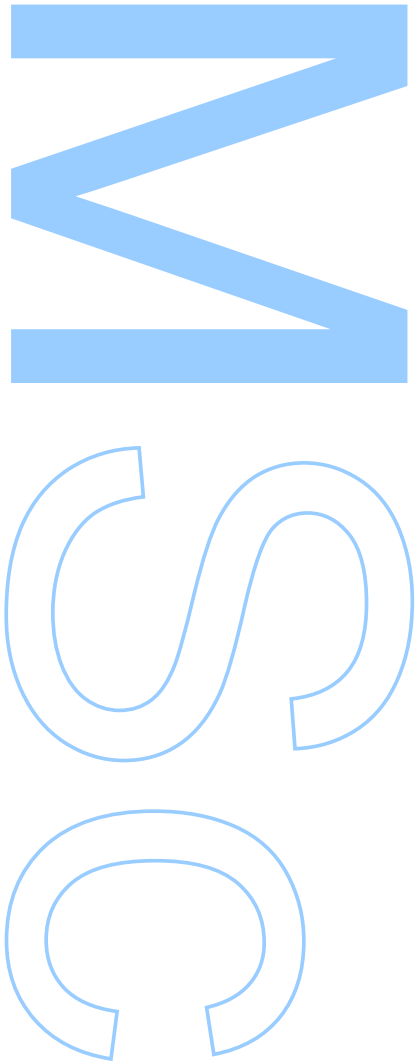
Seshat: A platform for seamless integration of AI processing models in real-time medical imaging systems

João Malheiro
Master in Computer Science
Department of Computer Science
Faculty of Sciences of the University of Porto
2025



Seshat: A platform for seamless integration of AI processing models in real-time medical imaging systems

João Malheiro

Dissertation carried out as part of the Master in Computer Science

Department of Computer Science

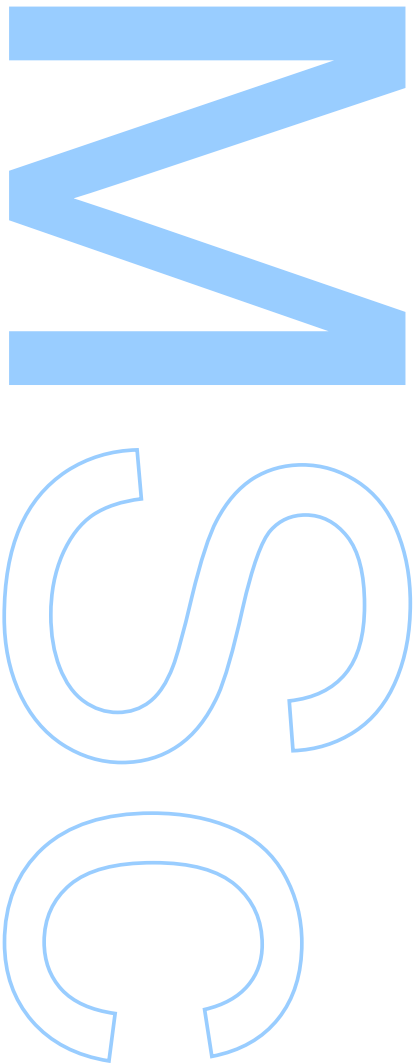
2025

Supervisor

Prof. Dr. José Paulo Leal, Faculty of Sciences and INESC TEC

Co-Supervisor

Prof. Dr. Miguel Coimbra, Faculty of Sciences and INESC TEC



Acknowledgements

In the first place, I sincerely thank my supervisor, José Paulo Leal and co-supervisor, Miguel Coimbra, for their incessant patience, guidance, support, and valuable insights and teachings along this research journey. Thank you for devoting your time to me and helping me achieve this milestone with your experience and wisdom, while also teaching me so much. I truly appreciate your commitment to me and this project.

I would also like to thank the BioImaging group for providing a great environment for researching, with a delightful combination of experience and entertainment throughout the whole project. Thank you for making all the meetings enjoyable and enlightening me with valuable knowledge and helpful advice.

I extend my gratitude to the Department of Computer Science of the Faculty of Science of the University of Porto and the entire staff for allowing me to develop this project in such a great environment for my personal and academic growth.

I deeply want to thank INESC TEC and IPO-Porto for contributing with innovative ideas, data and knowledge. Additionally, I want to thank the EndoRadiomics project for financing me within the scope of the Recovery and Resilience Mechanism of the European Union with reference 2024.07584.IACDC/2024.

My appreciation also goes to all the evaluators who participated in the usability test session and took the time to fill out all the forms. Your feedback and opinions were of extreme importance for the validity, reliability, and planning of this project and provided new perspectives that enhanced the depth of this work.

Finally, my deepest gratitude goes to my family, my partner, my closest friends, and my beloved pets. Their unconditional love and unwavering support were an anchor throughout this journey. Though their encouragement often came from outside the academic sphere, it is an invisible backbone of my character, sustaining me through challenges and celebrations along the way.

Summarising, without everyone mentioned here, the collective effort and all the help given, this journey would have been much more difficult. I am forever grateful for learning and carrying so much from each of you.

Resumo

Integrar modelos de inteligência artificial ou machine learning numa plataforma clínica e de investigação acarreta um desafio completo. As soluções existentes são tipicamente restritas, com falta de capacidades de tempo real e não desenhadas para integrar modelos de outros investigadores. Esta dissertação aborda a criação de uma plataforma flexível, com uma aplicação web capaz de integrar múltiplos modelos de inteligência artificial em imagens médicas.

O principal objetivo é desenhar, implementar e validar a Seshat, uma plataforma de software modular que permite que investigadores e profissionais de saúde incorporem modelos de inteligência artificial em imagens médicas pré-gravadas e em tempo real.

Este projeto segue uma arquitetura baseada em três pilares principais. O Repositório, que gerencia o armazenamento de exames médicos, os seus frames e anotações, organizados numa estrutura de índices temporais. O Controlador, que orquestra a plataforma sendo o backend principal da plataforma, que monitoriza o Repositório, gerencia anotadores e é responsável pela sua comunicação com a Interação. A Interação, uma interface web que permite aos utilizadores a visualização dos exames médicos e a anotação dos frames dos exames.

A integração dos anotadores é feita através de uma API dedicada. Os anotadores agem como um servidor HTTP que comunica com o Controlador notificando-o de estados, registos e alterações. Esta API permite varios anotadores em simultâneo e disponibiliza um estilo ”plug-and-play”de integração de anotadores.

Esta plataforma apresentou resultados empíricos que sustentam a prática médica, obtendo uma média de 25 frames por segundo e um atraso de 227-295 ms para display de frames em tempo real. Um teste de usabilidade foi realizado e revelou a eficiência da integração dos anotadores e a usabilidade geral da interface para analisar anotações.

A plataforma Seshat mostra uma solução eficaz para integrar modelos de anotação de IA em uma plataforma unificada de análise de imagens médicas. As suas principais contribuições são uma arquitetura de armazenamento altamente versátil e específica e um software que organiza a troca de informação entre os anotadores e os exames. Devido à versatilidade da arquitetura, a Seshat possibilita uma multitude de casos de uso e expansões em trabalho futuro para diversas adaptações como plataforma de anotação, tanto para fins de pesquisa como clínicos.

Palavras chave:

Desenvolvimento de Software, Anotadores, Integração, API, Bio Imaging: Desenvolvimento de Software, Anotadores, Integração, API, Bio Imaging

Abstract

Integrating artificial intelligence or machine learning models into clinical and research platforms presents significant challenges. Existing solutions are typically restrictive, lacking real-time capabilities and not designed to incorporate models from other researchers. This dissertation addresses the creation of a platform that integrates multiple artificial intelligence models for medical imaging analysis.

The primary objective is to design, implement, and validate Seshat, a software platform that allows researchers and healthcare professionals to incorporate AI and ML models into pre-recorded and real-time medical exams.

This project follows an architecture based on three main pillars. The Repository, which manages the storage of medical exam frames and annotations, is organised in a time index structure. The Controller, which orchestrates the platform as the main backend, monitors the Repository, manages annotators, and handles communication with the Interaction component. The Interaction is a web interface that allows users to visualise medical exams and annotate exam frames.

Annotator integration is achieved via a dedicated API. The annotators function as isolated servers that communicate with the Controller, notifying it of states, registrations, and changes. This API supports multiple simultaneous annotators and provides a plug-and-play style integration approach.

The Seshat platform represents an effective solution for integrating AI annotation models into a unified medical imaging analysis platform. Its main contributions are a highly versatile and specific storage architecture and software that organises information exchange between annotators and exams. Due to the architecture's versatility, Seshat enables multiple use cases and future expansions for various adaptations as an annotation platform, suitable for both research and clinical purposes.

Keywords: Software development, Annotators, Integration, API, Bio Imaging: Software development, Annotators, Integration, API, Bio Imaging

Table of Contents

List of Figures

1. Introduction

Medical imaging systems play a crucial role in the diagnosis, monitoring, and treatment of a vast range of medical conditions. These systems produce sequences of images—often in video form—that must be carefully interpreted by trained medical professionals. Such interpretation tasks are fundamental to modern clinical workflows and can severely impact the outcomes of the medical reports. However, the sheer volume and complexity of visual data generated can overwhelm even experienced practitioners, particularly in settings where time pressure or case load is intense.

Detecting anomalies such as tumours, lesions, or polyps is especially critical when early-stage identification can dramatically influence the diagnosis. Recent systematic reviews with meta□analyses have shown that artificial intelligence–assisted endoscopy significantly improves detection rates for gastrointestinal tumors and reduces miss rates when compared to conventional endoscopic procedures [1**?** –3]. Yet this task is not only demanding but also highly prone to variability. Factors such as fatigue, distraction, or subtle visual patterns may lead to missed diagnoses. Even expert clinicians may overlook relevant features in medical images, particularly when they are faint, obscured, or occur in unexpected or overlooked anatomical locations.

In this context, Artificial Intelligence (AI) and Machine Learning (ML) are increasingly being explored as tools to augment human interpretation and improve diagnostic accuracy[**?** **?**]. These models, once trained, can operate with a level of consistency, speed, and robustness that complements human capabilities, particularly in repetitive or attention-demanding tasks.

These AI/ML models are referred to as annotators since they are a tool that, given a certain image (also referred to as a medical exam or exam), automatically generates an analysis, interpretation, or mark-up of the visual data, such as a classification or a segmentation.

1.1. Motivation

Medical imaging video streams, in particular, present a unique opportunity and challenge for AI-assisted systems. These streams consist of long sequences of image frames, which may have significant information for the medical reports. Unlike static imaging, video-based exams (for example, endoscopies or ultrasound scans) are dynamic, context-dependent, and involve continuous motion and change in perspective. Moreover, ML

models intended for use in such environments must process a large number of frames rapidly and generate meaningful, interpretable outputs in real time. Studies have begun exploring real□time implementations of AI in endoscopy, including edge computing devices capable of lesion classification and live photodocumentation, showing both feasibility and high diagnostic performance in clinical settings [4, 5].

Despite the enormous potential and the narratives showing the large advantages of AI integration in the medical field [6, 7], most available solutions have proprietary ecosystems, developed and maintained by vendors of medical devices or specialised imaging software. The developers of these systems produce their own models and obscure technical details that would give valuable feedback and justify model outcomes. They also offer minimal extensibility and restrict how researchers or developers can interact with or improve upon models. As a result, teams wishing to test new algorithms, adapt models, or integrate research prototypes into operational workflows face significant delays in development and suffer a harsh conversion from the development stages into deployment. Moreover, many of these systems are designed with a narrow scope in mind, focusing on a specific imaging modality or medical field. This makes them less useful in academic and research contexts, where multi-purpose tools are needed to support development across different domains. Combining this with the proprietary approach limits the use of these systems not only to medical practitioners who cannot justify annotations, but also to developers who lack the freedom of development. Additionally, existing platforms hide data and/or rarely provide robust feedback mechanisms that help developers better train and develop models, as well as medical professionals complement their medical reports. Rather than focusing on a single disease, medical speciality, or use case, there is a pressing need for infrastructure that can support the dynamic integration of diverse annotation models, enable traceable and reproducible evaluations, and connect the development and deployment of ML models. Such a platform would support medical professionals in reports and provide researchers with the tools necessary to iterate rapidly from prototype to practical implementation.

1.2. Objectives

The goal of this dissertation is to develop a general-purpose, real-time medical image annotation platform capable of supporting multiple medical imaging modalities and offering a responsive user experience for developers, researchers, and clinicians. This system should be conceived not as a narrowly scoped platform for a specific medical field, but

rather as a flexible and extensible foundation on which diagnostic and research pipelines can be constructed, evaluated, and refined. The platform must fulfil the following functional requirements:

- Ingest video streams from medical diagnostic systems (for example, endoscopies) at 25 FPS to support real-time annotation during exams.
- Integrate multiple annotators developed by researchers to label video frames
- Display annotated videos and manage the annotation workflow without latency

It should also allow for the following use cases:

- Record and annotate pre-existing exams for retrospective analysis
- Automatically compare annotations across different annotators
- Manage a repository of annotated exams for research or model training

1.3. Approach

This dissertation proposes Seshat, a flexible and extensible platform for the real-time annotation of medical videos, supporting researchers, developers, and medical practitioners. At its core is a unifying data model. This format underpins the system's architecture, acting as a persistent, structured representation of the exam. This open format represents how the exams are stored and provides a standardised representation that is shared across all platform components and external annotators, ensuring interoperability while maintaining a persistent record of exams and their annotations. The format is designed to be both machine-processable and understandable to humans, with explicit indexing of video frames and decoupled storage of raw data from annotations. Every exam must be stored following this format, with Seshat supporting its conversion from video feeds.
Annotation itself is driven by a pluggable architecture based on annotator modules, each of which encapsulates a specific ML model. These annotators interface with the core platform through a standardised and versioned application programming interface (API). This design allows models to be seamlessly integrated and executed. By enforcing a consistent communication protocol, Seshat enables annotators to be hot-swapped, used concurrently, or compared under identical operating conditions. All annotations produced by the annotators are written back into the file system structure in a traceable and reversible manner. These annotations are organised based on their model characteristics

and the frame analysed.

To close the gap between system-level functionality and user interaction, Seshat includes a web-based interface. This component is a key pillar of the platform, designed to allow both developers and clinicians to offer intuitive and responsive control over the annotation workflow. Built to serve the medical images, the interface provides real-time playback of annotated frames, overlays showing annotations, and controls for (make list) navigating the exam timeline, model interchangeability, annotation, real-time annotation, exam selection (with live feed available), etc..

Recognising the practical limitations often encountered in medical environments, the platform is engineered to maintain functionality under adverse conditions. Whether due to computational constraints (for example, GPU saturation, large model inference times), inconsistent model behaviour (for example, variable output formats, intermittent failures), or limited network bandwidth, Seshat is capable of adapting its performance without compromising data integrity. For instance, annotation modules that cannot keep up with the frame rate may skip frames while logging performance warnings.

1.4. Thesis structure

The subsequent chapter and sections of this dissertation are organised as follows. Chapter 2 explores the research and market landscape. An in-depth analysis of the market for similar technologies applied to endoscopies and colonoscopies is made in Section 2.1. Section 2.2 introduces a subsequent analysis of research and market alternatives without focusing on the integration of models on medical images. In Sections 2.3 and 2.4, the concept of annotators is introduced and explained, and the challenges in integrating them into medical images are addressed. To finish this Chapter, in Section 2.5, an analysis of the image types used in the medical images is made, as well as a study on which could or could not be supported by Seshat.

In the next Chapter 3, the design choices for Seshat are presented and discussed. Section 3.1 describes the design goals set for the development of Seshat. Moreover, Section 3.2 exposes the different user profiles and their possible use cases. Additionally, Section **??** discloses the different design components that constitute Seshat and how they interact with each other. To conclude this Chapter, Section 3.4 explains the data storage format which supports Seshat's functionality and has unique characteristics.

The succeeding Chapter 4 details the implementation of the components of Seshat and their complexities. It reveals the coding mechanisms of the components explained in the

previous Chapter, their role for functionality and interaction with the rest of the system. Section 4.1 discloses the component that coordinates Seshat. Furthermore, Section 4.2 unveils the acquisition mechanism that converts the medical exams into the data storage format, the Repository. Moreover, Section 4.3 describes the API responsible for the integration of annotators into Seshat, how it works and how users can use it. Finally, Section 4.4 reveals how Seshat's interaction with the user operates, explaining the details of the web app from the frontend to the backend procedures.

The subsequent Chapter 5 exposes Seshat's compliance with the objectives. In Section 5.1, an empirical analysis is made of the performance of Seshat under different conditions. To finalise, Section 5.2 describes the usability tests made to Seshat and explains its results.

To conclude this dissertation, Chapter 6 reviews accomplished and not accomplished objectives and Section 6.1 discusses future work to improve Seshat and its functionality.

2. Background

A wide range of technologies has been developed to support annotation on medical images. Many of these systems are designed with commercial objectives, prioritising market adoption over openness. As a result, their architectural and implementation details are often proprietary and not publicly disclosed. Moreover, research efforts in this space frequently emphasise the performance of ML models, with little attention given to the surrounding system infrastructure. This creates a gap in the literature, where the technical frameworks that enable real-time annotation and interaction remain underexplored or insufficiently documented.

The scope of the investigation was broadened beyond medical imaging alone to include annotation systems developed for general video streams. While these systems may not be tailored to the specific demands of the medical domain, they offer a broader range of design patterns, interface strategies, and integration mechanisms that can inform the development of a more flexible platform.

Earlier in the development of this thesis, the focus was more narrowly defined, centred on the use of endoscopic imaging for gastric cancer screening. A survey of existing technologies in this subdomain was conducted, with particular attention given to their graphical user interfaces, user interaction models, and user experience. However, due to the limited availability of implementation-level details, this initial review was primarily descriptive and interface-focused, prompting the need for a more expansive exploration across domains. Section 2.1 explores these technologies and, on one hand, reviews their strengths, which reflect the different aspects that should be taken into account in the development of this platform; on the other hand, their weaknesses, which should be avoided.

Several prospective randomized trials have demonstrated that AI–assisted colonoscopy significantly improves detection of colorectal neoplasia compared to standard practice [8]. Moreover, some early deep learning models demonstrated the feasibility of real-time polyp detection in colonoscopy videos, laying the foundation for many subsequent systems [9]. Beyond colonoscopic procedures , Real-time AI systems have also been applied successfully in upper gastrointestinal endoscopy, achieving high sensitivity for early gastric cancer detection in multicenter prospective studies [10–12].

2.1. Endoscopy and colonoscopy systems

Odin Vision - CADU

Overview

The CADU [13–17] system, developed by Odin Vision, is an AI technology designed to assist endoscopists in identifying dysplastic regions in the oesophagus during upper gastrointestinal (GI) procedures. It features real-time detection of areas of concern, providing immediate feedback during the endoscopy. CADU identifies points of interest in the dysplastic region for endoscopists to identify the geometric centre and highly dysplastic regions. This information can then be interpreted by the user to make decisions regarding treatments, since there is a distinction between dysplastic and non-dysplastic tissue. CADU has a simple user interface that is controlled by the user via a foot pedal. CADU's computer module is integrated with the endoscopy machine by acquiring the video capture and intercepting the camera used in the procedure. It can be configured to use the monitor on an already existing endoscopy machine or a second auxiliary monitor. The AI model is cloud-based, and the intersected video is streamed from the module to the cloud. The stream sent to the cloud is evaluated by the AI model. If the model detects an area of interest in the stream, the cloud will communicate with the module to highlight it to the user. In this case, the highlight is accompanied by a sound alarm to alert the user of the area of interest.

