UNIVERSIDADE D
COIMBRA

Afonso Martingo da Costa Carvalho

REAL-TIME AI MODELS RUNNING ON LOW-POWER EMBEDDED GPUS FOR ARTHROSCOPIC SURGERY

Dissertation in the context of the Master in Electrical and Computer Engineering, specialization in Computers, advised by Prof. Doctor Gabriel Falcão Paiva Fernandes, Doctor Michel Almeida Antunes and Prof. Doctor João Pedro de Almeida Barreto and submitted to the Department of Electrical and Computer Engineering of Faculty of Sciences and Technology of the University of Coimbra.

September 2024

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Real-time AI Models Running on Low-power Embedded GPUs for Arthroscopic Surgery

Afonso Martingo da Costa Carvalho

Coimbra, September 2024

Real-time AI Models Running on Low-power Embedded GPUs for Arthroscopic Surgery

Supervisor:

Professor Doctor Gabriel Falcão Paiva Fernandes

Co-Supervisors:

Doctor Michel Antunes

Professor Doctor João Barreto

Jury:

Professor Doctor Jorge Miguel Sá Silva

Professor Doctor Jorge Nuno de Almeida e Sousa Almada Lobo

Professor Doctor Gabriel Falcão Paiva Fernandes

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra, September 2024

Agradecimentos

Após 6 anos nesta bonita cidade de Coimbra, é com enorme satisfação que dou por concluída esta etapa. Este percurso foi marcado por desafios exigentes, mas também por muitas alegrias e aprendizagens que levarei comigo para a vida.

Quero expressar a minha mais profunda gratidão à minha família, em especial aos meus pais, Vítor e Maria José, e aos meus avós, Arménio, Alcinda e Maria José. Com o seu amor incondicional e apoio constante, tornaram esta conquista possível. À minha irmã Maria, agradeço pelo carinho e companheirismo inabalável ao longo destes anos. Aos meus tios, o meu reconhecimento por toda a ajuda e suporte. Um agradecimento especial à Soraia, por ter estado ao meu lado nesta jornada, enfrentando os altos e baixos com amor e paciência.

Ao meu orientador, Professor Doutor Gabriel Falcão Paiva Fernandes, o meu sincero agradecimento pela orientação e valiosos ensinamentos durante este trabalho. Agradeço igualmente aos meus co-orientadores, Doutor Michel Antunes e Professor Doutor João Barreto e a toda equipa da Perceive3D pela confiança e apoio ao longo deste projeto.

Aos meus amigos, um agradecimento do fundo do coração. A amizade e o apoio que recebi de vocês foram essenciais para superar os momentos difíceis e celebrar as conquistas. Ao grupo de amigos que fiz ao longo destes anos, especialmente os Eletrões, sou grato por cada momento juntos. Um obrigado especial ao Bernardo Leite pelos conselhos sábios e pela amizade inabalável, ao André Ribeiro pela lealdade e irmandade, ao meu afilhado Luís Ventura pela amizade genuína e apoio constante e ao Diogo Gouveia pelo seu companheirismo nesta jornada.

Por fim, agradeço a todos os meus colegas de curso que foram companhia nas madrugadas de estudo e foram uma força extra ao longo deste caminho. A todos, o meu sincero agradecimento.

Abstract

With the increasing need to ensure quality healthcare services conforming to the highest safety and efficiency standards, the medical field has increasingly relied on technology to overcome the challenges posed by the complexity of human anatomy and medical conditions. This dissertation focuses on implementing real-time Artificial Intelligence (AI) algorithms for surgical navigation in arthroscopy.

Arthroscopy is a minimally invasive surgical procedure for treating joint injuries. It offers advantages, including smaller incisions, reduced soft tissue disruption, reduced postoperative pain, and improved visualization of intraarticular anatomy. However, this type of intervention presents challenges to surgeons due to indirect visualization and limited maneuverability within the joint.

Perceive3D is developing a Video-Based Surgical Navigation (VBSN) system for ACL reconstruction, which offers several clinical benefits. This system assists surgeons in navigating the knee joint arthroscopically and performing ACL tunnel placement with high precision. While the initial VSBN product is currently in the development phase, Perceive3D is actively researching and advancing new AI algorithms to enhance the system's capabilities. These algorithms, presently at the prototype stage in a research setting, are designed to improve surgical outcomes further. Perceive3D aims to efficiently deploy these innovative AI algorithms in real surgical environments, where low latency is critical.

This thesis aims to design and test an infrastructure that enables the real-time deployment of AI algorithms for VSBN. The target system consists of a video capture device and an NVIDIA platform, utilizing the NVIDIA Holoscan framework to streamline the deployment of AI models. Multiple AI models have been implemented and optimized within this framework, achieving the same level of accuracy as those developed in the research environment, while being significantly more efficient.

The system was experimentally evaluated to assess the performance of the deployed

models, and the results were promising. When using the AJA board to run the worst performance model, the system achieved a latency of approximately 50 to 83 ms and a frame rate of 20 fps, while the best-performing model reached a latency of 13 ms and a frame rate of 60 fps. These results highlight the system's potential for further evaluation in clinical settings.

Keywords: Artificial Inteligence; Surgical Navigation for Arthroscopy; Edge Computing.

Resumo

Com a necessidade crescente de garantir serviços de saúde de qualidade, em conformidade com os mais elevados padrões de segurança e eficiência, a área médica tem confiado cada vez mais na tecnologia para superar os desafios colocados pela complexidade da anatomia humana e das condições médicas. Esta dissertação foca-se na implementação de algoritmos de Inteligência Artificial (IA) em tempo real para navegação cirúrgica em artroscopia.

A artroscopia é um procedimento cirúrgico minimamente invasivo para tratar lesões articulares e oferece vantagens, incluindo incisões menores, redução da ruptura dos tecidos moles, redução da dor pós-operatória e melhor visualização da anatomia intra-articular. No entanto, este tipo de intervenção apresenta desafios aos cirurgiões devido à visualização indireta e à manobrabilidade limitada dentro da articulação.

A Perceive3D está a desenvolver um sistema de Navegação Cirúrgica Baseada em Vídeo (VBSN) para a reconstrução do Ligamento Cruzado Anterior (LCA), que oferece vários benefícios clínicos. Este sistema auxilia os cirurgiões na navegação da artroscopia da articulação do joelho e na realização do posicionamento do túnel de LCA com elevada precisão. Embora o produto inicial VSBN esteja atualmente na fase de desenvolvimento, a Perceive3D está a avançar ativamente no desenvolvimento de novos algoritmos de IA para melhorar as capacidades do sistema. Estes algoritmos, actualmente na fase de protótipo num ambiente de investigação, são concebidos para melhorar ainda mais os resultados cirúrgicos. A Perceive3D visa implementar eficientemente estes algoritmos inovadores de IA em ambientes cirúrgicos reais, onde a baixa latência é crítica.

Esta tese visa projetar e testar uma infraestrutura que permita a implementação em tempo real de algoritmos de IA para o VSBN. O sistema desenvolvido consiste num dispositivo de captura de vídeo e numa plataforma NVIDIA, utilizando a *framework* NVIDIA Holoscan para suportar a implementação dos modelos de IA. Vários modelos de IA foram implementados e otimizados nesta estrutura, alcançando o mesmo nível de precisão que os desenvolvidos no ambiente de investigação, sendo significativamente mais eficientes.

O sistema foi avaliado experimentalmente para validar o desempenho dos modelos implementados e os resultados foram promissores. Ao usar a placa AJA para executar o modelo de pior desempenho, o sistema alcançou uma latência de aproximadamente 50 a 83 ms e uma taxa de 15 fps, o modelo de melhor desempenho atingiu uma latência de 13 ms e uma taxa de 60 fps. Estes resultados demonstram o potencial do sistema para futura avaliação em ambientes clínicos.

Palavras-Chave: Inteligência Artificial, Navegação Cirúrgica para Artroscopia, Computação de Borda.

"Knowledge has always been power, but in the age of AI, it becomes the power to heal, transform, and transcend the limits of traditional medicine."

— Inspired by Francis Bacon

Contents

List of Acronyms

ACL	Anterior Cruciate Ligament
AI	Artificial Intelligence
AR	Augmented Reality
CADDIE	Computer-Aided Detection and Diagnosis in Endoscopy
CLAHE	Contrast Limited Adaptive Histogram Equalization
CNNs	Convolutional Neural Networks
CPU	Central Processing Unit
CSC	Color Space Converter
CUDA	Compute Unified Device Architecture
DICOM	Digital Imaging and Communications in Medicine
DL	Deep Learning
dGPU	Dedicated Graphics Processing Unit
FAI	Femoral Acetabular Impingement
FP32	32-bits Floating point
FP16	16-bits Floating-point
FPGA	Field-Programmable Gate Array
FPS	Frames Per Second
FDA	Food and Drug Administration
GI	Gastrointestinal

GPU	Graphics Processing Unit
GT	Ground Truth
GUI	Graphical User Interface
GXF	Graph Execution Framework
HDMI	High-Definition Multimedia Interface
HoloHub	Holoscan Hub
HPC	High Performance Computing
ICC	Intraclass Correlation Coefficient
IFSES	International Federation of Societies of Endoscopic Surgeons
IHU	Institute Surgery Guided Par L'image (Institut Hospitalo-Universitaire)
iGPU	Integrated Graphics Processing Unit
ISR-UC	Institute of Systems and Robotics - University of Coimbra
IT-UC	Institute of Telecommunications - University of Coimbra
LC	laparoscopic cholecystectomy
LGG	Low-Grade Gliomas
LA	Laser Augmentation
MAE	Mean Absolute Error
ML	Machine Learning
MPS	Multi-Process Service
NCHW	Batch size, Number of Channels, Height, Width
NHWC	Batch size, Height, Width, Number of Channels
NLA	No Laser Augmentation
ONNX	Open Neural Network Exchange

OSS	Open Source Software
PCA	Principal Component Analysis
PACS	Picture Archiving and Communication System
PCIe	Peripheral Component Interconnect Express
RDMA	Remote Direct Memory Access
RGB	Red, Green, Blue
ROI	Region of Interest
SAD	Sum of Absolute Differences
SDI	Serial Digital Interface
SDK	Software Development Kit
SL	Structured Light
SWaP	Size, Weight, and Power
UCSF	University of California, San Francisco
VBSN	Visual-Based Surgical Navigation
WM	World Marker
YUV	Luminance, Chrominance

List of Figures

List of Tables

1 Introduction

This section introduces the research's context and motivation, presenting the work's objectives and contributions. The objective of this work is to enable the deployment of AI algorithms developed by Perceive3D for Video-Based Surgical Navigation (VBSN) in real clinical settings. Initially designed and tested in a research environment, these algorithms need to operate in real-time with minimal latency to be viable in surgical applications. This thesis focuses on implementing these AI algorithms within an infrastructure that ensures real-time performance. Through a series of experiments, it will be demonstrated that the algorithms maintain the same accuracy as the research models while operating in real-time. The work will be developed in the context of Perceive3D, a company that develops solutions for surgical navigation in arthroscopy using AI models. The research will be carried out in collaboration with the Institute of Telecommunications (IT-UC) of the University of Coimbra.

1.1 Context and Motivation

AI has been used in various medical fields, such as medical imaging, diagnostics, drug discovery, personalized medicine, and patient monitoring [1] [2]. Recently, there has been a significant increase in the use of AI in medical surgery equipment to improve patient care and outcomes. Perceive3D is a company that develops solutions to facilitate surgical navigation in arthroscopy. It uses computer vision and AI techniques to support the surgeon in surgical procedures, improving the procedure's accuracy and efficiency.

1.1.1 Arthroscopy and ACL reconstruction

Arthroscopy is an advanced medical procedure for treating joint injuries minimally invasively. This work focuses on knee arthroscopy and Anterior Cruciate Ligament (ACL) reconstruction, a common procedure in orthopedic surgery. Arthroscopic knee surgery for ACL injuries

is the most common orthopedic procedure in countries with available data [3] [4] and on a global scale it is performed more than two million times each year [5, 6, 7, 8, 4].

ACL reconstruction is a surgical procedure that aims to restore the stability of the knee joint after an ACL injury. The ACL is a ligament that connects the femur to the tibia and it is essential for the stability of the knee joint. ACL injuries are common in sports involving sudden stops, direction changes, and jumping, such as soccer, basketball, and skiing. ACL injuries can also occur during everyday activities, such as stepping off a curb or missing a step [9]. ACL reconstruction is a surgical procedure that involves replacing the torn ACL with a graft, usually taken from the patient's hamstring or patellar tendon [10].During this surgical process, small incisions are made in the knee region, through which specialized surgical instruments and a camera equipped with a tubular lens - the arthroscope - are introduced to visualize the joint's interior [11]. The arthroscope also contains a fiber optic guide connected to an external light source to illuminate the interior of the joint and a supervision channel through which a fluid is pumped to: (i) distend the intra-articular space and (ii) prevent blood and other debris from obstructing the doctor's vision (figure 1.1). This camera transmits images in real-time to an external monitor, allowing the doctor to have a specific view of the inside of the knee during an intervention. Arthroscopy has increased the understanding of intra-articular pathology and enables surgery to be performed with lower morbidity and less pain than open surgery [12]. Compared to the open approach, arthroscopic surgery has substantial benefits for patients, including faster recovery and less postoperative pain due to the reduced size of the incisions, and also less hospital stay (high costs).

However, indirect visualization and limited maneuverability within the joint make surgery difficult, where novice doctors face a steep learning curve [13], and expert doctors can also make mistakes with significant clinical implications [14].

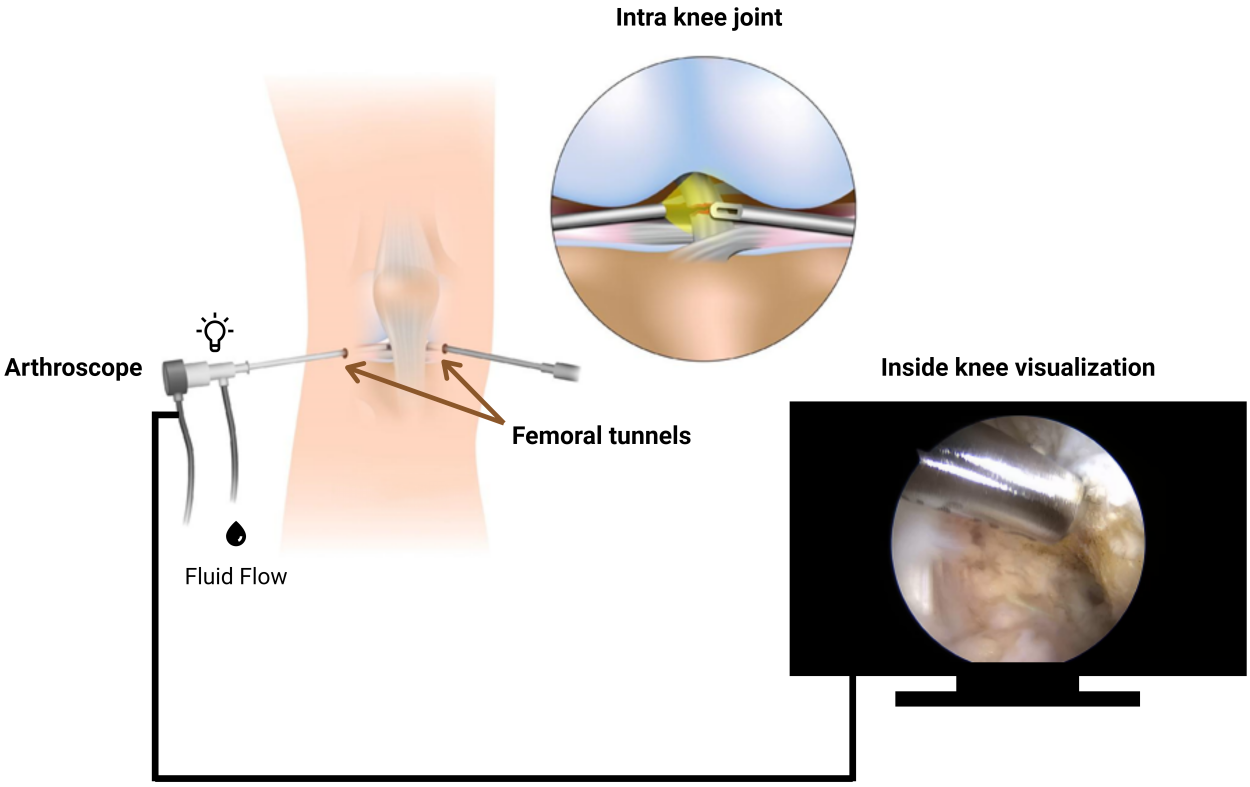

Figure 1.1: Arthroscopy procedure example where the arthroscope, with a fluid pump and a light source, is inserted into the knee joint through femoral tunnels and connected to a monitor for further visualization.

1.1.2 Video-Based Surgical Navigation (VBSN)

Video-Based Surgical Navigation (VBSN), jointly developed by Perceive3D and ISR-UC, has proven to have clinical benefits in arthroscopy procedures [10]. This method uses small, planar visual markers attached to the anatomical region and surgical tools. These markers enable the accurate estimation of 3D poses. A touch probe also instrumented with a visual marker, is used to reconstruct the 3D contours of the bone surface. This data aligns pre-operative models with patient anatomy, providing real-time guidance during surgery [10].

1.1.3 Structured Light for touchless 3D registration in VBSN

To enhance video-based surgical navigation (VSBN), Perceive3D developed a structured light (SL) system for touchless 3D registration, eliminating the manual intervention and improving overall efficiency. The SL system projects a light pattern onto the bone, which is captured by a camera, and the 3D surface is reconstructed. An AI model is then used for segmentation, improving accuracy and reducing errors in the registration process.

1.1.4 AI models for surgical navigation

Perceive3D is developing several AI models for further improving VSBN, and they are implemented in a research setting. These models include a femur segmentation model, an inside/outside knee classification model, and a lens mark visibility model.

- **Femur Segmentation Model** - This model is used in the SL system for touchless 3D registration and identifies the femur's interest structure, eliminating outliers and improving the registration model's accuracy. This model is critical for the registration process.
- **Inside/Outside Knee Classification Model** - This model classifies if the camera is inside or outside the knee, which is essential for control of the arthroscope's components, such as the light source and irrigation pump.
- **Lens Mark Visibility Model** - This model determines the visibility of the lens mark, which is essential for the camera calibration and orientation estimation.

These models are currently under experimental evaluation and have not been deployed in real-time processing.

1.2 Objectives

This thesis aims to develop and implement in real-time the proposed AI models on the edge that will allow advantages to be obtained during the surgical procedure. It will focus on segmentation of the femur, inside/outside knee classification model, and lens mark visibility model.

The goal is to make a comprehensive integration of NVIDIA hardware and software tools, namely a graphics processing unit Jetson AGX Orin Developer Kit, an AJA Corvid 44 12G BNC capture system, a 12G BiDirectional SDI/HDMI Micro Converter, and the recent framework NVIDIA Holoscan, which will be needed to optimize model implementation, ensuring a balance between computational efficiency and accuracy in a complete edge computing system.

The work developed in this dissertation also demonstrates the importance of this approach for improving the accessibility and efficiency of AI applications in resource-limited environments while also pointing out possible directions for future research and developments in this field.

1.3 Contributions

Motivated by the challenges of the work previously developed at Perceive3D, on which this research is based, and by the need to improve the efficiency of surgical procedures on clinical settings, this dissertation aims to contribute to developing real-time AI algorithms for surgical navigation in arthroscopy. The main contributions of this research are:

1. Integration and consolidation of an advanced edge computing system incorporating NVIDIA Orin, AJA video capture card, and NVIDIA Holoscan for enhanced performance and seamless data processing.

2. Deployment of femur segmentation model running in real-time in an edge computing system,required for identifying the images regions to be used for 3D registration with pre-operative model.

3. Deployment of the inside/outside knee classification model running in real-time in an edge computing system, needed to turn off light source (avoid dazzle or burning medical personal) and turn off irrigation pump (avoid flood surgical environment).

4. Deployment of lens mark visibility model running in real-time in an edge system, required for continuous and automatic camera calibration.

