

Capstone Project Phase B

**A Tool for Analyzing fNIRS Hyperscanning Using Graph Measures**

Project Number: 25-1-D-1

Author: Yonatan Sherer  
Supervisor: Dr. Anat Dahan

<https://github.com/YonatanSherer/fNIRS-Hyperscanning-Tool>

Table of Contents

1. **Abstract3**
2. **Introduction3**

2.1. Background3

2.2. Motivation4

2.3. Objectives4

1. **Problem Statement4**

3.1. Current challenges4

3.2. Specific Problems Addressed5

3.3. Success Criteria5

**4. System Requirements5**

4.1. Functional Requirements5

4.2. Non-Functional Requirements6

**5. System Architecture & Design7**

**6. General Description8**

6.1. System Characteristics8

6.2. Target Audience8

**7. Solution Description8**

7.1. System Architecture8

7.2. Global and Local Graph Measures Comparison 9

7.3. System Operation14

**8. Software Development Work16**

**9. Challenges and Solutions18**

**10. Results and Conclusions19**

**11. Lessons Learned19**

**User Guide21**

**Maintenance Guide25**

**1. Abstract**

This project presents the development of a comprehensive graph analysis tool for functional Near-Infrared Spectroscopy (fNIRS) hyperscanning data. The tool addresses the growing need for sophisticated analysis methods in social neuroscience research, where multiple participants' brain activities are simultaneously recorded during interactive tasks. The system implements advanced graph theory algorithms to analyze inter-brain connectivity patterns, providing researchers with intuitive visualization and statistical analysis capabilities.

The developed tool integrates signal processing techniques and graph theory metrics approaches to extract meaningful insights from hyperscanning data. Key features include connectivity analysis and statistical comparison between different experimental conditions. The system was validated using both synthetic datasets and real hyperscanning experiments, demonstrating its effectiveness in identifying significant inter-brain synchronization patterns.

Results indicate that the tool successfully captures complex network dynamics during social interactions, with applications in studying cooperation, empathy, and communication. The modular architecture ensures extensibility for future enhancements, while the user-friendly interface makes it accessible to researchers with varying technical backgrounds.

**Keywords:** fNIRS, hyperscanning, graph analysis, brain connectivity, social neuroscience, network analysis

**2. Introduction**

**2.1 Background**

Functional Near-Infrared Spectroscopy (fNIRS) has emerged as a powerful neuroimaging technique for studying brain function in naturalistic settings. Unlike traditional neuroimaging methods, fNIRS allows for the simultaneous recording of multiple participants' brain activity during interactive tasks, a technique known as hyperscanning. This capability has opened new avenues for understanding the neural basis of social cognition, communication, and interpersonal dynamics.

The analysis of hyperscanning data presents unique challenges due to the complexity of inter-brain connectivity patterns and the need for sophisticated analytical tools. Traditional single-brain analysis methods are insufficient for capturing the dynamic interactions between multiple neural systems. Graph theory provides a mathematical framework for representing and analyzing these complex network structures, making it an ideal approach for hyperscanning data analysis.

**2.2 Motivation**

Current tools for fNIRS hyperscanning analysis are often limited in scope, requiring researchers to use multiple software packages or develop custom solutions. There is a significant need for a comprehensive, user-friendly tool that can handle the entire analysis pipeline from raw data processing to statistical interpretation. This project aims to fill this gap by developing an integrated analysis platform specifically designed for fNIRS hyperscanning research.

**2.3 Objectives**

The primary objectives of this project include:

1. Development of a comprehensive analysis pipeline for fNIRS hyperscanning data.
2. Implementation of advanced graph theory algorithms for network analysis.
3. Creation of intuitive visualization tools for complex network data.
4. Integration of statistical analysis methods for experimental comparisons.
5. Validation using real-world datasets from social neuroscience experiments.
6. Design of a user-friendly interface accessible to researchers with varying technical backgrounds.

**3. Problem Statement**

**3.1 Current Challenges**

The analysis of fNIRS hyperscanning data faces several significant challenges:

**Data Complexity:** Hyperscanning experiments generate large volumes of multi-dimensional data with complex temporal and spatial relationships. Traditional analysis methods struggle to capture the full richness of inter-brain connectivity patterns.

**Lack of Standardization:** The field lacks standardized protocols for preprocessing, analysis, and interpretation of hyperscanning data. This inconsistency makes it difficult to compare results across studies and laboratories.

**Technical Barriers:** Many researchers in social neuroscience lack the technical expertise to implement sophisticated graph analysis algorithms, limiting the adoption of advanced analytical methods.

**Limited Tool Integration:** Existing solutions are often fragmented, requiring researchers to use multiple software packages with incompatible data formats and analysis approaches.

**3.2 Specific Problems Addressed**

This project specifically addresses the following problems:

1. **Connectivity Analysis:** Implementing reliable algorithms for measuring inter-brain connectivity while accounting for individual differences and experimental conditions.
2. **Network Analysis:** Providing comprehensive graph theory metrics for characterizing network topology and dynamics.
3. **Statistical Inference:** Developing appropriate statistical methods for hypothesis testing in hyperscanning experiments.
4. **Visualization:** Creating effective visualization tools for complex network data that facilitate interpretation and communication of results.

**3.3 Success Criteria**

The success of this project will be measured by:

* **Functionality:** The tool successfully processes fNIRS hyperscanning data and produces meaningful connectivity analyses.
* **Accuracy:** Results are validated against known ground truth and existing analysis methods.
* **Usability:** The interface is intuitive and accessible to researchers with varying technical backgrounds.
* **Performance:** The system handles large datasets efficiently with reasonable processing times.
* **Reproducibility:** Results are consistent and reproducible across different datasets and experimental conditions.

**4. System Requirements**

**4.1 Functional Requirements**

The system was designed to meet a set of functional requirements that reflect the needs of researchers working with fNIRS hyperscanning data. These requirements span data handling, connectivity computation, graph theoretical analysis, visualization, and statistical evaluation.

Data Import and Export - The tool supports importing .mat files generated by fNIRS acquisition software and allows for flexible integration with common experimental folder structures. Although the current version primarily works with MATLAB .mat files, the architecture is extendable to support other formats. The system can handle multi-participant (dyadic) datasets, assuming temporal synchronization between participants. Results from various analysis stages, including correlation matrices, graph metrics, and thresholded data, can be exported in commonly used formats such as CSV, JSON, and MATLAB-compatible .mat files, enabling interoperability with other research tools.

