Behind Density Lines: An Interface to Manually Segment Scanning Electron Microscopy Images

Team 1: Alex Lin, Anthony Nguyen, Luke DiGiovanna, John Siegel
Clients: Dr. Monet Roberts, Dr. Linyin Chen, Tanner Upthegrove

CS4624: Multimedia, Hypertext, and Information Access
Instructed by Professor Mohamed Farag
Virginia Tech, Blacksburg, VA 24061

November 30, 2023
Contents

1 Abstract ........................................ 4

2 Introduction .................................. 5
   2.1 Problem .................................. 5
   2.2 Motivation ............................... 5
   2.3 General Approach ....................... 6

3 Requirements ................................. 9

4 Implementation .............................. 11
   4.1 System Design ............................ 11
   4.2 Frameworks and Libraries .............. 11
   4.3 Architecture ............................. 12
      4.3.1 Initial Implementation ............ 12
      4.3.2 Multiplayer Overhaul .............. 14
   4.4 Frontend ................................. 15
      4.4.1 Segment Splitting .................. 15
      4.4.2 Server-Side Rendering ............. 15
   4.5 Backend .................................. 15
      4.5.1 Real-time Sockets .................. 15
      4.5.2 Microservices ....................... 17

5 Assessment .................................... 18

6 User’s Manual ............................... 21
   6.1 Home Page .............................. 21
   6.2 Segmenter ............................... 21
   6.3 Score ................................... 24
   6.4 Export to GIF ........................... 24
### 7 Developer’s Manual

7.1 Installation .................................................. 26
7.2 Repository Structure ......................................... 27
  7.2.1 Painter ................................................... 28
  7.2.2 Scoring Microservice .................................... 30
  7.2.3 Export to GIF Microservice ............................... 30

### 8 Lessons Learned

8.1 Timeline ..................................................... 32
8.2 Problems and Future Work ................................ 33
  8.2.1 Scalability and Performance ............................ 34
  8.2.2 Automated Segmentation ................................. 34
  8.2.3 Researcher Features ..................................... 37

### 9 Acknowledgements

38

### 10 References

39
List of Figures

1  Basic Painter User Interface ............................................. 7
2  High Level System Design .................................................. 11
3  Front End Component Architecture ...................................... 12
4  Initial Home Page ........................................................... 13
5  Room State Receiving Draw Events for Each Image .................. 14
6  Socket Connecting Client to a Room ...................................... 16
7  GIF Encoder Draws Frames on Draw Events ............................ 17
8  The Team and Clients at the Virginia Tech Science Festival ....... 18
9  Website Home Page .......................................................... 21
10 Example of Drawing Segments ............................................. 22
11 Drawing Toolbar with Brush Size Slider and Eraser .................. 23
12 Example of Segmenting a SEM Image with the Toolbar .............. 23
13 Toolbar Displaying Similarity Score ..................................... 24
14 Toolbar Displaying Export Options ....................................... 25
15 Apps file hierarchy .......................................................... 27
16 Packages for abstracted components .................................... 28
17 Next Application File Structure ........................................... 29
18 Scoring Microservice File Structure ..................................... 30
19 Export to GIF Microservice File Structure .............................. 31
20 Initial Timeline Containing Broad Goals ................................. 32
21 Timeline Containing Technical Milestones ............................. 33
22 Initial System Architecture with Automated Segmenting .......... 35
23 Meta’s SAM Segmentation on a sample SEM image .................. 36
Abstract

Scanning Electron Microscopy (SEM) is a powerful imaging technique that plays a crucial role in various scientific fields, including materials science, biology, and nanotechnology. SEM allows researchers to obtain high-resolution images of the surface morphology and topography of specimens, offering a detailed view of structures at the nanoscale. SEM images reveal intricate surface features, enabling scientists to study the texture, shape, and size of particles, cells, or materials with remarkable precision. This technique has revolutionized our understanding of the micro and nanoworld, contributing to advancements in fields ranging from semiconductor technology to biological research, ultimately expanding the boundaries of human knowledge.

Currently, researchers often perform manual segmentation of SEM images as a critical step in their analysis process. This manual segmentation involves meticulously outlining and labeling regions of interest within the images, such as specific structures or particles. Researchers typically utilize specialized software tools designed for image processing and analysis, where they trace object boundaries or delineate areas based on visual cues.