5. Improvement and optimization of the models to run with greater fluidity and frame rate.

6. Explore AJA capabilities to perform keyer functionality for AR overlay and backup video.

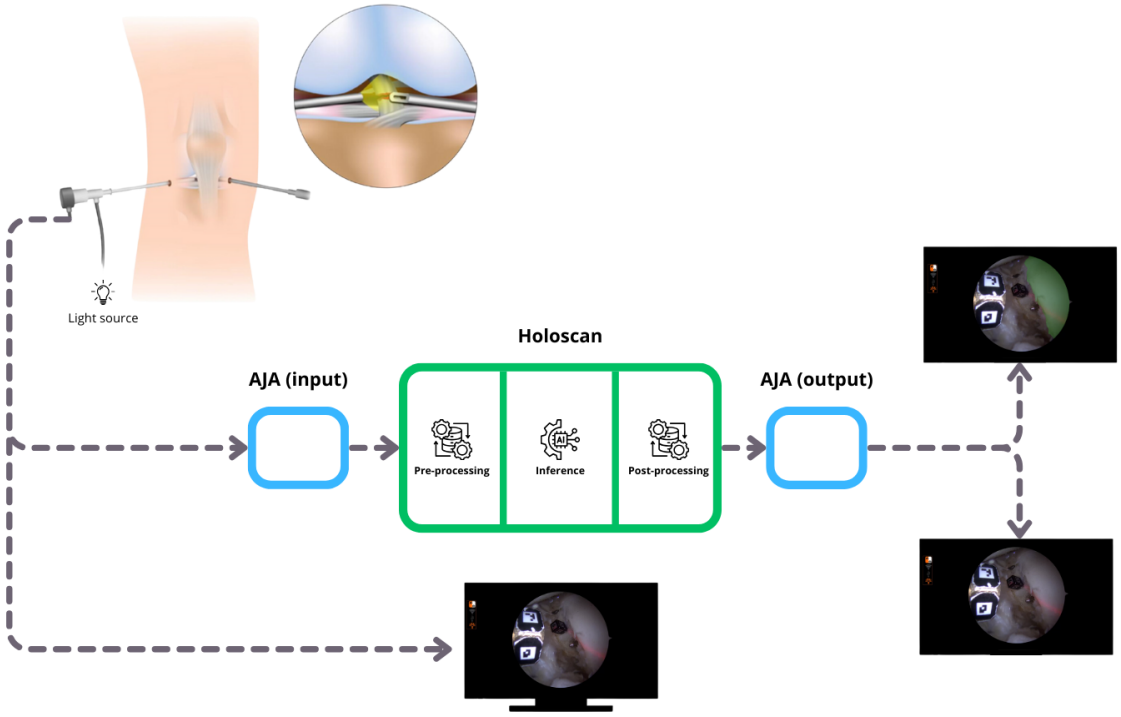

Figure 1.2: Edge computing system architecture, AJA Corvid 44 12G BNC capture card capturing the arthroscopic camera source, Holoscan as the framework for video processing, AI models deployment and visualization in the external screens.

1.4 Dissertation Outline

This dissertation is organized into several chapters, each addressing a specific aspect of the research:

- **Chapter 1: Introduction** - Introduces the work's context and motivation and presents the research's objectives and contributions.
- **Chapter 2: Background and Related Work** - Provides an overview of the state-of-the-art in AI models for medical applications and the technologies and tools used in this research.
- **Chapter 3: Edge Computing System** - Describes the edge computing system used in this research, including an overview of the architecture, the system components and the keyer for AR overlay.
- **Chapter 4: Deployment of Deep Learning Models** - Presents the deployment of Deep Learning models in the edge computing system, including an overview of the AI

models used to improve surgical navigation and their implementation in the Holoscan framework.

- **Chapter 5: Experimental Validation** - Describes the system's experimental validation, including the evaluation methodology and the results of the tests carried out to validate it.
- **Chapter 6: Conclusions and Future Work** - Summarizes the main findings of the research and outlines possible directions for future research.
- **Chapter 7: Bibliography** - Provides a bibliography of the references used in this research.

2 Background And Related Work

This chapter provides an overview of the research area, including the importance of medical image segmentation, the role of Deep Learning (DL) in medical applications, and the challenges of surgical navigation in arthroscopy. The chapter also introduces the Holoscan framework, a software and hardware platform that enables developers to build and deploy AI applications for edge computing systems.

2.1 Background on Medical Imaging

Medicine follows a trend of increasing innovation and research alongside technological evolution. Technology overcomes some difficulties encountered due to the complexity of human anatomy and medical conditions. It has revolutionized diagnoses, treatments, and patient monitoring, introducing more sophisticated equipment, advances in artificial intelligence, and telemedicine [1] [2].

ML and DL with Neural Networks, in particular, have recently shown great promise in achieving human- or near-human–level performance in a variety of highly complex perceptual tasks that were traditionally challenging for machines to perform, including image classification [15]. Medical image segmentation is an active area of research, and the introduction of DL and Convolutional Neural Networks (CNNs) has made significant advances. It is crucial for medical diagnosis as it helps reduce subjectivity in interpreting medical images. Manual visualization methods can be tedious, time-consuming, and subject to error on the part of the clinician [16]. With the emergence of the DL paradigm and recent advances in computational power, namely in the form of thousands of processors capable of performing parallel computing, new intelligent diagnostics based on computer vision are being developed. Patient safety is of utmost importance in the surgical environment, and implementing safe practices can help reduce complications and improve patient outcomes [17].

2.1.1 Deep learning

Deep Learning is a subset of ML that uses neural networks with many layers to model and make sense of data. DL has been used in the medical field to detect diseases, predict patient outcomes, and assist in surgical procedures [18]. DL models have been developed to assist radiologists in detecting and diagnosing diseases from medical images, such as X-rays, Computed tomography (CT) scans, and Magnetic resonance imaging (MRIs) [19].

One significant difficulty in the medical field is deploying DL models in edge computing systems. These models are usually trained in powerful servers and deployed in edge devices with limited computational resources. This challenging task requires careful optimization to ensure that the models can run efficiently on the edge devices [20]. In this work, we will use a framework called Holoscan [21].

2.1.2 Holoscan

NVIDIA Holoscan is a sensor processing platform that optimizes developing and implementing AI and high-performance computing (HPC) applications for real-time scenarios. This framework provides the necessary software for creating AI applications and deploying sensor processing capabilities from the edge to the cloud. From medical procedures to satellite operations, Holoscan empowers companies to explore new possibilities, expedite time to market, and reduce costs [21].

NVIDIA's Clara platform stands as a comprehensive suite of AI and accelerated computing tools crafted to revolutionize healthcare and life sciences. Both NVIDIA MONAI and NVIDIA Holoscan are part of NVIDIA's Clara platform, but they fulfill distinct roles within the healthcare and medical imaging fields [22]:

NVIDIA MONAI functions as an open-source framework tailored to accelerate the development of AI models in medical imaging. It offers a collection of enterprise-grade containers, AI models, and cloud APIs for cloud deployment, as well as local deployment options. MONAI is oriented explicitly toward medical imaging and aims to aid radiology departments and other medical imaging applications by enabling AI analysis of images from CT, MRI, and various other imaging modalities [23].

In contrast, NVIDIA Holoscan is a scalable, software-defined AI computing platform designed to process streaming data at the clinical edge. It enables real-time AI inference from sensors, a critical capability for applications such as robot-assisted surgery and other real-time medical procedures. Holoscan is deployed in operating rooms and other clinical

settings where immediate AI-driven insights are essential for patient-specific decisions. These are the primary features of the Holoscan platform accordingly to NVIDIA [24]:

- **Sensor Processing:** The Camera Serial Interface (CSI) protocol and front-end sensors allow video capture, ultrasound research, data acquisition, and connection to legacy medical devices.
- **Low Latency:** Holoscan's SDK provides a data transfer latency tool to measure complete, end-to-end latency for video processing applications.
- **Reference AI Pipelines:** Allows the creation of AI reference pipelines for radar, high-energy light sources, endoscopy, ultrasound, and other streaming video applications.

Use Cases

- **Medical Devices:** NVIDIA Holoscan for Medical Devices delivers the accelerated, full-stack infrastructure required for scalable, software-defined, and real-time processing of streaming medical data at the clinical edge. Holoscan offers the possibility to build medical devices that can take AI applications directly to the operating room, processing streaming sensor data for AI inference that helps clinical teams make patient-specific decisions and recommendations [21].
- **HPC Edge:** NVIDIA Holoscan was also developed to be used for HPC at the edge, so it is a universal computation imaging platform, purpose-built for high performance while meeting size, weight, and power (SWaP) constraints at the edge. It delivers a flexible software stack across a common high-performance hardware platform to accelerate data analysis and visualization workflows at the edge [21].

Example Application

Holoscan provides some example applications that demonstrate how to use the framework. One example is the "Ultrasound bone scoliosis segmentation", which demonstrates how to run a DL model on the NVIDIA Jetson AGX Orin Developer Kit using the Holoscan SDK and can be found at this Github repository [25]. Fig 2.1 shows the application that uses a U-Net model to segment bone scoliosis in ultrasound images.

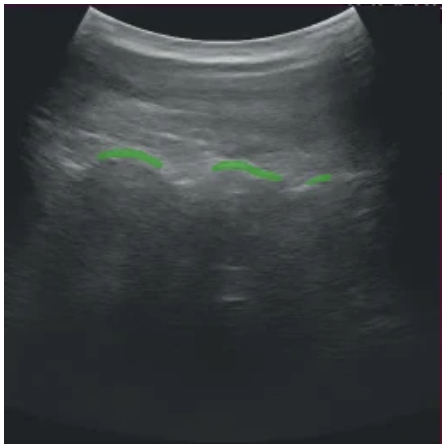

Figure 2.1: Ultrasound bone scoliosis segmentation visualization. Green overlay represents the scoliosis segmentation mask. Image source [25].

2.1.3 Surgical navigation in arthroscopy

Arthroscopy originated in the 19th century and is considered one of the greatest advances in orthopedics [26]. Evolving from the initial use of the cystoscope on cadaver knees to the development of sophisticated equipment, arthroscopy is now an essential component of modern orthopedic care [27].

Video-Based Surgical Navigation

Arthroscopic surgery poses significant challenges due to restricted visibility of the joint and limited maneuverability of instruments. These challenges underscore the importance of video-based surgical navigation (VBSN), which has demonstrated substantial clinical benefits by enhancing precision and reducing errors during arthroscopic procedures [10].

This dissertation builds upon the pioneering work carried out at Perceive3D, which has made significant strides in this area by developing solutions to facilitate surgical navigation in arthroscopy [10] [28]. They were among the first to create a guidance system based exclusively on intraoperative video guiding tunnel drilling during ACL reconstruction. This system utilizes small, easily recognizable visual markers attached to the bone and surgical tools to estimate their relative positions accurately (see Fig 2.2) [10].

The method developed by the Perceive3D team, led by Raposo et al. [10], involves the surgeon performing a random walk on the bone surface using a touch-probe instrumented with visual markers. This approach enables the reconstruction of 3D contours of the bone, which are then used to register a pre-operative MRI with the patient's anatomy, providing real-time guidance during the procedure.

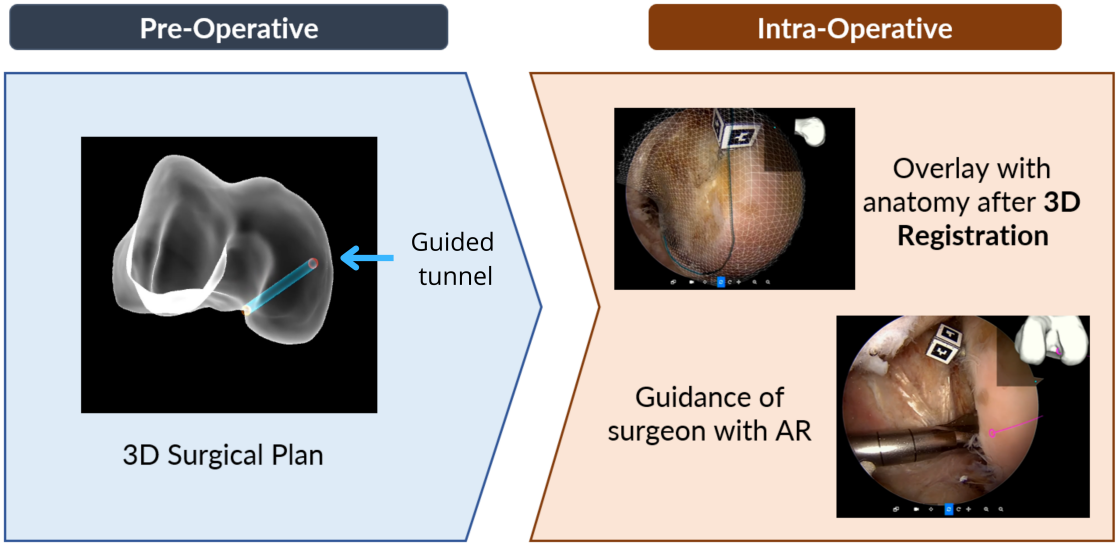

Figure 2.2: VBSN main steps: Pre-op, where the medical team plans the tunnel, and Intra-op, where the surgeon rigidly attaches a marker to the anatomy, serving as a reference frame to estimate the rigid transformation that better aligns the pre-operative 3D model with the anatomy. In this way, the system enables the overlay of the surgical plan onto the actual anatomical structure. (Courtesy of Perceive3D)

Structured Light For Touchless 3D Registration

VBSN relies on time-consuming and challenging scanning of the bone surface using a touch instrument to record intra-operative data. To address this limitation, Baptista et al. [28] introduced a novel approach using structured light for touchless 3D registration. This method replaces the touch probe with an off-the-shelf laser scanner, enabling more efficient and non-invasive registration of the bone surface [28]. This section will delve into the details of this innovative system, highlighting its key features and advantages over traditional touch-based methods.

This system uses a standard arthroscope and a light projector with visual markers for real-time extrinsic calibration. This innovation addresses the limitations of the previous touch-based method by eliminating the risk of bone or cartilage damage and increasing the surgeon's access to the area. The use of structured light significantly improves the efficiency and accuracy of the registration process, particularly in complex procedures like Femoral Acetabular Impingement (FAI) surgeries [28].

The extrinsic calibration is achieved by attaching visual markers to the light projector, which allows the arthroscope and the laser beam to move independently through separate portals. This calibration ensures that the 3D pose of both the camera relative to the bone

(World Marker, WM) and the light projector can be accurately determined at each time instant.

To detect the laser beam in the arthroscopic images, the system computes the difference between the green color channel corresponding to the laser-illuminated regions and the grayscale representation of the image. The structured light plane is calibrated with the projector, and by projecting a contour onto the anatomical surface and detecting it in the arthroscopic video, the 3D coordinates of the laser points are determined through triangulation [28]. These points are then transformed into the WM coordinate system, allowing progressive surface reconstruction (see Fig 2.3).

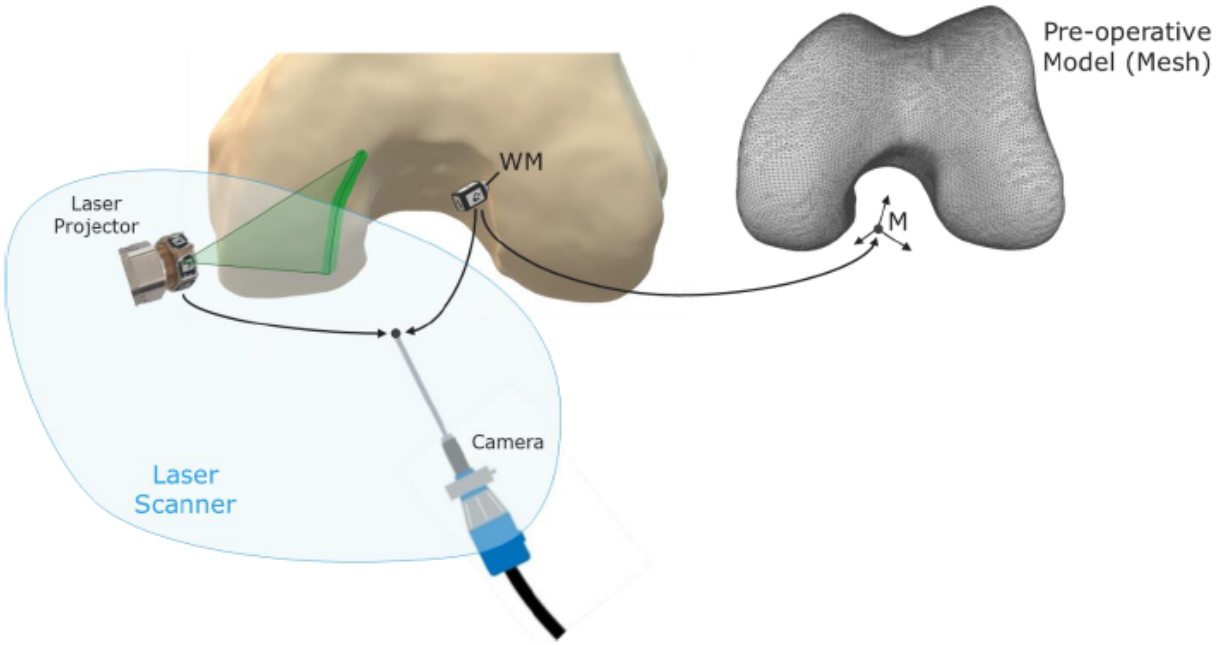

Figure 2.3: Laser detection system in arthroscopic images. The arthroscopic camera estimate the poses relatively to the WM attached to the bone and to the Laser projector through the visual markers attached. With this relation a 3D Mesh reconstruction is done that have a reference relatively to the WM. (Courtesy of Perceive3D)

A U-Net convolutional neural network automatically segments the arthroscopic images, identifying regions of interest corresponding to surfaces in the pre-operative anatomical model [29]. The segmentation model is trained using data augmentation techniques, including synthetic laser projections, to enhance the model's robustness. U-net convolutional neural networks are commonly used for image segmentation tasks, as they can capture spatial information and learn complex patterns in the data. They comprise an encoder-decoder architecture with skip connections that help preserve spatial information during the down-

sampling and upsampling process [30].

Synthetic laser projections are generated by back-projecting contours from the 3D model onto the images, simulating the dispersion of the laser light and improving the model's ability to handle real-world scenarios. Fig 2.4 shows the result of the automatic segmentation model trained without images having laser projection (NLA model) and the result of the automatic segmentation model trained with laser projection augmentation (LA model).

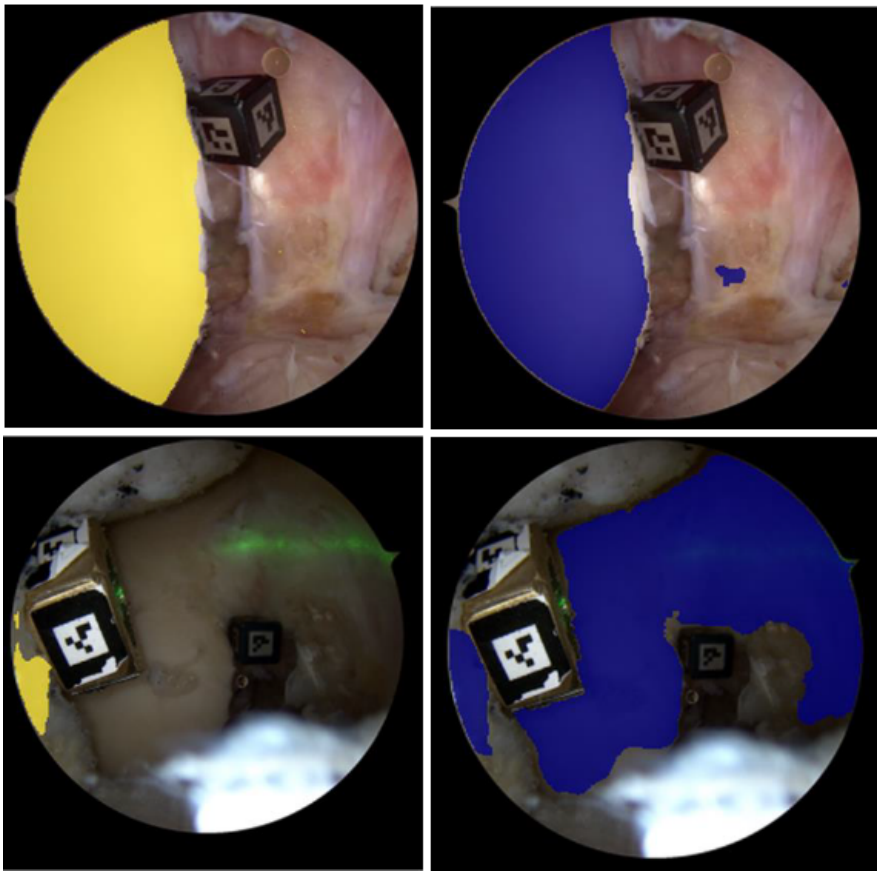

Figure 2.4: The left column (yellow) shows the result of the automatic segmentation model trained without images having laser projection (NLA model). The right column (blue) shows the result of the automatic segmentation model trained with laser projection augmentation (LA model). (Courtesy of Perceive3D)

In summary, integrating structured light for touchless 3D registration represents a significant advancement in VBSN, offering a more efficient and safer alternative to traditional touch-based methods. This progress shows how technology in surgical navigation is constantly improving, starting with the work of Raposo et al. [10] and further developed by Baptista et al. [28]. The two intraoperative registration methods are represented in Fig 2.5.

