



universität
wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

**„Integrating public protein-protein interaction data into
the Virtual Reality platform VRNetzer“**

verfasst von / submitted by

Till Pascal Oblau B.Sc.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of
Master of Science (MSc)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt /
degree programme code as it appears on
the student record sheet:

A 066 875

Studienrichtung lt. Studienblatt /
degree programme as it appears on
the student record sheet:

Masterstudium Bioinformatik

Betreut von / Supervisor:

Univ.-Prof. Dr. rer. nat. Dipl.-Phys. Jörg Menche

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor Univ.-Prof. Dr. rer. nat. Dipl.-Phys. Jörg Menche, for supervising this project and providing me with invaluable guidance and support. His expertise and dedication to the field of network science have been instrumental in shaping my research skills and knowledge. I appreciate the chances that he has given me to expand my knowledge and skills as a researcher, and for his constant encouragement throughout this project.

I want to express my deep appreciation to my colleagues in the Menche Lab. First, I would like to thank Sebastian Pirch and Martin Chiettoni for their exceptional technical expertise. Sebastian deserves special recognition for creating the VRNetzer ecosystem, which provided the framework for the work presented in this thesis. I am also thankful for his consistent support and guidance throughout the project. Additionally, I am thankful for the valuable insights into network science and network curation provided by Dr. Felix Müller and Christiane Hütter. I am also grateful for the feedback and suggestions from Mathilde and all the other colleagues who contributed to my thesis. Your contributions were greatly appreciated.

I am also indebted to Dr. Nadezha T. Doncheva from the JensenLab of the University of Copenhagen for her assistance and advice on Cytoscape app development.

I also want to express my gratitude to Jürgen Knapp, my former high school teacher, for early on discovering and encouraging my interest in natural science. His passion for teaching and science are inspiring, and I will always be thankful for the impact he had on my life and career.

Lastly, I would like to express my heartfelt appreciation to my parents for making it possible for me to study in the beautiful city of Vienna. Their unwavering support and encouragement were essential to my success, and I am forever grateful for that. I am also grateful for the emotional and advisory support of my family and friends, whose encouragement and belief in me were essential to my success. Their unwavering support helped me overcome challenges and stay motivated, and I am grateful for their presence in my life.

I apologize for not being able to mention everyone who contributed to my journey, but please know that your support has been invaluable to me. I am deeply grateful for the encouragement, guidance, and inspiration provided by all those who have supported me along the way.

KURZFASSUNG

Der VRNetzer ist eine Softwareplattform für Netzwerkvisualisierung, die es ermöglicht, sehr große, komplexe Netzwerke in einem immersiven Umfeld zu analysieren. Derzeit stehen jedoch nur einige wenige Netzwerke zur Verfügung, und die Aufbereitung neuer Netzwerke für eine Analyse in Virtual Reality kann für technisch weniger versierte Nutzer herausfordernd sein.

Im Rahmen dieser Masterarbeit wurden einige Werkzeuge und Funktionalitäten entwickelt, welche zur Verbesserung der VRNetzer Plattform beitragen. Dazu gehören Erweiterungen für das Back- und Frontend sowie die Ergänzung neuer, auf biomedizinische Forschung angepasste, Inhalte.

Die erste Verbesserung besteht aus einem neuen, erweiterungsbasierten System, das die Entwicklung eigener Inhalte wie neuer Analyse-Funktionen oder Benutzeroberflächenelementen erleichtert und den Austausch mit anderen Nutzern ermöglicht. In dieser Masterarbeit wurden zwei Erweiterungen entwickelt, die das Potenzial dieses Systems hinsichtlich Interoperabilität und Modularität für zukünftige Entwicklungsprojekte verdeutlichen. Beide Erweiterungen bringen ihren eigenen Quellcode, Benutzeroberflächen und andere Ressourcen mit, die verwendet werden können, um die Funktionalität der VRNetzer-Plattform zu erweitern. Insbesondere ermöglichen sie neue Verbindungen zwischen dem VRNetzer und anderen Softwareanwendungen und Werkzeugen, die in der biomedizinischen Forschung häufig verwendet werden.

StringEx ist die erste Erweiterung, welche die VRNetzer-Plattform mit Cytoscape verbindet. In Kombination mit der Cytoscape-App **VRNetzerApp** ermöglicht sie den Export und das Senden von Netzwerken an den VRNetzer sowie den Empfang von Netzwerken vom VRNetzer. Hierdurch wird Nutzern des VRNetzer der Zugang zu hochwertigen Protein-Protein-Interaktionsnetzwerken aus der etablierten STRING Datenbank (Search Tool For Recurring Instances Of Neighbouring Genes) ermöglicht. **StringEx** erleichtert die Nutzung dieser wertvollen Daten für VRNetzer-Nutzer, indem es nicht nur einen neuen Importweg bietet, sondern auch eine angepasste Benutzeroberfläche für die Analyse von STRING-Netzwerken bereitstellt. Dabei können Nutzer auch auf neun kuratierte STRING-Interaktome biomedizinisch relevanter Organismen zurückgreifen die durch **StringEx** zu Verfügung gestellt werden.

Mit der zweiten Erweiterung namens **ProteinStructureFetch** wird die Vorbereitung von vorhergesagten Proteinstrukturen aus der AlphaFold Datenbank für eine Darstellung in der Virtual Reality erleichtert. In Zusammenarbeit mit ChimeraX ermöglicht diese Erweiterung die Untersuchung von Molekülstrukturen mit der VRNetzer-Plattform in 37 verschiedenen Darstellungen. So wird es möglich, die zugrunde liegenden komplexen dreidimensionalen Molekularstrukturen zu analysieren, die normalerweise in einem Protein-Protein-Interaktionsnetzwerk zu Knoten abstrahiert werden.

ABSTRACT

The VRNetzer is a network visualization platform designed to explore large-scale networks. Through virtual reality technology, users can analyze their networks in an immersive environment. However, currently available networks are limited, and preparing new networks for an immersive exploration in virtual reality can be challenging for less technically experienced users.

Here, I present a collection of platform enhancing tools and features which were developed in the course of this master's thesis. These include additions to the back- and frontend of the VRNetzer platform, as well as the addition of new content that enhances the VRNetzer's usability in the field of biomedical research.

First, a new extension based system enables easier development and sharing of custom content like new analytic features or user interface elements. Two such extensions were developed throughout this master's thesis, which showcase the potential of this system especially regarding interoperability and modularity for future development projects. Both of these extensions come with their own custom code, user interfaces and other resources. All features developed in the course of this project serve the purpose of establishing a substantial connection of the VRNetzer with tools and software widely used in biomedical research.

The first extension, called **StringEx**, bridges a gap between the VRNetzer platform and Cytoscape. **StringEx** is accompanied by the **VRNetzerApp**, a Cytoscape App that allows networks to be exported, sent to, or received from the VRNetzer. By providing access to quality-controlled protein-protein interaction (PPI) networks from the well-established STRING database (Search Tool For Recurring Instances Of Neighbouring Genes), **StringEx** facilitates the use of this valuable data by VRNetzer users. This extension not only introduces a new import route to the VRNetzer, but also provides a customized user interface tailored to the analysis of STRING networks. Furthermore, it also includes curated STRING interactomes of various organisms, relevant in biomedical research, which are available for exploration within the VRNetzer platform.

The second extension, **ProteinStructureFetch**, streamlines the process of preparing predicted protein structures from the AlphaFold database for use in the VRNetzer platform. With the help of ChimeraX, this extension enables the study of protein structures in 37 distinct representations, providing a deeper understanding of the complex three-dimensional molecular structures that are typically abstracted as nodes in a PPI network.

CONTENTS

ABSTRACT	III
LIST OF TABLES	V
LIST OF FIGURES	V
LIST OF ABBREVIATIONS AND ACRONYMS	VI
1 INTRODUCTION	1
1.1 BIOLOGICAL NETWORKS	2
1.2 NETWORK BIOLOGY TOOLS	3
1.3 THE VRNETZER	7
1.4 STRUCTURE OF THIS THESIS	14
2 EMPOWERING EXTENSIONS: OPTIMIZING VRNETZER'S CUSTOMIZABILITY	15
3 THE VRNETZERAPP PROVIDES NEW INPUT FOR THE VRNETZER	16
4 STRINGEX FACILITATES THE ANALYSIS OF STRING NETWORKS IN VR	18
5 THE UNDERLYING NATURE OF MOLECULAR NETWORKS	20
6 ANNOTATION HIGHLIGHTING IN VR	22
7 MATERIALS AND METHODS	24
7.1 IMPLEMENTATION OF THE VRNETZERAPP	24
7.2 VRNETZER BACKEND EXTENSION SYSTEM	24
7.3 IMPLEMENTATION OF STRINGEX	26
7.4 VRNETZ UPLOAD PROCESS	26
7.5 INTERACTOME CURATION	27
7.6 INTERACTOME ANNOTATION MAPPING	28
7.7 PROTEIN STRUCTURE FETCH EXTENSION	29
7.8 ANNOTATION HIGHLIGHTING	32
8 DISCUSSION	34
BIBLIOGRAPHY	36
APPENDIX	42
APPENDIX A	42
APPENDIX B	43
APPENDIX C	44
APPENDIX D	45
APPENDIX E	46

LIST OF TABLES

1	STRING Interactomes	19
A1	File size comparison of exported STRING networks	42
A2	External links	42

LIST OF FIGURES

1	StringApp Provides Access To Interaction Data From STRING DB In Cytoscape .	5
2	User interfaces of the VR module	9
3	Architecture of the new VRNetzer version	10
4	StringEx connects the VRNetzer with Cytoscape and STRING DB	16
5	StringEx provides access to interactomes of nine model organisms	18
6	The ProteinStructureFetch extension	20
7	Inspection of a protein structure in VR	21
8	Three modes to highlight node and link annotations	22
9	Highlighting of node and link annotations	23
10	Loading of extension tabs	25
11	Preprocessed protein structure of Mucin-4	29
12	Visualization of the transformation process	31
B1	Usage of Socket.IO within the VRNetzer ecosystem	43
B2	VRNetz file example	44

LIST OF ABBREVIATIONS AND ACRONYMS

3D	Three-dimensional
aa	Amino acids
API	Application programming interface
app	Application
BMP	Bitmap
CSS	Cascading Style Sheets
CSV	Comma-separated values
DB	Database
DOID	Disease Ontology Identifiers
GLB	Binary Graphics Language Transmission
GO	Gene Ontology
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifier
JS	JavaScript
JSON	JavaScript Object Notation
NDEx	Network Data Exchange
PDB	Protein Data Bank
PLY	Polygon
PNG	Portable Network Graphics
PPI	Protein-protein interaction
PSICQUIC	Proteomics Standard Initiative Common QUery InterfaCe
RCSB	Research Collaboratory for Structural Bioinformatics
REST	Representational state transfer
RGB	Red, green, blue
STRING	Search Tool For Recurring Instances Of Neighbouring Genes
UI	User interface
UMAP	Uniform Manifold Approximation and Projection
UniProtID	UniProt Protein Identifier
URL	Uniform Resource Locator
VR	Virtual reality
XML	Extensible Markup Language

1 INTRODUCTION

Since the analysis of the seven bridges of Königsberg in 1736 [1] by the Swiss mathematician Leonard Euler, networks have become a well-established framework to capture relationships between entities in complex systems. Networks serve as a backbone for organizing and structuring many aspects of our daily lives. Whether it is in computer systems, where they help us to understand and manage large amounts of data, or in cities, where public transportation networks provide a means of efficiently navigating and accessing different locations [2]. They consist of only two basic components: Nodes, which represent entities, and links, which mediate the relationships between them.

Networks serve as a critical tool for researchers to understand complex systems across multiple disciplines, including the field of biology. By providing structure and organization to cellular mechanisms, networks help us uncover the processes of life and gain a deeper understanding of biological systems. These insights, how cells maintain and regulate themselves, have crucial implications in fields like medicine and biotechnology, as a deeper understanding of biological systems can lead to the development of new treatments and therapies [3].

However, biological systems are not only complex, but are usually also very extensive. The human genome is estimated to consist of more than 20,000 protein-coding genes [4]. While it is straightforward to answer specific questions by manually examining representations of small-scale networks, it becomes increasingly difficult when dealing with large-scale datasets [5].

The VRNetzer [6] is a new ecosystem tailored to the exploration of such large-scale datasets. It provides a set of tools that allows users to explore and analyze networks in a virtual reality (VR) environment. In this master's thesis, I developed new features, tools, and interfaces which connect the VRNetzer ecosystem with other established biological tools and software. This development process revealed inconsistencies in the VRNetzer's customizability, which were addressed by a new extension system. This system streamlines the development of new functionalities and fosters community contribution and collaboration by making it easier to exchange new features with others. The following sections will provide detailed descriptions of the essential tools and concepts.

1.1 Biological networks

The field of network science has experienced significant growth over the past decade. It benefits from viewpoints and ideas from a community which is spread over a broad variety of scientific areas [2]. In the field of biology, networks are utilized to address a wide range of scientific questions. Whatever the system of interest, relationships between organisms in an ecosystem, the connections of nerve cells in our brain, or the collaboration of molecular components in a cell, networks provide a structured approach to capture, analyze and process these complex relationships [2].

The newly introduced features of the VRNetzer ecosystem are geared towards the analysis of molecular networks, with a focus on protein-protein interaction (PPI) networks. To provide context, I will summarize various types of molecular networks, which can capture both physical and functional relationships between molecular components, as outlined by Buphamalai et al. [3]. One speaks of a physical relationship when the interacting molecules exhibit a direct physical interaction. An example for such a relationship would be the binding of a ligand to a receptor. On the other hand, functional interactions describe any kind of biologically relevant relationship between molecules, such as the co-expression of genes or the co-localization of the proteins they encode.

One example for a molecular network are regulatory networks which cover the regulation processes of gene expression under specific circumstances. Metabolic networks cover all processes in which cells obtain energy and how this energy is used to produce the necessary building blocks of a cell. Co-expression networks mediate relationships between genes based on expression level correlation. Another example are gene networks which are used to model the interactions of genes and describe their relationships. For example, a gene network might show how one gene regulates the expression of another gene, or how different genes work together to control a biological process.

The vast majority of molecular processes in cells involve complex interactions between multiple proteins, rather than being carried out by a single protein in isolation. These interactions involve large collections of proteins that collaborate to synthesize biochemical molecules, transduce signals, or perform other cellular tasks. The physical organization of these protein interactions is determined by a complex interplay between the individual proteins involved, guided by electrostatic forces. PPI networks are used to model these interactions and describe the relationships between proteins. It has been reported that when PPIs are perturbed, for instance

as a result of a protein's inability to form the necessary connections with other proteins, this can lead to the development of a variety of diseases including diabetes, obesity, Crohn's disease, and cancer [7] as outlined by Buphamalai et al. [3].

The majority of PPI networks encompass not only direct physical interactions but also chemical and functional relationships. For example, a protein may participate in the same metabolic pathway as another protein or its gene expression may be controlled by another protein. Taken together, PPI networks covering the complete set of physical, chemical and functional interactions within a cell are commonly referred to as an interactome.

The analysis of molecular networks is a powerful tool to gain insight into biological processes, helping to answer numerous scientific questions. In the following section tools and software which allow the exploration of such biological networks and are used throughout this thesis will be presented.

