NeuraViz: A Web Application For Visualizing Artificial Neural Network Structures

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin–La Crosse

La Crosse, Wisconsin

by

Bennett Wendorf

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

May, 2024

NeuraViz: A Web Application For Visualizing Artificial Neural Network Structures

By Bennett Wendorf

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Prof. Albert Einstein Examination Committee Chairperson	Date
Prof. Isaac Newton Examination Committee Member	Date
Prof. Marie Curie Examination Committee Member	Date

Abstract

Wendorf, Bennett, "NeuraViz: A Web Application For Visualizing Artificial Neural Network Structures," Master of Software Engineering, May 2024, (Jason Sauppe, Ph.D.).

This manuscript describes the software engineering processes and principles adhered to during the development of Neuraviz, a web application for visualizing artificial neural network structures. Users upload pre-trained machine learning models from popular frameworks including Pytorch and Keras, and Neuraviz generates a visual representation of the model's architecture. The following manuscript focuses on the design, implementation, testing, and deployment of NeuraViz in an effort to comprehensively encapsulate the entire development process.

Acknowledgements

I would like to extend my sincerest thanks to my project advisor, Dr. Jason Sauppe, for his guidance and support throughout the development of NeuraViz. His feedback was always crucial in pointing me in the right direction, especially when I was overwhelmed with possibilities.

Thank you also to the entire computer science department at the University of Wisconsin-La Crosse for tirelessly helping me through all my coursework and projects throughout my tenure at the university. My ability to complete this monumental task would not have been possible without them.

I would also like to thank the open source community for providing the tools and resources used in this project. Open source software is an integral part of the modern software space and none of our lives would be the same without it.

Finally, I would like to thank my parents, family, friends, and all the teachers, mentors, and colleagues who have helped, supported, and encouraged me throughout my life. I am more grateful thank I can possibly express in words.

Table of Contents

List of Tables

List of Figures

Glossary

1. Introduction

1.1. Overview

This project aims to develop a software system to visualize the architecture of artificial neural networks. Neural networks (NNs) are a class of machine learning models that are inspired by the structure and function of the human brain. They are composed of a large number of interconnected processing elements, called neurons, which work together to solve complex problems. The architecture of a neural network refers to the arrangement of neurons and the connections between them. In addition to the general structure of a network, the weights and biases that control its function can provide useful insight into the inner workings of the model. The final integral parts of the neural network architecture are the activation functions that govern how data propagates through the network. Visualizing the architecture of a neural network can help students and researchers understand the structure of the model, identify potential issues, and communicate the model to others. More information on neural networks in general can be found in section 1.4.

This project will develop a web application that allows users to upload pre-trained neural network models and generate visual representations of their architecture. The application will support models trained using popular machine learning frameworks such as PyTorch and Keras. The resulting graph structure will be visualized in a pannable and zoomable svg format that shows the ordering of neurons, biases on those neurons, the weights of the connections between them and activation functions on each layer.

The application will be designed to be user-friendly and accessible to a wide range of users, including students and researchers. It will be implemented using modern web technologies and will be deployed as a web service that can be accessed from any device with an internet connection. The project will follow best practices in software engineering, including requirements analysis, design, implementation, testing, and deployment. The resulting application will be a valuable tool for anyone working with neural networks, but will be primarily targeted toward post-secondary educational students learning about machine learning and neural networks for the first time.

1.2. Similar Projects

Neural networks are a notoriously difficult concept to understand, especially for those who are newer to the field of machine learning. The architecture of a neural network is a key component of understanding how the model functions, but it can be difficult to visualize and comprehend. There are a handful to tools available for visualizing neural network architectures, but they are often limited in their scope. For example, Google's TensorBoard is a popular tool, but only natively supports TensorFlow and Keras models. In researching existing tools such as TensorBoard, Netron, and ENNUI, it was found that none of them supported the same range of formats as NeuraViz. This project aims to develop a user-friendly tool that is accessible to a wide range of users, including students and researchers who are new to the field of machine learning, and to support a wider range of model formats than other offerings.

1.3. Goals

With an aim to solve the issues with existing tools, NeuraViz will be developed with a few key goals in mind. One main goal is to support a wide range of formats and to visualize each format in a standard way. This will allow users to upload models from a variety of frameworks and see a consistent and comparable visualization of their model.

NeuraViz is also designed specifically with user-friendliness and simplicity in mind. The minimal interface focuses on the visualizes of the neural networks themselves, with minimal distractions. The use of color and shape in the visualizations will help to make the network architecture more understandable at a glance, quickly identifying neurons and connections that are the most important. Where needed, users can also zoom in or out an click on elements to see more detailed information.

