



COLUMBIA ENGINEERING

Columbia University
Department of Computer Science

COMS 6902 E

From Brain–Computer Interfaces to AI-Enhanced Diagnostics: Developing Cutting-Edge Tools for Medical and Interactive Technologies

Submitted in partial fulfillment of the requirements for the admission to
the degree of Master of Science in Computer Science

By:
Haowen Wei

Supervisor:
Prof. Steven K. Feiner
Prof. Paul Sajda
Prof. Kaveri Thakoor

Date of submission: 05/10/2024

Contents

1	Abstract	4
2	Introduction	4
2.1	Motivation and background	4
2.1.1	Evolving Interfaces in Human-Computer Interaction	4
2.1.2	The Need for Enhanced Diagnostic Tools in Healthcare	5
2.1.3	Integration of Brain-Computer Interfaces and Extended Reality	5
2.1.4	Motivation for Advanced Input Methods in Immersive Environments	5
2.1.5	The Motivation for This Research	5
2.2	Contributions of the Thesis	5
2.2.1	Development of PhysioLabXR	6
2.2.2	Advancements in Glaucoma Diagnosis	6
2.2.3	Innovation in Eye-Tracking Input Systems	6
2.2.4	Integration Across Platforms and Technologies	6
2.3	Theoretical and Practical Implications	7
2.4	Overview	7
2.4.1	Background and Related Work	7
2.4.2	PhysioLabXR: A Platform for Real-Time, Multi-modal Experiments	7
2.4.3	Interactively Assisting Glaucoma Diagnosis	8
2.4.4	SwEYEpe: Eye Tracking and gesture Based Swype-like Input System	8
2.4.5	Discussion	8
3	Background and Related Work	8
3.1	Multi-modal experiments and supporting software	8
3.2	Software tools for real-time experiments	9
3.3	Vision-based computer-aided diagnosis	11
3.4	Deep learning identifying pathological RoI	11
3.5	Eye-tracking informed models for medical images	12
3.6	Eye tracking text input system	13
4	PhysioLabXR: A Python Platform for Real-Time, Multi-modal, Brain-Computer Interfaces and Extended Reality Experiments	14
4.1	Software Features	15
4.1.1	Data Stream API	15
4.1.2	Visualization	16
4.1.3	Visualization	17
4.2	Recording	19
4.3	User Scripting	21
4.3.1	Digital Signal Processing Modules	23
4.4	Software Architecture	25

4.4.1	Concurrency in the main process	25
4.4.2	Serialization interface	27
4.4.3	Multi-process servers for replay and scripting	28
4.5	Use Cases	29
4.5.1	NiDyn Experiment	30
4.5.2	P300 Speller	31
4.5.3	Micro-gesture Interface	31
4.6	Discussion	33
4.7	Limitation and Future Scope	34
4.8	Conclusion	34
5	Interactively Augmenting clinical decisions with expert knowledge-distilled Vision Transformer	35
5.1	Unsupervised RoI detection	35
5.1.1	Unsupervised RoI identification for glaucoma in OCT reports	36
5.2	Expert knowledge distillation	37
5.2.1	Implications for the user study	38
5.3	Perceptual RoI cuing	39
5.4	Perceptual Attention	40
5.5	User studies	43
5.5.1	System Overview	44
5.5.2	Experiment protocol	44
5.6	User interface	46
5.7	Results and Discussion	47
5.7.1	Quantitative Analysis of Glaucoma Diagnosis Accuracy and Confidence	47
5.8	Correspondence Between Participant Fixations and RoI Predictions	48
5.8.1	Qualitative analysis of post-experiment survey	49
5.9	Limitations and Future Directions	50
5.10	Conclusions	51
6	Vison SwEYEpe: Eye Tracking Based Swype-like Input System	51
6.1	Prototype Design	52
6.1.1	Gaze Filter	52
6.1.2	Experiment Paradigm	53
6.1.3	Spell Correction	53
6.1.4	Graphic User Interface	54
6.2	Incorporating Finger Pinch Gestures for Enhanced Usability	55
6.3	Limitation and Future Scope	55
6.3.1	Limitations	55
6.3.2	Future Scope	56
7	Conclusion	56

8 Acknowledgement	57
--------------------------	-----------

A Appendix	68
-------------------	-----------

1 Abstract

This thesis presents a series of studies that explore advanced computational techniques and interfaces in the domain of human-computer interaction (HCI), specifically focusing on brain-computer interfaces (BCIs), vision transformers for medical diagnosis, and eye-tracking input systems. The first study introduces PhysioLabXR [1]¹, a Python platform designed for real-time, multi-modal BCI and extended reality experiments. This platform enhances the interaction in neuroscience and HCI by integrating physiological signals with computational models, supporting sophisticated data analysis and visualization tools that cater to a wide range of experimental needs.

The second study delves into the application of vision transformers to the medical field, particularly for glaucoma diagnosis. We developed an expert knowledge-distilled vision transformer that leverages deep learning to analyze ocular images, providing a highly accurate and non-invasive tool for detecting glaucoma, thereby aiding in early diagnosis and treatment strategies.

The third study explores SwEYEpe, an innovative eye-tracking input system designed for text entry in virtual reality (VR) environments. By leveraging eye movement data to predict text input, SwEYEpe offers a novel method of interaction that enhances user experience by minimizing physical strain and optimizing input efficiency in immersive environments.

Together, these studies contribute to the fields of HCI and medical informatics by providing robust tools and methodologies that push the boundaries of how we interact with and through computational systems. This thesis not only demonstrates the feasibility of these advanced technologies but also paves the way for future research that could further integrate such systems into everyday applications for enhanced interaction and diagnostic processes.

2 Introduction

2.1 Motivation and background

2.1.1 Evolving Interfaces in Human-Computer Interaction

The realm of Human-Computer Interaction (HCI) has seen transformative changes over the past few decades, driven by advances in both hardware capabilities and software innovations. Central to this evolution is the development of interfaces that are not only more intuitive but also capable of capturing and interpreting complex user inputs. This thesis is anchored in the belief that the next frontier in HCI lies in seamlessly integrating physiological data into these interfaces, thereby enabling more natural and efficient user interactions.

¹PhysioLabXR was called RenaLabApp in early stage.

2.1.2 The Need for Enhanced Diagnostic Tools in Healthcare

In parallel to advancements in HCI, the healthcare sector has increasingly turned to technology to address diagnostic challenges. Particularly in the field of ophthalmology, early detection and accurate diagnosis of conditions like glaucoma are critical yet fraught with difficulties due to the subtlety of early symptoms and the complexity of the tests required. Advanced imaging techniques and machine learning models, such as vision transformers, represent a promising solution to these challenges by providing high accuracy in identifying pathological features from ocular images.

2.1.3 Integration of Brain-Computer Interfaces and Extended Reality

The integration of brain-computer interfaces (BCIs) with extended reality (XR) systems offers unprecedented opportunities for both research and practical applications in neuroscience and HCI. By leveraging real-time physiological data, these systems can enhance user engagement and provide novel methods of interaction within virtual environments. This is particularly pertinent in scenarios where traditional input devices are impractical or insufficient, such as in virtual reality (VR) applications where users' physical movements may be limited.

2.1.4 Motivation for Advanced Input Methods in Immersive Environments

As immersive environments become increasingly prevalent, the need for efficient and intuitive input methods grows. Eye-tracking technologies provide a compelling solution by enabling users to interact with digital content through natural eye movements. The development of systems like SwEYEpe, which utilizes eye-tracking for text input, exemplifies the potential of such technologies to revolutionize user interaction in VR and other immersive platforms.

2.1.5 The Motivation for This Research

This thesis is motivated by the potential of integrating cutting-edge computational techniques with user-centric interface designs to significantly enhance the interactivity and functionality of systems across healthcare and HCI domains. By focusing on the development of robust platforms and tools that leverage eye-tracking, vision transformers, and BCIs, this work aims to address the pressing needs for better diagnostic tools, more immersive and accessible interfaces, and enhanced understanding and facilitation of human-computer interaction.

2.2 Contributions of the Thesis

This thesis makes several significant contributions to the fields of human-computer interaction, brain-computer interfaces, and medical diagnostics. The following are the key contributions derived from the individual studies included in the thesis:

2.2.1 Development of PhysioLabXR

The thesis introduces *PhysioLabXR*, a versatile Python platform for real-time, multi-modal brain-computer interface experiments and extended reality applications. Key contributions include:

- A comprehensive platform for integrating physiological sensors with XR systems, facilitating a wide range of neuroscience and HCI experiments.
- Enhancement of data acquisition and processing capabilities, enabling efficient performance in real-time experimental setups through the use of concurrent and parallel programming.
- Open-source availability, which supports a broad adoption and collaborative improvement by the research community.

2.2.2 Advancements in Glaucoma Diagnosis

A significant contribution of this thesis is the development and validation of a vision transformer model for the diagnosis of glaucoma. Contributions include:

- Designing a novel deep learning model that incorporates expert knowledge to effectively identify diagnostic features from ocular images.
- Demonstrating high accuracy in glaucoma detection, which could significantly aid in early diagnosis and improve treatment outcomes.
- Integration with existing medical imaging platforms, enhancing their capabilities with advanced AI tools.

2.2.3 Innovation in Eye-Tracking Input Systems

The thesis presents *SwEYEpe*, an innovative eye-tracking based input system designed for virtual reality environments. Contributions include:

- Development of a novel text entry system using eye-tracking technology that mimics the Swype keyboard, tailored for VR.
- Improvement of text entry methods in VR, offering a more natural and efficient alternative to traditional input methods.
- Extensive user testing and validation, providing insights into usability and accessibility, and setting a foundation for further enhancements.

2.2.4 Integration Across Platforms and Technologies

A cross-cutting contribution of this thesis is the integration of these technologies within the *PhysioLabXR* framework, demonstrating:

- Seamless interoperability between advanced diagnostic tools and interactive systems, which exemplifies the potential for cross-disciplinary innovations in technology and healthcare.
- The potential for real-time, feedback-driven applications in immersive environments, enhancing user experience and system responsiveness.

2.3 Theoretical and Practical Implications

This work not only advances theoretical understanding but also offers practical implementations that can be readily adopted and adapted. It bridges the gap between experimental research and real-world applications, providing robust solutions that can enhance both user experience in HCI contexts and patient outcomes in healthcare settings.

2.4 Overview

This thesis is organized into six main sections, each designed to systematically explore the integration of advanced computational techniques with user-centric interface designs for enhanced diagnostic tools and human-computer interaction. The structure of the thesis is as follows:

2.4.1 Background and Related Work

This section reviews the literature and related works in the fields pertinent to the thesis. It covers:

- **Brain-Computer Interfaces:** Discussing the development and application of BCIs in various contexts.
- **Extended Reality in Medical and HCI Applications:** Examining how XR technologies are being integrated into healthcare and HCI.
- **Eye Tracking Technologies:** Reviewing the development and use of eye-tracking technologies in interactive systems.
- **Vision Transformers in Medical Imaging:** Exploring the application of vision transformers in medical diagnostics.
- **Review of Existing Technologies and Their Limitations:** Critically analyzing current technologies and identifying gaps that this thesis addresses.

2.4.2 PhysioLabXR: A Platform for Real-Time, Multi-modal Experiments

This section introduces PhysioLabXR, a Python platform designed for real-time, multi-modal BCI and XR experiments. It details the software architecture, design principles, integration with physiological sensors, and extended reality systems. The section presents

case studies and experimental results to validate the platform’s effectiveness, discussing the challenges encountered and the solutions implemented.

2.4.3 Interactively Assisting Glaucoma Diagnosis

Section 4 focuses on the application of a vision transformer model for glaucoma diagnosis. It begins with an overview of glaucoma and the challenges in its diagnosis, followed by the development process of the vision transformer model. This section also explores the integration of this model with PhysioLabXR and evaluates the impact of the tool on clinical practices through validation studies.

2.4.4 SwEYEpe: Eye Tracking and gesture Based Swype-like Input System

This section describes the development of SwEYEpe, an eye-tracking based text input system for VR. It details the concept, design, and implementation of the system, along with its integration into PhysioLabXR for real-time processing. User testing and accessibility considerations are discussed, along with potential applications and directions for future work.

2.4.5 Discussion

The final section synthesizes the findings from the thesis, comparing them with existing solutions. It discusses the limitations and challenges faced during the research and proposes directions for future research. This section aims to contextualize the contributions of the thesis within the broader field of HCI and healthcare technologies.

3 Background and Related Work

3.1 Multi-modal experiments and supporting software

Modern experiments in neuroscience and related field continuously put more emphasis on multiple modalities. Though the early-day experiment combining EEG and event markers can be considered multi-input. The potential of multi-modal experiment shines when different physiological measurement marries together. Sometimes streams can act as events to extract epochs from other sensors, such as fixation-related potential (FRP) studies where eye behavior and EEG are combined [2]. Multiple physiological signal can also be combined to enhance the predictive power for application including emotion recognition [3] [4] and movement actuation via sensorimotor rhythms [5]. They also help study how different physiological systems interact like using pupil dilation as a proxy for locus coeruleus activity[6].

Many analysis methods and computational modeling have been proposed specifically for multi-modal data including the analytical approach of FRP deconvolution [7], and the data-driven multi-modal deep learning (DL) models [8] [9]. Efforts have been

taken more recently to design real-time interactive systems (i.e., BCIs) using multi-modal data [10] and supporting software were developed [11][12]. Most commonly the software are networking protocols focusing on timestamp synchronization across different modalities. To provide researchers an open-source, community-driven software tool for the increasing needs in multi-modal experiments, PhysioLabXR offers a complete all-in-one GUI application for visualizing, recording, replaying past experiments, and deploying end-to-end DSP and ML pipelines.

3.2 Software tools for real-time experiments

Software tools for handling physiological data in real-time can be broadly categorized into two groups. The first group consists of proprietary software closely tied to specific hardware, often developed by the manufacturer of the hardware. These device-coupled software tools aim to provide a tailored experience for the types of signals captured by their hardware, offering specialized data visualization interfaces. For example, NIRX Aurora [13] provides a 3D scalp and cortex model for visualizing hemoglobin oxidation. Meanwhile, these software often offer powerful data analysis tools for the respective hardware, such as fixation detection in eyetracking software suites like Tobii [14]. While this line of software excel in their compatibility with the associated device, they often lack support for multi-modal experiments (e.g., pair fNIRS with eyetracking), as the data they capture remain isolated within their ecosystem and cannot be synchronized with other modalities. To address this limitation, researchers often resort to third-party solutions that support integration with all the data sources in their experiment. LSL solves the inter-device compatibility problem as a standardized network protocol for synchronizing data from different devices [11] [15]. ZMQ is another networking protocol popular among researchers for streaming high-throughput data with low latency between processes [16] [17] [18]. These libraries offer bindings with various programming languages, making them versatile choices for various platforms. Nevertheless, when devices lack built-in support for standardized data streaming protocols, researchers must create scripts to relay the data to the network. On the receiving end, LSL comes with a serialization protocol and a standalone software - LabRecorder [11], offering a GUI for monitoring available streams and recording them to the local file system.

At the same time, the data transfer middleware does not cover the ability to visually inspect data streams in real time, which is crucial for many experiments, particularly those involving devices prone to failure and artifacts during operation, such as EEG and fNIRS. Real-time visualization allows experimenters to react promptly to sensor failures and prevent wasting valuable participant time. Additionally, visualizing multiple streams as they come in helps researchers gain deeper insight. They can, for example, observe the stimulus event triggers on the event marker stream while inspecting the EEG streams for corresponding event-related potential [19].

With that, we come to the second group of device-independent graphic software tools; many have been proposed over the years to assist researchers and practitioners in

conducting experiments involving real-time data collection and feedback. These tools often utilize the networking protocols mentioned earlier for data transfer and may include native plug-ins for specific devices, such as Brainflow for OpenBCI devices [20]. Some offer features such as real-time data processing modules as in NeuroPype [21], and MNE S [22]. At the time of this article, a majority of popular experiment platforms such as OpenVibe, MNE Scan, and FieldTrip are primarily built using statically compiled languages (e.g., c and C++) or proprietary languages (e.g., MATLAB). While tools such as OpenVibe and NeuroPype offer scripting interfaces where users can add custom scripts in Python, it is worth noting that Python, despite its rising popularity as a programming language [23] for its ease of use and versatility, is less favored by developers as a backbone language for experiment platform that requires high precision and high data throughput. Although not impossible, implementing an experiment platform in an interpreted language like Python necessitates significant optimization efforts to match the performance level of platforms built with a compiled language. Nevertheless, one of the objectives that PhysioLabXR seeks to address is to prioritize ease of understanding, enabling users to expand and build upon its open-source foundation. As a result, Python dominates PhysioLabXR’s implementation from frontend GUI to the backend servers without sacrificing performance owing to its concurrent runtime architecture. Users are welcome to use PhysioLabXR as a scaffold and leverage Python’s extensive APIs to shape the platform according to their needs.

Custom scripting interface is a feature offered by most experiment software platforms, allowing users to build flexible data processing pipelines. OpenVibe, for instance, is known for its graphical pipeline builder while also allowing the addition of Python and LUA scripts [24]. iMotion uses a platform-specific iMotion scripting language to achieve the same goal [25]. With PhysioLabXR, users write Python scripts to interact with any data stream and stream out the processed results with built-in I/O modules. This flexibility allows users to design closed-loop systems, including deploying machine learning models and sending predictions to and from PhysioLabXR.

Platform	OpenVibe [24]	BCILAB [26]	Neuropype [21]	iMotion [27]	MNE Scan [22]	PhysioLabXR[1]
Is Open Source	✓	✓†	✓		✓	✓
Programming Language	C++	MATLAB	-	-	C++	Python
User Scripting	✓		✓	✓		✓
Custom Sensor Interface		✓				✓
Audio/Video Interface			✓	✓		✓

Table 1: Real-time software tools for physiological experiments or applications. This table provides a non-exhaustive list of tools that have a dedicated GUI and are not specific to a particular device. The entries included here are chosen as representative examples with the features they offer. †: The cited software itself is open-source, but it may rely on proprietary libraries (e.g., BCILAB builds upon proprietary MATLAB, requiring additional licenses for end-users).

3.3 Vision-based computer-aided diagnosis

Computer vision has significantly impacted medical processes involving imaging technologies, providing aids to tasks such as surgical planning and disease diagnosis [28] [29]. In the domain of vision-based CAD, models have been proposed to improve the interaction with these CAD systems by integrating Large Language Models (LLM) to provide a natural language interface to supplement vision-based CAD models, offering improved interactivity [30].

On the other hand, CAD tools in ophthalmology prioritize automatic disease detection [31, 32, 33]. Interactive CAD tools for ophthalmic imaging, such as fundus and optical coherence tomography (OCT) images, remain limited to targeting classification [34] [35] and segmentation [36]. Nevertheless, advanced automatic screening tools provide clinicians with minimal explanation regarding the decision-making process of the models, limiting their clinical utility [37].

Authors in [38] attempt to address the gap in interactive tools with a web application that supports uploading fundus images for glaucoma classification. This system employs gradient-weighted class activation mapping (Grad-CAM) to visualize the diseased area, offering a more intuitive understanding of the diagnostic process. The authors focus more on the supporting model in the backend as have most studies in this field. Further investigation is needed to validate the tool's clinical relevance.

3.4 Deep learning identifying pathological RoI

Deep learning methods have recently come to center-stage for automatic identification of regions of interest (RoI) in medical images [39] [40] [41]. This capability is not only essential for accurate classification but also for providing insights into disease-specific areas, thereby aiding clinicians in making informed decisions [42] [43] [44].

A number of studies have demonstrated the use of deep learning for detecting and analyzing RoIs in OCT images. For instance, Chen et al. [45] developed an object detection model that employs class-activation maps (CAM) to locate diseased areas in OCT images. Wang et al. [46] focused on automatic glaucoma screening with OCT images and showed qualitative analysis of the discriminated regions derived from CAM in OCT B-scans, whereas a CNN model proposed by Kim et al. used gradient-weighted CAM (GradCAM) to localize glaucomatous damage in fundus images [38]. Fang et al. [47] introduced an attention-based convolutional neural network (CNN) to detect lesions in OCT images, where salient feature extraction significantly aids classification. Similarly, Hassan et al. [48] proposed a more generalized scanner-independent model that combined segmentation and classification. These methods underscore the importance of RoI identification in enhancing the clinical value of deep learning for OCT image analysis.

Transformers, recently gaining traction in medical imaging, have been explored for their

efficacy in RoI identification. Gao et al. [49] introduced UTNet, a hybrid transformer architecture designed for this purpose. Additionally, Hossain et al. [50] examined the application of Vision Transformers (ViT) for RoI identification, providing a new perspective in this domain.

While Gao and Hossain’s models explicitly predict the RoI, attention rollout is a method to implicitly extract discriminating regions from transformers from the dot product between the query and the keys [51] [52]. Attention rollout is similar to CAM in the sense that no RoI information is provided during the training process, and the ability to locate relevant features is naturally from other training objectives such as classification [53]. Mbakwe et al. [54] conducted a qualitative analysis using attention rollout on chest scans. Their work with the CheXRelFormer model demonstrated the model’s capability in focusing on pertinent regions.

Despite the promising results of automatic RoI detection in medical images, more studies are needed to translate ML-based CAD system into clinically validated protocols, and make them understandable to clinical practitioners [37]. The challenge is also noted by Chen et al., who point out the lack of clinical translatability of these approaches and question whether RoI identification truly aids ophthalmologists in clinical decision-making [45]. Salas et al. [55] designed a study in which clinicians are asked to classify OCT reports, while being presented with the predicted probability of glaucoma and a Grad-CAM from the CNN model. They found the clinicians reported increased confidence when the information from the model was present. This study represents initial steps toward bringing ML-based RoI identification for medical image out of the realm of qualitative observations to formal user studies.