1.2 Network biology tools

The choice of tools for working with biological networks will depend on factors such as technical proficiency, available resources, and the desired outcome. This section introduces three tools that are used throughout this thesis. The first two tools are designed for analyzing networks, one with a graphical user interface, one tool offering a programmatic approach. The third tool is a widely-used database (DB) of PPI data.

Network analysis via a graphical user interface

The most prominent software used in network biology is Cytoscape [8]. Cytoscape provides a graphical user interface to visualize, analyze, and explore biological networks in a user-friendly manner. This free and open-source software is a good start for beginners in network biology. It supports various network formats, allowing the import of network data from numerous sources. Its broad tool set can be furthermore extended by a wide range of applications (apps), which add more features such as additional layout algorithms and more. These apps can be distributed or installed through the integrated Cytoscape App Store.

However, the visualization capacity of Cytoscape is limited. Modern high-throughput technologies generate large amounts of data, and a programmatic approach is often best suited to handle them.

Programmatic Network Analysis

For programmatic network analysis, a versatile Python library called NetworkX [9] can be utilized. This library offers a broad range of tools to create, manipulate, and study networks. Approaching network data programmatically offers several advantages, such as the automation of complex analyses, which allows for repeated and reproducible application to similar datasets. Additionally, due to its widespread use in many scientific fields, NetworkX benefits from frequent updates and a rich collection of resources and tutorials. This facilitates the processing of extensive network data, enabling the identification of patterns and crucial structural components.

Public Databases

A good starting point for any network analysis is a public DB. Well-maintained and widely used databases for PPI data include BioGrid [10], IntAct [11], HIPPIE [12], and the Search Tool For Recurring Instances Of Neighbouring Genes (STRING) DB [13, 14].

Among these, STRING DB provides the most comprehensive collection of interaction data covering over 20 billion interactions, from more than 14,000 organisms [15]. While the first three DBs focus on experimental interaction data, STRING DB also considers predicted functional associations like literature co-occurrence. As a result, each interaction listed in STRING DB is provided with a confidence score for each of the seven distinct evidence channels which allows for filtering and prioritization (see Box 1).

The STRING website is a useful interface to access small networks, for example a specific group of proteins associated with a certain pathway. The stringApp [16] (Fig. 1) however, allows for accessing PPI data from STRING DB to directly import networks into Cytoscape. Networks fetched with this app come with additional annotations from associated databases covering tissue expression rates, disease-gene associations, subcellular localization, and more. In the course of this thesis, stringApp will be used as the main source for small-scale PPI networks.

A key advantage of utilizing networks is that the human eye can quickly comprehend complex relationships and patterns within them with just a single glance [5]. While this becomes increasingly difficult as the numbers of edges and nodes in a network increase [2], VR technology can help with the visualization of such large-scale data sets. In the upcoming section, the VRNetzer ecosystem, specifically designed to tackle these challenges, will be introduced.

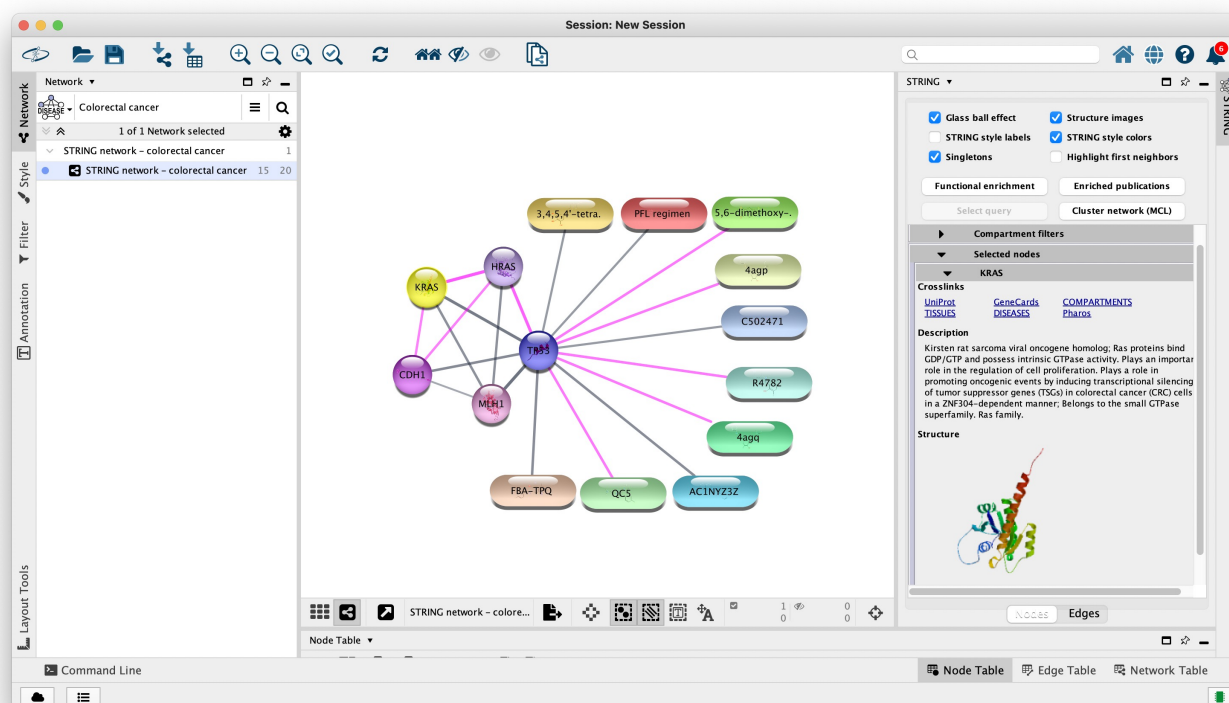


FIGURE 1 | STRINGApp PROVIDES ACCESS TO INTERACTION DATA FROM STRING DB IN CYTOSCAPE

Example result of a stringApp [16] disease query for the term “colorectal cancer“. The query resulted in a small network of the five highest scoring proteins encoded by the genes *TP53*, *KRAS*, *HRAS*, *DDX53* and *CTNNB1*. The network has been further expanded with STICH [17] compounds that are known to interact with these genes using the expand function of stringApp. All of them are connected to *TP53*, which is a well-known tumor suppressor gene that is commonly mutated in colorectal cancer [18]. Highlighted in purple are all interactions that are experimentally proven. On the right stringApp provides a panel with additional node information such as a description or the three-dimensional molecular structure of the protein. The maximum number of proteins was set to five and the used cutoff value for the confidence score was 0.4.

Box 1 | The STRING Evidence Channels[13, 14]

KNOWLEDGE

Experiments:	Considers PPI associations which were revealed through experiments and assays in the lab.
Database:	Considers PPI associations from other databases and therefore reflects the current state of knowledge.

GENOMIC CONTEXT

Neighborhood:	Considers the position of genes encoding the respective protein and their distance to other genes on the same chromosome. The closer the two genes are, the higher the respective score.
Fusion:	Is constructed by scanning all open reading frames of every genome to search for gene fusion events in which two previously separated genes are joined together forming a new gene. The score is depending on the homology of two genes, e.g. the closer the last common ancestor the higher the score.
Co-occurrence:	Considers pairs of genes that are frequently observed to occur together in the same genome.

OTHER

Co-expression:	Considers functional genomics measurements such as transcripts of proteins across a variety of conditions to find evidence of common expression regulation.
Textmining:	This channel is based on the frequency with which a pair of proteins is mentioned together in literature compared to the frequency of their individual mentions. This helps identify any functional associations between the proteins based on the context in which they are mentioned in the literature.
(Similarity:	This is not considered as an evidence channel, but highlights significant sequence similarities between two proteins.)

1.3 The VRNetzer

For almost three decades, VR has been used by scientists to make complex concepts and data tangible. In 1996, Bryson presented the capabilities of this paradigm, which was just emerging as a new way to visualize scientific data [19]. Since then, many scientists have made use of the immersive nature of VR to visualize, explore and present their data.

Over the past few years, the number of published papers on those applications has been particularly high [20]. In fact, VR, augmented reality, and mixed reality devices are becoming increasingly present in the consumer sector. Technological advances and the growing number of manufacturers have made these devices more affordable for individuals, academia, and other educational institutions.

One of the more recent applications of VR in science is the VRNetzer, an ecosystem developed by Pirch et al. [6], which enables the visualization and exploration of large-scale networks. As a proof of concept Pirch et al. demonstrated the capabilities of VRNetzer as a platform for visualizing and exploring genome-scale molecular networks with the aim of identifying genes linked to rare diseases through interactive analysis in VR. This is supported by state-of-the-art analysis methods such as a random walk with restart to pinpoint the broader neighborhood set of selected seed genes [6].

It is important to note that a new version of the VRNetzer was being developed while this thesis was being written. This means not only that the software architecture has been revisited, but also that only a limited set of analytical tools were available as most of them were not yet adapted to the new software architecture. However, the updated version introduced a new major feature, allowing to conduct multiplayer sessions in which network data can be explored, analyzed, and presented collaboratively. Additionally, a three-dimensional (3D) web-based previewer has been added, enabling users to preview VRNetzer projects in a conventional web browser.

These changes indicate that there have been some major changes to the software architecture of the VRNetzer. Throughout the duration of this master's thesis, all advancements were made within the context of this updated version of the VRNetzer. Accordingly, the following sections will be focused solely on this latest version.

User Interfaces

The VRNetzer user interface (UI) consists of two parts: the VR UI and the desktop UI. The VR module of the VRNetzer is built with the Unreal Engine 4, a game engine developed by Epic Games [21]. Its UI consists of two panels termed main panel and node panel. Both panels display a web browser which navigates to a defined Uniform Resource Locator (URL) (Fig. 2) hosted by the backend server of the VRNetzer. All functionalities, including project selection, node and link opacity adjustments, can be accessed through these webpages. To reduce the memory load caused by opening multiple web browsers in Unreal Engine, a tab system is implemented for organizing the web pages, providing access to multiple functionalities through a single web browser loading a single web page.

The main panel provides an interface for interacting with the network. This includes access to available projects, different network layouts, as well as the ability to adjust the size and opacity of nodes and links.

The node panel provides extensive information about the selected node, which may include various types of data depending on the type of network, such as molecular or geographical information. Both panels are also accessible in desktop mode, enabling changes to be made on the desktop that are synchronized with the VR session.

The execution of some tasks within the VRNetzer ecosystem is more suitable for a desktop environment. This includes the uploading of a new network project, e.g. uploading network data to the VRNetzer server, or the preview of a project in the new web-based 3D previewer. These tasks can be performed via the upload page and the preview page, respectively. To allow users to easily switch between different input formats and upload their data efficiently, without the need to navigate to separate pages for each format, the upload page is also organized in tabs. Although currently only one format is supported, the infrastructure is in place to accommodate multiple data formats in the future.

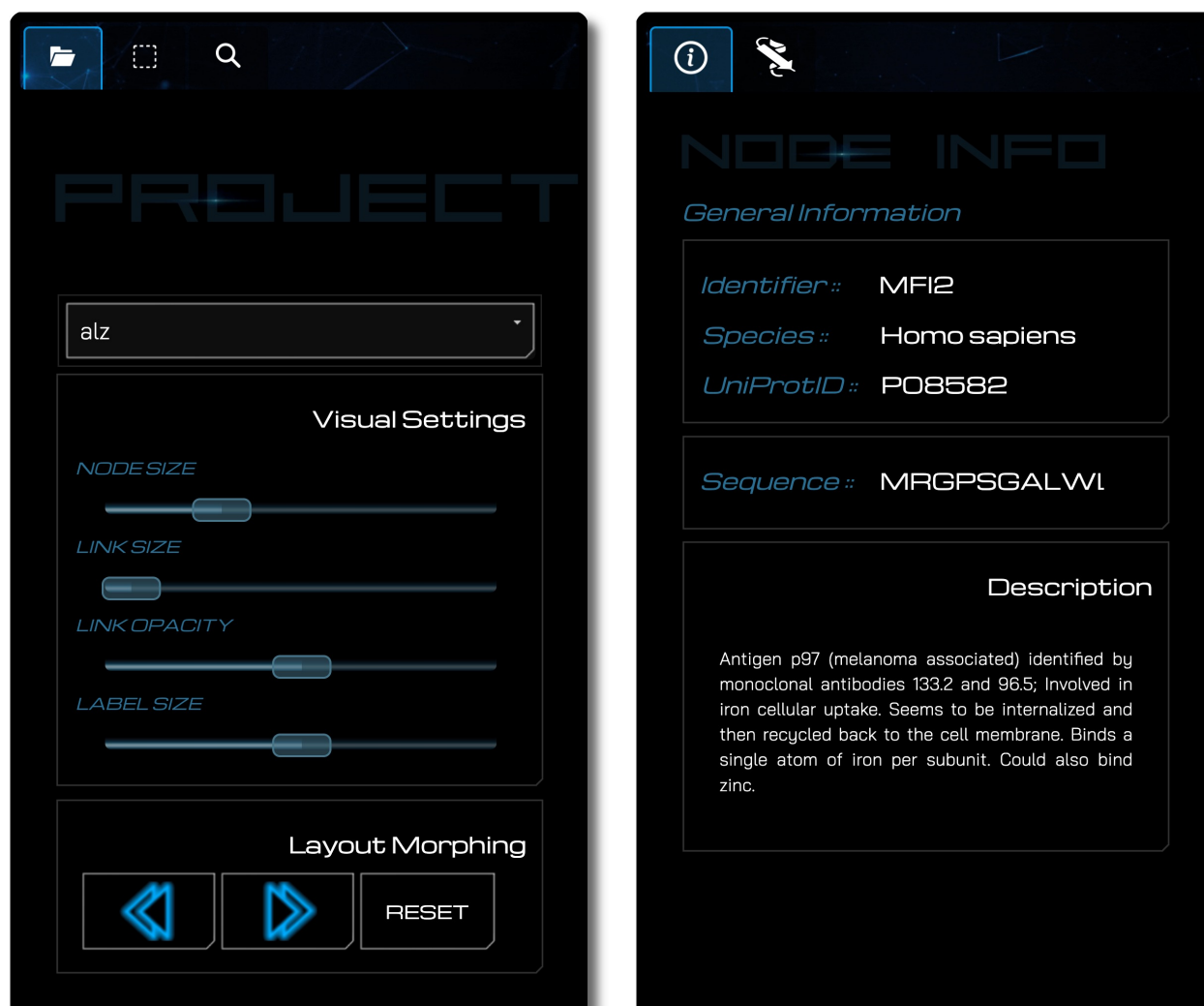


FIGURE 2 | USER INTERFACE OF THE VR MODULE

The UI of the VRNetzer [6] in VR consists of two panels running a web browser which navigates to a defined URL: the main panel (left) and the node panel (right). The main panel provides an interface for interacting with the network itself, including access to all available projects, different network layouts, as well as the ability to adjust the size and opacity of nodes and links. The node panel provides extensive information about the selected node and its content varies depending on the type of network. Both panels are organized in tabs to allow users to easily switch between different functionalities and to reduce the memory load caused by opening multiple web browsers in Unreal Engine.