Due to the time and effort it takes to manually segment the SEM images, we implemented a gameified multiplayer web application for individual contributors to manually segment an SEM image in real time. A significant purpose of this was to engage the future generation of scientists and researchers with a showcase at the Virginia Tech Science Festival in November 2023.

To provide instant feedback for the participants, we implemented a comparison scoring algorithm for a given segmentation to a reference segmentation for a particular SEM image. This allowed individuals and groups to assess their performance and provide a game element.

From this project, we gained a holistic understanding of creating a full-stack project. From the backend, we learned how to use Amazon Web Services, like EC2, to make our website’s infrastructure scalable. By utilizing Javascript frameworks and libraries like NextJS, Socket.io, and ThreeJS, we built a user interface that is pleasant to view and use for collaborative manual segmentation.
2 Introduction

2.1 Problem

Currently, manual segmentation of SEM images is an important stage in the analysis process for researchers. Manual segmentation entails painstakingly drawing and naming sections of interest within photographs, such as specific structures or particles. Researchers often use sophisticated image processing and analysis software tools to trace object boundaries or designate areas based on visual clues. This time-consuming and subjective process is dependent on the individual’s skill and interpretation of the image data. Despite its limitations, manual segmentation is still a popular method, particularly when dealing with complicated or irregularly shaped structures, because it provides a high level of control and accuracy. However, as the Internet has connected researchers across the world, live collaboration can prove integral towards faster segmentation and community engagement.

2.2 Motivation

The motivation for the project is to reduce the tediousness of manual SEM image segmentation for researchers while also integrating community interaction for scholars and the general public. In academia, researchers have to manually segment thousands of images throughout their time researching different specimens. Allowing for live collaboration can reduce overall times for image segmentation. All the time saved from manually segmenting these images alone can now be redirected towards more insightful activities regarding their own research. Live collaboration also invites for a deeper understanding through discussion.

Public outreach is also important for this project as it aims to achieve two objectives. The first objective is to potentially crowd-source completed segmentations for other researchers. By having researchers attempt their own segmentation, they can provide references for others to improve their own segmentation skills. The community interaction that stems from our applications aims to be a catalyst for innovation in the bioengineering field. The second objective is to garner interest from the general public to Virginia Tech’s bioengineering department. This project is poised to be displayed at Virginia Tech’s Science Festival in November of this year. A successful showcase can inspire young students to enroll in Virginia Tech’s bioengineering program and become potential researchers.
2.3 General Approach

For our project, our first task was to create an interface for the user to manually segment an image. During development, we tried various types of images (large, small, black and white, etc.) as the SEM images sourced from our client are simply PNG files. From our understanding of the task, recreating the manually segmenting process on a website’s frontend led us to utilizing ThreeJS to create a canvas that functioned as a coloring tool. We also broke down our into two parts: drawing tools, and image segmentation. Images are essentially a 2D matrix of arrays with integers representing the color of a specific pixel. We can use this idea to approach the task by creating a tool that changes those arrays to a set color and redisplay the changes. A configurable drawing tool is also necessary for detailed segmentation, so we created a component class that stored the state of our tool’s shape, color, and size. The second part, drawing on the image, showed more obstacles than the first. The base functionality was simple enough, edit the pixels in the surrounding coordinates of the cursor and render the image with its changes. At first, this bare-bones implementation worked on small images, but would suffer performance-wise with larger images. Larger images would be effected by a noticeable delay when rendering the image with the changed colors. To solve this obstacle, the image was partitioned into four quadrants. By partitioning the larger image into four smaller ones, only the quadrants that had edits made to it had to be rendered again for display. This prevented the interface from making unnecessary renders and improved performance drastically. Another obstacle that showed while implementing the first task was how to deal with intersecting segments. If a user first draws a rectangular shaped segment, and then draws another segment that cuts the first one in half, what should happen? Our clients instructed us that the first segment should be split into two small segments. With these instructions, we then worked on an algorithm that takes all segments made on the canvas, looks for intersections, and then splits intersecting segments.
The second task consists of setting up our backend infrastructure. Live collaboration requires a web socket that synchronizes users when segments are being drawn. To this end, we utilized Socket.IO, a Javascript library that enables communication between clients and a web server [3]. The web server itself was handled by utilizing Next.js’s backend capabilities. It held a server-wide state for each image. When clients are connected to it, any events made are applied to the server’s state, and propagated to all clients. Socket.io was helpful to this end as it handles synchronization of events to prevent data races [3]. Microservices were also developed to fulfill smaller features of this project, a scoring system for gamification and an export to GIF feature. The scoring system took reference images and converted them into a JSON representation. This JSON representation stored the pixel color as a value in an array index that corresponds to its position in the image. When segmentation events are made, the server converts the live image to this same JSON representation and is compared to our reference image for a score. The export to GIF feature is created by listening to all segmentation events that occur to an image and adds a frame to a GIF.