Portability is another key goal of the development. Modern web technologies ensure the application is accessible from a range of devices, though it is most optimized for a desktop or laptop experience. Since the application is deployed as a web application, users don't need to install software on their own machines, or even sign in to be able to use the tool. To enhance this ability further, graph representations can also be exported as raw SVG or in tikz format for use within LaTeX documents.

1.4. Neural Networks

Neural networks are a type of machine learning model that are inspired by the human brain. They are made up of layers of neurons, which are connected to each other. Each layer of neurons takes in a number of inputs, processes them, and then outputs a value. The value that is output is then passed to the next layer of neurons, and so on, until the final layer of neurons outputs the final result.

1.4.1. Neurons

In an artificial neural network, neurons are the primary pieces of the network that perform the computation necessary. They are organized into groups called layers, typically represented in a graph structure organized vertically so the neurons in a layer are in a sort of column. Typically, the first layer is called the input layer, and behaves differently than other layers. For this reason, input neurons are represented as grey squares in NeuraViz as opposed to the typical neurons' circles. Neurons run the computations needed for the network to function with complex algorithms that I'll skip over here for simplicity. In general, neurons in a layer are connected to all the neurons in the previous layer, and all the neurons in the next layer. These connections are represented as lines between the neurons in NeuraViz.

1.4.2. Edges

Edges are the connections between neurons in a neural network. They are the primary way that information is passed between neurons. In NeuraViz, edges are represented as lines between neurons. Edges also have weights, which are used in the computation to determine how important that connection is. For small enough networks, the weights can be seen by hovering over edges in NeuraViz.

1.4.3. Activation Functions

Activation functions are a key part of how neural networks work. They are used to determine the output of a neuron based on the inputs it receives. There are many different activation functions, but one of the most common ones is the sigmoid function. The sigmoid function takes in a number and returns a number between 0 and 1. This is useful because it allows the network to output a probability. In NeuraViz, activation functions are shown at a layer-level, and represented as icons toward the top of each layer. Hovering over these icons will show the activation function used in that layer.

2. Software Development Process

2.1. Overview

Developing software is an extensive and complex process that requires a lot of planning, both in relation to the methodologies used during the development process and the requirements, both functional and non-functional of the software. This section details life cycle models considered for NeuraViz's development, the model that was eventually chosen, and modifications to the model necessary for the development of this particular system. It also outlines functional and non-functional requirements for NeuraViz.

2.2. Life Cycle Model

Prior to beginning development of NeuraViz, a number of software life cycle models were considered to govern the pace and structure of development. In all, the waterfall model, iterative model, and agile model were considered. More specifically with agile, a variation of scrum, modified for a single developer, was considered. Ultimately, the modified scrum model was chosen for its flexibility and ability to adapt quickly to changing requirements.

2.2.1. Waterfall Model

The waterfall model is one of the oldest software development lifecycle (SDLC) models, originally proposed by Winston Royce in 1970 [1]. The model is a linear, sequential approach to software development, with each phase of the development process directly following the previous phase. Each subsequent phase relies on the previous phase, and as such the model does not allow going back to previous phases once they are completed. The first phase of the waterfall model is the requirement analysis phase, in which project requirements, both functional and non-functional, are gathered and documented. At this phase, requirements are also often analyzed for traits like consistency and feasibility. The second phase is the system design phase, in which the architecture of the software system is designed in full. All details of what needs to be done and how it will be completed are considered and documented during this phase. Third, the implementation phase is where the code for the software is written and the design from the previous step is implemented in full, exactly as specified during the design phase. During the fourth phase, the software is tested for bugs and errors, and issues are resolved as needed. Fifth, the software is deployed to the client in its entirety. In the waterfall model, this is the first time the client has seen the software. Finally, the software is maintained and updated as needed for as long as the client needs it. Figure 1 shows a diagram of the waterfall model and its constituent parts.

Because of its rigid structure, the waterfall model excels at being very easy to understand and pick up quickly for new developers, which was initially intriguing during model selection for NeuraViz. In addition, it is easy to manage with relatively little overhead in management. When project requirements are well understood up front and unlikely to change, the waterfall model also serves the benefit of ensuring design is completed before implementation begins, which leads to fewer mistakes and less necessity to change the code once it has been written. However, for projects where requirements are less well understood or are likely to change, such as in this project, the waterfall model struggles to adapt and may lead to a design that was flawed in the first place with no way to fix it. In addition, the waterfall model does not allow for client feedback until the software is fully completed, which can lead to a lot of wasted time and effort if the client is not satisfied with the final product. In the case of this project where adaptability was crucial, the waterfall model would have been a poor choice.