Strengths

- The cloud-based approach guarantees that any software update is performed without the need for maintenance costs and on-site intervention from a technical specialist.
- CADU's real-time capabilities enable immediate lesion identification.
- CADU works with various endoscopy equipment and existing clinical workflows.
- The foot pedal controls are intuitive, easy to use, and convenient, since the endoscopist's hands are occupied with the endoscopy machine controller.
- Areas of interest are immediately shown.

Weaknesses

The lack of customisation of user input controls and model segmentation highlights hinders the user experience.

Figure 2.1: Example of the UI of CADU | Source: ”https://odin-vision.com/en$_gb/cadu-2/$”

AI Medical Services - gastroAI™ Model-G

Overview

The gastroAI™ Model-G [18, 19] is an endoscopic image diagnosis support software designed specifically for the gastric region. It aids endoscopists by highlighting lesion candidates and indicating possible biopsy targets using rectangular markers. The system provides two diagnostic states—”Consider Biopsy” and ”Low Confidence”—which help endoscopists decide whether additional examination, such as a biopsy, is necessary. The model is displayed on a separate 11-24” monitor adjacent to the endoscopy device, and the system allows the endoscopist to freeze images for closer inspection. Although

separate from the endoscopy machine's monitor, this setup supports a streamlined diagnostic approach within the procedure.

Strengths

gastroAI™ Model-G's easy setup and intuitive interface facilitate quick adoption and support endoscopists in making more informed diagnostic decisions. Moreover, the dual-state system ("Consider Biopsy" and "Low Confidence") can guide endoscopists in assessing lesions with greater accuracy, offering practical reference points for more decisive in-procedure actions.

Weaknesses

- Although there are various monitor sizes, there is no possible integration with the endoscopy machine's monitor.
- The diagnosis states shown by the model can take up the necessary space on the screen.
- The system's lack of customizable input options or foot-pedal controls could hinder workflow efficiency, as endoscopists must use their hands, which are often occupied during the procedure.

Chinese University of Hong Kong (CUHK) - AI-Endo

Overview

AI-Endo [20–22], developed by the Chinese University of Hong Kong (CUHK), is an AI-based diagnostic support system tailored for gastric cancer detection in endoscopy procedures. Using an ML model, AI-Endo analyses endoscopy images in real-time and highlights areas of interest for the endoscopist. The system captures each video frame and uses colour-coded indicators to distinguish various tissues, helping to increase adenoma detection rates, particularly for endoscopists with less experience. Clinical trials report substantial improvements in detection rates, underscoring AI-Endo's potential for increasing diagnostic accuracy across user experience levels. The AI model is open-source and stored in a public GitHub repository, allowing for transparent development and potential

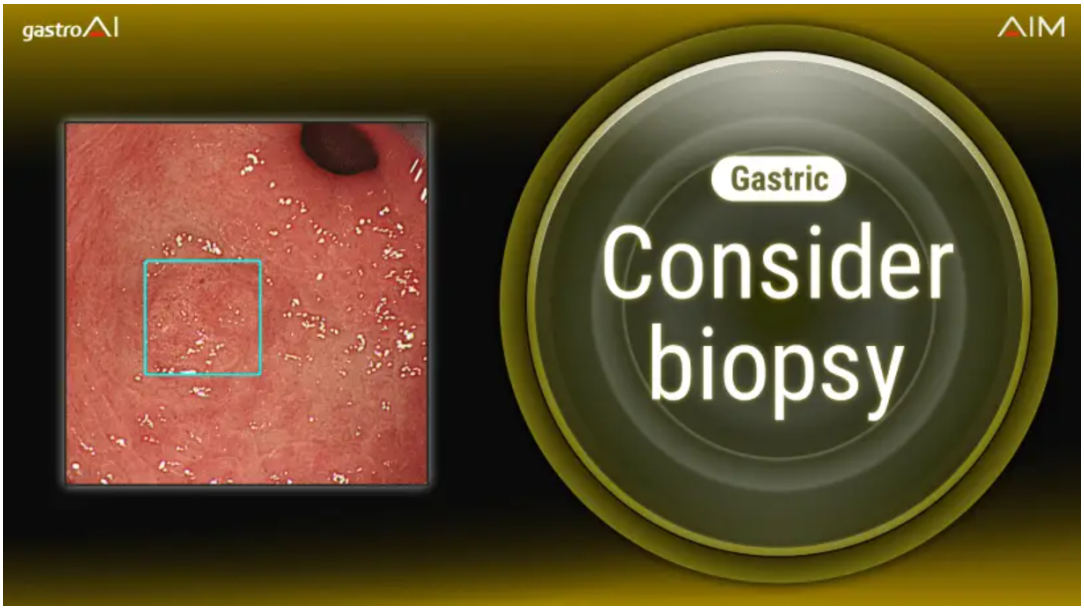


Figure 2.2: gastroAI™ Model-G's UI with the state of "Consider Biopsy" | Source:"https://endo.ai-ms.com/product/uppergi"

community contributions.

Strengths

AI-Endo supports a very intuitive highlight system that applies different colours to different detections, improving the distinction between tissues.

Weaknesses

AI-Endo primarily targets the needs of less experienced endoscopists, potentially limiting its functionality for highly skilled users or broader endoscopy applications. Moreover, the lack of a dedicated hardware module may lead to performance issues or compatibility limitations with endoscopy equipment, potentially affecting processing speed and real-time detection reliability. In contrast, physical modules, such as those in CADU, offer enhanced system stability and integration with clinical equipment.

Medtronic - GI Genius

Overview

GI Genius [23–25] is an endoscopy module powered by an AI model to help endoscopists detect and diagnose gastric cancer. The system's AI model was trained to specialise in

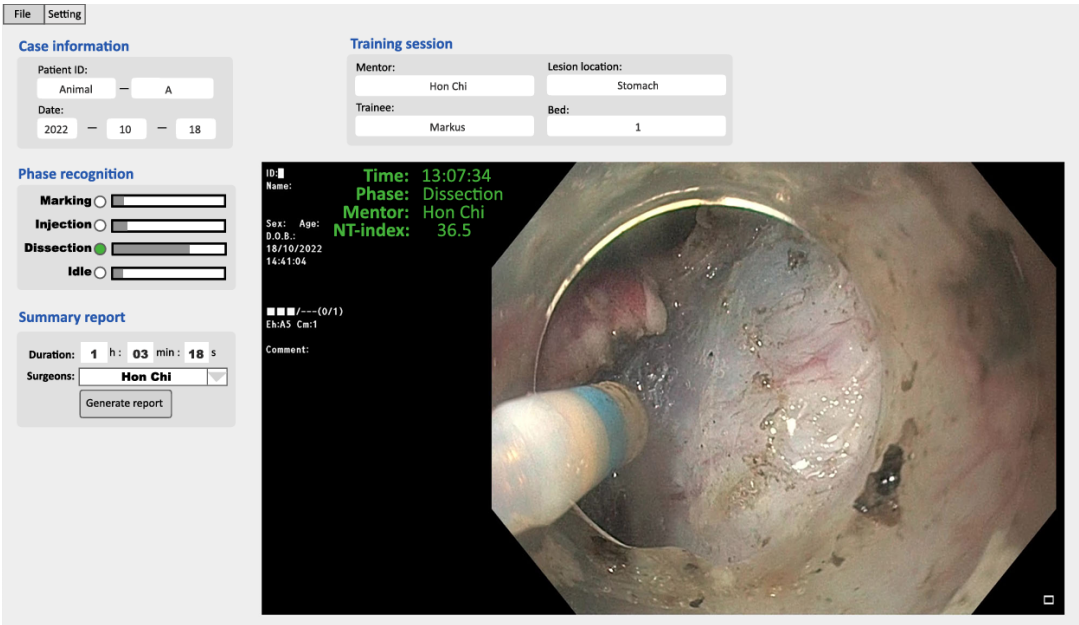


Figure 2.3: The UI of AI-Endo | Source: ”https://www.nature.com/articles/s41467-023-42451-8.pdf”

polyp detection on endoscopy procedures. This computer module can be integrated with a set of endoscopy machines and highlight the adenomas and lesions to the user. The module offers simple hardware controls embedded in its case for installation and connects to the endoscopy machine via cable. GI Genius has a very strong market presence and notorious performance results with 99.7% sensitivity, 82% faster recognition than the endoscopist, and less than 1% false positives.

Strengths

The hardware module’s integration with endoscopy machines is practical and efficient, offering easy installation and robust compatibility.

Weaknesses

The module can’t integrate with every different endoscopy machine. It might not be compatible with some brands’ equipment.

WisionAI - EndoScreener

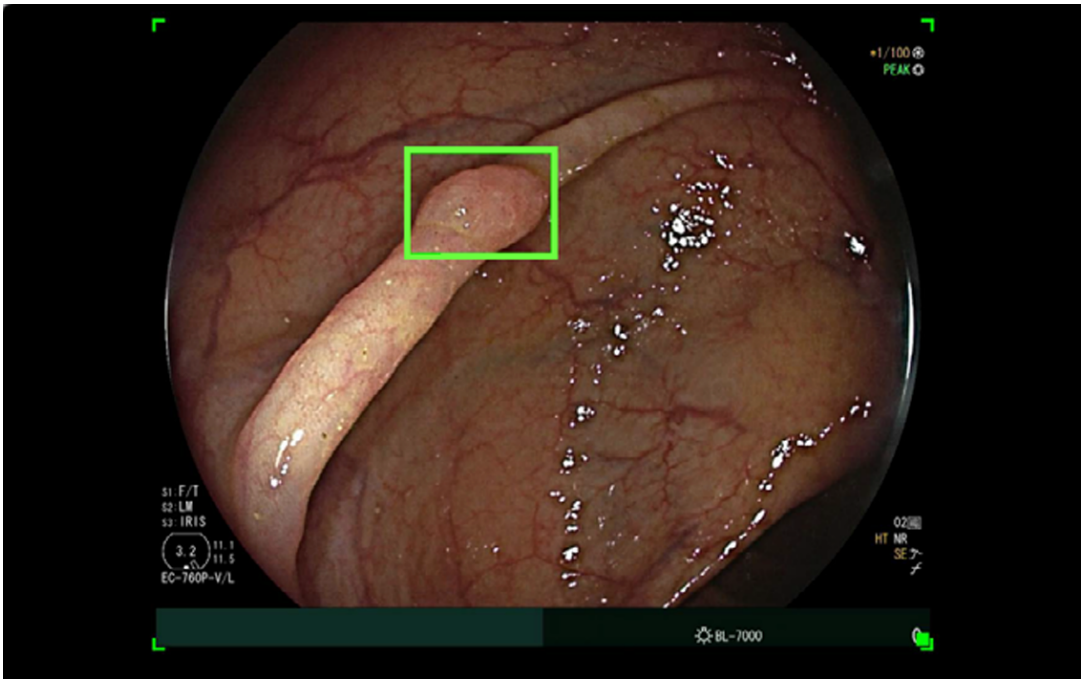


Figure 2.4: The UI of GI Genius

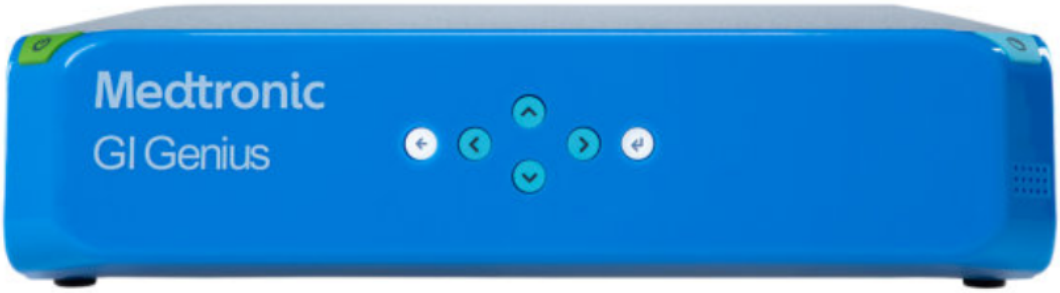


Figure 2.5: GI Genius's physical module that integrates into the endoscope | Source: "https://www.medtronic.com/en-us/healthcare-professionals/products/digestive-gastrointestinal/gastrointestinal-artificial-intelligence/gi-genius-intelligent-endoscopy-module.htmlordering-information"

Overview

EndoScreener [26] is an AI-based system that analyses live video from colonoscopies to detect polyps in real-time, assisting doctors by highlighting abnormalities that may be missed during the procedure. It takes the colonoscopy video stream and analyses it in real-time with an AI model. The model highlights potential polyp regions with a blue rectangle to improve gastric cancer diagnosis. The system is installed next to the endoscopy machine, intercepts the video capture of the camera and displays the endoscopy in real-time in a separate monitor that mirrors the endoscopy and displays the highlights. The adenoma detection rate (ADR) while using this system was increased by approximately

32% when compared to standard colonoscopies. Not many details were found regarding the AI model of this system.

Strengths

- Being a separate system that intercepts the video capture and runs separately to the colonoscopy machine provides practicality in terms of maintenance and customisation to the colonoscopist's liking.
- Having a second monitor mirroring and adding details to the colonoscopy procedure allows the colonoscopist to focus on either one or the other screen, depending on preference.
- If the system's monitor is visually polluted with highlights, the colonoscopist might prefer to use the colonoscopy's screen to better analyse the area highlighted. On the other hand, during the rest of the procedure, it may be preferable to watch the EndoScreener's monitor to view highlights.
- There are no compatibility issues since everything that is happening with the system is built entirely separately from the functioning of the colonoscopy machine.

Weaknesses

The highlight rectangle is not the most adequate visualisation tool for segmentation

EndoVigilant

Overview

EndoVigilant [27], developed by the company with the same name, integrates with standard colonoscopy equipment to provide real-time AI-assisted polyp detection during colonoscopies. The system analyses the video feed and marks potential polyps, helping reduce missed detections. It does not require significant hardware changes, as it works alongside the existing setup by processing video input directly. Details on specific user control mechanisms like foot pedals, hand, or voice activation were not found, suggesting that it operates passively during the procedure, with the focus on seamless integration into the clinical workflow. The primary outcome of the mean number of adenomas per colonoscopy was higher in the CAD than in the control group (1.35 vs 1.07, p=0.099),

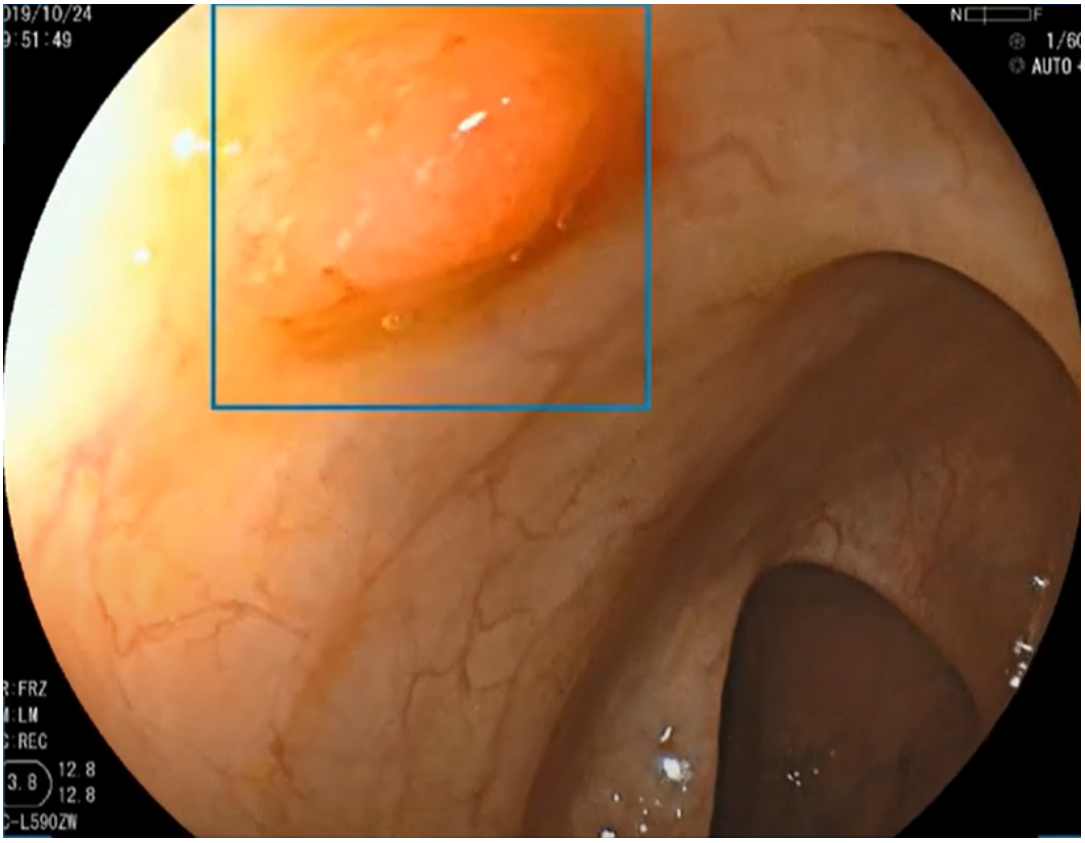


Figure 2.6: The UI of EndoScreener highlighting a polyp | Source: ”https://www.wision.com/index.html/endoScreener”

though this did not achieve statistical significance. The detection model is made with single-shot detectors that have no public performance results. The video signal from the colonoscope is fed into a computer running the EndoVigilant CAD software in addition to the standard video output to the procedure monitor. The EndoVigilant CAD software displays an annotated video on an additional monitor, which includes both the colonoscopy picture and software highlights indicating the location of any polyps on the screen.

Strengths

- Similar to the previous system, EndoVigilant is built separately and is completely independent from the colonoscope, improving the maintenance issues, visibility,4 and practicality for the user.
- Any update to the system or model can be made by simply updating EndoVigilant’s software, making this system convenient for the rapid evolution of AI models.

Weaknesses

The improvements are not notable; there was no difference in the rate of detection of neoplastic polyps with less than 10 mm.

Pentax Medical - DISCOVERY

Overview

DISCOVERY [28] uses deep learning algorithms integrated with real-time colonoscopy video feeds to assist in detecting lesions. The AI model works autonomously during procedures but can be fine-tuned or adjusted through simple user interfaces connected to the scope or processor. It's integrated with Pentax's endoscopes to enhance polyp detection without requiring separate equipment. Its physical module connects to the endoscope and can be set up to be used on the endoscope's monitor or on an auxiliary one, where it mirrors the procedure and highlights the areas of interest. The integration of the portable physical module into the endoscope is simple and is also compatible with all brands. Moreover, in studies, the system demonstrated an 11% improvement in SSL detection among endoscopy experts, raising SSL detection rates from 19% to 30%, a statistically significant enhancement. This improvement has substantial clinical implications, as each 1% increase in SSL detection correlates with a 7% reduction in interval colorectal cancer risk.

Strengths

The simplicity of the integration module is concrete enough to guarantee the functioning of the system and still be compatible with every endoscope on the market.

Weaknesses

Even though the dataset is composed of many different conditions to better train the model, the improvements in lower conditions are not notable when compared to a regular procedure. Moreover, if the system is integrated into the endoscope's monitor, the software uses a significant portion of the screen to display the model's diagnosis, worsening the visibility of the procedure.