Software Architecture

The key component of the VRNetzer is the VR module which now allows to conduct multiplayer sessions. However, these sessions require that each participant sees the same data and that they keep track of what changes have been made by other participants. This is achieved by connecting the VR engine, the web-based UI, and the data server using Socket.IO [22].

The biggest change to the software architecture of the new version of the VRNetzer is that the data is no longer stored in a MySQL DB, but instead all network data is now stored in JavaScript Object Notation (JSON) files and bitmaps (BMPs). This makes using the VRNetzer much easier as there is no need for a DB setup. As a result, the data server has been transformed into a simple Python Flask [23] web server that implements a representational state transfer (REST) [24] application programming interface (API) allowing to access data and perform actions via Hypertext Transfer Protocol (HTTP). The updated software architecture of VRNetzer is depicted in Figure 3. More technical details about the implementation of the VRNetzer can be found in Appendix E.

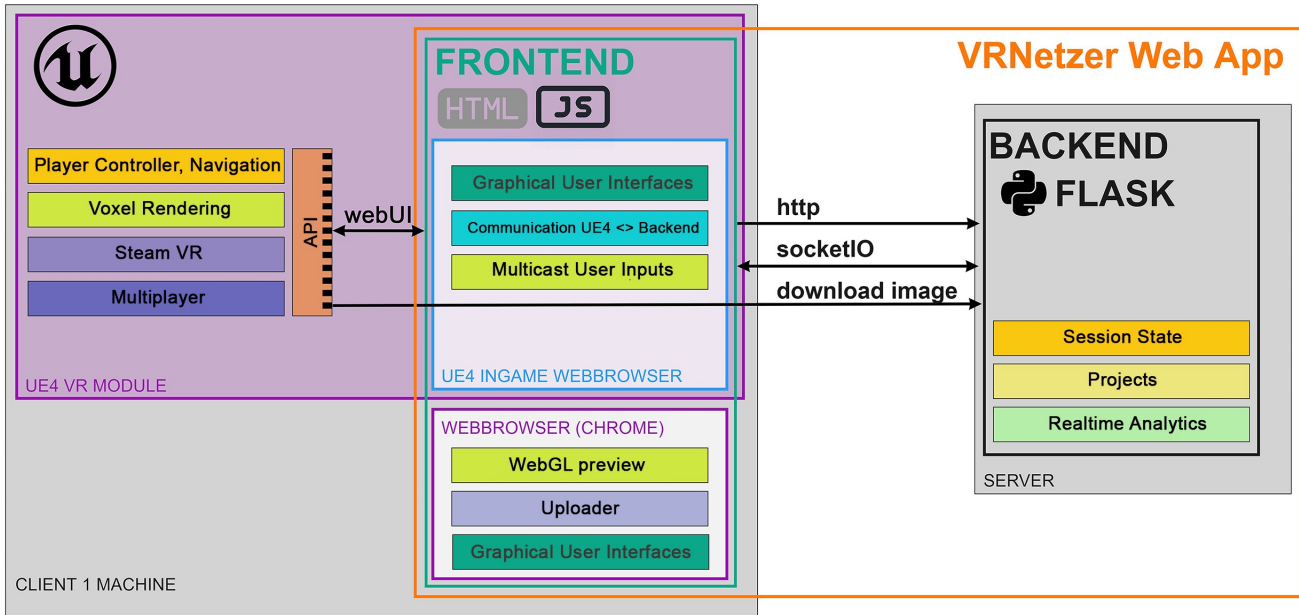


FIGURE 3 | ARCHITECTURE OF THE NEW VRNETZER VERSION

The new VRNetzer ecosystem consists of four modules illustrated in this figure adapted from [6]. These can be separated in client side modules (left) and server side modules (right). The VR module is powered by Unreal Engine 4 [21] and responsible for the visualization of network data in an immersive VR environment on the client side. The web based UI is presented by the integrated web browser of the VR module and provides an interface to the server. For tasks such as the preprocessing of data, which are more suited to be performed on a conventional computer screen, designated web based UIs are provided. Furthermore, a web based preview is now available, which allows to preview VRNetzer projects in a conventional web browser. All actions performed through the UI are handled by the backend module which consists of a Python Flask server [23]. This server keeps track of session information, such as the current selected project or layout and provides the necessary data. The communication between the Server, the VR module and the UI is handled by Socket.IO and the implemented REST API.

Uploading Data to VRNetzer

The incorporation of new data into the VRNetzer ecosystem involves four steps: (i) preparing the network data, (ii) calculating a 3D network layout and normalizing calculated coordinates, (iii) preparing the required data format, and (iv) uploading the data to the VRNetzer server.

Especially the first three steps require a certain level of expertise in the field of network analysis and data preparation. As described in Section 1.2, DBs are always a good start for any network analysis project. Once acquired, the data can be analyzed using graphical network analysis tools, such as the software Cytoscape [8] introduced in Section 1.2. Cytoscape offers a range of built-in layout algorithms that enable users to effectively visualize and analyze network data. These algorithms arrange the nodes and links of a network in a visually meaningful way, making it easier to identify patterns and relationships. However, the built-in algorithms are limited to two-dimensional layouts, and additional manual preparation is required to make the exported network data compatible with the VRNetzer.

A programmatic approach utilizing libraries such as NetworkX [9] for preparing network data is recommended for use with the VRNetzer platform. NetworkX provides a collection of 3D layout algorithms that can be used to calculate a 3D network layout. Subsequently, the calculated node coordinates have to be normalized to fit the 3D space of the VRNetzer, and the network has to be split into node and link data and exported as a comma-separated values (CSV) file in a specific format.

These CSV files do not provide any header information, thus it is essential to strictly follow a specific order of the columns to ensure that the data is accepted by the VRNetzer. A node CSV file contains eight columns: x-coordinate, y-coordinate, z-coordinate, red, green, blue, alpha, and an attribute column which includes the node label. The red, green and blue color channels define the color of the node and allow to highlight a certain nodes or a cluster of nodes. The alpha channel defines the brightness of a node which can be used to highlight a certain node attribute.

The attribute column in the CSV files does not have a defined length and can store as many attributes as needed. However, it is important to note that the first attribute is used as the node label and the second attribute contains an integer-based node identifier (ID). In cases where the network is a PPI network, the subsequent attributes can store additional annotations such as a protein's description or its UniProt Protein Identifier (UniProtID).

A link CSV file contains six columns: source, target and the four color channels red, green, blue and alpha. The source and target columns contain the node ID of the nodes that are connected by the link. The red, green, and blue columns are used to define the color of the link allowing it to reflect different kinds of interactions like physical or functional interactions in PPI networks. The alpha column determines the brightness of the link and can be used to represent various features, such as link weight.

As we can see, preparing network data for the VRNetzer can be challenging and requires attention to detail to avoid potential pitfalls. The last step of the process is to upload the data to the VRNetzer server. This is done by navigating to the upload page and selecting the prepared nodes and links CSV files. This creates a VRNetzer project which stores the network data in the form of JSON files and BMPs. The project is then available in the VRNetzer ecosystem and can be accessed via the UI.

The data encoding of a VRNetzer project is described in detail in the following paragraph, which explains how the platform is able to handle genome-scale networks.

Data encoding

The VRNetzer platform is able to handle large networks with up to 16,384 nodes and 262,144 links. To achieve this, Pirch et al. encode node coordinates, links, and their colors in BMPs. The advantage of storing the data in BMPs is that the data can be highly compressed which reduces the file size to a few kilobytes. BMPs store every used color only once and each pixel is represented by a single value which can be mapped to one of the stored colors [25]. For example if an image is black and white, each pixel can be represented by a single bit.

This becomes particularly relevant when it comes to the transmission of network data from the server to a VR client. The larger the files to transmit, the longer it takes to load the data which may become a problem when working with such large scale networks. Using BMPs, network data can be encoded by files that are no larger than a few kilobytes in size.

The spatial information of each node in the 3D space is encoded by a single pixel value in a BMP. Each of the red, green, blue (RGB) color channels encodes for one of the three spatial coordinates. Each color channel in a regular BMP can only store eight bits. As the coordinates are normalized between 0 and 1 and the maximal value of a color channel is 255, all three spatial coordinates are multiplied by 256^2 . To ensure a precise encoding of the spatial coordinates a trick is used to reduce roundoff errors. In fact, two BMPs are generated, whereby the first BMP encodes the rounded values of all spatial coordinates divided by 255 and the second BMP encodes all spatial coordinates modulo 255. Using this technique the spatial information of up to 16,384 nodes in 3D space can be transmitted in a precise and low-latency manner. This allows to prepare multiple node layouts for the same network, between which the VR engine can easily switch.

Similar to the nodes of the network, also its links are encoded by BMPs. Here it is also possible to create different link layouts. Each link contained in the layout is encoded by two pixels. The first pixel encodes the ID of the first node and the second pixel encodes the ID of the second node. As already mentioned, each of the three color channels a pixel holds can store eight bits. Therefore, each pixel can encode 24 bits which would allow to encode up to 16.7 million IDs, which is significantly more than the 16,384 nodes the VRNetzer can handle. Each ID is encoded by a single pixel value in the BMP and encoded by the following formula:

$$\begin{aligned} r &= x \bmod 128 \\ g &= \lfloor x \div 256 \rfloor \bmod 128 \\ b &= \lfloor x \div 256^2 \rfloor \\ x &= r + g \cdot 256 + b \cdot 256^2, \end{aligned}$$

where x is the ID and r , g , and b are the RGB color channels of the pixel. For example, the node ID 3259 can be encoded by the RGB color tuple (59,25,0).

When considering a network with 262,144 links, the corresponding BMP file with a resolution of 1024×512 pixel has an approximate file size of 1500 kilobytes. This file can be transmitted in approximately 163 milliseconds with the global average download speed of 78 megabits per second and a latency of 9 milliseconds [26].

Based on the information encoded in these BMPs corresponding voxels¹, representing nodes and links are rendered by the VR engine of every client. Additionally, the colors of nodes and links are stored in a Portable Network Graphics (PNG) BMP respectively. In this file format each pixel can hold 32 bits, allowing to encode four color channels red, green, blue, and the alpha channel.

Customizability

A prominent design feature of the VRNetzer platform is the fact that UI elements can be easily extended, as they consist of basic Hypertext Markup Language (HTML) and JavaScript (JS) web pages and therefore, require no knowledge of the VR engine [6]. Furthermore, the underlying Flask server allows a straightforward connection of new UI elements with custom analytic tools implemented in Python. This provides developers with the capability to fully customize the VRNetzer ecosystem to their desired use case.

¹Voxels are the homologue of a pixel in 3D computer graphics.

1.4 Structure of this thesis

The following sections will present the different enhancements to the VRNetzer ecosystem which were introduced by this thesis. In Section 2, a new extension-based system will be presented, which aims to address several limitations to current customizability. The section is followed by the introduction of two extensions utilizing this system to connect the VRNetzer with software and tools widely used in biomedical research.

The first extension, called **StringEx**, is presented in Section 3 and 4. It is accommodated by a new Cytoscape app called **VRNetzerApp**, which connects the VRNetzer ecosystem with Cytoscape and the STRING database. This extension enables users to easily create VRNetzer projects from STRING networks within Cytoscape, streamlining the extensive data preparation process required for visualization in VR.

The second extension facilitates the visualization of protein structures fetched from AlphaFold DB within the VRNetzer platform. This extension, named **ProteinStructureFetch** is presented in Section 5.

To provide some basic analytic functionalities to the current VRNetzer version, a third extension has been developed which is introduced in Section 6.

Section 7 outlines the technical details behind these enhancements. Finally, Section 8 presents an overview of the recent additions to the VRNetzer ecosystem, evaluates their effectiveness, and discusses potential future improvements.

2 EMPOWERING EXTENSIONS: OPTIMIZING VRNETZER'S CUSTOMIZABILITY

This master's thesis project was the first attempt to expand the feature set of the VRNetzer serving as a test case to assess its customizability. As a result, it uncovered significant limitations. While it is true that many enhancements to the functionality and the UI of the VRNetzer are possible without any knowledge of the VR engine, exchanging and implementing new features is prone to errors: Thus, new features are implemented by directly editing key components of the back- and frontend. By integrating custom code in this fashion, it becomes difficult to manage and maintain it over time, especially for those who lack a deep understanding of the code structure. Additionally, it can lead to code pollution and make future upgrades more challenging.

To address these issues, I designed and implemented an extension-based system to simplify the exchange of features and streamline their development. The aim of this system is to separate custom features from key components of the VRNetzer, while still maintaining full customizability.

This is achieved by introducing VRNetzer extensions, which are small packages containing source code and other resources such as images. They enable the integration of new UI elements and functionalities into the VRNetzer ecosystem. In essence, an extension is a Flask blueprint, which will be automatically registered by the VRNetzer backend eliminating the need to modify essential components of the backend. This allows extensions to further extend the REST API of the VRNetzer.

Furthermore, to allow extensions to send and receive Socket.IO messages, the Flask blueprints class has been extended based on previous work by [27] and [28], such that every extension can access the Socket.IO instance of the main VRNetzer Flask web app. This allows developers to directly integrate new features within the context of the VRNetzer backend. And since extensions are independent from the VRNetzer installation, they can be shared easily with others in a plug-and-play fashion.

All of these features and improvements not only streamline the development process and improve the overall user experience, but they also make it easier for a wider community of both users and developers to contribute to the platform by developing and exchanging custom features.

Results

A Git repository is provided which serves as a comprehensive guide for developers to learn about the tools and principles involved in the creation of an VRNetzer extension. The repository provides an example extension that demonstrates all available functionalities and features, which can be used as a starting point for the development of any custom feature. The link to this repository is provided in Table A2.

3 THE VRNETZERAPP PROVIDES NEW INPUT FOR THE VRNETZER

The **VRNetzerApp** is a Cytoscape app for exporting and sending network data to the VRNetzer. This allows for a seamless, immediate transition to an analysis in a running VR session. An exported network is stored in a JSON-based file format called **VRNetz**.