3.5 Eye-tracking informed models for medical images

Recent advancements in medical image analysis have increasingly incorporated data-driven approaches like transfer learning and feature fusion to enhance model performance and accuracy [56]. Among them, integrating human expertise into machine learning models has been an emerging area of research. A particular type of integration hinges upon distilling clinical expertise. Li et al. asked ophthalmologists to note the important features in fundus images by mouse clicks and trained a glaucoma detection model incorporating the ophthalmologists’ “attention” [57]. As an implicit proxy for attention, eyetracking data generated tacitly as clinicians analyze medical images has demonstrated significant clinical relevance [58]. Eye-tracking data from clinicians encapsulates expert knowledge in identifying salient features in medical images, such as in OCT scans. Moreover, this expert knowledge can be a vital ingredient in delineating clinically-relevant features for medical image datasets that are limited in quantity such as OCT, making eye-tracking a valuable tool for informing and enhancing computational models [59] [60].

Several innovative approaches have been developed recently to utilize eye-tracking data

for the training of models that process medical images. One such approach is the Radio-Transformer [61], which employs an Intersection-with-Union loss function. This method designates the student network to learn from a teacher network trained on radiologists' eye-tracking data, enhancing the model's ability to identify regions of interest in chest scans. Patra and colleagues (2019) present a sonograph model, another example of a teacher-student knowledge-transfer framework. This model benefits from sonographer gaze data to pinpoint RoIs and achieves comparable results with models that have many more parameters [62]. Stember and colleagues (2019) utilizes segmentation masks generated from eye-tracking data to train models that predicts semantic segmentation on radiology images; they find that the model trained on eyetracking masks is comparable to that trained with hand annotation [63]. In the realm of OCT images, Kaushal and colleagues (2023) present the fixation-order-informed ViT, where the input image patches to the model are chosen based on the fixation-order of ophthalmology resident gaze data [64]. The Ophthalmologist-Gaze-Augmented ViT introduced in this work forms the basis for the interactive ViT described in subsequent sections.

3.6 Eye tracking text input system

The integration of eye-tracking with virtual reality (VR) environments has further enhanced the capabilities and potential applications of gaze-controlled text input systems. Previous works, such as those by Kumar et al. [65] and Hummer et al. [66], have demonstrated the feasibility of swipe-like keyboards that utilize screen-based eye-tracking systems. These systems interpret the gaze path to predict text input, requiring explicit user actions to initiate the swipe. This method compares favorably with traditional swipe keyboards on smartphones by offering a hands-free alternative that can significantly benefit users with physical disabilities.

Building upon these foundations, the development of such systems within VR platforms has taken a significant step forward. In VR, the immersive environment provides a more intuitive and spatially aware interface for eye-tracking inputs. This integration not only enhances accessibility but also improves the interaction fidelity and comfort for the user. The VR setting allows for a more natural calibration and tracking of eye movements, which, coupled with advanced predictive text algorithms, can deliver an even more efficient and user-friendly experience. By leveraging the unique capabilities of VR, these advanced gaze-controlled text input systems ensure that communication is more inclusive and accessible, further refining the accuracy and efficiency of gaze-based interactions to create systems that are both responsive and comfortable for the user.

4 PhysioLabXR: A Python Platform for Real-Time, Multi-modal, Brain-Computer Interfaces and Extended Reality Experiments

2

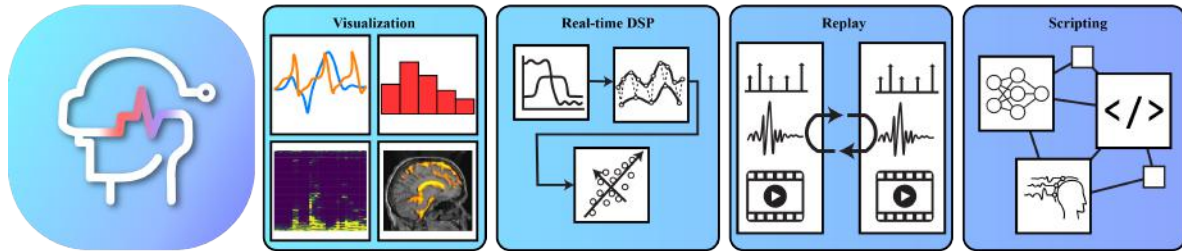


Figure 1: PhysioLabXR and its major features for real-time, multi-modal experiments, including various visualizations, digital signal processors, replay, and user scripting.

PhysioLabXR is a Python-based open-source software platform for developing experiments for neuroscience and human-computer interaction (HCI) that involve real-time and multi-modal physiological data processing and interactive interfaces. PhysioLabXR provides native support for data sources such as electrophysiological sensors (e.g., EEG,

2

Haowen "John" Wei co-founded the PhysioLabXR project with Ziheng "Leo" Li. His contributions were pivotal, including leading the creation of the platform's core components like its data visualization capabilities, graphical user interface, and device integration system. He designed an intuitive user interface that simplified user interactions and streamlined experiment configurations. John also developed a sensor fusion framework, custom real-time visualizers for TMS and fMRI data, and built-in DSP modules to enhance the platform's data processing capabilities. He crafted multiple brain-computer interface paradigms, including the P300 Speller, and compiled extensive documentation to aid users and promote educational outreach.

Ziheng "Leo" Li, as a co-founder, played a central role in building the platform from scratch. He was the original visionary behind the idea for the PhysioLabXR project. His contributions included significant GUI refinement and creation of the entire scripting interface, which allowed for advanced customization and functionality within the platform. Leo's work on the recording tab and the preset system facilitated user interaction by saving user settings during platform use. He also refined the stream settings widgets and added functionalities to image and spectrogram displays, enhancing the platform's visual data representation capabilities.

Ziwen Xie focused on developing the official website and authored documentation on integrating PsychoPy with PhysioLabXR, thereby supporting the platform's usability in psychological experiments.

Yunxiang Peng enhanced the platform by adding the .xdf saving format to the recording tab, which is essential for data storage and analysis. He also created builds for Windows and iOS systems, thus extending the platform's accessibility across different operating systems.

Together, the team's diverse contributions have solidified PhysioLabXR as a robust and versatile platform, supporting a wide array of brain-computer interface applications and extended reality experiments.

For more detailed information about individual contributions, please refer to the [GitHub contributors page](#).

EMG, and EOG), fNIRS, eye trackers, cameras, microphones, and screen capture, and implements the popular data transfer protocols Lab Streaming Layer [LSL] [11] and ZeroMQ [ZMQ; [67]]. It features multi-stream visualization methods, real-time digital signal processing (DSP) modules, support for recording and replay experiments, and a Python-based scripting interface for creating custom pipelines.

PhysioLabXR has an architecture optimized through concurrency and parallelism to ensure efficient performance. We provide a set of detailed tutorials covering all features and example applications, such as a P300 speller with a Unity frontend [68] and a mental arithmetic experiment interfacing with PsychoPy [69]. An accompanying set of benchmarks demonstrates the ability of PhysioLabXR to handle high-throughput and multi-stream data reliably and efficiently. Published use cases show its versatility for VR and screen-based experiments [70][71] and sensor fusion studies [72].

4.1 Software Features

In this section, we cover the key features of PhysioLabXR including visualization, recording, replaying, and scripting, DSP modules, and device connectivity. Each feature is designed to address user needs in different aspects of experiments involving real-time data collection and processing. We provide here a brief overview for each feature: 1) the **data stream API** establishes connection with data sources, through native plugins, network layers (LSL or ZMQ), or user-defined stream API. 2) PhysioLabXR provides various real-time **visual representations** to help users better understand their data. 3) **Recording** lets users capture experimental data in real-time and export them for further analysis. 4) **Replay** enables users to play back data streams from past experiments, and if needed, test their data processing script and algorithm in real-time as if the experiment is currently running. 5) Scripting allows the user to add custom script with modular input-output (IO) API. Users can incorporate data from various sources into their scripts and create new data sources as needed. 6) Modular DSP is another powerful feature in PhysioLabXR that allows users to apply various predefined signal processing algorithms from a wide range of applications to their data streams. All these features can be used in combination with each other, enhancing the overall potential of PhysioLabXR. For example, filter can first be applied to EEG data, and the user can visualize the filtered stream that can simultaneously drive a BCI with custom script.

Next we will discuss each feature and their usage in experiment scenarios with multiple modalities. For a more comprehensive guide on how to use PhysioLabXR, we refer user to the [online documentation](#), which includes step-by-step instructions on how to utilize each feature to its fullest potential.

4.1.1 Data Stream API

PhysioLabXR offers a variety of ways to connect to a wide range of hardware. The stream API is modularly designed: classes extending `StreamInterface` are responsible

for producing data and connection with external data sources. PhysioLabXR has the following built-in stream interfaces:

- **network:** connects to data streams broadcasted through local area network. PhysioLabXR supports two network libraries: LSL [11] and ZMQ [67], both widely adopted by researchers and practitioners for streaming experiment data. Users may add LSL streams to PhysioLabXR by their stream names or types. For zmq, PhysioLabXR supports the subscriber-publisher socket pattern, because this pattern is more scalable when the cardinality of data is high and less error-prone in case of either the publisher or subscriber process crashes. PhysioLabXR robustly isolates the threads where the network interfaces reside, and notifies the user when network errors occur.
- **video device:** PhysioLabXR is designed to work seamlessly with two types of video data sources: screen capture and webcam. Any webcams and monitors connected to the host computer is automatically recognized with a stream interface ready for use. When recorded, streams from video devices are saved in uncompressed frames with timestamps synchronized to all other streams. Note that while the video device API is specifically for screen capture and devices recognized as webcams, the user can plot any stream as images.
- **audio device:** PhysioLabXR supports audio input devices, such as microphones, through its built-in interface. The software automatically connected audio input devices on the host computer. The names of the available audio devices are listed alongside other data streams, which the researcher can add. They can also customize the device in ways including editing audio device parameters (i.e., sampling rate, number of channels, buffer size), and adding digital filters. This feature enables the user to incorporate audio data of desired properties into their real-time experiments within PhysioLabXR.
- **native device:** PhysioLabXR offers support for a variety of physiological recording and environment sensing devices, including: PupilLabs and Tobii eyetrackers, BioSemi ActiveTwo, OpenBCI, ABM B-Alert, NIRX fNIRS, and TI mmWave sensor IWR6843 [72]. The experimenter can easily integrate any combination of these devices into their multi-modal experiment. [documentation](#)

Custom Stream Interface In cases where a specific device is not yet natively supported by PhysioLabXR, users have the flexibility to add custom data interfaces. A comprehensive guide on how to establish connections to any device can be found in the Data Stream API page of the [online documentation](#).

4.1.2 Visualization

Visualization is an essential aspect of data handling software, and PhysioLabXR follows this convention. The primary entry point of PhysioLabXR is its *Stream* tab, where users can connect to data sources by specifying the required parameters for each stream type

(e.g., stream name for LSL streams, as seen in section 4.1.1). Once added, the streams' widgets are displayed in the *Stream* tab, where users to customize their visualization settings. Next, we will cover several UI features to make the data visualization more accessible to the user.

4.1.3 Visualization

Visualization is an essential aspect of data handling software, and PhysioLabXR follows this convention. The primary entry point of PhysioLabXR is its *Stream* tab, where users can connect to data sources by specifying the required parameters for each stream type (e.g., stream name for LSL streams, as seen in section 4.1.1). Once added, the streams' widgets are displayed in the *Stream* tab, where users to customize their visualization settings. Next, we will cover several UI features to make the data visualization more accessible to the user.

Grouping and Separating Channels Channels within the same stream can be organized into groups thus plotted in separate plots. This feature is particularly helpful when the channels measure different phenomena with different vertical scales. For example, given a stream of a six-degree-of-freedom inertia-measurement unit, the user can group the gyro-axis and accelerometer-axis channels together. Some device streams include the timestamps in their data frames that are measured in epoch time that reaches up to $1e9$ if represented by an integer in seconds. If plotted as line char, the different scales will flatline the more important data channels when plotted into the same graph. Figure 2 illustrates an example of separating different channels into plot groups (e.g., the BioSemi stream on the upper left where EEG and ECG are plotted in individual plots). Furthermore, the user have the flexibility to hide specific groups or channels from the plots, reducing clutter and optimizing the plotting area (e.g., the meta information is hidden in figure 2).

Organizing multiple streams PhysioLabXR is designed to cater to experiments that involve multiple data sources, aligning with the demands of modern neuroscience and HCI research. While the software can also be utilized for single data source experiments, many of its features truly shine when handling multiple streams concurrently, such as EEG with eyetracking and video with audio. To help researchers inspect multiple streams simultaneously, PhysioLabXR offers a convenient feature: each stream's plotting widget can be popped out as a separate window from the main interface. Figure 2 (a) shows an example where four streams are organized in a tiled layout. Moreover, PhysioLabXR is designed to scale when researchers opt to use large display and multi-monitor setups, as shown in the triple screen setup depicted in figure 7 for a neuroscience experiment use case). Such configuration is particularly advantageous for experiments involving a substantial number of input feeds. Researchers can personalize the arrangement of stream widgets by detaching them from PhysioLabXR's main window and positioning them across different screen areas according to their specific needs for visually inspecting each data source.

A. Visualization of the data sources



B. Arranging channels for each data source

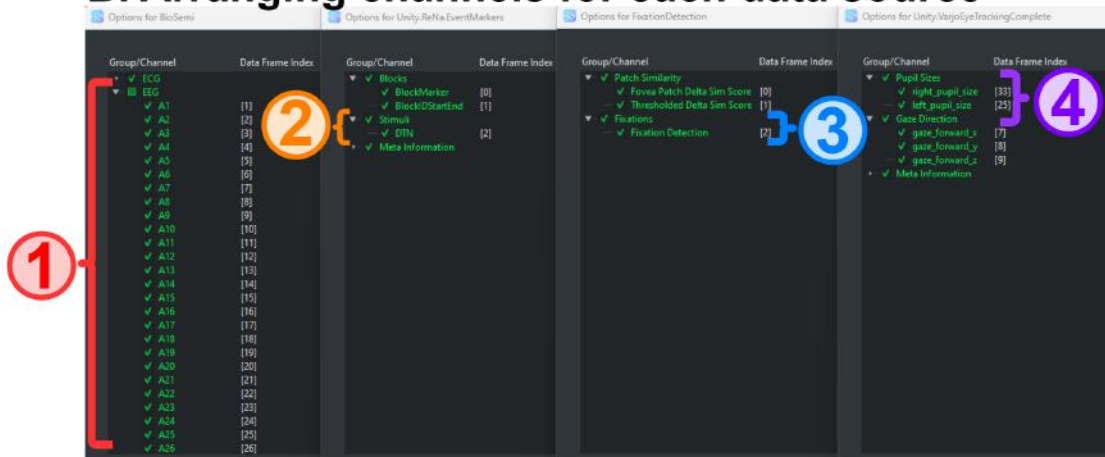


Figure 2: Multi-stream visualization of PhysioLabXR in an oddball experiment. The app is connected to four streams: BioSemi [73] for EEG, Varjo HMD [74] for eyetracking and pupillometry, event markers from Unity [68] as the stimulus presentation software, and user-defined fixation detection. To facilitate monitoring multiple streams at the same time, the user pops the stream widgets from the main app window and arranges them in a grid layout on the desktop, as shown in (a). In addition, using the tree widgets in stream options, the user groups the channels based on what signal they represent, as shown in (b). Examples of grouped channels in this layout include: 1) grouping the 64 EEG channels and the ECG channel for better visual representation, 2) using the distractor/target/novelty (DTN) channel in the EventMarker stream to represent different stimuli types in the oddball paradigm (target, distractor, novelty), 3) displaying fixations and saccades in the fixation detection channel, and 4) separating eyetracking, pupil sizes, and gaze direction into distinct plots, with meta information hidden to prioritize more visually relevant signals.

Choosing visualization method PhysioLabXR offers a variety of visualization formats at the researcher’s disposal to help best capture a signal’s characteristics. Fig. 3 gives an experiment setup where all the plot types are used concurrently. The current release (v0.1.0) supports the following visualization formats:

- **line chart** is effective for disclosing temporal trends. It is a commonly used format for time series such as most neuroelectrical signals, eyetracking, and other physiological signals when the researcher is interested in observing changes in amplitude over a period.
- **bar plot** plots one frame of data at a time, making it suitable for scenarios where temporal changes are less relevant. For example, researchers can employ this format to visualize the predictions generated by a machine learning model, with each class’s probability represented by a distinct bar.
- **Image**, 2D or 3D with color channels can be plotted by data sources as images accordingly, such as those obtained from medical imaging devices.
- **Spectrogram or FFT** prove valuable for visualizing the spectral content of a signal. While the FFT plots the present power spectral density, the spectrogram shows its variation over time. They are commonly employed in the analysis of EEG and audio data. Researchers can leverage PhysioLabXR’s spectrogram/FFT visualization to gain insights into the frequency components and their changes across time.

4.2 Recording

A recurring use case of data handling software in HCI, neuroscience, and related fields is exporting the streamed data for offline analysis later. PhysioLabXR’s recording interface is designed to record multi-stream, high-throughput data. PhysioLabXR’s native .dats data format stands for *Dictionary of Array and Timestamps*, where the stream name and data are stored in key-value pairs. Each key’s corresponding value is two arrays: the data matrix (with shape $N_{channel} \times N_{timesteps}$) and the timestamp array (with shape $N_{timesteps}$). The .dats files can be loaded back into Python or MATLAB as a dictionary for downstream analysis (see listings 1 and 2).

Listing 1: Example Code for Loading Recorded Data

```
from rena.utils.data_utils import RNStream

stream = RNStream("path/to/data.dats")
data = stream.stream_in()
```

Listing 2: Example Code for Loading Recorded Data in Matlab

```
fn = "path/to/data.dats";
data = stream_in(fn);
```

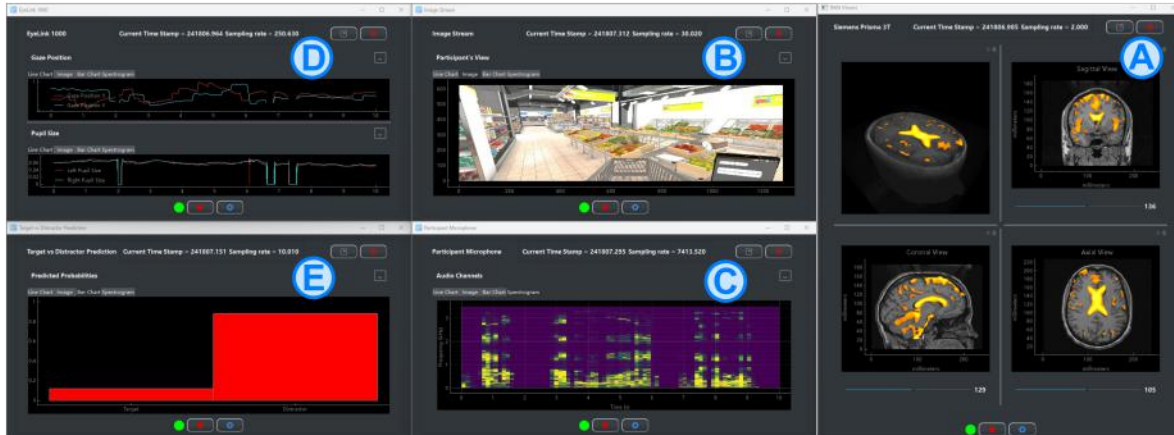


Figure 3: Example use case of PhysioLabXR in a memory formation and retrieval experiment involving realtime processing of pupillometry and fMRI streams. This example demonstrates the diverse visualization options provided. In this experiment, the participant is asked to navigate a virtual shopping mall and respond verbally during their task. (A) The 3D fMRI visualizer shows fMRI data streamed in real-time. (B) The experimenter uses PhysioLabXR to monitor and record the scene from the participant’s first-person view while they perform the task. (C) The participant’s speech is captured using a microphone connected to the software that visualizes the audio data as a spectrogram. (D) Eye movement and pupillometry data are recorded through an eye tracker outside the scanner that receives the participant’s eye image via a mirror. The time series of the eye-tracking data are plotted in a line chart. (E) Simultaneously, an ML model deployed through PhysioLabXR’s scripting interface predicts from the fMRI data and pupillometry if a target memory is retrieved, with the two-class inference result visualized as a bar plot.

In addition to the native data format, users can choose other save formats to best fit their working environment, eliminating the need to install additional software. PhysioLabXR supports save formats, including Comma-separated Values (CSV), Extensible Data Format (XDF), Pickle (Python), and MAT-files (MATLAB).

Synchronization between streams Synchronized timestamp is a crucial feature, especially for multi-stream experiments when multiple data sources are involved. Following the standard of many widely-used experiment data handling interfaces such as LSL [11], PhysioLabXR creates timestamps for recording data relative to the system’s hardware local clock. PhysioLabXR includes three ways of timestamping: 1) **Internal Timestamping**: A timestamp is attached to each frame (single time point) upon the frame’s retrieval. If multiple data frames are received at one pull data cycle, the timestamps are interpolated for these frames between the current and last time when data was received. PhysioLabXR uses this method as the default timestamping for data streamed via ZMQ, audio/video devices, and user-defined sources (see section 4.1.1 for creating custom data sources). 2) **LSL Timestamping**: if the data is streamed via LSL, PhysioLabXR prioritizes the timestamps that are automatically attached to each data frame by LSL as they are pulled from inlets [11]. 3) *Override with Channel Timestamp* Users have the option to select a channel within the data stream and override the other two timestamp types with the values from that channel. This method is helpful when some devices generate more precise timestamps than the computer’s local clock and the experimenter prefer to use the device-generated ones. On the other hand, as long as the device timestamps are streamed as a data channel, they will be available in the data matrix, and the user will have both the device timestamps and the other timestamps (one of the two discussed above depending on whether the data stream comes from LSL).