Olympus - ENDO-AID

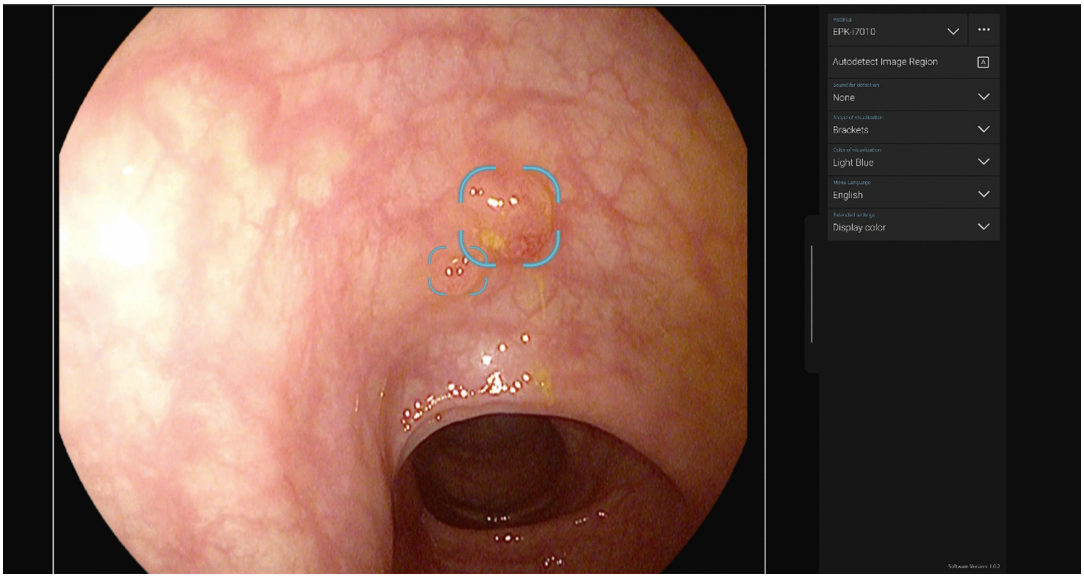


Figure 2.7: The UI of DISCOVERY while highlighting different tissues | Source: ”https://www.pentaxmedical.com/en/products/discovery-ai”

Overview

ENDO-AID [29] by Olympus is an AI-enhanced platform designed to support gastroenterologists in detecting and classifying gastrointestinal (GI) lesions during endoscopic procedures. The platform, embedded into the EVIS X1 endoscopy system, is designed for the real-time detection of cancerous lesions. ENDO-AID employs deep learning algorithms to process endoscopic images as they are captured, highlighting potential areas of concern. The user interface is built to provide a visual overlay of flagged areas without interfering with the endoscopist’s standard workflow. This design choice indicates Olympus’s goal to make the tool accessible and non-intrusive. The overall ADR was significantly higher in the CADe group than in the control group. Performance analyses show that ENDO-AID can improve ADR and reduce the miss rate for certain types of lesions, particularly small and flat adenomas. However, its real-world efficacy depends heavily on consistent operator use and the specific GI landscape of each institution, as performance may vary depending on lesion prevalence and the skill level of the endoscopy team.

Strengths

- Since the module is only compatible with endoscopes from the Olympus brand, the integration is very simple and practical.
- The user interface provides two modes to better suit the situation at any given time of a procedure, a practical feature that is user-friendly.

Weaknesses

ENDO-AID is only compatible with a single endoscope system, making it inaccessible for hospitals and institutions that do not have EVIS X1 in their facilities.

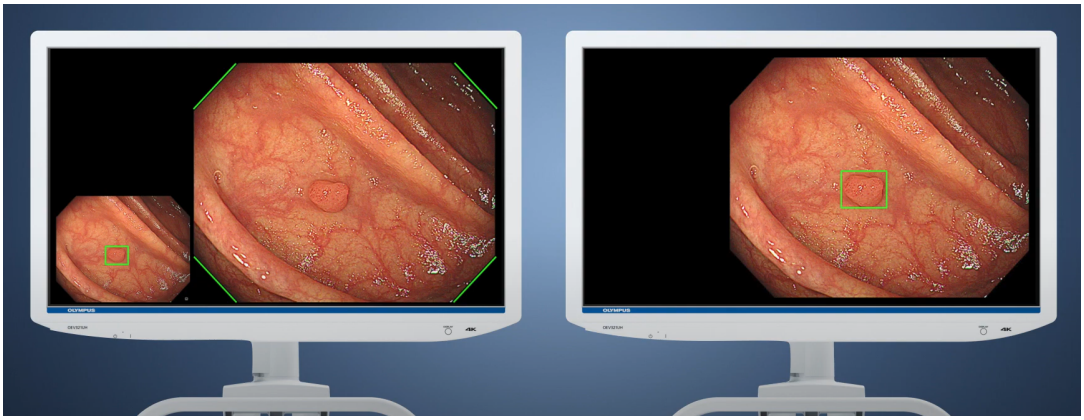


Figure 2.8: ENDO-AID's different modes side by side, Normal on the left and Target on the right, while highlighting a lesion | Source: "https://www.youtube.com/watch?v=dDZ5eaciPi0"

FujiFilm - CADEYE

Overview

CADEYE [30–32] by Fujifilm is a computer-aided detection (CAD) system integrated within the Fujifilm ELUXEO endoscopy platform, aimed at improving the detection and characterisation of colorectal lesions, especially during colonoscopies. Using deep learning algorithms, CADEYE analyses endoscopic images in real-time to identify potential lesions, highlighting them with on-screen markers to prompt endoscopist attention. This detection functionality is paired with an optical enhancement mode that enhances lesion visualisation, aiming to assist clinicians in distinguishing benign polyps from potentially pre-cancerous or cancerous lesions. CADEYE is only compatible with ELUXEO systems. CADEYE's interface is composed of the video capture, the status bar that indicates the status of characterisation analysis regarding the area of interest, the visual assist circle that has different colours for neoplastic and hyperplastic characterisation, the position map that indicates the position of the area of interest and the characterisation result (adenoma, polyp, etc.). The detection is highlighted in the form of rectangles surrounding the area of interest and a visual assist circle. The alert is given by an audio cue with customizable volume.

Strengths

CADEYE not only highlights the detected polyp, but it also provides a visual assist circle that lights in the direction of the detection.

Weaknesses

- CADEYE is only compatible with ELUXEO systems and cannot operate on others.
- The graphical interface has a large number of elements, and it's not customizable.

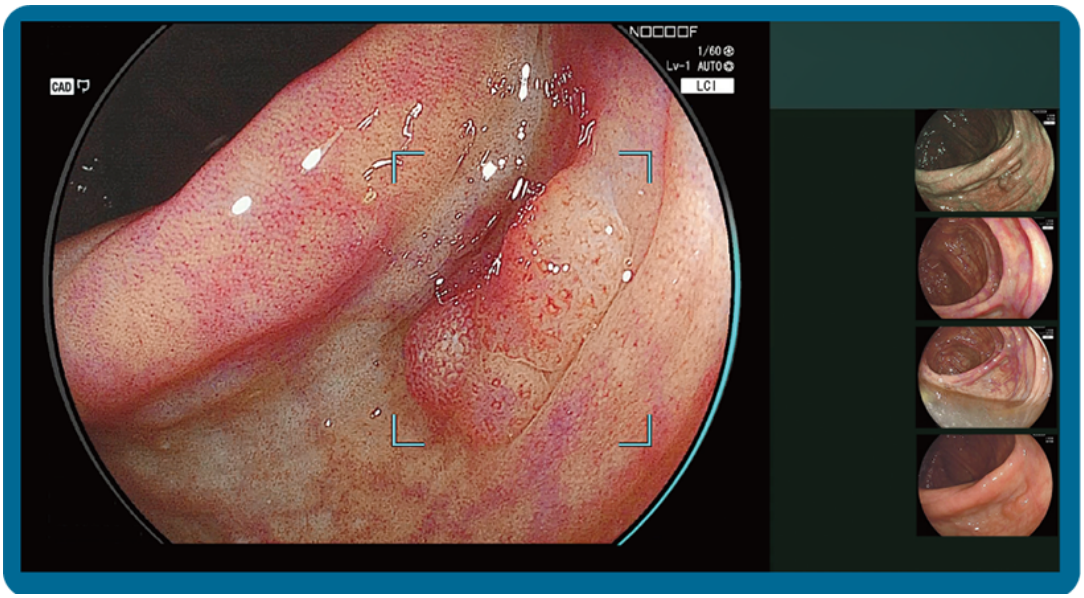


Figure 2.9: CADEYE's system highlighting a lesion | Source: "https://www.fujifilm.com/vn/en/healthcare/endoscopy/endoscopy-processors/cadeye"

Table 2.1 provides a structured comparison of existing endoscopic AI technologies, highlighting key operational and architectural distinctions. The Input Control parameter indicates whether the system permits procedural intervention by medical practitioners, including annotation requests, video playback manipulation, or other real-time interactions. Among the surveyed technologies, only the gastroAI Model-G incorporates such interactive capabilities.

The Audio designation specifies whether the system provides auditory alerts upon detecting potential pathological findings, ensuring clinician awareness during examinations. This feature varies significantly across platforms, with approximately 30% of analysed systems implementing audio notification protocols.

Regarding anatomical specificity, the Organ classification reveals a binary distribution between oesophageal (endoscopic procedures) and colonic (colonoscopic examinations)

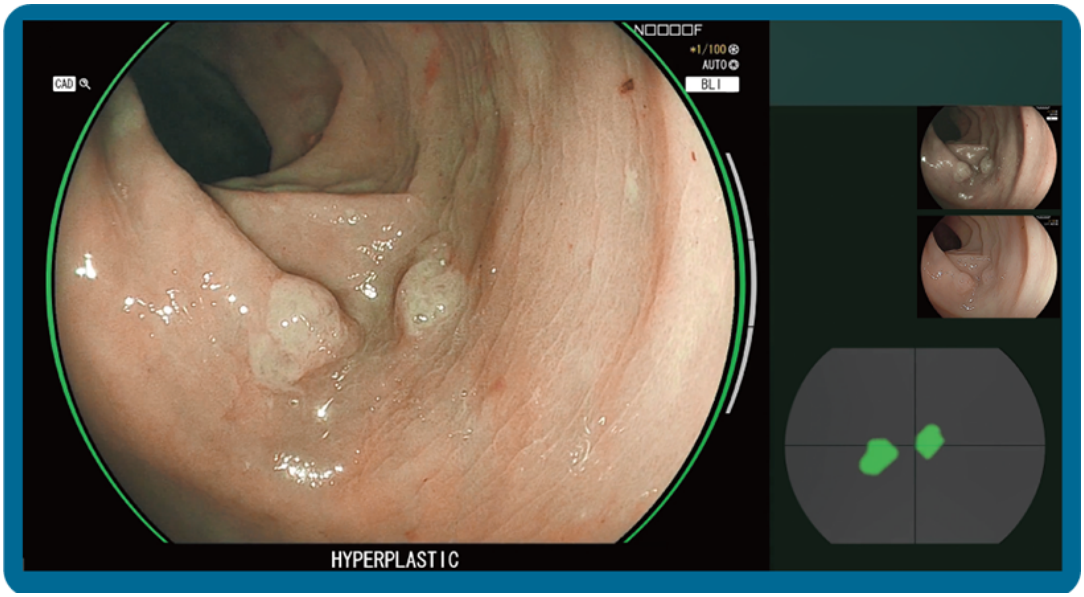


Figure 2.10: CADEYE's system classifying a lesion | Source: "https://www.fujifilm.com/vn/en/healthcare/endoscopy/endoscopy-processors/cadeye"

Table 2.1: Summary of the last technologies

Technology	Input Control	Audio	Organ	Storage
CADU	No	Yes	Oesophagus	Cloud
CADDIE	No	Yes	Colon	Cloud
gastroAI Model-G	Yes	No	Oesophagus	Local
AI-Endo	No	No	Oesophagus	Local
GI Genius	No	No	Oesophagus	Local
EndoScreener	No	No	Colon	Local
EndoVigilant	No	No	Colon	Local
DISCOVERY	No	Yes	Colon	Local
ENDO-AID	No	No	Colon	Local
CADEYE	No	Yes	Oesophagus	Local

applications. This distinction inherently determines the associated clinical workflow and instrumentation requirements.

The Storage parameter differentiates between cloud-based and local architectures. Only CADU and CADDIE employ cloud infrastructure for model hosting and data processing. This architectural choice directly influences system deployment, as locally stored models require dedicated hardware modules separate from standard endoscopic equipment, as indicated in the Physical/Software column. The predominance of physical systems reflects the clinical preference for self-contained diagnostic units in procedural environments.

Overall, the comparison reveals several patterns in the current landscape of AI-assisted

endoscopy systems. Most platforms are designed as local, hardware-dependent modules rather than cloud-based solutions, with CADU and CADDIE standing out as the only cloud-integrated approaches. Interactive input control is rare, with gastroAI Model-G being the only system allowing actions from the users, such as freezing images. Audio alerts are inconsistently implemented, present in only a minority of systems. Collectively, these findings suggest that while the field prioritises stable, self-contained local solutions, there remains room for improvement in interactivity, user feedback mechanisms, and flexible deployment platforms.

2.2. Integration of annotators on non-medical videos

To our best knowledge, no project that integrates ML models to annotate videos provides technical details regarding the interface that connects the outputs of the models to the video images and the overall data flow of the systems. This can be explained by the lack of systems whose focus is not the development of ML models but instead their integration into a real-world setting. Below, summarise existing works and identify gaps in architectural transparency. ScribllеBox [33] interactive video object segmentation system demonstrates impressive annotation refinement through user scribbles, but the technical discussion abruptly stops at the Python notebook level. The paper mentions ”real-time feedback loops” without specifying how the UI communicates with segmentation models - whether through REST endpoints, WebSockets, or embedded libraries. Similarly absent are considerations for concurrent user support or model versioning, which would be essential for real-world deployment in clinical or industrial settings.

LabelMe [34] and later works like Efficient Video Annotation focus on crowdsourcing interfaces and label aggregation, but treat the underlying infrastructure as a black box. The WebRTC community appears equally focused on transport mechanisms at the expense of ML integration. Mahmoud and Abozariba’s [35] comprehensive survey catalogues streaming protocols but never addresses how annotators might inject metadata into WebRTC data channels. This oversight is particularly striking given WebRTC’s potential for low-latency model outputs in applications like live sports analytics or robotic surgery. Deep-Framework [36] introduces an edge-oriented system for real-time video analysis. Though it mentions ”distributed scalability,” the architecture lacks specifics on fault tolerance, load balancing, or how models are dynamically deployed across edge nodes.

Scalable Camera Networks[37] discusses distributed camera management but prioritises

network-layer optimisations (e.g., bandwidth reduction) over the software stack handling model inference. WebRTC Beyond Streaming surveys WebRTC for low-latency data transfer but does not address how ML models (e.g., annotators) are embedded into these pipelines (e.g., frame preprocessing, model chaining) Learning Video Annotation explores how models can guide annotation, yet the system design (e.g., how user feedback loops are implemented) is not elaborated.

2.3. Types of annotators

Regarding medical images, an annotator refers to the process of evaluating and labelling medical images or video frames to highlight relevant findings concerning a certain medical condition/illness. Annotations can range from simple classification labels to more detailed markings, such as segmentation masks outlining specific regions of interest, to other forms of medical report information. The goal of annotation is to enhance diagnostic accuracy, improve training datasets for AI models, and facilitate better medical decision-making. Depending on the use case, annotation can be performed manually by medical professionals, automatically by AI models, or in a hybrid approach combining both.

Manual annotation is performed by health professionals who carefully analyse images and mark findings. This approach ensures high accuracy but can be time-consuming and subject to individual interpretation differences. For example, during a medical exam, the health professional can annotate a bounding box with custom dimensions and position to contain an area of interest. Although this approach guarantees a superior quality of annotations, it is not suited for real-time annotation of medical procedure exams, such as an endoscopy or colonoscopy.

Automatic annotation uses machine learning models to detect and label regions of interest in an image. These models, often trained on large datasets of expert-labelled cases that usually use manual annotation, can quickly process videos and provide preliminary annotations for review. Although their accuracy depends on the dataset and training quality, and model limitations, these are the most suitable options for real-time medical procedures and therefore the kind used in this project.

2.4. Challenge for the integration of annotators

Integrating annotation systems into real-time medical imaging workflows presents several software engineering and architectural challenges. These systems, such as those

used in endoscopy or ultrasound, demand minimal latency to ensure that clinicians can rely on live visual feedback without disruption. However, the addition of AI-based annotators introduces new bottlenecks and trade-offs that must be carefully managed. Real-time video processing pipelines consist of multiple stages that each introduce their own latency. Even with a highly optimised architecture, delays can degrade the user experience and hinder clinical workflows, especially over slower connections or in resource-constrained machines. This can lead to long delays when fetching both exam video frames and ML model annotations. Latency and responsiveness are important considerations in interactive systems; delays beyond certain thresholds can degrade user performance and disrupt workflow continuity [38, 39].

These latencies become more noticeable when playing high-resolution exam videos and when the system needs to maintain a consistent frame rate—in such cases, delayed or skipped frames cause visual stuttering or desynchronization between what the clinician sees and the actual procedure timeline, leading to potential loss of critical information.

Real-time medical imaging systems, such as endoscopic or ultrasound video feeds, require near-instantaneous processing to support live diagnosis. However, ML models introduce hard-to-counter delays due to factors like model complexity and input resolution. This creates a fundamental trade-off: High-accuracy models risk introducing unacceptable lag, while lightweight models may be fast but less reliable. Balancing between execution time and model accuracy becomes an important design choice. The model should not compromise the exam's workflow while still delivering useful annotations.

2.5. Image formats

The selection of an appropriate image format is a crucial design decision in the development of a medical imaging platform. The selection of appropriate image formats is not only a technical consideration but also a critical decision that can impact diagnostic accuracy, computational efficiency, and patient data. While numerous image formats exist, in the medical imaging domain, the most commonly used is DICOM (Digital Imaging and Communications in Medicine), with JPEG and PNG serving specific roles in certain contexts. Each format embodies different design architectures, compression mechanisms, and metadata capabilities that directly influence the preference of developers and medical practitioners.

DICOM serves as the international standard for medical imaging, encompassing both a file format and a communication protocol. It integrates pixel data with metadata, ensuring

that medical images remain associated with their complete clinical context. The DICOM file structure consists of a header containing metadata tags followed by the actual image dataset, organised according to a well-defined information object model that supports various medical imaging modalities and procedures. They can store an extensive array of patient data elements, numbering in the thousands of possible data fields. These include fundamental patient demographics such as name, identification number, date of birth, and sex, along with detailed acquisition parameters specific to each imaging modality or procedure. The format preserves procedure equipment information, including manufacturer, model, and institution details, alongside comprehensive study context such as referring physician, study description, and series information. This metadata environment certifies that images maintain their diagnostic context throughout the medical workflow, greatly supporting diagnosis.

The comprehensive nature of DICOM files results in substantial storage requirements that can challenge computational efficiency. More advanced studies, like whole-slide imaging in pathology, can produce single files exceeding several gigabytes. These substantial file sizes impact network transmission times, storage infrastructure demands, and processing capabilities, particularly in real-time applications where rapid image access and manipulation are required.

JPEG (Joint Photographic Experts Group) utilises a lossy compression algorithm that significantly reduces file size by eliminating visual information considered less critical to human perception. The format makes use of discrete cosine transform (DCT) compression technology, which allows for adjustable compression ratios, though at the potential cost of introducing visible artefacts, particularly around sharp edges and detailed textures. Its design focuses exclusively on visual data representation without consideration for clinical context or patient information requirements. Similarly, JPEG does not encounter the substantial file size challenges characteristic of DICOM formats, as its compression methodology specifically targets a minimal storage footprint.

Similarly, PNG utilises a lossless compression algorithm that preserves all original image data without quality degradation, employing a deterministic compression approach that ensures perfect reconstruction of the original image. The format supports transparency channels and higher colour depths than basic JPEG. The format's design does not accommodate the complex structured metadata requirements of medical imaging applications. Regarding file size considerations, PNG generates larger files than JPEG due to its lossless compression approach, but these sizes remain substantially smaller than typical

DICOM files.

