each time the user wants to segment an image. With Boostlet, the user can click on the *PowerBoost* bookmark while visiting OpenNeuro.org, select the segmentation tool, draw the region of interest, and immediately view the segmentation overlay without downloading. The exact same process would work on another popular data repository, the Imaging Data Commons from the National Cancer Institute [15].

We built Boostlet.js as a unified programming interface to extend existing web-based visualization libraries with image-processing capabilities. In this paper, we describe Boostlet's underlying software architecture and software engineering infrastructure, including automated integration testing and documentation.

We then present the *PowerBoost* user interface that simplifies the access of existing processing modules and the development of new functionalities. Finally, we collect and discuss feedback from experts who develop web-based medical image processing algorithms and test the Boostlet library This might have to change with a new structure. All our developments and experiments are available as open science at: https://github.com/mpsych/boostlet.

2 Related Work

Many frameworks for web-based visualization of biomedical images exist. They all represent significant contributions to the visualization community, and Boostlet.js can help improve the functionalities they offer.

WE CAN GET RID OF THIS PARAGRAPH Cornerstone2D.js [18] is acclaimed for its Digital Imaging and Communications in Medicine (DICOM) image-rendering abilities and customizability, making it a staple in clinical settings. NiiVue.js [2] leverages WebGL's power for neuroimaging, providing high-performance interactive visualization through direct access to the graphical processing unit in the browser. OpenSeadragon [1] excels in displaying high-resolution images, essential for visualizing microscope data. Papaya.js [13] is a tool dedicated to DICOM and Neuroimaging Informatics Technology Initiative (Nifti) image rendering, providing visualization overlay capabilities and a suite of image control options, similar to the first medical visualization toolkit XTK [4]. All frameworks offer limited in-built image processing features, a gap now being bridged by Boostlet.js, which extends their functionalities into advanced image processing that can be developed once and used with all of them.

WE CAN GET RID OF THIS PARAGRAPH

2.1 DICOM and Neuroimaging Visualization Tools

Cornerstone2D.js [18] is acclaimed for its Digital Imaging and Communications in Medicine (DICOM) image-rendering abilities and customizability, making it a staple in clinical settings. **Papaya.js** [13] is dedicated to DICOM and Neuroimaging Informatics Technology Initiative (Nifti) image rendering, providing visualization overlay capabilities and a suite of image control options. **NiiVue.js** [2] leverages WebGL's power for neuroimaging, providing high-performance interactive visualization through direct access to the graphical processing unit in the browser.

Gap: While these tools excel in visualization, they offer in-built image processing features that are non-cross-compatible with each other. Boostlet.js addresses this by extending its functionalities with advanced image processing capabilities that can be easily integrated into them simultaneously through our API.

2.2 High-Resolution Image Visualization Tools

OpenSeadragon [1] excels in displaying high-resolution images, essential for visualizing microscope data.

Gap: Although OpenSeadragon provides excellent visualization capabilities, it lacks advanced image processing functions without installing extra plugins. Since the user can't install plugins on a public website, Boostlet.js fills this gap by providing a range of processing plugins that add-on functionalities like ROI selection, filters, and more.

2.3 Machine Learning and Advanced Processing Tools

Brainchop.org [9] represents a significant advancement in processing large-scale neuroimaging data using machine learning, offering advanced neural network models applied to brain imaging datasets on the client side.

Gap: Brainchop.org focuses on machine learning models for neuroimaging but does not provide a unified framework for integrating various processing tools across different web-based platforms. Boostlet.js complements this by providing an accessible framework that facilitates the application of such machine-learning techniques throughout multiple websites. This enables researchers to perform complex analyses with improved efficiency and ease without high-end equipment. Together, Boostlet.js and platforms like Brainchop.org enhance neuroimaging capabilities and ensure compatibility across different frameworks.

2.4 3D Volume Visualization Tools

XTK [4] is a general-purpose toolkit for 3D visualization in the browser.