The internal timestamps are guaranteed to be synchronized across streams. If streams from different hardware (e.g., fNIRS and eyetracking) are captured and sent to PhysioLabXR via LSL, they are also synchronized with the local clock timestamps. On the other hand, if the data sources came from different computer systems and are streamed via LSL, the timestamps will need to synchronize post-hoc by computing constant offset between different computers’ local clocks.

4.3 User Scripting

The scripting interface, or RenaScript, is one of the most extensible features PhysioLabXR offers. It empowers researchers to run user-defined Python scripts, providing full agency for creating and deploying custom data processing pipelines. With Python’s versatility and vast array of fast-growing open-source libraries, users are encouraged to creatively explore and experiment with novel applications such as close-loop neuro-feedback. Users can implement DSP algorithms, run ML models with real-time data streams, or communicate with external applications such as a cloud-computing platform through Python APIs.

With the script widget as part of the PhysioLabXR's GUI, users can customize their script by tuning its attributes to influence the behavior of the data processing pipeline. The following list shows the editable attributes and what they control:

- *Inputs* defined using the data stream API (see section 4.1.1) can be added to any scripts. Added *inputs* can be accessed in the script using `self.inputs[my_stream_name]`. Accessing the input gives a data matrix and the timestamp vector. The data matrix is of shape $(f_{nominal} * T_{bufferduration}, N_{channels})$, where $f_{nominal}$ is the nominal sampling rate, $T_{bufferduration}$ is the amount of data in seconds the buffer contains and $N_{channels}$ is the number of channels for this stream. The timestamp vector has $f_{nominal} * T_{bufferduration}$ items, each corresponding to a row of entries in the data matrix.
- *Input buffer duration* defines $T_{bufferduration}$, and determines the size of the circular buffer that holds inputs. For example, the user defines an input to a script that is an EEG stream with 2048Hz nominal sampling rate and 64 channels. If the *buffer duration* is set to 1 second, the data matrix obtained from calling `self.inputs[my_stream_name]` will have shape (2048, 64).
- *Run frequency* defines F_{run} . It controls how often the *loop* function in the user script is called per second.
- *Outputs*: user can create output streams within scripts by adding *outputs*. An *output* is defined by the output stream name and its number of channels. An output stream is broadcasted on the network through LSL or ZMQ per the user's choice, allowing it to be visualized the same way as an external data source in the stream tab. For example, if the experimenter has an ML model decoding the amplitude of frequency bands for a steady state visually evoked potentials (SSVEP) tic-tac-toe game [75]). Each of the nine cells in the tic-tac-toe game is flashing at a different frequency, and a participant needs to gaze at the cell where they wish to place their piece. The experimenter can create an output stream called "SSVEP Band Power Classification" with nine channels. The probabilities can then be plotted as bar plots (see section 4.1.3 for various plotting formats) to monitor the model's predictions visually. Meanwhile, a script may take output streams from another script as input, thus creating a cascading pipeline. The same experimenter can create a script to receive the SSVEP predictions and drive the tic-tac-toe game.
- *Parameters* are useful in cases when a pipeline requires variables that the user wishes to alter during runtime (e.g., probability threshold for ML models) or variables that depend on the host computer (e.g., the path to a pre-trained ML model) when the researcher wants to avoid hardcoding these variables into the script. *Parameters* in PhysioLabXR's scripting interface meet these needs. In the script widget, the user can define *parameters* by their names, data type, and values. Then throughout the script, *parameters* are accessible using its

name: `self.params[my_parameter_name]`. Changes to parameter values are also reflected immediately in real time while the scripting is running.

In addition to the attributes the user can edit in the GUI, the core of a `RenaScript` is the Python script containing three abstract methods that the user can override to specify the script's behavior: *init*, *loop*, and *cleanup*. *init* is called once when the start button in the widget is clicked. This function is intended for the user to instantiate objects to be used later in the primary duty-cycle function: *loop*. Users may add routines such as defining variables, loading pre-trained parameters for an ML model from the file system, and connecting with servers and cloud platforms. After *init* concludes, `PhysioLabXR` will call *loop* F_{run} times per second. *loop* is presumably the most often called function and uses the most processor time to continuously process *input* data and communicate with other processes via the *outputs*. At the same time, the user can tune the behavior by modifying the *parameters*. The *cleanup* is called when the stop button is clicked. Here, the user may free occupied memory, disconnect hardware ports and close any opened file system resources.

Script Performance When a script is actively running, `PhysioLabXR` helps the user monitor its performance with two key metrics: 1) *loop per second with overhead* is the actual number of loops the `PhysioLabXR` is able to call per second. User-defined frequency is the upper bound, but the actual loop frequency may be lower due to the inclusion of long-running functions in the loop, causing the run time of a loop call to exceed the designated period ($1/F_{run}$). A script running slower than expected maybe also be caused by the overhead, which is time spent in inter-process communication between the main process and the scripting server. 2) *average loop call time* measures the average time the loop function takes. This metric does not include `PhysioLabXR`'s internal overhead and solely depends on how the user defines the loop function.

Built-in Scripts `PhysioLabXR` offers several built-in scripts that implement commonly used algorithms for processing data from various types of sensors. Notable examples include channel averaging, band power calculation, and fixation detection. The built-in scripts are shipped with `PhysioLabXR`'s standalone distribution and can be added in the same way as a user script. The user will need to provide the *input(s)* to the algorithm the script implements, and the scripts expose *output(s)* to stream out processed results. For example, the user may provide a gaze sequence from an eyetracker to the fixation detection script. In real time, the experimenter can test different backend algorithms (currently supporting I-VT [77], I-DT [78], and patch similarity proposed by Steil et al. in [79]) by choosing the corresponding script *parameter*. In addition, they can adjust the detection sensitivity by changing other parameters of the I-VT algorithm, such as minimal fixation time and saccade amplitude.

4.3.1 Digital Signal Processing Modules

`PhysioLabXR` offers a comprehensive selection of built-in digital signal processing (DSP) modules. These modules encompass commonly utilized signal processing algorithms,

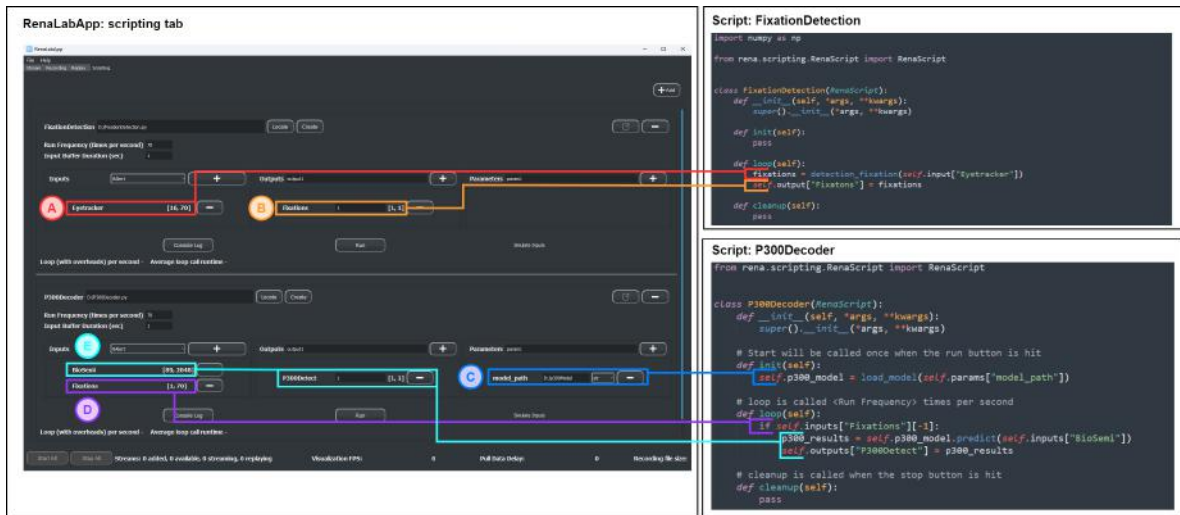


Figure 4: Example script setup a fixation-related potential (FRP) experiment. The FixationDetection script (upper right) resolves gaze behavior, while the P300Detector script (bottom right) processes fixations to determine if a FRP is elicited by a target object. This setup is similar to the experiment by conducted by Rämä et al. [76]. The labels in the figure shows the following: A. The eyetracking data as an input is processed by the fixation-detection script to check if a fixation is made by the participant. B. The fixation results are streamed through the output LSL outlet named "Fixations". C. In the .init function of P300Detector, the P300 classifier model is loaded from the file system path indicated by the script parameter model_path. D. and E. If a fixation is detected, the model takes the EEG epoch time locked to the fixation and makes a prediction. A loop call is completed by writing the prediction results to the output stream named P300Detect.

including low-pass filters, high-pass filters, band-pass filters, root mean square calculations, and clutter removal techniques [72]. The parameters for these digital filters are generated using the SciPy library [80] and applied to the rational transfer function defined as follows and optimized with circular buffers and Cython [81]:

$$Y(z) = \frac{b(1) + b(2)z^{-1} + \dots + b(n_b + 1)z^{-n_b}}{1 + a(2)z^{-1} + \dots + a(n_a + 1)z^{-n_a}} \cdot X(z)$$

The [Real-time Digital Signal Processing](#) tutorial demonstrates an example of applying DSP modules to the EEG signal. The interface is integrated within the stream group settings, located below the plotting format widget in the stream options window, and becomes visible when an individual group is selected. Users have the flexibility to apply DSP modules to each individual group and specify the corresponding parameters, and each data processor will be applied sequentially to the input stream. It should be noted that the DSP feature described in this section does not alter the number of channels in a stream, as the processed results replace the original data content. Transformations that result in a different number of output channels compared to the input, such as averaging and band power computations, are provided through PhysioLabXR’s scripting interface (Section 4.3).

4.4 Software Architecture

Smooth runtime experience is a significant challenge for large-scale Python software when dealing with high-throughput data, complex graphics, and frequent i/o operation from serialization in large-scale Python software, given the interpreted nature of the language. In this section, we delve into the architectural design of PhysioLabXR, addressing this challenge and helping interested users interpret the framework behind the features discussed in the previous section. As we will show, the architecture of PhysioLabXR follows rigorous software design patterns, minimizing maintenance efforts while maximizing scalability. We aim to offer readers seeking to contribute to or build upon PhysioLabXR’s foundation a deeper understanding of the backend. This section presents a high-level overview of the key design features, including the threaded pipeline of data workers and the visualization GUI, the recording API, and the *.dats* serialization interface, as well as the parallelism in servers for replay and scripting. For a comprehensive development guide, we recommend referring to the developer section of PhysioLabXR’s docs.

4.4.1 Concurrency in the main process

PhysioLabXR utilizes concurrency in various components, including the GUI controller, data workers, visualization, and serialization, to optimize for a smoother user experience. Concurrency is achieved through different feature threads, allowing one routine to run

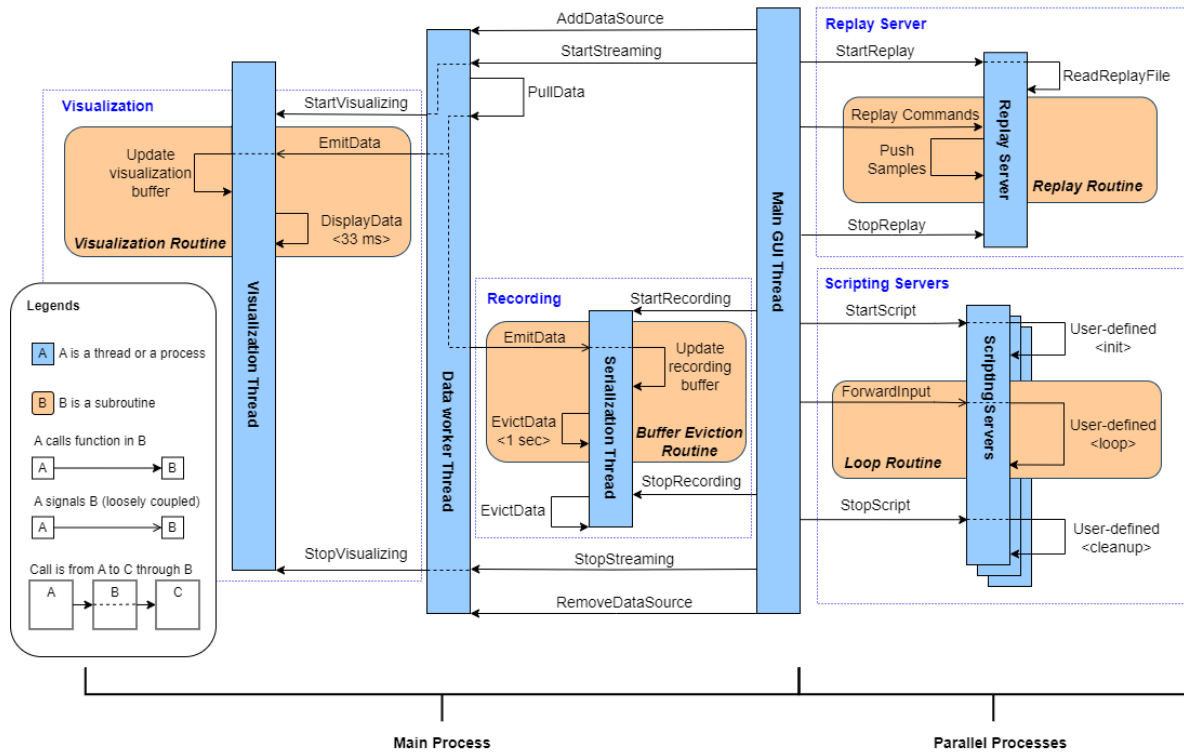


Figure 5: Sequence diagram showing the information exchange between PhysioLabXR's threads and processes. The main process contains concurrent threads of the main GUI controller, serialization, data worker, and visualization. When the user adds a new data source, the GUI thread forks a data worker and a visualization thread. More demanding operations run on separate server processes, including replay and scripting. User commands such as *StartStreaming*, *StartReplay*, and *StopRecording* are passed from the main GUI to the corresponding threads or processes.

while others block an external event and take advantage of available CPU time. The following list details the behavior of these concurrent threads:

- **Main GUI Controller** remains dormant unless the user interacts with the interface widgets. It relays the user's input to other routines for handling those interactions. For example, in figure 5, the replay server receives signals such as *StartReplay*, *ReplayCommands* *StopReplay* from the GUI controller.
- **Data worker threads** pull data from external data sources, signaled by a global timer with a default interval set at *1ms*. In other words, the thread attempts to capture data a thousand times per second. We choose this value empirically through testing with different data source combinations. It allows ample runtime for other threads while respecting data sources with high sampling rates. When data is received, the worker emits the data, along with the meta information containing the timestamp and stream name to the visualization and recording thread. It should be noted that the receive-data calls in the data workers are non-blocking. In other words, the call for pulling data should return immediately and not wait when no data is available, allowing other threadings to run. Many networking and serial communication modules include an argument to set the timeout in their pull-data calls. It is especially important when users implement custom data interfaces; they should set such timeout to zero and ensure the function would not block.
- **Visualization thread** receives data from the data worker and stores them in circular buffers. The size of the circular buffer is calculated by each stream's nominal sampling rate and the *number of seconds to display* chosen by the user (default at 10 seconds). The plot routine is signaled by a timer set to tick every *33ms*, giving a global nominal 30 fps refresh rate. Users may change the refresh rate in settings. In addition to plotting, the visualization thread also probes the data worker for the availability of the stream at timer ticks. If a stream becomes unavailable and remains so when a timeout (default two seconds) is reached, PhysioLabXR considers this stream lost; it then proceeds to remove the resources allocated for it and notify the user.

4.4.2 Serialization interface

PhysioLabXR implements a type-length-value (TLV) serialization interface designed to work with multi-modal, multi-dimensional data with timestamps. Once recording starts, all the data streams are routed to a specialized buffer. The buffer's content is retained in the host's memory until the eviction interval is hit. Default at one second, the eviction interval controls how often we offload the buffer to the disk. When the data throughput are high, it is important to evict the buffer in time to prevent out-of-memory. User may adjust the eviction interval in the settings to optimize for their use case. During an eviction, each stream's data and timestamps is appended to the file as a TLV packet whose structure is detailed in figure 6. As each packet is independent and contains all

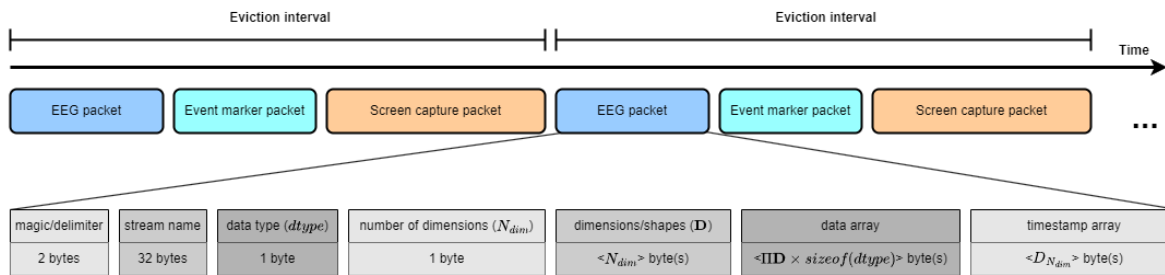


Figure 6: Organization of a `.dats` file PhysioLabXR’s built-in serialization scheme in relation with the eviction interval. In this example, the software records EEG, event markers and screen capture. The file content is segmented by first by eviction intervals. Within each interval, each type-length-value (TLV) packet contains the data for individual streams. The figure shows the anatomy of a TLV packet. Starting with a delimiter sequence called *magic*, the packet contains the data array and timestamps preceded by meta information: stream name, data type, number of dimensions, and the shape of the data. When loading `.dats` back, the loader uses the dimension information to determine the number of bytes to read as data and timestamp.

information needed to decode the enclosed data. Figure 5 gives a graphic illustration of how the serialization thread receive emitted data from the data worker threads and evict buffers.

4.4.3 Multi-process servers for replay and scripting

Heavy-lifting processes, or routines whose runtime is highly variable can block GUI process, making the app irresponsible and leading to frustration for the users. PhysioLabXR’s two major bottleneck processes are sandboxed from the main thread and put on parallel server processes: replay and scripting, both of whose runtime depends heavily on how user use them. The multi-process architecture of PhysioLabXR observe best industry practices to maintain system integrity and can recover from fault. The servers are forked from PhysioLabXR as needed to optimize system resource usage, while the communication between with the server are made through TCP sockets. Next, we will cover the multi-process setup and communication for replay and scripting to help user understand and potentially extend the potent backend of PhysioLabXR.

Replay server Replay typically involved a time-consuming read operation from the file system at the beginning, following by a high-frequency push data loop to faithfully reenact the timing characteristic of the data sources being replayed. The replay server, created at the start of the app as a separate process alleviates much of the strict timing requirements for the GUI process. A command-info-socket implemented in router-dealer pattern [67] exchanges information between the replay process and the GUI, from which user can asynchronously issue commands such as start, stop, pause, and seek. From the server side, the socket relay information including if loading of a file returns success,

how much data out of the replay file is already broadcasted, and if a replay has just completed. The replay process stays idle until it receives a start command with a replay file location.

The runtime of the scripts added via the scripting interface typically depends on the complexity of the algorithm they implement and the i/o throughput. In answer to the resulting variability, PhysioLabXR creates a separate process for each script when user starts it from the GUI. The app maintains four router-dealer sockets for each script:

- **info socket** receives performance metric (see section 4.3) when a script is running.
- **standard output socket** The standard output (stdout, such as the print calls) of the running script are redirected to this script and displayed through the console log window.
- **command socket** deals two types of commands (1) notify the script process to close when the close command is issued from the GUI. (2) when user change the value for any parameters when the script is running, the command interface notifies the script process and change is reflected in the next *loop* call.
- **input socket** relies user-defined inputs to the forked process. The frequency at which the inputs are forwarded controls the run frequency f_{run} of the script because the script process triggers a *loop* call when receiving a message from the input socket.

Similar the replay, each script widget also periodically checks if the forked script process is alive. However, this process does not have an idle state like replay as they are joined when the stop command is issued. Therefore, for each script process, the app keeps a thread monitoring if its process ID (PID) exists. If the process no longer exists, the app treats as if the script is closed, and notify the user of the abnormal termination. The user can proceed to investigate and restart the script. On the other hand, if a stop command is issued to an living script process but a timeout (default two seconds) is reached without a response. The script process is killed, leaving no possibility for orphaned processes unduly consuming system resources.

4.5 Use Cases

In this section, we showcase the use of PhysioLabXR in a variety of applications, ranging from neuroscience experiment to driving BCI or HCI paradigms. Each example is more involved than the previous utilizing more advanced features in PhysioLabXR. The first example is a basic usage of the visualization and recording features of PhysioLabXR for a multi-modal physiological experiment, while the other two examples creates close-loop interactions through the scripting feature, a P300 speller and a gesture interface using a novel radar sensor. We encourage users refer to the online documentation for the comprehensive tutorials for all the use cases.

4.5.1 NiDyn Experiment

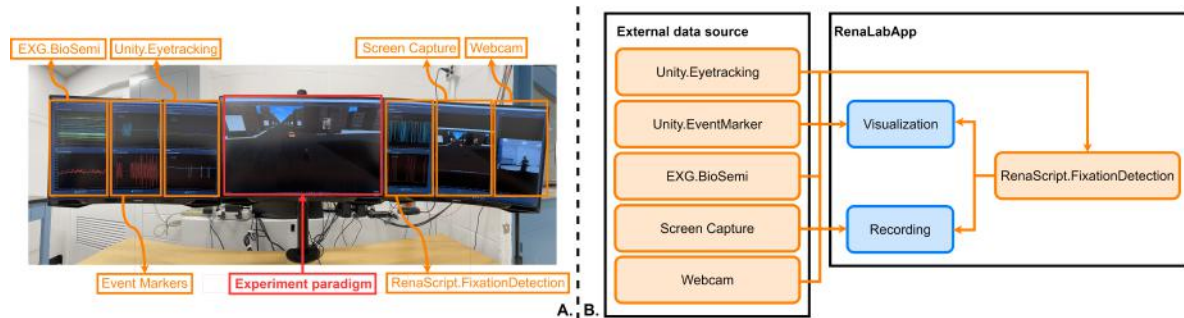


Figure 7: A. Workspace setup with PhysioLabXR with three monitors for the NiDyn experiment, allowing the researcher to keep track of all the data sources of the experiment. The user is running a paradigm built with Unity [68] in the middle monitor. The app is streaming (from left to right) EEG, event, markers, eyetracking, experiment paradigm, fixation detection, screen capture, and camera recording of the participant. B. all data sources, including a fixation detection script within PhysioLabXR processing the eyetracking stream, are visualized through PhysioLabXR and being recorded at the same time.