The study of these image formats was crucial in supporting the decision on which image formats would be supported by Seshat. Although DICOM is a standard in medical images, it is not supported by Seshat for the following reasons. The trade-off between the notable gain in medical information and the loss of computational efficiency would compromise the performance objectives of the platform. As the overhead for DICOM images would be considerably larger than JPEG or PNG, there would be a significant delay in the image fetching operations, which would render the performance lacklustre.

Additionally, with the patient metadata present, composed of confidential data, another concern when handling these image formats would be the security of Seshat. If DICOM were to be supported by Seshat, the platform would have to be implemented with a robust security layer to prevent the confidential patient information from being accessible by authorised personnel, which would be out of the scope of this project. Since this layer was not an option, the patient's data would be vulnerable; therefore, the DICOM format was discarded.

3. Design

This section describes the software design of Seshat, a platform for seamless integration of multiple annotators within video streams. It shows the connections between medical imaging systems producing video, AI/ML models annotating those frames, and user interfaces enabling practitioners to visualise and interact with the enriched content. Seshat's design was guided by the following goals.

3.1. Design Goals

Integration of external annotators A fundamental architectural principle governs the platform's interaction with external annotators through a rigorously defined service interface. This design intentionally decouples analysis implementations from core processing logic, enabling seamless integration of diverse computational models regardless of their development environment or architectural paradigm. Versioned API contracts ensure backwards compatibility, allowing researchers to upgrade their annotators' capabilities without disrupting existing workflows.

Multi use case support Designed to serve both clinical and research workflows, the platform implements distinct interaction paradigms tailored to each user role. For medical practitioners, the system provides real-time annotation capabilities during live procedures, with specialised interfaces for collaborative case review. For research users, the platform offers advanced dataset curation tools, including version-controlled annotation sets and blinded evaluation modes. A unified knowledge management layer connects these workflows, allowing de-identified clinical annotations to feed model improvement cycles while maintaining strict patient privacy controls.

Condition adaptive platform The deployed platform will face a diverse range of operational scenarios. Video visualisation may occur on devices with varying network connectivity to the server. Concurrently, annotation efficiency will fluctuate considerably, ranging from rapid, frame-by-frame input to more sporadic contributions. Consequently, the platform must dynamically adapt to these evolving conditions, such as low bandwidth. If the network issues compromise the visualisation, then Seshat should not freeze the video feed, but instead either lower the frame rate or the video ”quality”.

Medical deployment Seshat must be ready to be deployed in a specially assigned room at the Porto IPO to allow medical practitioners to use and experience it. The room has an endoscopy machine and space for another machine to run Seshat on and operate on live endoscopic images via the endoscope video feed. It must have functionality, reliability and readiness to assume a role in future cases regarding integration on the medical scene. This design ensures that, as the platform evolves toward formal medical integration, its core architecture already aligns with clinical requirements, reducing future adaptation costs and accelerating potential certification pathways. The IPO deployment will serve as a critical validation step, proving Seshat's capacity to operate alongside diagnostic hardware while providing actionable insights during procedures.

Frame rate In medical images such as endoscopies, colonoscopies and laparoscopies, the videos are commonly recorded at 25 frames per second. This frame rate ensures clinically smooth visualisation and is aligned with widely used datasets. It is therefore the aim of Seshat to not only acquire the video feed at 25 fps but also deliver it to the user at such a rate, and in a stable and reliable way.

Human annotators Seshat should allow users to annotate frames of exams by contributing their own analysis. These contributions from human annotators would serve as valuable ground truth data for medical datasets and further intensify the role of Seshat in the development of ML models.

Adequate User Interface Seshat's interaction with the user should be intuitive, efficient and unobtrusive. Therefore, the user interface (UI) should fit the clinical environment, allowing for maximised efficiency within the medical workflow. In particular, it should minimise cognitive load, avoid unnecessary distractions, and ensure that critical information is presented clearly and at the right moment.

In this chapter, the different design choices are anchored by sections containing a diagram and a detailed explanation for each. Section 3.2 explores the different use cases that Seshat provides its users, along with an analysis of the intervening actors. Moreover, in Section 3.3, the main software components of Seshat are detailed and explained. These are the parts of the software that handle distinct functions, such as video playing or video acquisition, and cooperate with each other by exchanging data. Additionally, these interactions are explored later in this Section to shine a light on how these provide the needed Seshat functionalities.

3.2. Use cases

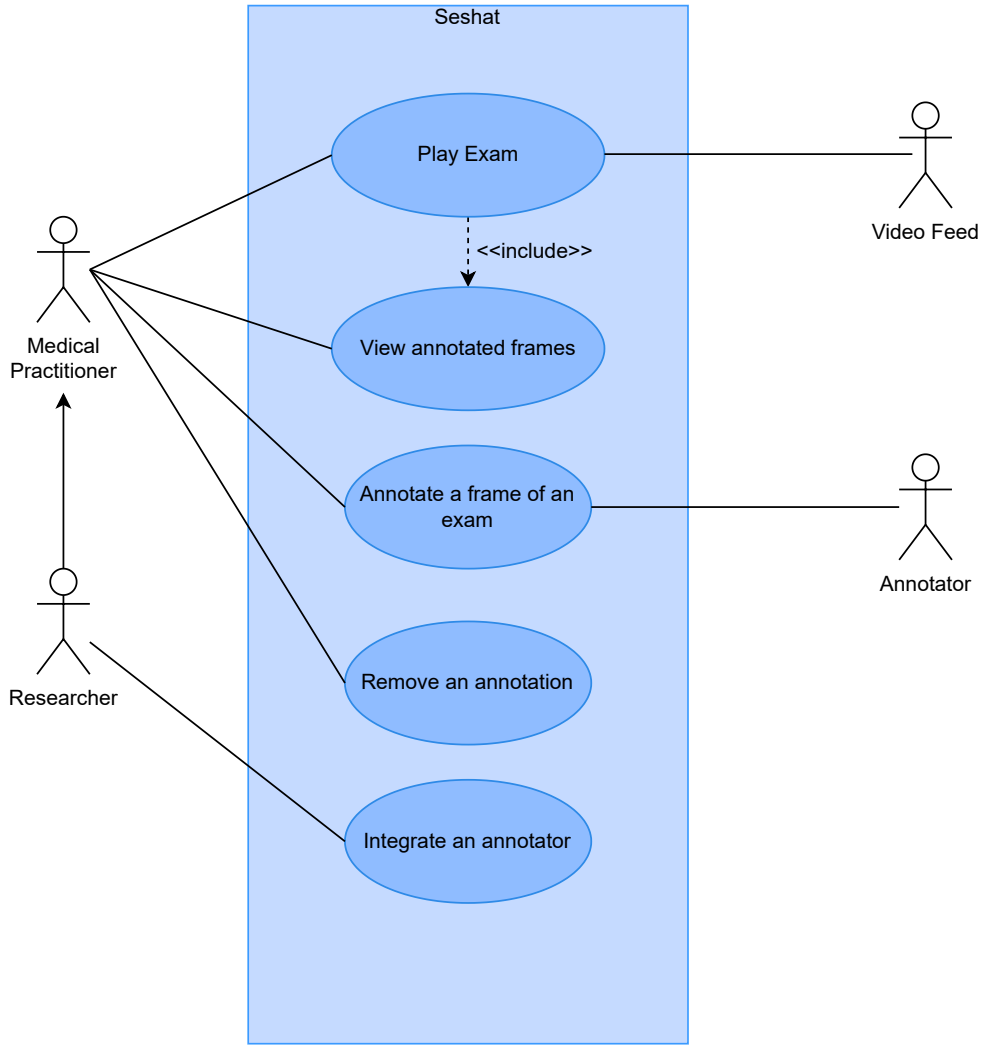


Figure 3.1: Use cases of Seshat.

3.1. This section details Seshat's use cases based on the UML diagram in Figure 3.1, depicting the relevant use cases, actors and relationships among them. On the left side of Seshat, the human actors are represented: Medical practitioners, the professionals responsible for using Seshat to annotate on medical images and medical reports; Researchers, the developers of the annotators responsible for their functioning and integration. Although they integrate annotators, they can also use Seshat for any other function that the medical practitioners do. No hierarchical relation is imposed between these two distinct actors; instead, there is a generalisation where the researchers specialise the medical practitioners in the sense that their usage of Seshat is very similar, with the exception that Researchers are additionally capable of integrating annotators.

The system allows medical practitioners to *play an exam*, with an interface for reviewing diagnostic exams through a video playback engine. The platform enables frame-accurate temporal navigation, supporting both manual seeking via an interactive timeline scrubber and precise controlled jumps. A core capability of the system is its on-demand frame annotation pipeline. When practitioners identify areas of clinical interest, they can trigger immediate analysis from an available annotator of their choosing to *annotate a frame of an exam*. The selected model will process the frame and save the outcome to be displayed on the exam player. The platform implements intelligent annotation visualisation, automatically overlaying analysis results on the source video frames. The rendering engine adapts to different annotation types, properly displaying segmentation masks as well as textual and numerical labels. A configurable temporal persistence system controls how long annotations remain visible, with specialised handling for sequential analyses from the same model to ensure optimal visibility of clinically relevant information. Seshat also allows *annotation management*. Users can navigate between frames with saved annotations through visual timeline markers to view them, remove them, or request newer ones.

3.3. Components

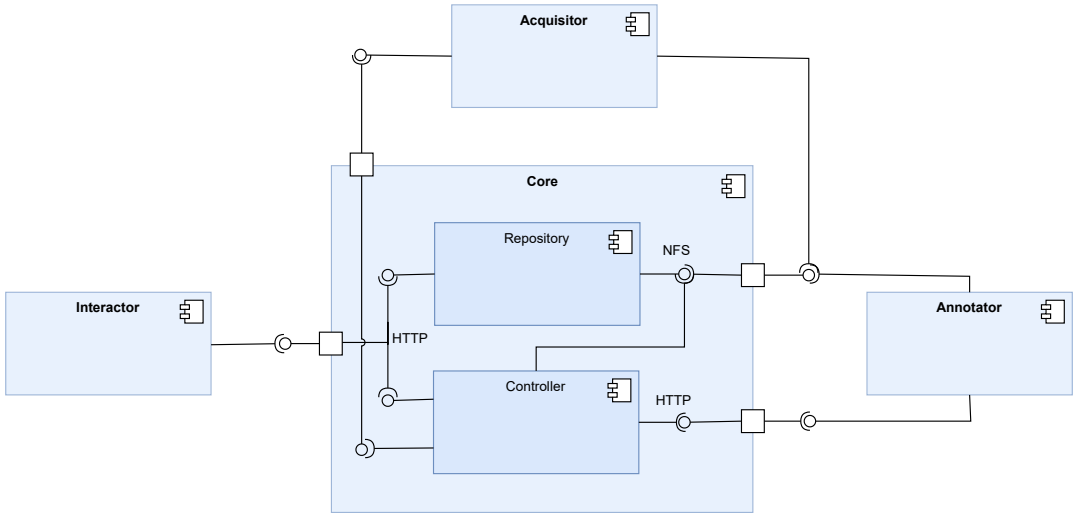


Figure 3.2: Seshat's UML component diagram.

Figure 3.2 illustrates how the different components of Seshat interact with each other. All components are represented with a plug module, which indicates that Seshat is highly modularised, and these can be replaced by other modules that develop the same functions. This enables that in the future, if any of the components require updating or a

functionality extension, developers are able to build the new version of a component without interfering with the remaining and to integrate it into the platform seamlessly.

The Core, being the central unit, is connected to every other component of Seshat. The Repository connections to other components are done either via regular file system paths or via NFS (Network File System) in case it is stored on a different machine from them. These two types of connections do not interfere with its interactions. This component is responsible for the storage of the medical images, their attributes and their annotations. It is connected to the Annotator for it to read the frames being annotated and store its annotations. The Acquisitor also establishes a connection to the Repository to store the medical images it converts from the video feeds. The Repository's connection to the Controller is essential for the latter to monitor changes made to the former by either the Annotator or the Acquisitor. The latter is responsible for controlling most data flow and communications inside Seshat. This crucial component is therefore connected to the Acquisitor to manage the acquisition of the medical images. Additionally, for the Controller to manage the workflow of the Annotator, these establish an HTTP connection.

The Interation connects to the Core via HTTP, to both Controller and Repository, to perform its main task of interacting with the user to enable them to visualise the medical images stored and request annotations from the Annotators.

Although Annotators are represented in the figure by one component icon, Seshat supports multiple annotators simultaneously, which interact with Seshat in an equal manner.

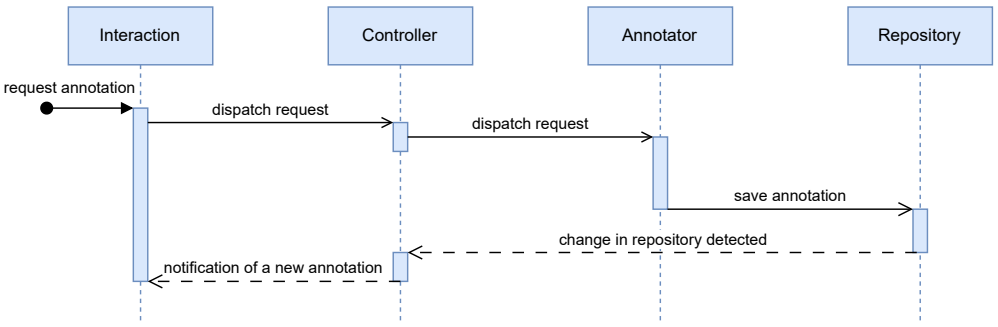


Figure 3.3: Sequence diagram of user request for an annotation.

Figure 3.3 illustrates the request for an annotation. The annotation request process follows an orchestrated sequence to ensure responsive user interaction while maintaining data integrity. When the user requests an annotation to a frame, the interaction receives the request and immediately transmits a structured annotation petition to the Controller. In this request, not only the frame index but also the desired annotator is captured to

suit the synchronisation of the expectations of the user with the result of the annotation. This operation is made asynchronously due to the fact that the Interaction can perform other operations while the desired annotation is processed and saved. When receiving a request for an annotation, the Controller forwards it to the corresponding available annotator, who performs the annotation and saves it to the repository following its specified format. After this, a notification is sent to the Interaction that the annotation is made and ready to be fetched so it can be displayed. This further illustrates the importance of the Controller and the central role it plays in Seshat.

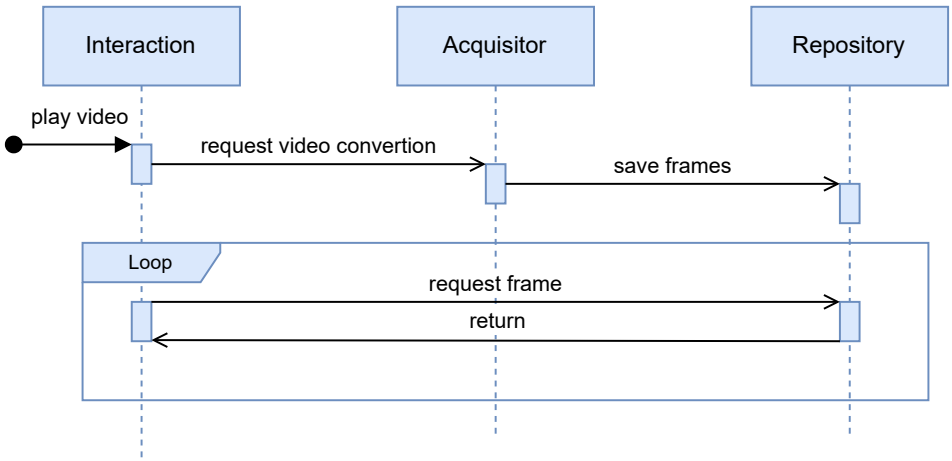


Figure 3.4: Sequence diagram of video acquisition.

The system's live video acquisition pipeline activates when a practitioner initiates a real-time exam through a connected imaging device. As depicted in Figure 3.4, this process combines immediate visual feedback with data management to support clinical decision-making. The workflow begins when the user activates the camera interface through the Interaction. This action triggers a cascading sequence where the Interaction transmits a formatted initialisation command to the Controller. The Controller then establishes a dedicated processing pipeline that coordinates directly with the camera hardware. The camera feed is converted to a sequence of frames where each processed frame enters a conversion pipeline that transforms the native camera output into the standardised repository format while preserving all diagnostic quality attributes. Concurrently, the system maintains a tight display loop where the Interaction continuously requests the most recent available frames. This loop operates under strict timing constraints to maintain fluid visualisation, employing adaptive buffering strategies that balance latency and stability. Termination of the acquisition occurs through either explicit user command or stoppage

of the Acquisitor. When the practitioner signals session completion through the Interaction or when the Acquisitor shuts down or the camera device is disconnected, the system initiates an orderly shutdown sequence orchestrated by the Controller while ensuring no frames are lost during the transition. The live view interface provides clinicians with immediate visual feedback while the system handles the complex background operations transparently, allowing for a functional clinical workflow. This triggered a choice of developing the leaving requests from the Interaction to be asynchronous, to allow for its multitasking and responsiveness. This method of video playing is referred to as a Live video/exam. Non-live videos, on the other hand, refer to the videos that have previously been recorded and are complete in the repository file format. These have a similar acquisition pipeline when being played in the Interaction. They differ at the beginning of the pipeline, in which the Interaction requests the conversion from the camera feed into the repository. This part is skipped, and instead, when the user plays a Non-Live video, the Interaction enters the loop block directly, where it, until the end of the video, requests the frames in an orderly manner, to assure the frames are displayed in the right sequence, and in an adaptive way to account for users with worse bandwidth connections.

3.4. Repository

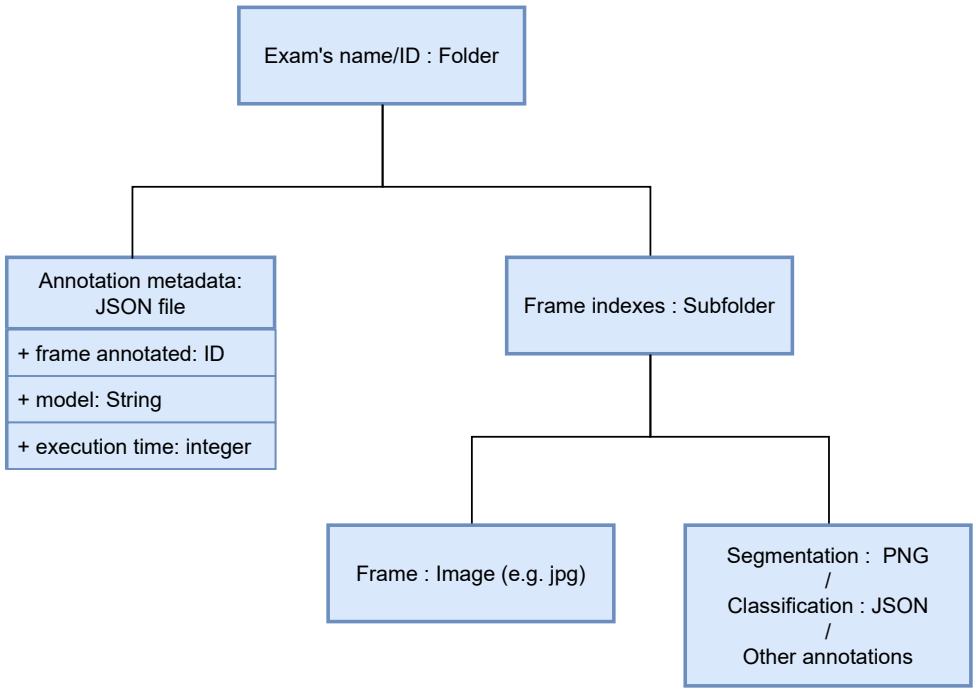


Figure 3.5: Seshat's Repository Format.