The final task of our project was to host it to display it at the Virginia Tech Science Festival. We utilized AWS EC2 to run and host our website and the microservices. During the festival, we prepared iPad’s with the URL and gave them to attendees to try and segment SEM images. We also connected to our application from a computer that was projected within the Cube, an immersive digital environment. The Cube allowed for a large projection of a single SEM image and display live collaboration from all users. This proved to be the final test of our project’s backend, assessing
whether or not our application could handle collaboration from a maximum of 10 users at once.
3 Requirements

There are 3 main deliverables for this project:

1. Web application that allows multiple users to manually segment SEM images
2. Microservice that scored users’ segmentations to a correct reference image
3. Microservice that exports the segmentation history of an image as a GIF

All of these base requirements are viewed through the web application, making them connected and rather straightforward in terms of functionality and implementation. After intermediate meetings with our clients, the main workflow that needs to be delivered through our website is as follows: the user manually segments a SEM image, then compares it to a reference image that was segmented beforehand. This scoring generates a percentage score that represents how similar the user’s drawn segments are to the reference image. After segmenting, a user can decide to export the history of the image’s segmentation as a GIF.

The user experience while manually segmenting images was the largest priority in this project. Although the image size of SEM images are generally very small (around 500x500 pixel images), ensuring a smooth and responsive canvas was imperative to this project. So optimizations to handle cases like large images were necessary. Another feature the segmenter required was the ability to handle intersecting segments. The clients instructed that all segments are to be contiguous. When two segments intersect, one must split into two separate segments. This logic also applies to erasing a gap in a segment. To handle this, a special algorithm was devised which computes all intersecting segments and recolors segments which have been drawn or erased over.

The scoring microservice needs to update in real time how similar the users’ segments are to a reference image. These reference images were manually completed by members of the team before the festival. This score is a percentage representing how similar the users’ segments are to the segments on the reference image.

The other microservice, export to GIF feature, required the ability to store the entire segmentation session of an image, including all draws and erases, in a GIF. This GIF would be sped up and resemble a time-lapse of all the segments drawn by the users.

The final requirement, although not a deliverable, was that we must display this project as an exhibit at the Virginia Tech Science Festival. Over the course of this day-long event, families from the Blacksburg community will come and test our application by trying to segment a rotating set of SEM images themselves. By presenting
our project at the festival, our application must be able to handle and synchronize segmentation events from a maximum of 10 users simultaneously. It also needed to operate without crashing due to the heavy payload of inputs from different users. Another aspect of this requirement was that our application must be able to be integrated with the Cube’s projector and movement sensors for input. This aspect was mainly handled by our clients because the projector connected to a laptop which can easily connect to our web application.
4 Implementation

4.1 System Design

Our project is encapsulated in a full-stack web application. Detailed explanations for each part (frontend and backend) are included below in their respective subsections.

The initial focus was creating a user interface that included a canvas for users to manually segment SEM images. All of the canvas functionalities were implemented in TypeScript utilizing NextJS and ThreeJS.

After implementing the frontend, we created a scoring microservice and a GIF encoding microservice, and the backend logic which is ran on the NextJS server.

4.2 Frameworks and Libraries

The front end of our application was built using NextJS because it easily allows us to leverage server-side rendering, performed client-side interaction, and the ability to
easily deploy to the web with Vercel. Vercel maintains NextJS, so they optimize their hosting service for NextJS projects, allowing us to streamline continuous deployments with a config-free GitHub integration. We decided to use ThreeJS, a library providing useful WebGL wrappers, and react-three-fiber, a library that wraps ThreeJS functionality in React components. These dependencies were necessary for high-performance rendering to a canvas. Socket.IO was used for realtime websocket communication between clients [3]. The NPM module GIF-encoder was used for encoding GIF’s.

4.3 Architecture

4.3.1 Initial Implementation

Our initial implementation was organized in components, with two routes. The top-level route was a page that allows the user to upload an image. The title and button text were both server-side rendered. The client-side interactivity shipped as JavaScript in the HTTP response.
Once the user uploaded an image, the image was stored as a data URI in local storage and the user routed to /paint. The /paint route was made up of multiple components: The Background Loader, Overlay, Canvas, Action History, Controls, Drawing Layer, and Renderer.