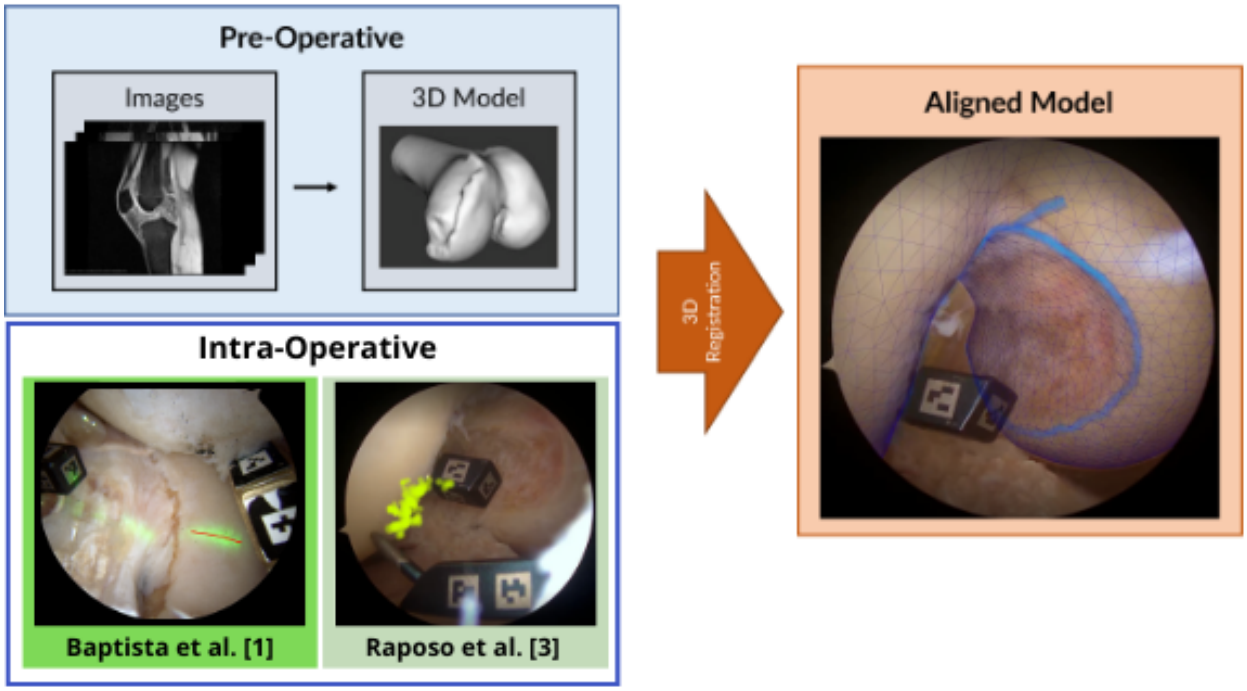

Figure 2.5: Two different intraoperative registration methods. At the bottom right, the VBSN touch registration (by Raposo et al. [10]). At the bottom left, structured light for touchless 3D registration in VSBN (by Baptista et al. [28]). (Courtesy of Baptista et al. [28])

2.2 Related Work

The related work section provides an overview of the current state-of-the-art in AI applications for medical imaging using Holoscan. It includes a summary of this area's most relevant studies and projects, highlighting each work's essential findings and contributions. As Holoscan is a relatively new framework, only a few studies were conducted using this platform at the beginning of this work. At the end of this work, we can verify that the number of studies and applications using Holoscan has increased over the last year. Therefore, the related work section focuses on the most recent and impactful studies conducted in this area.

2.2.1 Previous research work using Holoscan

The following studies were selected based on their relevance to the research area and their use of the Holoscan framework.

The Role Of AI In Accelerating Medical Breakthroughs

To prove the potential of AI in revolutionizing clinical trials, Chopra et al. [31] conducted a study that explores how AI can transform the clinical trial process by addressing data-related limitations, reducing manual efforts, and enhancing patient monitoring. The authors highlight that AI-driven tools can automate data generation and management throughout the trial lifecycle, improve recruitment and adherence, and streamline analysis [31]. AI can intelligently interpret data, fill out analysis reports, and save significant time and cost, leading to faster drug development and improved efficiency. Specific applications of AI in clinical trials include biosimulation, early disease diagnosis, and patient recruitment using data mining [31]. The paper emphasizes the potential of AI to revolutionize clinical trials by improving accuracy, reducing rework, and accelerating medical research, confirming that the Clara Holoscan platform offers an all-in-one, medical-grade standard design and long-term software support to speed up innovation in the medical device market [31].

Generalized System for In-Clinic Deployment and Validation of ML Models in Radiology

One key study using the Clara framework was conducted at the University of San Francisco, California (UCSF), through the Center for Intelligent Imaging. This study successfully implemented a generalized system, with MONAI, for deploying and validating ML models in clinical radiology workflows, as detailed by Hawkins et al. [32]. The system was designed to address the complexities of integrating ML models within clinical environments, focusing on infrastructure and usability challenges.

- **System Architecture:** The system developed at UCSF is built on open-source software (OSS) and incorporates NVIDIA's Clara Deploy framework for ML pipeline deployment. It connects clinical imaging modalities, a Picture Archiving and Communication System (PACS), and an ML inference server. The results and clinician feedback are stored in a customized XNAT instance, an open-source imaging informatics platform, separate from the clinical record but accessible via PACS workstations.

- **Validation Studies:** The system was evaluated using three different ML models: brain tumor segmentation, liver transplant segmentation, and hip fracture detection. Each model was integrated into clinical workflows, with the system facilitating prospective clinical validation and feedback collection. The system processed over 600 LGG

exams, 100 liver ROIs, and 2000+ pelvic X-ray studies, running automatic inference on five hip exams per day and 25 LGG exams on-demand per week without impacting clinical infrastructure.

The study demonstrated the feasibility of using open-source tools for developing a scalable and flexible ML model deployment platform in clinical settings, offering a template for other institutions looking to translate ML research into clinical practice. It also proved that in most cases, the review of X-ray exams by segmentation models took significantly less time to detect anomalies than when reviewed by humans [32]. In Fig 2.6, we can see the three different model validation studies conducted at UCSF.

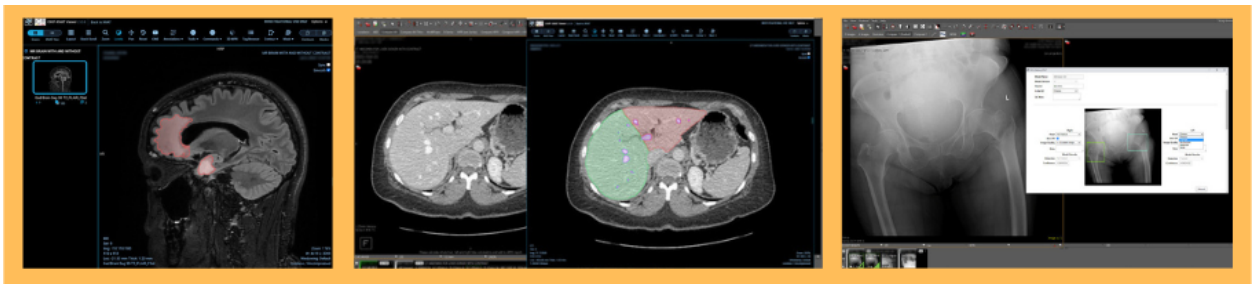

Figure 2.6: Three different model validation studies were conducted at UCSF. We have the Brain Tumor Segmentation, Liver Transplant Segmentation, and Hip Fracture Detection from the left to the right. Image source: [32].

Real-Time Laparoscopic Intra-Procedural Assistance

Another pioneering study conducted by Mascagni et al. [33] at the Institute of Image-Guided Surgery, IHU-Strasbourg, explored the feasibility of deploying AI models for real-time cognitive assistance, using Holoscan as a base platform to run optimized AI models during laparoscopic cholecystectomy (LC) [33]. This study represents one of the first clinical evaluations of AI in the operating room, where DL models were concurrently deployed to analyze endoscopic video streams in real-time.

- **AI Models and Deployment:** The AI models used included deep neural networks for surgical phase recognition, instrument localization, anatomy recognition, and the Critical View of Safety (CVS) assessment. These models were optimized using NVIDIA's TensorRT for efficient real-time inference on the NVIDIA Clara Holoscan platform, which was integrated into the operating room's video systems.

- **Clinical Implementation:** The study successfully demonstrated the real-time deployment of these AI models in three LC procedures, providing high-quality, real-time

predictions without malfunctions.

The study highlights AI systems' technical feasibility and potential clinical value in enhancing surgical safety and efficiency, setting the stage for future advancements in AI-assisted surgical procedures. The procedure was conducted at the surgical center of the IHU on November 25, 2021, and broadcast live at the 32nd Digestive System Surgery Congress and the 17th IFSES World Congress of Endoscopic Surgery [33]. Fig 2.7 shows the real-time AI assistance in laparoscopic cholecystectomy broadcast.

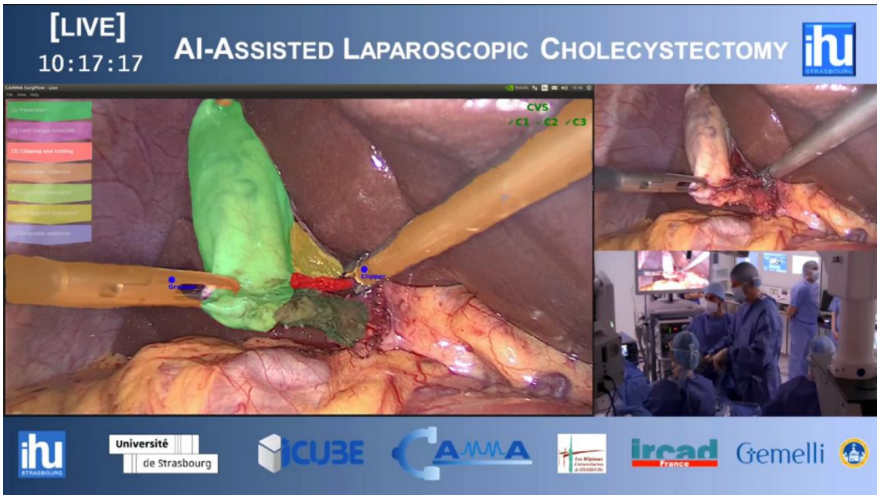

Figure 2.7: Real-time AI assistance in laparoscopic cholecystectomy broadcast. Image source: [33].

AI in Real-Time Instrument Deocclusion during Augmented Reality Robotic Surgery

To address the problem of instrument occlusion in AR during robotic surgery, a study conducted by Hofman et al. [34], ORSI academy implemented a real-time binary segmentation pipeline using NVIDIA's Clara AGX developer kit and Holoscan SDK. The study involved three live surgeries: partial nephrectomy, migrated endovascular stent removal, and liver metastasectomy. The pipeline achieved real-time binary segmentation of 37 non-organic surgical items, preventing occlusion by the AR overlay. The segmentation model demonstrated a frame-by-frame latency of 13 ms, making it feasible for real-time surgical use. The integration improved surgical safety and ergonomics, with surgeons reporting significant improvement in perceived latency and utility during procedures [34].

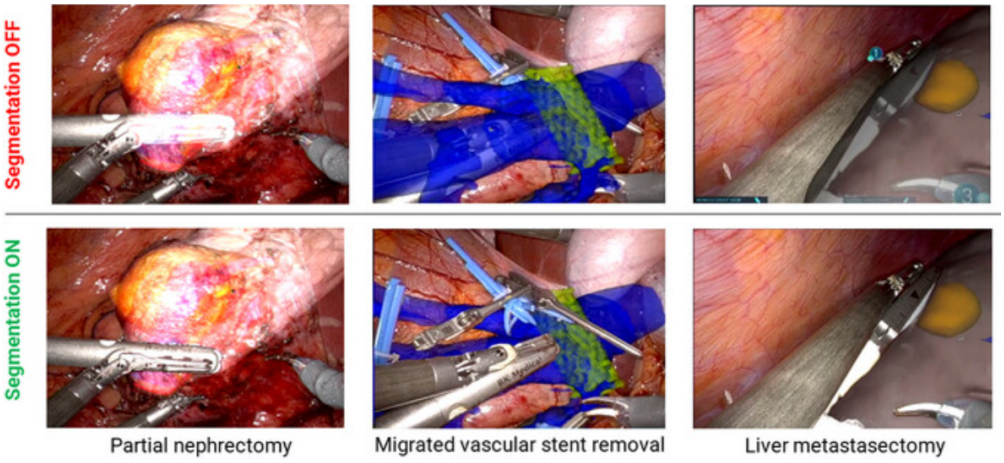

Figure 2.8: Illustration of different segmentation scenarios during three live surgeries. AR pipeline was applied in liver metastasectomy, endovascular stent removal, and partial nephrectomy, with enabled and disabled segmentation to illustrate the de-occlusion of non-organic items. Image source: [34].

AI-Assisted Augmented Reality in Thoracic Surgery

Based on the last study referenced, Hofman et al. [34] conducted a study at ORSI academy that explores the integration of AI, AR, and robotic-assisted thoracic surgery (RATS) to enhance the precision and outcomes of lung surgery and was developed with NVIDIA Holoscan SDK (leveraging the Clara AGX developer kit). ORSI academy, is a surgical training center in Belgium that focuses on developing innovative surgical techniques and technologies. The research addresses the challenges of RATS, such as the lack of tactile feedback and anatomical deformation during surgery, by employing real-time AI and AR to provide dynamic 3D anatomical models. These models offer surgeons accurate, real-time representations of patient anatomy, potentially improving surgical precision and safety.

The study focuses on developing PulmoVR, a DL-based segmentation method that creates 3D reconstructions from pre-operative computed tomography (CT) scans. These reconstructions are transformed into interactive and deformable simulated reality models using finite element method simulations. The real-time system incorporates a DL algorithm for instrument deocclusion, which segments non-organic items and overlays them on patient-specific 3D models to enhance depth perception in AR.

The setup was tested in two consecutive lobectomy cases, demonstrating successful real-time AR registration and dynamic tracking. The results showed that the AR superimposition of the 3D models was anatomically correct and enhanced the surgical procedure without influencing decision-making. This technological advancement addresses some of the inherent

challenges of RATS, such as visual access limitations and anatomical deformations during surgery, by providing a dynamic and interactive 3D visualization. Future improvements aim to incorporate automated registration and dynamic tissue tracking to enhance surgical precision further.

The study concludes that integrating AI and AR in thoracic surgery is a significant step forward, offering improved intraoperative guidance and potentially reducing operation times and learning curves [35]. We can see the AR in thoracic surgery at Fig 2.9.

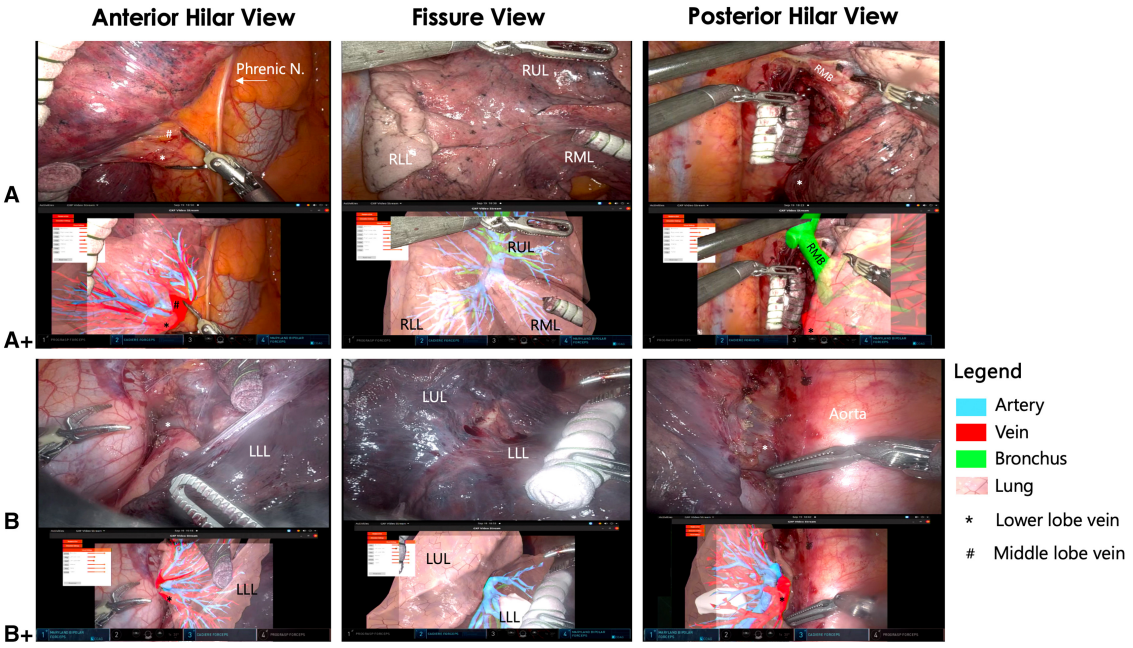

Figure 2.9: Intraoperative console images of robotic-assisted lung surgery enhanced with augmented reality. Intraoperative console screenshots of a right lower lobectomy (row A) and left lower lobectomy (row B) are presented during three standard surgical phases of a robotic lower lobectomy. Both the surgical view (A and B, top row) and the Tilepro input, augmented with 3-dimensional projections of deformable simulated reality models, (row A+ and row B+) can be seen simultaneously in the surgeon's console view. Image source: [35].

Mitigation of End-to-End Latency for Medical AI Systems

In terms of performance optimization, a study conducted by Sinha et al. [36] addresses the challenge of unpredictable end-to-end latency in medical AI systems using NVIDIA Holoscan due to GPU resource contention. The study proposes a system design optimized for heterogeneous GPU workloads by leveraging CUDA Multi-Process Service (MPS) for spatial partitioning compute workloads and isolating compute and graphics processing onto separate GPUs. Multi-Process Service (MPS) is a feature that allows multiple processes to

share the same GPU [37]. The empirical evaluation demonstrated significant performance improvements in latency determinism metrics, reducing maximum latency by 21–30% and improving latency distribution flatness by 17–25% for up to five concurrent endoscopy tool tracking AI applications compared to a single-GPU baseline. The proposed design also reduced maximum latency by 35% for up to six concurrent applications compared to a default multi-GPU setup by improving GPU utilization by 42%. These results provide clear design insights for AI applications in edge-computing domains, including medical systems, where performance predictability of concurrent and heterogeneous GPU workloads is critical [36].

2.2.2 Full system medical AI applications

The AI applications in the medical field have been increased, mainly in the last year, with several companies developing innovative solutions to meet the needs of healthcare providers and patients [38]. Some of the key players in the market include Medtronic and Odin Vision. These companies leverage AI to improve diagnostic accuracy, enhance patient care, and streamline clinical workflows. The following are some of the AI applications that these companies have developed:

Medtronic

In partnership with NVIDIA, Medtronic developed an intelligent endoscopy module, **GI Genius**. This is the first computer-aided polyp detection system using DL, capable of detecting polyps of different shapes, sizes, and morphologies in real-time locally [39]. The module is taken to the operating room and connected to the endoscope, where it analyzes the images in real-time, providing the surgeon with immediate feedback on the presence of polyps. In Fig 2.10, we can observe the Medtronics GI Genius system working in a real medical environment.

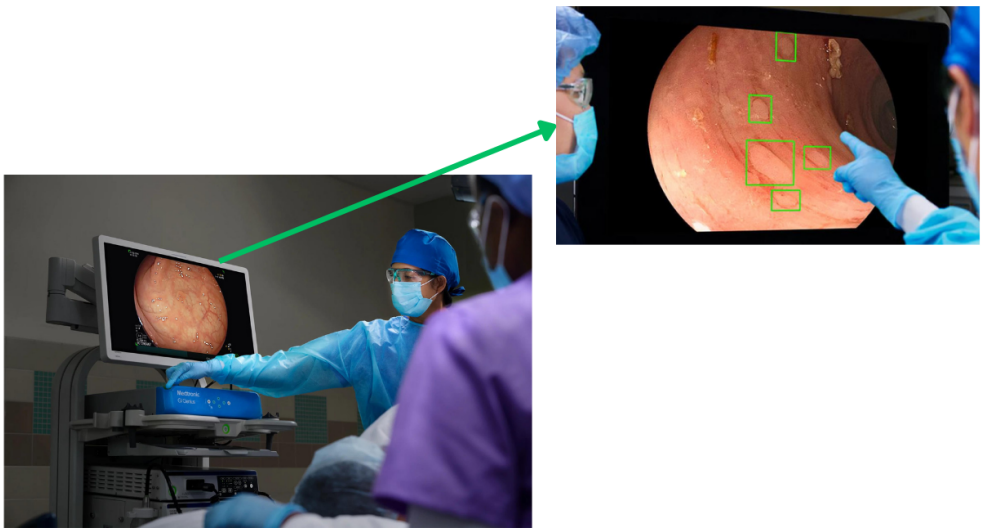

Figure 2.10: Medtronics GI Genius polyp detection system in action during a colonoscopy procedure. The system uses deep learning algorithms to analyze endoscopic images in real-time, providing immediate feedback to the surgeon about the presence of polyps. Image source: [40] [41].

Odin Vision

Odin Vision developed **CADDIE**, a cloud-based system that also uses AI to assist doctors in detecting and characterizing polyps during colonoscopy procedures. This product distinctively combines AI with cloud computing to provide real-time insights and improve diagnostic accuracy. The system leverages cloud technology to process and store data, ensuring that it can be easily scaled and integrated into existing clinical workflows. Due to the intuitive nature of CADDIE, minimal training is required, hands-free interaction during the procedure is possible, and the system can be easily integrated into existing clinical workflows [42]. The CADDIE system is at the Fig 2.11.

Figure 2.11: CADDIE cloud-based polyp detection system in action during a colonoscopy procedure. The system uses artificial intelligence and cloud computing to provide real-time insights and improve diagnostic accuracy. It processes and stores data in the cloud, allowing for easy scalability and integration into existing clinical workflows. Image source: [42].