When a researcher’s main interest is in collecting data and analyze them offline, PhysioLabXR offers a simple solution for visually monitoring the data during collection with a robust recording interface. The NiDyn experiment from [82] demonstrate a use case of PhysioLabXR involving multi-modal data streams, and the use of a fixation detection algorithm deployed through the scripting interface. The experiment investigates human reorientation in naturalistic environment; the paradigm, built in Unity [68], is in VR with the participants driving through a procedurally generated city. Billboards are placed at two sides of the road with target or distractor images on them. While driving, participant needs to count the number of target images while keeping a safe distance from the car in front of them. To study the neural dynamics when decoding the images, this paradigm requires the collection of EEG, eyetracking, screen capture, webcam video of the participant, fixation detection and most importantly, the event markers indicating when a target or distractor image is presented.

Among the data sources, the eyetracking (The VR HMD has built-in eyetracker [74]) and event markers is pushed via LSL from Unity with which the experiment environment is implemented. The EEG device is BioSemi ActiveTwo [73] and PhysioLabXR connects with the EEG hardware with a native plugin. The screen capture and webcam are added though the PhysioLabXR’s video device interface. In addition, the experiment deployed a real-time fixation detection algorithm via the scripting interface (see section 4.3). The fixation script takes in the eyetracking stream as input and outputs the detected eye behavior during experiment. With EEG being synchronously captured by PhysioLabXR, experimenter can easily extract the fixation-related potential and characterize neural activity during the target identification task [83].

4.5.2 P300 Speller

The P300 Speller [84] is a well-known use case of EEG in the field of neural science and brain-computer interfaces that provides real-time feedback to the user. It is often used as a "Hello World" example due to its simplicity and effectiveness in demonstrating the potential of brain-computer interface technology. In this example, we demonstrate the collaboration between PhysioLabXR, the OpenBCI Cyton board, and Unity platform to develop an experimental paradigm from [85] that trains the model during the training session and provides real-time visual feedback to the user during the testing session as shown in figure 8.

During the experiment, OpenBCI Cyton board connects to PhysioLabXR using the native device interface (refer to section 4.1.1 for details on the Data Stream API) and applies a 60Hz notch filter and a Butterworth band-pass filter with cutoff frequencies of 2Hz and 40Hz using digital signal processing modules, as discussed in section 4.3.1. Unity provides event markers including the flashing of each row and column, the index of the activated row or column, the beginning and end event of each trial, and session as well as the target character, if the training session is running. The P300SpellerClassifier.py script incorporates a state machine that collects trial epochs after receiving the training session start event marker and begins collecting trail epochs to train the logistic regression model based on the gathered data. During the subsequent testing session, the pre-trained model is applied to the collected data for each trial, predicting the row and column indices with the highest probability of containing the target character that the user was looking at. This information is then transmitted to Unity through the LabStreamingLayer (LSL) interface, highlighting the predicted target on the character board in Unity Platform.

4.5.3 Micro-gesture Interface

In this example, we demonstrate the use of PhysioLabXR to power a gesture interface called IndexPen [72], an input system that utilizes micro-gestures and mmWave Radar. IndexPen enables users to write using their index finger on their thumb, mimicking real-world handwriting including 26 characters, as well as Space, Backspace, Enter, and Activation functions.

Figure 9 illustrates the user performing micro-gestures to the mmWave radar, which is connected to PhysioLabXR via a device interface. In this particular use case, the mmWave radar serves as the sole data source. PhysioLabXR receives range-doppler (RD) and range-azimuth-elevation (RAE) data, which are then post-processed using a clutter removal data processor as mentioned in 4.3.1. The RD and RAE maps are split and reshaped into two separate 2D heatmaps within the visualization tab. To enable the real-time interactio, RenaScript. IndexPen takes the time-series clutter-removed RD and RAE feature maps and applies them to the pre-trained IndexPen gesture model. Upon detecting a specific gesture, the corresponding keystroke is invoked. While the previous two use cases both involves physiological data, this example aims to broaden the scope of PhysioLabXR in enacting an standalone HCI paradigm involving a novel

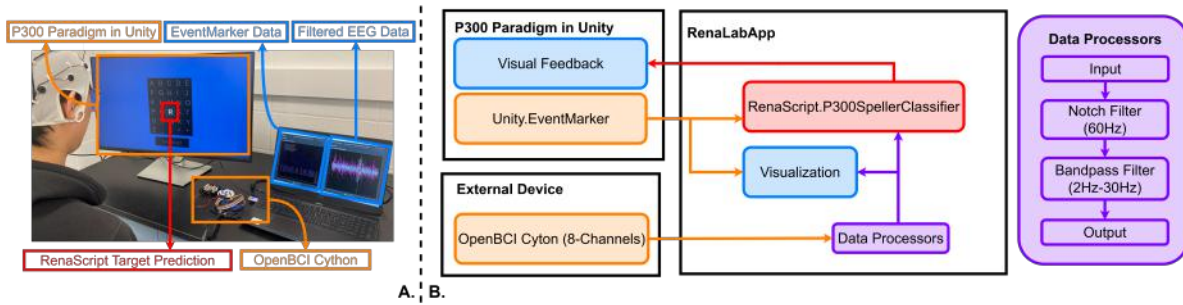


Figure 8: A: The experiment utilized the PhysioLabXR, OpenBCI, and Unity platform to enable participants to train and test the P300 speller in real-time. Participants were seated in front of a monitor presenting a character board in Unity. PhysioLabXR streamed EEG data and event markers while also providing prediction results to the Unity platform through LSL. The figure above indicates that the character 'R' was classified as the target the participant was looking at. B: PhysioLabXR received input streams from an OpenBCI Cyton (8-channel) board, connected through the device interface, and event markers from the Unity platform. The EEG stream underwent data processing before being forwarded to RenaScript, as depicted in the figure on the right side. Real-time data exchange occurred between PhysioLabXR and the Unity platform. The P300SpellerClassifier trained a logistic regression model during the training session using the collected data. In the testing session, the pre-trained model predicted the indices of the row and column with the highest probability of containing the target character for each trial. This information was then transmitted to the Unity platform via LSL and displayed within the Unity environment.

sensor, the processing of its data using deep learning models, and driving a keyboard-like interaction.

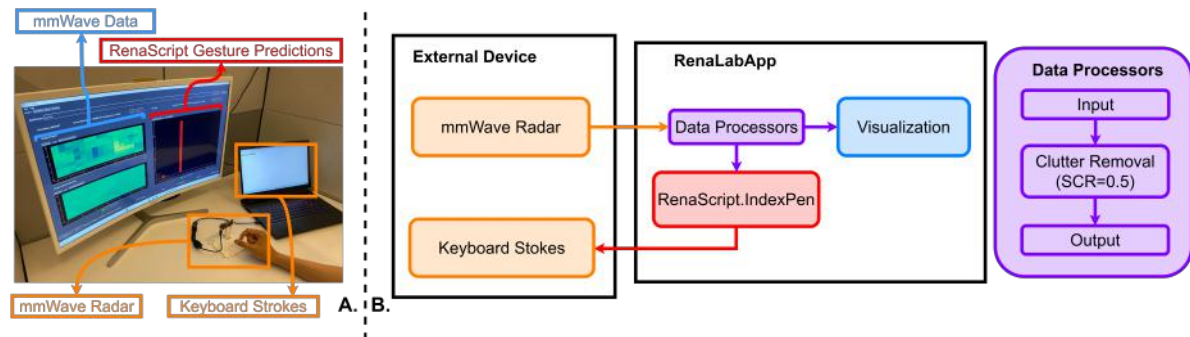


Figure 9: A: The user was sitting in front of the mmWave radar, performing micro-gestures towards it. The monitor displayed the range-doppler (RD) and range-azimuth-elevation (RAE) data streamed to PhysioLabXR. The data was reshaped into a 2D signal channel heat-map, and the gesture predictions were streamed from the scripting interface. In the figure above, the participant was writing the gesture 'L', as indicated by the spike in the probability of 'L' in the gesture predictions barplot. B: The mmWave radar was connected to PhysioLabXR using its device interface and applied with a clutter removal data processor, as shown on the right side. The RenaScript.IndexPen module received the clutter-removed RD and RAE data and predicted the performed gesture, thereby triggering the corresponding keystrokes.

4.6 Discussion

As demonstrated through the use cases, PhysioLabXR is a versatile software platform for both analytical research and prototyping close-loop interaction systems. PhysioLabXR acts as a data exchange hub for multi-modal experiments. Experimenter can connect their stimulus-presentation software such as PsychoPy [69], Unity [68], or even custom stimuli if the event marker or trigger is funneled to PhysioLabXR. In the P300 BCI example above, the developer created script to send event markers from the speller game to PhysioLabXR. On the other hand, user can simultaneously synchronize the stimulus with psychophysics recording device, while PhysioLabXR supports a vast array of neural imaging hardware, eyetracker, environment sensors, video and audio device.

Befit the multi-modal nature of many neural science and HCI experiment today, the scalable design of the PhysioLabXR's visualization features makes it easy for researcher to visually monitor almost any number of streams at the same time. The highly optimized serialization interface can record this high-through put data with synchronized timestamps across all the streams. When pair with replay, RenaScript allows user to test their pipeline in real-time as if during a live experiment. This is especially relevant when designing close-looped interaction paradigms. The experimenter can pre-record a session with the participates, and later use the replayed session to test different approaches to

implement the interaction or experiment with different ways to model the user's data, now being reenacted with same timing characteristics to the original session.

PhysioLabXR seeks to facilitate a fast-paced research & development cycle. The user layer is implemented fully in Python and user does not need to compile any source code to run PhysioLabXR. While packaged executable is available on all major operating systems, users are welcome to clone, run, and modify the source code. Once the user completes setting up the Python environment, PhysioLabXR can become a scaffolding on top of which user can build their applications in rapid development cycles as no compilation is needed after changing the source. PhysioLabXR offer several developer features tailored to users who would like get more value out of the framework's robust features such as visualization, recording and replay. Few of the many developer tools include data stream API for adding custom data sources, and RenaScripting for deploying any data processing pipeline utilizing the vast and growing Python libraries.

4.7 Limitation and Future Scope

PhysioLabXR aims to be a versatile platform for real-time experiments, primarily for, but not limited to, HCI and neuroscience. It is designed to be a community-driven project, with our core team of developers maintaining architectural integrity and functional correctness. At the same time, we welcome contributions from researchers and practitioners in related fields to build on this scaffolding and expand its capabilities. While the stream interface supports data sent through LSL or ZMQ, we are currently developing native plugins for sensors that lack network support. These plugins will enable the use of certain brands of fMRI, TMS (transcranial magnetic stimulation), and invasive neuroimaging devices [e.g., Neuropixels, [86]]. We are also adding real-time analysis and processing modules for more modalities, such as real-time source localization for EEG and speech recognition for audio. Moreover, the current scripting interface is designed to provide maximum flexibility, thus requiring users to write Python code for their pipelines. In coming releases, we plan to make the scripting features more accessible to users with less programming experience by providing a visual programming interface and code generation.

4.8 Conclusion

PhysioLabXR is a pioneering tool in the era of spatial and physiological computing, addressing the increasing demand for a comprehensive software platform that drives multi-modal data integration and close-loop interaction. PhysioLabXR offers an array of interconnected functionalities, including visualization, recording, replay, real-time DSP modules, and a scripting interface. These all empower researchers and practitioners to explore novel experiment paradigms and design intricate feedback loops. With its Python-based frontend and C++-powered backend, the framework is built with scalability in mind while having optimized performance. The continued development of

PhysioLabXR will be driven by the community and aims to fuel research insights into the intersection between the brain, behavior, and human-computer interaction.

5 Interactively Augmenting clinical decisions with expert knowledge-distilled Vision Transformer

3

Glaucoma constitutes a significant challenge in global public health, accounting for 15% of the total burden of world blindness [87]. Optical Coherence Tomography (OCT) has emerged as a crucial imaging technique for diagnosing glaucoma and many other eye diseases. [88] [89]. With the expected rise in eye disease prevalence due to an aging population, leveraging Artificial Intelligence (AI) to streamline diagnosis and minimize the demand on ophthalmology human resources is imperative [90].

Previous studies have underscored the potential of computational intelligence to enhance clinical decision-making processes [42]. A majority of these studies have focused on training machine learning models to identify Regions of Interest (RoIs) in medical imaging [49] [50]. This paper introduces a novel approach for automatically suggesting RoIs during classification tasks. Our system enables clinicians to interactively learn from a model that harnesses distilled expert knowledge (in the form of clinician eye movements on OCT reports during model training), facilitating its integration into routine clinical practice, and thus bridging the gap between Computer-Aided Diagnosis (CAD) tools and practical clinical applications [37].

5.1 Unsupervised RoI detection

Our system detects RoIs in OCT reports that are pertinent to diagnosing glaucoma. The key components of our system are two-fold: (1) a ViT trained and distilled with experts' knowledge, and (2) an interactive front-end that lets clinician explore the discriminating features produced by the model for the reports not seen during the training process. The front-end combines the RoI detected by the model and user's attention as they are analyzing the reports to deliver personalized guidance.

We first discuss the unsupervised RoI detection models we developed, ViT and CNN-based, of which the former is a relatively new approach; only CNN-based models have been quantitatively studied for their clinical impact [55]. Next, we present how we augment the ViT model with experts' eyetracking data with a method that we term *expert attention loss*. The eyetracking data was collected in a previous study while

³Haowen "John" Wei created the entire experiment paradigm in Unity, along with the Scripting Interface in PhysioLabXR, and conducted both user studies. Ziheng "Leo" Li was responsible for all the deep learning-related work for this project. Kuang Sun prepared the dataset with John and assisted with both user studies.

clinical experts analyzed the OCT reports in our dataset. We give more details about the dataset in Section 5.2.

5.1.1 Unsupervised RoI identification for glaucoma in OCT reports

While an ML model can be trained to predict RoIs within a dataset containing RoI labels, medical datasets with only the classification label come in far more abundance than datasets with annotated RoIs. This is understandable considering the amount of manual labor needed to produce expert-labeled RoIs. This is particularly the case with OCT reports for glaucoma diagnosis given OCT is a relatively new and rapidly evolving imaging modality, and the clinical value of OCT reports in glaucoma diagnosis is still under scrutiny. We therefore address the problem of identifying glaucomatous RoIs in OCT reports with datasets that only have disease/no-disease labels.

Detecting RoIs from ViT attention rollout. Various methods have been proposed over the years for automatic RoI detection while the model is trained on a classification task. For transformer-based models, attention rollout [52] multiplies the dot products between query and key throughout the layers; thus, it is a proxy for the model’s “attention” for its tokens. In the case of ViT, the tokens are the image patches which together make up the input image [91]. For classification tasks, it is common to add a classification token (CLS) that is passed through the attention layers and aggregates information from the other tokens that are discriminatory. To learn which tokens have contributed the most to the classification results, we look at how the CLS token as a query is attended by other image patches as keys. Patches that the model deems to be more discriminatory will have higher attention values when attended by the CLS token:

$$\text{Attention}(x_{CLS}, \mathbf{x}_{\text{key}}) = \text{softmax}\left(\frac{x_{CLS}W_q(\mathbf{x}_{\text{key}}W_k)^T}{\sqrt{d_k}}\right), \quad (1)$$

$$\text{AvgAttention}(l) = \frac{1}{H} \sum_{h=1}^H \text{Attention}(*; l, h), \quad (2)$$

$$\text{AttentionRollout}(L) = \prod_{l=1}^L \text{AvgAttention}(l), \quad (3)$$

where in the first equation, $\text{Attention}(x_{CLS}; \mathbf{x}_{\text{key}})$ are the attention weights computed for the classification token (x_{CLS}) against the key tokens (\mathbf{x}_{key}) in a transformer layer. The softmax function normalizes the dot product of the query embeddings of the classification token ($x_{CLS}W_q$) and the key embeddings of the other tokens ($\mathbf{x}_{\text{key}}W_k$), scaled by the dimensionality of the key vectors ($\sqrt{d_k}$). This normalization ensures that the attention weights sum up to one, signifying the relative importance of each key token to the classification token. Note that we omit the head h and layer l from the Attention for simplicity.

In the subsequent equations, $\text{AttentionRollout}(L)$ calculates the cumulative attention map across all L layers of the transformer. Each layer’s contribution is given by $\text{AvgAttention}(l)$, which averages the attention matrices across all H heads in layer l . The function $\text{Attention}(*; l, h)$ refers to the attention values for head h in layer l as in equation 1. By taking the product of these average attention maps from layer 1 to layer L , the attention rollout aggregates the attention flow throughout the layers, promoting the most important features for its inference.

Detecting RoIs from CNN-based models via GradCam. As contenders for the ViT, we tested three widely used CNN-based models: VGG19 [92], ResNet [93], and Inception-V4 [94]; we generated RoIs from these models using Grad-CAM. Grad-CAM shows which parts of an image lead to most activation for a given label (e.g., glaucoma presence) and has been shown to improve clinician performance when shown side-by-side with OCT images in a previous study [55]:

$$\text{Grad-CAM} = \text{ReLU} \left(\sum_k \alpha_k^c A^k \right), \quad (4)$$

where A^k represents the activation maps of a convolutional layer, and α_k^c are the neuron importance weights for class c , calculated by averaging the gradients of y^c (the score for class c) with respect to the feature maps A^k . The resulting heatmap $L_{\text{Grad-CAM}}^c$ highlights the important regions for the decision of class c .

We aim to find the model that generates RoIs with the most clinical relevance. Specifically, we want to test the following hypotheses through a user study:

H1: Regardless of the backend model, clinician performance for analyzing OCT images is improved when clinicians are cued with RoIs generated from a trained glaucoma classification model.

H2: The RoIs generated from attention rollout of a ViT model have higher clinical relevance than those generated via Grad-CAM from CNN-based models.

5.2 Expert knowledge distillation

To prepare the model that can generate heatmaps highlighting parts of an image that are more clinically relevant, particularly with relatively small medical image datasets, we propose to augment vision transformer models with expert eye tracking data, effectively distilling expert knowledge embedding it in our ViT models, and providing this knowledge to novice participants via our user interface.

Dataset. Our OCT image dataset was collected from 185 eyes of 185 patients who visited the NewYork-Presbyterian / Columbia University between 2010 and 2023. The ground truth glaucoma diagnoses for this dataset were generated by a consensus of 2–3 expert ophthalmologists who had access to OCT reports, visual field tests, and chart

history for these patients (unlike our deep learning system, which is just trained to detect glaucoma using OCT reports alone).

Our tests showed that a vision transformer trained without expert Region of Interest (RoI) loss, \mathcal{L}_{RoI} , can still identify relevant regions. We believe that integrating domain knowledge from medical reports could help focus the vision transformer’s attention on areas crucial for accurate classification.

We recruited 15 clinicians to review the OCT reports and collected their gaze data. In total, 476 reviews were conducted, with each report being reviewed one to three times by different clinicians. During these reviews, we recorded the gaze data of the clinicians. To ensure robustness in our data handling, we treated each review of the same image as a separate sample. We also made sure that no review of the same image appeared in both the training and testing datasets during our splits.

We calculate the attention with the following equation:

$$\mathbf{A}_{patch}^{ij} = \text{softmax}^*(\langle x_i W_q, (x_j W_k)^\top \rangle), \quad (5)$$

where softmax^* is computed along the rows of \mathbf{A}_{patch} converting the raw similarity kernel (i.e., inner product between query and key embeddings) into a probability distribution such that for each query patch, the weights across all key patches sum up to 1.

H3: Expert knowledge is effectively distilled with the proposed loss function used during the training.

This hypothesis will be tested by the model’s downstream glaucoma classification accuracy—how many parameters it requires while retaining a high level of performance. Note that we have only developed a method to incorporate expert attention into the training process of ViT. The CNN-based models (including VGG19, ResNet50, and Inception-V4) do not participate in this hypothesis.

5.2.1 Implications for the user study

Due to the time constraints of the user study, we cannot test the RoIs generated from each model listed above. Therefore, we use classification accuracy as a proxy for the quality of the RoIs generated by the model. In our user study, to test H2, we pick one CNN-based model and one ViT model with highest classification performances to produce RoIs for the clinicians to view. In addition to the static heatmap derived from the model’s RoI, we introduce perceptual RoI cuing, where the user can request updates to the RoI when they are actively viewing an OCT report. This will be the topic for the next section.

Further details about the paradigm used to collect the experts’ eyetracking data when analyzing OCT reports, about our models’ performances compared to other state-of-the-art models, and an ablation study for the expert RoI loss can be found in the appendix.

5.3 Perceptual RoI cuing

While methods such as attention rollout and Grad-CAM offer a glimpse into a model’s decision criterion, a model’s RoI may not always be interpretable by a clinician, especially given the intricacy of medical images, wherein subtle features can significantly influence a clinical decision. Moreover, even experts do not always agree on which part of an image helped them reach a decision (5.2).