Gap: While XTK is versatile for 3D visualization, Boostlet.js compliments this framework by providing unified capabilities compatible with all other supported frameworks.

WE CAN GET RID OF THIS PARAGRAPH Most related to Boostlets is Brainchop.org [9] represents Most closely related to Boostlet.js is Brainchop.org [9] which represents a significant advancement in processing large-scale neuroimaging data using machine learning, offering advanced neural network models applied to brain imaging datasets on the client side. Boostlet.js complements this by providing an accessible framework that facilitates the application of such machine-learning techniques throughout multiple websites. This enables researchers to perform complex analyses with improved efficiency and ease without high-end equipment. Together, Boostlet.js and platforms like Brainchop.org enhance neuroimaging capabilities and ensure compatibility across different frameworks.

INTRODUCE HERE VIS/HCI papers about medical analysis

3 Boostlet.js Framework Overview and Examples

The Boostlet.js library was is designed to provide user-friendly installation (as simple as adding a bookmark) for image processing that is independent of a visualization framework. The library enabled enables the development of processing modules as plugins. These plugins are called *Boostlets*. CLASSIFY THE PLUGINS USING LISTS These plugins can access default functionalities such as pixel data access, filtering, segmenting, and data display with a unified API. This way, developers can wrap their algorithm once and support a variety of visualization toolkits. We provide a range of Boostlets as examples.

Instead of existing approaches, any new techniques as a contribution? One is Meta's Segment Anything algorithm [5], which allows the user to segment an area of interest in data displayed by any supported visualization frameworks, as seen in Fig. 2. For instance, a researcher can search for a relevant case on OpenNeuro.org and segment the corpus callosum by clicking on the Segment Anything Boostlet and dragging a bounding box. The Boostlet will detect NiiVue.js as the underlying visualization framework and query the pixel data. The Boostlet then generates the embedding for the data and computes the segmentation fully on the client side. While Meta provides an ONNX.js compatible predictor module, we also provide the encoding component of Segment Anything in ONNX.js [10]. Edge computing with ONNX.js or Tensorflow.js [16] to execute machine learning algorithms allows a wide range of applications. But we also provide examples that use traditional processing and information visualization, such as generating histograms with Plotly.js, Sobel filter-based edge detection [14], image captioning powered by Huggingface [7], fiber track decompression using the Trako package [3], and a melanoma predictor [6]. Any processing is possible! Get rid of the last sentence, instead "These examples cover a wide variety of image processing venues to develop image processing and visualization."

Replace this figure with the workflow image showing the process

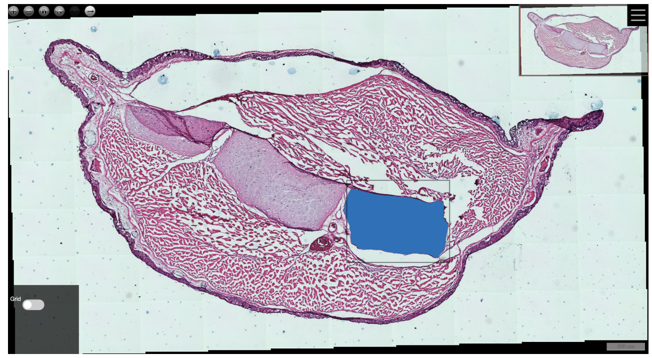

Figure 2: The Segment Anything boostlet executed in OpenSeaDragon.js allows segmenting a region of interest within a slice of microscopy data showing an axolotl limb.