Moon Surgical

The Maestro Robotic System, developed by Moon Surgical, is an advanced platform designed to assist surgeons during laparoscopic procedures. It leverages NVIDIA's Holoscan technology to provide real-time AI-driven support, enhancing the precision and efficiency of surgeries. The system's key features include AI-driven assistance that analyzes surgical scenes in real-time, offering valuable insights and guidance to surgeons during procedures. This capability enhances surgical precision by providing real-time feedback, enabling surgeons to make more accurate movements and reducing the risk of errors. Additionally, the system improves overall efficiency by streamlining various surgical tasks, which can reduce the time required for procedures. The system is primarily applied in laparoscopic surgeries, which are minimally invasive procedures where precision and efficiency are of utmost importance. Moreover, it is a valuable tool for training and education, offering real-time feedback and guidance to new surgeons as they develop their skills [43].

Figure 2.12: The Maestro Digital Surgical Assistant in action during a laparoscopic procedure. The system uses NVIDIA's Holoscan technology to provide real-time AI-driven support, enhancing surgical precision and efficiency. It offers valuable insights and guidance to surgeons, enabling more accurate movements and reducing the risk of errors. Image source: [44] [45].

AI-Assisted Fetal Biometry Measurements

One example of an alternative software developed and posteriorly commercialized with the goal of using AI analysis in medical imaging is the CUPID software developed by Han et al. [46]. This software is designed to assist in the automated measurement of fetal biometry parameters during the mid-trimester. The study evaluated the accuracy and efficiency of CUPID in comparison to senior and junior radiologists. The results indicate that CUPID's measurements have high intra-class correlation coefficients (ICC) and Pearson correlation coefficients (PCC) with the senior radiologist's measurements, showing good reliability. CUPID demonstrated significant time efficiency, measuring nine biometric parameters in 0.05-0.07 seconds, compared to the 4.79-11.68 seconds required by senior radiologists. The software also showed better performance in terms of mean absolute error (MAE) and percentage error (PE), highlighting its potential to reduce inter-observer variability and enhance the accuracy of fetal biometric assessments [46]. We can observe the CUPID interface in Fig 2.13.

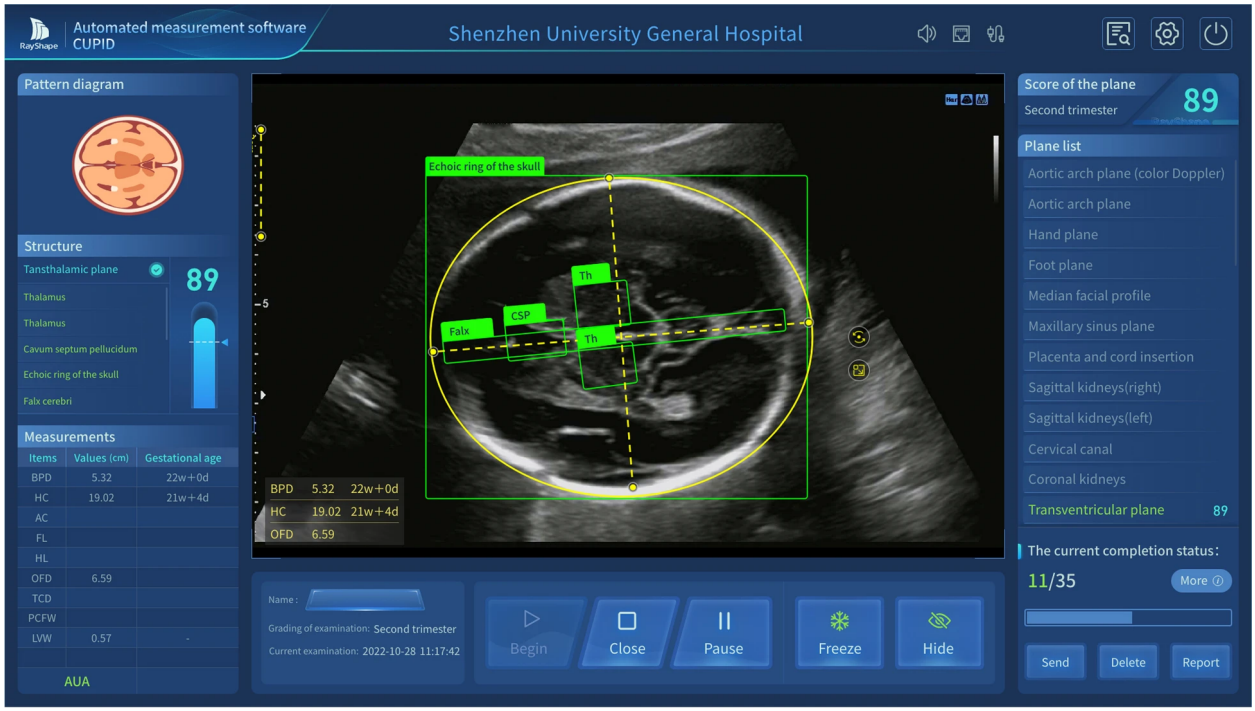

Figure 2.13: CUPID Interface demonstrating an AI-Assisted Fetal Biometry Measurement. The fetal structures are represented in green (Thalamus, Cavum Septum Pellucidum, Echoic ring of the skull and Falx Cerebri), showcasing the software's capability to provide precise and reliable measurements. Image source: [46].

2.3 Summary

This chapter has provided an overview of the research field, discussing the importance of medical image segmentation, the role of DL in medical applications, and the challenges of surgical navigation in arthroscopy. It includes a summary of the most relevant studies and projects in this area, as well as real systems used in medical environments, highlighting the key findings and contributions of each work. The section also introduces the Holoscan framework, which is a software and hardware platform essential for building and deploying AI applications for our edge computing systems. Although Holoscan is a relatively new framework, only a few studies have been conducted using this platform at the beginning of this work. In other, after observing the real systems in this chapter that used this platform and had FDA (Food and Drug Administration) approval, we can conclude that Holoscan is a feasible and secure platform for developing AI applications in the medical field.

3 Edge Computing System

This chapter describes our edge computing system for deploying AI models. It provides an overview of the architecture, the components used, and the integration of these components to optimize the performance of the AI models. We will also discuss the keyer technology used for AR overlay of DL model inference on the video feed and to provide a backup video feed.

3.1 Overview of the System Architecture

The system's architecture is designed to provide real-time AI models running on a low-power embedded GPU for arthroscopic surgery. The system comprises three main components: the Jetson AGX Orin Devkit, the NVIDIA Holoscan, and the AJA Corvid 44 12G BNC capture system. Each component was selected to fit our requirements and is integrated to optimize the performance of the AI models and ensure seamless data processing during arthroscopic surgery. The Jetson AGX Orin Devkit is the processing unit that provides high-performance computing capabilities for running AI models. The NVIDIA Holoscan platform streamlines the development and deployment of AI applications for real-time insights, offering the software needed to build AI applications and deploy sensor processing capabilities on the edge. The AJA Corvid 44 12G BNC capture system is used for video capture, enabling the system to process video data from the surgical environment with high quality and low latency.

By integrating these components, the system can deliver real-time AI models that enhance the efficiency and safety of surgical procedures, providing surgeons with advanced navigation tools during arthroscopy. The system is demonstrated at Fig 3.1.

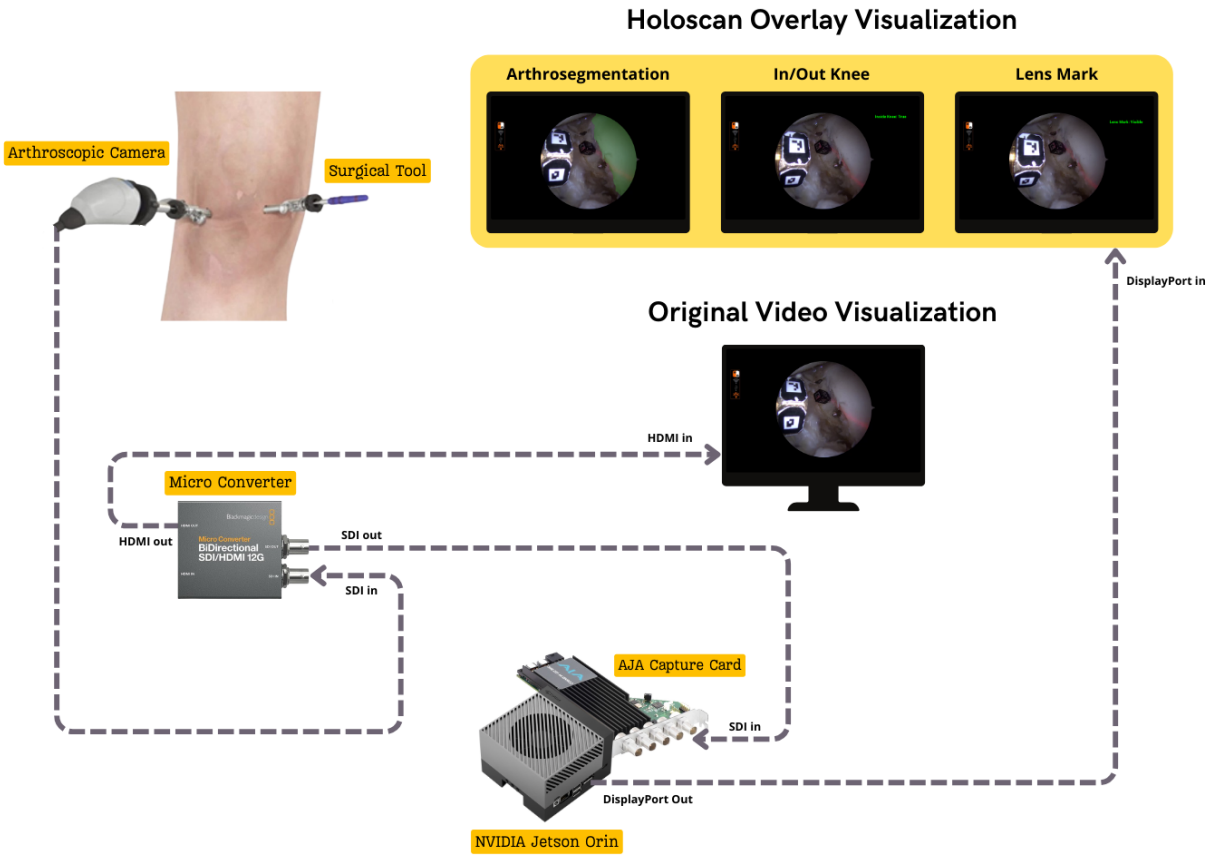

Figure 3.1: System Implementation Scheme. An arthroscopic camera connected to the Micro Converter that split the video feed to an external monitor and to the AJA Corvid 44 12G BNC capture card, which acquire the video data and sends it to the Jetson AGX Orin Devkit. The Jetson AGX Orin Devkit preprocess the video and runs the three AI models using the NVIDIA Holoscan platform to further visualization.

3.2 Image Acquisition and Processing System

The image acquisition and processing system was chosen according to the project requirements, low latency, high quality, and real-time processing.

3.2.1 NVIDIA Jetson AGX Orin Devkit

The NVIDIA Jetson AGX Orin Developer Kit is a robust platform tailored for AI and robotics applications. It leverages the Orin system-on-a-chip (SoC), which integrates several high-performance computing components, including a CPU, GPU, and deep learning accelerators (DLA) cores, to handle complex AI tasks like DL, computer vision, and autonomous navigation [21].

Regarding the GPU architecture, the Jetson AGX Orin is classified as an integrated GPU (iGPU). An iGPU is a graphics processing unit built directly onto the same die as the central processing unit (CPU). This integration allows for a more compact design and often results in lower power consumption than discrete GPUs (GPU), which are separate from the CPU. One of the critical advantages of an iGPU architecture, particularly in the context of the Jetson AGX Orin, is the efficient and rapid data transfer between the CPU and GPU. Since the CPU and GPU share the same memory pool and are located on the same chip, data does not need to traverse across different components, reducing latency and increasing overall processing efficiency [47][48].

This devkit has various interfaces and connectivity options supporting sensors and peripherals. It represents a cutting-edge technology platform that enables developers to prototype and deploy AI-powered solutions in industries like healthcare, manufacturing, and autonomous vehicles. This devkit plays a crucial role in pushing the boundaries of innovation in various industries. Its impact extends beyond individual projects, contributing to the advancement of AI technology. Additionally, the Jetson AGX Orin's iGPU benefits from NVIDIA's advanced technologies, such as Tensor Cores, DLA cores, and CUDA support, to accelerate AI computations [49].

Processing AI models in real-time requires intensive computational effort on the part of the machine, which involves hardware with high processing power. The NVIDIA Jetson AGX Orin Developer Kit allows developers to develop and deploy AI applications that can be scalable to higher-performance machines such as the NVIDIA IGX Orin (an enterprise solution) and Clara AGX (a medical solution) in the future [50].

This kit allows us to create advanced robotics and AI on-the-edge applications in the medical field. This module has an AI performance of 275 TOPS (Tera Operations per Second), eight times higher than the performance of the previous generation [49]. The specific description of the hardware NVIDIA Jetson AGX Orin Devkit is detailed in Appendix C.1.

3.2.2 AJA Corvid 44 12G BNC

Hardware

Due to the need for a high-quality video capture device, we chose the AJA Corvid 44 12G BNC, a compact and high-speed PCIe 3.0 card equipped with four bidirectional 12G-SDI HD-BNC connections. 12G-SDI can handle a considerable amount of data, offering significantly greater bandwidth, with a data transfer capacity of approximately 12 Gbit/s per second.

This card supports various resolutions and enables multi-channel 4K workflows up to 50/60p. Designed for more demanding video and audio operating flows, the card supports 10-bit 4:2:2 and 4:4:4 color formats, reaching up to 12 bits of color space. In addition to offering multiple video inputs/outputs, it stands out for its ability to perform Mixer/Keyer operations. This specific feature expands the video manipulation possibilities provided by this capture system and allows us a second output from the acquisition board as a backup in case the algorithm's output presents unexpected behaviors [51]. The specifications of this capture system are described in detail in Appendix C.2.

Software

It is necessary to sign a nondisclosure agreement to take advantage of all the features offered by AJA Corvid 44 12G BNC, including access to the SDK provided by the company. Due to this contractual condition, providing a detailed description of the SDK in this specific context is impossible. However, it is essential to mention that the SDK contains detailed information about the board's operation and also access to exclusive software, providing comprehensive knowledge about the device [1] . This access to detailed documentation allows for a broader and more in-depth understanding of the capabilities and functionalities of the AJA Corvid 44 12G BNC [51].

[1]Firmware Considerations: Initially, the AJA Corvid 44 12G BNC capture card came with a firmware version that did not include essential widgets such as the Color Space Converter (CSC) and Keyer, which were critical for our application, particularly for real-time augmented reality overlay during surgery, for having a backup video and for RGB conversion for Holoscan. Widgets implement signal processing functions in firmware. This led to the need to flash the board with the correct firmware version that supported these functionalities. For more details on how this firmware update impacted the deployment of DL models, refer to Section 4.2.

3.2.3 SDI/HDMI 12G Bidirectional Micro Converter

Hardware

The selected hardware has the function of converting both SDI to HDMI and HDMI to SDI, supporting a variety of formats. This versatility allows handling different SD, HD, or Ultra HD video standards in each conversion direction, effectively providing functionality equivalent to having two converters in a single device [52]. The detailed description of th is bidirectional micro converter is provided in Appendix C.3.

Software

The 12G SDI/HDMI Bi-Directional Micro Converter has software provided by Blackmagic Designs that allows us to control the adjustment up to certain converter restrictions, such as the color formats passed through the video inputs [53].

3.2.4 NVIDIA Holoscan

As mentioned, the Holoscan framework is a software and hardware platform enabling developers to build and deploy AI applications for edge computing systems. It provides tools and libraries for developers to create AI applications that run on edge devices, such as the NVIDIA Jetson AGX Orin Developer Kit. The Holoscan framework is designed to be easy to use, flexible, and scalable, making it ideal for various applications, from medical imaging to autonomous vehicles [21].

To provide real-time insights, the Holoscan framework leverages NVIDIA TensorRT to perform AI inference, optimizing the performance of DL models for efficient execution on edge devices. TensorRT is a high-performance inference library that provides optimizations such as layer fusion, precision calibration, and kernel auto-tuning, which help reduce latency and increase throughput. It uses the GPU to accelerate the execution of DL models [21] [24].

To utilize TensorRT within Holoscan, models must be in a format compatible with TensorRT's engine files. This is where ONNX (Open Neural Network Exchange) comes into play. ONNX is a standard, open-source format for representing ML models across different frameworks (Pytorch, TensorFlow, Matlab), facilitating interoperability [54]. Integrating a model with the Holoscan framework typically involves converting the model to ONNX format, which can then be parsed and loaded by TensorRT. When a model is converted to ONNX, Holoscan loads the ONNX model and uses TensorRT to build an optimized inference

engine. TensorRT takes the ONNX model as input and generates an optimized engine file, which is then used to execute inference on the GPU. This process includes optimizations such as precision scaling (e.g., FP32 or FP16), layer fusion, and memory efficiency improvements, which ensure the model can run efficiently on edge devices with limited resources [24]. By using ONNX as an intermediate format, Holoscan provides flexibility for developers, allowing models trained in a wide variety of frameworks to be easily integrated into the real-time AI pipelines that Holoscan supports [54] [55]. Fig 3.2 shows the interoperability of ONNX with different frameworks, and figure 3.3 shows the workflow of TensorRT.

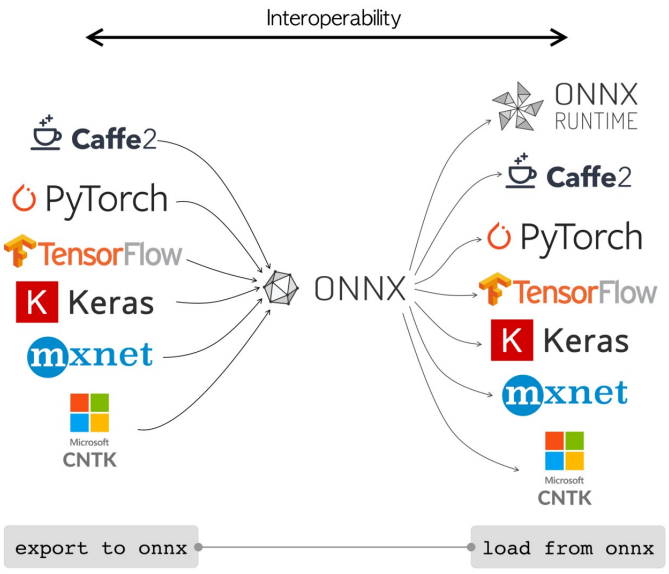

Figure 3.2: ONNX interoperability between many frameworks serving as bridge. Image source: [56].

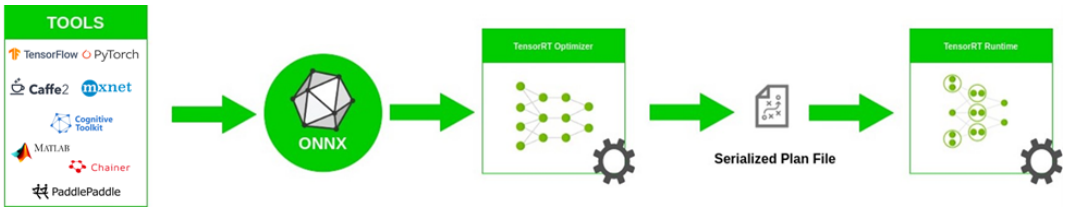

Figure 3.3: TensorRT Workflow from compatible format conversion to inference runtime. Image source: [57].

Understanding how ONNX serves as the bridge between model development and deployment in TensorRT highlights the importance of both technologies in the Holoscan ecosystem, ensuring that AI models are portable and optimized for real-time execution. To understand the Holoscan framework, it is essential to understand the core concepts of this platform:

Operators

Operators are the building blocks of the Holoscan framework. They are small, self-contained units of computation that can be combined to create complex applications, as seen in figure 1.1.

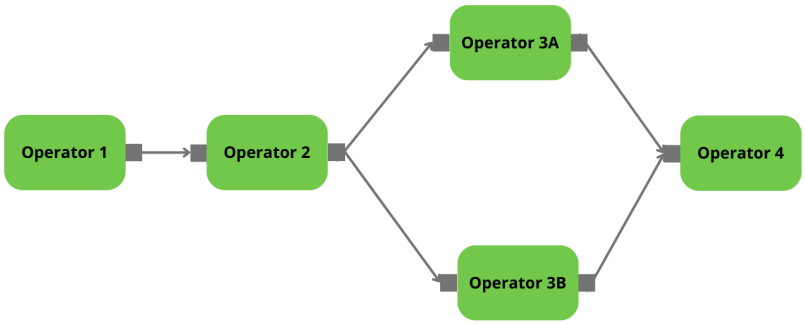

Figure 3.4: **Core concepts:** Operators

Operators can be used to perform a wide range of tasks, from simple image processing to complex machine-learning algorithms. The Holoscan framework provides a library of pre-built operators for developers to build their applications. Developers can also create custom operators to meet specific requirements [21]. Below are explained the pre-built operators:

- **AJASourceOp** (`aja`): Captures video data from AJA video capture devices, enabling real-time video streaming into the pipeline.
- **BayerDemosaicOp** (`bayer_demosaic`): Converts raw Bayer image data into full RGB images by performing a demosaicing process.
- **FormatConverterOp** (`format_converter`): Converts image or video data from one format to another, ensuring compatibility across different parts of the pipeline.
- **HolovizOp** (`holoviz`): Visualizes image, video, or other data streams, providing a real-time display of the pipeline's output. This operator can use Vulkan or OpenGL for rendering.
- **InferenceOp** (`inference`): Executes ML inference tasks with pre-trained models using TensorRT, leveraging GPU acceleration for fast processing.