We propose perceptual RoI cuing, whereby the user can explore a model’s attention in an intuitive way. This is in contrast to displaying the RoI straight from the model, which we call *static RoI*. The aim is to 1) take advantage of the self-attention between image patches, which does not use regular attention rollout techniques [52] 2) foster more personalized collaboration between the clinicians and the ML backend by letting the user explore the model’s knowledge combined with their own experience.

Specifically, we formulate an algorithm to integrate the user’s attention at the time of viewing the image and the attention rollout from the ViT. When the user requests an update to the RoI, the newly computed RoI will be the regions that are most informative for classification given the user’s attention so far. We formalize this claim through a proof sketch towards the end of this section.

Real-time user attention

Fixation detection. In the context of perceptual Region of Interest (RoI) cuing, we first compute the real-time user attention starting by filtering the raw gaze data into fixations, effectively reducing the noise in the data. Fixations are points where the gaze is held relatively steady and are indicative of where the user has attended to extract information [77]. Our method first fills in the gaps in gaze data caused by eyetracker glitches [95]. Gap-filling handles periods where gaze data is missing or deemed invalid, ensuring only continuous data are used for subsequent fixation detection. Any invalid data intervals that are less than 75 ms are treated as a gap and interpolated. We then detect fixations using the Identification by Dispersion-Threshold (IDT) algorithm [77]. The algorithm evaluates a moving window of gaze data and calculates the dispersion of gaze points within this window. If the dispersion is below a certain threshold, the window is classified as a fixation. In our implementation, we use a window size of 175 ms and a dispersion threshold of 0.5° [96].

Estimate user attention with spatial Gaussian and temporal decay. We next estimate human attention from the fixations. This process dynamically updates the attention map, incorporating recent fixations and accounting for the decay of attention from previous fixations. The attention map of a fixation is computed by applying a Gaussian function around each fixation point, which spreads the focus of attention around the fixation:

$$\mathbf{F}_t(i_t, j_t) = \exp\left(-\frac{(i_t - x)^2 + (j - y)^2}{2\sigma^2}\right), \quad (6)$$

Algorithm 1 Fixation Detection using IDT

```

1: function FIXATIONDETECTIONIDT(gaze_xyz, timestamps, window_size, dispersion_threshold)
2:   gaze_angles_degree ← CALCULATEGAZEANGLES(gaze_xyz)
3:   windows ← CREATEWINDOWS(timestamps, window_size)
4:   fixations ← []
5:   for each [start, end] in windows do
6:     if end − start < saccade_min_sample then
7:       continue
8:     center_time ← timestamps[start] + window_size/2
9:     if COMPUTEDISPERSION(gaze_angles_degree[start : end]) <
       dispersion_threshold_degree then
10:      fixations.append([1, center_time])           ▷ 1 for fixation
11:    else
12:      fixations.append([0, center_time])           ▷ 0 for non-fixation
13:  return fixations

```

where (i, j) are the pixel coordinates in the attention map. The spread centers at (x, y) , the fixation point. σ is the standard deviation, which we choose to be $\rho \cdot 5^\circ$, corresponding to the size of human fovea. ρ is the pixel per degree on the screen. The resulting σ is 60 pixels.

The attention map is updated over time, taking into account the current fixation and the decay of past attention. Let A_t represent the attention map at time t , and F_t represent the Gaussian function applied to the fixation at time t :

$$A_t = p \cdot \mathbf{F}_t + (1 - p) \cdot \mathbf{A}_{t-1}, \quad (7)$$

where $p \in [0, 1]$ is a decay parameter that balances the influence of the current fixation against the attention from previous fixations. A higher value of p gives more emphasis to the most recent fixation, while a lower value retains more of the past attention map. We choose a p of 0.995 from our empirical observations, making the most recent fixation have the largest contribution; a fixation made a second again retains 0.5% of its value.

5.4 Perceptual Attention

We define the *perceptual attention (RoI)* as a measure of how informative a patch is given the user’s and the model’s attention. This measure integrates the user’s attention with that derived from the Vision Transformer (ViT) model, leading to more user-centric understanding of the most informative parts of the image.

H4: Clinical performance is improved when the user can interact with the perceptual RoI while analyzing the OCT reports.

Here, we first show how we compute perceptual attention (RoI) and give an empirical interpretation of our approach. Then we offer a proof sketch formulated in a Bayesian framework and argue that the results indeed compute the informativeness of the patches knowing the user’s and the model’s attention.

The inputs for this computation are:

- ViT attention between image patches, denoted as $\mathbf{A} := \mathbf{A}_{patch} \in \mathbb{R}^{n \times n}$, where n is the number of patches. This is the inner product between patch key embeddings and query embeddings, indicating the relatedness of patches (eq.)
- ViT attention between the key embedding of the CLS token and all the patch’s query embeddings, denoted as $\mathbf{C} := \mathbf{A}_{CLS} \in \mathbb{R}^n$ (eq. 1).
- The user’s attention from eq.6, denoted as $\mathbf{U} := \mathbf{A}_{user} \in \mathbb{R}^n$.

Empirically, the CLS token’s attention, \mathbf{C} , relates to which patches contribute more to the classification decision. On the other hand, the self-attention matrix, \mathbf{A} , is the similarity between patches that is often not used when visualizing the rollout from ViT [51] [97]. By considering these two aspects together, we can identify patches that are dissimilar to the ones the user paid more attention to and contribute significantly to the classification decision. For example, if an inexperienced clinician is looking at the green regions in the RNFL probability map [98], which don’t contribute to the classification of this report, the perceptual RoI will guide them to attend the red regions that they have not attended that are more important for the classification. On the other hand, if they are already attending the red regions, the system will highlight the thinning of the nerve fiber in the RNFL thickness map [98], which also contributes to the classification but is a feature not noticed by the user. Our formula for perceptual attention is:

$$\mathbf{A}_{percep}^j = \sum_i^n A_{percep}(i, j), \quad (8)$$

$$A_{percep}(i, j) = \frac{\mathbf{C}^j \cdot \mathbf{A}^{ij}}{\mathbf{U}^i}, \quad (9)$$

where informativeness of patch j $A_{percep}(i, j)$ is a result of the user attending other patches. Summing over all image patches that the user may attend, we have the perceptual AOI value for patch j . The numerator, $\mathbf{C}^j \cdot \mathbf{A}^{ij}$, computes the product of the CLS token’s attention and the self-attention between patches, highlighting the patches that are relevant to the classification. The denominator, \mathbf{U}^i , brings attention to patches that escaped the user’s attention.

Proof sketch using Bayesian formulation

In the Bayesian framework, we interpret the perceptual attention computation as a probabilistic update. Let the events be

$$\begin{array}{c}
 \boxed{\mathbf{U} \quad \cdot \quad \mathbf{A}^T \quad \odot \quad \mathbf{C}} \\
 \left[\frac{1}{U_1} \quad \frac{1}{U_2} \quad \frac{1}{U_3} \quad \frac{1}{U_4} \right] \quad \begin{array}{c} \left[\begin{array}{cccc} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{array} \right]^T \end{array} \quad \left[C_1 \quad C_2 \quad C_3 \quad C_4 \right] \\
 \\
 \mathbf{A}_{percep} \\
 = \left[C_1 \left(\frac{A_{11}}{U_1} + \frac{A_{12}}{U_2} + \frac{A_{13}}{U_3} + \frac{A_{14}}{U_4} \right) \quad C_2 \left(\frac{A_{21}}{U_1} + \frac{A_{22}}{U_2} + \frac{A_{23}}{U_3} + \frac{A_{24}}{U_4} \right) \quad C_3 \left(\frac{A_{31}}{U_1} + \frac{A_{32}}{U_2} + \frac{A_{33}}{U_3} + \frac{A_{34}}{U_4} \right) \quad C_4 \left(\frac{A_{41}}{U_1} + \frac{A_{42}}{U_2} + \frac{A_{43}}{U_3} + \frac{A_{44}}{U_4} \right) \right] \\
 \dots \\
 \left(\begin{array}{c} \mathbf{U} \\ \text{[Visual representation of U]} \end{array} \cdot \mathbf{A}^T = \begin{array}{c} \text{[Visual representation of } \mathbf{A}^T \text{]} \end{array} \right) \odot \begin{array}{c} \mathbf{C} \\ \text{[Visual representation of C]} \end{array} = \begin{array}{c} \mathbf{A}_{percep} \\ \text{[Visual representation of } \mathbf{A}_{percep} \text{]} \end{array}
 \end{array}$$

Figure 10: Illustration for computing the perceptual attention A_{percep} . The inverse of the user attention for each patch $\frac{1}{U_i}$ is multiplied with the self-attention matrix A_{ij} . Summing across all the patches that the user may attend, this intermediate result highlights patches that are dissimilar to the ones the user attended. Scaled with the CLS attention C , the resulting perceptual attention underscores regions that have features unattended by the user and that contribute to the classification.

- A : the patch j is informative for classification.
- B : user attended the patch i .

The posterior probability $P(A|B)$ is the probability that patch j is informative given the user has attended patch i . The prior probability $P(A)$ is represented by the CLS token’s attention \mathbf{C}^j . The likelihood $P(B|A)$, the probability of the user attending patch i given the informativeness of patch j , is approximated by the self-attention between patches \mathbf{A}^{ij} . Finally, $P(B)$ is the user’s attention \mathbf{U}^i .

The Bayesian update rule can then be applied as follows and produce the form in eq. 9:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B) + \epsilon} = \frac{\mathbf{A}^{ij} \cdot \mathbf{C}^j}{\mathbf{U}^i + \epsilon}, \quad (10)$$

where ϵ is a small constant, such as 10^{-8} , added to avoid division by zero. This equation reflects the updated belief about the informativeness of patch j given the user’s attention to patch i , combining the model’s classification focus (CLS attention) with the user’s focus (self-attention) in a probabilistic formulation.

5.5 User studies

Our research employs a two-part user study to evaluate the efficacy of different Region of Interest (RoI) guidance methods in clinical decision-making, specifically for OCT images in ophthalmology. There are two variables under scrutiny: First, we investigate the model used to produce the RoI heatmap. To compare the clinical relevance of rollout from ViT and that of the Grad-CAM from CNN as proposed in [55], among the CNN-based models we tested, we use the classification performance as a heuristic for the quality of their Grad-CAM. In this case, *ResNet50* performs the best with an averaged cross-fold accuracy at 94.7%. Indeed, a qualitative examination over the heatmaps produced by the different CNN-based models shows that *ResNet50* yields more clear-cut RoIs comparing to *Inception-V4* (72.5% accuracy). *Inception-V4*’s heatmaps are more subtle as noted by one of our pilot ophthalmology faculty (see fig. 20 and fig. ??). 2) Second, we investigate whether being able to personalize the RoI with the perceptual attention (Section 5.3) gives an edge over the “static” heatmap generated by the model without input from the user. Note that the perceptual RoI can only be computed from a ViT. Combining the variables gives us four conditions:

- *No RoI guidance.*
- *Static RoI guidance* using a heatmap generated from *Grad-CAM* of the best-performing CNN-based model (*ResNet50*), or using a heatmap from the best-performing Vision Transformer. To the study participants, we abstract the details of the backend, calling them "*Model A*" and "*Model B*" in the interface.

- *Interactive RoI with perceptual attention* integrating the user’s real-time attention and the model’s attention (as detailed in Section 5.3).

To isolate the effects of each approach, the user study is divided into two parts. User Study 1 tests $H1$ and $H2$ using the first three conditions. User Study 2 compares static versus perceptual attention from ViT to test $H4$.

5.5.1 System Overview

The experiment paradigm is build on top of PhysioLabXR [1] and Unity [68]. Given a relatively small dataset of 208 OCT image, we enhance the performance of our vision transformer through the integration of expert knowledge in the form of Area of Interest (AOI) maps. The AOI maps are estimated by applying gaussian kernels to the fixations clinical experts made while studying the images. In addition to the standard classification loss, we introduce an expert-attention-loss term, calculated as the cross-entropy between the attention map from the final transformer layer and the expert-generated AOI maps. Weighted by a factor of α , the attention loss encourages the model to focus on the expert’s AOI when making classifications.

The ViT tuned on expert attention generalizes to new OCT images and assists clinicians in making decisions (glaucoma vs. health) on them. During image assessment, a clinician can request AOI guidance, invoking the following routine: 1) a Gaussian kernel integrates the fixations leading up to the present request, creating a weighted matrix that assigns relevance scores to image patches. 2) Each patch’s self-attention is scaled by its relevance score and aggregated to produce a single, predicted AOI map. 3) The user interface overlays delineated contours from the AOI map onto the original OCT image, letting clinicians know where to look next. The approach sets itself apart from traditional static AOI augmentation. Instead of statically displaying the model’s attention, it dynamically integrates the user’s implicit attention based on perceptual information, offering an interactive and personalized solution.

PhysioLabXR serves as the backend server for real-time fixation detection, inference, overlay generation, and recording, while Unity functions as the software for stimulus presentation, providing real-time feedback to participants. In the static mode (User Study 1), PhysioLabXR’s scripting interface forwards the image and visual cue overlay to Unity via ZeroMQ. In the perceptual interaction mode (User Study2), the scripting interface calculates the participant’s attention map in real time, based on the participant’s gaze data. It computes the perceptual attention map and then sends it to Unity.

5.5.2 Experiment protocol

Participants. We recruited five ophthalmology residents, one OCT technician, and two experts (glaucoma specialists) for Study 1 (for a total of eight participants). For Study 2, we recruited two ophthalmology residents, one OCT technician, and two experts (glaucoma specialists) (for a total of five participants). We excluded the results of four

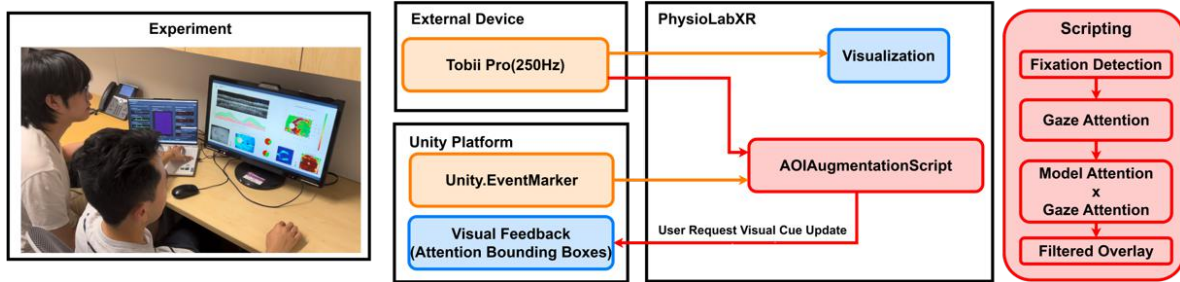


Figure 11: The Tobii Eye Tracker, which is integrated with Unity, captures and sends gaze data to PhysioLabXR. During the second user study, the scripting interface in PhysioLabXR processes the participant’s gaze data in real time. It calculates where the participant is looking on the image and updates the perceptual attention map accordingly whenever there is a change in the visual cues by the participant.

participants (three from Study 1 and one from Study 2), to maintain the integrity of our data. Specifically, three participants were excluded because they were part of an early piloting phase, which could bias their responses due to prior exposure to the study. Additionally, one participant’s data was excluded due to an incomplete post-experiment survey, crucial for our analysis.

Apparatus. The experiments use a Tobii Pro Fusion 250 Hz eye-tracker mounted on the bezel of a 1920×1080 -pixel monitor with a 60-Hz refresh rate. The OCT reports, varying in aspect ratio, are displayed on the screen occupy a maximum screen space of 1400×1750 pixels. Participants are seated comfortably in front of the monitor, with access to a keyboard and mouse to interact with the paradigm.

Procedure. We start by calibrating the eye-tracker for the participant, taking approximately five minutes. Participants are greeted with an introductory screen. A practice trial for each condition included in this user study is conducted. The study proceeds in blocks, where participants complete all trials in one condition before moving to the next. The order of conditions is counterbalanced among participants to minimize ordering effects. Instructions are provided before each block to inform participants of the upcoming condition. In each trial, participants are required to determine the presence of glaucoma, write a brief rationale (a few sentences) for their decision, and indicate the confidence-level of their decision. Each session is designed to last less than 30 minutes. Upon completion of the trials, participants are given a post-experiment survey to gather feedback on their experience. In between each trial, we briefly check the eyetracker’s calibration by displaying the gaze point in the interface, and proceed to the next trial if no issue is found.

The number of trials for the conditions organized under each user study are detailed in Tables 2 and 3.

Condition \ Image label	Glaucoma	Health
No guidance	3	3
Static (Grad-Cam+ResNet)	3	3
Static (rollout+ViT)	3	3

Table 2: Number of images used in user study 1.

Condition \ Image label	Glaucoma	Health
Static (rollout+ViT)	5	5
Perceptual (rollout+ViT)	5	5

Table 3: Number of images used in user study 2.

5.6 User interface

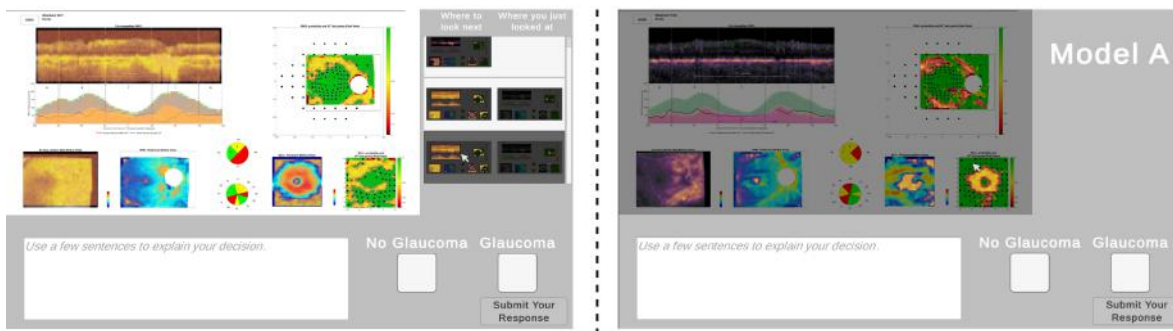


Figure 12: User interface for the guidance system. Left is interactive guidance and right is static. The interactive guidance has an additional pane on the right displaying the history of the cue updates, which shows thumbnails of the perceptual ROI under "Where to look next" and the corresponding user attention under "Where you just looked at". On the other hand, the static just shows which model is being used to generate the ROIs. To avoid bias, ViT is coded Model A and ResNet is coded Model B. In the static interface (right), the user has tuned down the opacity of the report to improve the contrast of the guidance overlay.

We provide full details of the user interface used in the paradigm. The interface is design to make guidance heatmap provide an additional layer of information for their diagnose. The interface is shown in Figure 12. Here are the interactions the system offers.

- Hide/show guidance overlay: user may click the right mouse button on the image to toggle show or not show the overlay.
- Change opacity of the OCT report: move the mouse scroll wheel on the image changes the opacity of the OCT image. Making the guidance more pronounced against the background report.
- (Only interactive) user can click on the image to request an update to the guidance computed from the perceptual ROI based on where on the report they had fixated.

- (Only interactive) user can access the history of the perceptual RoI as seen in Figure 12.

5.7 Results and Discussion

5.7.1 Quantitative Analysis of Glaucoma Diagnosis Accuracy and Confidence

	No Guidance	ViT Guidance	CNN Guidance
Accuracy	89.58% \pm 8.07%	89.58% \pm 11.60%	87.50% \pm 11.02%

Table 4: Study 1 Accuracy

	No Guidance	ViT Guidance	CNN Guidance
Confidence	1.73 \pm 0.97	1.54 \pm 0.95	1.46 \pm 0.60

Table 5: Study 1 Confidence

User study 1. The results (Tables 3 & 4) showed a comparison between no guidance, guidance generated by ViT, and guidance generated by CNN. Both the no guidance and ViT guidance scenarios demonstrated an identical accuracy rate of 89.58%, while the CNN guidance condition lagged slightly at 87.50%. This suggests that, in terms of accuracy, ViT guidance holds a slight edge over CNN guidance. Regarding confidence levels, measured on a scale from 0 to 3, participants felt marginally more confident with no guidance (1.73) compared to ViT (1.54) and CNN (1.46) guidance. The findings imply that ViT not only slightly surpasses CNN in guiding participants toward accurate outcomes but also indicates that the type of guidance can influence user confidence.

	Static Guidance	Perceptual Guidance
Accuracy	88.00% \pm 16.00%	90.00% \pm 6.32%

Table 6: Study 2 Accuracy

	Static Confidence	Interactive Confidence
Confidence	2.18 \pm 0.50	1.96 \pm 0.35

Table 7: Study 2 Confidence

User study 2. The results (Tables 5 & 6) showed a comparison between static guidance and perceptual guidance generated by ViT. The accuracy achieved under static guidance was 88.00%, while perceptual guidance led to a slightly higher accuracy of 90%. This improvement suggests that perceptual guidance, perhaps by offering more dynamic or context-sensitive cues, can enhance task performance slightly more effectively than static guidance. In terms of confidence, which was rated on a scale from 0 to 3, participants reported higher confidence with static guidance (2.18) compared to perceptual guidance

(1.96). This indicates that while perceptual guidance may lead to better performance outcomes, static guidance seems to instill greater confidence in participants.