Connectivity Analysis - The core functionality includes computing connectivity matrices based on Pearson correlation, which is appropriate for hyperscanning studies where simultaneous time-series data are recorded across participants. The system design is modular and can be extended to include other connectivity measures such as coherence or mutual information.

Graph Analysis - Once connectivity matrices are generated, the system constructs graphs representing intra- and inter-brain networks. It computes a wide range of standard graph-theoretical metrics using NetworkX, including clustering coefficient, global and local efficiency, modularity, and degree centrality. Users can compare these measures across different conditions or dyads to identify task- or group-related differences in brain network organization.

Visualization - Effective visualization was a central design goal of the system. Users can generate interactive heatmaps of correlation and thresholded matrices, and a variety of statistical comparison charts. These visualizations include bar charts, line graphs, and radar plots for comparing graph metrics across dyads and conditions. All visual outputs can be exported as publication-ready figures in PNG or PDF format to support scientific communication and reporting.

Statistical Analysis - The tool provides a framework for comparing metrics at both the global and local levels. Users can select multiple dyads and conditions and view comparative plots for selected measures.

**4.2 Non-Functional Requirements**

In addition to the core functional requirements, the system was developed to meet several non-functional criteria that ensure its usability, performance, reliability, and maintainability. These requirements support the tool’s effectiveness in real research environments and lay the groundwork for future extensibility.

Performance - The system is optimized to perform all standard analysis steps, from file parsing and correlation computation to thresholding and graph analysis, within a reasonable timeframe for typical fNIRS datasets. For a standard dyadic experiment with multiple conditions, the full pipeline completes in under ten minutes on a modern personal computer. Memory usage has been carefully managed through the use of NumPy and vectorized operations to accommodate larger datasets. The system’s architecture supports scalability, allowing it to adapt to experiments with varying numbers of channels, conditions, or dyads without compromising stability or speed.

Usability - Usability was a guiding principle throughout the development process. The tool features an intuitive graphical user interface (GUI) built with PyQt6, structured around a step-by-step analysis pipeline. Each screen provides clear guidance on the next operation, supported by tooltips, dialog boxes, and a dedicated user guide. Error messages are descriptive and help users correct issues such as missing files or unsupported data structures.

Reliability - The system incorporates robust error handling and validation mechanisms to ensure consistent performance. Integrity checks are applied during data import and graph generation to verify that all inputs are well-formed and compatible. Parameters provided by the user, such as threshold values, are validated to prevent invalid configurations. The tool is designed to recover from common issues, and wherever possible, informative messages guide the user toward resolution without requiring a restart of the application.

Maintainability - The software was developed with long-term maintainability in mind. Its architecture is modular, separating core functionality such as data processing, analysis, and visualization into distinct, well-documented components. This allows for easy extension, testing, and debugging of individual parts. The codebase includes meaningful comments and follows consistent naming conventions, making it accessible to future developers.

**5. System Architecture & Design**

The system follows a modular architecture with clear separation of concerns, organized into the following main components:

Data Layer: Handles data import, storage, and export operations  
Processing Layer: Implements folder processing and parsing algorithms  
Analysis Layer: Contains graph theory and statistical analysis methods  
Visualization Layer: Provides plotting and interactive visualization capabilities  
User Interface Layer: Manages user interactions and workflow orchestration

**6. General Description**

This project addresses the need for an accessible and effective analysis tool for functional near-infrared spectroscopy (fNIRS) hyperscanning data. Hyperscanning enables simultaneous brain activity recording from two or more participants, providing insights into inter-brain synchrony and social cognition.

The fNIRS Hyperscanning Analysis Tool was designed to help neuroscientists and cognitive researchers analyze these complex datasets using graph theory measures. It provides a pipeline that transforms raw data into interpretable graphs and offers both global and local comparisons across experimental conditions.

6.1 System Characteristics

* Intuitive, modular graphical user interface (GUI)
* Fully implemented in Python with PyQt6
* Supports correlation matrix generation, thresholding, and graph metric computation
* Visualization tools for comparing dyads and conditions

6.2 Target Audience

* Neuroscientists and researchers conducting hyperscanning experiments
* Research assistants or lab technicians with limited programming experience

**7. Solution Description**

7.1 System Architecture

The system is modular and screen-based:

1. Opening Screen: Entry point with user guidance
2. Analysis Pipeline Screen: Step-by-step preprocessing, correlation, and graph generation
3. Graph Measures Comparison Screen: Compare global graph metrics across conditions and dyads
4. Local Measures Comparison Screen: Explore local node-specific metrics such as node strength

The architecture is organized with two main types of files: UI logic for each screen and .ui files designed in Qt Designer.

All components interact with MATLAB .mat files and generate CSV, PNG and PDF outputs.

7.2 Global and Local Graph Measures Comparison

During the development process, I came to recognize the importance of providing two distinct but complementary forms of analysis: one focusing on global graph measures and the other on local node-level metrics. This distinction reflects how neuroscientists typically approach functional connectivity analysis, either by examining the network as a whole to understand general connectivity patterns, or by drilling down to individual nodes to uncover localized brain activity differences.  
My objective was to design the system in a way that supports both types of comparisons within the context of fNIRS hyperscanning data.