- **InferenceProcessorOp** (`inference_processor`): Processes the outputs from `InferenceOp`, performing tasks like post-processing, filtering, or aggregation of inference results.
- **PingRxOp** (`ping_rx`): Receives ping messages, typically used for testing communication between different parts of the pipeline.
- **PingTxOp** (`ping_tx`): Sends ping messages, often used in conjunction with `PingRxOp` to test pipeline connectivity.
- **SegmentationPostprocessorOp** (`segmentation_postprocessor`): Post-processes segmentation results from inference tasks, such as applying thresholds or filtering.
- **VideoStreamRecorderOp** (`video_stream_recorder`): Records video streams from the pipeline, saving the output to disk for later review or analysis.
- **VideoStreamReplayerOp** (`video_stream_replayer`): Replays recorded video streams into the pipeline, allowing for testing or analysis of pre-recorded data.
- **V4L2VideoCaptureOp** (`v4l2`): Captures video from V4L2-compliant video devices (Linux), enabling the integration of a wide range of video capture hardware into the pipeline.

Ports

An Operator receives streaming data at an input port, processes it, and publishes it to one of its output ports. Operators can have multiple input and output ports to process multiple data streams simultaneously.

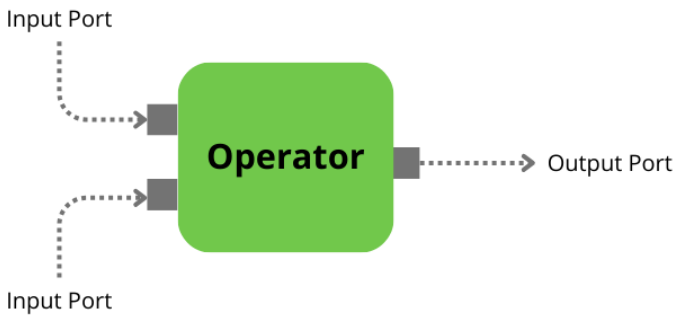

Figure 3.5: **Core concepts:** Operator Ports

Operator Communication

Operators communicate by passing messages through these ports. When an operator

processes data, it takes input messages from its input ports, performs the necessary computations or transformations, and then publishes the resulting messages to its output ports. The framework ensures that these messages are efficiently routed from the output ports of one operator to the input ports of connected downstream operators.

Message

A message is a generic data object used for communication between operators. Messages carry the data required for processing, enabling a seamless flow of information through the workflow. Messages are from a type dictionary, where each key-value pair represents a piece of data. Messages abstract the underlying data structures, providing flexibility in handling different data types.

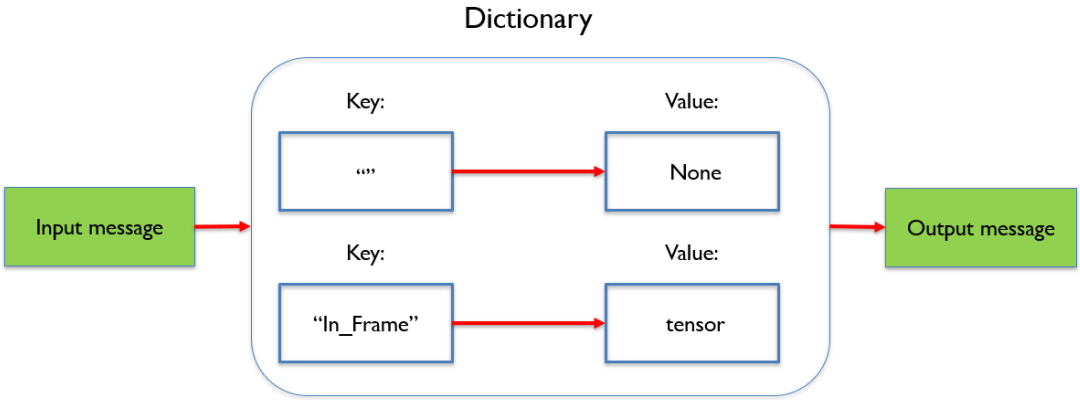

Figure 3.6: **Core concepts:** Message

Execution and Synchronization

The Holoscan runtime manages the execution of operators, ensuring that data is processed in a timely and efficient manner. Conditions can be used to control whether an operator executes based on the state or content of the incoming messages, providing a way to synchronize and manage complex workflows. Holoscan supports zero-copy techniques to minimize overhead and improve performance. This allows messages to be passed between operators without unnecessary data copying, which is crucial for high-performance streaming applications.

These core concepts of Holoscan are fundamentally based on the Graph Execution Framework (GXF), which provides the underlying architecture for the execution of streaming data applications. GXF is a modular and extensible framework for building high-performance AI applications that provides modularity at the application level since existing entities can be swapped or updated without recompiling any extensions or applications. While Holoscan

introduces its interface and enhancements, it leverages the robust and proven components of GXF to ensure high performance and scalability [21].

NVIDIA provides complete documentation for this platform [21] [50]. NVIDIA Holoscan has a Software Development Kit (SDK) open source and also a Github repository with examples and relevant information for testing and work carried out [24] [25].

3.3 Keyer for AR overlay

The AJA Corvid 44 12G BNC capture system includes a sophisticated keyer technology that enhances real-time surgical navigation by managing the overlay of AI-driven augmented reality (AR) onto live video feeds. This keyer allows the system to control the visibility of DL model inferences on the video stream, ensuring that critical insights are presented to the surgeon promptly and accurately and if needed bypassing the original video directly with almost no latency. This means the system can merge the AI inferences directly with the video feed or display them separately, depending on the specific surgical requirements. Additionally, the system can generate a secondary output from the acquisition board, serving as a backup if the AI algorithm's output behaves unexpectedly, thus ensuring continuous and reliable visual guidance. This adaptability is particularly important during arthroscopic surgeries, where precision and real-time feedback are essential for efficiency and safety. By simultaneously processing the live video and AI outputs, the keyer technology ensures that the augmented reality information seamlessly integrates into the surgeon's field of view. This seamless integration is a testament to the system's performance, helping surgeons navigate complex arthroscopic environments with greater confidence and accuracy and ultimately contributing to better patient outcomes. The keyer technology is a critical feature of the AJA Corvid 44 system, significantly enhancing surgical precision, confidence, and decision-making [51].

3.4 Summary

In this chapter, we have described the architecture of our edge computing system for AI model deployment. We have discussed the components used in the system, including the NVIDIA Jetson AGX Orin Devkit, the AJA Corvid 44 12G BNC capture system, and the SDI/HDMI 12G Bidirectional Micro Converter. We have also provided an overview of the NVIDIA Holoscan platform, which enables the development and deployment of AI

applications for real-time insights. Finally, we have discussed the keyer technology usage for AR overlay of DL model inference on the video feed and to provide a backup video feed. These components are integrated to optimize the performance of the AI models and ensure seamless data processing during arthroscopic surgery.

4 Deployment of Deep Learning Models in Edge Computing System

This chapter describes the deployment of DL models in an edge computing system for arthroscopic surgery. It provides an overview of the AI models used to improve surgical navigation, their implementation in the Holoscan framework, and the tests carried out to validate the system.

4.1 Deep Learning Models for Improved Surgical Navigation

To address the challenges of arthroscopic precise navigation due to the complexity of the knee's internal structures and the variability in the intraoperative environment, Perceive3D developed some models to enhance surgical navigation in knee arthroscopy and previously trained and validated these models. However, they were only experimentally validated offline in a research setting. These models include a femur segmentation model, an inside/outside knee joint detection model, and a lens mark visibility detection model.

4.1.1 Femur segmentation model

The femur segmentation model is designed to facilitate accurate and efficient registration of the knee anatomy during arthroscopic surgery, specifically used for the previously mentioned structured light for touchless 3D registration method [28]. This model is critical for segmenting the femur bone and cartilage regions within the arthroscopic images while excluding other structures, such as the tibia, ligaments, and meniscus. By accurately identifying the regions of interest, this model aids in aligning intra-operative data with pre-operative 3D models, an essential step in surgical navigation.

The segmentation model utilizes a U-Net architecture, a CNN tailored for biomedical image segmentation. We can observe the model architecture in Fig 4.1. The network was trained on a dataset of arthroscopic images, supplemented by a novel data augmentation technique that introduces synthetic laser projections to improve the model's robustness in real-world scenarios. This augmentation addresses the challenges of laser projection interference, a common issue during intra-operative imaging.

Experimental validation demonstrated that the model performs well, achieving a high Dice score for segmenting femur and cartilage regions. Moreover, introducing synthetic laser projections during training significantly improved the model's image performance with actual laser projections, making it a vital component of the overall navigation system.

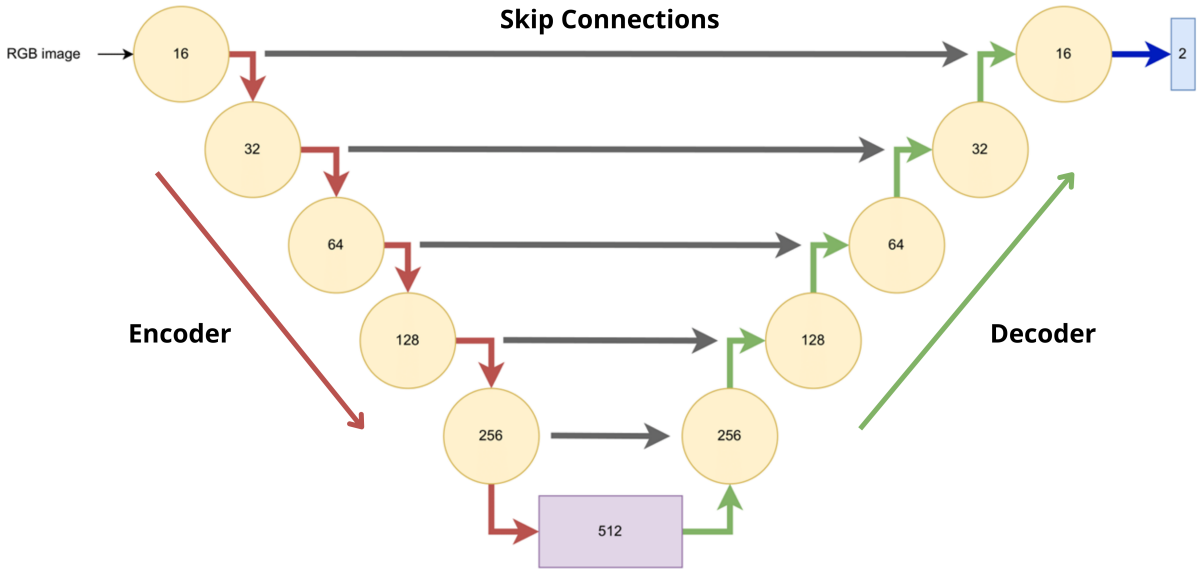

Figure 4.1: The femur segmentation model follows a U-Net architecture, consisting of encoder and decoder blocks, with the number of channels specified in each block. The encoder blocks (shown in red) extract important features from the input image, progressively downsampling it to capture abstract representations. The decoder blocks (shown in green) then upsample the feature maps to reconstruct the segmented image. Skip connections (represented in gray) are used to directly connect corresponding encoder and decoder blocks, ensuring the model retains spatial information at multiple scales, which enhances segmentation accuracy. The output is a segmentation mask highlighting the femur region. (Courtesy of Perceive3D)

4.1.2 Inside/Outside knee joint detection model

The Inside/Outside classification model was developed by Perceive3D and is a DL approach specifically designed to detect whether an arthroscopic image is inside or outside the knee joint during knee arthroscopy procedures. The primary goal of this model is to automate the control of the arthroscope's components, such as the light source (avoid dazzle or burning medical personal), irrigation pump (avoid flood surgical environment), which are crucial for maintaining a clean and functional surgical environment.

The Inside/Outside classification model, developed using MATLAB's Deep Learning Toolbox, aims to determine the location of the arthroscope (inside or outside the knee) based on the input video frames from the arthroscopic camera. This binary classification task labels images as either "inside" (0) or "outside" (1). The model is built using a CNN, which consist of several layers, including convolutional layers, batch normalization layers, ReLU (Rectified Linear Unit) layers, max-pooling layers, and a fully connected layer as seen in fig 4.2. These layers are responsible for processing the input images and extracting relevant features that help the model make accurate predictions.

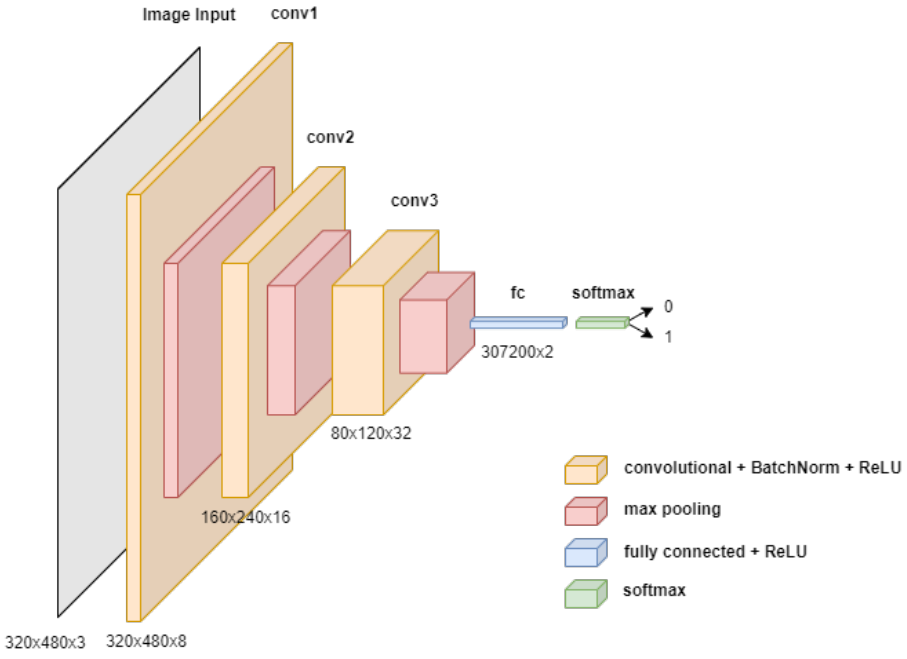

Figure 4.2: Inside/Outside classification model architecture.

The dataset used for training and testing the model comprises thousands of images extracted from actual arthroscopic videos, supplemented with additional images obtained through web scraping to simulate various scenarios outside the knee. Data augmentation

techniques, such as random rotation and scaling, were applied to diversify the training set and improve the model's generalization ability.

The classification model demonstrated high accuracy during the training and testing phases, achieving accuracy rates as high as 99.80% on the test set in its best iteration.

4.1.3 Detection of lens mark visibility

This model was developed by Perceive3D and is specifically focused on improving the detection and localization of a lens mark on the arthroscopic camera, which is crucial for accurate camera calibration and orientation estimation during surgery. By leveraging CNNs and synthetic data augmentation techniques, this model will improve the detection and localization of a lens mark on the arthroscopic camera. This is crucial for accurate camera calibration and orientation estimation during surgery. The main objective of this project was to create two complementary DL models: a classification model and a regression model.

We will focus on deploying the classification model to detect whether the lens mark is visible in the current image. A large dataset of annotated arthroscopic videos was used to train the models. Authentic arthroscopic images were supplemented with synthetic images created by the Perceive3D synthesizer. This augmentation process generated diverse images that mimic real-world surgical conditions, including variations in light, lens angle, and magnification. One key challenge Perceive3D faced was detecting the lens mark, even in cases located near the edge of the image or obscured by lighting issues.

This model was developed using MATLAB's Deep Learning Toolbox, which provides a framework for building and training CNNs. The classification model is based on a CNN structure commonly used for image classification tasks. The architecture includes multiple layers that process the input image through a series of convolutional operations, pooling, and fully connected layers, leading to a final classification output that determines whether the lens mark is visible (class 1) or not visible (class 0) and was optimized to detect the presence of the lens mark with an accuracy of 94% on the test set.

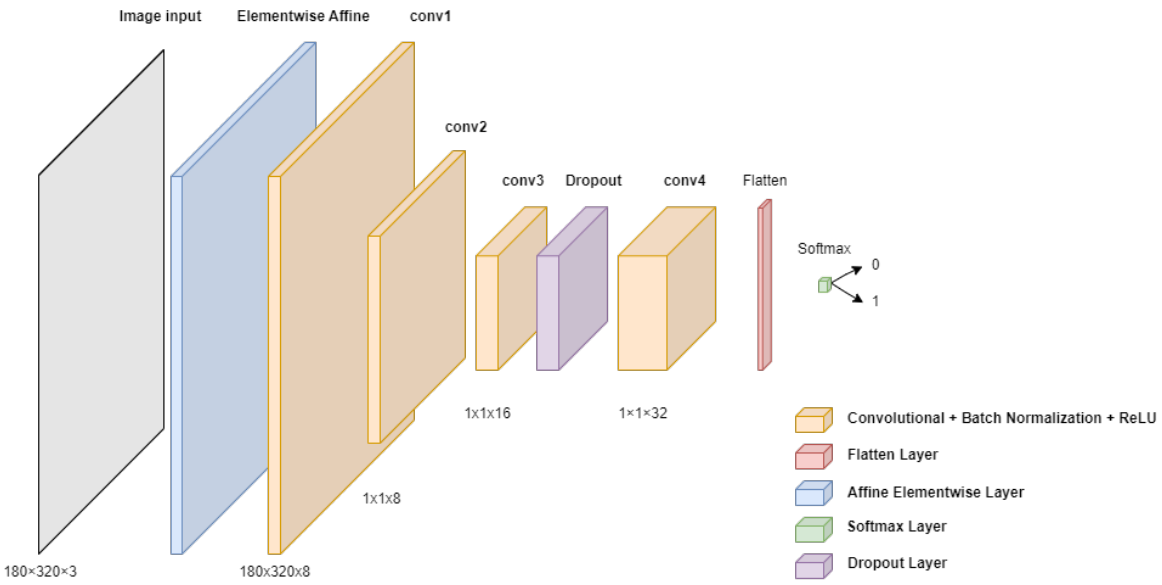

Figure 4.3: Lens Mark classification model architecture.

The final model showed significant improvements over previous solutions.

4.2 Implementation of DL Models in Holoscan

To develop our application and deploy the models in a real-time system, the primary approach was to study the NVIDIA Holoscan framework, evaluate the existent applications, and, using the tutorial guidelines presented at Holoscan SDK documentation, do the setup of NVIDIA Jetson AGX Orin devkit, needed to deploy our application [24].

Holoscan provides us with a set of examples. These were first tested to ensure the correct setup of the system and to understand the basic concepts of the Holoscan SDK. The examples are designed to be simple and easy to understand, providing a starting point for users to build their own applications.

We tested the example mentioned earlier in section 2.1.2 because it shares similarities with the architecture of our own system. The application ran successfully on our setup, displaying real-time results on the screen, as demonstrated in Fig 2.1. We used this as a baseline for developing our own system.

To build our application and deploy the models within the Holoscan framework, we followed a series of steps, from capturing the input video to displaying the segmentation results. We used several flags to control the execution, including options for selecting the video source, the model, and result visualization.

Conditional Workflow Paths

The application defines different workflows based on the flags set for the video feed source and the preprocessing requirements. The application is composed of two main files: a Python script and a YAML configuration file designed to explicit the operator's parameters configuration. The Python script defines the main flags but can be changed when calling the application to run at the command line. Passing as arguments to the application we can set the following flags: video source, model to be used, data path and scheduler type. The flags that influence the workflow are:

- `is_aja`: Indicates whether the video source is the AJA capture system.
- `do_preprocessing`: Indicates whether preprocessing is required.
- `arthrosegmentation_model`, `in_out_model`, `lens_mark_classification_model`, `boths_models`: Indicate the type of model being used.

Depending on these flags, the paths through the application differ as described in detail at Appendix D.9. The application can handle different video sources and preprocessing requirements, providing a robust and adaptable workflow for various scenarios.

In each case, the preprocessed video frames are sent to the inference module, where the results are further processed and visualized. The visualized frames are forwarded to the Holoscan built-in recorder if output recording is enabled. This conditional logic ensures the application can flexibly handle different video sources and preprocessing requirements, providing a robust and adaptable workflow for various scenarios.

We have two methods for feeding Holoscan with a video source. The first is to use a video file as input. The second is to use the AJA Corvid 44 12G BNC capture system, a professional video capture card that can capture video from various sources.

Video File Source

To feed Holoscan with a video file source, the application utilizes the built-in `VideoStream ReplayerOp` operator. This operator is designed to replay a video stream encoded as GXF entities. The tensors will be saved with metadata indicating that the data should be copied to the GPU on read. To do this, the pre-recorded arthroscopy HD video must be converted into the GXF tensor format. This conversion is accomplished using a script called `convert_video_to_gxf_entities`, which has been modified to accept individual image frames instead of entire video files, thereby preserving the lossless quality of the video source. The output is stored as a GXF file, which is subsequently processed by the

`VideoStreamReplayerOp` operator. `VideoStreamReplayerOp` reads the GXF file and publishes video frames as deserialized tensors. In the examples distributed with the SDK, the tensors are generally GPU-based. The detailed configuration parameters for this operator can be found in Appendix D.1.1.