5.8 Correspondence Between Participant Fixations and RoI Predictions

Method. To assess the correspondence between the RoIs suggested by our ViT and CNN-based models and the fixations made by participants, we computed the divergence (or relative entropy) of participant fixation distributions and model-predicted RoI distributions:

$$relative\ entropy = \begin{cases} \log \frac{x}{y}, & x > 0, y > 0 \\ 0, & x = 0, y \geq 0 \\ \infty, & otherwise \end{cases} . \quad (11)$$

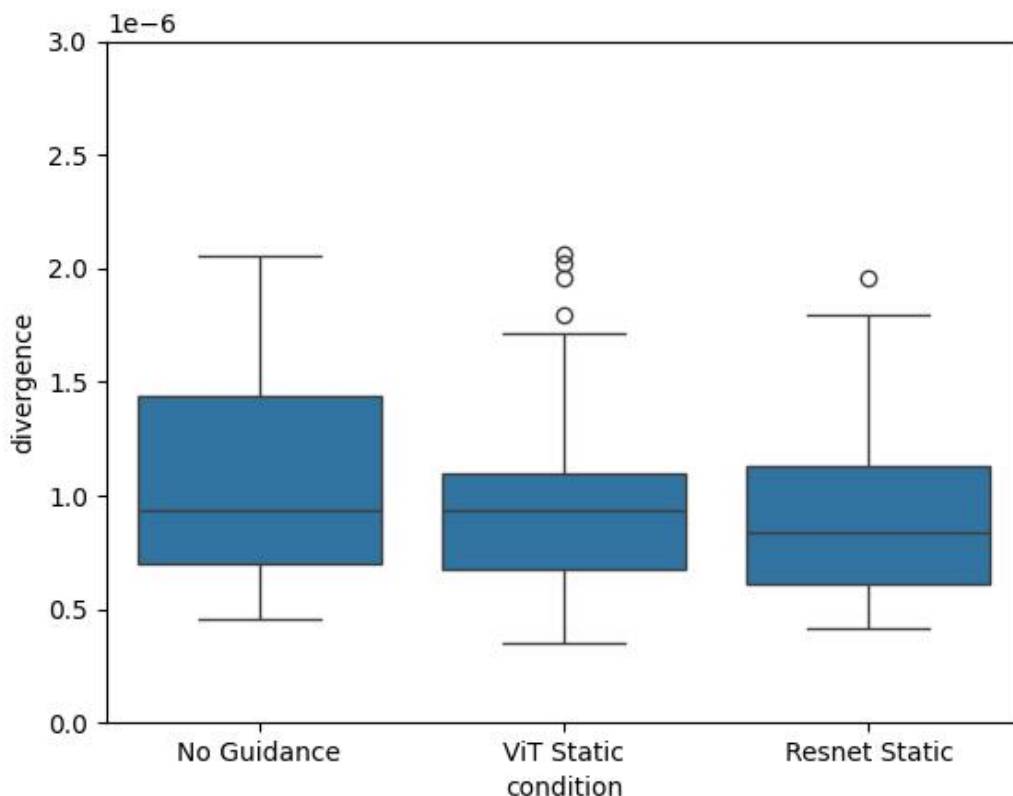


Figure 13: Median and variance for the divergence (relative entropy) between participants' fixation distributions on the OCT image and models' predicted-RoI distributions.

We found that the mean divergence for the no-guidance setting and ViT-based guidance setting (see Figure 13) was comparable, suggesting that our ViT-based model effectively learned RoIs that were almost indistinguishable from the patterns of viewing behavior our participants inherently used/were taught to use. However, interestingly, when using the ViT-based guidance, participants tended to exhibit less variance (shorter box-plot in Figure 13) in that viewing behavior compared to when they had no guidance. This suggests that our ViT-based guidance was effective in improving participant efficiency of glaucoma diagnosis even though it was not particularly effective at significantly improving glaucoma diagnosis accuracy or confidence compared to the no-guidance setting. Furthermore, as also indicated by lower accuracy and confidence results for the CNN-based feedback in the previous section, the CNN-based model did not support effective viewing behavior, as clinicians did not tend to align with this model’s predicted RoIs; Figure 13 exhibits that participants had very high relative entropy for the CNN-based guidance compared to that for ViT-based guidance or no-guidance.

5.8.1 Qualitative analysis of post-experiment survey

Following the completion of the experimental tasks, participants were asked to provide feedback on their experience through a post-experiment survey. Here, we present a thematic analysis of the responses, highlighting significant perceptions and experiences that emerged from the feedback.

User study 1: ViT vs. CNN guidance. The survey aimed to gather insights on the system’s interface and usability, the effectiveness of guidance in identifying potential glaucoma cases, and comparisons among no guidance, and guidance generated by ViT rollout and CNN GradCAM.

Participants generally found the system’s interface intuitive and user-friendly. One participant described the interface as “intuitive, pretty easy to use, with the toggle to hide/show.” (P4) Another highlighted its utility for less experienced individuals, noting it was “Very helpful, especially for residents that have not had much exposure reading glaucoma images.” (P9) The ease of use and the intuitive nature of the system were recurrent themes, suggesting that the interface design effectively supports users across different levels of expertise.

Feedback on the comparative effectiveness of ViT and CNN revealed a preference for ViT guidance. Participants felt ViT guidance was better at highlighting potential pathology, with one stating, “I liked model A better because I feel like it pointed better the possible pathology.” (P9) Another feedback emphasized ViT guidance’s focus, noting, “Model A (ViT) looks at the more relevant parts of the RNFL and GCL maps.” (P15) These responses suggest that ViT guidance may be more effective in guiding users to relevant areas, potentially due to its targeted attention roll-out approach for identifying RoIs compared to the CNN’s Grad-CAM approach.

User study 2: static ViT vs. interactive ViT. In our second user study, participants

evaluated the effectiveness of static and perceptual guidance—both generated by ViT—in aiding the diagnosis of potential glaucoma cases. The study aimed to understand how different types of guidance provided by the same classification model affect glaucoma diagnostic processes and user experience, and if the perceptual RoI cue provide additional benefits (Section 5.3). Responses to the system’s interface and usability were overall positive, with participants describing it as “Easy to use, easy to understand,” (P14) and “reasonably easy to use.” (P15) These positive impressions underscore the system’s design effectiveness in creating an intuitive user experience.

Participants expressed distinct preferences and perceptions regarding the static and perceptual guidance modes. Static guidance was appreciated for its directness and simplicity, offering clear, unchanging insights throughout the diagnostic process. Conversely, perceptual guidance, which updates based on user interaction and attention, was noted for its potential to draw the user’s focus to less obvious areas of interest, enriching the diagnostic exploration. However, some feedback indicated that the interactive nature of perceptual guidance could sometimes distract from critical features, especially when the guidance was updated to highlight larger sections of the image than were useful for decision-making. A participant reflected, “In almost all cases, the requested update made the guidance less helpful rather than more. There was one slide where the requested update was helpful, but in most cases, it ended up highlighting large sections of the image, which wasn’t helpful in decision-making.” (P15)

Moving forward, this feedback suggests a need to refine the balance between providing sufficient interactive engagement through perceptual guidance and maintaining the clarity and focus afforded by static guidance. Participants recommended improvements such as “Needs to track eye movements longer” (P15) and suggested that understanding “how many places on the image/how long you have to look for guidance update would make it better,” (P14) indicating areas for enhancement. Enhancements to the perceptual guidance mode will focus on improving its intuitiveness and relevance, ensuring that updates enhance rather than complicate the diagnostic process. Implementing these suggestions is crucial for leveraging the full potential of advanced AI models like ViT in supporting precise and efficient medical diagnoses.

5.9 Limitations and Future Directions

Given the cutting-edge nature of this work, our study has limitations that we aim to address in future work. One area for improvement is that currently the expert gaze attention accounts only for gaze position and not for fixation duration; this means we potentially miss training our image models on valuable cognitive information that could be inferred from fixation duration (e.g., complexity of a given RoI). In future work, we will leverage fixation location, as well as duration and order for our model training by using directed-graph-based approaches, where each node in the graph could represent time spent on a given RoI, and the directionality could represent fixation order. One of the biggest limitations of our work currently is in our user study 2: as elaborated

in the qualitative feedback of some subjects, the interactive feedback would be more valuable to users if it was more localized to RoIs of highest importance rather than showing multiple global regions. In future work, we aim to address this by implementing a temporal CNN or transformer-based model to predict the next few patches of highest importance conditioned on the previous patches viewed by the user rather than using the current Bayesian approach which simply highlights opposing regions compared to what the user has already viewed.

Ground-truth textual descriptions are not currently available for the OCT images. In future work, given ground-truth text summaries for each image, we could compute the similarity between the user responses to these ground-truth image descriptions to quantify if the interface helped the user make more insightful diagnoses. In future work, we could also use EEG (FRP) and pupillometry to probe more into the user's cognitive state at the time given fixations on particular RoIs. We intend to explore this method of using experts' eyetracking data to provide additional gradient signals during the training process in a future work.

5.10 Conclusions

Out of our four study hypotheses, we were able to effectively show evidence to support two. First, we were successfully able to distill expert knowledge into our ViT model through our expert-aligned loss; this was exhibited by the similar mean divergence exhibited by participants when they experienced no guidance or ViT static guidance in our user study. Second, we observed that ViT-based guidance supported effective glaucoma diagnosis viewing behavior compared to CNN-based guidance. This was exhibited through the higher accuracy and confidence value achieved by participants when they received ViT guidance compared to when they received CNN-based guidance cues. In contrast, our current findings show that our user interface does not significantly enhance participants' clinical performance when using guidance cues from either model (regardless of the back-end model). Future work should address this by using more complex imagery (beyond OCT reports) and by employing a next-patch-prediction guidance approach as opposed to global-level guidance.

6 Vison SwEYEpe: Eye Tracking Based Swype-like Input System

4

In recent years, the proliferation of Virtual Reality (VR) has opened new avenues for immersive experiences. A critical component in this evolving interface is eye tracking,

⁴Haowen "John" Wei proposed this project and created the entire experiment paradigm using Unity and PhysioLabXR. Ziheng "Leo" Li helped the team replace the button press input with gestures. Jingyuan Xu modified the spell correction algorithm and integrated it into the project.



Figure 14: SwEYEpe enables a Swype-like process using eye tracking in virtual reality. The system inherits all the typing keyboard functions of the iOS system, and users can trigger SwEYEpe with a button press. The image above demonstrates that the user was typing “Virtual Reality,” and the spelling correction algorithm carried out the correct suggestion.

a technology that offers a unique window into the user’s attention and intent. Our research specifically focuses on leveraging eye tracking for text entry in VR environments. Eye tracking in VR harnesses the natural and intuitive motion of the eyes, creating a more immersive and efficient method of communication. Previous research has explored text entry methods such as dwell time and button presses [99]; however, these methods have inherent drawbacks. The reliance on dwell time necessitates a waiting period, which constrains interaction freedom. The button-click method, which requires users to press a button when their gaze intersects with each target key, can lead to errors when coordinating hand and eye movements.

In this project, we present SwEYEpe, a novel text input methodology for VR platforms that diverges from conventional keyboard or button-centric interfaces. We explore the integration of eye-tracking technology with a Swype keyboard layout, aiming to reduce physical interaction while increasing throughput rates. We have replaced the button press with a gesture-based pinch using hand tracking in VR as the trigger for swiping, which feels more natural.

In our pilot study, SwEYEpe represents a promising step towards more intuitive and ergonomic VR interactions. Its potential effectiveness and user experience benefits will be further investigated in future studies.

6.1 Prototype Design

6.1.1 Gaze Filter

The initial method we explored involves using a fixation detection algorithm to filter out non-target keys along the gaze path on the keyboard. We employed the Velocity-Threshold Identification (IVT) method for this purpose [77], with the algorithm details provided in Algorithm 2.

We then grouped fixation points by interspersing them with saccade or undefined gaze points and identified the most frequently gazed-at characters in each group. Subsequently,

Algorithm 2 Velocity-Threshold Identification (IVT)

```

1: function IVT(gaze_points)
2:   for  $i \leftarrow 1$  to  $N$  do
3:     if gaze_points[ $i - 1$ ].valid and gaze_points[ $i$ ].valid then
4:        $V1 \leftarrow \textit{gaze\_points}[i - 1].\textit{gazeVector}$ 
5:        $V2 \leftarrow \textit{gaze\_points}[i].\textit{gazeVector}$ 
6:       if ANGULARVELOCITY( $V1, V2$ ) < 100 rad then
7:         gaze_points[ $i$ ].type  $\leftarrow$  "Fixation"
8:       else
9:         gaze_points[ $i$ ].type  $\leftarrow$  "Saccade"
10:      else
11:        gaze_points[ $i$ ].type  $\leftarrow$  "Undefined"
12:   return gaze_points

```

these character sequences were input into a neural network for spelling correction [100], which generated the word candidates.

6.1.2 Experiment Paradigm

Data acquisition, real-time fixation detection, and spell correction are facilitated by PhysioLabXR [1]. The display and interaction components are implemented using Unity 2022.3.10 and are integrated with the Varjo headset, which samples gaze information at a rate of 200 Hz. As illustrated in Figure 14, Unity captures gaze data from the Varjo headset, along with user inputs and event markers. This data is streamed to PhysioLabXR via the Lab Streaming Layer (LSL) [101]. Within PhysioLabXR, the scripting interface processes this data, executing real-time fixation detection algorithms and performing spell corrections. The processed data, including word candidates, are subsequently sent back to Unity through LSL, ensuring a seamless interactive experience.

6.1.3 Spell Correction

Upon acquiring the character sequence, we employ a pretrained character-aware neural language model as described by Kim et al. [102], which is adapted from the method outlined by Jayanthi et al. [100]. This approach diverges from traditional word-based models by focusing on character-level inputs, allowing for a more detailed analysis of text. The model architecture integrates a Convolutional Neural Network (CNN) for character encoding, coupled with a highway network to enhance feature extraction. The resulting encoded characters are then processed by a Long Short-Term Memory (LSTM) network, responsible for predicting subsequent characters.

In keeping with the keyboard paradigm established by Apple, the top four predictions from the model are selected as our spell correction candidates. These candidates are encoded in the Type-Length-Value (TLV) format and subsequently transmitted to Unity

via PhysioLabXR, ensuring a seamless integration of the correction suggestions into the user interface.

6.1.4 Graphic User Interface

Our keyboard layout is modeled on the design principles of Apple's iPhone keyboard, shown in Figure 15, and incorporates the heuristic methodologies inherent in the Swype algorithm. This approach is predicated on the widespread familiarity and ergonomic efficiency of Apple's keyboard design. By aligning with these established standards, our keyboard design aims to offer an intuitive user experience for first time user. As the speech-to-text functionality is not yet implemented, the voice button on the keyboard is disabled.



Figure 15: The system natively supports the basic click typing modality, similar to the iOS mobile system. The keyboard will input the word with the highest probability into the text input box and display three word candidates in the suggestion strip.

Currently SwEYEpe supports dual modalities of input, catering to both button press and SeEYEpe. The first modality is similar to conventional typing methods, allowing users to select individual keys using their gaze and confirm selections by pressing Button 1. This process is complemented by both auditory and visual feedback, informing users when a key is focused upon. Gaze Swype session is activated by press and hold Button 2, concurrently, the system engages in real-time tracking of the user's gaze as it navigates across the keys. Gaze data is subsequently transmitted through the PhysioLabXR and

processed by fixation detection algorithms and spell correction model. Upon processing, the top four correction candidates will be sent back to Unity. The highest-ranked candidate will be automatically populated in the input field, while the remaining three candidates will be displayed on the suggestion bar for user selection. Users can confirm their preferred candidate with a press of Button 1.

6.2 Incorporating Finger Pinch Gestures for Enhanced Usability

To improve usability and offer a more natural interaction interface, the SwEYEpe system has moved away from conventional button presses and towards intuitive gesture-based interactions. As illustrated in Figure 16, we have innovated the user input mechanism by introducing a finger pinch gesture to replace the swipe trigger. Users simply need to bring their index finger and thumb together to initiate the swiping action and release them to end it. This interaction is harmoniously integrated with the Varjo headset's built-in hand-tracking technology, ensuring a smooth and intuitive experience that closely mimics natural hand movements. This advancement not only reduces the learning curve but also significantly amplifies the immersive quality of the virtual environment.

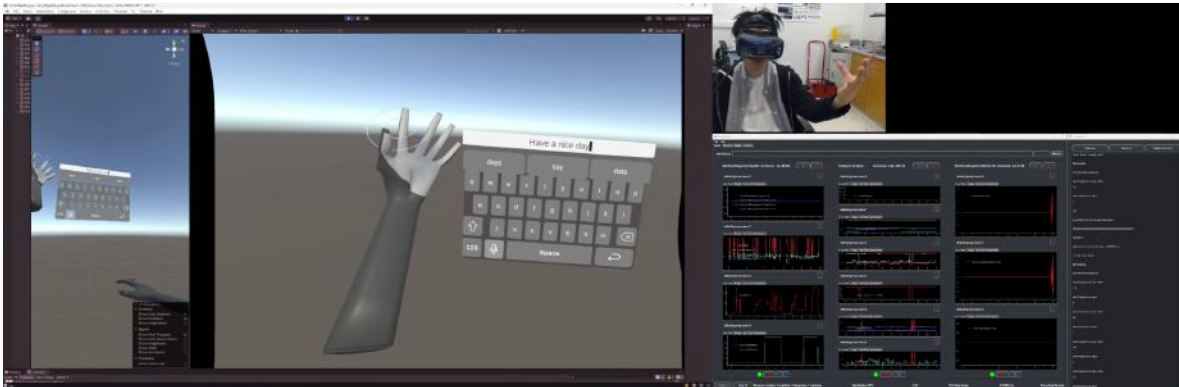


Figure 16: Depicting the transition to gesture-based interactions, the user's palm turnover reveals the keyboard, prompting a finger pinch to trigger the swiping function. The real-time script within PhysioLabXR leverages the process outlined in Section 6.1 to deduce word candidates, which is then communicated to Unity and displayed in the text input box and the suggestion strip.

6.3 Limitation and Future Scope

6.3.1 Limitations

The SwEYEpe system, while innovative, encounters several limitations that currently hinder its full potential within virtual reality environments. One notable limitation is the inaccuracy in hand tracking. The current technology struggles to precisely track hand movements, which can lead to errors in text entry or command execution. Advanced

devices, such as the Apple Vision Pro, offer improved sensors and algorithms that could enhance hand tracking accuracy significantly, potentially resolving this limitation.

Another critical challenge is the robustness of using fixation sequences to filter the gaze path for word candidate identification. The existing method, which relies heavily on the fixation data, often fails to reliably predict the intended input due to the variability in user gaze behavior and environmental conditions. A promising solution would be to develop a geometric model of the keyboard layout, generating ideal gaze paths for a comprehensive lexicon. By comparing the user's actual gaze path with these predefined paths, the system can more accurately determine the intended word based on the similarity of the paths, leveraging advanced computational techniques to minimize errors and enhance prediction accuracy [103].

6.3.2 Future Scope

Looking forward, there is substantial scope for incorporating additional gestures to improve interaction with the SwEYEpe system. Introducing intuitive gestures such as pinching fingers to move the cursor or select words could make the system more user-friendly and reduce reliance on gaze-based inputs, which some users may find less intuitive or physically straining over long periods.

Moreover, integrating multimodal feedback mechanisms, such as haptic or auditory feedback, could significantly enrich the user experience. These feedback mechanisms would provide users with immediate and intuitive responses to their actions, enhancing the interaction quality and potentially reducing the cognitive load and error rate.

Furthermore, leveraging recent advances in machine learning and artificial intelligence could allow for real-time adaptation of the system to individual user behaviors. By continuously learning from each user's interaction patterns, SwEYEpe could dynamically adjust its algorithms to improve accuracy and efficiency, making the system more personalized and effective.

7 Conclusion

This thesis has introduced three pioneering contributions to the domains of medical diagnostics, brain-computer interfaces, and human-computer interaction. Each project demonstrates the transformative potential of integrating advanced computational techniques with user-centric interfaces and diagnostics.

1. **PhysioLabXR: A Platform for Multi-modal Experiments:** PhysioLabXR serves as a robust Python platform for real-time, multi-modal brain-computer interface experiments that merge physiological data streams with extended reality systems. It supports an extensive array of data sources and offers powerful real-time data processing capabilities, which makes it an invaluable tool for neuroscience and human-computer interaction researchers. Its architecture, which optimizes

performance through concurrency and parallelism, ensures efficient handling of complex, high-throughput data streams.

2. **Vision Transformer for Glaucoma Diagnosis:** This project adapts vision transformer technology for the diagnosis of glaucoma, significantly advancing the field of medical imaging. By distilling expert knowledge into a deep learning model, it enhances diagnostic accuracy and reliability, outperforming traditional imaging techniques. Moreover, the flexibility of this approach holds promise for extending its application to other medical reports and imaging types, such as CT scans and MRI scans, potentially leading to earlier and more accurate diagnoses across a range of conditions.
3. **SwEYEpe: Eye Tracking Based Input System:** SwEYEpe revolutionizes text input within virtual reality (VR) environments by integrating eye-tracking technology with a Swype-like keyboard layout. This system addresses ergonomic challenges posed by traditional text entry methods in VR and leverages natural eye movements to provide a more intuitive and efficient communication method. The inclusion of real-time fixation detection and an advanced spell correction model enhances text entry accuracy and speed, significantly improving the user experience in VR settings.

Each project pushes forward its respective field, showcasing the extensive potential of combining eye tracking and machine learning technologies with user-centric interfaces and medical diagnostics. While promising, challenges remain in improving technology accessibility in low-resource settings and further enhancing user interface designs to accommodate a broader range of users with diverse capabilities and preferences.

Future work will focus on addressing these challenges, refining the technologies based on user feedback, and exploring broader applications. Continued interdisciplinary collaboration will be crucial to overcome technical and usability hurdles and to achieve broader adoption of these innovative systems.

8 Acknowledgement

I am immensely grateful for the guidance and support of my advisors throughout the journey of this thesis. My sincere thanks go to Professor Steven K. Feiner, Professor Paul Sajda, and Professor Kaveri Thakoor, whose insights and expertise have been invaluable.

I extend my gratitude to the laboratories that have provided me with the necessary equipment and working space, enriching my research experience. Special thanks to the Laboratory for Intelligent Imaging and Neural Computing (LIINC) under the leadership of Professor Paul Sajda, the Computer Graphics and User Interfaces (CGUI) Lab directed by Professor Steven K. Feiner, and the Artificial Intelligence for Vision Science (AI4VS) Lab headed by Professor Kaveri Thakoor.