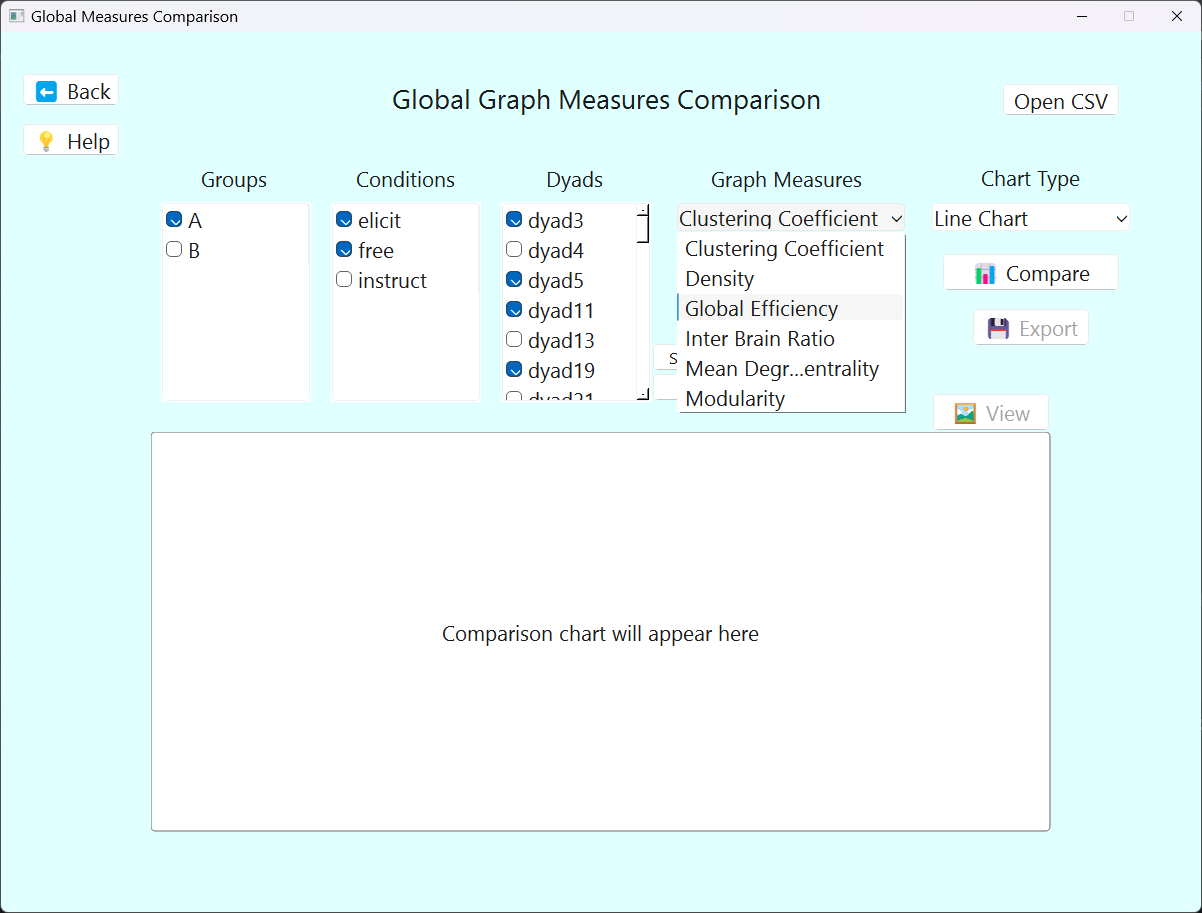
For the global graph measures comparison, the tool allows users to assess metrics such as Clustering Coefficient, which reflects local grouping of nodes, Global Efficiency that indicates the efficiency of information transfer across the network, and Modularity, which detects community structures within the graph. The selection interface for these analyses is shown in Figure 1, where users can choose graph metrics, dyads, and experimental conditions. Once selected, the results are visualized in comparative plots, such as bar charts, as seen in Figure 2, which highlights variation in global metrics across dyads or conditions. Additionally, the tool supports domain-specific measures like the Inter-brain Ratio, which quantifies the balance between within-brain and between-brain connectivity, a key concept in hyperscanning research. Mean Degree Centrality is also included, offering insights into the average level of connectedness in a network.

Since all these metrics are computed on weighted graphs, the tool ensures that edge weights, representing the strength of functional correlations between brain regions, are preserved throughout the analysis. These weights directly influence results, especially in measures such as global efficiency, modularity, and node strength, making their accurate representation essential for neuroscientific validity.  
The final output can be exported as a comprehensive PDF report, which includes the selected parameters, plotted graphs, and auto-generated textual insights, as demonstrated in Figure 3.

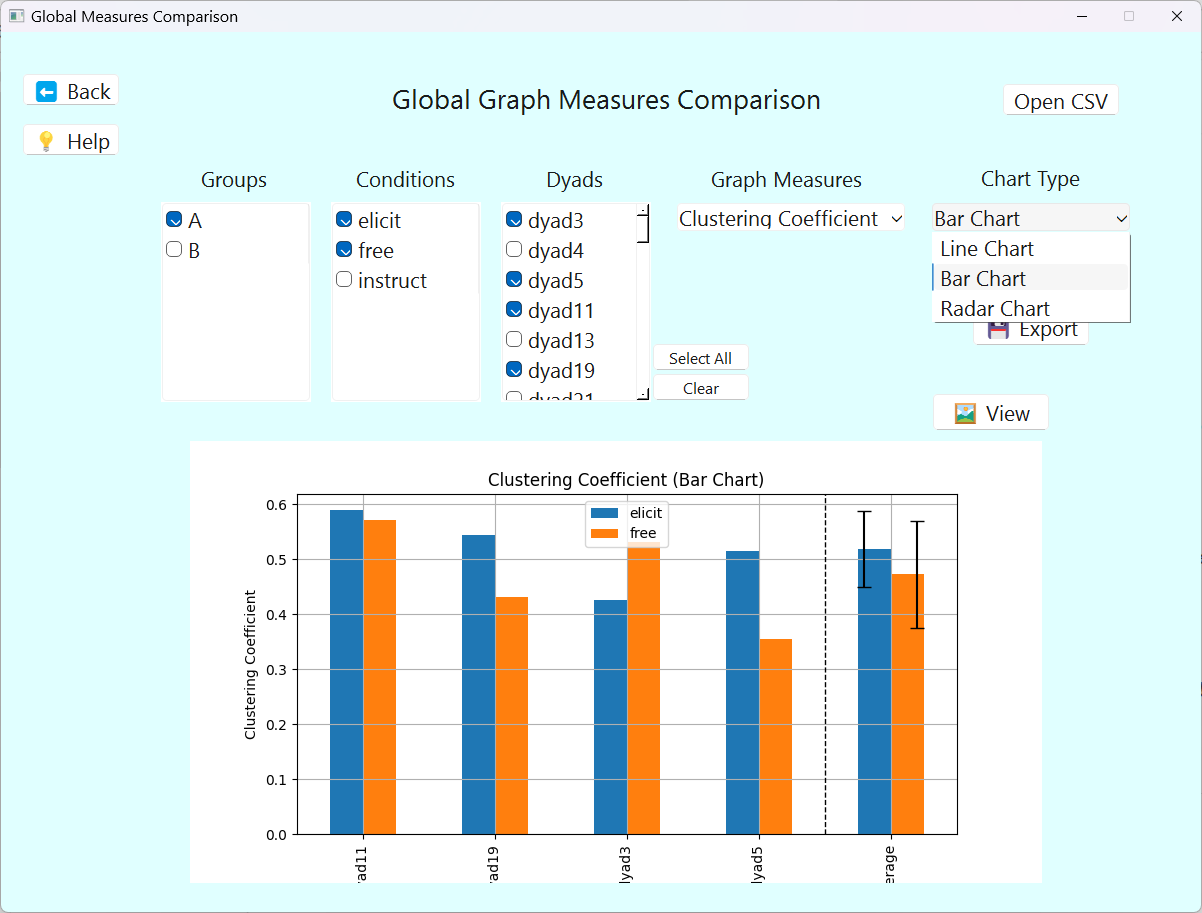
To support meaningful local graph measure comparisons, I implemented a dedicated screen where users can explore metrics like Node Strength and Local Efficiency across various combinations of groups, dyads, conditions, and nodes. Through this process, I realized that the flexibility to select specific comparisons was crucial. For example, users may wish to compare the same node across multiple conditions for a single dyad or group to detect task-related activation changes, or they may compare corresponding nodes between two participants in dyads to examine asymmetry in brain activity. These options are reflected in the bar chart visualization shown in Figure 4, where node-level metrics are plotted across selected conditions.