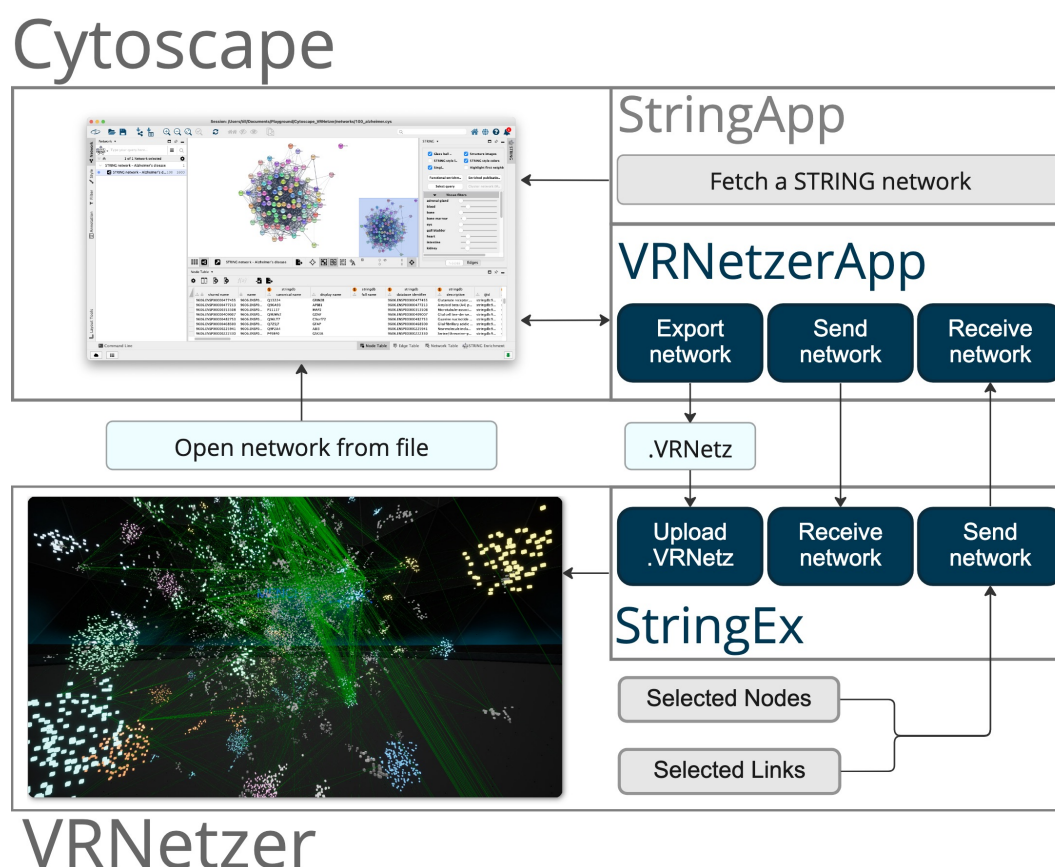


FIGURE 4 | STRINGEX CONNECTS THE VRNETZER WITH CYTOSCAPE AND STRING DB

StringEx provides a new workflow to bring networks to the VRNetzer. Networks can be exported from Cytoscape as a VRNetz file using the VRNetzerApp. Alternatively the VRNetzerApp can be used to send a network directly to a running VR session utilizing a new REST API route provided by StringEx. Together with the stringApp [16], this workflow enables access to PPI networks sourced from the STRING DB [13, 14]. To further extend the interoperability between the VRNetzer and Cytoscape, StringEx provides a functionality to send a sub-network to Cytoscape for further analyses using the tools provided by Cytoscape.

This file format is intended to store node, link, and network data in a single, lightweight file that can be easily uploaded to the VRNetzer to create a new project.

Unlike the existing upload route, users do not need to prepare their network data in any additional way, but can directly upload this file using the new upload route provided by the **StringEx** (Fig. 4) extension which will be further introduced in Section 4. This also includes the calculation of a 3D network layout as users can select one of eight layout algorithms provided by the two Python packages cartoGRAPHS [29] and NetworkX [9] during the upload process.

Furthermore, **VRNetzerApp** is designed with stringApp [16] in mind. Thus, STRING specific annotations and retrieved enrichments are also considered while exporting or sending. Moreover, in comparison to other file formats, **VRNetz** files store STRING networks in more than three fold smaller files (Tab. A1), thus making it easier to handle and transfer them.

The **StringEx** extension developed alongside **VRNetzerApp** allows for easy transfer of networks from Cytoscape to the VRNetzer platform without requiring additional data preparation. This not only enables access to STRING networks, but also other networks available through Network Data Exchange (NDEX) [30] or the Proteomics Standard Initiative Common QUery InterfaCe (PSICQUIC) [31] web services, which can be natively queried through Cytoscape. Therefore, all users of Cytoscape, STRING and the VRNetzer can benefit from this newly established interoperability. Cytoscape and STRING users are provided with an easy-to-use workflow to explore their networks in a fully interactive VR environment allowing for new perspectives and enabling the analysis of even larger networks in real-time. The availability of quality-controlled PPI networks sourced from the STRING DB provides VRNetzer users with a valuable resource for investigating complex relationships in biological systems.

The **VRNetzerApp** and **StringEx** are available as a pre-release in a Git repository and the **StringEx** extension can be added to any VRNetzer installation as a Git submodule (see Tab. A2).

4 STRINGEX FACILITATES THE ANALYSIS OF STRING NETWORKS IN VR

The newly established connection between the VRNetzer and Cytoscape is not unidirectional, but rather bidirectional (Fig. 4). StringEx enables the VRNetzer to transmit network data to Cytoscape utilizing Cytoscape's CyREST API and the p4cytoscape Python package [32]. This feature enables users to export a sub-set of their large-scale VRNetzer project to a running instance of Cytoscape, thereby providing access to its extensive and powerful tool set for analysis.

Beyond the newly established connection, the **StringEx** extension provides additional tools and features tailored to the analysis of STRING networks. One of the main characteristics of STRING networks are the multiple association evidence channels (see Box 1). StringEx provides a customized UI tailored to the analysis of STRING networks, allowing for an effortless transition between these evidence channels not only in VR, but also in the new web-based preview of the VRNetzer.

Furthermore, STRING networks fetched with stringApp come with a vast range of node annotations, which are made available and displayed on the node panel of the VRNetzer UI. To allow for exploration of larger STRING networks beyond the size limit of 2,000 nodes that can be directly retrieved with stringApp, **StringEx** provides access to a human STRING interactome as well as those of eight model organisms used in biomedical research (Fig. 5, Tab. 1). This allows users to fully utilize the visualization capacity of the VRNetzer and to explore STRING networks in an immersive environment. These interactomes provide access to a collection of annotated enrichment terms retrieved from the STRING DB originating from various sources, such as the Gene Ontology (GO) [33, 34], the DISEASES web resource [35], and others.

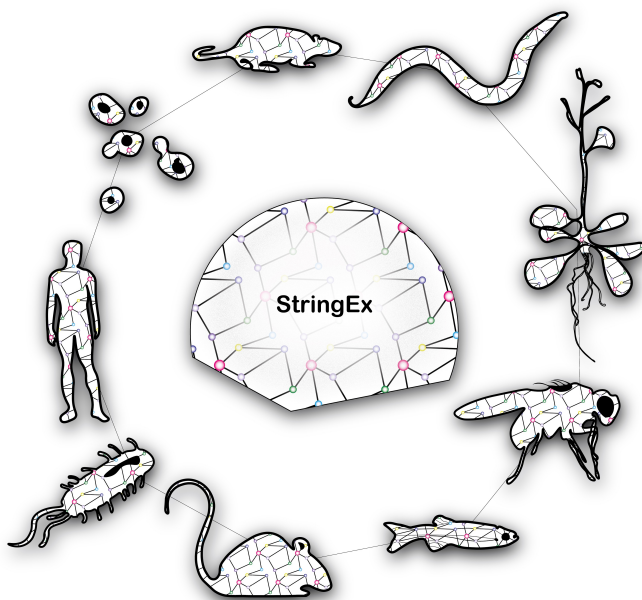


FIGURE 5 | STRINGEX PROVIDES ACCESS TO INTERACTOMES OF NINE MODEL ORGANISMS

StringEx provides interactomes for nine organisms: *H. sapiens*, *S. cerevisiae*, *R. norvegicus*, *C. elegans*, *A. thaliana*, *D. melanogaster*, *D. rerio*, *M. musculus*, and *E. coli* 536. All of them are annotated with various functional associations and provide a broad range of network layouts.

Moreover, these interactomes can be used to map small-scale STRING networks retrieved with stringApp onto them, extending the interactomes with additional annotations and enabling a more in-depth examination of a specific subset of actors in the context of the interactome.

TABLE 1 | STRING INTERACTOMES

Organism	# Nodes	# Links	# layouts		
			force-directed	structural	functional
<i>Homo sapiens</i>	13,485	262,144	1	2	8
<i>Mus musculus</i>	12,285	262,144	1	2	7
<i>Danio rerio</i>	12,720	262,144	1	2	6
<i>Rattus norvegicus</i>	11,269	262,144	1	2	7
<i>Caenorhabditis elegans</i>	10,546	262,144	1	2	5
<i>Drosophila melanogaster</i>	9,338	262,144	1	2	6
<i>Arabidopsis thaliana</i>	7,706	262,144	1	2	5
<i>Saccharomyces cerevisiae</i>	5,790	262,144	1	2	6
<i>Escheria coli</i>	4,594	262,144	1	2	4

Listing of the nine curated STRING interactomes with their respective number of nodes, links and prepared layouts. The list is sorted based on the number of nodes. All interactomes have an equal number of links, which is determined by the maximum number of possible links that can be handled by the VRNetzer. The respective link list is first sorted based on the experimental association evidence channel and subsequently lined up according to the combined STRING DB score. Following this, the list is trimmed after the 262,144th element. From this filtered link list the node list is generated. For every interactome a force-directed layout has been calculated using the Fruchterman-Reingold force-directed algorithm implemented in the NetworkX Python package [9]. Additionally two structural layouts have been calculated for each interactome using (i) the global and (ii) local UMAP algorithm implemented in the cartoGRAPHS Python package [29]. For each interactome at least five functional layouts have been prepared. These layouts highlight tissue expression [36, 37], association to biological processes, molecular functions [33, 34], subcellular localization [38], and biological pathways [39]. The human interactome comes with an additional layout highlighting disease-gene associations [35].

5 THE UNDERLYING NATURE OF MOLECULAR NETWORKS

Protein structures are one of the latest additions to the VRNetzer introduced by us [40] as part of the software development project course¹. To provide an automated way to visualize them for any node in a PPI network, **ProteinStructureFetch** is presented here. It establishes a connection between the VRNetzer ecosystem AlphaFold DB [42, 43] and ChimeraX [44] to allow for the integration of high-quality predicted protein structures for a wide range of organisms in VR (Fig. 6).

In molecular networks, molecular components are often abstracted to a node. This extension allows to reveal the underlying molecular structure of proteins up to a size of 2,700 amino acids (aa), when requested. The protein structures are obtained from the AlphaFold DB and processed with ChimeraX to make them suitable for visualization in virtual reality (Fig. 7). By utilizing ChimeraX's diverse range of tools, users can choose from 37 different visualization styles to emphasize various molecular characteristics of the structure they are analyzing, such as the electrostatic potential.

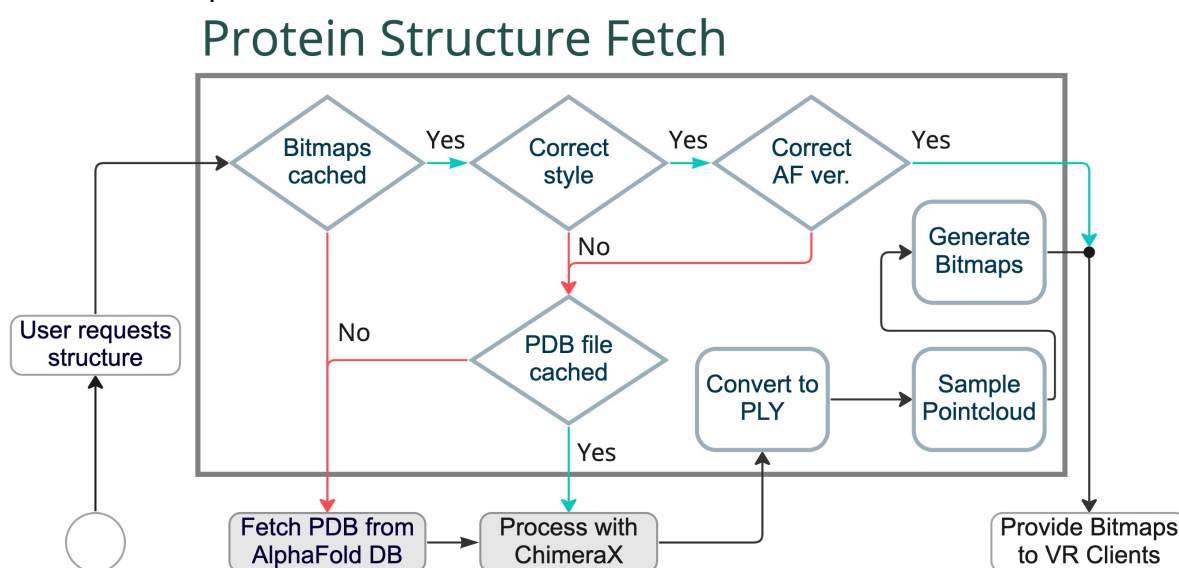


FIGURE 6 | THE PROTEINSTRUCTUREFETCH EXTENSION

The ProteinStructureFetch (PSF) extension follows a workflow that begins by a user requesting the 3D molecular structure of a protein. (Bottom) The respective PDB file is fetched from the AlphaFold DB [42, 43] and opened with ChimeraX [44]. A script is executed to apply the desired visualization style and export the structure as a GLB 3D object file. After converting this file to a PLY file, a point cloud is sampled, resulting in a collection of points with color and coordinates. This information is encoded to BMPs which can be distributed to the requesting VR clients. The VR module of the VRNetzer uses these BMPs to render a point cloud representing the protein structure in VR. (Top) If the requested structure has been already processed earlier with the desired visualization style and the correct AlphaFold version, the cached BMPs are directly distributed to the requesting VR clients.

¹Course BIOINF05 of the master's programme in bioinformatics, see [41].

The extension is configurable, allowing users to have full control over settings such as the amount of cached structures (i.e., previously processed structures that are stored for faster retrieval), the version of the AlphaFold DB, and the style of visualization used.

ProteinStructureFetch leverages the functionalities provided by the Python package **vrprot** (see Tab. A2) to retrieve protein structures from the AlphaFold DB and to process them with ChimeraX. In the course of this master's thesis, I continued with the development of this package and enhanced its functionality and performance.

As a result, the **vrprot** package now enables the processing of bulk downloads from AlphaFold DB [42, 43], which includes structures exceeding 2,700 aa. This capability enables the preparation of all organism structures for analysis in an interactive VR environment.

Independent of the **ProteinStructureFetch** extension, a package containing the full human proteome in two distinct representations prepared with **vrprot** is provided here (see Tab. A2). This allows for an immediate inspection of human protein structures in VR.

ProteinStructureFetch is provided as a Git repository which can be added as a Git submodule to any existing VRNetzer installation (see Tab. A2).