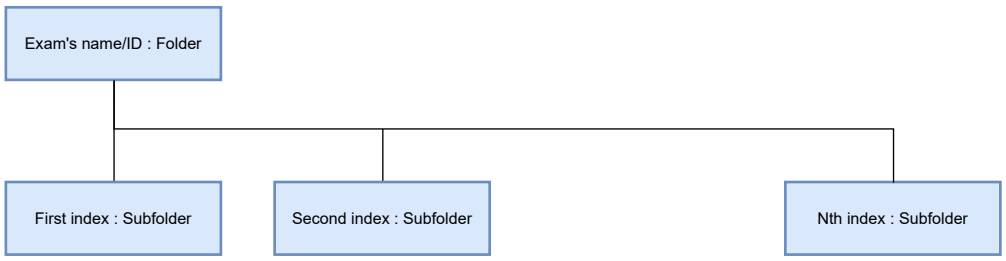


Figure 3.6: Seshat's Repository Format Continued.

The Repository is where the exams are stored and accessed. Although it is where exams are stored, how they are stored follows a specific format represented in Figure 3.5 and Figure 3.6. The system employs a hierarchical directory structure that maintains strict temporal correspondence with the original video content. Every exam is assigned to its cognominal folder inside the Repository. Inside the folder of an exam will be a subfolder that represents each frame that the exam has. Every subfolder is indexed, named after the order in which the frame it represents appears on the video. The index of each frame is then multiplied by 40, due to the display target of 25 frames per second (FPS). Since there are 25 frames in a second then there will be one frame every 40 milliseconds, resulting from the division of 1000 milliseconds by 25 frames, therefore the representation of frames is multiplied by 40 to represent the time increments from one second to another in a video with 25fps. The names of the subfolders are also padded to 9 digits. With these designs, the first, second, and third frames would be represented by the subfolders named 000000000, 000000040, and 000000080, respectively. This deterministic naming convention enables:

- Constant-time frame access regardless of temporal position
- Efficient range queries for temporal segments
- Direct mathematical conversion between directory names and video timestamps

Each subfolder contains the frame image it represents, along with a set of annotations. To annotate a frame, the annotator must access its subfolder, process the image, and save the annotation. The extension of the annotations depends on the type of labels; segmentations are normally PNG files to allow for a visible mask surrounding the area of interest, along with a transparent background to overlay the frame so that both image and annotation are visible in the exam player simultaneously, while classification and detection labels are often JSON or UTF-8 encoded TXT files.

Additionally, every exam has a metadata JSON file containing data about the annotations made to its frames. This file acts as a bookkeeper that represents the list of all the annotations in the following format:

- The frame annotated
- The name of the responsible annotator
- The time it took for the annotator to complete the processing of the frame

Figure 3.5 illustrates the file types and their extensions in the repository; it presents the annotation metadata structure with its typed components and the structure of the subfolders corresponding to the frames. On the other hand, Figure 3.6 represents the organisation of the frame subfolders in the exam folder, where each assigned frame has its subfolder named after its index in the exam video's timeline. This design ensures that all data related to a specific frame is self-contained within its corresponding folder. It allows for easy integration of new annotators without changing the structure and quick access and review of frame-level data.

This format was chosen because it is a simple and easy-to-understand approach. Moreover, the division and indexation of the frames allows for simplified access to the frames and a more structured pathing for read and write operations, to either display the frames or read them to save an annotation to their subfolder, O(1) time complexity for frame access through direct path calculation, human-readable directory structure enables direct inspection without specialized tools and simplified debugging through standard filesystem utilities. It makes the trade-off of scaling problems, but these constraints are justified by the fact that most medical exams are not extremely long in duration and therefore do not

4. Implementation

This chapter presents the implementation details of the different components of Seshat, explaining the code and the software development decisions taken when developing it. Each component of the system is examined in terms of its functionality, integration role, and contribution to the overall architecture. Particular emphasis is placed on how individual modules interact, how data flows through the platform, and how specific technical choices, as libraries, protocols, and file formats that support the platform's objectives of modularity, extensibility, and real-time performance.

4.1. Core

The Core component is the central piece of Seshat, as illustrated in Figure 3.2, composed of the Repository and the Controller. The latter represents the core orchestration component that governs most aspects of the medical video analysis platform. It implements an event-driven architecture that coordinates all components through interfaces and protocols. Implemented in Python [40], it utilises the asynchronous characteristics of Flask [41] and the assistance of other libraries. At its foundation, the Controller supports an event monitoring engine that maintains real-time awareness of any system activity happening inside the Repository with the use of Watchdog [42, 43], a well-known Python library that allows the monitoring of file system events in a given directory. The choice of this technology was made based on various factors. Since it is a very popular library, the community support is vast, which eases the error-handling process with readily available solutions. Moreover, the development and deployment are very simple and do not require large amounts of time.

This control over every aspect of changes is essential because when the user wants to annotate a certain frame, Seshat will display it as soon as it is available, by having the controller alert the Interaction of the new annotation available the moment it detects the change made to the repository.

The controller scans changes to the Repository and employs filtering mechanisms to distinguish between annotation events and routine filesystem operations (e.g. access to an image), ensuring only the former propagates through the notification pipeline. This structure communicates the filtered notifications through a message queue. The finality of this

is to send every received message through the queue to the Interaction via a Server-Sent-Event (SSE), which will alert it of all the annotations made to the exam that is accessed by the user.

Flask is a web framework that unlocks the Controller's functionalities through a straightforward endpoint routing system. It allows the assignment of specific URL paths to their corresponding Python functions that handle each endpoint on their own. The endpoint functions are flexible to handle different HTTP request types made to the same endpoint. The same URL path can process GET, POST, or other request types through conditional logic within the function, allowing each endpoint to serve multiple purposes based on the request's characteristics. This allows functions to return different data formats according to the needs of the request and its type. The choice for this tool falls mostly on its popularity, being one of the most used web frameworks of Python, its simplicity of implementation and the familiarity with the framework. Additionally, due to its popularity, it possesses a strong community support for error handling, which aids development.

To monitor changes within the exam folders, the Controller uses the Watchdog Python library, a file system watcher that actively listens for modifications in the Repository. Watchdog asynchronously filters the operating system file events that occur within a given folder. Events can be the creation, modification, and deletion of a file or folder belonging to the target folder. This allows it to be immediately notified when new frames or annotations are added, whether by a live video feed or an asynchronous model processing step. This event-driven approach guarantees that Seshat remains responsive and ready for real-time applications.

The Handler class processes these events, ensuring only relevant annotation-related changes trigger further actions. Initially, the handler discards directory modifications and focuses exclusively on file changes, specifically targeting frame images (.jpg, .jpeg) and annotation metadata (annotations.json). Since the file path for the events is given in an absolute path format, a filter is applied to the last 4 elements of the path. This is due to the fact that the annotations will be the fourth element in the repository directory, with the indexed frame number being the third and the exam being the second. This guarantees that the event analysed is always an annotation (example of a file path: repository/exam_X/000000000/annotation_example.json). Lastly, events in which an annotation is newly created or modified, which means that the same model reannotated the same frame, are placed. This filter excludes the deletion of annotations, which does not need to

be notified to the client. After these filters, an analysis is made of the file path where the basic information about the annotation is stored in a message. This includes the exam's ID, the frame of the corresponding annotation and the type of event.
The Controller implements a Flask route through the ”/api/stream” endpoint, which maintains a persistent HTTP connection that delivers the messages created by the Handler to clients. This endpoint utilises the Server-Sent Events protocol, an efficient web standard for unidirectional server-to-client communication. It enters a suspended state via a ”threading.Event wait()” call, which blocks execution until a filesystem event triggers the directory_changed flag, which indicates the arrival of a message. The message formatting adheres strictly to the SSE specification, with each event being transmitted as a JSON-encoded string. The message formatting adheres strictly to the SSE specification, with each event being transmitted as a JSON-encoded string. The endpoint maintains indefinite persistence by executing an infinite loop.
Furthermore, the endpoint ”/ api/listExams” was implemented to enumerate available exams. This GET request handler implements a filesystem-based discovery mechanism that dynamically inventories exam datasets while calculating their temporal durations in frames. The implementation firstly scans the repository directory, through the ”is_dir()” check, ensuring only exam videos in the adequate repository format are processed while ignoring loose files and other components of the folder. Furthermore, using os.walk(), the function performs a single-level traversal of each exam directory. The frame count derives from summing the subdirectories, typically frame containers and along with the exam's ID, it is attached to the list of all the exams available and sent as a JSON list.
To allow for the annotations, both on demand and in real/time, the Controller needs to bridge the communication between the client and the annotator. To do that, it employs three different endpoints that will receive requests from the client, be processed and then notify the target annotators with requests to personalised endpoints:

- On-demand annotation: the route ”api/annotation/” receives a request with the exam to be annotated and the corresponding frame, as well as the name of the annotator the user wishes to annotate with. This function implements a time system to extract the processing time of the annotator by registering the start time of the operation, indicated by the arrival of the request. Furthermore, depending on the type of annotation the annotator performs, the function communicates differently; if the annotator produces images (e.g. a segmentation mask) the function will send a request for the it to annotate based on the current loaded image details (RGB and opacity of

the produced image), and the same parameters sent with the client's request; On the other hand, if the annotator produces textual annotations then the function send a request with the same parameters sent by the client. To check information on the annotators, a list of available models with their information regarding essential details for communication is consulted. Finally, after the annotator processes the frame, a response is sent, and the function registers the current time and returns the time difference from the request to the client.

- Starting real-time annotation: Starting and stopping real-time annotations are implemented in two distinct endpoints. The starting operation, housed in the "api/startRealTimeAnnotation/" endpoint, handles requests that send the exam ID, the interval of frames to annotate, represented by the first and last indexes of the limiting frames, and the target annotator. The function sets the real-time annotation flag to indicate the beginning of the annotation and sets a timer for this operation through shared/global variables. Lastly, a request for the target model is sent by checking the annotators' list with the given annotator name on the request form, and, similarly to the previous operation, depending on the type of annotation, different requests are sent, to include (or not) information regarding the colour and opacity information for the annotator to follow.

- Stopping real-time annotation: To stop a model from performing real-time annotation (whether the user does not desire to annotate any further before the end of the given interval is played, wants to start a new real-time annotation with a different annotator, etc.) the endpoint "api/stopRealTimeAnnotation/" receives a request with the target annotator. The flag for the stoppage is adjusted to false to indicate the end of the annotation. A request is made to the target annotator to prompt the stoppage of the real/time annotation.

To load the colours, the client sends a request to the "api/loadColors" endpoint with the colour attributes in RGB format, along with the opacity value. The function stores these values and returns a success message.

Annotations are registered in the client and kept in a list to be stored in the JSON file located in the repository. The client keeps the annotations on and shares the list with the Controller to then be stored in the repository. For this purpose, the Controller employs two endpoints to save and load the annotations list. The former receives a request with the list as the payload, as a JSON and an exam ID. This list contains all the annotations registered for the target exam by the client, and upon receiving it, the function will save

the exact content of the sent list to the annotation.json file of the target exam in the repository. The latter will receive a request to load the annotation list from the target exam in the repository. It accesses the exam's directory and retrieves the annotations.json file. The contents are then returned in a JSON format. If the file does not exist yet (because no annotations were made), the return is an empty JSON object.

As annotators are integrated and removed from Seshat, the management of the currently available annotators is vital. Therefore, the Controller employs a single endpoint that handles different requests. On one hand, the endpoint ”modelRegistration” handles two types of requests. The POST requests, sent by the annotator, which, upon reaching availability to annotate, register themselves in Seshat. This POST request contains the basic information about the annotator, such as the ID, which will be used for reference, the endpoint and address it is on and the type of file it produces for the annotations. The function then inserts the newly registered annotator into the list along with its info and returns a registration success message to the annotator along with the registration data. On the other hand, incoming DELETE requests are the opposite of the previous, in which case the annotator sends this request to unregister itself from Seshat. The function removes the annotator from the list and returns an unregister success message with useful data. For the client to access the list of available annotators, the endpoint ”/api/getModels” handles its requests and returns the list exactly as is.

4.2. Acquisition

Seshat uses FFmpeg [44], a multimedia framework for video acquisition and processing. This component handles both live streams and pre-recorded video files, extracting individual frames and transforming them into Seshat's standardised repository format. The system implements a two-phase operation consisting of frame extraction followed by organisation into the Repository format.

When the user requests the beginning of a live video exam, the acquisition module is called to start decoding the exam video feed. Through FFmpeg commands, the module starts converting the video feed into individual indexed frames by first creating an output directory homonymous to the exam's ID to store all extracted data. The FFmpeg command used is summarised in Table 4.1. The function runs FFmpeg as a subprocess rather than directly calling it, and it maintains a global reference to it, because this allows the Python code to maintain control over the extraction process and terminate it if the user

wishes to stop the live exam.

Argument	Function	Purpose in Seshat
`-i <source>`	Specifies the input source	Defines the video feed to be processed, whether from a live device or a pre-recorded file
`-vf fps=25`	Applies a video filter to set the frame rate	Decodes video at 25 frames per second, normalising temporal sampling regardless of source framerate
`-frame_pts 1`	Encodes frame number based on presentation timestamps	Deconstructs video into sequentially indexed frames matching the Repository's timestamp-based convention
`output_%09d.jpg`	Defines output filename pattern	Saves frames to exam folder with zero-padded sequential names (e.g., `frame_000000001.jpg`)

Table 4.1: FFmpeg Command Arguments for Video Acquisition

After frame extraction completes, the code processes each image file to reorganise them into index-based subfolders. It multiplies the frame number by 40 milliseconds (the reciprocal of the 25fps rate; Frames have a 40 millisecond time window in a second) to convert sequence numbers into indexes. The nomenclature of the indexes passes through zero-padding to 9 digits, so all timestamps have equal string length, which maintains proper sorting order in filesystem operations. For each frame, the system creates a dedicated subfolder named after its calculated index, moves the frame into it and renames it to ”frame.[desired image format]” because having a consistent filename simplifies later frame access and rendering in the web interface.

To stop the decoding of the FFmpeg command, a function terminates the FFmpeg subprocess by sending a termination signal, stopping the deconstruction of frames and proceeding to their organisation into the Repository's format. The global ffmpeg_process variable maintains access to the subprocess so that Python can control any externally running process.

Besides acquiring live exams, the Acquisition module also allows the conversion of already existing exams. Equally to the previous implementation, it follows a two-phase operation with the first phase being the conversion of the exam video into frames and the second being their organisation. From the start to the end of the video, the FFmpeg command follows the same rules but changes the input source to the exam's relative path instead of the video feed device. Instead of a subprocess to initiate the conversion, this

implementation starts converting all the frames of the video, and when it reaches the end, it begins the process of organising the frames into the Repository's format.

4.3. Annotator

To enhance extensibility and flexibility, Seshat is designed with modular annotators that operate independently by adhering to an API. This is a plug-and-play architecture that allows for the seamless addition, removal, and replacement of annotators based on the specific needs of a given medical procedure.

Each annotator runs as a self-contained HTTP server, optionally within its own containerised environment (e.g., Docker), making it possible to isolate dependencies, hardware requirements (such as GPU access), or language runtimes. They are implemented as a self-contained service capable of running separately from the main Controller. Annotators do not share internal state or execution logic, and instead communicate with the Controller over asynchronous HTTP requests.

This modular structure enables a plug-and-play style of development. As long as a new annotator conforms to the platform's predefined API contract, it can be integrated into the platform without modifying the Controller or other annotators, allowing for a multipurpose medical annotation. Additionally, it can run on a different network from Seshat's and allow contributions from a variety of researchers.

Using the Python programming language, the system implements a standardised interface for integrating machine learning models through a Flask-based HTTP API, enabling seamless incorporation of diverse annotation capabilities. This architecture establishes a clear contract between the core platform and external models, requiring each model to expose three fundamental endpoints that handle both individual frame processing and continuous video annotation.

At initialisation, each model automatically registers itself in the Controller through the dedicated registration endpoint, sending its network address and output file type. This registration persists until explicit unregistration or service termination, facilitated by the ”atexit” hook. The implementation supports TensorFlow/Keras models by default, loading the computational graph during startup from a predefined .keras file, while still maintaining flexibility for other frameworks through modular import patterns.

The entirety of the annotation mechanism revolves around a ”singleAnnotate” function that will set the annotator to perform its annotation on an image. Its primary purpose is to handle the complete pipeline from frame retrieval to mask generation for individual

timestamps. This function operates by first constructing the appropriate file paths using the exam identifier and the padded index values. In this case, the API was prepared to expect the annotator to annotate and save the annotation on the output path by reading the frame image (input path). It then returns a success message. In the case of the annotator producing PNG images as output, it also takes the colour and opacity attributes as arguments to reflect them in the visual characteristics of the annotation. For example, if the RGB translates to the colour blue and the opacity to 0.1, then the annotation will result in a PNG with a blue mask and an opacity of 0.1, allowing for an easy view of the area of interest caught by the annotator when displayed over the target frame.

To allow for the usage of real-time annotation, the function ”RTannotate” extends the core functionality through an interval-based processing loop that automatically handles frame advancement and termination conditions. It takes as arguments the first and last frame indexes to be annotated. The first indicates the index of the frame displayed in the Interaction when the user requested the beginning of the real-time annotation, and the last indicates the index of the last frame of the exam (for a pre-recorded exam video), indicating its duration. In a live video, since the duration is not defined (due to the stoppage of a live video), the last index is 999999999999 to indicate an exam with an unpredictable/unknown duration. In a loop that runs until the condition of a global metric indicating annotation is false, it checks if the frame is valid for processing, and then calls the ”singleAnnotate” function for it. The interval global parameter controls the temporal jump by specifying the milliseconds between processed frames and is updated regularly to allow adaptability to variations in processing times. This implies that the next frame to annotate will be the frame whose index is the sum of the current processed frame’s index with the interval. The logic behind this mechanism is that Seshat adapts to the processing times of the annotators and annotates at a rate compatible with the visualisation of the exam. For models that need colour attributes, this function has additional RGB and Opacity values to use them in the ”singleAnnotate” function call and reflect the user’s real-time annotation preferences.

The registration system establishes communication between annotation models and the Controller, and executes during model initialisation, transmitting essential metadata. This metadata includes the model’s network address and output file format specifications, packaged as a JSON payload and sent as an HTTP POST request to the Controller’s address.

The unregister function performs the inverse operation, removing the annotator from the

controller's service registry through an HTTP DELETE request. This clean-up process prevents the controller from attempting to route requests to unavailable services, maintaining system integrity. An ”atexit” hook ensures reliable clean-up by automatically invoking the unregistration process upon service termination. This mechanism registers the unregister function as a termination callback during initialisation, guaranteeing its execution independently of the shutdown circumstances. The hook triggers whether termination results from normal shutdown, errors, or external interrupts, ensuring lifecycle management without requiring explicit clean-up calls.

For communicating with the Controller, a series of endpoints was implemented. Together, these endpoints establish control over the annotation system that spans from individual to continuous frame processing. Firstly, the ”annotation” endpoint receives, through an HTTP request, the exam identifier, target frame index, and colour and opacity parameters to execute single-frame annotation operations. Its fundamental role involves redirecting these parameters to the ”singleAnnotate” function. Additionally, the mechanism for initiating and terminating the real-time annotation is handled by the endpoints ”/startRealTimeAnnotation” and ”/stopRealTimeAnnotation”, respectively. The starting endpoint receives a request with the exam identifier, the first and last frame indexes, and colour and opacity parameters. It sets the global flag for the annotation to true and calls the ”RTannotate” function with the same parameters sent with the request. On the opposite side, the stopping endpoint sets the global annotation flag to false so that the ”RTannotate” function stops its loop, terminating the annotator's processing. Lastly, to handle interval changes, the endpoint ”/setInterval” receives requests with the new interval value. It sets the global interval to the new value, implementing validation logic to ensure only values configure the annotation system.