These choices are facilitated by intuitive controls that allow multi-selection of nodes and experimental conditions, with axes and labels adapting based on the comparison mode.   
The tool also provides an automatic interpretation of the results, giving researchers immediate textual insight into observed trends, as illustrated in Figure 4.1. Users can explore trends such as increased activation in specific brain areas during social interaction tasks or role-dependent differences in participant engagement within a dyad.

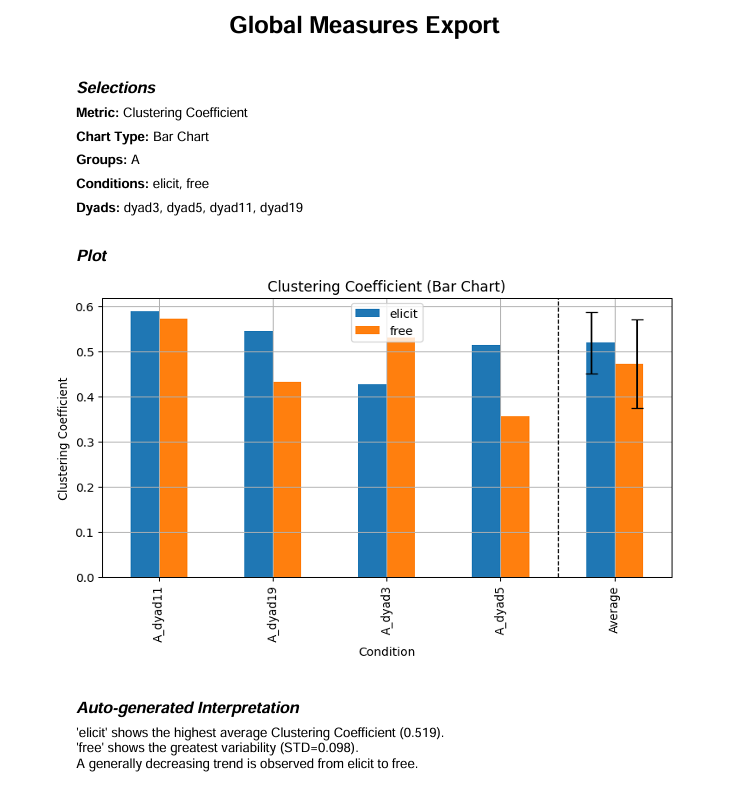
Designing the tool in this way required analyzing typical experimental setups in hyperscanning studies and anticipating which comparisons would yield the most scientific insight. As a result, the system enables researchers to uncover patterns not only within individual participants or conditions, but also across dyads, roles, and tasks, contributing to a deeper understanding of inter-brain dynamics in collaborative and cognitive experiments.



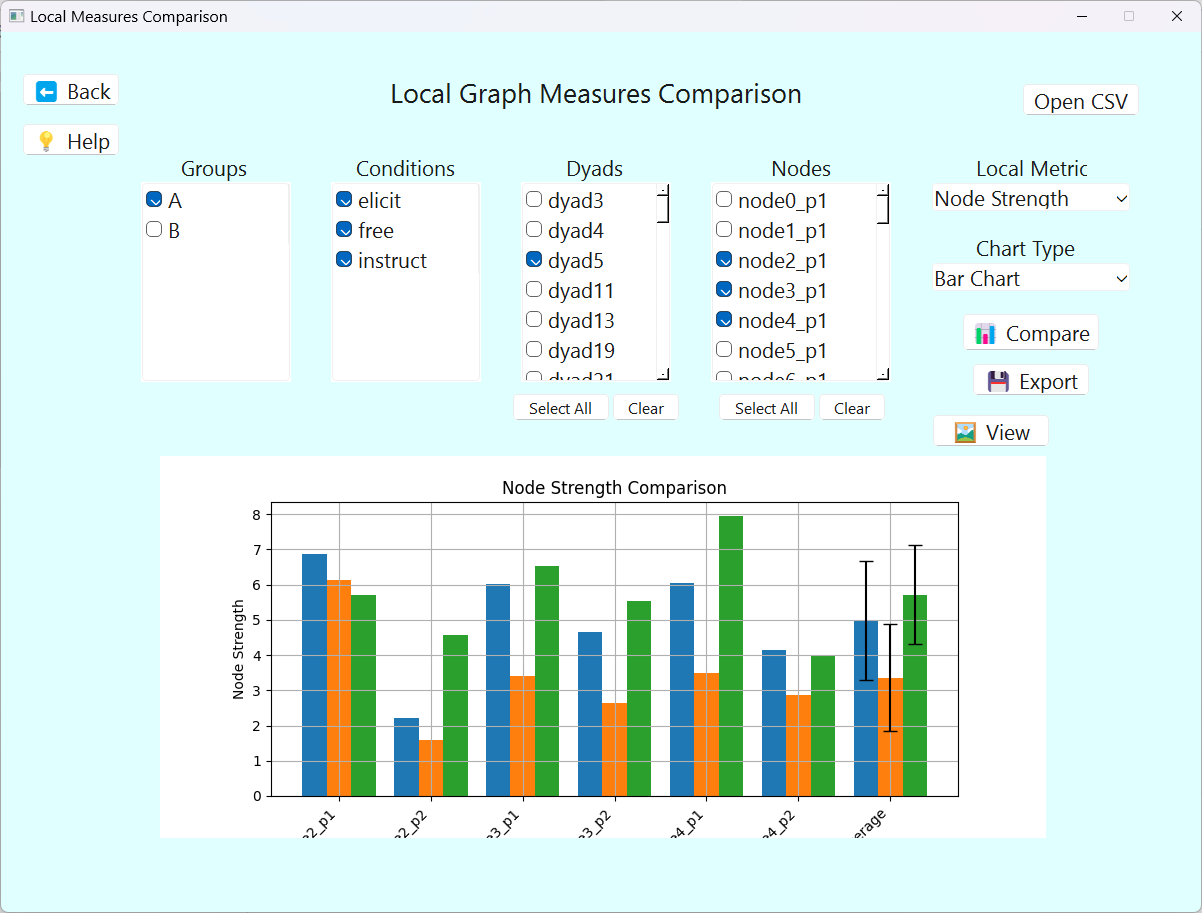
**Figure 1. User Interface for Global Measure Analysis**  
*The interface enables selection of graph metrics, dyads, and conditions to visualize trends in global connectivity.*