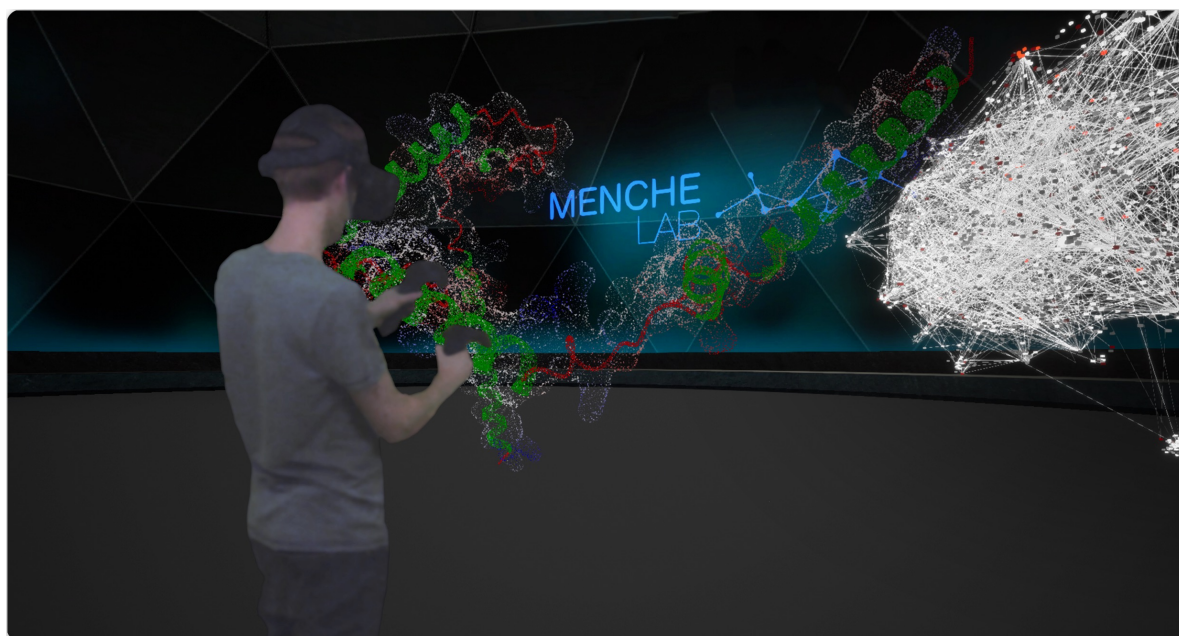


FIGURE 7 | INSPECTION OF A PROTEIN STRUCTURE IN VR

A visualization of the analysis of the 3D molecular structure of a protein. Shown are two visualizations which are layered on top of each other. The first is a cartoon representation of the protein's secondary structures, with the loop regions being colored in red, the helices in green, and the β -sheets in blue. The second is a molecular surface representation highlighting the electrostatic potential of the protein surface (red negative, white neutral, blue positive electrostatic potential).

6 ANNOTATION HIGHLIGHTING IN VR

Given the broad range of annotations that come with STRING networks, highlighting them within the network is crucial to enable a more targeted analysis. These functions were already available in the previous version of the VRNetzer, but have not yet been transferred to the new version. To address this issue, I introduced two new panels, allowing users to select a set of nodes and links based on their annotations. These features are implemented as an extension utilizing the extension system introduced in Section 2. They are intended to be included with every VRNetzer installation as a core extension.

The current node selection method is limited to physical pointing and selection in VR or using the search panel to query node annotations. However, this method does not allow filtering or highlighting of links based on their annotations. To address this limitation, the new utility extension provides a new UI that enables selection and filtering of nodes and/or links based on annotation values. Numeric annotations can be filtered by values above, below, or equal to a certain threshold, while categorical annotations can be filtered by specific values.

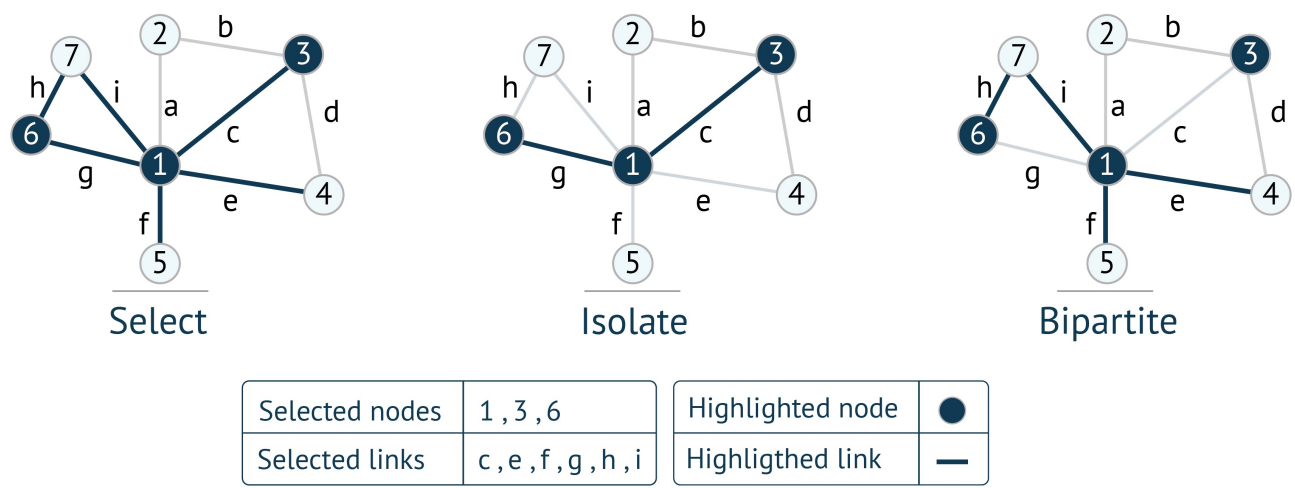


FIGURE 8 | THREE MODES TO HIGHLIGHT NODE AND LINK ANNOTATIONS
Illustration of the three highlighting modes. The *select* mode highlights all nodes and the set of selected links (c, e, f, g, h, i) that are connected to a selected node (1, 3, 6). In the *isolate* mode only the set of selected links between the selected nodes are highlighted. To only highlight the set of selected links that connects the selected nodes with unselected nodes, the *bipartite* mode can be used.

Selected nodes and links can then be highlighted using one of three highlighting methods: *selection*, *isolating*, and *bipartite* (Fig. 8). While the *selection* method highlights all selected nodes and links (Fig. 9), the other two only highlight a certain set of links. The *isolating* mode highlights links which connect selected nodes. The *bipartite* method, highlights links which connect selected nodes with nodes that are not selected.

An applied highlighting can be stored, enabling users to revisit the highlighted network at a later state. This feature is particularly useful for comparing multiple networks with different highlighting methods.

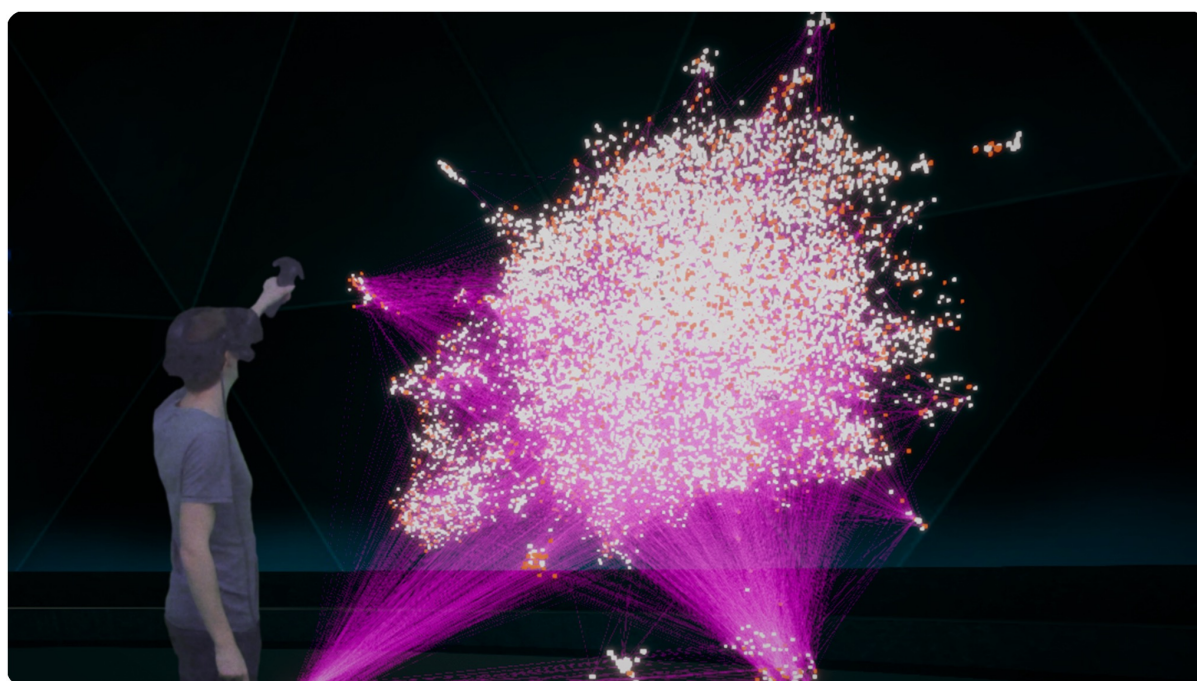


FIGURE 9 | HIGHLIGHTING OF NODE AND LINK ANNOTATIONS

The representation of the human STRING interactome displays a selection of links and nodes, with a specific set highlighted. The set of highlighted nodes, colored in red, consist of 3,376 proteins that are localized in the nucleus, annotated with the gene ontology term GO:0005634 and the qualifier *located in*. The links were filtered based on the *stringdb experiments score* with a threshold of 0.8, resulting in a subset of 72,754 links. The highlighting used was the *selection* highlight.

7 MATERIALS AND METHODS

7.1 Implementation of the VRNetzerApp

The **VRNetzerApp** is implemented in Java under the guidance of the Cytoscape App Ladder [45] and utilization of the Cytoscape 3.9 app API. The main function of the app is the export of networks in the VRNetz format. The **VRNetzerApp** adds a new export method to Cytoscape which allows users to extract node, link and network information into a single JSON-based **VRNetz** file. Furthermore, it enables to send networks to a running VRNetzer session via a REST API route provided by **StringEx**. This app has been developed with stringApp [16] in mind. Therefore, certain node columns of STRING networks retrieved with stringApp are treated differently. Thus, some node columns are skipped because they do not provide any value outside of the Cytoscape software and are therefore redundant. Since stringApp allows the retrieval of functional and publication enrichments, these are also considered during the export.

7.2 VRNetzer backend extension system

The plug-and-play based extension system for the VRNetzer backend is built around Flask [23] blueprints. These blueprints allow to extend any Flask app in a modular way. Similar to the core Flask app, a blueprint is able to define routes and the underlying code which is executed if the respective URL is requested. Furthermore, a blueprint can define additional resource directories to bring their own HTML, Cascading Style Sheets (CSS), JS files, or other resources with them. On boot of the VRNetzer backend a loading script is executed which registers all extensions located in the extension directory. This is enabled through the use of the “importlib” library which imports the respective Python module that contains the Flask blueprint on runtime and registers it to the main app.

To empower the extensions with Socket.IO, the Python class IOBlueprint has been adapted based on previous work [27] and [28]. This class extends the Flask blueprint class with the Socket.IO instance of the main app, enabling the emitting and receiving of Socket.IO events. By default, extensions emit and receive Socket.IO events in their own namespace to keep the communication separated. However, it is also possible for extensions to communicate with each other or the main app by defining the appropriate namespace for message emission and receipt.

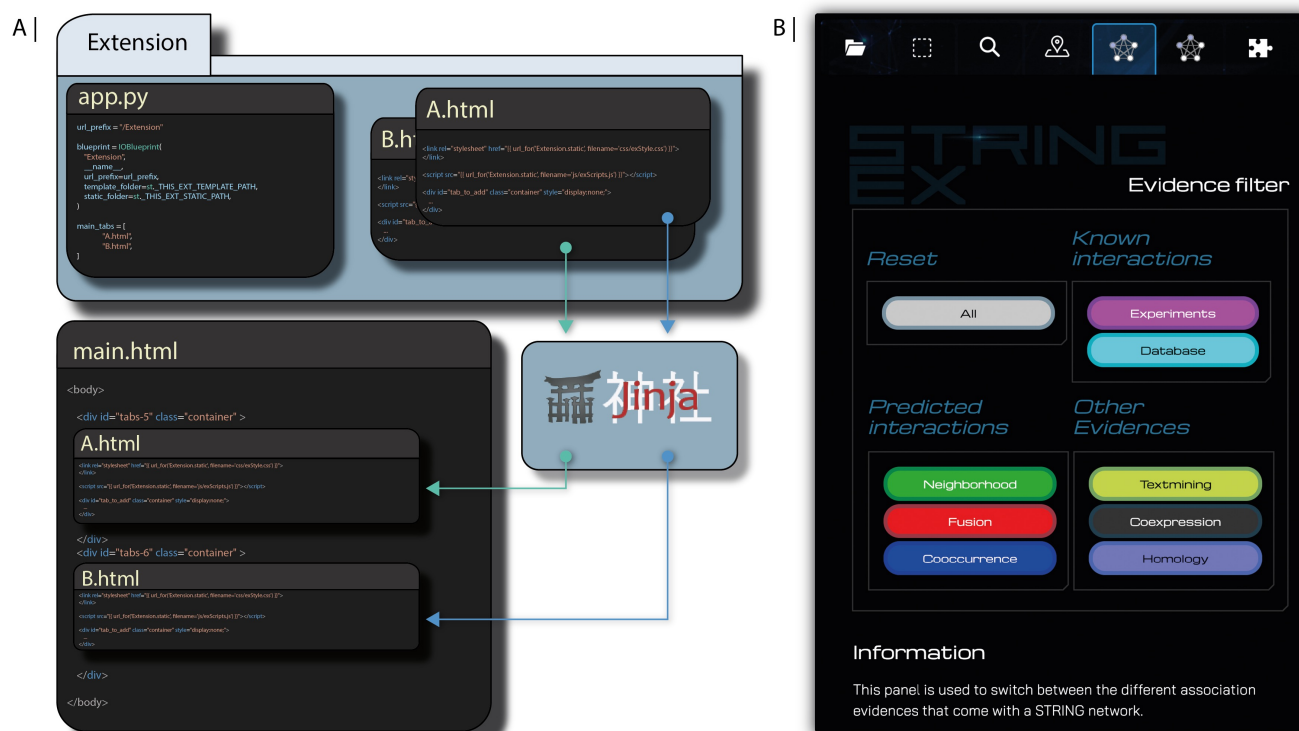


FIGURE 10 | LOADING OF EXTENSION TABS

By utilizing the Jinja2 template engine, custom UI elements provided by an extension are dynamically integrated into the existing UI panels of the VRNetzer. Here, the main panel is extended with two new tabs provided by the extension. **A |** The HTML code from the extension is integrated using the Jinja2 [46] template engine. Then, a JS function is executed to add a new tab and position the integrated HTML code within it. **B |** As a result, the main panel now displays the extended tabs, providing the user access to the new functionalities added by the extension. The shown example is an additional tab provided by the **StringEx** extension, which allows the user to easily switch between the link association evidence channels of a STRING network.

To allow the loading of new tabs to existing panels of the VRNetzer UI, these panels are extended with a JS which assigns a tab to the content provided by the extension. To add the content to the respective HTML template in the first place Jinja2 [46] is used. This tab loading process is depicted in Figure 10.

7.3 Implementation of StringEx

StringEx was implemented in Python utilizing the newly established VRNetzer backend extension system. It enables several functions to bring Cytoscape networks, and STRING networks in particular, into the VRNetzer platform. These functions include: (i) Uploading VRNetz network files, including the calculation of a network layout using layout algorithms from cartoGRAPHS [29] and NetworkX [9]. (ii) Mapping of VRNetz network files on preprocessed STRING interactomes. (iii) Representing STRING specific features on the VRNetzer platform. (iv) Sending a subset of a VRNetzer project to a running Cytoscape instance.