The Background Loader displays a loading text box until the background data URI is loaded from local storage and decoded. When the background is finished loading, the Canvas is rendered as the child of the Background Loader.

The Overlay is rendered mostly by the server, and its client-side interactivity is bundled in JavaScript. The overlay is displayed immediately, overlaid on top of the Background Loader. The Overlay provides UI elements for the user to interact with, allowing them to switch tools, increase/decrease brush sizes, and undo/redo.

The Canvas is rendered once the Background Loader is finished loading. The canvas is sized appropriately to fit the entire background on the browser screen. It provides cursor events to the controls. The Canvas is the immediate parent of the Action History component.

The Action History component is responsible for undoing/redoing actions. This component updates the Drawing Layer. The Action History is the immediate parent of the Controls component.

The Controls component is responsible for handling events provided by the Canvas. It triggers undo/redo on the Action History, updates pan and zoom state, and updates inputs to the tool’s frame callback. The Controls component is the parent of the Drawing Layer.
The Drawing Layer is responsible for tracking segments, statistics about segments, and updating the shader uniforms for the Renderer. The Drawing Layer is the parent of the Renderer.

The Renderer runs the shader using inputs from the Drawing Layer and Controls components.

4.3.2 Multiplayer Overhaul

Upon the completion of our initial implementation, collaborative discussions with our clients gave a more directed vision: transforming the application into a multiplayer collaborative experience would significantly enhance its engagement factor during the Virginia Tech Science Festival. However, to align with this ambitious goal within the constraints of our timeline, strategic decisions were made, necessitating the removal of certain features, namely image uploading and undo/redo functionalities. This decision was necessary as there was not enough time to integrate those features to a multiplayer context.

To begin our implementation, we used the socket.io server NPM module in our NextJS backend API, and used the socket.io-client module in the NextJS frontend to interface with the websocket [3]. Users send ”Draw Events” to the socket.io server, which updates its own representation of the current canvas state, then propagates the event out to all other connected clients. We also used the GIF-encoder NPM module in a separate microservice to allow the client to download a GIF animation of the canvas drawing. Lastly, we added another microservice that would calculate a score for the canvas based on its accuracy when compared to a reference drawing. The microservices can be ran with bun.

```text
export type RoomState = State & {
  rawLog: DrawEvent[];
  shortLog: {
    draws: DoublyLinkedList<DrawNode["value"]>;
    fills: DoublyLinkedList<FloodFillNode["value"]>;
  }
  canvas: State["canvas"] & { node: DrawNode; fill: FloodFillNode | null }[];
};
```

Figure 5: Room State Receiving Draw Events for Each Image
4.4 Frontend

4.4.1 Segment Splitting

Our canvas monitors for user-generated drawings in the interactive canvas environment by noting the segments that intersect and capturing all of the pixel coordinates of these intersections. The program then traverses all of the recorded junction sites, undertaking a careful examination to find occasions when a segment was divided during the drafting process. To ensure the contiguity of segments, the algorithm dynamically recalibrates the damaged segment, assigning a new color and considering it as an independent entity in the event of a split.

4.4.2 Server-Side Rendering

To facilitate swift and efficient renderings of the canvas when draw events occur, we employ NextJS server-side rendering, a feature that improves the user experience. Leveraging this approach, NextJS responds to the client’s request with JSON data to hasten the reconstruction of the canvas. The key data encapsulated within these responses consists of Draw Events, packets of information encompassing user brush strokes represented as vectors, complete with their respective widths. Additionally, the data includes segment splitting events, represented by a single point, serving as a starting point for a flood-fill operation. These segment splitting points aid in the rapid reconstruction of any segments that undergo recoloring due to being split into disjoint parts from a draw event. This design decision helps faster canvas rendering after many segments are split from drawing intersections.

4.5 Backend

The backend is composed of a central server and multiple microservices that handle different jobs such as scoring and encoding to lessen the load on the central server that we want to remain performant. The central server is NextJS’s backend since we are also using it as a proxy to prevent users from directly accessing the microservices themselves to prevent any security issues.