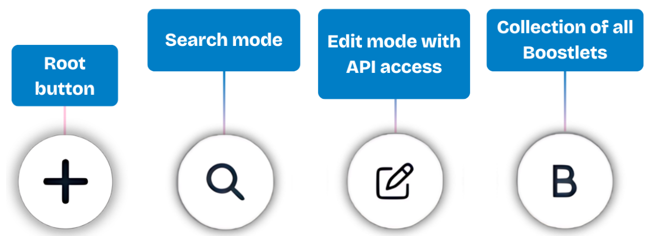

Figure 3: Instead of only icons, we need to make this a screenshot of the powerboost injected into the host website, then we could add pointers to what everything means. PowerBoost is a versatile plugin that equips any frontend environment with a range of Boostlets and a code editor for API access. Easily activated by dragging and dropping the bookmark from our main website into the bookmarks bar. Upon activation, PowerBoost injects code into the host website, revealing a floating icon. Clicking this icon unveils a user interface that expands into sections for Search, Edit, and Boostlets.

```
import {Util} from './util.js';
import {Framework} from './framework.js';

export class Boostlet {

  constructor() { }

  init(name, instance) { }

  async select_box(callback) { }

  async select_seed(howmany) { }

  async load_script(url, callback) { }

  async send_http_post(url, data, callback) { }

  get_image(from_canvas) { }

  set_image(new_pixels) { }

  set_mask(new_mask) { }

  convert_to_png(uint8array, width, height) { }

  filter(pixels, width, height, kernel) { }

  hint(message, duration) { }
```

4 User Interface and Experience

The interface features a search bar for quick Boostlet recommendations and a dropdown menu for browsing various categories like Data Visualization, Filters, LLMs, and Machine Learning, with specific examples such as Plotly.js and Sobel. Additionally, the live code editor mode offers a comprehensive panel for executing various functions, including framework detection and HTTP POST requests, enabling users to create and test Boostlets in real time. Thus, accessibility is prioritized Get rid of this last sentence.

5 Development and Integration

To enhance accessibility for advanced image manipulation, we offer user-friendly functions that integrate various frameworks discussed in Sec. 2.

```
}
```

Listing 1: The Boostlet superclass has a modular design to allow easy extension and integration with various web-based visualization frameworks.

Move code to the supplemental material or remove it, instead we can explain from high level

Listing 1 shows examples of utility functions from the superclass *Boostlet*. These include *load_script()* for external script execution, *send_http_post()* for HTTP requests to process API calls. There are also data transformation functions such as *convert_to_png()*, *grayscale_to_rgba()*, *rgba_to_grayscale()*, and *harden_mask()* to enable image manipulation and prepare image data for the final output.

The *Boostlet* superclass also offers functions that allow user interaction during Boostlet execution. For example, *select_box()* lets the user select a rectangular region of interest (ROI) for manual segmentation or region-based analysis. If a framework does not support a specific mode of interaction (such as OpenSeaDragon.js), the integrated BoxCraft.js [17] library implements custom widgets. Another interactive function is *set_mask()*, which applies a mask to the image or canvas, which can be used to highlight, extract, or contrast different types of pixel data.

5.1 Framework Integration

Boostlets require a compatible visualization framework to access pixel data and modify the scene, or they will use a Canvas fallback.

Boostlet.js interacts directly with the individual API of the framework, leveraging each native capabilities like ROI selection. Due to the object-oriented structure of Boostlet, each new framework is treated as a subclass of the superclass Framework displayed in Listing 1. GET RID of this last sentence, breaks the flow of the idea.

If the host website does not use a Boostlet-compatible framework, the user can still access the **2D canvas fallback mechanism** for baseline image processing and, in the near future, a **WebGL hook to handle buffers** (Sec. 7.2). This approach ensures that the core functionalities of Boostlets remain operational, providing a consistent user experience across various web environments. CanvasFallback's *get_canvas()* function dynamically identifies the largest canvas element on the page but can be forced with user selection. The canvas element is the working area for the image processing tasks. The CanvasFallback dynamically identifies the largest canvas element on the page but can be forced with user selection. The canvas element is the working area where the image data gets rendered.

5.2 Development of Functionalities

To develop functionalities like segmentation and filtering, users can utilize the convenience functions provided by our API to interact with the image data from the host website.