Since the VRNetzer is not designed for multigraphs – graphs in which two nodes can be connected with each other by more than one link – the seven evidence channels of a STRING network are divided into seven link layouts (see Box 1). The similarity channel is also considered as the eighth link layout even though it is not defined as a separate evidence channel by the STRING DB. The opacity of a link in these layout is scaled with the respective score of the evidence channel. Additionally, a ninth link layout is prepared that considers all evidence channels. The opacity of a link in this layout is scaled with the highest score over all evidence channels.

7.4 VRNetz Upload Process

Based on the integrated upload process of the VRNetzer, **StringEx** implements a new process which allows the upload of **VRNetz** files. During the upload process all annotations that are consolidated in a **VRNetz** file – especially if it covers a STRING network – are extracted and stored in a way that can be accessed by the VRNetzer.

The designated uploader UI offers the choice for one of the implemented layout algorithms and its parameters. It also allows to specify a project name and layout name to enable the assignment of multiple node layouts to the same project.

The submitted **VRNetz** file is subsequently processed and the respective node layout is calculated. Following this, all layout positions are transformed into the coordinate space of the VRNetzer, i.e., they must be within the interval $[0, 1]$.

To achieve this, the coordinates of every point p_j are first shifted to the positive space

$$p_{j,d}^+ = p_{j,d} + \left\lfloor \min_{p_{i,d} \forall i \in [1,n]} \right\rfloor ,$$

where p is a set containing all n points, and d is the respective spatial dimension (x -coordinate, y -coordinate, z -coordinate). Subsequently, the coordinates of every shifted point p_j^+ are normalized by the maximum value of all shifted points so that all points lie in the interval $[0, 1]$:

$$\bar{p}_{j,d} = \frac{p_{j,d}^+}{\max_{p_{i,d}^+ \forall i \in [1,n]}} ,$$

where p^+ is the set of all n points after they have been shifted to positive space. After the calculation of the layout, BMPs reflecting the node position are generated. The transformation of color values into two separate BMPs is performed as outlined in Section 1.3.

7.5 Interactome Curation

The interactomes provided by StringEx are PPI networks imported from the current STRING DB v11.5 [13, 14]. The VRNetzer platform has the capability to display a maximum of 262,144 links within a single project, however, it should be noted that the STRING DB has a much larger number of links available for the specific organisms. For that reason the respective link lists are first sorted based on the experimental association evidence channel and subsequently lined up according to their combined score. Following this, the list is trimmed after the 262,144th element. From this link list, the respective set of nodes they cover is extracted including their attributes such as gene name and descriptions provided by the STRING DB. Using the node list and all links for which the experimental score is larger than a defined threshold, a graph layout is generated and calculated using layout algorithms provided by NetworkX [9] and cartoGRAPHS [29].

To construct functional layouts, the enrichment terms data set provided by STRING DB is used. This data set consolidates all annotation terms which are used in a STRING DB enrichment analysis. For each category provided with this data set, such as “biological process“, “cellular component“ or “disease-gene association“, all respective annotation terms are extracted. Subsequently, each node is checked for annotations with any of the respective terms resulting in a feature matrix of $N \times M$ where N is the number of nodes and M is the number of annotation terms in the respective category. If a node is annotated with a term, the respective entry in the feature matrix is set to 1, otherwise it is set to 0. In a later step this feature

matrix is provided to catoGRAPHS [29] functional UMAP (Uniform Manifold Approximation and Projection) layout algorithm allowing to spatially cluster nodes based on their similar functional associations. On the resulting layout a hierarchical density based clustering is performed using the HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) algorithm to collect the respective node clusters. These clusters are color-coded to highlight the functional relationships among the proteins and labeled with an appropriate term that describes the cluster. The label is obtained by retrieving functional enrichment using the respective API method provided by STRING DB [13, 14]. The description of the enrichment with the lowest p-value is used for each cluster. Although the current version of the VRNetzer does not support the display of cluster labels, the obtained labels are already prepared for future updates where this feature will be enabled.

After the layout calculation, the respective link layouts representing the different STRING evidence channels are prepared as described in section 7.3.

StringEx provides a Python script which enables the reconstruction of all interactomes from scratch and allows to update the interactomes with the latest STRING DB version.

7.6 Interactome annotation mapping

The nodes in a small-scale STRING network are mapped to a prepared interactome using their respective identifiers (e.g., UniProtID), with the aid of the ID Mapping tool [47] when necessary. Similarly, links are mapped based on their source and sink. The mapped nodes and links update the corresponding node or link in the interactome, providing additional annotations. Mapped nodes are highlighted in the interactome by coloring them with the same color they had in Cytoscape and adjusting their opacity based on their size in Cytoscape. Unmapped nodes are assigned a defined color to make them distinguishable.

7.7 Protein Structure Fetch Extension

7.7.1 Preparation of 3D protein structures

As outlined in Section 1.3, large-scale networks are encoded as lightweight BMPs to enable easy distribution across multiple VR sessions. Similarly, protein structures are encoded as point clouds to facilitate quick and easy distribution between session members. The use of this representation for protein structures allows for less performance requirements and makes it suitable for visualization within the VRNetzer platform as a “network” without links. For each protein structure, three BMPs are generated which reflect the position of every point in the point cloud and its respective color.

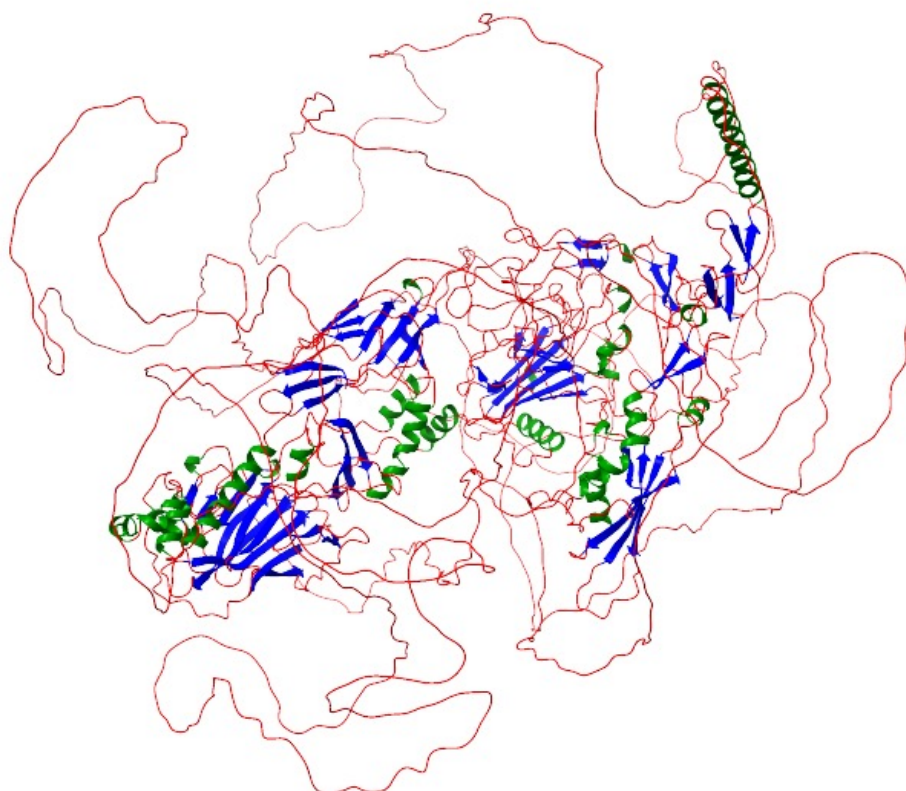


FIGURE 11 | PROCESSED MODEL OF THE PREDICTED PROTEIN STRUCTURE OF THE MUCIN-4

The structure has a length of 2169 aa (UniProtID Q99102) [42, 43] and is one of the largest structures which can be fetched directly from the AlphaFold DB by the method described in 7.7.1. In this preprocessed model we colored the secondary structures: Loop regions are colored in red, helices green, and β -sheets blue [40].

The process to generate these BMPs consists of six steps. (i) First, the protein structure is fetched as a Protein Data Bank (PDB) file from a public database. Here, the AlphaFold DB [42, 43] serves as the main resource. Using an HTTP GET request, the protein structure can be fetched directly from the AlphaFold DB. With this method, all Protein structures up to a size of 2,700 aa can be fetched. The method for larger structures is described in Section 7.7.2.

(ii) The protein structures are pre-processed using ChimeraX [44]. This involves tasks such as color highlighting and changing the visualization style. By default, all secondary structures in ChimeraX's cartoon representation are colored during pre-processing (Fig. 11). To automate this process, I implemented a Python script which is executed by ChimeraX's built-in Python interpreter.

Then, in step (iii) the processed structure is exported as a Binary Graphics Language Transmission (GLB) file and subsequently converted to a Polygon (PLY) file. After a second converting step, all structures are scaled into a $1 \times 1 \times 1$ cube. The necessary scaling factor can later be accessed to represent the structures in the correct relative scale to each other. After a transformation step (iv) to guarantee that all point coordinates lie within the interval $[0, 1]$, a point cloud is created (v) by sampling the 3D structure with 65,536 points.

To maximize numeric precision and avoid floating point rounding errors outlined in Section 1.3, the X-, Y-, and Z- coordinates are stored in two separate BMPs. Finally, (vi) the pixel data is stored in three BMPs reflecting the coordinates and the RGB values of every sampled point. These textures can then be used by the VR engine to construct and present the point cloud similar to a network without links. However, this does not overwrite the current active network making it possible to load and display multiple structures simultaneously (Fig. 6).

7.7.2 Preprocessing of large structures

To make protein structures larger than 2,700 aa available within the VRNetzer platform, users can download the AlphaFold DB bulk data for model organism proteomes (refer to Tab. A2). This data covers all predicted structures for the 48 available organisms. Structures larger than 2,700 aa are split into n fractions, and their filenames contain "F1", "F2", up to "F n ".

The `vrport` package allows to process PDB files directly from these bulk downloads, performs a pre-processing with ChimeraX, and combines the multi-fraction structures into a single GLB file. This combined GLB file can then be used in the third step of the workflow described in Section 7.7.1.

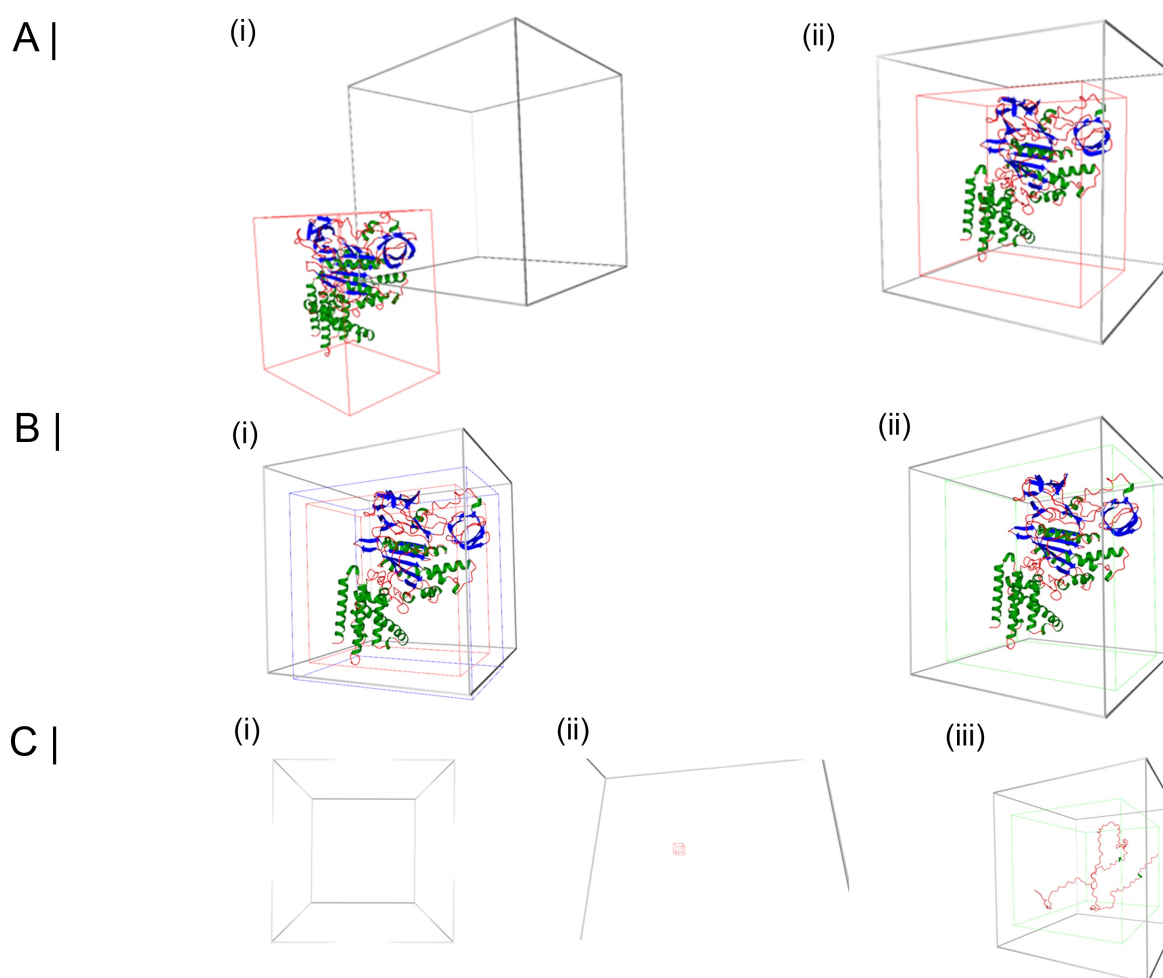


FIGURE 12 | VISUALIZATION OF THE TRANSFORMATION PROCESS

The model shown in A and B originates from the predicted AlphaFold structure with the UniProtID P38606. It represents a subunit of the human protein “catalytic subunit A of the V-type proton ATPase” [42, 43]. All visualizations are created using the Python library Open3D [48].

A | Visualization of a protein structure with its boundary box (red) in relation to the $1 \times 1 \times 1$ normalized cube (grey). (i) The coordinates of the protein structure are not positioned within the required normalization boundaries visualized by the grey cube. (ii) The protein structure has now the correct position and all coordinates lie within the interval $[0, 1]$.

B | Visualization of the scaling process. (i) The boundary box of the protein structure (red) does not fully exceed the boundaries of the normalized cube (grey). When scaled with the wrong scaling factor, the new boundary box (blue) of the structure exceeds the boundaries of the normalized cube. (ii) By using the correct scaling factor the boundary box of the structure (green) exploits the full extent of the normalized cube without exceeding its boundaries.

C | Visualization of up-scaling a structure. Shown is the predicted protein structure of the 165 aa long *D. melanogaster* protein GEO04367p1 with UniProtID Q9W3H6 [42, 43] as an example for a small protein structure. (i) The structure is too small to see it. (ii) By zooming in, the structure becomes visible with its red boundary box. (iii) Through scaling, the structure exploits the full extent of the normalized cube without exceeding its boundaries.

7.7.3 Data availability

For easy useability, a bulk download of all human protein structures is provided. This archive of protein structure BMPs contains all human protein structures in two processing modes: Highlighting of secondary structures in ChimeraX's cartoon style *(i)*, where loop regions are red, helices in green and β -sheets in blue (Fig. 11), and *(ii)* electrostatic highlighting, where areas with negative electrostatic potential are colored red through white to blue for areas with positive electrostatic potential. In the latter case, the protein structure's molecular surface is represented. This package also includes all multi-fraction structures, combined using the method described in Section 7.7.2. All structures contained in this package originate from version 4 of the AlphaFold DB.