4.4. Interaction

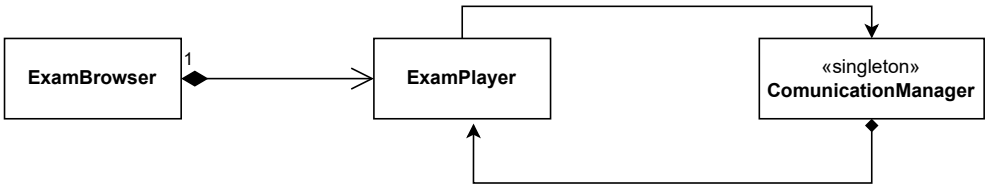


Figure 4.1: Class diagram for the three Interaction classes.

The Interaction component provides a responsive web-based interface that enables users to manage and interact with the platform directly through their browser, offering access

to the Repository's content for browsing, playing exams, and annotating them using the desired annotations. It is therefore the physical manifestation of this project, and where all the relevant features for the annotation of medical images become practical and concrete. It is implemented in the languages of HTML, CSS and JavaScript. Built with HTML5 for structural markup, CSS for visual styling, and JavaScript for interactive logic. Due to the fact that this web app mostly requires scripts, JavaScript is the richest in content and employs three major components, which are implemented classes, essential to the workflow of the app. Firstly, the ”ExamBrowser” is the orchestrator of the app, managing the navigation, UI, and global settings for reviewing and annotating exam videos. It coordinates the opening/closing of exams, propagates annotator and colour selections, and provides an interface for multi-exam workflows. Secondly, the ”CommunicationsManager” is responsible for handling the incoming server messages and distributing them across the different active exams. Lastly, the ”ExamPlayer” is a review tool for exams, supporting playback and annotation, with a focus on usability and extensibility for different annotators. This class reflects the exams in the tabs managed by the ”ExamBrowser”. Figure 4.1 contains a class diagram illustrating how these three classes interact.

The Interaction's graphical user interface (GUI) is constituted by a horizontal toolbar at the top, a large central content area destined for the exam playback, and a control bar along the bottom to control the exam's video flow. At the toolbar on the left side, the title of ”ExamBrowser” is displayed and next to it, an adder button to summon the list of available exams. On its right side, there are icons for turning the GUI full screen, a menu for selecting the colours, and a menu to select the available annotators. The space in between these icons is designated for the tabs for the available exams selected by the user to seamlessly navigate between them. Figure 4.2 displays the opened colour menu, which displays the colours, Blue, Yellow and Red, available for the user to select. All other menus of the GUI behave equally to this one.

When selecting an exam, its content will be displayed in the central area, which maximises the video playback area. A tab will be created at the top. Since only one exam can be played and selected at a time, the colour of the selected exam's tab will appear as white in contrast with the other tabs, which will be blue. A control bar at the bottom of the exam is displayed when selected, containing a seek bar for the video's duration, a button to toggle the visualisation of the annotation highlight menu, a play/pause button and a button destined for the real-time annotation that will toggle presentation according to its state (on or off). These enable control over the exams. The GUI is visible in Figure

4.3 where the exams ”test” and ”video” are available and the latter is selected.

Figure 4.2: Colour selection menu opened.

Figure 4.3: Seshat’s GUI.

ExamBrowser class

The ”ExamBrowser” class, illustrated in Figure 4.4 is initialised at the opening of the session. Upon initialisation, it constructs the complete Document Object Model (DOM) structure, generating the main browser container, header panel, and control elements, including the exams adder button, fullscreen toggle, colour palette selector and annotator selection menu. The user interacts with these elements through the use of ”onclick” mouse events. The exam adder functionality operates through a two-phase process. When users hover over the adder button, it automatically triggers an asynchronous request to the ”api/listExams” endpoint of the controller through the askForExams method. This fetch operation retrieves a JSON array containing all available exams from the server,

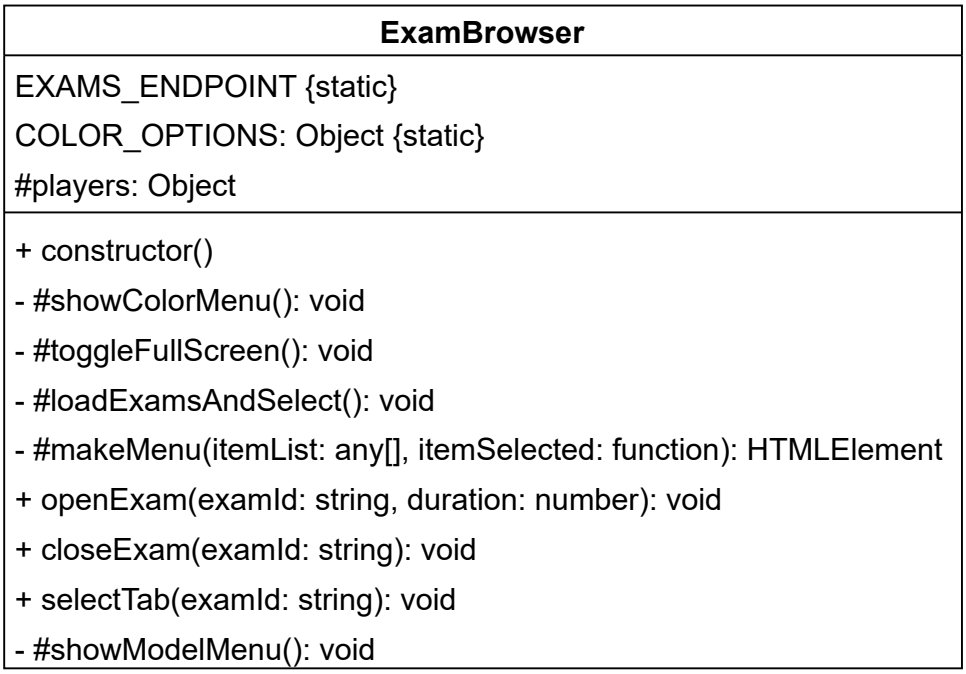


Figure 4.4: Class diagram of the ExamBrowser class.

which the system stores internally for subsequent access. The actual click event on the adder button then invokes a method that verifies if exam data has been successfully retrieved and proceeds to present the selection menu to the user. The menu presentation mechanism employs the selectExamFrom method, which processes the retrieved exam list. A live video feed option is added to the exam list since, in the Controller, there won't be a folder designated to it. The menu implementation includes sophisticated dismissal logic featuring both mouse-based closure and automatic timeout-based disappearance for minimal interference with medical workflow. The selection process translates user choices into concrete exam opening commands through the openExam method, which manages tab creation, exam player instantiation, and visual state management.

The fullscreen implementation uses the native Fullscreen API while implementing custom header behaviour that automatically shows and hides based on mouse proximity detection. This is calculated based on the user's cursor position relative to the viewport edges. An animation is then triggered to show or hide the headers.

Certain annotators require colour options to produce their outputs, such as segmentation masks. Since medical images have different environments with various colours, these annotations can visually blend with the medical images and become unnoticed, which would delay diagnosis. To counter this, a colour menu, built with the colours blue, red and

yellow already defined, is implemented. When open, it displays these three options, and when one of them is selected, a request is sent to the Controller indicating a colour switch. This request will be propagated to the annotator for it to change the colour attribute of the annotations. This feature provides users with the ability to change the colour of the annotations according to the different types of medical images.

To manage annotators, the system implements dynamic menu population through asynchronous fetching of available annotation models from the Controller. The method retrieves the current annotator registry and presents users with available options, with the selection propagating to all active exam players through a comprehensive notification system. This ensures consistent annotation capabilities across all the currently opened exams and maintains synchronisation between the interface state and the Controller.

The tab management system implements complete lifecycle control through openExam, closeExam, and selectTab methods that coordinate visual representation, player instantiation, and state synchronisation. The selection mechanism manages visual highlighting through CSS class manipulation and coordinates the showing/hiding of corresponding exam players. When opening an exam, a new tab is created in the header with the exam's name and a close button, then a new ”ExamPlayer” instance is created for that exam and added to the workspace. If the target exam is already open, then its tab is automatically selected. Upon clicking the close button, the closeExam method is called, which deletes the tab and player references from the internal mappings. If the closed exam is the currently selected one, it resets the selected exam state to null, ensuring the workspace reflects the change and no exam is selected.

CommunicationsManager class

Demonstrated in Figure 4.5, the ”CommunicationsManager” class is designed as a singleton to ensure that only one instance manages real-time communication between the client and the backend using Server-Sent Events (SSE). This is enforced by a static flag and a private static instance variable, so any attempt to instantiate the class directly results in an error, and the only way to obtain an instance is through the static ”getInstance()” method. When the singleton is created, it initialises an SSE connection to the ”api/stream” endpoint. The class maintains a dictionary of callbacks, each associated with a specific exam ID. When an SSE message arrives, the event handler parses the JSON data to extract the exam ID and timestamp, then checks if a callback has been registered for

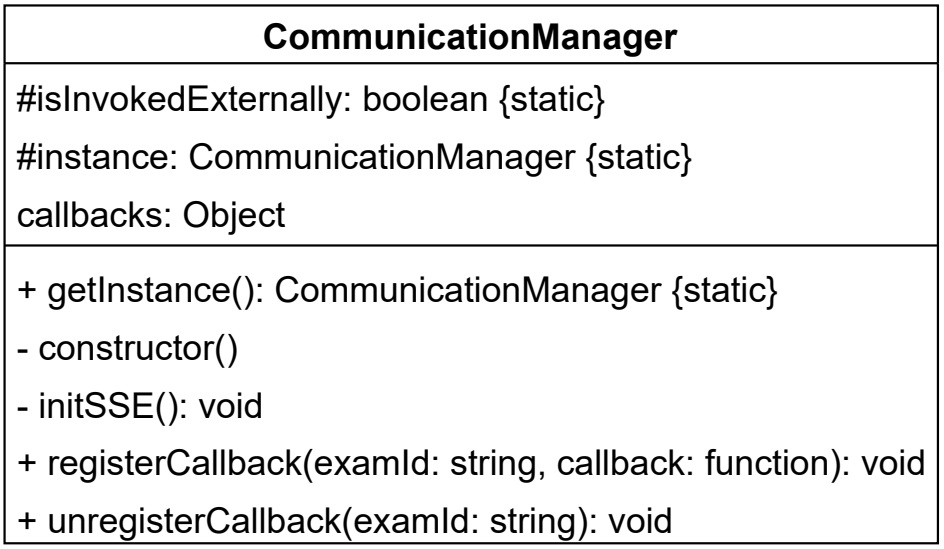


Figure 4.5: Class diagram of the CommunicationsManager class.

that exam ID; if so, it invokes the callback with the timestamp, allowing the corresponding part of the application (such as an ”ExamPlayer” instance) to react to the update. Furthermore, it employs methods that allow other components to subscribe or unsubscribe from updates for specific exams, making the communication system flexible and decoupled. For example, when an exam is selected, it registers itself in the class, and when it is closed by the user, it decouples from the class.

ExamPlayer class

When the ”ExamPlayer” class is created, it first gets an instance of the ”CommunicationsManager” to allow for the reception of server-sent events from the Controller. The constructor initialises an exam environment by first distinguishing between live and pre-recorded exam types through their ID. For live video exams, it generates a unique timestamp-based ID to ensure filesystem compatibility and intuitiveness, then triggers live video acquisition through the ”api/liveVideo” endpoint. For pre-recorded exams, it directly uses the provided ID. Figure 4.6 illustrates a diagram of this class.
The visualisation foundation establishes an HTML5 canvas element configured to medical imaging standards with a resolution of 1920x1080, providing adequate detail for medical review. The canvas rendering context enables the display of the frames of the videos and an overlay composition for the annotations. The temporal management system uses performance.now() timestamps to track the exam’s initiation time, current playback position, and synchronisation references to allow for accurate frame calculation based on elapsed

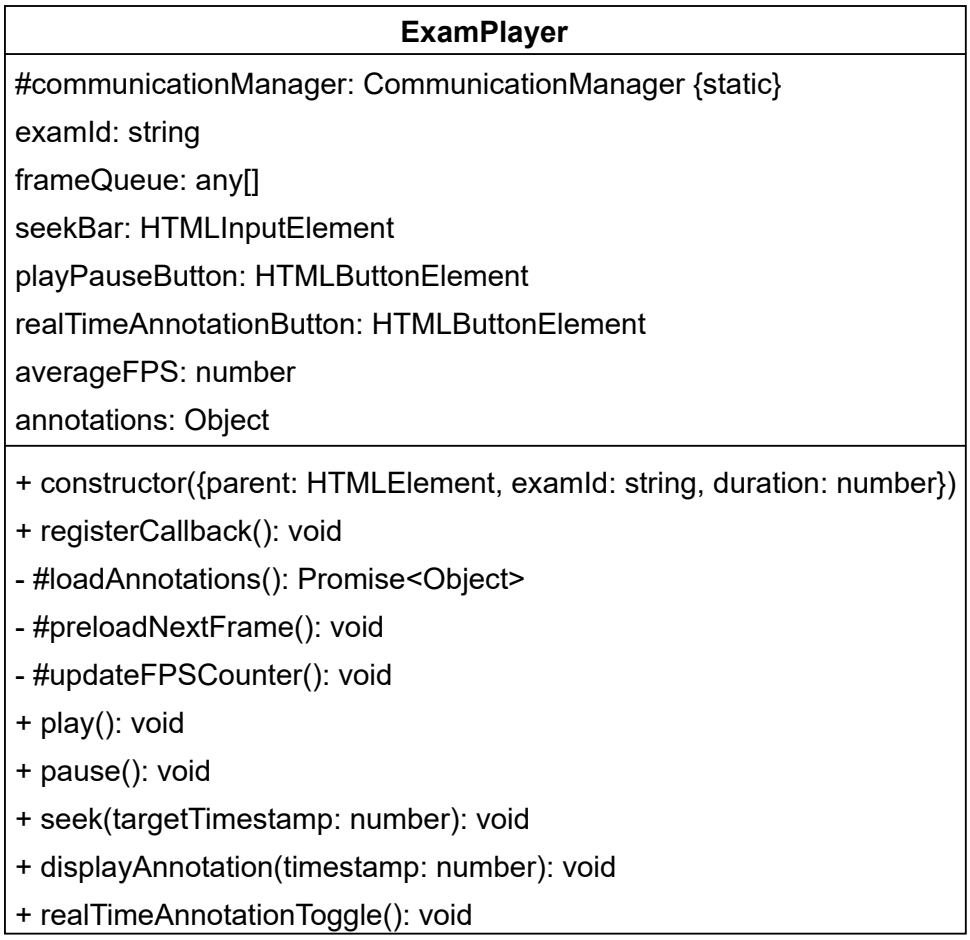


Figure 4.6: Class diagram of the ExamPlayer class.

time. To handle the frame fetching, the playback parameters are configured. These include dynamically configurable frame rates that adapt between live and pre-recorded exams due to the logic behind the decoding mechanisms of FFmpeg. The decoding of the video feed into frames is made firstly by extracting the frames and then by renaming and organising them. Since, after the first step of this mechanism, the frames are named in increments of one, the logic for fetching the live video frames is also made in increments of one in the naming system. This eliminates the need for the second step of decoding to further reduce the time taken for the display of frames. In the case of a pre-recorded exam, this logic is implemented by default in increments of 40, equal to the increments of the exams in the Repository, to account for the target 25 fps.

To provide video playback features to the user, the control interface constructs an overlay panel containing interactive elements. The seek bar is associated with the seek method that implements frame-accurate navigation through a range input element bound to the

exam's total duration, with event handlers that translate user input into precise frame index seeking operations. In the case of Live exams, this element of the UI provides no use since the seeking operation is not used during these procedures. Additionally, a play/pause button is bound to the play and pause methods that play and pause the fetching and display of frames. Finally, a button for starting and stopping Real-Time annotation is added that will coordinate the beginning and the end of continuous frame annotation.
Since maximising viewing area is crucial for medical imaging while maintaining immediate access to controls, the system implements auto-hiding behaviour that reveals the playback controls on mouse movement and gradually fades them during inactivity. The system employs a timeout-based hiding with immediate cancellation on user interaction. Finally, in the constructor, keyboard event listeners provide additional navigation capabilities through the arrow keys for temporal skipping, adding depth to the video playback controls.
Since maximising viewing area is crucial for medical imaging while maintaining immediate access to controls, the system implements auto-hiding behaviour that reveals the playback controls on mouse movement and gradually fades them during inactivity. The system employs a timeout-based hiding with immediate cancellation on user interaction. Finally, in the constructor, keyboard event listeners provide additional navigation capabilities through the arrow keys for temporal skipping, adding depth to the video playback controls.

Annotation handling system

The annotation management system maintains a persistent record of all annotation activities associated with an exam. This is managed with a list of all annotations made by storing the frame index corresponding, information on the responsible annotator and the time taken to process the annotation. When an exam is loaded, the system retrieves stored annotation metadata from persistent storage on the Repository, reconstructing the complete annotation history for the exam session. This metadata maps temporal positions to corresponding annotation outputs, preserving the relationship between video frames and their associated analyses. The system periodically saves annotation data to ensure no analytical work is lost during extended review sessions or unexpected interruptions.
A timeline visualisation component provides an overview of annotation distribution across

the exam duration. This interface element can be shown or hidden based on user preference, toggling between expanded and collapsed states to balance information density and screen space utilisation. The visualisation displays colour-coded markers representing different annotation models at their respective temporal positions, enabling quick assessment of annotation coverage and density. Users can remove individual annotations through an interaction mechanism that first confirms the deletion intent before permanently removing the annotation reference from the metadata storage and updating the visual representation accordingly. In addition to this, Seshat also has the feature to seek the video playback to a certain annotation by left-clicking the target annotation on the highlight menu, which results in the video setting its current playback time to the frame that corresponds to the target annotation. An example of the annotation highlight menu feature is shown in Figure 4.7, where an endoscopy exam is annotated with two different annotators.

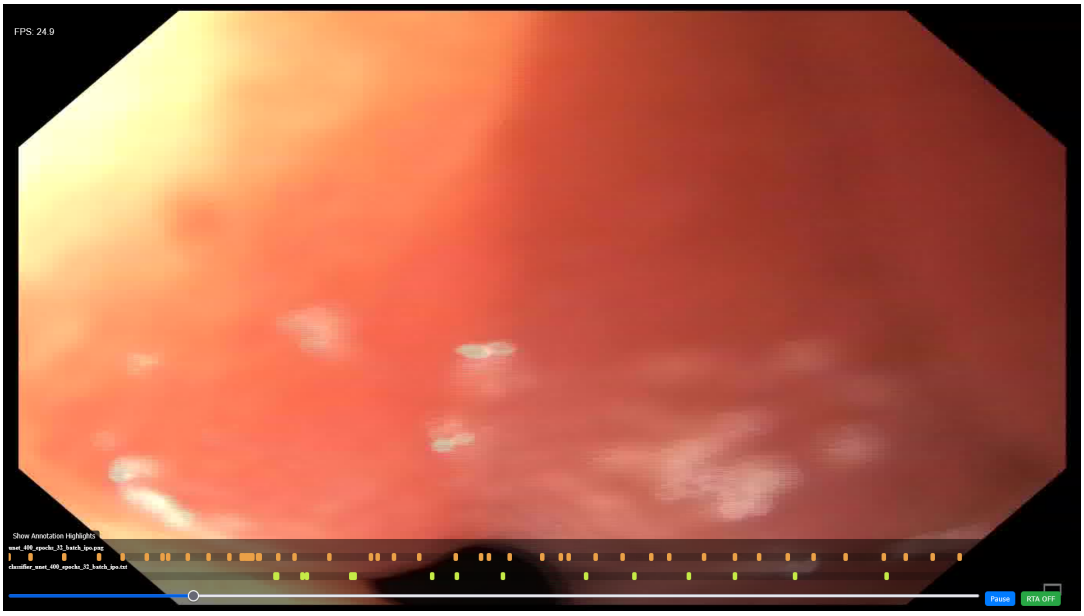


Figure 4.7: Annotation highlight menu on an endoscopy exam.