4.5.1 Real-time Sockets

To achieve the collaborative aspects of this project, we used the Socket.io library for real-time communication between multiple users [3]. We chose this library over
Websockets because Socket.io has many features built in already that are beneficial in a collaborative setting such as error handling, automatic reconnection, and fallback to HTTP long-polling. Websocket is just a protocol enables us to build those features but Socket.io saved us a lot of time to focus on other aspects of this project.

```javascript
/**
 * Provides a socket connection to children
 */

export function SocketConnectionProvider(
    props: PropsWithChildren
): JSX.Element {
    const [socket, setSocket] = useState<Socket | null>(null);

    useEffect(() => {
        const newSocket = SocketIOClient(process.env.BASE_URL, {
            path: "/api/socketio",
            addTrailingSlash: false,
        });

        newSocket.on("connect", () => {
            setSocket(newSocket);
        });

        newSocket.on("disconnect", () => {
            setSocket(null);
        });

        return () => {
            setSocket(null);
            newSocket.disconnect();
        }, []);
    });

    return children;
}
```

Figure 6: Socket Connecting Client to a Room

Structurally, we used a room based system where users can join a room where they can join others in segmenting an image together.
4.5.2 Microservices

Using microservices for scoring and GIF encoding was essential due to the single-threaded nature of Node/Bun. Running the GIF encoding or scoring features on the NextJS backend would slow down the server-side rendering and websocket server. We used Bun/Node to run these microservices. The scoring system uses a websocket server (not Socket.IO, rather Bun/Node built in websocket server), to emit score updates to the client. The GIF encoder microservice uses the NPM module GIF-encoder to encode an uncompressed GIF representation of the canvas drawing. The GIF encoder microservice uses the built in Bun/Node http server framework to respond to HTTP requests with GIF blobs.

```javascript
socket.on(
  "draw",
  (data: {
    imageIndex: number;
    draw: Omit<DrawResponse, "historyIndex">;
    fills: FloodFillResponse[];
  }) => {
    const imageState = state[data.imageIndex];
    imageState.nextSegmentIndex = Math.max(
      imageState.nextSegmentIndex,
      data.draw.segment + 1
    );
    applyDrawEventClient(imageState, data.draw, true);
    for (const fill of data.fills) {
      imageState.nextSegmentIndex = Math.max(
        imageState.nextSegmentIndex,
        fill.segment + 1
      );
    }
    floodFillClient(imageState, data.fills, true);
    imageState.gifEncoder.addFrame(imageState.drawing.image.data);
  }
)
```

Figure 7: GIF Encoder Draws Frames on Draw Events
5 Assessment

The project was completed in time to be presented for the Virginia Tech Science Festival. Since this was a large focus from our clients, this assessment of our project will pertain to how the application performed during the festival. During this time period, users freely enter and interface with our application through the use of iPad’s or the Cube’s sensor system.

![Figure 8: The Team and Clients at the Virginia Tech Science Festival](image)

In the assessment of our project’s performance, a pivotal aspect was the simultaneous segmentation of SEM images by users, enhancing the interactive experience. The successful execution of this feature allowed participants to collaboratively annotate and draw on the SEM image, fostering engagement and collaboration. The projection of these segmented images onto the Cube’s floor provided users with a larger-than-life view of their contributions, adding an immersive layer to the visualization process. This not only showcased the technical proficiency of our application in handling multiple segmentation events concurrently but also demonstrated its practical utility in facilitating collaborative exploration and analysis of scientific images. However, our assessment also revealed challenges related to the translation of input
sensor data from the Cube to the computer’s cursor. Specifically, issues arose due to inherent element clipping constraints within the built-in sensors. This clipping limitation affected the precision and responsiveness of the cursor movement, posing a hurdle when the user wanted to click buttons on the UI’s toolbar. Addressing this challenge became crucial for refining the user experience and ensuring the accuracy of interactions. Because the motion sensor was a helmet for users to walk around in, our team would sometimes have to manually move the helmet to avoid cursor clipping so a user can use the toolbar. Another large issue was when our system was strained by trying to maintain synchronization across 10 users at the same time. This issue would sometimes cause lag due to our backend being overloaded with computation requests. Our bandage fix for it during the festival was to restart the server during down times in attendee traffic. Overall it did not cause too much trouble for our users but at times made the user experience less than ideal while segmenting.

The scoring microservice emerged as a notable success in our project’s performance assessment, delivering precise evaluations of user-drawn segments’ similarity to the reference image with proper segmentations. The real-time updating of scores after each segment draw, even when multiple users were actively contributing to the same image, showcased the robustness of this feature. It not only provided an instant feedback loop to users but also demonstrated the seamless integration of collaborative input into a cohesive evaluation framework. However, despite the accuracy and responsiveness of the scoring mechanism, our assessment highlighted an interesting facet concerning user engagement. Notably, the primary user demographic comprised children, who were inherently captivated by the immersive experience of seeing their segmentations projected onto the Cube’s floor. While the scoring microservice functioned as intended, it did not generate as much engagement among the younger audience as anticipated. This insight prompted us to reconsider the balance between analytical features and the inherently captivating aspects of the interactive SEM image segmentation experience.

