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

1.2. Background

1.3. Goals

2. Software Development Process

2.1. Overview

2.2. Life Cycle Model

2.3. Requirements

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 1 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.

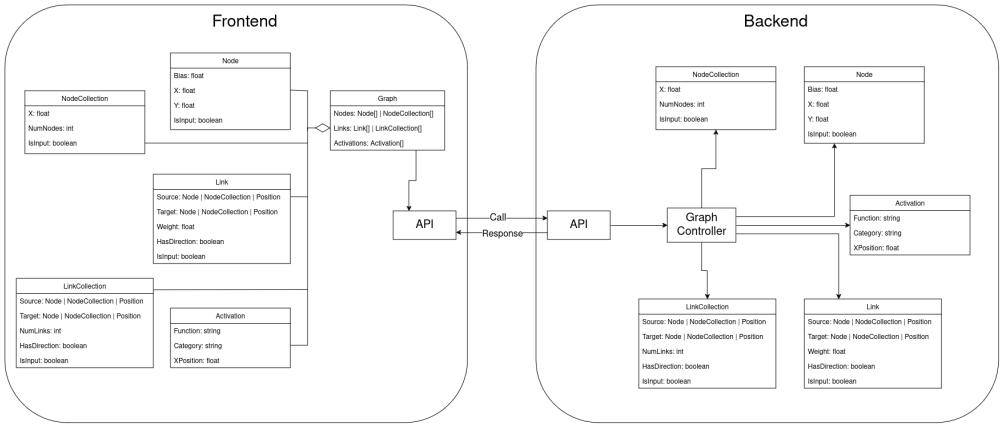

Figure 1. UML Class Diagram

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 1, 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 2 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.

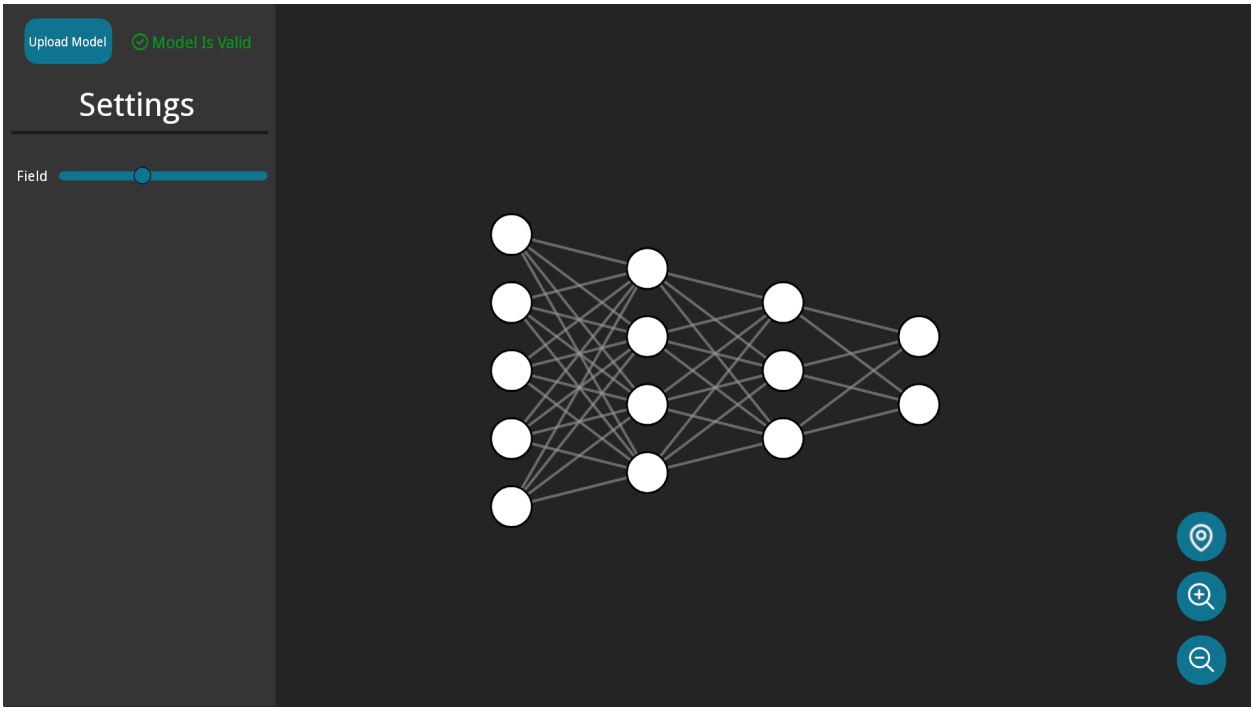

Figure 2. User Interface Mockup

3.4.1. 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. Bibliography

9. Appendices