The primary output of the `VideoStreamReplayerOp` operator is a `Tensor`, which contains a video frame deserialized from the GXF file stored on disk. The configuration of these parameters allows for flexible control over the video replay, including adjusting playback speed, handling corrupted data, and turning on or off looping of the video stream. Then we send one of the paths to the Visualizer (`HolovizOp`), where the video feed is displayed on the screen, and the other path to the Format Converter (`FormatConverterOp`) to proceed with the inference steps.

AJA Corvid 44 12G BNC Video Capture Source

The second method employed to capture a video feed from the arthroscopic camera and provide it as input to the Holoscan framework utilized the AJA Corvid 44 12G BNC capture system. In this setup, an SDI/HDMI 12G Bidirectional Micro Converter was used to convert the video feed from the SDI arthroscopic camera into an HDMI output for monitoring while simultaneously routing another SDI output to the NVIDIA Jetson AGX Orin Developer Kit via PCIe. AJA's drivers were installed on the NVIDIA Jetson AGX Orin Developer Kit to enable the system to recognize the capture card. Using AJA's SDK built-in tools, the video feed from the arthroscopic camera was captured and displayed on the screen.

To integrate the AJA Corvid 44 12G BNC capture system with the Holoscan framework, we utilized the built-in `AJASourceOp` operator. The `AJASourceOp` operator is responsible for reading the video stream directly from the AJA capture card and feeding it into the Holoscan framework for further processing and provides various configuration options that allow fine-tuning of the video capture process to meet the application's specific needs. The detailed configuration parameters for this operator can be found in Appendix D.1.2.

Additionally, the operator can accept an optional input, `overlay_buffer_input`, which allows mixing the incoming overlay with the captured video when overlay functionality is activated. By using the CSC widget in the AJA Corvid 44 12G BNC capture system, the video feed is converted from YUV to RGB format, which is the format expected by the Holoscan framework. In figure 4.4, we can observe the AJA Corvid 44 12G BNC capture card widgets for Color Space Conversion, Frame store, Mixer Keyer, and overlay.

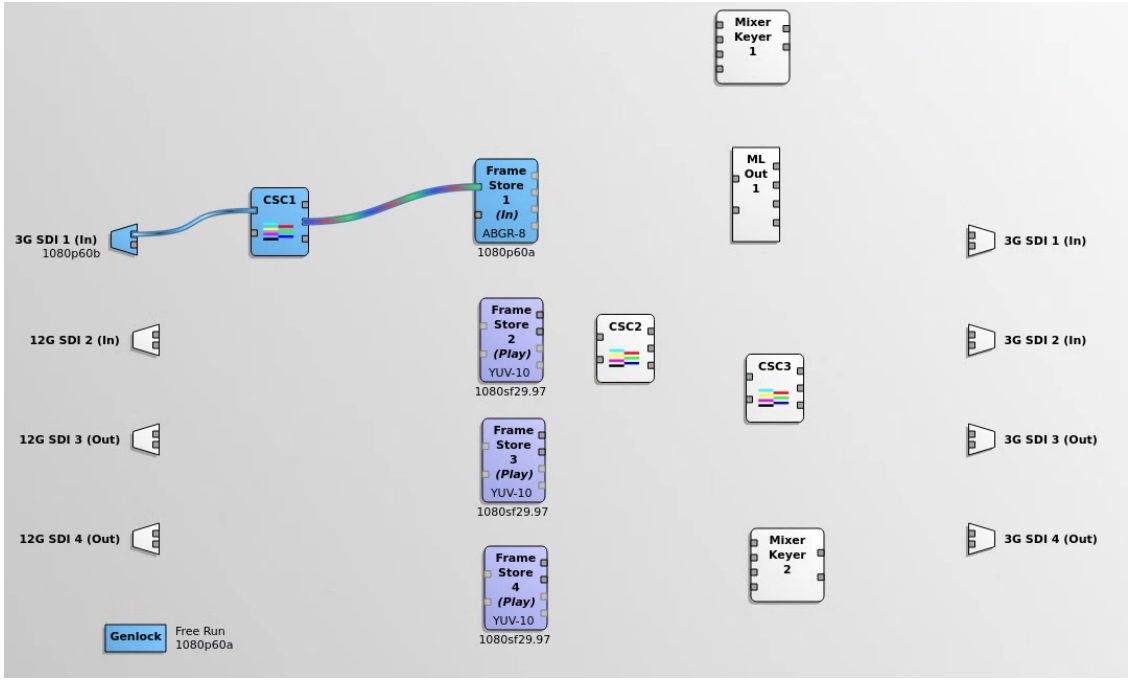

Figure 4.4: AJA Corvid 44 12G BNC capture card widgets for Color Space Conversion, Frame store, Mixer Keyer, and overlay. A FrameStore is a hardware widget within FPGA firmware that manages video data between video sources and SDRAM (memory). It operates in two modes: capture (writing data to SDRAM) and playback (reading data from SDRAM). A Color Space Converter (CSC) is an FPGA-based device used to convert video data between YCbCr (luminance and chrominance) and RGB (red, green, blue) color spaces. A Mixer/Keyer, implemented in FPGA firmware, combines or "keys" YCbCr video signals. It processes only YCbCr video and is configured via a pair of registers.

AJA output adds an alpha channel to the RGB image, which is unnecessary for our application, so it's removed at the next step. Then we send one of the paths to the Visualizer (`HolovizOp`), where the video feed is displayed on the screen, and the other path to the Format Converter (`FormatConverterOp`) to drop alpha channel and proceed with the inference steps.

Firmware Update for AJA Corvid 44 12G BN:

During the deployment phase, it became evident that the AJA Corvid 44 12G BNC capture card's default firmware did not include support for the CSC and Keyer widgets, both of which were crucial for our real-time AI inference, backup video feed, and augmented reality overlay capabilities. These features are necessary for the system to convert the YUV color format to RGB (expected by Holoscan) and to the keyer functionality. To address this, we flashed the board with the correct firmware version

that included the needed functionalities. After updating the firmware, the card was enabled with these required features.

Drivers Update for AJA Corvid 44 12G BN:

In addition to the firmware update, we encountered issues with the drivers for the AJA Corvid 44 12G BNC capture card. Holoscan was unable to recognize the card's drivers with the RDMA (Remote Direct Memory Access) feature enabled. RDMA allows the device to access the host's memory directly, bypassing the CPU, which is critical for real-time video processing and enabling the keyer functionality. During development, we worked closely with both the AJA and NVIDIA support teams to resolve this issue. However, the drivers provided by AJA were incompatible with the NVIDIA Jetson AGX Orin Developer Kit, and vice versa. By the end of the development phase, we were unable to enable the RDMA feature, which also meant the keyer functionality was unavailable. Instead we used the standard AJA drivers without RDMA support and our application can be easily adapted to use the keyer feature once compatible drivers are released [58].

4.2.1 Deployment of femur segmentation model

The femur segmentation model is used for segmenting arthroscopic images. Before the inference process, some steps are necessary to prepare the images and the model.

Image Processing

Preprocessing the images is essential because the DL model requires input images to be in a specific format. The following steps outline the preprocessing pipeline:

- **Initial Image Resizing:** The input image is first resized to match a standardized size of `1920x1080` pixels. This ensures that all input images have consistent dimensions before further processing. OpenCV is a powerful library for image processing that provides a wide range of functions for image manipulation and analysis [59]. The resizing uses OpenCV's `INTER_LINEAR` interpolation method to maintain image quality during scaling.

- **Cropping Outside the Circle:** After resizing, the image is cropped to focus on the region of interest (ROI), a circular area within the image. This step ensures that only relevant areas, such as the knee, are preserved for further analysis. The cropping is

done by identifying the largest connected component in the resized image and masking out everything outside the circle.

- **CLAHE Contrast Enhancement:** Contrast Limited Adaptive Histogram Equalization (CLAHE) is applied to the circular ROI to enhance the image contrast. This step improves the visibility of subtle features in the image, making it particularly useful for medical images where fine differences in tissue structure are essential.

- **Final Image Resizing:** After contrast enhancement, the image is resized again to a final size of `1024x1024` pixels, as the DL model requires. This final resizing ensures the processed image is correctly formatted for model inference.

The preprocessing pipeline was initially implemented in Python by the Perceive3D team. To improve performance, it was converted to C++ and optimized using CUDA. The C++ implementation was wrapped into Python for integration into the NVIDIA Holoscan framework. Since Holoscan does not provide native preprocessing operators, a custom operator called `ImageProcessingOp` was created to handle these preprocessing tasks. OpenCV functions were used to perform the resizing, cropping, and CLAHE contrast enhancement. Detailed implementations for C++ can be found in Appendix E.1, and the Python implementation is covered in Appendix D.2.

An extra `FormatConverterOp` operator is used to convert the data to float32 format and ensure the pixel scale of 0 to 1, which is the format expected by the model. The detailed configuration parameters for this operator can be found in Appendix D.1.3.

CUDA optimization

We implemented the image preprocessing pipeline using CUDA to accelerate the preprocessing steps. This allowed us to leverage the GPU's parallel processing capabilities to speed up the preprocessing of arthroscopic images significantly. Using NVIDIA Visual Profiler, we detected the bottlenecks in the code and optimized it to achieve the best performance possible.

Amdahl's Law is used to find the maximum improvement in the performance of a task when only part of the task can be improved. It is often used in parallel computing to predict the theoretical maximum speedup using multiple processors [60]. Using Amdahl Law, we confirmed theoretically that the only process advantageous to implement in CUDA was the CLAHE operation because it is a very computationally expensive operation, and the GPU

can handle it better than the CPU. The other operations were already optimized in the OpenCV library, but the GPU did not significantly improve.

This formula gives Amdahl's Law:

$$S = \frac{1}{(1-P) + \frac{P}{N}}$$

Where:

- S is the speedup.
- P is the proportion of the program that can be parallelized.
- N is the number of processors.

In C++, preprocessing time took 35 ms, and CLAHE was responsible for 29 ms of this time. So, we have a task where 82.8% of the task can be parallelized ($P = 0.828$), and we are using 2048 processors ($N = 2048$).

$$S = \frac{1}{(1-0.828) + \frac{0.828}{2048}}$$

$$S = \frac{1}{0.18028 + \frac{0.828}{2048}} = \frac{1}{0.18028 + 0.000404} = \frac{1}{0.180684} \approx 5.54$$

So, the maximum speedup we can achieve with 2048 processors is approximately 5.54 times. The CUDA implementation was optimized to take advantage of the GPU's architecture, enabling faster image processing and real-time performance. The function `cv::cuda::CLAHE` was used to perform the CLAHE operation on the GPU. This operation enhances the image's contrast by applying adaptive histogram equalization to each channel. The CLAHE operation is performed on the GPU using CUDA for faster processing. The detailed implementation of these CUDA preprocessing steps can be found in Appendix E.2.

Model conversion to TensorRT and inference

This model was previously trained at PyTorch, as mentioned before it need to be converted to a format compatible with NVIDIA Holoscan. ONNX will work as a bridge between PyTorch and TensorRT. We developed a script for this conversion that converts the PyTorch model to ONNX format. The script loads the PyTorch model, sets it to evaluation mode, and then exports it to ONNX format. The detailed implementation of this conversion can be found in Appendix A.3.

When converting the model from PyTorch to ONNX, our input channel order is NCHW (batch size, number of channels, height, width) and must be converted to NHWC (batch size, height, width, number of channels). To this, we use an example transformation script named graph_surgeon.py that NVIDIA provides. The detailed implementation of this conversion can be found in Appendix A.2.

Finally, we can give the ONNX model to Holoscan as input. Holoscan operator `InferenceOp` will then generate the TensorRT engine and is responsible for running the inference on the TensorRT engine based on our configurations. Fig 4.5 illustrates the conversion pipeline described. The operator reads the input tensor, performs inference using the TensorRT engine, and publishes the output tensor. The detailed configuration parameters for this operator can be found in Appendix D.1.4.

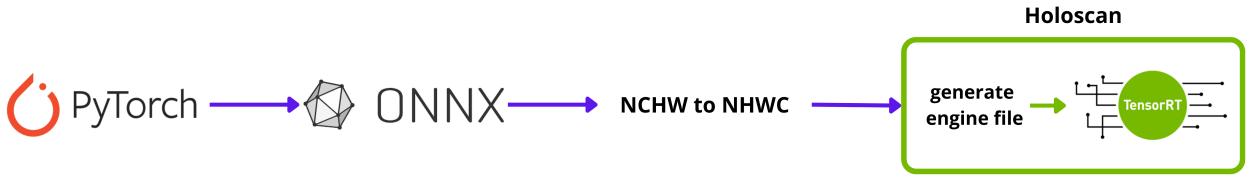

Figure 4.5: Pytorch to Holoscan conversion pipeline.

Usually, Holoscan inference uses FP32 precision, a feature that allows the model to use 32-bit floating-point precision but allows us to use FP16 precision, which reduces memory usage and increases the inference speed. All the results presented in this work were obtained using FP16 precision. This flag is defined at the `InferenceOp` operator. `InferenceOp` output is a probability map that represents the segmentation mask for the femur and cartilage regions in the arthroscopic images. This probability map is then used to generate the final segmentation mask by applying a threshold to the probability values. The thresholding operation converts the probability map into a binary mask, where values above the threshold are set to 1 (indicating the presence of the femur or cartilage), and values below the threshold are set to 0 (indicating the absence of the femur or cartilage). This is done at the `SegmentationPostprocessorOp` operator. The detailed configuration parameters for this operator can be found in Appendix D.1.5.

Post-processing

The inference process generates a segmentation mask for the femur and cartilage regions in the arthroscopic images. This mask is then post-processed to reshape to the correct

visualization format. We created a custom operator, the `PostImageProcessingOp`. The detailed configuration parameters for this operator can be found in Appendix D.3.

The postprocessing steps are as follows:

- **Image Resizing:** The `resize` method resizes an image to a specified size (1080x1920). The method uses OpenCV function `cv2.INTER_NEAREST` interpolation to preserve the label values.
- **Saving Mask:** The `save_mask` method then handles the saving of the mask. The mask is resized and cropped to fit the original size of 1080x1920 pixels. Padding is applied so the resized mask is correctly centered within the output frame.

Detailed implementation of these postprocessing steps can be found in Appendix D.3.

Visualization

The final step in the pipeline is to visualize the segmentation mask overlaid on the original arthroscopic image. This is done using the `HolovizOp` operator, which reads the original image and the segmentation mask and overlays the mask on the image as a semi-transparent overlay, at the end we have the visualization as shown in fig 4.6. The detailed configuration parameters for this operator can be found in Appendix D.1.6.

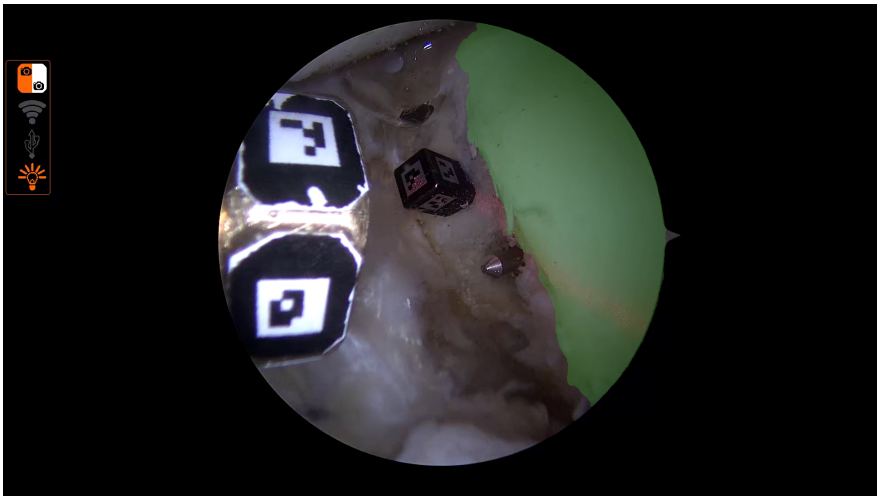

Figure 4.6: Arthrosegmentation overlay visualization, at green is represented the mask of the femur segmentation with some opacity.

4.2.2 Deployment of inside/outside knee classification model

The Inside/Outside classification model deployment was similar to the femur segmentation model, with some differences in the preprocessing steps, the model conversion and inference

process of the different nature of the models, and the visualization overlays. Keeping this in mind, we will omit some details already explained in the previous section, such as the video feed source input, and focus on the preprocessing steps, model conversion process, inference, and visualization.

Preprocessing

The preprocessing for the Inside/Outside classification model is simple since it only relies on a resize of the input images to the required size of 320x480 pixels. For this, we created a custom operator, the `ImageProcessing_in_out_Op` operator. OpenCV resize function was used to resize the image with the interpolation method `INTER_LINEAR`. The detailed code implementation of this preprocessing can be found in Appendix D.4.

Before we feed the model with the preprocessed images, we need to convert the image to float32 format and the pixel scale of 0 to 255, which is the format expected by the model. A `FormatConverterOp` operator is used to perform this conversion. For this model, we named it `preprocessor_in_out`. The detailed configuration parameters for this operator can be found in Appendix D.1.3.

Model conversion

Deep Learning Toolbox Converter for ONNX Model Format was used to convert the Matlab model to ONNX format. This toolbox supports importing and exporting ONNX models in MATLAB [61]. With this purpose, a Matlab script was developed. The script loads the model and then exports it to ONNX format. The detailed implementation of this conversion can be found in Appendix A.4. Matlab to Holoscan conversion pipeline is shown in fig 4.7.

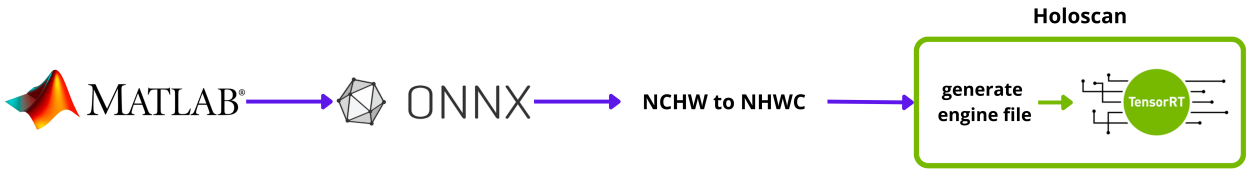

Figure 4.7: Matlab to Holoscan conversion pipeline.

Inference

Once again, we use the `InferenceOp` operator to generate the TensorRT engine and run the inference on the TensorRT engine based on our configurations. The detailed configuration parameters for this operator can be found in Appendix D.1.4.

Now, instead of using the `SegmentationPostprocessorOp` operator, we develop a custom operator, the `PostInferenceOp` operator, to post-process the output of the model because the output is a classification label, not a segmentation mask. `PostInferenceOp` operator reads the inference output tensor, extracts the values from the first two dimensions of the label tensor, compares them, and assigns the corresponding label True or False (Inside or Outside). If the first value is greater than the second value, the label is set to True. Otherwise, it is set to False. Then, it creates one output message that contains the following two fields:

- **in_out_text**: A tensor containing coordinates for the text overlay is created. The coordinates (x=0.74, y=0.04) specify the position where the text will be displayed on the image.
- **output_specs**: A list containing the text to be displayed and the color of the text. The text and color are set based on the state determined earlier ("True" or "False"). If the state is "True" (inside), the text is set to "Inside Knee: True" and the color is set to green. If the state is "False" (outside), the text is set to "Inside Knee: False" and the color is set to red.

After the operator publishes the output message to the next operator, `HolovizOp` could overlay the label on the original image. The detailed implementation for this operator can be found in Appendix D.6.

Visualization

The final step in the pipeline is to visualize the classification label overlaid on the original arthroscopic image. This is done using the `HolovizOp` operator, which reads the original image, the classification label tensor, the specifications list and overlays the label on the image as a semi-transparent overlay, at the end we have the visualization as shown in fig 4.8. The detailed configuration parameters for this operator can be found in Appendix D.1.6.

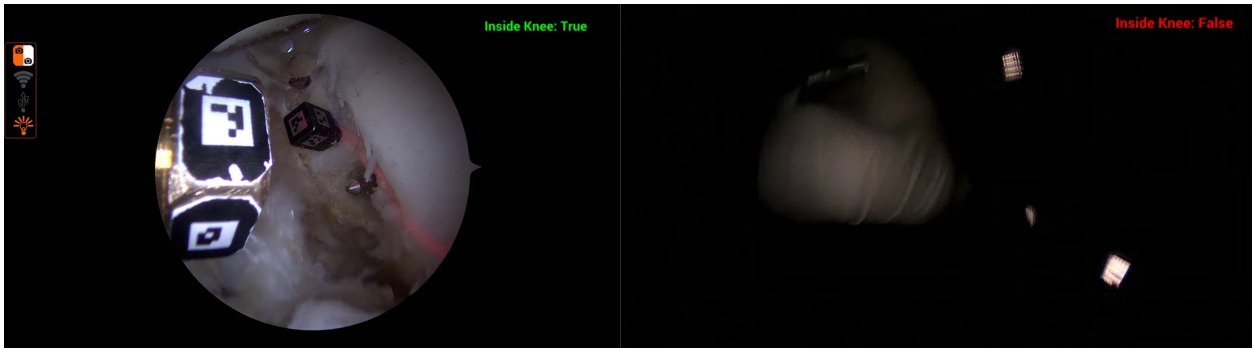

Figure 4.8: Inside/Outside classification overlay visualization. At the left image, the knee is classified as inside, and at the right image, the knee is classified as outside.