The inclusion of the "export to GIF" feature added a valuable dimension to our project, allowing the team and clients to showcase the dynamic drawings made by users during the festival in a captivating time-lapse format. Prior to the event, our testing confirmed that the feature successfully generated GIF’s with a comprehensive history of segmentations and erasures, providing a visual narrative of the users’ creative process. However, two notable issues surfaced during testing that deserve attention. Firstly, we observed a degradation in image quality as the duration of the GIF increased. This issue stemmed from inherent limitations in GIF encoding, a challenge that emerged when attempting to compress and represent an extended history of events. Secondly, we encountered a slowdown in the microservice process responsible for generating and downloading the GIF, particularly when the SEM image had
an extensive history of events to compile.

Overall, our project proved to be a success at the Virginia Tech Science Festival. The project was showcased to many users over the course of the festival, and proved to be an immersive and fun experience for the attendees. This was a result of hard work and innovation from the team whilst developing. It is also thanks to the client for giving direction towards developing an application that would be a success at the festival.
6 User’s Manual

A user can access this application via any web-browser. It has been developed mobile-first for primary use on a tablet such as an iPad, but performs equivalently well on a laptop. The following sections explain each feature for a user and exactly how to navigate to them.

6.1 Home Page

To access the home page of this application, a user simply navigates to the hosted URL of the website. They will be greeted by a home page which instructs the user to choose one of multiple rooms. Each room contains a different SEM image for the user to segment. Descriptions of each image are provided to provide context. When selected, the user will enter the room and can begin segmenting.

6.2 Segmenter

Once in a room, the user is presented with a canvas of the SEM image of their choosing. They can begin segmenting the image by drawing on the image. This can be done by dragging their cursor across their desired location on the image. Because different color segments are necessary to be discernible, the user cannot control what color their segments will be. A randomly selected color will be used
each time the user draws on the screen. The user should notice that when they draw intersecting segments, the previously drawn segment will split into two separate ones, with different colors. This is automatic and similar to drawing segments, will have a color randomly chosen for them.

![Figure 10: Example of Drawing Segments](image)

To the left of the canvas, the toolbar can be used to configure the drawing tool. This toolbar includes a link to the home page, buttons to switch between a brush and eraser, and a slider to adjust the size of the drawing tool. The toolbar also includes the score and export options, but these features will be covered in the following sections.
Figure 11: Drawing Toolbar with Brush Size Slider and Eraser

The toolbar enables users to adjust their segmenting experience and erase their mistakes. Different configurations can help users finish segmenting images faster and more accurately. Similar to how drawing over segments earlier, any segment that is split into two from the eraser splits into two new segments with different colors.

Figure 12: Example of Segmenting a SEM Image with the Toolbar
6.3 Score

The toolbar also displays a similarity score for the user. This score determines how close the user’s segments are to a reference copy as the user progresses with segmentation. This score is updated every time a change is made to the canvas. There is no reward or event that occurs when a user gets a perfect score. It is also unrealistic to get a perfect score as the reference image was created manually by the team. This feature is mainly to guide the user into the right direction when they begin segmenting a image.

![Toolbar Displaying Similarity Score](image)

Figure 13: Toolbar Displaying Similarity Score

6.4 Export to GIF

At the bottom of the toolbar, there is a drop-down menu for export options of the user’s segmentation. The export options are shown in Figure 14. There are four options, all completely different from each other. The first option, 'Overlay', exports a downloadable image of the segments that the user drew without the underlying SEM image. 'Full Image' also exports a downloadable image of the segments but includes the underlying SEM image. 'Animation' exports a downloadable GIF which shows a time-lapse of the changes made to the SEM image. This includes segmentations and erases. The last option, 'JSON', exports a downloadable JSON file. This JSON file contains a numerical representation of the pixels of the image and its segments. Pixels that are not edited are represented as -1, while colored pixels are a positive integer. This JSON file was utilized for our scoring algorithm but is accessible for the user to experiment with.