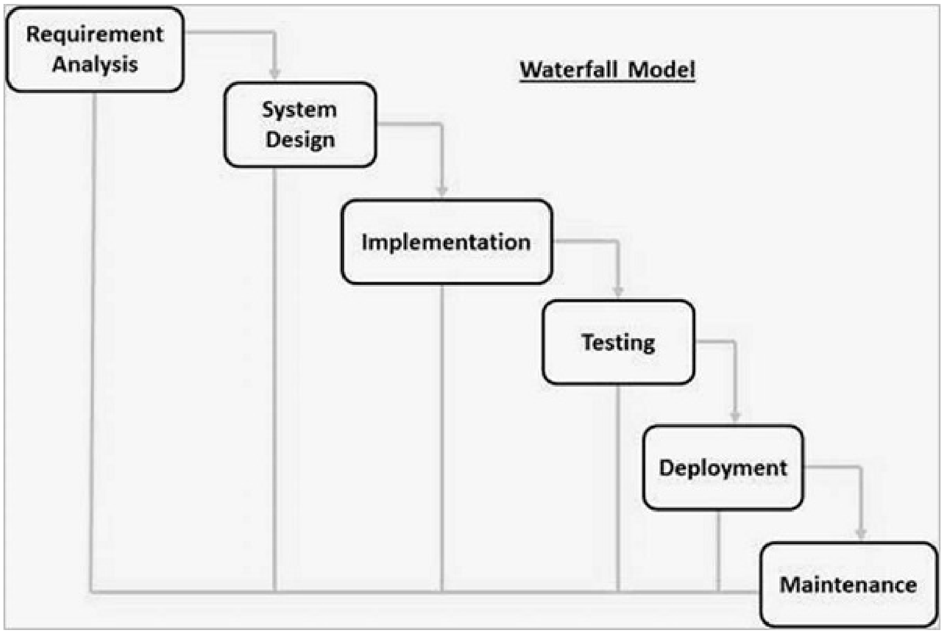

Figure 1. Waterfall Model Diagram [4]

2.2.2. Iterative Model

Like the waterfall model, the iterative model is mostly linear and sequential with a relatively rigid structure where each step directly follows the previous step. The phases in the iterative model, as shown in Figure 2, roughly match those in the waterfall model, including a requirements analysis phase, a design phase, and implementation phase, a testing phase, and a deployment phase. Unlike the waterfall model, however, the iterative model runs these phases, with the exception of requirements analysis, multiple times, restarting the sequence of phases after each deployment. This serves the major benefit of allowing the ability for the client to give feedback on the project sooner and more often.

Due to its similarity to the waterfall model, the iterative model exhibits many of the same benefits of waterfall in that it is easy to understand with a relatively linear structure and minimal overhead. Like the waterfall model, when requirements are understood at the project outset, the design is likely to be almost fully complete before development begins, so the code is also less likely to require changes later in the process. While the iterative model does improve on the waterfall model's lack of ability for client feedback, it still struggles with changing requirements as each iteration of the project is still long and expected to be a relatively complete implementation of the software. The lack of adaptability made the iterative model a poor choice for NeuraViz's development where changing requirements were expected from the beginning.

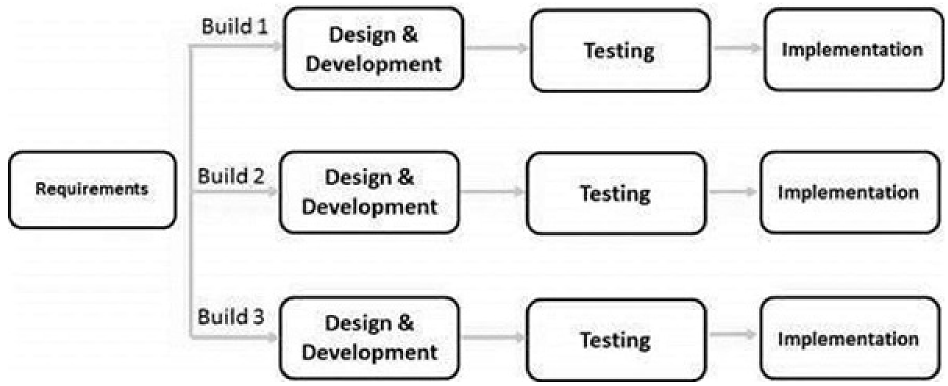

Figure 2. Iterative Model Diagram [3]