A comprehensive ReadMe instruction file is included in this archive, outlining the steps to add the structures respective directory in the VRNetzer backend server.

7.8 Annotation highlighting

Annotations of all projects are scanned during VRNetzer backend boot up and written to a JSON file in the respective project directory to allow for a quick access whenever they are requested by a client a parallelized approach is used for this process to handle multiple projects at once. Whenever a client requests the annotations of a network, the VRNetzer backend checks if the annotations are already processed, and if not, whether they are currently processed by one of the parallelized processes.

If necessary, the requested annotations are placed on the first position of the job queue to be processed by the next available process.

Each client requests the annotations of the current active project as soon as the main panel is loaded. The UI of the selection tab is constructed as soon as the requested annotations are transmitted from the server to the client.

When requested by a client, the selected nodes and links can be highlighted using one of the three highlighting modes (Fig. 9). In this process, the node and link layout colors of the currently active project are masked using a calculated BMP. The resulting temporary project is then set as the active project, and the clients get informed about the project change.

In the case that only a selection of links is provided, all nodes that are connected to the selected links will be highlighted. As a result, only the *selection* mode is available in this scenario, since choosing the *isolation* mode would result in the same highlighting as in the *selection* mode. Similarly, the *bipartite* mode would highlight all links except the selected ones, which may not align with the user's expectation.

8 DISCUSSION

Summary

During the course of this master's thesis, significant improvements were made to the VRNetzer ecosystem through the development of new tools and features. Two packages, namely **StringEx** and **ProteinStructureFetch**, were developed to connect the VRNetzer with other tools, resources, and software which are well-established and commonly used in biomedical research. These extensions allow for the import of high-quality PPI networks from STRING DB and the visualization of predicted protein structures from AlphaFold DB, significantly expanding the available content of the VRNetzer.

The **VRNetzerApp** provides Cytoscape and STRING users with an easy-to-use workflow to facilitate the analysis of their networks in a fully interactive VR environment. Finally, the newly introduced annotation highlighting feature provides a useful tool for the analysis of STRING networks and other networks which come with a broad collection of node and link annotations. This feature represents an important initial step towards potential future improvements in integrating network analysis tools within the VRNetzer ecosystem. While most of the features developed in the course of this thesis did not require modifications of the VR engine itself, the development of the presented packages revealed substantial inconsistencies in terms of the customizability of the VRNetzer ecosystem. To address and resolve them, I introduced a newly developed extension system which streamlines the development process and facilitates the addition of new features, content, and tools to the VRNetzer ecosystem.

Overall, these enhancements greatly improve the usability of the VRNetzer ecosystem and its usefulness in biomedical research and other areas that benefit from 3D visualization and interaction with complex data.

Possible enhancements to the StringEx extension

The **StringEx** extension enables the access to publicly available PPI interaction data sourced from STRING DB. However, networks fetched by stringApp are limited in their size as they are intended to be analyzed on a two dimensional computer screen. As the VRNetzer is tailored to the exploration of large-scale data sets, these networks do not fully exploit the potential of the platform. To compensate for this, **StringEx** provides curated interactomes of nine organisms which come with a broad collection of node and link annotations. By providing a programmatic access to large datasets of similar quality to those retrieved by stringApp, this connection could fully exploit the visualization potential of the VRNetzer.

Possible enhancements to the ProteinStructureFetch extension

The **ProteinStructureFetch** extension enables the preparation of predicted protein structures from AlphaFold DB [42, 43]. However, the preparation of large-scale structures requires the fetching of the full proteome of the respective organism. In many circumstances where only a single large structure is of interest, this becomes a significant overhead.

Furthermore, here, only predicted structures sourced from a single DB are provided. To address both of these issues, the **ProteinStructureFetch** extension and its underlying Python package **vrprot** could be extended to allow the preparation of structures from other sources, such as the Research Collaboratory for Structural Bioinformatics (RCSB) PDB [49].

Although the visualization of protein structures as point clouds does not allow for a molecularly accurate analysis of the structure, these point clouds are primarily intended to provide a visual representation that gives a general understanding of the molecular structure underlying a node of interest in a molecular network. This means that if the integration of protein structures significantly enhances the analysis of PPI networks or other molecular networks, it may be worth considering options that allow for a molecularly more accurate representation.

BIBLIOGRAPHY

- [1] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.
- [2] Mark Newman. *Networks Second Edition*. Oxford University Press, 2018.
- [3] Pisanu Buphamalai, Michael Caldera, Felix Müller, and Jörg Menche. Network medicine. In Nataša Pržulj, editor, *Analyzing Network Data in Biology and Medicine: An Interdisciplinary Textbook for Biological, Medical and Computational Scientists*, chapter 10, pages 414–458. Cambridge University Press, 2019.
- [4] Mihaela Pertea and Steven L Salzberg. Between a chicken and a grape: estimating the number of human genes. *Genome biology*, 11(5):1–7, 2010. doi: 10.1186/gb-2010-11-5-206.
- [5] Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2): 167–256, 2003. doi: 10.1137/S003614450342480.
- [6] Sebastian Pirch, Felix Müller, Eugenia Iofinova, Julia Pazmandi, Christiane VR Hütter, Martin Chiettoni, Celine Sin, Kaan Boztug, Iana Podkosova, Hannes Kaufmann, et al. The VRNetzer platform enables interactive network analysis in Virtual Reality. *Nature communications*, 12(1):1–14, 2021. doi: 10.1038/s41467-021-22570-w.
- [7] Marc Vidal, Michael E Cusick, and Albert-László Barabási. Interactome networks and human disease. *Cell*, 144(6):986–998, 2011. doi: 10.1016/j.cell.2011.02.016.
- [8] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S. Baliga, Jonathan T. Wang, Daniel Ramage, Nada Amin, Beno Schwikowski, and Trey Ideker. Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13:2498–2504, 11 2003. ISSN 1088-9051. doi: 10.1101/GR.1239303. URL <https://genome.cshlp.org/content/13/11/2498.full><https://genome.cshlp.org/content/13/11/2498><https://genome.cshlp.org/content/13/11/2498.abstract>.
- [9] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring Network Structure, Dynamics, and Function using NetworkX. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008. URL <https://www.osti.gov/biblio/960616>.

- [10] Rose Oughtred, Chris Stark, Bobby-Joe Breitkreutz, Jennifer Rust, Lorrie Boucher, Christie Chang, Nadine Kolas, Lara O'Donnell, Genie Leung, Rochelle McAdam, et al. The biogrid interaction database: 2019 update. *Nucleic acids research*, 47(D1):D529–D541, 2019. doi: 10.1093/nar/gky1079.
- [11] Sandra Orchard, Mais Ammari, Bruno Aranda, Lionel Breuza, Leonardo Briganti, Fiona Broackes-Carter, Nancy H Campbell, Gayatri Chavali, Carol Chen, Noemi Del-Toro, et al. The mintact project—intact as a common curation platform for 11 molecular interaction databases. *Nucleic acids research*, 42(D1):D358–D363, 2014. doi: 10.1093/nar/gkt1115.
- [12] Martin H Schaefer, Jean-Fred Fontaine, Arunachalam Vinayagam, Pablo Porras, Erich E Wanker, and Miguel A Andrade-Navarro. HIPPIE: Integrating protein interaction networks with experiment based quality scores. *PloS one*, 7(2):e31826, 2012. doi: 10.1371/journal.pone.0031826.
- [13] Damian Szklarczyk, Annika L Gable, Katerina C Nastou, David Lyon, Rebecca Kirsch, Sampo Pyysalo, Nadezhda T Doncheva, Marc Legeay, Tao Fang, Peer Bork, Lars J Jensen, and Christian von Mering. The STRING database in 2021: customizable protein–protein networks, and functional characterization of user-uploaded gene/measurement sets. *Nucleic acids research*, 49(D1):D605–D612, 2021. doi: 10.1093/nar/gkaa1074.
- [14] Damian Szklarczyk, Rebecca Kirsch, Mikaela Koutrouli, Katerina Nastou, Farrokh Mehryary, Radja Hachilif, Annika L Gable, Tao Fang, Nadezhda T Doncheva, Sampo Pyysalo, et al. The string database in 2023: protein–protein association networks and functional enrichment analyses for any sequenced genome of interest. *Nucleic Acids Research*, 51(D1):D638–D646, 2023.
- [15] STRING consortium. STRING DB: functional protein association networks, 2022. URL <https://string-db.org/>. Accessed: 2022-07-13.
- [16] Nadezhda T Doncheva, John H Morris, Jan Gorodkin, and Lars J Jensen. Cytoscape StringApp: network analysis and visualization of proteomics data. *Journal of proteome research*, 18(2):623–632, 2018. doi: 10.1021/acs.jproteome.8b00702.
- [17] Michael Kuhn, Christian von Mering, Monica Campillos, Lars Juhl Jensen, and Peer Bork. Stitch: interaction networks of chemicals and proteins. *Nucleic acids research*, 36(suppl_1):D684–D688, 2007. doi: 10.1093/nar/gkm795.

- [18] Cancer Genome Atlas Network et al. Comprehensive molecular characterization of human colon and rectal cancer. *Nature*, 487(7407):330, 2012.
- [19] Steve Bryson. Virtual reality in scientific visualization. *Communications of the ACM*, 39(5):62–71, 1996. doi: 10.1016/0097-8493(93)90117-R.
- [20] Matthias Kraus, Johannes Fuchs, Björn Sommer, Karsten Klein, Ulrich Engelke, Daniel Keim, and Falk Schreiber. Immersive analytics with abstract 3D visualizations: A survey. In *Computer Graphics Forum*, volume 41, pages 201–229. Wiley Online Library, 2022. doi: 10.1111/cgf.14430.
- [21] EPIC GAMES. Unreal Engine 4, 2015. URL <https://www.unrealengine.com/en-US/>. Accessed: 2023-02-08.
- [22] Socket.IO. Socket.io: Bidirectional and low-latency communication for every platform, 2012. URL <http://socket.io/>. Accessed: 2023-02-08.
- [23] The Pallets organization. Flask: A simple framework for building complex web applications., 2010. URL <https://flask.palletsprojects.com/en/2.2.x/>. Accessed: 2023-02-03.
- [24] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.
- [25] Paul Bourke et al. A beginners guide to bitmaps. *Renderings and Models by Peter Diprose and Bill Rattenbury*, 1993. URL <http://paulbourke.net/dataformats/bitmaps/>.
- [26] Ookla. Median country speeds february 2023, 2023. URL <https://www.speedtest.net/global-index>. Accessed: 2023-02-08.
- [27] Michael Housh. Flask-socketio blueprint, 2016. URL <https://github.com/m-housh/io-blueprint>. Accessed: 2023-02-08.
- [28] Brian Wojtczak. Flask-socketio blueprint, 2020. URL <https://gist.github.com/astrolox/445e84068d12ed9fa28f277241edf57b>. Accessed: 2023-02-08.
- [29] Christiane VR Hütter, Celine Sin, Felix Müller, and Jörg Menche. Network cartographs for interpretable visualizations. *Nature Computational Science*, 2(2):84–89, 2022. doi: 10.1038/s43588-022-00199-z.

-
- [30] Dexter Pratt, Jing Chen, David Welker, Ricardo Rivas, Rudolf Pillich, Vladimir Rynkov, Keiichiro Ono, Carol Miello, Lyndon Hicks, Sandor Szalma, et al. Ndex, the network data exchange. *Cell systems*, 1(4):302–305, 2015. doi: 10.1016/j.cels.2015.10.001.
- [31] Bruno Aranda, Hagen Blankenburg, Samuel Kerrien, Fiona SL Brinkman, Arnaud Ceol, Emilie Chautard, Jose M Dana, Javier De Las Rivas, Marine Dumousseau, Eugenia Galeota, et al. Psicquic and psiscore: accessing and scoring molecular interactions. *Nature methods*, 8(7):528–529, 2011. doi: 0.1038/nmeth.1637.
- [32] Keiichiro Ono, Tanja Muetze, Georgi Kolishovski, Paul Shannon, and Barry Demchak. CyREST: turbocharging cytoscape access for external tools via a restful api. *F1000Research*, 4, 2015. doi: 10.12688/f1000research.6767.1.
- [33] Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Allan P Davis, Kara Dolinski, Selina S Dwight, Janan T Eppig, et al. Gene ontology: tool for the unification of biology. *Nature genetics*, 25(1):25–29, 2000. doi: 10.1038/75556.
- [34] The gene ontology resource: enriching a gold mine. *Nucleic acids research*, 49(D1): D325–D334, 2021. doi: 10.1093/nar/gkaa1113.
- [35] Sune Pletscher-Frankild, Albert Pallejà, Kalliopi Tsafou, Janos X Binder, and Lars Juhl Jensen. DISEASES: Text mining and data integration of disease–gene associations. *Methods*, 74:83–89, 2015.
- [36] Alberto Santos, Kalliopi Tsafou, Christian Stolte, Sune Pletscher-Frankild, Seán I O’Donoghue, and Lars Juhl Jensen. Comprehensive comparison of large-scale tissue expression datasets. *PeerJ*, 3:e1054, 2015. doi: 10.7717/peerj.1054.
- [37] Oana Palasca, Alberto Santos, Christian Stolte, Jan Gorodkin, and Lars Juhl Jensen. TISSUES 2.0: an integrative web resource on mammalian tissue expression. *Database*, 2018, 02 2018. ISSN 1758-0463. doi: 10.1093/database/bay003. URL <https://doi.org/10.1093/database/bay003>. bay003.
- [38] Janos X Binder, Sune Pletscher-Frankild, Kalliopi Tsafou, Christian Stolte, Seán I O’Donoghue, Reinhard Schneider, and Lars Juhl Jensen. Compartments: unification and visualization of protein subcellular localization evidence. *Database*, 2014, 2014. doi: 10.1093/database/bau012.
-

- [39] Marc Gillespie, Bijay Jassal, Ralf Stephan, Marija Milacic, Karen Rothfels, Andrea Senff-Ribeiro, Johannes Griss, Cristoffer Sevilla, Lisa Matthews, Chuqiao Gong, et al. The reactome pathway knowledgebase 2022. *Nucleic acids research*, 50(D1):D687–D692, 2022. doi: 10.1093/nar/gkab1028.
- [40] Felix Fischer and Till P Oblau. Implementation of 3D protein structures into a Virtual Reality data exploration platform. *[Unpublished manuscript]*, 2022.
- [41] Curriculum für das masterstudium bioinformatik, 2022. URL http://senat.univie.ac.at/fileadmin/user_upload/s_senat/konsolidierte_Masterstudien/MA_Bioinformatik.pdf. Accessed: 2023-01-22.
- [42] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873): 583–589, 2021. doi: 10.1038/s41586-021-03819-2.
- [43] Mihaly Varadi, Stephen Anyango, Mandar Deshpande, Sreenath Nair, Cindy Natassia, Galabina Yordanova, David Yuan, Oana Stroe, Gemma Wood, Agata Laydon, et al. AlphaFold protein structure database: massively expanding the structural coverage of protein-sequence space with high-accuracy models. *Nucleic acids research*, 50(D1): D439–D444, 2022. doi: 10.1093/nar/gkab1061.
- [44] Eric F Pettersen, Thomas D Goddard, Conrad C Huang, Elaine C Meng, Gregory S Couch, Tristan I Croll, John H Morris, and Thomas E Ferrin. Ucsf chimeraX: Structure visualization for researchers, educators, and developers. *Protein Science*, 30(1):70–82, 2021. doi: 10.1002/pro.3943.
- [45] Alexander Pico, Fred Eisele, and Barry Demchak. Cytoscape app ladder · cytoscape/cytoscape wiki, 2018. URL <https://github.com/cytoscape/cytoscape/wiki/Cytoscape-App-Ladder>. Accessed: 2022-10-07.
- [46] The Pallets organization. Jinja: A fast, expressive, extensible templating engine., 2007. URL <https://jinja.palletsprojects.com/en/3.0.x/>. Accessed: 2023-02-12.
- [47] Hongzhan Huang, Peter B McGarvey, Baris E Suzek, Raja Mazumder, Jian Zhang, Yongxing Chen, and Cathy H Wu. A comprehensive protein-centric ID mapping service for molecular data integration. *Bioinformatics*, 27(8):1190–1191, 2011. doi: 10.1093/bioinformatics/btr101.