The high-level view of this process includes:

1. **Initialization**: The Boostlet library automatically detects the active visualization framework on the host website or triggers the fallback.
2. **Image Data Retrieval**: Using the *get_image()* function, users can fetch pixel data from the current image displayed on the website.
3. **Image Processing**: The *filter()* function, as an example, enables users to apply pixel transformations, such as convolution with a specified kernel.
4. **Displaying Results**: The *set_image()* function updates the canvas with the processed image data, reflecting the applied transformations in real-time.

This modular approach ensures that image processing tasks can be executed efficiently on the client side without additional development overhead or leaving their browsers.

```
script = document.createElement("script");
script.type = "text/javascript";
script.src = "https://boostlet.org/dist/boostlet.
    min.js";

script.onload = run;
document.head.appendChild(script);
eval(script);

function run() {

  // detect visualization framework
  Boostlet.init();

  image = Boostlet.get_image();

  kernel = [
    -1, 0, 1,
    -2, 0, 2,
    -1, 0, 1
  ];

  filtered = Boostlet.filter(image.data, image.
    width, image.height, kernel);

  Boostlet.set_image( filtered );

}
```

Listing 2: Sobel filter example that uses some convenience functions to add custom functionality with a kernel for pixel manipulation.

GET RID OF THIS CODE LISTING - Feedback suggests supplemental or deletion.

Listing 2 shows a use-case scenario for the superclass Boostlet described in Listing 1 and works as an example of developing image filtering functionalities. This example begins with the initialization process using the *init()* method to detect active visualization frameworks automatically.

After framework detection, Boostlet employs the *get_image()* function to fetch pixel data from the current image displayed on the canvas. Later, it uses the *filter()* function to execute the pixel transformation with the kernel. Finally, *set_image()* is used to reflect the pixel manipulation on the canvas. Overall, the core interaction with data relies on *get_image()* and *set_image()* functions since these allow users to retrieve the current image displayed by the visualization framework and apply processed data back to the canvas.

6 Rename to: Impementation and Testing Buildsystem and Automated Testing

For Boostlet.js developers, we use a build system managed with Parcel, a fast, zero-configuration web app bundler [11] to compile the project into minified Javascript. The command `npx parcel build` compiles the project into minified Javascript and bundles all assets into a single file suitable for production Remove this sentence. In addition, we use node package manager (npm) scripts to manage submodules like the integrated Boxcraft.js [17], live development server initiation, and testing scripts get rid of this, npm is standard so doesn't add value I think.

The Boostlet library includes a testing environment for local development, which facilitates collaboration between members of the visualization community. The testing process for Boostlet.js is designed in a multi-step pipeline (Fig. 4) and focuses on validating

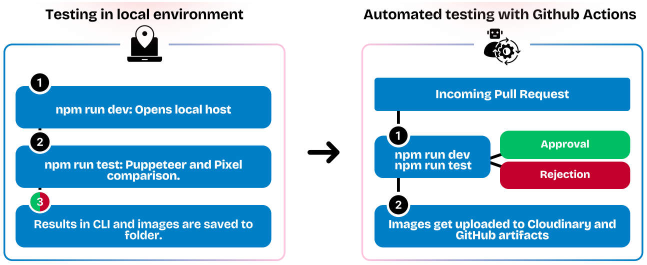

Figure 4: Pipeline for automated local and online testing.

the integrity of the image processing functionalities within the used framework during the development process. Testing is performed both locally on the developer's machine and automatically after submitting a pull request on GitHub. The aim is to ensure interoperability and continuous integration of newly added features within the existing codebase.

In the first step, a local server application runs the automated tests, which include the execution of several boostlets and screen captures after each run. In the next step, the produced screenshot is evaluated by Puppeteer [12] to compare pixel-wise against predefined ground truth images. Any discrepancy larger than 5% of the total number of pixels results in a failed test, indicating a possible issue or bug. Any new code is only merged when all tests pass. GET RID OF THIS LAST PARAGRAPH DESCRIBING THE PIPELINE, based on feedback not too relevant.