2.2.3. Agile (Scrum) Model

In contrast to the other models, agile models, and specifically scrum, was designed with the express intent to adapt to changing requirements quickly. Since the exact requirements and design for NeuraViz were not known well ahead of time, scrum was chosen for development due to its ability to pivot quickly when new design details were discovered. While agile is a category of software development models that focus on adaptability, scrum is a specific type of agile model that places emphasis on small, self-organized teams working in short, iterative cycles called sprints. Each sprint is typically two to four weeks long and ends with a review of the work completed during the sprint and a planning session for the next sprint. The scrum model is shown in Figure 3. While the diagram shows a two to three month timeline per iteration, scrum more typically follows a shorter sprint length.

Since NeuraViz only has one developer, the scrum model doesn't fit perfectly. However, many aspects of the model do fit relatively well, with some slight modifications. Scrum typically emphasizes daily stand-up meetings with each team of developers. Due to the longer timeline of NeuraViz's development and the single developer, these meetings were partially dealt with through daily review of the project board to help ensure that projects stay on track. In addition, the developer met every week with the project advisor, Dr. Jason Sauppe, who in some sense served as a stakeholder on the project and a product owner. While these meetings don't match exactly with any part of the scrum model, they serve as a combination of stand-up meetings and sprint retrospectives. For the purposes of this project, sprints were completed every week. This allowed for very quick turnaround on features, and enabled constant feedback and reflection on project requirements.

2.3. Development Process Technologies

An important part of the scrum model is managing small, independent projects that can be completed in a short amount of time. To help manage these projects, Jira's scrum template was used. This template allows for a main project backlog where user stories can be converted to sprint tasks. Each sprint can then be created and tasks can be scheduled into one. Once each sprint begins, projects/tasks move through columns including backlog, programming, testing required, and done. This allows for a clear view of exactly what is being worked on

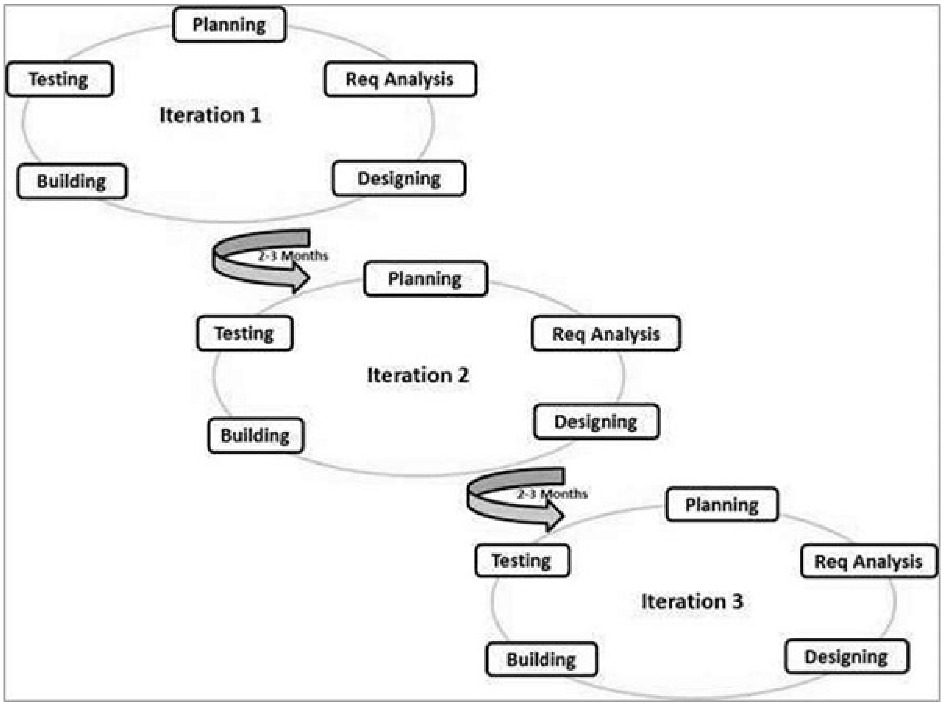

Figure 3. Scrum Model Diagram [2]

at any given time. The scrum board for sprint 21 (March 5th, 2024 - March 19, 2024) can be seen in Figure 4. Jira provides exceptional features for managing projects in a scrum format, allowing the tracking of not only sprint tasks through the stages of development on the project board, but additional features such as time tracking on projects as well.