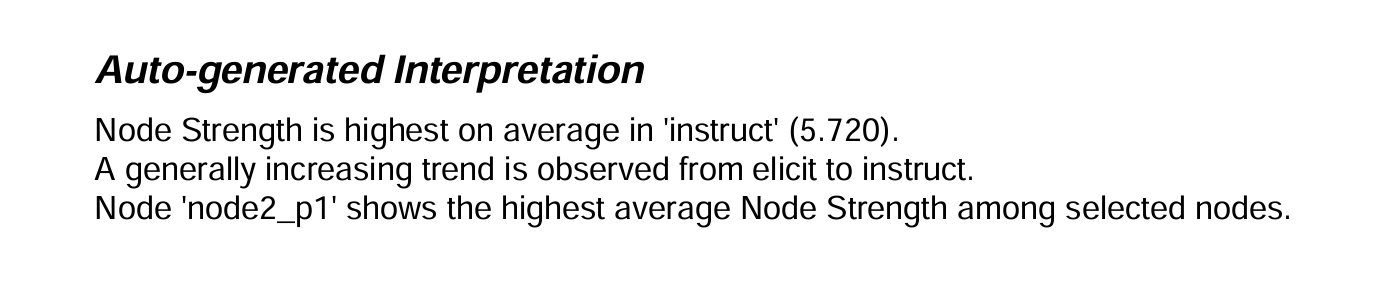
**Figure 2. Visualization of Global Connectivity Metrics***The bar chart illustrates how global graph metrics vary between dyads or conditions, supporting insights into group-level network behavior.*



**Figure 3. Exported Summary of Global Graph Analysis***This PDF output includes the user selection,  
 corresponding bar chart, and an automatically generated interpretation to support research insights.*



**Figure 4. Bar Chart Comparing Node-Level Graph Measures***The plot compares node strength for a fixed dyad or group, showing how connectivity patterns differ between conditions.*



**Figure 4.1. Insight Summary Generated from Node Strength Comparison**  
*This text interprets the bar chart results by highlighting condition-specific differences in connectivity across selected brain regions.*

7.3 System Operation (Flowchart)

Opening Screen

Organize Files

Analysis Pipeline Screen

Generate Correlations

Comparison Screens

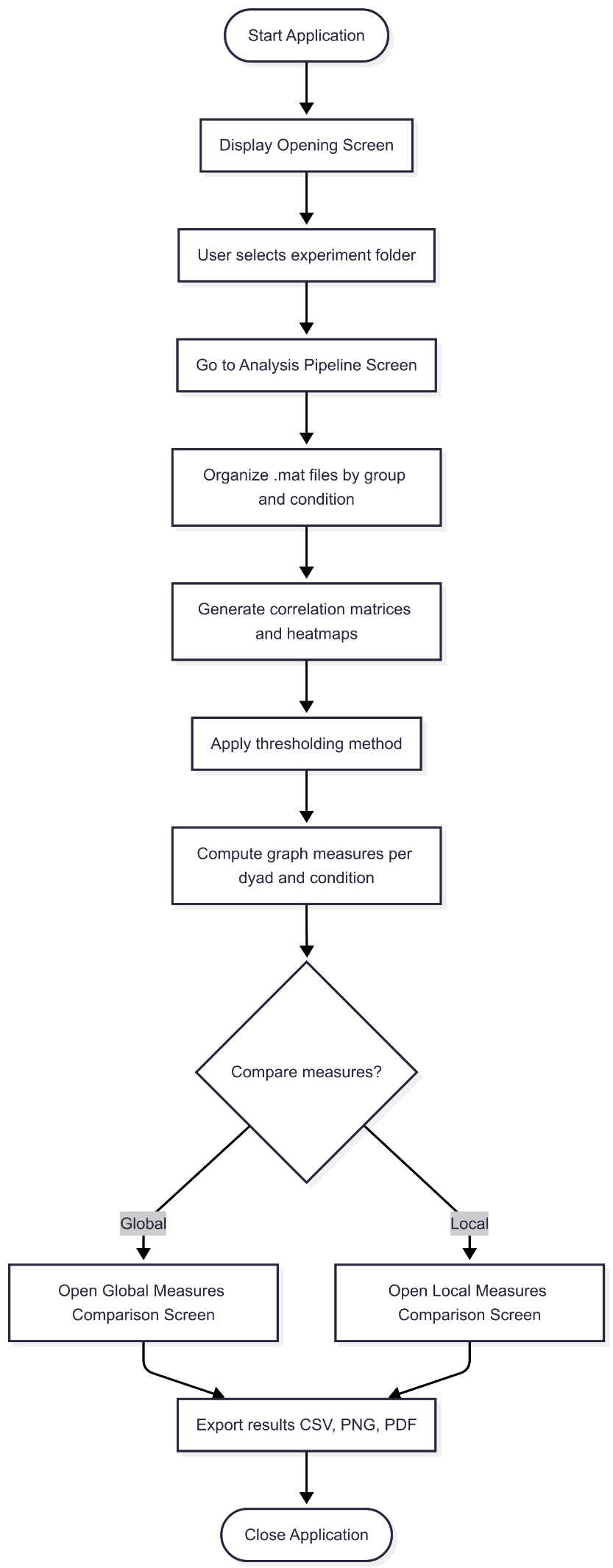
Apply Thresholding

Analyze Graphs

Global Measures Comparison

Global Measures Comparison

Each step includes status updates, progress dialogs, and visual feedback.