7 Discussion and Future Work

7.1 Community Survey

We asked a group of medical imaging and visualization professionals of different institutions to provide feedback and evaluation by completing an 8-question survey (Tab. 2). We received three anonymous responses; the results are in Fig. 5.

No.	Question
1	I found Boostlet.js easy to integrate with existing web-based frameworks.
2	The process of adding Boostlet.js to my browser as a bookmark is straightforward.
3	The PowerBoost user interface is easy to navigate.
4	The client-side processing capabilities of Boostlet.js are satisfactory.
5	The API for pixel data manipulation provided by Boostlet.js is user-friendly.
6	Boostlet.js makes it easy to share or develop image processing plugins with the medical imaging community.
7	I am likely to recommend Boostlet.js to other members of the medical imaging community.
8	Please provide any additional comments or suggestions for improving Boostlet.

Table 2: List of questions given to an audience of three medical imaging and visualization professionals regarding their experience with Boostlet.

According to the survey results, Boostlet has a strong user experience and functionality foundation, as seen in responses to questions 1 through 7 (Tab. 2, Fig. 5). However, for question 8, respondents suggested that Boostlet could improve in making its processes more intuitive, such as allowing the drawing of multiple bounding boxes without refreshing and displaying helpful messages about

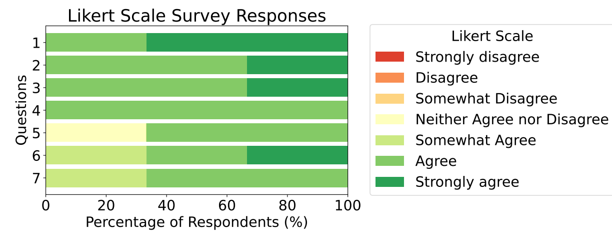

Figure 5: Results of the Likert scale survey show that all users either stay neutral, agree, or strongly agree with the questions prompted in Tab. 2. *Results for question 8 are not plotted since it is a short-answer feedback question.

ongoing processes. Another suggestion was to expand the model library to include a wider range of medical imaging tasks, which is now the current objective. Sharing Boostlet.js with the medical imaging and visualization community could help to achieve this goal.

7.2 Limitations and Future Work

Boostlet only supports a 2D canvas fallback when the framework is not detected as compatible Sec. 5.1. However, some frameworks use WebGL or WebGPU. Thus, current efforts focus on a WebGL/WebGPU fallback mode that would allow pixel manipulation. Early experiments confirm that JavaScript injected code can wrap around the WebGL context to monitor all rendering commands sent to the graphical processing unit (GPU). These then can be used to perform pixel data extraction from the framebuffers before data gets rendered. These pixels can then be processed using JavaScript code or in a fragment shader as part of a Boostlet plugin.

7.3 Security Concerns

One important aspect to consider is the potential for Cross-Site Scripting (XSS) attacks if the host website does not follow best practices for request headers CITE OWASP BEST PRACTICES. This risk is inherent to any JavaScript injection process and is not unique to Boostlet.js. To mitigate this, it is crucial that host websites implement robust security measures, such as setting appropriate Content Security Policy (CSP) headers, to prevent malicious exploits.

8 Conclusion

Boostlet.js enhances web-based visualization frameworks with medical image processing capabilities by injecting JavaScript code. This allows to development of a processing plugin (Boostlet) that then works with a range of supported visualization frameworks without any further tailoring. Boostlets are installable as bookmarklets or available within the bundled PowerBoost user interface.

Acknowledgments

The authors wish to thank Jenna Kim, Dhruv Shah, Adnan Ali, Sai Bharadwaj, Likhith Charugundla, Gopi Krishna, Neha Rapolu, Roshan Vemula and our anonymous evaluators for additional testing and improvements. This work was partly supported by grants from the Alfred P. Sloan Foundation and Oracle for undergraduate research.