My heartfelt appreciation goes to Ziheng "Leo" Li, a cherished friend since our undergraduate days at Worcester Polytechnic Institute. His consistent guidance and support have been pivotal to the completion of this thesis.

I am also thankful for the camaraderie and support of my peers: Ziwen Xie, Jingyuan Xu, Yunxiang Peng, Kuang Sun, Zixi Chen, Xinran Shi, and Yide Li, whose collaboration and friendship have greatly enriched my research experience.

I would like to acknowledge the invaluable support from the CS department, especially Carol Begg and Jinling Quan, as well as the Graduate Student Wellness and Support Services, particularly Charlene Bernasko, for their encouragement and support during challenging times.

Finally, I owe my deepest gratitude to my family, whose endless love and support have been my anchor and sustenance throughout this journey.

References

- [1] Z. L. Li, H. J. Wei, Z. Xie, Y. Peng, J. P. Suh, S. Feiner, and P. Sajda, “Physiolabxr: A python platform for real-time, multi-modal, brain–computer interfaces and extended reality experiments,” *Journal of Open Source Software*, vol. 9, no. 93, p. 5854, 2024. [Online]. Available: <https://doi.org/10.21105/joss.05854>
- [2] A. R. Nikolaev, R. N. Meghanathan, and C. van Leeuwen, “Combining eeg and eye movement recording in free viewing: Pitfalls and possibilities,” *Brain and cognition*, vol. 107, pp. 55–83, 2016.
- [3] S. Koelstra, C. Muhl, M. Soleymani, J.-S. Lee, A. Yazdani, T. Ebrahimi, T. Pun, A. Nijholt, and I. Patras, “Deap: A database for emotion analysis; using physiological signals,” *IEEE transactions on affective computing*, vol. 3, no. 1, pp. 18–31, 2011.
- [4] Z. He, Z. Li, F. Yang, L. Wang, J. Li, C. Zhou, and J. Pan, “Advances in multimodal emotion recognition based on brain–computer interfaces,” *Brain sciences*, vol. 10, no. 10, p. 687, 2020.
- [5] T. Sollfrank, A. Ramsay, S. Perdikis, J. Williamson, R. Murray-Smith, R. Leeb, J. Millán, and A. Kübler, “The effect of multimodal and enriched feedback on smr-bci performance,” *Clinical Neurophysiology*, vol. 127, no. 1, pp. 490–498, 2016.
- [6] P. R. Murphy, R. G. O’connell, M. O’sullivan, I. H. Robertson, and J. H. Balsters, “Pupil diameter covaries with bold activity in human locus coeruleus,” *Human brain mapping*, vol. 35, no. 8, pp. 4140–4154, 2014.
- [7] O. Dimigen and B. V. Ehinger, “Regression-based analysis of combined eeg and eye-tracking data: Theory and applications,” *Journal of vision*, vol. 21, no. 1, pp. 3–3, 2021.
- [8] R. Wang, W. Jo, D. Zhao, W. Wang, B. Yang, G. Chen, and B.-C. Min, “Husformer: A multi-modal transformer for multi-modal human state recognition,” *arXiv preprint arXiv:2209.15182*, 2022.
- [9] D. Nie, H. Zhang, E. Adeli, L. Liu, and D. Shen, “3d deep learning for multi-modal imaging-guided survival time prediction of brain tumor patients,” in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17–21, 2016, Proceedings, Part II 19*. Springer, 2016, pp. 212–220.
- [10] S. Ahn and S. C. Jun, “Multi-modal integration of eeg-fnirs for brain-computer interfaces–current limitations and future directions,” *Frontiers in human neuroscience*, vol. 11, p. 503, 2017.

- [11] C. Kothe and C. Mandel, “A software frame- work for synchronizing a large array of data collection and stimulation devices.” accessed: June 7, 2023. [Online]. Available: <https://github.com/sccn/labstreaminglayer>
- [12] M. Razavi, V. Janfaza, T. Yamauchi, A. Leontyev, S. Longmire-Monford, and J. Orr, “Opensync: An open-source platform for synchronizing multiple measures in neuroscience experiments,” *Journal of neuroscience methods*, vol. 369, p. 109458, 2022.
- [13] “Nirx,” <https://www.nirx.net/>, accessed: June 7, 2023.
- [14] Tobii AB, “Tobii,” Tobii AB, 2023. [Online]. Available: <https://www.tobii.com/>
- [15] Q. Wang, Q. Zhang, W. Sun, C. Boulay, K. Kim, and R. L. Barmaki, “A scoping review of the use of lab streaming layer framework in virtual and augmented reality research,” *Virtual Reality*, pp. 1–16, 2023.
- [16] G. Michalareas, I. M. Rudwan, C. Lehr, P. Gessini, A. Tavano, and M. Grabenhorst, “A scalable and robust system for audience eeg recordings,” *bioRxiv*, pp. 2022–12, 2022.
- [17] J. J. MacInnes, R. A. Adcock, A. Stocco, C. S. Prat, R. P. Rao, and K. C. Dickerson, “Pyneal: open source real-time fmri software,” *Frontiers in neuroscience*, vol. 14, p. 900, 2020.
- [18] T. Baltrušaitis, P. Robinson, and L.-P. Morency, “Openface: an open source facial behavior analysis toolkit,” in *2016 IEEE winter conference on applications of computer vision (WACV)*. IEEE, 2016, pp. 1–10.
- [19] G. Pei, G. Guo, D. Chen, R. Yang, Z. Shi, S. Wang, J. Zhang, J. Wu, and T. Yan, “Brainkilter: a real-time eeg analysis platform for neurofeedback design and training,” *Ieee Access*, vol. 8, pp. 57 661–57 673, 2020.
- [20] B. Developers, “Brainflow,” 2023. [Online]. Available: <https://brainflow.org/>
- [21] Neuropype, “Neuropype,” 2023. [Online]. Available: <https://www.neuropype.io/>
- [22] L. Esch, L. Sun, V. Klüber, S. Lew, D. Baumgarten, P. E. Grant, Y. Okada, J. Haueisen, M. S. Hämäläinen, and C. Dinh, “Mne scan: Software for real-time processing of electrophysiological data,” *Journal of neuroscience methods*, vol. 303, pp. 55–67, 2018.
- [23] K. Srinath, “Python—the fastest growing programming language,” *International Research Journal of Engineering and Technology*, vol. 4, no. 12, pp. 354–357, 2017.

- [24] Y. Renard, F. Lotte, G. Gibert, M. Congedo, E. Maby, V. Delannoy, O. Bertrand, and A. Lécuyer, “Openvibe: An open-source software platform to design, test, and use brain–computer interfaces in real and virtual environments,” *Presence*, vol. 19, no. 1, pp. 35–53, 2010.
- [25] Infineon, *How to Use iMOTION Script Language*, Infineon Technologies AG, Neubiberg, Germany, March 2016. [Online]. Available: https://www.infineon.com/dgdl/Infineon-How_to_Use_iMOTION_Script_Language-ApplicationNotes-v01_02-EN.pdf?fileId=5546d46265487f7b0165c7d2d1633a66
- [26] C. A. Kothe and S. Makeig, “Bcilab: a platform for brain–computer interface development,” *Journal of neural engineering*, vol. 10, no. 5, p. 056014, 2013.
- [27] iMotions, “imotion,” 2023. [Online]. Available: <https://imotions.com/>
- [28] H. Song, A.-D. Nguyen, M. Gong, and S. Lee, “A review of computer vision methods for purpose on computer-aided diagnosis,” *Journal of International Society for Simulation Surgery*, vol. 3, no. 1, pp. 1–8, 2016.
- [29] X.-X. Yin, S. Hadjiloucas, and Y. Zhang, *Pattern classification of medical images: computer aided diagnosis*. Springer, 2017.
- [30] S. Wang, Z. Zhao, X. Ouyang, Q. Wang, and D. Shen, “Chatcad: Interactive computer-aided diagnosis on medical image using large language models,” *arXiv preprint arXiv:2302.07257*, 2023.
- [31] Y. Hagiwara, J. E. W. Koh, J. H. Tan, S. V. Bhandary, A. Laude, E. J. Ciaccio, L. Tong, and U. R. Acharya, “Computer-aided diagnosis of glaucoma using fundus images: A review,” *Computer methods and programs in biomedicine*, vol. 165, pp. 1–12, 2018.
- [32] D. Mirzania, A. C. Thompson, and K. W. Muir, “Applications of deep learning in detection of glaucoma: a systematic review,” *European Journal of Ophthalmology*, vol. 31, no. 4, pp. 1618–1642, 2021.
- [33] M. J. Zedan, M. A. Zulkifley, A. A. Ibrahim, A. M. Moubark, N. A. M. Kamari, and S. R. Abdani, “Automated glaucoma screening and diagnosis based on retinal fundus images using deep learning approaches: a comprehensive review,” *Diagnostics*, vol. 13, no. 13, p. 2180, 2023.
- [34] M. Elsharkawy, A. Sharafeldeen, A. Soliman, F. Khalifa, M. Ghazal, E. El-Daydamony, A. Atwan, H. S. Sandhu, and A. El-Baz, “A novel computer-aided diagnostic system for early detection of diabetic retinopathy using 3d-oct higher-order spatial appearance model,” *Diagnostics*, vol. 12, no. 2, p. 461, 2022.

- [35] Y. Ma, T. Xu, X. Huang, X. Wang, C. Li, J. Jerwick, Y. Ning, X. Zeng, B. Wang, Y. Wang *et al.*, “Computer-aided diagnosis of label-free 3-d optical coherence microscopy images of human cervical tissue,” *IEEE Transactions on Biomedical Engineering*, vol. 66, no. 9, pp. 2447–2456, 2019.
- [36] A. ElTanboly, M. Ismail, A. Shalaby, A. Switala, A. El-Baz, S. Schaal, G. Gimel’farb, and M. El-Azab, “A computer-aided diagnostic system for detecting diabetic retinopathy in optical coherence tomography images,” *Medical physics*, vol. 44, no. 3, pp. 914–923, 2017.
- [37] C. J. Kelly, A. Karthikesalingam, M. Suleyman, G. Corrado, and D. King, “Key challenges for delivering clinical impact with artificial intelligence,” *BMC medicine*, vol. 17, pp. 1–9, 2019.
- [38] M. Kim, J. C. Han, S. H. Hyun, O. Janssens, S. Van Hoecke, C. Kee, and W. De Neve, “Medinoid: computer-aided diagnosis and localization of glaucoma using deep learning,” *Applied Sciences*, vol. 9, no. 15, p. 3064, 2019.
- [39] A. A. Adegun, S. Viriri, and R. O. Ogundokun, “Deep learning approach for medical image analysis,” *Computational Intelligence and Neuroscience*, vol. 2021, pp. 1–9, 2021.
- [40] D. Shen, G. Wu, and H.-I. Suk, “Deep learning in medical image analysis,” *Annual review of biomedical engineering*, vol. 19, pp. 221–248, 2017.
- [41] B. J. Erickson, P. Korfiatis, Z. Akkus, and T. L. Kline, “Machine learning for medical imaging,” *Radiographics*, vol. 37, no. 2, pp. 505–515, 2017.
- [42] S. M. Kumar and R. Gunasundari, “Computational intelligence in eye disease diagnosis: a comparative study,” *Medical & Biological Engineering & Computing*, vol. 61, no. 3, pp. 593–615, 2023.
- [43] H. Jiang, Z. Diao, T. Shi, Y. Zhou, F. Wang, W. Hu, X. Zhu, S. Luo, G. Tong, and Y.-D. Yao, “A review of deep learning-based multiple-lesion recognition from medical images: classification, detection and segmentation,” *Computers in Biology and Medicine*, p. 106726, 2023.
- [44] X. Chen, X. Wang, K. Zhang, K.-M. Fung, T. C. Thai, K. Moore, R. S. Mannel, H. Liu, B. Zheng, and Y. Qiu, “Recent advances and clinical applications of deep learning in medical image analysis,” *Medical Image Analysis*, vol. 79, p. 102444, 2022.
- [45] X. Chen, Y. Xue, X. Wu, Y. Zhong, H. Rao, H. Luo, and Z. Weng, “Deep learning-based system for disease screening and pathologic region detection from optical coherence tomography images,” *Translational Vision Science & Technology*, vol. 12, no. 1, pp. 29–29, 2023.

- [46] X. Wang, H. Chen, A.-R. Ran, L. Luo, P. P. Chan, C. C. Tham, R. T. Chang, S. S. Mannil, C. Y. Cheung, and P.-A. Heng, “Towards multi-center glaucoma oct image screening with semi-supervised joint structure and function multi-task learning,” *Medical Image Analysis*, vol. 63, p. 101695, 2020.
- [47] L. Fang, C. Wang, S. Li, H. Rabbani, X. Chen, and Z. Liu, “Attention to lesion: Lesion-aware convolutional neural network for retinal optical coherence tomography image classification,” *IEEE transactions on medical imaging*, vol. 38, no. 8, pp. 1959–1970, 2019.
- [48] B. Hassan, S. Qin, R. Ahmed, T. Hassan, A. H. Taguri, S. Hashmi, and N. Werghi, “Deep learning based joint segmentation and characterization of multi-class retinal fluid lesions on oct scans for clinical use in anti-vegf therapy,” *Computers in Biology and Medicine*, vol. 136, p. 104727, 2021.
- [49] Y. Gao, M. Zhou, and D. N. Metaxas, “Utnet: a hybrid transformer architecture for medical image segmentation,” in *Medical Image Computing and Computer Assisted Intervention–MICCAI 2021: 24th International Conference, Strasbourg, France, September 27–October 1, 2021, Proceedings, Part III 24*. Springer, 2021, pp. 61–71.
- [50] M. S. Hossain, G. M. Shahriar, M. M. Syeed, M. F. Uddin, M. Hasan, S. Shivam, and S. Advani, “Region of interest (roi) selection using vision transformer for automatic analysis using whole slide images,” *Scientific Reports*, vol. 13, no. 1, p. 11314, 2023.
- [51] S. Abnar and W. Zuidema, “Quantifying attention flow in transformers,” *arXiv preprint arXiv:2005.00928*, 2020.
- [52] H. Chefer, S. Gur, and L. Wolf, “Transformer interpretability beyond attention visualization,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 782–791.
- [53] —, “Generic attention-model explainability for interpreting bi-modal and encoder-decoder transformers,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 397–406.
- [54] A. B. Mbakwe, L. Wang, M. Moradi, and I. Lourentzou, “Hierarchical vision transformers for disease progression detection in chest x-ray images,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2023, pp. 685–695.
- [55] J. Salas, R. Zukerman, O. Moussa, S. Z. Gu, A. Leshno, J. M. Liebmann, G. Cioffi, and K. Thakoor, “Impact of ai on retrospective glaucoma diagnosis,” *Investigative Ophthalmology & Visual Science*, vol. 64, no. 8, pp. 385–385, 2023.

- [56] X. Xie, J. Niu, X. Liu, Z. Chen, S. Tang, and S. Yu, “A survey on incorporating domain knowledge into deep learning for medical image analysis,” *Medical Image Analysis*, vol. 69, p. 101985, 2021.
- [57] L. Li, M. Xu, X. Wang, L. Jiang, and H. Liu, “Attention based glaucoma detection: A large-scale database and cnn model,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 10 571–10 580.
- [58] T. T. Brunyé, T. Drew, D. L. Weaver, and J. G. Elmore, “A review of eye tracking for understanding and improving diagnostic interpretation,” *Cognitive research: principles and implications*, vol. 4, no. 1, pp. 1–16, 2019.
- [59] C. F. Nodine, H. L. Kundel, S. C. Lauver, and L. C. Toto, “Nature of expertise in searching mammograms for breast masses,” *Academic radiology*, vol. 3, no. 12, pp. 1000–1006, 1996.
- [60] R. Zhang, J. He, S. Shi, X. Kang, W. Chai, M. Lu, Y. Liu, E. Haihong, Z. Ou, and M. Song, “Computer-aided diagnosis of ophthalmic diseases using oct based on deep learning: A review,” in *Human Centered Computing: 5th International Conference, HCC 2019, Čačak, Serbia, August 5–7, 2019, Revised Selected Papers 5*. Springer, 2019, pp. 615–625.
- [61] M. Bhattacharya, S. Jain, and P. Prasanna, “Radiotransformer: a cascaded global-focal transformer for visual attention-guided disease classification,” in *European Conference on Computer Vision*. Springer, 2022, pp. 679–698.
- [62] A. Patra, Y. Cai, P. Chatelain, H. Sharma, L. Drukker, A. T. Papageorghiou, and J. A. Noble, “Efficient ultrasound image analysis models with sonographer gaze assisted distillation,” in *Medical Image Computing and Computer Assisted Intervention—MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part IV 22*. Springer, 2019, pp. 394–402.
- [63] J. N. Stember, H. Celik, E. Krupinski, P. D. Chang, S. Mutasa, B. J. Wood, A. Lignelli, G. Moonis, L. Schwartz, S. Jambawalikar *et al.*, “Eye tracking for deep learning segmentation using convolutional neural networks,” *Journal of digital imaging*, vol. 32, pp. 597–604, 2019.
- [64] S. Kaushal, Y. Sun, R. Zukerman, R. W. Chen, and K. A. Thakoor, “Detecting eye disease using vision transformers informed by ophthalmology resident gaze data,” in *2023 45th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, 2023, pp. 1–4.
- [65] C. Kumar, R. Hedeshy, I. S. MacKenzie, and S. Staab, “Tagswipe: Touch assisted gaze swipe for text entry,” in *Proceedings of the 2020 chi conference on human factors in computing systems*, 2020, pp. 1–12.

- [66] A. Hummer, M. Ritter, M. Tik, A. Ledolter, M. Woletz, G. Holder, S. O. Dumoulin, U. Schmidt-Erfurth, and C. Windischberger, “Eyetracker-based gaze correction for robust mapping of population receptive fields,” *Neuroimage*, vol. 142, pp. 211–224, 2016.
- [67] ZeroMQ. (2021) Zeromq - the intelligent transport layer. [Online]. Available: <https://zeromq.org/>
- [68] Unity Technologies, “Unity,” 2005, accessed: May 10, 2023. [Online]. Available: <https://unity.com/>
- [69] J. W. Peirce, “Psychopy—psychophysics software in python,” *Journal of neuroscience methods*, vol. 162, no. 1-2, pp. 8–13, 2007.
- [70] P. Lapborisuth, S. Koorathota, and P. Sajda, “Pupil-linked arousal modulates network-level eeg signatures of attention reorienting during immersive multitasking,” *Journal of Neural Engineering*, 2023.
- [71] S. C. Koorathota, “Multimodal deep learning systems for analysis of human behavior, preference, and state,” Ph.D. dissertation, Columbia University, 2023.
- [72] H. Wei, Z. Li, A. D. Galvan, Z. Su, X. Zhang, K. Pahlavan, and E. T. Solovey, “Indexpen: Two-finger text input with millimeter-wave radar,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 2, pp. 1–39, 2022.
- [73] “BioSemi ActiveTwo EEG system,” <https://www.biosemi.com/products.htm>, 2021, accessed: May 10, 2023.
- [74] V. Technologies, “Varjo xr-3,” <https://varjo.com/products/xr-3/>, 2021, accessed: May 12, 2023.
- [75] S. E. Kober, M. Ninaus, E. V. Friedrich, and R. Scherer, “Bci and games: playful, experience-oriented learning by vivid feedback?” in *Brain-Computer Interfaces Handbook*. CRC Press, 2018, pp. 209–234.
- [76] P. Rämä and T. Baccino, “Eye fixation-related potentials (efrps) during object identification,” *Visual Neuroscience*, vol. 27, no. 5-6, pp. 187–192, 2010.
- [77] D. D. Salvucci and J. H. Goldberg, “Identifying fixations and saccades in eye-tracking protocols,” in *Proceedings of the 2000 symposium on Eye tracking research & applications*, 2000, pp. 71–78.
- [78] L. Itti, C. Koch, and E. Niebur, “A model of saliency-based visual attention for rapid scene analysis,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 20, no. 11, pp. 1254–1259, 1998.

- [79] J. Steil, M. X. Huang, and A. Bulling, “Fixation detection for head-mounted eye tracking based on visual similarity of gaze targets,” in *Proceedings of the 2018 ACM Symposium on eye tracking research & applications*, 2018, pp. 1–9.
- [80] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright *et al.*, “Scipy 1.0: fundamental algorithms for scientific computing in python,” *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.
- [81] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, “Cython: The best of both worlds,” *Computing in Science & Engineering*, vol. 13, no. 2, pp. 31–39, 2010.
- [82] P. Lapborisuth, S. Koorathota, Q. Wang, and P. Sajda, “Integrating neural and ocular attention reorienting signals in virtual reality,” *Journal of Neural Engineering*, vol. 18, no. 6, p. 066052, 2022.
- [83] J. E. Kamienskowski, M. J. Ison, R. Q. Quiroga, and M. Sigman, “Fixation-related potentials in visual search: A combined eeg and eye tracking study,” *Journal of vision*, vol. 12, no. 7, pp. 4–4, 2012.
- [84] L. A. Farwell and E. Donchin, “Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials,” *Electroencephalography and clinical Neurophysiology*, vol. 70, no. 6, pp. 510–523, 1988.
- [85] D. J. Krusienski, E. W. Sellers, D. J. McFarland, T. M. Vaughan, and J. R. Wolpaw, “Toward enhanced p300 speller performance,” *Journal of neuroscience methods*, vol. 167, no. 1, pp. 15–21, 2008.
- [86] Interuniversity Microelectronics Centre, “Neuropixels,” 2023, accessed: Oct 28, 2023. [Online]. Available: <https://www.neuropixels.org/>
- [87] B. Thyelfors and A. Negrel, “The global impact of glaucoma.” *Bulletin of the World Health Organization*, vol. 72, no. 3, p. 323, 1994.
- [88] D. Huang, E. A. Swanson, C. P. Lin, J. S. Schuman, W. G. Stinson, W. Chang, M. R. Hee, T. Flotte, K. Gregory, C. A. Puliafito *et al.*, “Optical coherence tomography,” *science*, vol. 254, no. 5035, pp. 1178–1181, 1991.
- [89] M. Usman, M. M. Fraz, and S. A. Barman, “Computer vision techniques applied for diagnostic analysis of retinal oct images: a review,” *Archives of Computational Methods in Engineering*, vol. 24, pp. 449–465, 2017.
- [90] R. Nuzzi, G. Boscia, P. Marolo, and F. Ricardi, “The impact of artificial intelligence and deep learning in eye diseases: a review,” *Frontiers in Medicine*, vol. 8, p. 710329, 2021.