4.2.3 Deployment of lens mark visibility model

In many steps, the lens mark visibility model deployment was similar to the inside/outside knee classification model. Keeping this in mind, we will omit some details already explained and focus on the slight differences between both.

Preprocessing

The preprocessing for lens mark visibility model is simple since only relies on a resize of the input images to the required size of 180x320 pixels, for this we created a custom operator, the `ImageProcessing_lens_mark_Op` operator. OpenCV resize function was to resize the image with the interpolation method `INTER_LINEAR`. The detailed implementation for this operator can be found in Appendix D.5.

Then we again use a `FormatConverterOp` operator to convert the data to float32 format and ensure the pixel scale of 0 to 255, for this model we named it `preprocessor_lens_mark_classification`. The detailed configuration parameters for this operator can be found in Appendix D.1.3.

Model conversion

Matlab to Holoscan conversion pipeline is the same as the inside/outside knee classification model, shown before in fig 4.7.

Inference

The lens mark visibility model inference has minor differences compared to the inside/outside knee classification model. After we run the inference using the `InferenceOp` operator, we

need to post-process the model's output to generate the classification label. The detailed configuration parameters for this operator can be found in Appendix D.1.4.

So we created a new operator to distinguish the models, the `PostInference_lens_mark_classificationOp` operator. This operator reads the inference output tensor, extracts the values from the first two dimensions of the label tensor, compares them, and assigns the corresponding label True or False (Lens mark visible or not visible). If the second value is greater than the first value, the label is set to True, otherwise it is set to False. Then, it creates one output message that contains the following two fields:

- **lens_mark_text**: A tensor containing coordinates for the text overlay is created. The coordinates (x=0.74, y=0.09) specify the position where the text will be displayed on the image.
- **lens_mark_specs**: A tensor containing the text to be displayed and the color of the text. The text and color are set based on the state determined earlier ("True" or "False"). If the state is "True" (visible), the text is set to "Lens Mark visible: True" and the color is set to green. If the state is "False" (Not visible), the text is set to "Lens Mark visible: False" and the color is set to red.

After the operator publishes the output message to the next operator, `HolovizOp` could overlay the label on the original image. The detailed implementation for this operator can be found in Appendix D.5.

Visualization

The final step of the pipeline is to visualize the classification label overlaid on the original arthroscopic image, which is the same as the inside/outside knee classification model. The final visualization is shown in fig 4.9. The detailed configuration parameters for this operator can be found in Appendix D.1.6.

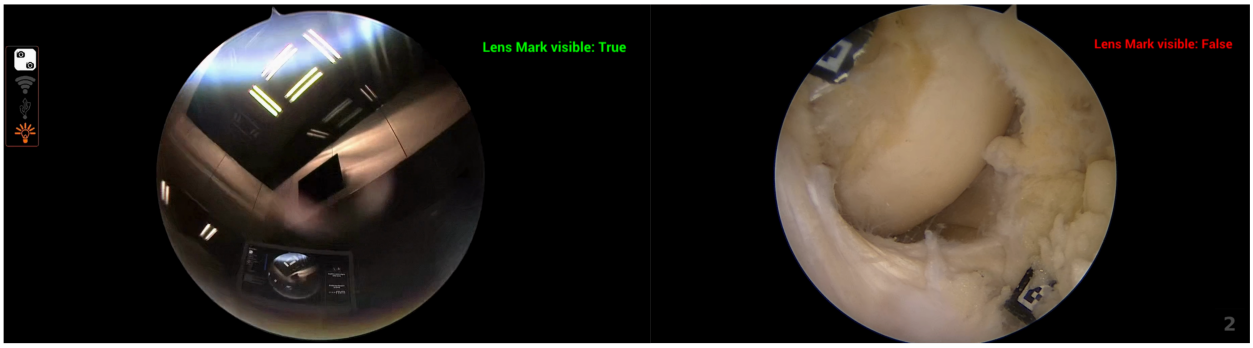

Figure 4.9: Lens mark classification overlay visualization. At the left image, the lens mark is classified as visible, and at the right image, the lens mark is classified as not visible.

4.3 All Models Deployment

To test the capabilities of the Holoscan framework and develop a more complex application, we decided to deploy all three models simultaneously. The deployment process involved using the previously described methods and others, like defining a map of connections between the operators and mapping inference inputs and outputs to the corresponding model preprocessing and postprocessing operators. For visualization, we developed an intermediary operator, the `visualizerOp`, to handle the overlay of the models. This operator takes the lens mark visibility model, inside/outside knee classification model text labels and specifications, and combines them in a single tensor in order `HolovizOp` operator can overlay all the labels on the original image. The detailed operator implementation can be found in Appendix D.7. The final visualization is shown in fig 4.10.

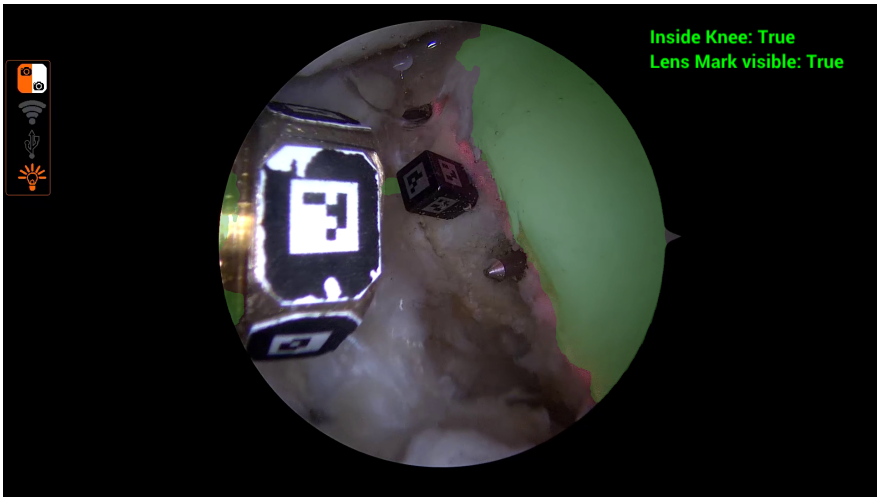

Figure 4.10: All models simultaneously visualization.

4.4 Summary

In this chapter, we presented the deployment of the DL models in the NVIDIA Holoscan framework. We described the steps taken to integrate the models into the Holoscan framework, including preprocessing the input images, converting the models to TensorRT format, and the inference process. We also discussed deploying the models using different video sources, such as video files and the AJA Corvid 44 12G BNC capture system. The deployment process involved converting the models to a format compatible with Holoscan, optimizing the models for real-time processing, and visualizing the results. The deployment of the models in the Holoscan framework enables real-time processing of arthroscopic images and provides valuable insights for orthopedic surgeons during surgery. The next chapter will present the deployment process results and discuss the models' performance in a real-time surgical environment.

5 Experimental Validation

After achieving the final application deployment of the models in the Nvidia Holoscan, the next step is to evaluate the system's performance. For this purpose, several performance analysis tests were carried out in the laboratory to validate the performance of the developed system and compare the evaluation between the original and deployed models. This application was developed and tested at version 0.6.0 of the Holoscan SDK, but we tested the compatibility on version 2.4.0, showing that the software is still in development and has a lot of room for improvement. This chapter presents the different experimental configurations to be evaluated in this work, the different evaluation metrics, and the tools used to obtain the results. Based on the results obtained, an analysis is carried out, and a discussion about the work is launched.

5.1 Evaluation Methodology

The system's evaluation was conducted in two phases. The first phase focused on assessing the performance of the individual models, while the second phase evaluated the overall system performance. The evaluation of individual models involved comparing the inference outputs of the original models with the deployed versions to ensure that the outputs were as identical or similar as possible. Each pipeline stage was meticulously evaluated, with custom debug operators inserted between stages to guarantee consistency in input, preprocessing, postprocessing, and output between the original and deployed models.

To evaluate the performance of the models, we utilized several key metrics. The *Dice Score* was used for the femur segmentation model, while *accuracy*, *recall*, and *precision* were employed for the inside/outside knee classification model and the lens mark visibility model. Additionally, the *Mean Absolute Error (MAE)* was selected as a metric to assess the differences in continuous data between the original and deployed models. MAE is beneficial in this context because it provides a straightforward interpretation of the average magnitude

of errors between corresponding outputs without considering their direction [62].

The *Dice Score* is a widely used metric for evaluating the similarity between two samples, particularly in segmentation tasks [63]. It is defined as:

$$DiceScore = \frac{2 \times |A \cap B|}{|A| + |B|} \tag{5.1}$$

Where A represents the ground truth, and B is the predicted output. The Dice Score ranges from 0 to 1, where 0 indicates no overlap between the two samples, and 1 indicates a perfect overlap. This metric was explicitly applied to assess the femur segmentation model.

For the inside/outside knee classification model and the lens mark visibility model, the metrics of *accuracy*, *recall*, and *precision* were used. These metrics are defined as follows:

Accuracy is the proportion of total correct predictions (both true positives and true negatives) out of all predictions made [64].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2}$$

Recall measures the proportion of actual positives that were correctly identified by the model. It focuses on the ability of the model to find all relevant cases within a dataset [64].

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

Precision measures the proportion of predicted positives that are actually positive. It reflects the accuracy of the positive predictions made by the model [64].

$$Precision = \frac{TP}{TP + FP} \tag{5.4}$$

Where TP is the number of true positive predictions, TN is the number of true negative predictions, FP is the number of false positive predictions, and FN is the number of false negative predictions.

The Mean Absolute Error (MAE) is defined as:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{5.5}$$

The overall system performance was evaluated by analyzing the application speed and latency when running the models individually and simultaneously. To reference 70 ms was the latency fixed as the maximum acceptable latency for the system to be used in a clinical setting and 24 fps was the minimum frame rate required for the system to be considered real-time. Different approaches were used to achieve this, depending on the input source.

5.1.1 Video file source evaluation

To measure the system's performance using a video file as the source, we utilized Nvidia Holoscan's built-in latency measurement tool. However, this tool cannot be used with the AJA capture card as the source. Nvidia Visual Profiler was also used because it provided a comprehensive view of each pipeline stage. This tool enabled us to calculate the frames per second (fps), representing the number of frames the entire pipeline can process within one second.

5.1.2 AJA source evaluation

When the AJA capture card was used as the source, a standard and practical method was implemented to measure the system's latency and frame rate. This method involved playing a video with a clock ticking at 60 fps and observing the delay between the original clock video and its display on the Holoscan visualization screen. The difference in clock ticks between the two screens allowed us to determine the system's latency in terms of frames. Additionally, the frame rate was determined by counting the number of skipped clock ticks displayed by the Holoscan system. Fig 5.1 illustrates the complete system configuration for measuring the system's latency and frame rate.

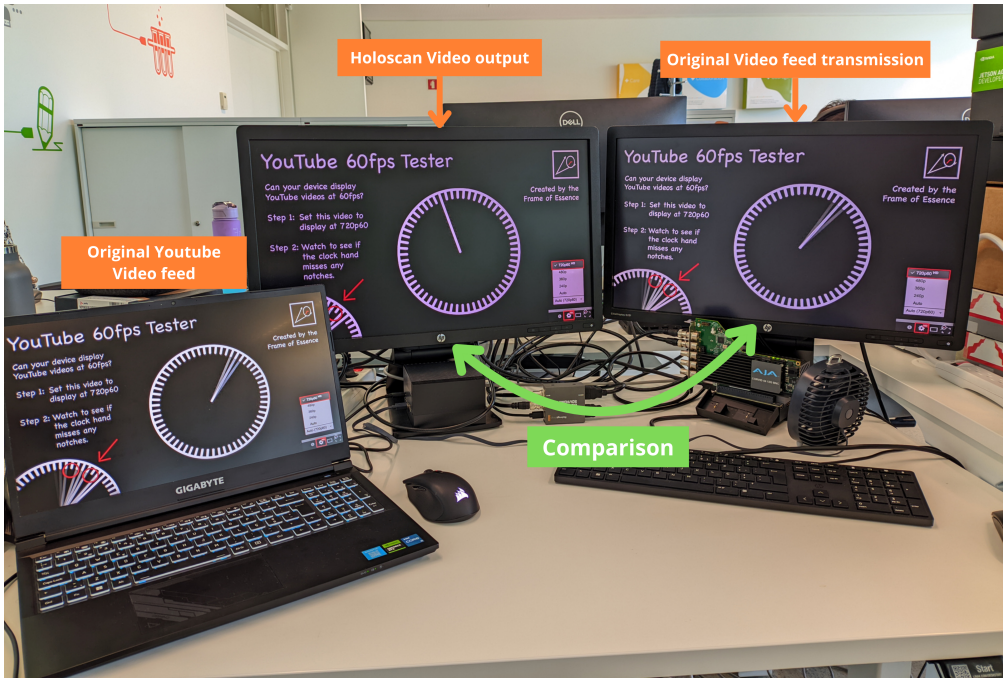

Figure 5.1: System configuration used to measure the latency and frame rate via AJA source. The computer video output on the left is mirrored on the visualization screen on the right, with the Holoscan inference output displayed in the center. The delay between the original clock video and the Holoscan visualization screen is noticeable.

Latency was measured in milliseconds (ms), representing the time required for the system to process and display a frame on the screen. The frame rate, measured in fps, was calculated using the Nvidia Visual Profiler and represents the number of frames the pipeline can process per second. Both metrics were recorded for each model individually and for the system when all models ran simultaneously.

5.1.3 Models validation datasets

For the model's validation, the data used is composed of 3 frames from each of 7 different video sequences taken from a real cadaver lab experiment. We have chosen these frames to be the most challenging ones for the models and be a generalization of all the scenarios as being inside/outside of the knee, inside the tunnel, and the lens mark visible/not visible.

5.2 Evaluate Femur Segmentation Model

To validate the ONNX conversion of the femur segmentation model, we compared the inference outputs between the original PyTorch model and the deployed ONNX model. The

validation process involved performing inference on the seven datasets.

The Dice score, which measures the similarity between the segmentation outputs, was calculated for each pair of masks generated by the original and the ONNX models. A Dice score of 1 was obtained for all dataset images, confirming that the outputs were identical. This indicates that the model was correctly converted from PyTorch to ONNX without any loss of accuracy. The table attached in Appendix B.1 presents the Dice score comparison between the original PyTorch and the ONNX models across the datasets. This result further validates the correctness of the ONNX conversion process and ensures that the model's accuracy and performance are preserved post-conversion.

5.2.1 Pytorch model and Holoscan model outputs comparison

The next step was to compare the inference outputs between the original PyTorch model and the Holoscan deployed model. The same Python preprocessing steps were applied to the images fed into the Holoscan model to have credible results. The comparison was performed using the same seven datasets. The Dice score was calculated for each pair of masks generated by the original and the Holoscan models. The Dice score results calculated between the original PyTorch and the Holoscan models, with the same preprocessing, across the datasets are shown in figure 5.2.

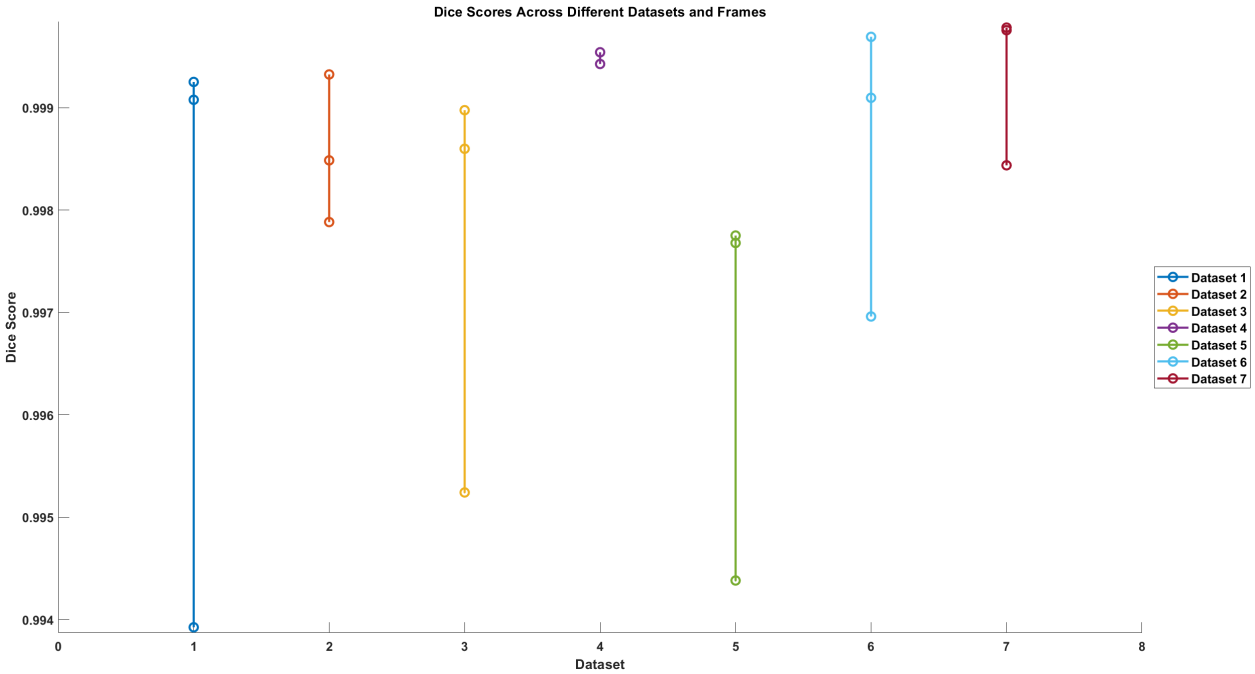

Figure 5.2: Dice Score Comparison Between Original PyTorch Model and Holoscan Model for Femur Segmentation. A circle represents each frame, and a different color represents each dataset. The X-axis represents the datasets, and the Y-axis represents the Dice score.

The results show that the Dice score is close to 1 for all frames, indicating that the segmentation outputs from the Holoscan model are identical to those from the original PyTorch model. As the first dataset's first frame has no femur, the model output is a black image, which leads to a Dice score of Nan (not a number). Some precision model calculations can explain the small under 0.1 percent variations in the Dice score. This confirms that the model was correctly deployed in the Holoscan platform, the preprocessing steps were correctly implemented in the Holoscan model, and the conversion process was successfully done.

5.2.2 Performance optimization and evaluation

Following this validation, it was recognized that preprocessing, along with inference, is one of the most performance-demanding tasks, as mentioned in the deployment section E.2. To address this, the preprocessing stage was optimized by developing a C++ script that leverages CUDA to perform the CLAHE operation on the GPU. This optimization was identified as beneficial through Amdahl's Law at section 5.2.2.

Using a video file as the source, we measured the latency across 100 frames using the Holoscan latency tool. The average latency for the original Python preprocessing was about 197 ms, whereas the optimized C++ CUDA preprocessing reduced this latency to approximately 42 ms. Regarding frame rate, the Python preprocessing achieved close to 5 fps, while the optimized C++ CUDA preprocessing significantly improved to around 73 fps. Consequently, the overall application frame rate increased from approximately 5 fps to 24 fps. Table 5.1 compares the preprocessing performance between the original Python implementation and the optimized C++ CUDA implementation.

Metric	**Python Preprocessing**	**C++ CUDA Preprocessing**
Preprocessing Frame Rate (fps)	4.62 fps	72.92 fps
Overall Application Latency (ms)	197.24 ms	41.53 ms
Overall Application Frame Rate (fps)	5.06 fps	24.07 fps

Table 5.1: Preprocessing Performance Comparison, preprocessing frame rate refers to the preprocessing stage only, overall application latency is the time taken to process a frame and display it on the screen, and overall application frame rate is the number of frames processed by the pipeline in one second.

This optimization was crucial to improve the overall system performance. It represented a 4.6x speedup in latency and fps, compared with Amdahl's Law prediction of a maximum

of 5.54x speedup. The results demonstrate the effectiveness of the C++ CUDA optimization in enhancing the system's performance and efficiency, the difference between the theoretical and the practical speedup results can be attributed to the overhead of data transfer between the CPU and GPU, still the speedup achieved was significant. By analyzing these results, we conclude that the Holoscan pipeline only processes one frame at a time, so the fps is correlated with the latency. Confronting the NVIDIA team with these results, they confirmed that the Holoscan pipeline is not optimized to process multiple frames simultaneously, which could lead to better performance, and it is a feature that will be implemented in the future.

As the C++ preprocessing operations are not the same as the Python preprocessing operations, mainly the CLAHE CUDA operation compared to the CPU CLAHE operation, qualitatively, the inferences outputs compared between the original PythoPythonrocessing and the optimized C++ CUDA preprocessing are not the same. The Dice score was calculated for each pair of masks generated by the original and the optimized C++ CUDA preprocessing. The results are shown in figure 5.3.