A Detailed Overview   
of the System Workflow

**8. Software Development Work**

The development process for the fNIRS Hyperscanning Analysis Tool began with designing a clean, modular graphical user interface (GUI) using PyQt6.   
From the beginning, it was important for me to build an application that would not only function reliably but also guide researchers through the complex analysis pipeline in an intuitive, step-by-step manner. I implemented a multi-screen architecture that reflects each stage of the fNIRS analysis process, from file organization, through correlations matrices generation and thresholding, to the final comparison of graph measures.

A central part of the system was integrating support for MATLAB .mat files, which are commonly used in fNIRS research. I developed conversion logic that reads MATLAB structures and translates them into NumPy arrays suitable for analysis in Python. This allowed seamless integration between domain-specific data and Python’s scientific libraries. With the data prepared, I implemented graph construction and metric computation using NetworkX, enabling the extraction of key measures such as global efficiency, modularity and degree centrality.

Throughout the development, I aimed to ensure that the user experience mirrored real-world research workflows. For example, I realized that there are multiple valid methods for thresholding correlation matrices, each with different implications for the resulting network structure. To reflect this, I implemented three distinct thresholding options: fixed, median-based, and top-percentile, giving users the flexibility to choose the method most appropriate for their research context.

It was also essential to me that the tool support both global and local graph analyses, so I created separate comparison interfaces for each.   
The global comparison screen enables users to evaluate entire network structures across conditions and dyads, while the local comparison screen focuses on node-level insights such as node strength and local efficiency.

To understand which combinations of group, dyad, and condition would provide the most meaningful insights for researchers, I began by studying common experimental designs used in hyperscanning studies. I identified that researchers often want to compare metrics not only across different experimental conditions, but also between dyads within the same group, or even across separate groups. To support this, I designed the comparison screens to allow multiple selections simultaneously: users can select any number of dyads, groups, and conditions through intuitive list interfaces. I implemented internal logic to ensure that the system dynamically loads only the relevant data for the selected combinations, and I structured the comparison functions to generate aggregated results when appropriate.

For example, average values across selected dyads. This flexibility allows users to explore trends within a single group, compare across groups, or isolate effects of specific experimental conditions. Additionally, I integrated chart type selectors, including bar charts, line charts, and radar plots to help users visualize these combinations clearly, depending on their analytical objectives. By enabling these combinations and visualizations, the tool empowers researchers to tailor their comparisons and uncover meaningful patterns in their hyperscanning data.

Overall, the development process was guided by a balance of technical implementation and a deep consideration of the end user's analytical needs. I continuously refined the tool’s features and structure based on my understanding of what would deliver meaningful and interpretable insights to researchers working with hyperscanning data.

As for the technologies, I developed the tool using Python 3.10 as the core programming language, chosen for its readability, wide range of scientific libraries, and strong community support. For the graphical user interface, I used PyQt6, which allowed me to build a responsive, multi-screen desktop application suitable for both novice and advanced users.

To support the analytical and data-processing components of the tool, I integrated several key libraries: NumPy and SciPy for numerical operations and matrix manipulation, pandas for structured data handling and CSV output, and NetworkX for the construction and analysis of complex graphs derived from correlation matrices. For visualization, I employed matplotlib and seaborn, which enabled the generation of clear and customizable heatmaps and plots that support user interpretation. Additionally, for report and image export functionality, I used ReportLab and FPDF, both of which provided flexible tools for generating high-quality PDFs.

I designed the user interface layout using Qt Designer, a visual tool that simplifies the creation of .ui files. These UI definitions were later compiled into Python modules to integrate seamlessly with the application logic. To bundle the application into a standalone executable suitable for distribution, I used PyInstaller, which allowed the entire system to run on Windows machines without requiring users to install Python or manage dependencies manually.

Together, these tools provided a robust foundation for building a professional, user-friendly, and technically capable analysis platform tailored to the needs of hyperscanning researchers.

**9. Challenges and Solutions**

During the development of the fNIRS Hyperscanning Analysis Tool, several technical and algorithmic challenges emerged, each requiring thoughtful solutions to ensure a robust and functional system.

One of the earliest technical challenges involved parsing .mat files, which contain structured MATLAB data. These files often include nested variables and non-uniform formats, which made direct extraction complex. To address this, I implemented a robust parsing mechanism using scipy.io.loadmat, to correctly extract the relevant variables regardless of their structure. Another related challenge was the dynamic pairing of dyads. Since each participant’s data was stored in individual files, I designed a filename convention system and developed a parser that could automatically detect and pair files based on consistent naming patterns, such as dyad1\_Mom.mat and dyad1\_Baby.mat.

Implementing flexible thresholding presented its own difficulties. I needed to support multiple methods, including fixed thresholding, median-based filtering, and top-percentile selection, each with different assumptions and outcomes. I designed the system to allow seamless switching between these approaches, and ensured that the internal logic would remain consistent regardless of the selected method. Another key challenge was related to graph construction. It was critical to accurately distinguish and process intra-brain (within participant) and inter-brain (between participants) connections. This required careful adjustments to ensure that graph metrics such as global efficiency or modularity reflected the true underlying structure of the dyadic brain networks.