- [48] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.
- [49] Helen M Berman, John Westbrook, Zukang Feng, Gary Gilliland, Talapady N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. The protein data bank. *Nucleic acids research*, 28(1):235–242, 2000. doi: 10.1093/nar/28.1.235.
- [50] Barry Demchak. py4cytoscape, 2020. URL <https://pypi.org/project/py4cytoscape/>. Accessed: 2023-02-08.
- [51] Miguel Grinberg. *Flask web development: developing web applications with python*. "O'Reilly Media, Inc.", 2018.
- [52] Miguel Grindberg. Flask-socketio, 2014. URL <https://flask-socketio.readthedocs.io/>. Accessed: 2023-02-11.
- [53] Damien Arrachequesne et al. URL <https://github.com/socketio/socket.io-client>. Accessed: 2023-02-11.
- [54] Tim Berners-Lee. The original http as defined in 1991. *World Wide Web Consortium (W3C)*, 1991.

APPENDIX

APPENDIX A TABLES

TABLE A1 | FILE SIZE COMPARISON OF EXPORTED STRING NETWORKS

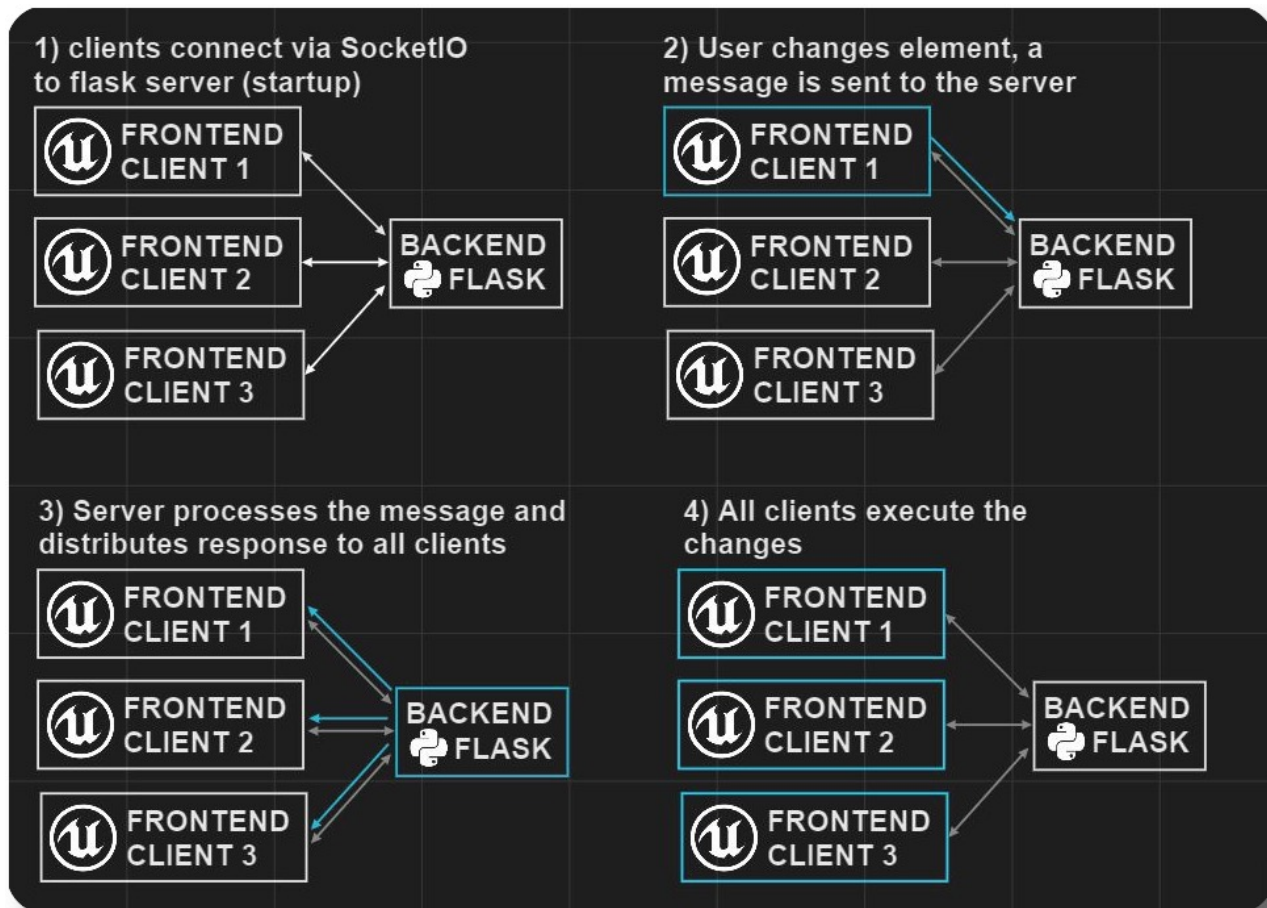
# Nodes	# Links	File size in megabytes		
		cyjs	GraphML	VRNetzer
100	1,600	5,63	5,99	1,83
200	4,939	11,52	12,05	2,87
300	9,353	18,22	19,16	4,29
400	13,747	24,50	25,85	7,57
2,000	89,728	131,50	140,18	36,68

File size comparison of STRING network exported from Cytoscape in three distinct file formats. Networks have been fetched using the disease query of Doncheva et al.'s stringApp [16] ver. 1.7.1. The fetched network is the full STRING Alzheimer's disease network with the Disease Ontology Identifiers (DOID) DOID:10652. A cutoff of 0.4 was chosen and a functional enrichment was loaded. For each row, a new network has been fetched by changing the respective maximum number of proteins. The file size is given in megabytes.

TABLE A2 | EXTERNAL LINKS

External resource	Link
STRING-VRNetzer Project repository	https://github.com/menchelab/STRING-VRNetzer
VRNetzer Branch	https://github.com/ObT1337/VRNetzer_Backend_THESIS
Example Extension	https://github.com/menchelab/ExampleExtension
VRNetzerApp	https://github.com/menchelab/STRING-VRNetzer/blob/main/VRNetzerApp/target/VRNetzerApp-1.0.0.jar
StringEx	https://github.com/menchelab/StringEx
ProteinStructureFetch	https://github.com/menchelab/ProteinStructureFetch
vrprot	https://pypi.org/project/vrprot/
Processed Human proteome	https://ucloud.univie.ac.at/index.php/s/Ozz3XXHyZ6HSGKP
AlphaFold bulk downloads	https://alphafold.ebi.ac.uk/downloads

APPENDIX B FIGURES

**FIGURE B1 | USAGE OF SOCKET.IO WITHIN THE VRNETZER ECOSYSTEM**

SocketIO is used in the VRNetzer ecosystem to provide a communication channel between the Flask server and the respective web clients. As illustrated in this figure provided by [6] the client connects (1) to the server and requests via a SocketIO connection the state of the current session. Whenever a user changes the state of an element on their UI, the server is notified (2) and updates the session state. The server then broadcasts the new session state to all connected clients (3) and (4) the UI elements on their end are updated accordingly.

APPENDIX C FILE EXAMPLES

C1 EXAMPLE VRNETZ FILE

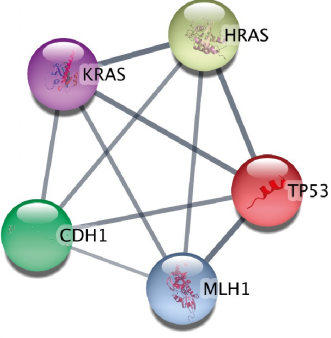
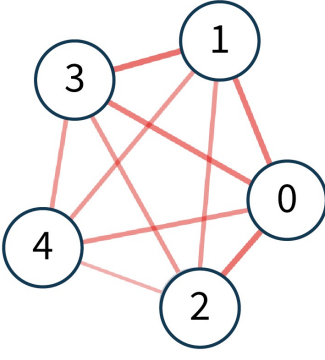
<pre>{ "nodes": [{ "id": 0, "size": 1, "cy_pos": [5, 3], "cy_col": [213, 109, 110], "n": "TP53", "additionalAttributes": "...", }, { "additionalNodes": "...", }], </pre>	
<pre> "links": [{ "s": 0, "e": 1, "id": 1, "name": " TP53 (pp) HRAS", "additionalAttributes": "...", }, { "additionalLinks": "...", }], </pre>	
<pre> "network": { "name": "Network", "additionalAttributes": "...", }, </pre>	Number of Nodes: 5 Number of Links: 7 Network diameter: 2 ...

FIGURE B2 | THE VRNETZ FILE CONSISTS OF THREE MAIN COMPONENTS Nodes, links, and network attributes. It stores attributes such as name, ID, and more for both nodes and links. Additionally, the network attributes contain information about the network, such as its name, database origin, and statistics. Show is a small scale network fetch with stringApp [16] and visualized in Cytoscape [8].

APPENDIX D DESIGN CHOICES

D1 EXPORTING NETWORK DATA FROM CYTOSCAPE

The first prototype, which covered the process of exporting data from Cytoscape in an appropriate format that could be uploaded to the VRNetzer, was written in Python. This prototype automated the fetching of STRING networks with the stringApp and their export. It utilized py4cytoscape [50] to communicate with Cytoscape RESTAPI CyREST [32]. Cytoscape natively supports a variety of data formats for the export of network data, and these formats were utilized by this prototype.

In particular two file formats showed the most promise as they are easily to parse with Python. The first format is the Extensible Markup Language (XML)-based file format GraphML which is widely-used to store graph/network data and can be easily parsed by NetworkX [9]. The second promising file format Cytoscape offers natively is Cytoscape.cyjs. This JSON-based file format is designed to be used with Cytoscape's JSgraph library Cytoscape.js. JSON files are similar to Python dictionaries in that they contain human-readable attribute-value pairs.

However, all natively available file formats have two main problems. First, they do not contain the applied network style, whereby information like node color are not included. To extract this information, one would also need to export the style and subsequently recreate the underlying color scheme. Moreover, they all contain every column of the node and edge table. STRING networks in particular store some of their additional information in these very tables, including BMPs of the protein structure displayed on each node (see Fig. 1). Accordingly, the exported files are unnecessarily large and contain some redundant information.

Thus, I decided to develop a Cytoscape app to gain more control over the exported data and to make the export more user-friendly.

D2 VRPROT: A STANDALONE PYTHON PACKAGE

Since its first introduction by Fischer and Oblau, the solution to fetch and process protein structures from AlphaFold DB has become a sophisticated software with an extensive range of functionality. To harness its potential and allow for greater versatility, I decided to make this solution a standalone Python package, providing other developers with the ability to leverage its functionality in their own projects and expand upon it as needed.

By utilizing the functionalities provided by `vrprot`, the development of **ProteinStructureFetch** became more streamlined and efficient, as redundant code was eliminated. This also makes the extension more adaptable to future enhancements, such as access to a broader variety of protein structure DBs.

APPENDIX E TECHNICAL DETAILS

E1 VRNETZER VERSION

As mentioned in Section 1.3, the VRNetzer ecosystem is currently undergoing a transition to a new version. To ensure compatibility with the features presented in this thesis, here a frozen version of the VRNetzer is provided. The link to download this version can be found in Table A2.

E2 PYTHON FLASK

This section gives a small overview of the Python Flask [23] framework based on Grinberg [51]. Flask has been developed by Armin Ronacher and can be used in any kind of web app. It is termed to be a “lightweight microframework” providing only basic functionalities to keep things simple while allowing to extend its tool set with extensions such as `Socket.IO` (see E3). This gives developers the freedom to construct the framework that is applicable to their needs and requirements.

Every Flask app consists of an app instance which handles all requests passed by the server. The two basic concepts of a Flask app are routes and views. A route defines which Python code, i.e. view function, is executed when the corresponding URL is requested. The view function provides the response to the request which can be a simple string, a HTML web page or data such as JSON objects or images. To provide extensibility to an app, Flask offers a system called blueprints. This system enables developers to construct apps in a more modular way. A blueprint can define routes just like the app itself, making it easy to use and organize. The new VRNetzer extension system utilizes blueprints, to empower extensions with the capability to define their own routes, integrate their own source code and bring custom resources with them.

Additionally, Flask employs Jinja2 [46] for defining and constructing HTML templates, which enables dynamic insertion of data into HTML templates. For instance this is used in the VRNetzer extension system to dynamically add custom UI panels to existing UI elements like the main panel or the uploader panel.

E3 SOCKET.IO

Socket.IO is a library implemented in many programming languages including Python, JS, and C++. It allows for low-latency, bidirectional, and event-based communication [22] between multiple clients and a data server. The VRNetzer ecosystem uses this feature to synchronize the UIs and data between clients. Along with the implemented REST API, this allows for a secondary method of data exchange using the Socket.IO extension for Flask [52] and the Socket.IO library for JS [53].

E4 REST API

The representational state transfer software architecture was first described by Fielding with the aim to provide a uniform interface to access resources [24]. A REST API acts as a bridge that allows multiple clients to access and manipulate data on a server. The most widely used protocol for this type of API is HTTP, which is utilized by everyone on a daily basis when browsing the internet. Our web browser acts as a client sending an HTTP request to a server whenever we navigate to a website. The server then responds with a message containing the requested data, i.e. the content of the website, which is then rendered by the browser. This kind of request is termed as a GET request.

The received message does not always have to be website content, it can also be data in a different format, such as JSON or XML. HTTP also provides other methods such as POST, PUT, PATCH or DELETE [54]. When uploading a network to the VRNetzer, a POST request is sent to the VRNetzer REST API with the provided data to create a new project. The **VRNetzerApp** can also send networks directly to the VRNetzer ecosystem using the VRNetzer REST API through a POST request. Similarly, **StringEx** uses the CyREST [32] API to send sub-networks to Cytoscape for in-depth analysis. These examples demonstrate the common usage of REST APIs in software or web applications, which provide a uniform interface for communication between different applications. They not only allow access to data but also facilitate the interaction between them.