In addition to Jira, Github and specifically branches were used to help keep track of individual sprint tasks. A new branch was created for each task, with the name of the branch including the task key from Jira and a brief description. Jira's integration with GitHub provided a link to see what phase the code was actually in directly from the Jira task, including whether a pull request had been created or merged.

2.4. Functional Requirements

Since agile methodologies were chosen for the development of this project, a set of functional requirements in the form of user stories were collected prior to the start of development. These user stories served as a guide for features to implement and helped ensure that no major functionality was missed during development.

NeuraViz is a relatively simple application with only one type of user. As such, the functional requirements are relatively straightforward, including the ability to upload a pre-trained machine learning model trained in either Pytorch or Keras. In addition, users should be able to see either the full graph of the uploaded model, or a collapsed version depending on the scale of the uploaded model. Additional functionality is also documented such as the ability to navigate the page via pan and zoom functionality on the graph, and clicking or hovering on various network components.

In addition to the functionality of viewing the network itself, user stories were also docu-

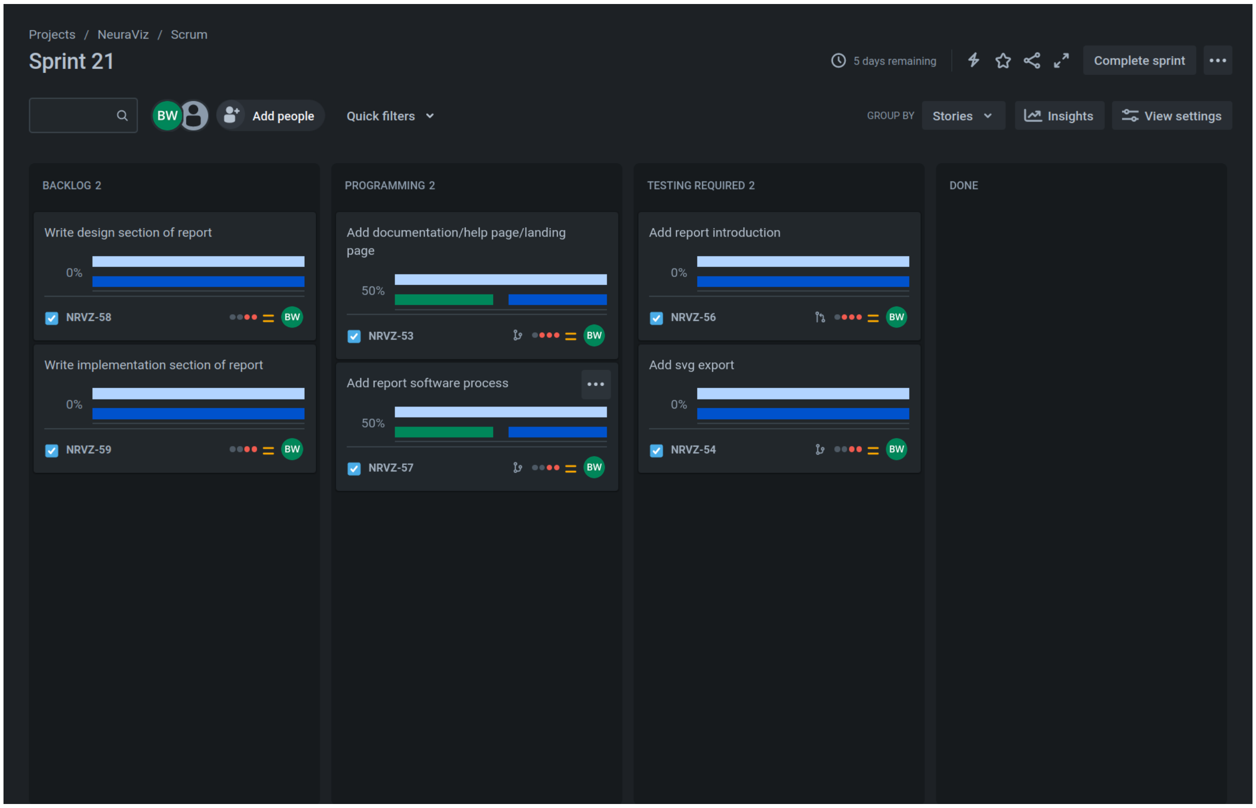

Figure 4. Jira Scrum Board

mented for export functionality, allowing the graph of the model to be exported both as an svg and as a LaTeX document in the format of the tikz library.