Frame handling

The system responsible for handling video frames operates through a sequence of loading, retrieval, and display processes. These components work together to maintain accurate timing while ensuring consistent playback quality. The preloading mechanism functions as a buffer that obtains frames of the current frame being displayed to give continuity to the player. This component continuously determines the next expected frame based

on the current playback indexes, then begins loading subsequent frames asynchronously. Each preloaded frame is stored with its corresponding index information, allowing the system to identify and retrieve the appropriate frame when required. The design for this methodology was a particular challenge due to the different types of implementation. In contrast to a frame-based mechanism, this time-based implementation presents issues, such as the desynchronization between time and video playback in cases of low bandwidth due to some frames arriving late to the pipeline. This time-based mechanism guarantees full accuracy of video playback and adapts to lower bandwidth by skipping missing frames.

The frame preloading process collaborates with timing mechanisms that synchronise what is shown on screen with the actual passage of time. The system uses a time-based approach where each frame has an assigned 40-millisecond display period that corresponds to the standard 25 frames per second playback rate. The retrieval mechanism verifies whether the needed frame exists in the preparatory buffer - if available, it obtains and displays the frame; if unavailable, it records a missed frame and moves to the next time slot. This temporal coordination guarantees that video playback maintains a consistent pace independent of processing delays or network variations.

The navigation feature functions in a manner distinct from the continuous playback mechanisms. When navigation occurs, the system immediately stops all preparatory loading and retrieval activities, clears the existing frame collection, and calculates the exact destination frame based on the navigation position, given by the HTML range. Rather than depending on time-based progression, navigation uses direct frame identification to jump to the specified moment. The system loads the target frame in a focused manner, ensuring prompt display, then restarts the preparatory loading sequence from the new position. During standard playback, the seek bar progress indicator position updates continuously according to the current timestamp calculation, offering real-time feedback on playback advancement. When users manually adjust the progress indicator, the system captures the input value and performs a navigation operation to the corresponding timestamp. For live video exams, the progress indicator dynamically expands as new frames are obtained, with the maximum value increasing as time passes to translate the increasing time taken by the procedure.

Play and pause operations manage the overall operational mode of the playback system. Activating play mode starts the timing cycle, initiating the continuous frame advancement process. The system begins preloading frames from the current position and enters the

display cycle, where it regularly checks and shows frames following its schedule. Pause mode halts this advancement, keeping the current frame visible while preserving the playback status. The system continues to preload frames near the paused position to guarantee immediate responsiveness when playback continues, but suspends the time-based advancement calculations.

The performance counter implements a moving average system that supplies performance information in real-time. The system maintains a rotating buffer of frame display times over a one-second period, continually computing the average frames per second based on successfully shown versus missed frames. The counter refreshes at regular intervals, smoothing out temporary variations to provide stable performance measurements. This monitoring works underneath all other frame handling processes, passively observing the display pipeline without disrupting playback operations. The displayed performance value represents the actual display capability rather than the intended frame rate, providing users with precise feedback on system performance. This metric is shown on a UI element placed on the bottom right corner of the exam player to avoid visual clutter and provide a precise analysis of the performance. This feature is visible in Figure 4.8.

The complete video playback sequence follows a pipeline that starts with the timing system identifying which frame should be currently visible based on elapsed time. This calculated frame index is verified against the buffer, and if it is present, the frame moves directly to display; contrarily, the system attempts to retrieve it directly while recording a missed frame. During display, the canvas is cleared and the new frame is presented, followed by any annotation overlays matching the current timestamp. At the same time, the preparatory loading system looks forward to buffering subsequent frames, the progress indicator updates to show the current position, and the performance counter recalculates performance measurements. This ongoing cycle maintains a steady flow of frame processing where each element works concurrently but stays synchronised through the central timing mechanism.

When network conditions cause frame delivery delays, the playback continues with frame omission rather than interruption, maintaining the flow of the video player despite occasional skipped frames. For navigation operations that target frame indexes where frames are not yet available, the system shows loading indicators while asynchronously obtaining the required content. The preparatory loading system dynamically modifies its look-ahead distance based on current performance measurements, extending the buffer during unstable network conditions and reducing it when performance is optimal.

Annotation display

The system provides dual modes for annotation display, handling both real-time annotation streams and previously stored annotations. When new annotations are generated, they are immediately rendered as overlays on the current video frame, with visual representations tailored to the annotation type—whether graphical overlays for image-based annotations or formatted text displays for textual analyses. When they are requested, the video player pauses, requests an annotation from the current model to the Controller, and when the SSE notifies the conclusion of the annotation, it is displayed for the user to see the annotation on the frame. For previously stored annotations, the system retrieves the annotation data from local metadata stores and renders them using the same visualisation logic, basing itself upon the list of annotation metadata, guaranteeing consistency between real-time and historical annotation displays. Since the annotations are made to a corresponding frame, if no other logic for displaying were to be applied, the annotation would disappear in the frame next to it, which would result in the saved annotation not being seen in detail. To counter this issue, the display of annotations saved in the list lasts for a period of two seconds, guaranteeing that the user can view the annotation and interpret it. This works by maintaining a list of annotations to be displayed in the following frames with information regarding the time/frames left to be displayed. In Figure 4.8, a segmentation mask is shown in the colour blue over an endoscopy exam.

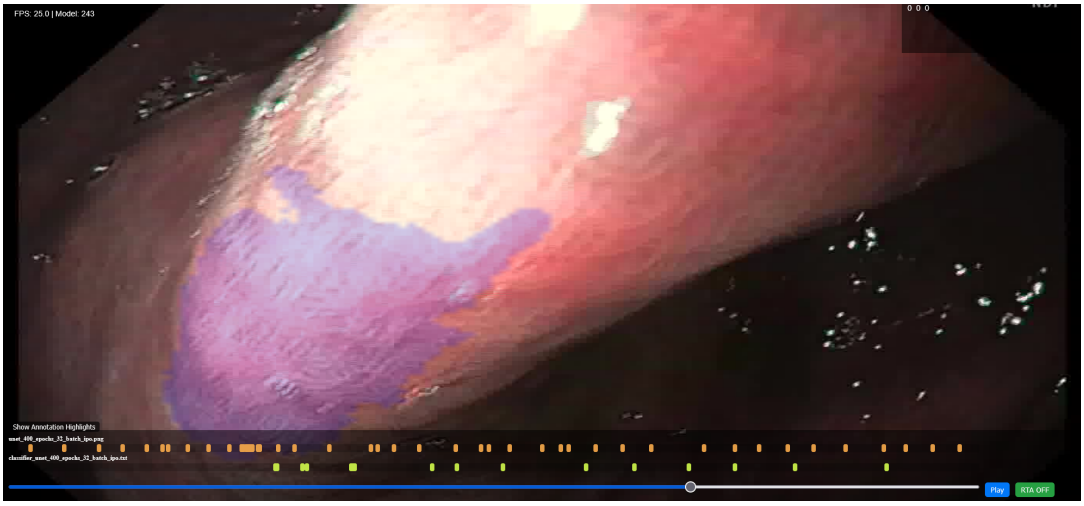


Figure 4.8: Segmentation annotation on an endoscopy exam.

Real-time annotation

To handle real-time annotation, a toggle mechanism controls continuous annotation processing during video playback. This serves as the primary interface for managing this automated annotation process, switching between inactive and active states via a button on the control bar. When activated, the system initiates communication with the Controller to signal the intent to start processing images in real-time with a certain target annotator. This is made via a request that contains the index of the frame in which the activation occurred, the current frame, the total duration of the exam and the target exam. The activation propagates to the Controller to the target annotator, who starts to annotate from the current frame onwards until the exam ends. When annotations arrive precisely at their scheduled timestamp during playback, the system executes an immediate display process. The annotation content is processed and rendered directly onto the current video frame, providing real-time visual feedback. In the case of an annotation completion notification arriving before the corresponding video frame has been reached during playback, the system stores the annotation on the annotation metadata to be shown when the playback reaches its time. On the opposite case, annotations that arrive after their scheduled display time has passed during playback are immediately added to the annotation metadata list but not displayed in the current playback due to their lateness. Even though these late annotations are no longer useful for the current visualisation, they are saved to be reviewed in a future playback.

When deactivated, the system terminates the annotation stream, while preserving any annotations generated during the active period. The Controller is sent a request signalling this operation,n which is propagated to the annotator to stop the real-time annotation. This functionality supports both retrospective analysis of pre-recorded content and real-time analysis of live video feeds, with the system adapting its behaviour based on the exam type.

5. Validation

This chapter presents the validation of the Seshat platform. This topic is divided into two major categories that reflect the different validation approaches: the empirical results, which measure the compliance of Seshat with the performance objectives and the usability tests, which involve a selected group of evaluators to test Seshat and review it according to its usability. In this chapter, the term real-time refers to near real-time performance, where strict per-frame timing guarantees are not enforced but overall responsiveness and usability are preserved.

5.1. Empirical results

The performance evaluation of the Seshat platform was conducted via testing across different deployment environments to assess its compliance with performance objectives. An initial testing phase was performed on consumer-grade hardware consisting of an Acer Predator Helios 300 laptop equipped with an Intel Core i7-9750H processor and 16GB of RAM. This hardware configuration, while adequate for development and preliminary testing, does not represent optimal production environment specifications. Subsequent testing was conducted on a dedicated server environment to provide more representative performance metrics for clinical deployment scenarios.

The evaluation framework focused on video acquisition latency and visualisation frame rates. Video acquisition performance measures the temporal delay between the moment the user initiates video playback (whether live camera feed or pre-recorded content) and the display of the first frame within the exam player. For live exams, this means the delay between what is happening during the procedure and the visualisation of the live procedure on Seshat. This is measured using the difference between the point in time at which the user requests the beginning of the video conversion and the point in time at which the first frame is displayed.

Visualisation performance assesses the system's ability to maintain consistent frame rates during exam review, which is crucial for diagnostic accuracy. This is measured by calculating the average FPS in the exam player throughout the duration of the exams.

For pre-recorded exams, the platform exhibits no acquisition latency, as the video conversion process occurs before user access, meaning it is already in the Repository format. However, live video acquisition involves substantial processing overhead as the system must capture, convert, and format the exam video feed in real-time before transmission

to the client interface. This fundamental difference accounts for the significant disparity in performance metrics between the two exam types.

Comparing both tests, the server-based environment revealed substantial improvements in system responsiveness compared to the initial consumer hardware deployment. As detailed in Table 5.1, Firefox exhibited an average acquisition latency of 295ms with a standard deviation of 51.78ms compared with 652ms with 197ms standard deviation, while Microsoft Edge and Opera demonstrated more consistent performance with averages of 228ms and 227ms compared with 462ms with 30ms standard deviation, respectively, accompanied by minimal standard deviations of 15.5ms and 16.9ms compared with 562ms with 93ms standard deviation. All tested browsers successfully maintained the target frame rate of 24.5 frames per second on average, ensuring smooth video playback suitable for medical diagnostics.

The reduced latency and improved consistency observed in the server environment can be attributed to several factors: in the server environment, the client and server processing were being done in separate machines, decreasing the computational load of the server, faster server processing capabilities, optimised resource allocation, and reduced background contention. The narrow standard deviations, particularly evident in Edge and Opera, indicate stable performance characteristics essential for predictable clinical workflows. The maintained frame rate of 24.5 fps across all platforms confirms the system's ability to deliver consistent visualisation quality in different possible client environments.

These empirical results demonstrate that while Seshat remains functional on consumer hardware, its performance characteristics significantly improve when deployed on dedicated server infrastructure. The achieved latency figures and frame rate consistency meet the requirements for clinical use. These results indicate that Seshat achieves near real-time responsiveness suitable for interactive medical annotation workflows and although it does not guarantee hard real-time constraints, occasional frame skipping or latency variation does not compromise its intended use as an interactive annotation platform.

Browsers	**Firefox**	**Edge**	**Opera**
Average $\pm$ σ laptop tests	652ms $\pm$ 197ms	462ms $\pm$ 30ms	562ms $\pm$ 93ms
Average $\pm$ σ server tests	295ms $\pm$ 52ms	228ms $\pm$ 15ms	227ms $\pm$ 16ms

Table 5.1: Delays on the laptop and server tests

5.2. Usability tests

To further validate Seshat, a usability test was conducted to evaluate the integration of annotators into the platform. It employed a mixed-methods approach combining heuristic evaluation and simplified cognitive walkthrough to assess the researcher-facing components of the Seshat platform. The study focused specifically on the annotator integration workflow, targeting the primary user group of research scientists with software development knowledge.

The evaluation involved 4 participants with backgrounds in software development and machine learning research. This participant profile was selected as the evaluators represent the intended users responsible for integrating ML annotators into the Seshat ecosystem, since these, contrary to medical practitioners, are the ones who will integrate the developed annotators into the platform, and therefore the selected user group for this usability test.

The testing environment utilised containers accessed via SSH, with port forwarding enabling participants to interact with the web application interface. The installation and requirements for running Seshat were prepared to ensure that the containerised approach mirrored real-world deployment scenarios and simultaneously provided consistent testing conditions across all participants. Participants were presented with the core task of integrating their machine learning annotators into Seshat using the provided API framework. For researchers who did not have readily available annotators, a pre-configured annotator was supplied to ensure consistent task execution across all participants. Tasks given to the evaluators ranged from running Seshat to integrating annotators and visualising their annotations, covering the essential tasks needed to use the platform after deployment. The simplified cognitive walkthrough success rates for each task segment were recorded to quantify the integration workflow's difficulty and accessibility. The heuristic assessment employed Nielsen's ten usability principles without domain-specific adaptations, applying the standard criteria to evaluate the annotator integration experience. Researchers assessed the interface against established heuristics, including system status visibility, match between system and real world, user control and freedom, consistency and standards, error prevention, recognition rather than recall, flexibility and efficiency of use, aesthetic and minimalist design, help users recognise, diagnose and recover from errors, and help and documentation.

The evaluation primarily generated quantitative data through success rate metrics in the

simplified cognitive walkthrough, measuring the difficulty with which the evaluators completed each task. The heuristic evaluation provided more qualitative data on the degree to which the heuristics impacted each task according to each user's personal opinion. Furthermore, additional qualitative insights emerged through a post-test debriefing session where participants collectively shared their experiences, challenges, and suggestions. The moderated discussion format facilitated comparative analysis of different researchers' perspectives on the integration process.

Across all the evaluators, the key findings are the learning curve of tasks, their choke points and the most frequent heuristics. As expected, when handling a new platform, the learning curve is low at the beginning and increases as tasks are completed, and so was the case during these tests. Most users took more time completing the early tasks, and as they got familiarised with the structure and understood the role of each component, the time taken per task decreased.

The most frequent heuristic was Nielsen's first: Visibility of system status. Researchers encountered situations where the system failed to provide clear indications of annotation processing progress or error states during API configuration. The root cause of this violation stems from the platform's primary focus on functionality rather than user interface refinement. Without a dedicated UI/UX design study and development, the system often left evaluator uncertain about whether their integration attempts were progressing successfully, had stalled, or had failed. For instance, during annotation processing, the absence of progress indicators or clear success/failure notifications forced evaluators to rely on indirect cues or technical workarounds to ascertain the state. Table 5.2 summarises the most frequent heuristics the evaluators noted and their severity.

The results from the simplified walkthrough revealed a general success from the evaluators when completing most tasks. Noticeably, only the final task was executed with some difficulty by evaluators, which can be explained by the increased complexity of this task. Apart from this, the remaining tasks were completed without much difficulty, which reflects the usability of using Seshat to integrate annotators.

Heuristics	Frequency	Highest severity
1 - Visibility of System Status	3	8
8 - Aesthetic and Minimalist Design	2	8
10 - Help and Documentation	1	4
3 - User Control and Freedom	1	4

Table 5.2: List of the heuristics' frequency on the usability tests.

6. Conclusions

Seshat is an ongoing development effort aimed at providing a flexible and modular architecture for near real-time video annotation in medical contexts. It is best characterized as a near real-time system, designed to support interactive medical video annotation rather than to satisfy strict hard real-time constraints. As a work in progress, the current development version already demonstrates promising results towards real-time annotation and well-structured core functionalities. Initial tests already show sub-second delays and stable frame rates, which are promising results and strong indicators of the platform's near real-time capabilities. While the foundational architecture is in place, further iterations will naturally refine aspects such as performance optimisation, user interface robustness, improved user experience, and adaptation to restrictions on bandwidth and latency. Rather than presenting a fixed set of future work tasks, we view this project as evolving in alignment with feedback and real-world deployment challenges.

This dissertation set out to design, implement and validate Seshat, a platform to address the challenges of integrating AI models into medical video analysis and clinical workflow. The primary objective was to create a flexible and modular system that allows researchers and medical practitioners to incorporate annotators into pre-recorded exams as well as near real-time exams, overcoming the limitations of the restrictive and ambiguous existing solutions. The development supported by the empirical tests made to the platform's core components demonstrates that this objective has been achieved. Therefore, Seshat prevails as a functional and validated platform that effectively joins AI models and their capabilities in a single environment fit for the clinical workflow.

Designing and implementing Seshat to be based on the Repository, the Controller, the Interaction and the Acquisition was proven to be highly effective in achieving modularity and near real-time performance. Each of these components plays a role, with the Repository providing a standardised indexed structure for storing data, the Controller, which orchestrates other components and manages their communication, the Acquisition, which converts the video feeds into a compatible format for Seshat to function and the Interaction that offers a user interface to unlock Seshat's features. This architecture is enabled by two essential key contributions: a plug-and-play API that allows for the seamless integration of multiple, simultaneous AI models by adhering to a simple HTTP contract, and a low-latency notification pipeline built on file system monitoring and Server-Sent Events.

This systematically detects and propagates annotations to the user interface with minimal delay, a crucial requirement for live clinical use.

By achieving a stable average of 24.9 frames per second and low latency figures in server empirical tests, the platform has a performance adequate for practical use, such as clinical review and being a valuable tool for the development of annotators. The platform's utility was further validated through heuristic evaluation and simplified cognitive walkthroughs conducted with evaluators possessing software development expertise. The findings confirmed that the API successfully integrates annotators into Seshat. Furthermore, the elements and features of the user interface, although with room for improvement, proved to be effective for the core task of annotating exams.

In conclusion, Seshat delivers a versatile and modular solution to the challenge of incorporating AI in medical images. Its architecture addresses the problem of integration and establishes a versatile base for future expansions. Therefore, Seshat creates opportunities for accelerating AI research in medicine and supports multiple applications, including manual annotation, model validation, and telemedicine by having achieved near real-time responsiveness. These capabilities align the platform with its integration in medical environment.

6.1. Future work

Beyond the previously mentioned technical enhancements, the versatile architecture of Seshat allows bright opportunities for expansion into several distinct use cases. A high-impact extension would be to evolve the platform into a comprehensive tool for manual data annotation. The existing infrastructure for displaying video frames and handling annotators and their annotations can be extended with new user interface elements to allow humans to draw bounding boxes, segmentation, or input textual annotations. By serving as a centre for creating validated datasets, Seshat could directly intensify the development and training of the very AI models it is designed to integrate.

Furthermore, the web-based nature of the Interaction component provides a foundation for using Seshat as a telemedicine and collaborative annotation tool. The platform could be extended with real-time features, allowing various clinicians in different networks to view the same examination and discuss their findings.