- [91] C. Ployon, R. Duval, M. C. Boucher, and F. Cheriet, “Focused attention in transformers for interpretable classification of retinal images,” *Medical Image Analysis*, vol. 82, p. 102608, 2022.
- [92] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [93] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016. [Online]. Available: <https://dx.doi.org/10.1109/CVPR.2016.90>
- [94] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.
- [95] T. Technology, “Tobii i-vt fixation filter default values whitepaper,” <https://stemedhub.org/resources/2174>, 2014.
- [96] P. Blignaut, “Fixation identification: The optimum threshold for a dispersion algorithm,” *Attention, Perception, & Psychophysics*, vol. 71, pp. 881–895, 2009.
- [97] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020.
- [98] A. Leshno, E. Tsamis, S. Hirji, G. A. Gomide, N. Harizman, C. G. De Moraes, A. G. Shukla, G. A. Cioffi, D. C. Hood, and J. M. Liebmann, “Detecting established glaucoma using oct alone: Utilizing an oct reading center in a real-world clinical setting,” *Translational Vision Science & Technology*, vol. 13, no. 1, pp. 4–4, 2024.
- [99] V. Rajanna and J. P. Hansen, “Gaze typing in virtual reality: impact of keyboard design, selection method, and motion,” in *Proceedings of the 2018 ACM symposium on eye tracking research & applications*, 2018, pp. 1–10.
- [100] S. M. Jayanthi, D. Pruthi, and G. Neubig, “Neuspell: A neural spelling correction toolkit,” *arXiv preprint arXiv:2010.11085*, 2020.
- [101] C. Kothe *et al.*, “Lab streaming layer (lsl),” 2014.
- [102] Y. Kim, Y. Jernite, D. Sontag, and A. Rush, “Character-aware neural language models,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.
- [103] A. Kurauchi, W. Feng, A. Joshi, C. Morimoto, and M. Betke, “Eyeswipe: Dwell-free text entry using gaze paths,” in *Proceedings of the 2016 chi conference on human factors in computing systems*, 2016, pp. 1952–1956.

A Appendix

1

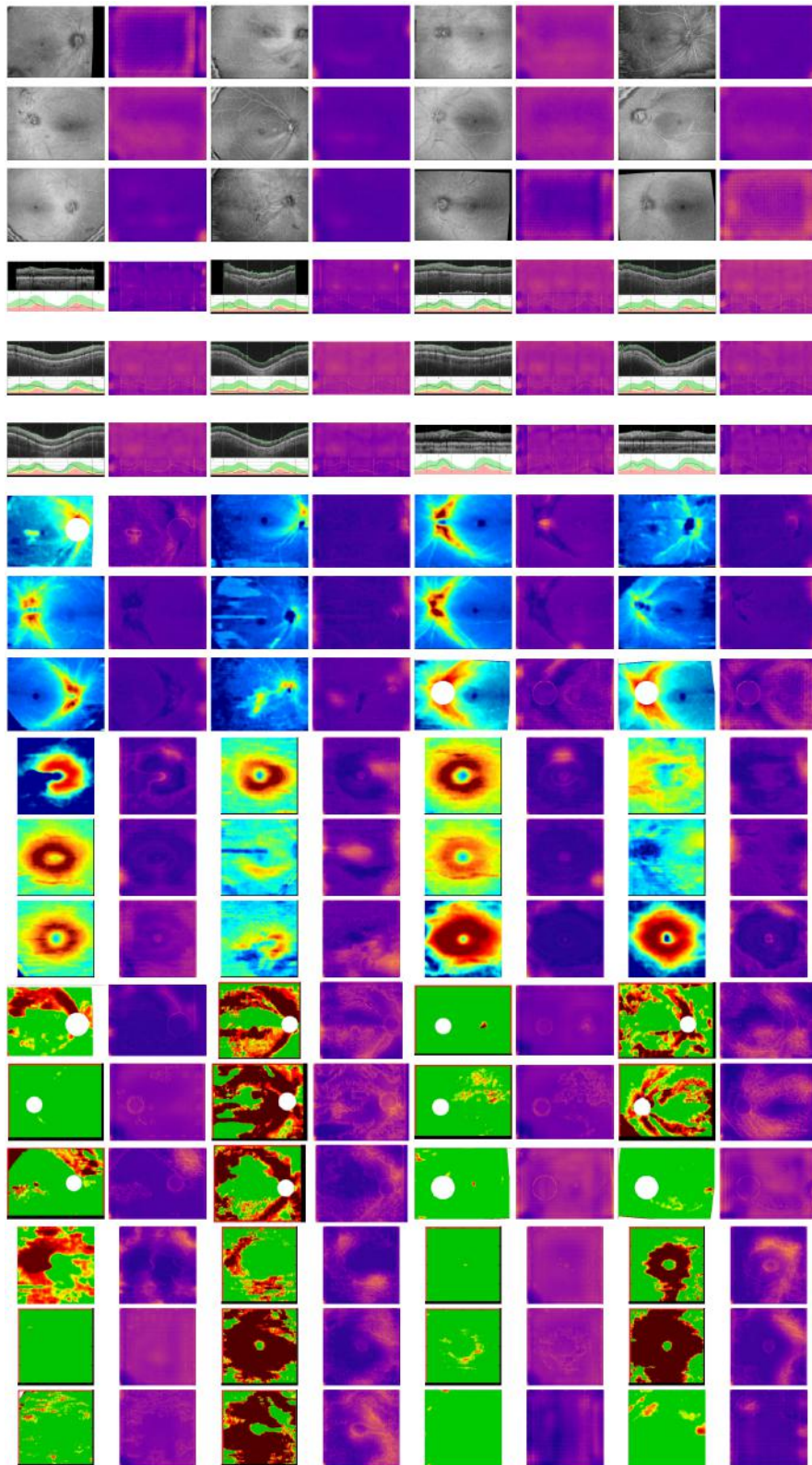


Figure 17: Sample RoIs: Inception-V4's Grad-CAM

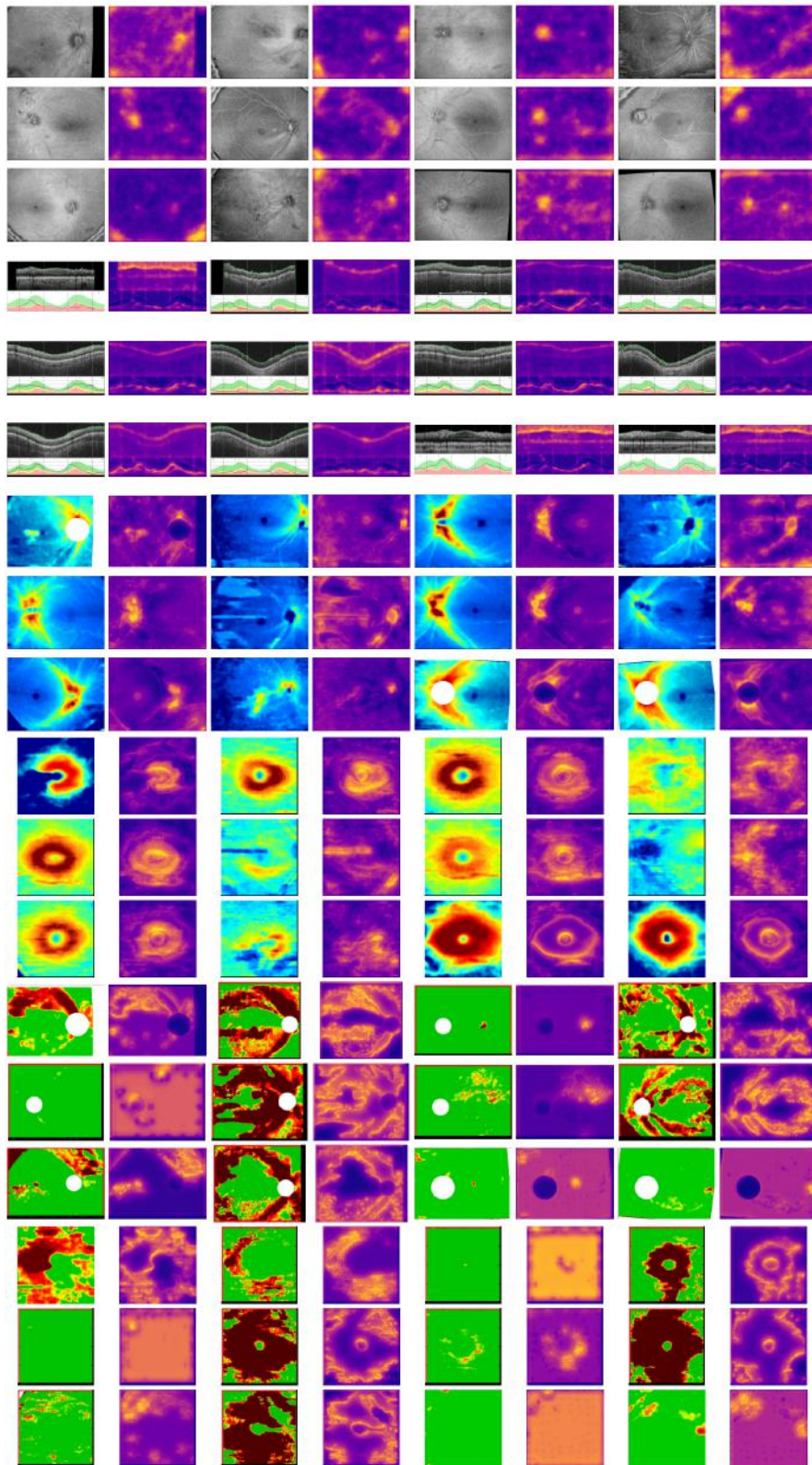


Figure 18: Sample RoIs: VGG19's Grad-CAM

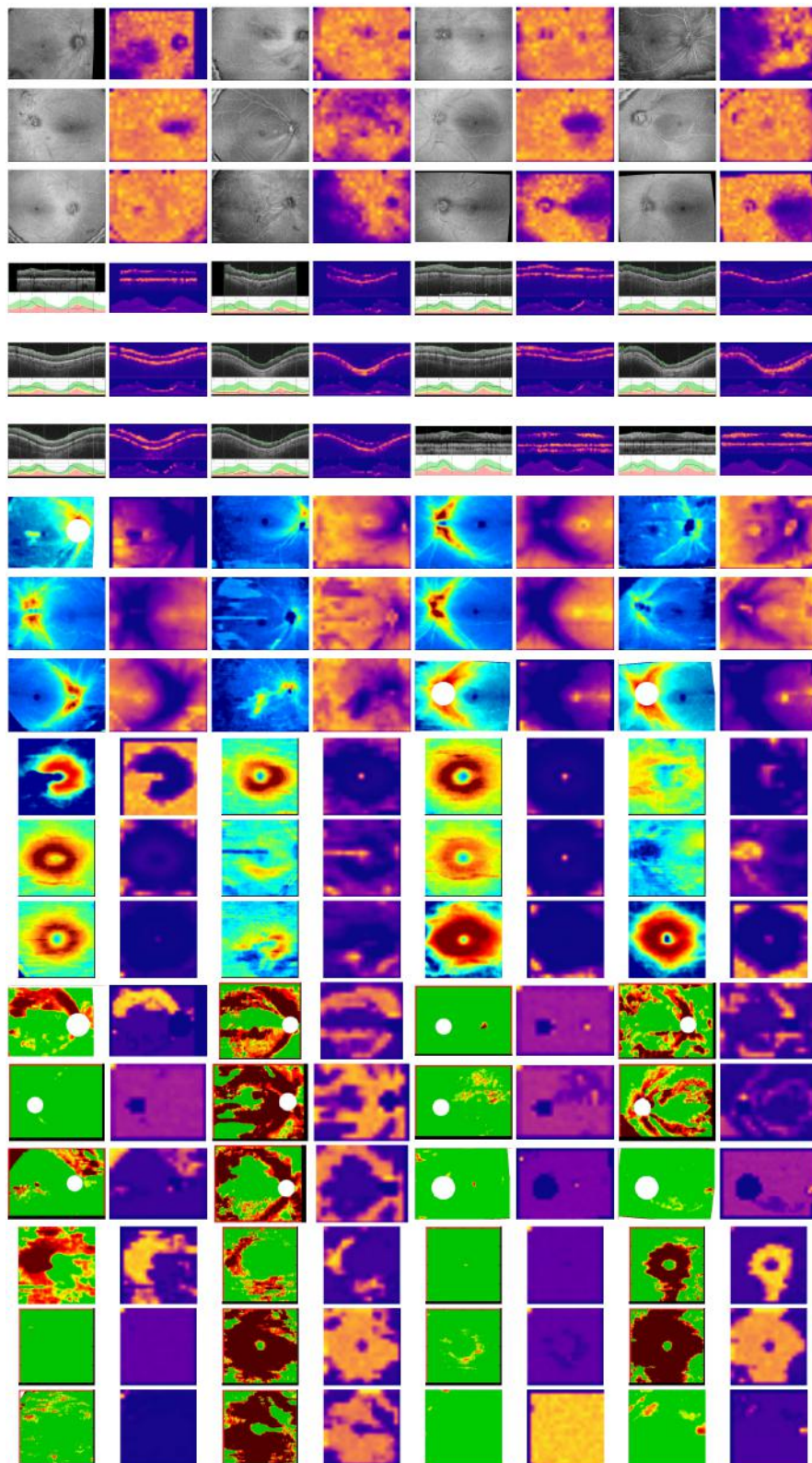


Figure 19: Sample RoIs: Attention rollout from the base ViT, showing the rollout of the first and only attention layer.

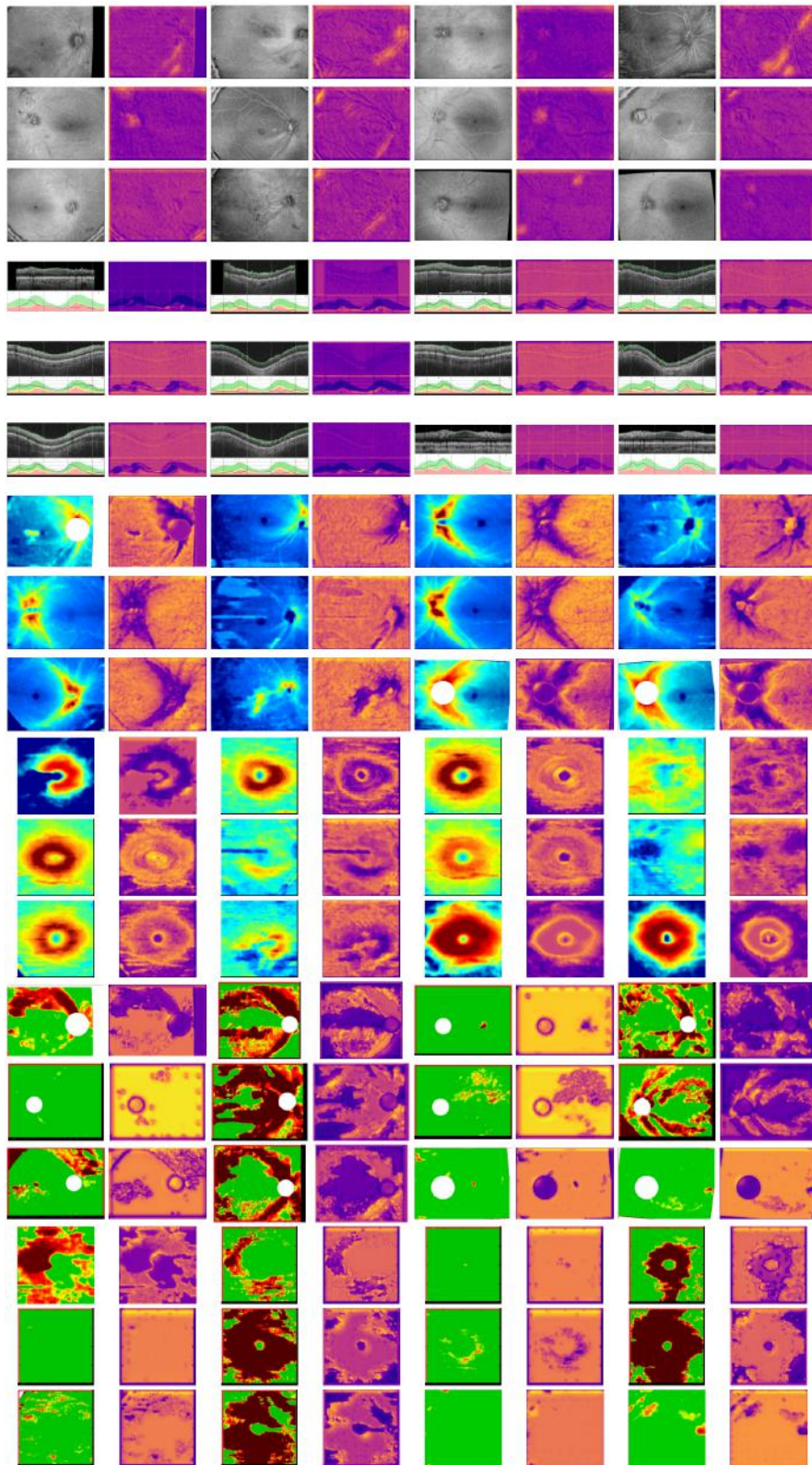


Figure 20: Sample RoIs: ResNet50's Grad-CAM

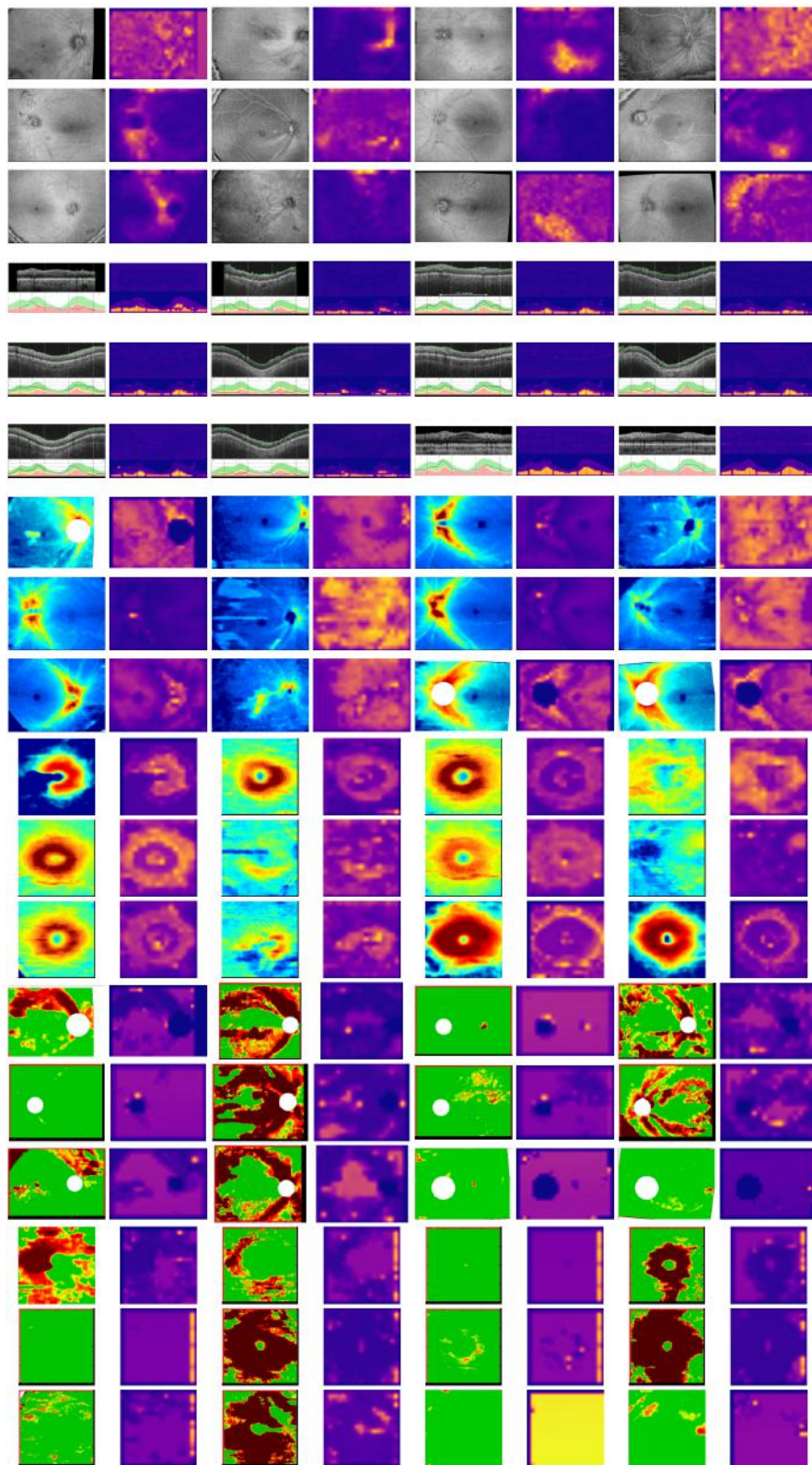


Figure 21: Sample RoIs: Attention rollout from the pretrained ViT, showing the rollout of the final attention layer.