The full set of user stories is documented in the user stories document. This document also includes user stories for functionality that was not implemented in the current version of NeuraViz, but may be implemented in future versions.

2.5. Non-Functional Requirements

In addition to the functionality documented as user stories, non-functional requirements were also documented to ensure that the user experience of NeuraViz was as smooth as possible. Identified non-functional requirements are as follows:

- Large network layers are collapsed if they are too big to reasonably render.
- If I am unable to view the model visually, labels exist for screen readers as much as possible.
- If the site takes a long time to load, skeletonized components are shown to indicate that the site is still loading.
- As a user, my data is reasonably secure, both during transmission and processing.
- Themes are sufficiently differentiable for colorblind users.

- Invalid models are rejected and not stored unnecessarily.

3. Design

3.1. Overview

NeuraViz follows a fairly standard server-client web application architecture. The client is responsible for rendering the user interface and allowing the user to interact with the application. The server handles the actual computationally intensive processes such as parsing the uploaded model and generating the structure of the visual representation. The server also handles the storage of the uploaded models during user sessions. It also handles translation of the visualization into various formats.

3.2. UML Class Diagram

The UML class diagram in Figure 5 shows the classes and their relationships in the NeuraViz application. The diagram is divided into two main sections: the frontend and the backend, which are also commonly referred to as the client and server respectively.

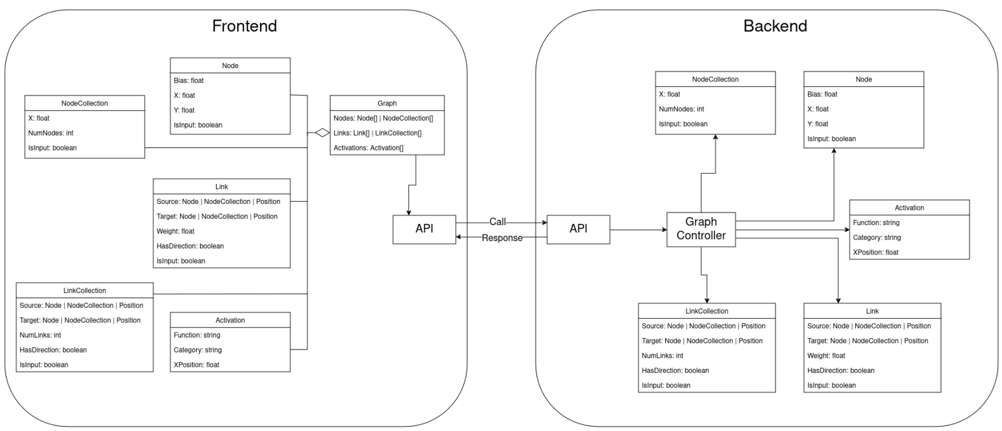

Figure 5. UML Class Diagram

3.2.1. Frontend/Client

The frontend primarily relies on the Graph object, which is comprised of a number of Nodes and/or Node Collections, Links and/or Link Collections, and Activation Functions. Nodes represent individual nodes as represented in the graph, and these are used for nodes in graph layers that are smaller than 10 nodes by default. For layers that are too large, the graph representation instead contains a Node Collection that represents the layer as a whole. Links and Link Collections operate a similar way. Activation Function objects represent the activation functions that can be seen as small icons at the top of each layer in the NeuraViz interface. The graph object houses the representation of the neural network model as ready for rendering. As shown in the UML diagram, the frontend also houses an API component that is responsible for communicating with the backend architecture via standard HTTP requests.

3.2.2. Backend/Server

The backend is responsible for handling the computationally intensive processes of parsing the uploaded model and generating the structure of the visual representation. As seen in Figure 5, the backend houses objects that almost perfectly mirror the frontend components. However, on the server, these components are all related to the graph controller: the component responsible for the actual graph parsing. In addition to parsing the actual graph, the controller also handles additional requests for retrieving a stored model and converting the representation into various formats. Like with the frontend portion of the application, the backend houses an API component that is responsible for receiving the HTTP requests from the client and routing them to the correct controller endpoint for processing, as well as sending the response back to the client.

3.3. Database

At the outset of NeuraViz's development, no database was planned to be used. The nature of the application is such that the primary functionality of the application should not require a user to log in, and NeuraViz itself does not need to store information of any kind. Initially, the users' uploaded models get saved to disk during processing, but are then deleted immediately after for security and space efficiency. However, once the LaTeX export feature was introduced, it became necessary to maintain the graph's representation for longer, or to send it back and forth between the client more. Since the graph representation can be quite large, it was decided to use a NoSQL database, namely MongoDB, to store the parsed graph information as a session.