From an algorithmic perspective, calculating modularity, centrality, and other graph-theoretical measures posed efficiency challenges, especially when dealing with large or sparse matrices. A major concern was avoiding skewed results caused by low-connectivity or noise-dominated regions in the correlation matrices. To overcome this, I adopted a modular development approach that allowed me to test each algorithm separately. I also validated the correctness and behavior of each implementation using synthetic data where the expected outcomes were known. Furthermore, I incorporated visual tools such as heatmaps and node-based graph plots to support manual inspection and verification of the computed results.

These challenges ultimately strengthened the tool, leading to a more stable, adaptable, and accurate system that is both scientifically rigorous and user-friendly.

**10. Results and Conclusions**

The goals of the fNIRS Hyperscanning Analysis Tool were fully achieved. The system successfully enables complete import of raw fNIRS dyadic data, including automated file organization based on group, condition, and dyad roles. It supports the generation of correlation matrices from synchronized time-series data, followed by user-selectable thresholding methods to refine these matrices for graph construction. The tool also includes a comprehensive graph analysis module that extracts both global and local graph-theoretical metrics, such as modularity, efficiency, degree centrality, and node strength. These results can be explored through interactive comparison interfaces and exported in various formats including CSV, PNG and PDF, enabling researchers to analyze and present their findings with clarity.

Several key decisions were made during development, each guided by practical reasoning and the intended needs of end users. The choice of PyQt6 for the graphical user interface was driven by its modern look-and-feel, cross-platform compatibility, and strong support for professional multi-screen applications. For graph analysis, NetworkX was selected due to its extensive collection of built-in graph measures, intuitive Python API, and robust handling of weighted, undirected graphs, which align well with the properties of fNIRS correlation data. Implementing multiple thresholding methods was also a deliberate design choice, allowing researchers to adapt the analysis to different experimental contexts and statistical assumptions. For example, using fixed thresholds for theoretical consistency or percentile-based methods for data-driven exploration.

In conclusion, the tool provides a full analysis pipeline tailored to hyperscanning data, with the flexibility, transparency, and usability required for real-world neuroscience research.

**11. Lessons Learned**

The development of the fNIRS Hyperscanning Analysis Tool offered valuable lessons in both software engineering and applied research tool design. Early planning decisions, particularly the adoption of a modular architecture, proved essential in managing the system’s complexity. By designing the software as a series of interconnected yet independent screens and functional modules, I was able to test and iterate on individual components without compromising the stability of the overall system. The sequential, step-by-step pipeline not only benefited the user experience by maintaining clarity and orientation, but also aligned naturally with the stages of analysis commonly followed in neuroscience research. Additionally, using Qt Designer to develop the graphical layout accelerated the design process and made it easier for non-programmers to provide feedback and review the UI during development.

Reflecting on the process, one improvement I would pursue in future projects is the earlier separation of backend logic into a standalone processing engine. This would have simplified testing and enabled better reusability across screens. I also recognized the value of structured logging: incorporating a formal logging mechanism from the beginning would have improved the debugging process and made it easier to track the flow of execution and errors, especially during integration of multiple analysis stages.

When evaluating the project against the goals I had set, I am satisfied with the outcomes. From a usability standpoint, the tool is intuitive and includes tooltips, help dialogs, and a comprehensive user guide, all of which contribute to a smooth user experience. In terms of performance, the system handles larger datasets effectively, particularly in the correlation and thresholding steps, which were optimized to maintain responsiveness. Finally, accuracy was verified using synthetic test data, with the resulting graph metrics matching expected patterns based on known network properties. These validations gave me confidence that the tool produces reliable and interpretable results for real experimental data.

User Guide - fNIRS Hyperscanning Analysis Tool

Tool Name: fNIRS Hyperscanning Analysis Tool  
Audience: Neuroscientists, researchers and lab assistants working with hyperscanning data  
Language: English  
Version: 1.0  
Platform: Windows Desktop Application (Python + PyQt6)

**Purpose**

This guide explains how to operate the fNIRS Hyperscanning Analysis Tool, designed to help users analyze dual-participant brain data (hyperscanning) recorded using fNIRS.

The tool enables:

* Organizing .mat files into structured experiments
* Generating correlation matrices and visual heatmaps
* Applying thresholding
* Computing graph theory measures
* Comparing results across conditions and participants

**Getting Started**

Installation Requirements

* Windows OS
* Python 3.10+
* Required Python packages: PyQt6, NumPy, pandas, matplotlib, seaborn, SciPy, NetworkX

Or use the provided executable (.exe) version; no installation needed.

Running the Application

* Executable: Double-click the fnirs-tool.exe file
* From Source Code: Run the opening\_screen.py using Python

**Navigating the Tool**

Opening Screen

* Start - Enter the analysis pipeline
* User Guide - View help text
* About - See project description and credits

Analysis Pipeline Screen

Step 1: Select Folder

* Click "Select Folder"
* Choose a root folder containing your .mat files  
  Files should follow this format: dyad1\_Group\_Condition\_Role.mat

Step 2: Organize Files

* Click "Organize Files" to auto-structure files by group and condition  
  Output: /Experiment/GroupName/ConditionName/\*.mat

Step 3: Generate Correlations

* Click "Generate Correlations" to compute correlation matrices for each dyad
* Heatmaps are also saved for visual inspection

Step 4: Apply Thresholding

* Choose a method from the dropdown:
  + Fixed Threshold: Keep correlations ≥ specified value (e.g., 0.3)
  + Median-Based: Keep correlations ≥ median of all positive values
  + Top Percentile: Keep the top x% of values
* Click "Apply Thresholding" to apply and save results

Step 5: Analyze Graphs

* Click "Analyze Graphs" to compute graph measures:
  + Global Efficiency
  + Modularity
  + Degree Centrality
  + Density
  + Clustering Coefficient
  + Inter-Brain Ratio
  + Node Strength
  + Local Efficiency
* Three CSV files will be created:
  + hyperscanning\_graph\_metrics.csv
  + node\_strengths\_matrix.csv
  + local\_efficiency\_matrix.csv

**Comparison Screens**

Global Measures Comparison

* Compare metrics across conditions, dyads, and groups
* Select:
  + One or more conditions
  + One or more dyads
  + One or more groups
  + One global metric (e.g., Modularity)
* Chart types:
  + Bar Chart
  + Line Chart
  + Radar Chart (for one dyad & one condition)
* Click "Compare" to generate the plot
* Click "Export" to save results (image + data + summary)

Click “Help” for more information and tips.