![Export Options](image)
Figure 14: Toolbar Displaying Export Options
7 Developer’s Manual

7.1 Installation

First, clone the project from our GitHub project repository. Our GitHub repository can be found at:

https://github.com/SEM-Analysis-AI-Game/sem-analysis-ai-game

Our project utilizes Bun, a JavaScript runtime package manager, test runner, and bundler. We chose this due to its superior performance compared to NodeJS for package installation and management. As an extremely new technology, it only supports macOS and Linux. Therefore, development on a Windows device needs to be through WSL (Windows Subsystem for Linux). Make sure you clone the project repository to the virtual machine itself or performance will hinder drastically.

To install Bun, run the following command:

curl -fsSL https://bun.sh/install | bash

Once the project is cloned and bun is installed, navigate to the project’s root directory. Run `bun install` to install all required packages. Finally, navigate to `apps/next-app` and use `bun run build` to build the application, then `bun run start` to run the website in production mode. You may also use `bun run dev` to run developer mode, but there are many issues with running in developer mode because NextJS does not retain allocated memory on the server when running in developer mode, since pages are compiled dynamically. You may run into issues running the website due to Bun’s version. This was a problem some members faced when trying to develop, but solved using `bun upgrade`. Also, ensure `node` version 18. is installed. It is important to also run the microservices, which is explained further below.
7.2 Repository Structure

As you can see in Figure 15, our repository contains 3 main folders within the `apps` folder. We also structured our development flow by utilizing the `main` branch as our production branch while making separate branches for each feature. The three folders under `apps` corresponds to each of the deliverables described earlier. The frontend and painter are contained in the `next-app` folder. The scoring and export to GIF microservices are located under the `scoring` and `gif-encoder` folders respectively.
We also utilize packages to abstract features and isolate each feature from each other.

### 7.2.1 Painter

The painter is a part of the central server and is ran when the NextJS application is started. To reiterate, make sure that the app is started in production mode if fully correct functionality is needed since there are issues with memory retention in dev mode with NextJS.
As seen in Figure 17, all files related to the painter and the toolbar are under the
painter folder. The backend that keeps track of each room’s state is kept under the server folder.

7.2.2 Scoring Microservice

Scoring of a painted segmented image is done through the scoring microservice that keeps a synchronized state with its corresponding image room. Because of that, its important that this microservice is running while painting to be able to score correctly. The service is ran with `bun dev` in is project directory.

![Scoring Microservice File Structure](image)

Figure 18: Scoring Microservice File Structure

7.2.3 Export to GIF Microservice

Exporting a segmented image is done through the GIF encoder microservice that keeps a draws frames to a GIF file as updates to the state occur with its corresponding image room. Because of that, its important that this microservice is running while painting to be able to score correctly. The service is ran with `bun dev` in is project directory.
Figure 19: Export to GIF Microservice File Structure
8 Lessons Learned

8.1 Timeline

Our initial timeline was set with broad goals by the client, who at the start of the semester, did not have a concrete idea of what they wanted except for their individual motivations. To reiterate from the motivations section, these motives involved: a painter to segment SEM images, a way to compare to reference images (originally looked at potential use of AI to create reference), and to garner community interest by presenting at the VT Science Festival.

The team first met in late August around the start of the class’s project launch to gauge the scope of the project from the clients. We were introduced to the motivations, goals, and constrictions for the project. From there, large milestones were set to meet each of the large goals the clients had. In addition, bi-weekly meetings were held in order to set smaller, more technical goals which are described in an improved timeline.

As the semester progressed, our clients redirected the focus of the requirements of our project. The main focus of the VT Science Festival changed our goals to heavily
focus on creating an exhibit where multiple users may interact with the segmenting tool. This change of focus resulted in a new timeline.

The new timeline illustrates all the features and their technical development throughout the semester. It also illustrates the difference in direction we took with this project in comparison to the start of the semester. We first fleshed out the individual user’s experience using the segmenter tool, implementing a splitting algorithm and painter toolbar. We then worked on integrating a web socket to support live collaboration during the science festival. Finally, we completed the scoring and export microservices. This progression highlights the change in focus from simply creating a segmenter interface for one to a web application that can support the audience of the festival.