When a user makes their first request to the NeuraViz application, a session is created and the client is given an identifier. Upon graph parsing, the graph representation is stored in the database under the session identifier. Further requests can then retrieve the stored graph representation from the database, rather than having to re-upload the model and re-parse it. In addition to the LaTeX export feature, this also allows for the possible future features of saving the graph representation of a user's account for future reference, providing further granularity on larger networks, and more.

3.4. User Interface

A major step in the design process was developing the look and feel of the interface that users would be interacting with. A user interface mockup was drawn in Gimp to give a visual representation of what the application would look like. The mockup served as a guide in developing the actual user interface, though some changes were made to the final product. Since NeuraViz operates on a single page with one main piece of functionality, the required mockup was fairly simple. Figure 6 shows a number of components that were included in the final product. The file upload button can be seen in the side panel on the left, along with its model validation text. Below that can be seen a section for settings with an example of what a setting with a slider might look like. While no final components use a slider so far, future development may include more complex settings. In addition to the sidebar, the primary visualization window can be seen with a sample model visualization. In the bottom right corner, navigation buttons can be seen in the mockup, mirroring the final interface.

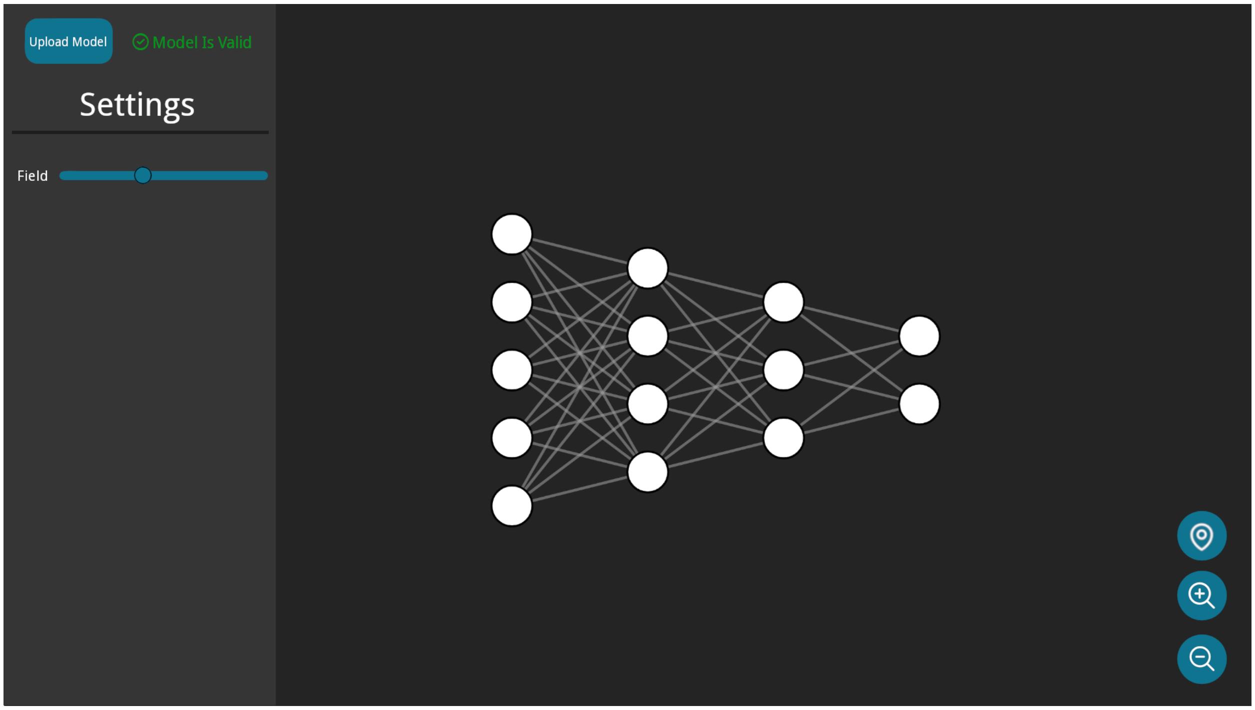

Figure 6. User Interface Mockup

3.4.1. Final Interface

The final interface of NeuraViz is shown in Figure 7. The interface is divided into two main sections: the sidebar and the main visualization window. At the top of the sidebar, the file upload section can be seen, including a file picker, upload button, and model validation text. Below that, the options for visualization export can be seen, with buttons for both exporting the visualization to LaTeX and to SVG. Next the settings panel can be seen, with a mode toggle for the color scheme of the application. At the bottom of the sidebar there is a color reference key for the colors used in the main visualization window.