Finally, the platform is ideally positioned to become a benchmarking and validation suite for AI annotators. The system could be extended to include automated metrics calculation and host thresholds to serve as validation for annotators. Researchers could submit

their models to be evaluated against a standardised battery of examinations adequate for the annotator's fields, with Seshat generating the annotations performance report. This would provide a valuable framework for validating model robustness and performance, accelerating their development and deployment.

References

[1] n.a., “Artificial intelligence–assisted endoscopy in diagnosis of gastrointestinal tumors: A review of systematic reviews and meta□analyses,” *PubMed*, 2024. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/40980773/ [Cited on page 1.]

[2] ——, “Real□time artificial intelligence versus standard colonoscopy in the early detection of colorectal cancer: A systematic review and meta□analysis,” *Healthcare*, 2025. [Online]. Available: https://www.mdpi.com/2227-9032/13/19/2517

[3] ——, “Artificial intelligence in upper gastrointestinal endoscopy,” *Digestive Diseases*, 2025. [Online]. Available: https://karger.com/ddi/article/40/4/395/827931/ [Cited on page 1.]

[4] ——, “Edge artificial intelligence device in real-time endoscopy for classification of gastric neoplasms,” *MDPI Diagnostics*, 2025. [Online]. Available: https://www.mdpi.com/2313-7673/9/12/783 [Cited on page 2.]

[5] ——, “Evaluation of an artificial intelligence□based system for real□time high□quality photodocumentation during esophagogastroduodenoscopy,” *Scientific Reports*, 2025. [Online]. Available: https://www.nature.com/articles/s41598-024-83721-9 [Cited on page 2.]

[6] ——, “Artificial intelligence in digestive endoscopy training: Past, present, and future,” *PubMed*, 2025. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/41139585/ [Cited on page 2.]

[7] ——, “Artificial intelligence in gastrointestinal bleeding analysis for video capsule endoscopy: Insights, innovations, and prospects,” *arXiv*, 2024. [Online]. Available: https://arxiv.org/abs/2409.00639 [Cited on page 2.]

[8] R. et al., “Artificial intelligence-assisted colonoscopy for detection of colorectal neoplasia: Prospective, randomized trial,” *Gastroenterology*, 2022. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/35185345/ [Cited on page 6.]

[9] U. et al., “Deep learning for real-time detection of polyps in colonoscopy videos,” *PLoS ONE*, 2018. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0207881 [Cited on page 6.]

[10] Z. et al., “Real-time ai-assisted detection of early gastric cancer: A multicenter prospective study,” *Gastrointestinal Endoscopy*, 2023. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/36851731/ [Cited on page 6.]

[11] W. et al., “Development and validation of a deep learning algorithm for detection of gastric cancer in endoscopic images,” *Gastroenterology*, 2020. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/32713852/

[12] G. et al., “Recent advances in medical video analysis using deep learning: A review,” *IEEE Access*, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10093707 [Cited on page 6.]

[13] Health Research Authority, “Feasibility of cadu system and bio-impedence in barrett’s oesophagus,” https://www.hra.nhs.uk/planning-and-improving-research/application-summaries/research-summaries/feasibility-of-cadu-system-and-bio-impedence-in-barretts-oesophagus/, n.d., accessed: 2025-09-30. [Cited on page 7.]

[14] Odin Vision, “Caddie brochure,” https://odin-vision.com/wp-content/uploads/2022/03/caddie-brochure.pdf, 2022, accessed: 2025-09-30.

[15] ——, “Cadu-2,” https://odin-vision.com/en_gb/cadu-2/, n.d., accessed: 2025-09-30.

[16] Olympus Corporation of the Americas, “First cloud-based ai endoscopy system for colonoscopy receives fda clearance,” https://olympusamerica.com/press-release/2024-09-05/first-cloud-based-ai-endoscopy-system-colonoscopy-receives-fda-clearance, 2024, accessed: 2025-09-30.

[17] Odin Vision, “Odin vision homepage,” https://odin-vision.com/en_gb/, n.d., accessed: 2025-09-30. [Cited on page 7.]

[18] AI-MS, “Product announcement – 2024-03-04,” https://en.ai-ms.com/news/product/20240304, 2024, accessed: 2025-09-30. [Cited on page 8.]

[19] AI Medical Service Inc., “Endoscopic ai: Ai medical service inc. has obtained regulatory approval from thailand’s fda for its gastric ai-based endoscopic diagnosis support system,” https://www.businesswire.com/news/home/20250724574631/en/Endoscopic-AI-AI-Medical-Service-Inc.-Has-Obtained-Regulatory-Approval-from-Thailands-FDA-for-Its-Gastric-AI-based-Endoscopic-Diagnosis-Support-System, 2025, accessed: 2025-09-30. [Cited on page 8.]

[20] J. Cao, H.-C. Yip, Y. Chen, M. Scheppach, X. Luo, H. Yang, M. K. Cheng, Y. Long, Y. Jin, P. W.-Y. Chiu *et al.*, "Intelligent surgical workflow recognition for endoscopic submucosal dissection with real-time animal study," *Nature Communications*, vol. 14, no. 1, p. 6676, 2023. [Cited on page 9.]

[21] CUHK Medicine, "Cu medicine proves ai-assisted colonoscopy increases adenoma detection rate by 40% and trains a new ai platform to assist early-stage gastrointestinal cancer treatment," https://www.med.cuhk.edu.hk/press-releases/cu-medicine-proves-ai-assisted-colonoscopy-increases-adenoma-detection-rate-by-40-and-trains-a-new-ai-platform-to-assist-early-stage-gastrointestinal-cancer-treatment/, 2023, accessed: 2025-09-30.

[22] CUHK OAL / CU Medicine, "Ai-powered breakthrough boosts early cancer detection rates," https://www.oal.cuhk.edu.hk/cuhkenews_202401_detect_gastrointestinal_cancers/, 2024, accessed: 2025-09-30. [Cited on page 9.]

[23] J. H. Kim, J. Wang, C. Pence, P. Magahis, E. Dawod, F. Schnoll-Sussman, R. Z. Sharaiha, and D. Wan, "Gi genius increases small and right-sided adenoma and sessile serrated lesion detection rate when used with endocuff in a real-world setting: a retrospective united states study," *Clinical endoscopy*, vol. 58, no. 3, p. 438, 2025. [Cited on page 10.]

[24] A. Savino, E. Rondonotti, S. Rocchetto, A. Piagnani, N. Bina, P. Di Domenico, F. Segatta, and F. Radaelli, "Gi genius endoscopy module: a clinical profile," *Expert Review of Medical Devices*, vol. 21, no. 5, pp. 359–372, 2024.

[25] Medtronic, "Gi genius™ intelligent endoscopy module," https://www.medtronic.com/en-us/healthcare-professionals/products/digestive-gastrointestinal/gastrointestinal-artificial-intelligence/gi-genius-intelligent-endoscopy-module.html, n.d., accessed: 2025-09-30. [Cited on page 10.]

[26] Wision A.I., "Endoscreener — wision a.i." https://www.wision.com/#/endoScreener, n.d., accessed: 2025-09-30. [Cited on page 12.]

[27] M. T. Wei, U. Shankar, R. Parvin, S. H. Abbas, S. Chaudhary, Y. Friedlander, and S. Friedland, "Evaluation of computer-aided detection during colonoscopy in the community (ai-see): a multicenter randomized clinical trial," *Official journal of the*

American College of Gastroenterology| ACG, vol. 118, no. 10, pp. 1841–1847, 2023. [Cited on page 13.]

[28] PENTAX Medical, "Discovery™: Ai-powered colorectal cancer detection," https://www.pentaxmedical.com/en/products/discovery-ai, n.d., accessed: 2025-09-30. [Cited on page 15.]

[29] Olympus Corporation, "Endo-aid: Ai-powered endoscopy cad system," https://www.olympus-europa.com/medical/en/Products-and-Solutions/Products/Product/ENDO-AID.html, n.d., accessed: 2025-09-30. [Cited on page 16.]

[30] E. Hayrapetyan, "Case study: Cad eye® by fujifilm–ai-enhanced colonoscopy detection and diagnosis," 2025. [Cited on page 17.]

[31] FUJIFILM Corporation, "Cad eye — endoscopy ai support (global)," https://www.fujifilm-endoscopy.com/cadeye, n.d., accessed: 2025-09-30.

[32] ——, "Cad eye — endoscopy ai support (singapore / sg site)," https://www.fujifilm.com/sg/en/healthcare/endoscopy/endoscopy-processors/cadeye, n.d., accessed: 2025-09-30. [Cited on page 17.]

[33] B. Chen, H. Ling, X. Zeng, J. Gao, Z. Xu, and S. Fidler, "Scribblebox: Interactive annotation framework for video object segmentation," in *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII*. Springer International Publishing, 2020, pp. 293–310. [Cited on page 20.]

[34] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "Labelme: A database and web-based tool for image annotation," *International Journal of Computer Vision*, vol. 77, no. 1-3, pp. 157–173, 2008. [Online]. Available: https://doi.org/10.1007/s11263-007-0090-8 [Cited on page 20.]

[35] H. Mahmoud and R. Abozariba, "A systematic review on webrtc for potential applications and challenges beyond audio video streaming," *Multimedia Tools and Applications*, pp. 1–38, 2024. [Cited on page 20.]

[36] A. Sassu, J. F. Saenz-Cogollo, and M. Agelli, "Deep-framework: A distributed, scalable, and edge-oriented framework for real-time analysis of video streams," *Sensors*, vol. 21, no. 12, p. 4045, 2021. [Cited on page 20.]

[37] K. Jain, “Scalable distributed architecture for managing large scale camera networks,” Doctoral dissertation, International Institute of Information Technology Hyderabad, 2024. [Cited on page 20.]

[38] Dobslaw and Wechsung, “Latency and responsiveness in interactive systems: A review,” *ACM Computing Surveys*, 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3502324 [Cited on page 22.]

[39] G. et al., “Response time and human performance: A review for interactive systems,” *Human–Computer Interaction*, 2001. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1207/S15327051HCI15234_02 [Cited on page 22.]

[40] Python Software Foundation, “Python programming language – official website,” https://www.python.org/, n.d., accessed: 2025-09-27. [Cited on page 34.]

[41] Pallets Projects, “Flask documentation,” https://flask.palletsprojects.com/en/stable/, n.d., accessed: 2025-09-27. [Cited on page 34.]

[42] n.a., “Watchdog documentation,” https://python-watchdog.readthedocs.io/en/stable/index.html, n.d. [Cited on page 34.]

[43] ——, “watchdog: Python api and shell utilities to monitor file system events,” https://pypi.org/project/watchdog/, n.d. [Cited on page 34.]

[44] ——, “Ffmpeg,” https://ffmpeg.org/, n.d. [Cited on page 38.]

Heuristical Evaluation forms

The forms of the different evaluators from the usability tests.

Heuristic Evaluation Worksheet – Seshat System

Evaluator Name: Hugo Silva Date: 2025-09-17

Task 1: Run Seshat

Description: Prepare all the needed components for Seshat to run and run them.

Suggested Observations	**Notes**
Did you find any problem when running seshat?	No problems were encountered
Were the servers easy to set up?	The instruction file explained everything very well

Issue #	**Heuristic**	**Description**	**Severity**
1			
2			
3			

Task 2: Run your annotator and load it in Seshat

Description: Upload your model to the API, run it and load it in Seshat.

Observation Area	Notes
Was it difficult to understand what is the purpose of the API?	Not at all
Was it difficult to adapt your model to the API?	The adaptation process was easy
Did you manage to get all the features from the API and your model working?	The model I managed to use worked without any problems

Issue #	**Heuristic**	**Description**	**Severity**

1			
2			
3			

Task 3: Play an exam and annotate a frame

Description: Play one of the available exams and annotate one of its frames.

Observation Area	Notes
Was the API intuitive?	Although it could use some enhancements in the usability department, the API was intuitive for a normal software developer
Was it clear when an annotation was made?	It wasn't very clear whenever an annotation was made and if it was successful
Was loading the model and annotating difficult?	The loading and processing was very easy to use

Issue #	**Heuristic**	**Description**	**Severity**
1			
2			
3			

Task 4: Visualise your annotation and remove it using Seshat

Description: Visualise your annotation and remove it using the annotations menu of Seshat.

Observation Area	Notes
Were annotations visible?	They were visible, but not very responsive
Was it visually clear which annotation came from which model?	No
Any visual clutter or confusion?	Even though there wasn't clutter, the lack of

	visual feedback caused confusion

Issue #	Heuristic	Description	Severity
1	1	The system should always keep users informed about what is going on, through clear and timely feedback.	**8**
2			
3			

Task 5: Use a second annotator concurrently and compare annotations

Description: Run two different annotators on the same video and visually compare their outputs.

Observation Area	Notes
Were annotations clear and visually distinct?	Not really
Was it visually clear which annotation came from which model?	Yes, the colors help a lot
Any visual clutter or confusion?	No

Issue #	Heuristic	Description	Severity
1	**1**	The system should always keep users informed about what is going on, through clear and timely feedback.	**7**
2			
3			

Heuristic Evaluation Worksheet – Seshat System

Evaluator Name: Nuno Teixeira Date: 2025-08-21

Task 1: Run Seshat

Description: Prepare all the needed components for Seshat to run and run them.

Suggested Observations	Notes
Did you find any problem when running seshat?	No
Were the servers easy to set up?	Yes

Issue #	Heuristic	Description	Severity
1			
2			
3			

Task 2: Run your annotator and load it in Seshat

Description: Adapt your model to the API by following the API instructions and run it. After that, load the model on the Seshat.

Observation Area	Notes
Was it difficult to understand what the purpose of the API is?	No
Was it difficult to adapt your model to the API?	Not applicable, I used an example model
Did you manage to get all the features from the API and your model working?	Yes

Issue #	Heuristic	Description	Severity
1			
2			
3			

Task 3: Play an exam and annotate a frame

Description: Play one of the available exams to your liking and annotate one of its frames with your loaded model.

Observation Area	Notes

Was the API intuitive?	Everything was intuitive other than the annotation button. I thought the annotation was being made on the RTA button
Was it clear when an annotation was made?	No, you need to check the logs to check if the annotation is made. The annotation marker is only shown when you reload the page, would be cool to have it immediately
Was loading the model and annotating difficult?	No

Issue #	Heuristic	Description	Severity
1	**1**	There was an error in the model (accessing the data folder) and the only way to check if the annotations were being made or not was to check the logs.	3
2			
3			

Task 4: Visualise your annotation and remove it using Seshat

Description: Visualize your annotation and remove it using the annotations menu of Seshat.

Observation Area	Notes
Were annotations visible?	Yes
Was it visually clear which annotation came from which model?	Yes
Any visual clutter or confusion?	No

Issue #	Heuristic	Description	Severity
1	**3**	It would be interesting to have the annotation being shown immediately in the UI, allowing the user to remove them without needing to reload the page	6
2			
3			

Task 5: Use a second annotator concurrently and compare annotations

Description: Run two different annotators on the same video and visually compare their outputs.

Observation Area	Notes
Were annotations clear and visually distinct?	Yes

Was it visually clear which annotation came from which model?	Yes
Any visual clutter or confusion?	No

Issue #	Heuristic	Description	Severity
1			
2			
3			

Heuristic Evaluation Worksheet – Seshat System

Evaluator Name: Rodrigo Emídio Date: 2025-09-17

Task 1: Run Seshat

Description: Prepare all the needed components for Seshat to run and run them.

Suggested Observations	Notes
Did you find any problem when running seshat?	No
Were the servers easy to set up?	Yes

Issue #	Heuristic	Description	Severity
1			
2			
3			

Task 2: Run your annotator and load it in Seshat

Description: Upload your model to the API, run it and load it in Seshat.

Observation Area	Notes
Was it difficult to understand what is the purpose of the API?	A little
Was it difficult to adapt your model to the API?	No
Did you manage to get all the features from the API and your model working?	Yes

Issue #	Heuristic	Description	Severity
1	**8**	Interface should be simplified to understand and control better	2
2			
3			

Task 3: Play an exam and annotate a frame

Description: Play one of the available exams and annotate one of its frames.

Observation Area	Notes
Was the API intuitive?	More or less
Was it clear when an annotation was made?	Yes
Was loading the model and annotating difficult?	No

Issue #	Heuristic	Description	Severity
1			
2			
3			

Task 4: Visualise your annotation and remove it using Seshat

Description: Visualise your annotation and remove it using the annotations menu of Seshat.

Observation Area	Notes
Were annotations visible?	Yes
Was it visually clear which annotation came from which model?	Yes, but the visualization can be improved
Any visual clutter or confusion?	No

Issue #	Heuristic	Description	Severity
1			
2			
3			

Task 5: Use a second annotator concurrently and compare annotations

Description: Run two different annotators on the same video and visually compare their outputs.

Observation Area	Notes
Were annotations clear and visually distinct?	Yes
Was it visually clear which annotation came from which model?	More or less
Any visual clutter or confusion?	No

Issue #	Heuristic	Description	Severity

1	**8**	The annotations should have maybe a different background (for classification) and be the core of the attention from the user, simplify interface	8
2			
3			

Heuristic Evaluation Worksheet – Seshat System

Evaluator Name: Tomás Ferreira Date: 2025-09-17

Task 1: Run Seshat

Description: Prepare all the needed components for Seshat to run and run them.

Suggested Observations	**Notes**
Did you find any problem when running seshat?	No, it ran smoothly
Were the servers easy to set up?	Yes

Issue #	**Heuristic**	**Description**	**Severity**
1	**10**	There was a small typo on the README file that wrote a command as ´systemcyl´ instead of ´systemctl´, with an easily understandable fix (as systemctl was written above in another command)	4
2			
3			

Task 2: Run your annotator and load it in Seshat

Description: Upload your model to the API, run it and load it in Seshat.

Observation Area	Notes
Was it difficult to understand what is the purpose of the API?	No
Was it difficult to adapt your model to the API?	No, the code was pretty straightforward
Did you manage to get all the features from the API and your model working?	Yes

Issue #	Heuristic	Description	Severity
1			
2			
3			

Task 3: Play an exam and annotate a frame

Description: Play one of the available exams and annotate one of its frames.

Observation Area	Notes
Was the API intuitive?	Yes
Was it clear when an annotation was made?	Pretty clear
Was loading the model and annotating difficult?	No

Issue #	Heuristic	Description	Severity
1			
2			
3			

Task 4: Visualise your annotation and remove it using Seshat

Description: Visualise your annotation and remove it using the annotations menu of Seshat.

Observation Area	Notes
Were annotations visible?	Yes
Was it visually clear which annotation came from which model?	Yes
Any visual clutter or confusion?	Not really, very clear

Issue #	Heuristic	Description	Severity
1			
2			
3			

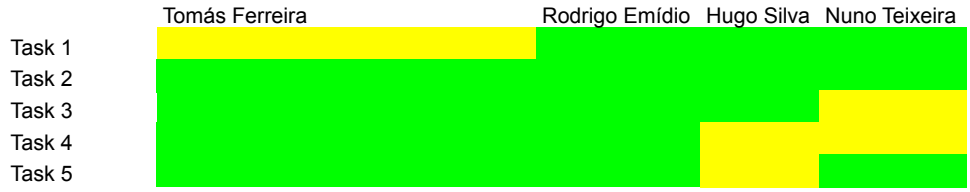
Task 5: Use a second annotator concurrently and compare annotations

Description: Run two different annotators on the same video and visually compare their outputs.

Observation Area	Notes
Were annotations clear and visually distinct?	Yes
Was it visually clear which annotation came from which model?	Yes
Any visual clutter or confusion?	Not really, no

Issue #	Heuristic	Description	Severity
1			
2			
3			

Simplified Cognitive walkthrough



Fill the column B with the following colours

- User completed the task successfully
- User completed the task with difficulty
- User did not complete the task