Local Measures Comparison

* Compare node-level measures (Node Strength, Local Efficiency)
* Select:
  + Conditions
  + Dyads
  + Nodes (channels)
  + Groups
* Choose a chart type: Bar or Line
* Click "Compare" to generate the visualization
* Click "Export" to save results (image + data + summary)

Click “Help” for more information and tips.

**Exporting Results**

You can export:

* CSV Files: Processed metric tables
* Heatmaps: Saved as .png per condition/dyad
* Plots: Saved as .png and embedded into PDF/summary reports
* Reports: Automatically generated summaries include:
  + Metric values
  + Graph images
  + Selections and interpretation hints

**Help & Support**

* Each screen includes a "Help" button with guidance.
* Hover over buttons to see tooltips.
* If a required step is skipped, the system shows a warning.

**Tips**

* Ensure .mat files are properly named (e.g., dyad2\_Control\_Baseline\_Dad.mat)
* Only proceed to comparison screens after full graph analysis
* Use "Radar Chart" to view multiple global metrics for a single dyad/condition
* Select "Average" in bar charts to summarize results across dyads

Maintenance Guide - fNIRS Hyperscanning Analysis Tool

Tool Name: fNIRS Hyperscanning Analysis Tool  
Audience: Future developers, maintainers, or researchers responsible for extending or modifying the system  
Version: 1.0  
Platform: Desktop Application (Python 3.10+, PyQt6)

**Purpose**

This maintenance guide provides all the necessary technical details to:

* Run the system on a new machine
* Understand the software structure and dependencies
* Extend or modify the functionality
* Ensure long-term usability and adaptability of the tool

It is intended for developers who may need to maintain, update, or build upon this project after its initial deployment.

**System Overview**

The fNIRS Hyperscanning Analysis Tool is a PyQt6-based desktop application that allows researchers to:

* Organize raw .mat fNIRS files
* Generate correlation matrices and heatmaps
* Apply thresholding methods
* Analyze graphs using global and local network measures
* Compare results across dyads, groups, and conditions

The tool processes MATLAB .mat files, extracts data, and generates various output formats including CSV, PNG, and PDF.

**Software Components**

| **File** | **Description** |
| --- | --- |
| opening\_screen.py | The main entry point with navigation and help dialogs |
| analysis\_pipeline\_screen.py | Handles all data processing and graph analysis steps |
| graph\_measures\_comparison\_screen.py | UI for comparing global graph metrics |
| local\_measures\_comparison\_screen.py | UI for comparing local (node-level) metrics |
| opening\_screen\_ui.py | UI file for the opening screen (auto-generated by Qt Designer) |
| analysis\_pipeline\_ui.py | UI file for the analysis pipeline (auto-generated by Qt Designer) |
| graph\_measures\_comparison\_ui.py | UI file for the global measures comparison screen (auto-generated by Qt Designer) |
| local\_measures\_comparison\_ui.py | UI file for the the local measures comparison screen (auto-generated by Qt Designer) |
| \*.ui files | UI design files for PyQt6 screens (optional if included) |
| dist/ | Contains the executable if built using PyInstaller |
| docs/ | Project book, user guide, and this maintenance guide |
| requirements.txt | Python dependencies list |

**Operating Environment**

Hardware Requirements

* Windows 10 or later
* Minimum 4 GB RAM
* At least 500 MB of free storage

Software Requirements

* Python 3.10 or higher
* Pip (Python package manager)

Required Python Packages

* Install dependencies using:

pip install -r requirements.txt

* Dependencies:
  + PyQt6
  + numpy
  + pandas
  + scipy
  + matplotlib
  + seaborn
  + networkx
  + scikit-learn
  + reportlab
  + fpdf

**Installation and Execution**

Option 1: Running from Source Code

1. Clone the repository or extract the source files.
2. (Optional) Create a virtual environment:

python -m venv venv

venv\Scripts\activate # Windows

1. Install dependencies:

pip install -r requirements.txt

1. Run the tool:

python opening\_screen.py

Option 2: Running the Executable

If provided:

1. Go to the /dist/ folder
2. Double-click opening\_screen.exe
3. No installation or Python setup is needed

**Testing & Debugging**

* Use .mat test files named in this format: dyad1\_Group\_Condition\_Role.mat
* Test each step of the pipeline independently:
  + Organizing files
  + Generating correlations
  + Thresholding
  + Graph analysis
* Console logs and message boxes provide basic error feedback
* For debugging, insert print statements or use breakpoints in PyCharm

**Extending the System**

Adding New Thresholding Methods

* Edit apply\_thresholding() inside analysis\_pipeline\_screen.py
* Add method to dropdown and implement logic

Adding New Graph Metrics

* Edit the graph analysis section in analysis\_pipeline\_screen.py
* Use NetworkX functions like:
  + nx.closeness\_centrality()
  + nx.betweenness\_centrality()

Adding UI Features

* Use Qt Designer to update .ui files
* Recompile .ui files using:

pyuic6 opening\_screen.ui -o opening\_screen\_ui.py

Exporting New Formats

* Export logic exists in both comparison screens
* Extend export\_results() to support formats like .xlsx, .json, etc.

**Folder & File Structure**

fNIRS\_Hyperscanning\_Tool/

│

├── src/

│ ├── opening\_screen.py

│ ├── analysis\_pipeline\_screen.py

│ ├── graph\_measures\_comparison\_screen.py

│ ├── local\_measures\_comparison\_screen.py

│ ├── \*.ui, \*\_ui.py

│

├── dist/

│ └── fnirs-tool.exe

│

├── docs/

│ ├── project\_book.pdf

│ ├── user\_guide.pdf

│ └── maintenance\_guide.pdf

│

├── requirements.txt

└── README.md

**Versioning and Build Notes**

To create an updated executable:

pyinstaller --name=fnirs-tool --windowed --onefile opening\_screen.py

This generates the executable in the /dist/ folder.  
Make sure all required files (icons, images, etc.) are bundled or copied manually if not embedded.

**Maintenance Recommendations**

* Always back up .mat data files before running new versions.
* Test new features with test datasets before applying to real experiments.
* Use version control (e.g., Git) to track code changes.
* Keep requirements.txt updated if dependencies change.