The main visualization window contains the visualization of the model, or an indication that the user should upload a model. The colors of nodes and edges correspond to the color key found on the sidebar. Navigation buttons can be found at the bottom right of the page.

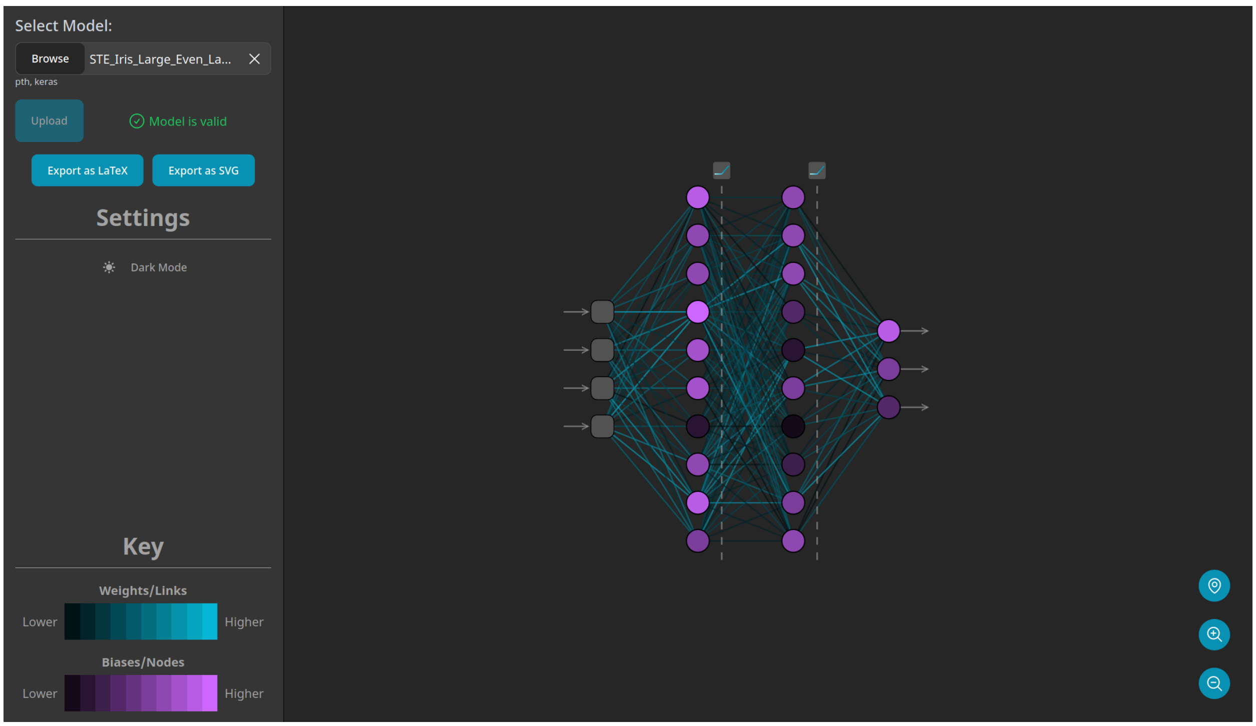

Figure 7. Final Interface

4. Implementation

4.1. Overview

4.2. Technologies Used

4.2.1. Client

4.2.2. Server

4.2.3. Data Layer

4.3. Development

4.4. Deployment

5. Testing

5.1. Overview

5.2. Verification

5.3. Validation

6. Security

6.1. Overview

6.2. Threat Model

6.3. Session Management

6.4. Web Application Security

7. Conclusion

7.1. Overview

7.2. Challenges

7.3. Future Work

8. References

[1] Gagan Gurung, Rahul Shah, and Dhiraj Jaiswal. Software development life cycle models-a comparative study. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pages 30–37, 07 2020.

[2] TutorialsPoint. Sdlc - agile model. `https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm`. accessed: 03.14.2024.

[3] TutorialsPoint. Sdlc - iterative model. `https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm`. accessed: 03.05.2024.

[4] TutorialsPoint. Sdlc - waterfall model. `https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm`. accessed: 03.05.2024.

9. Appendices