References

[1] An open-source, web-based viewer for zoomable images, implemented in pure JavaScript. https://github.com/openseadragon/openseadragon. 1, 2

[2] WebGL2 based medical image viewer. Supports over 30 formats of volumes and meshes. https://github.com/niivue/niivue. 1, 2

[3] D. Haehn, L. Franke, F. Zhang, S. C. Karayumak, S. Pieper, L. O'Donnell, and Y. Rathi. Trako: Efficient transmission of tractography data for visualization. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer, Cham, 2020. 3

[4] D. Haehn, N. Rannou, B. Ahtam, P. E. Grant, and R. Pienaar. Neuroimaging in the browser using the x toolkit. *Frontiers in Neuroinformatics*, 2012. 1, 2

[5] Y. Huang, X. Yang, L. Liu, H. Zhou, A. Chang, X. Zhou, R. Chen, J. Yu, J. Chen, C. Chen, et al. Segment anything model for medical images? *Medical Image Analysis*, p. 103061, 2023. 3

[6] S. Kim, E. Gaibor, and D. Haehn. Web-based melanoma detection, 2024. 3

[7] J. Li, D. Li, C. Xiong, and S. Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation, 2022. 3

[8] C. J. Markiewicz, K. J. Gorgolewski, F. Feingold, R. Blair, Y. O. Halchenko, E. Miller, N. Hardcastle, J. Wexler, O. Esteban, M. Goncalves, et al. Openneuro: An open resource for sharing of neuroimaging data. *BioRxiv*, pp. 2021–06, 2021. 2

[9] M. Masoud, P. Reddy, F. Hu, and S. Plis. Brainchop: Next generation web-based neuroimaging application, 2023. 2, 3

[10] Microsoft. Microsoft/onnxjs: Onnx.js: Run onnx models using javascript. `https://github.com/microsoft/onnxjs`, 2018. 3

[11] Parcel-Bundler. Parcel-bundler/parcel: The zero configuration build tool for the web. `https://github.com/parcel-bundler/parcel`, 2018. 4

[12] Puppeteer. Puppeteer/puppeteer: Node.js api for chrome. `https://github.com/puppeteer/puppeteer`, 2018. 5

[13] Rii-Mango. Rii-mango/papaya: A pure javascript medical research image viewer. `https://github.com/rii-mango/Papaya`, 2012. 1, 2

[14] C. Rorden, R. Newman-Norlund, C. Drake, D. R. Glen, J. Fridriksson, T. Hanayik, and P. A. Taylor. Improving 3d edge detection for visual inspection of mri coregistration and alignment. *bioRxiv*, pp. 2022–09, 2022. 3

[15] D. P. Schacherer, M. D. Herrmann, D. A. Clunie, H. Höfener, W. Clifford, W. J. Longabaugh, S. Pieper, R. Kikinis, A. Fedorov, and A. Homeyer. The nci imaging data commons as a platform for reproducible research in computational pathology. *Computer Methods and Programs in Biomedicine*, 242:107839, Dec. 2023. doi: 10.1016/j.cmpb.2023.107839 2

[16] D. Smilkov, N. Thorat, Y. Assogba, A. Yuan, N. Kreeger, P. Yu, K. Zhang, S. Cai, E. Nielsen, D. Soergel, S. Bileschi, M. Terry, C. Nicholson, S. N. Gupta, S. Sirajuddin, D. Sculley, R. Monga, G. Corrado, F. B. Viégas, and M. Wattenberg. Tensorflow.js: Machine learning for the web and beyond, 2019. 3

[17] S. Varade. Shrutivarade/boxcraft: Roi selection widget. `https://github.com/shrutivarade/BoxCraft`, 2023. 4

[18] E. Ziegler, T. Urban, D. Brown, J. Petts, S. D. Pieper, R. Lewis, C. Hafey, and G. J. Harris. Open Health Imaging Foundation Viewer: An Extensible Open-Source Framework for Building Web-Based Imaging Applications to Support Cancer Research. doi: 10.1200/CCI.19.00131 1, 2