8.2 Problems and Future Work

Although the demo at the Virginia Tech Science Festival was a great success and engaged hundreds of young students, there is still much more work that could be done to improve the application. A very notable issue during the festival was the progressive degradation of the application speed as a given image was segmented. This posed a significant hindrance during the festival as the loading time for a room would progressively increase. Beyond improving the performance of the application, several auxiliary improvement features could be made. These include implementing automatic segmentation and comparison facilities as well as a dashboard for researchers to examine the crowd-sourced segmentations. These areas for improvement are highlighted below.
8.2.1 Scalability and Performance

At the Virginia Tech Science Festival, our application successfully handled up to approximately 10 simultaneous users. While this performance met our immediate needs, our application ran into some performance issues for some users. Users would experience some delay in the propagation of events from the server to clients. This made their experience a bit laggy. To address this limitation, future endeavors in scalability must commence with rigorous stress testing of the current application, aiming to identify and understand any performance bottlenecks.

It is speculated that the single-threaded nature of JavaScript runtime environments might be a significant hindrance when confronted with a higher user volume. To mitigate this, a prospective approach involves transitioning the socket code to a more adept and multi-threading-capable language, such as C++ or Go. This strategic shift aims to enhance the application’s capacity to handle a larger number of concurrent users seamlessly.

Presently, the application architecture involves the user’s drawn segments being communicated to the server, with the updated state subsequently broadcasted to all users within a given room. While this method served its purpose during implementation, it prompts us to consider more efficient state synchronization techniques that minimize the bandwidth requirements for state change requests. Investigating and implementing such methods could significantly benefit the application’s overall performance, especially in scenarios where bandwidth limitations may pose challenges.

It is important to note that delving into advanced state synchronization methods was beyond the scope of our project’s requirements. However, recognizing the potential impact on scalability and user experience, future development efforts should consider exploring and integrating these optimizations to ensure the application’s robustness and responsiveness as user numbers increase.

8.2.2 Automated Segmentation

The initial scope of this project included a facility to automatically segment an SEM image and use that for comparison to a manual segmentation.
An initial choice we made was to attempt to implement Meta’s Segment Anything Model on the client using a WONNX runtime [1], [2]. The motivation for this was to reduce application dependency on a network and provide decreased latency in segmentation analysis. This decision, however, introduced many complex technical challenges. It was a field of software engineering that nobody on the team had real experience with. In the end, we opted for a very viable solution of which most team members had relevant experience with, and that was to simply implement the segmentation analysis as part of a backend microservice.

Unfortunately, after this significant effort, we realized that a practical implementation of an accurate, fully automatic segmentation was beyond the scope of what we could achieve. We determined the performance of Meta’s SAM model on SEM images was weak and would not fit our purposes out of the box [1].
As can be seen on the sample automatic segmentation in Figure 22, Meta’s SAM fails to segment a significant portion of the image [1]. Additionally, as the ambiguity of segments in higher complexity SEM images increases, SAM’s efficacy decreases markedly. We believe that this is due to the fact that SAM is trained on natural images of the real world and did not include any images that were taken through specialized means such as SEM images.

Future work could involve determining how to improve the accuracy and reliability of an automated approach to automatic segmentation. This may involve fine tuning existing models such as SAM, training new models out of the box, or utilizing novel computer vision techniques particularly suited to this task [1]. Another potential approach could be a hybrid between manual methods and automatic segmentations. For example, a very rough manual segmentation could be performed where a user simply marks where each individual in the image is. Then a refinement algorithm or model could fill in each manual point with the proper complete segmentation.

Ultimately, this was beyond the scope of work our team had experience in and felt it was beyond the scope of the project.
8.2.3 Researcher Features

In a much more scaled variation of this application that attempts to seriously crowdsource segmentation data for SEM images, a more robust result handling system would be necessary. This was a feature explored at the beginning of this project, but other features took precedence such as the gamification of the segmenter and the multiplayer component. Future extensions to this application could include the ability for verified researchers to create accounts and upload images that they need to be segmented. A dashboard would also exist to allow researchers to manage the images they need segmented and view the results of their segmented images. When considering the expansion of this application in the domain of serious research endeavours, this would be considerably useful.
9 Acknowledgements

We would like to thank our clients, Dr. Monet Roberts, Dr. Yinlin Chen, and Tanner Upthegrove, for introducing us to the project, leading us throughout the semester with bi-weekly meetings, and helping develop an application that can positively impact the bioengineering community.

Dr. Monet Roberts - monetr@vt.edu
Dr. Yinlin Chen - ylchen@vt.edu
Tanner Upthegrove - upty@vt.edu

We also would like to thank our professor, Dr. Mohamed Farag, for advising us to complete this project throughout the entirety of the semester.

Mohamed Farag - mmagdy@vt.edu
10 References

References