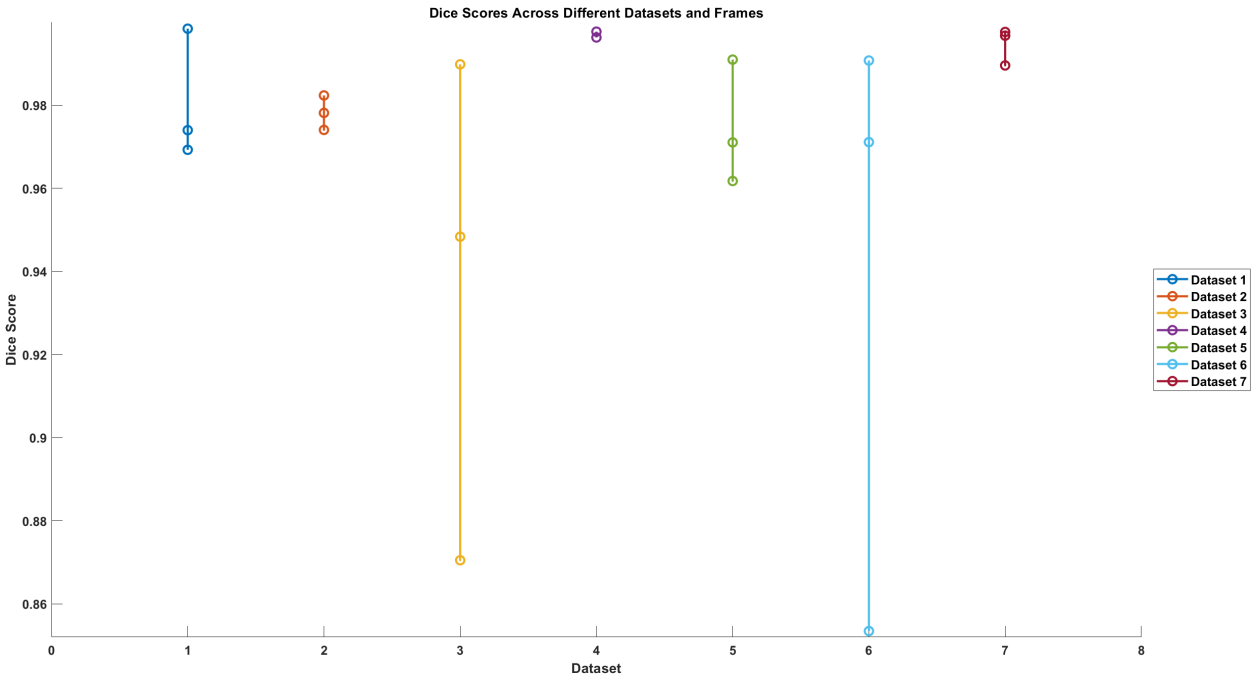

Figure 5.3: Dice Score Comparison Between Original Python Preprocessing and Optimized C++ CUDA Preprocessing for Femur Segmentation. A circle represents each frame, and a different color represents each dataset. The X-axis represents the datasets, and the Y-axis represents the Dice score.

Most of the frames have a Dice score above 0.9, indicating a good level of similarity between the segmentation outputs. Some outliers exist with a maximum Dice score difference

of 0.14, which can be attributed to the differences in preprocessing. We visually evaluated the segmentation outputs and observed that the optimized C++ CUDA preprocessing produced slightly different images than the original Python preprocessing. This difference led to a slight variation in the segmentation outputs, resulting in a lower Dice score. Despite this, the segmentation outputs were still accurate and preserved the femur's boundaries, indicating that the preprocessing optimization did not significantly affect the model's performance. At Fig 5.4 we can observe the visual comparison between the original Python preprocessing and the optimized C++ CUDA segmentation outputs.

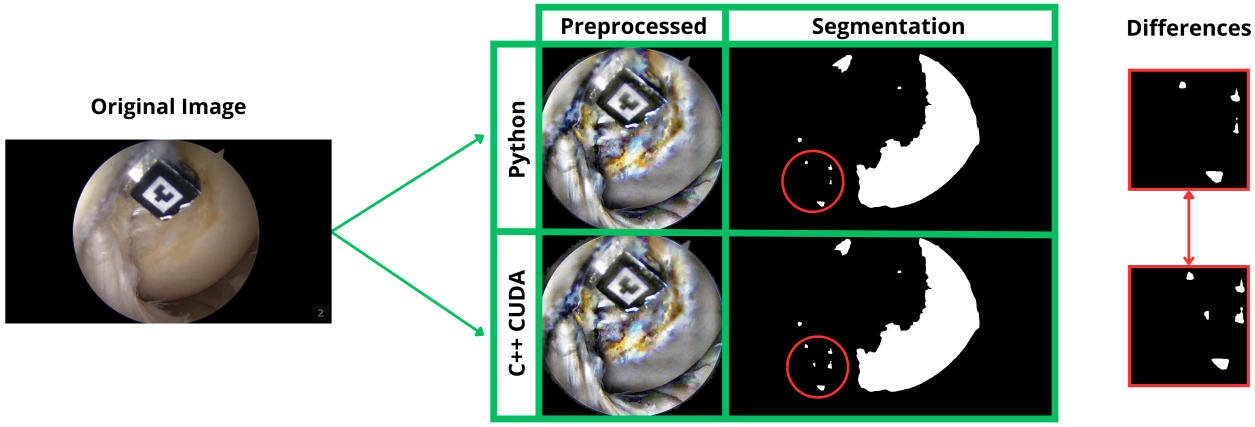

Figure 5.4: Visual Comparison Between Original Python Preprocessing and Optimized C++ CUDA Preprocessing for Femur Segmentation. The first column shows the original image, the second column shows the preprocessed outputs from both methods and the third column shows the segmentation outputs from both methods.

We conclude that the significant improvement in latency and frame rate achieved through the C++ CUDA optimization outweighed the minor differences in the segmentation outputs.

Using AJA as the source and C++ CUDA preprocessing established as our preprocessing method, the latency and frame rate were measured using the method described at the beginning of this section. Recording a slow motion video of both screens was observed a frame rate of approximately 20 fps (the Holoscan visualization only updated the screen every three frames) and average latency between approximately 50 ms and 83 ms (3 to 5 frames). At the Fig 5.5, we can observe, a schematic representation of the timeline, simulating a side-by-side comparison of the screens stages.

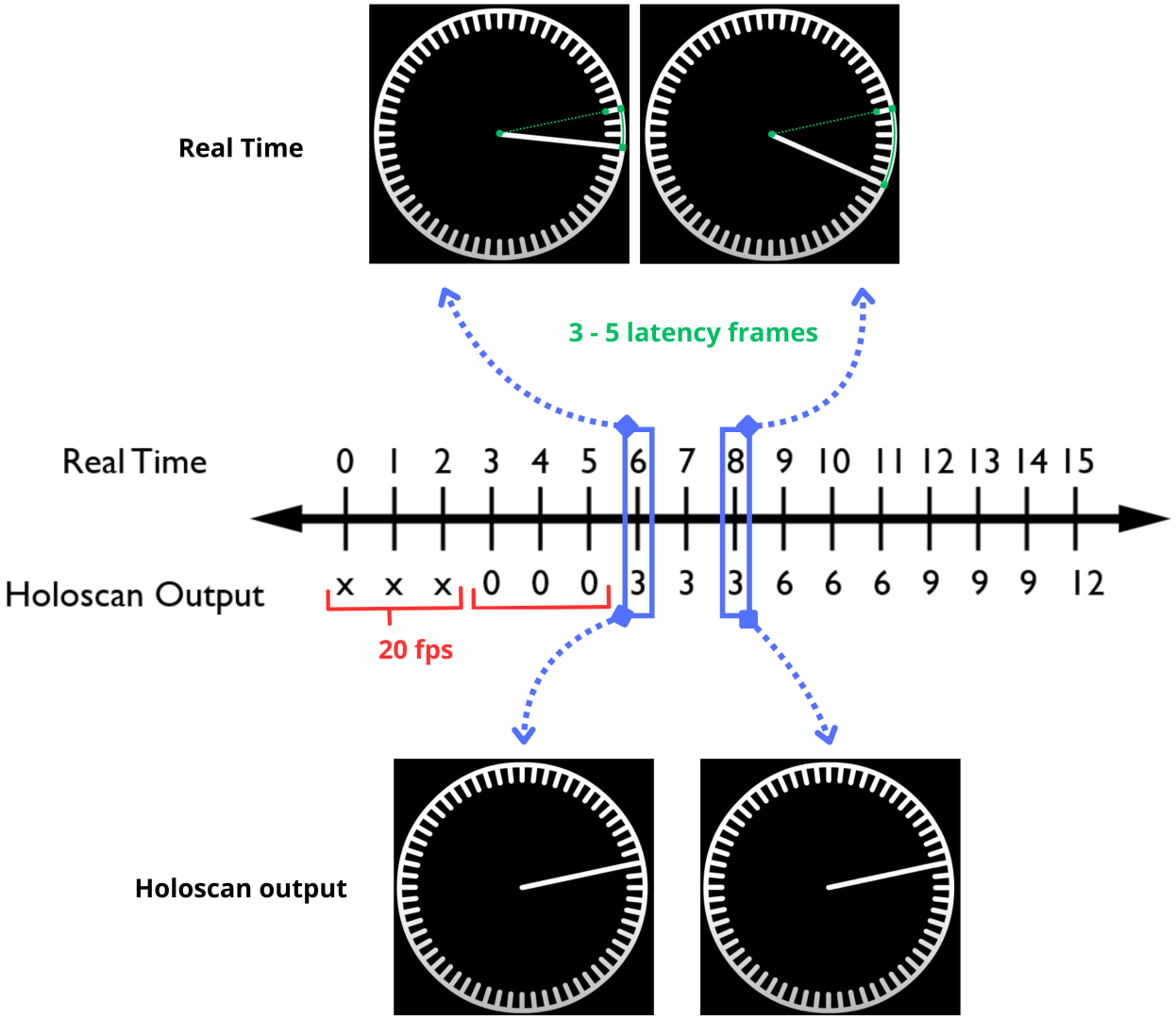

Figure 5.5: Schematic representation of the results observed using AJA as the source. The above screen is the real-time video output at 60 fps, and the below screen is the Holoscan visualization, in the middle we see the timeline representation. Green lines represent the minimum and maximum latency points observed. Red brackets represent the ticking rate of the Holoscan visualization screen.

As observed in the figure 5.5, the Holoscan visualization screen is updated every three frames, which leads to a frame rate of 20 fps. When the real-time screen displays frame number 8, the Holoscan visualization screen shows frame number 3, representing the maximum observed latency point. Then, Holoscan ticks and displays frame number 6, the minimum latency point observed. This leads to a latency between the two screens of approximately 3 to 5 frames, which corresponds to approximately 50 ms to 83 ms. Keeping in mind that the femur segmentation model is the most computationally demanding model in the pipeline and that Perceive3D before this work had a 12 fps offline system, the results obtained are promising, indicating that the system can be further evaluated in clinical settings.

5.3 Evaluate Inside/Outside Knee Classification Model

This section will describe the evaluation process of the inside/outside knee classification model.

5.3.1 Matlab to ONNX conversion

The first step of the evaluation process was to verify if the ONNX converted model output was equal to the Matlab original model output. Exporting the Matlab model to ONNX and importing it proved to us that the conversion was done correctly. These results are described in detail in Appendix B.2 comparing the accuracy, precision, and recall of the original Matlab and ONNX models across all datasets. The results show that the accuracy, precision, and recall are equal for the original and deployed models, indicating that the model was correctly converted to ONNX.

5.3.2 Matlab model and Holoscan model outputs comparison

Comparing the inference output between the original Matlab and Holoscan deployed models, we observed that the outputs were inconsistent with the ONNX conversion tests. The image preprocessing was the only thing that was susceptible to modifying the Holoscan performance model. The only preprocessing done to the image was the resizing, so a comparison between the resizing function of Matlab and OpenCV was performed. To make this comparison, SAD (Sum of Absolute Differences) was calculated between the Matlab and OpenCV resizing outputs. The SAD is a method that calculates the absolute differences between the pixel values of two images and sums them up. The lower the SAD value, the more similar the images are.

$$SAD = \sum_{i=1}^{n} |I1(i) - I2(i)| \tag{5.6}$$

Where $I1$ and $I2$ are the pixel values of the two images, and n is the total number of pixels in the image. The SAD was calculated for one frame to prove the difference in the resizing function between Matlab and OpenCV. The SAD value was 20831, which indicates that the images are different. The figure 5.6 below shows a visual color map highlighting the differences between the Matlab and OpenCV resizing outputs.

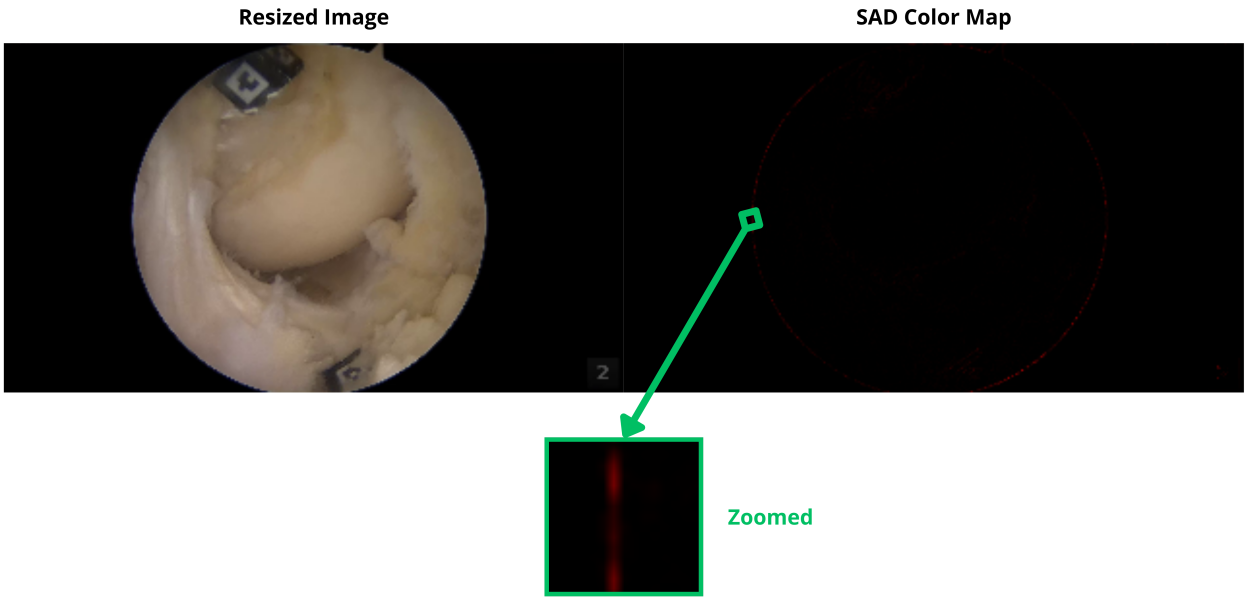

Figure 5.6: At the left, the resized image, and at the right, the Color map shows the differences between the Matlab and OpenCV resizing outputs. The color map makes it possible to see the differences between the two images, where the red color indicates the highest differences. The same interpolation method was used in both Matlab and OpenCV, but the output has noticeable differences.

Considering that, to perform a fair comparison between Matlab and Holoscan models, the Matlab preprocessed image was saved and used as GT (ground truth) to feed in Holoscan. As expected, the output was the same as the Matlab output. The results are described in detail at Appendix B.3. This evaluation allowed us to conclude that the models were correctly converted to ONNX, confirming that the Matlab resize function and openCV resize function have different algorithms, which leads to a difference in the output, even using the same interpolation methods. In order to estimate the difference between the Holoscan and Matlab model outputs, we used the MAE metric. The MAE was calculated for each dataset, and the results are shown in table 5.2. The MAE values are shallow, indicating that the Holoscan model output is similar to the Matlab model output. This confirms that the model was correctly deployed in the Holoscan platform.

Dataset	1	2	3	4	5	6	7
In/Out MAE	0.0001	0.0001	0.0003	0.0000	0.0001	0.0012	0.0003

Table 5.2: Mean Absolute Error (MAE) for Holoscan vs Matlab Inside/Outside Model Outputs

5.3.3 Performance evaluation

The performance evaluation of the individually running inside/outside knee classification model was conducted using the same methodology as the femur segmentation model.

- Using a video file as the source, the average latency across 100 frames was approximately 13 ms, and the frame rate was around 75 fps.
- Using the AJA source as the input and the practical method described in the femur segmentation model evaluation section 5.2.2, the latency observed was approximately 13 ms (1 frame), and the frame rate was around 60 fps (the Holoscan visualization frame rate was the same as the video).

These results indicate that the model ran efficiently on the Holoscan platform and achieved real-time performance. Due to the model's simplicity, we can see the difference in the frame rate compared to the femur segmentation model.

5.4 Evaluate Lens Mark Visibility Model

This section describe the evaluation process of the lens mark visibility model.

5.4.1 Matlab to ONNX conversion

The first step of the inside/outside knee classification model evaluation process was again to observe if the ONNX converted model output was equal to the Matlab original model output. As the original model was developed in Matlab, the same problem of the resizing functions was found. The same solution was applied to this model, and the metrics were consistent, with the accuracy, precision, and recall being equal for the original Matlab and ONNX models across all datasets. The results are at described it detail at Appendix B.4. This evaluation allows us to conclude that the models were correctly converted to ONNX.

5.4.2 Matlab model and Holoscan model comparison

As shown in Table 5.3, the Holoscan model exhibits lower accuracy on the third dataset, primarily due to a false negative where the model failed to detect a lens mark. This result impacts both precision and recall, highlighting that the model does not perform with perfect

precision or recall in this case. As demonstrated earlier, the Holoscan model is sensitive to changes in the input image and calculation precision, which affects its overall performance.

	Dataset						
	1	2	3	4	5	6	7
Accuracy	1.000	1.000	0.667	1.000	1.000	1.000	1.000
Precision	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Recall	1.000	1.000	0.667	1.000	1.000	1.000	1.000

Table 5.3: Comparison of Accuracy, Precision, and Recall for Holoscan Lens Mark Model across all Datasets

The MAE was also calculated for each dataset, as presented in Table 5.4. Although the MAE values are relatively low, indicating that the Holoscan model's output is similar to the MATLAB model's output, they are slightly higher when compared to the inside/outside model. This increase in MAE further supports the notion that the model is sensitive to variations in input images and precision calculations. Despite this sensitivity, the model was successfully deployed on the Holoscan platform.

	1	2	3	4	5	6	7
LensMark MAE	0.0301	0.0233	0.0686	0.0222	0.0104	0.0033	0.0173

Table 5.4: Mean Absolute Error (MAE) for Holoscan vs Matlab Lens Mark Model Outputs

5.4.3 Performance evaluation

The performance evaluation of the individually running lens mark visibility model was conducted using the same methodology as the inside/outside knee classification model and the femur segmentation model.

- Using a video file as the source, the average latency across 100 frames was approximately 17 ms (1 frame), and the frame rate was around 59 fps.
- Using the AJA source as the input and the practical method described in the femur segmentation model evaluation section 5.2.2, the latency observed was approximately 17 ms (1 frame), and the frame rate was around 60 fps (the Holoscan visualization frame rate was the same as the video).

These results indicate that the model ran efficiently on the Holoscan platform and achieved real-time performance. We can see the difference in the frame rate between the models due to the model's simplicity.

5.5 Evaluate System Running All DL Models Simultaneously

With the individual model's accuracy and performance validated, the final evaluation involved running all three DL models simultaneously on the Holoscan platform. The models ran in parallel, and the system's performance was evaluated in terms of latency and frame rate.

- Using a video file as the source, the average latency across 100 frames was approximately 53 ms, and the frame rate was around 19 fps.
- Using the AJA source as the input and the practical method described in the femur segmentation model evaluation section 5.2.2, the latency observed was approximately between 67 ms to 116 ms (4 to 7 frames), and the frame rate was around 15 fps.

These results indicate that the system ran efficiently with all models running simultaneously, paralleling the individual models' performance. The decrease in frame rate and the increase in latency compared to the individual models was expected due to the increased computational load on the system. However, the system proved to be capable of processing all models simultaneously in real-time, demonstrating its efficiency.

5.6 AJA board and Keyer Evaluation

To evaluate the final system, an additional test was conducted to determine if the AJA board and Holoscan could support 4K high-definition video. The test was successful, with the system able to process 4K video in real-time, although at a lower frame rate compared to 1080p video. This demonstrates the potential of the system to scale, particularly when using platforms like NVIDIA IGX, which could offer improved performance in the future.

As previously mentioned, the keyer functionality in Holoscan could not be implemented due to compatibility issues between the RDMA drivers and Holoscan. Since the system was running on an iGPU, the AJA RDMA drivers were not updated to be compatible. This

issue has been recognized by both the NVIDIA SDK and AJA teams, and a fix is expected in future releases [58]. During the testing phase, the AJA board was damaged due to an unrelated hardware issue. The board was promptly replaced, and testing resumed.

5.7 Summary

The system's performance was evaluated in a controlled lab environment, confirming that the models were accurately converted to ONNX and deployed on the Holoscan platform without losing accuracy. The femur segmentation model achieved a Dice score close to 1, and the inside/outside knee classification and lens mark visibility models achieved near-perfect accuracy, precision, and recall. Using Holoscan's built-in latency tool as reference metric for a video file as source, performance metrics showed the femur segmentation model at around 24 fps, the knee classification model at around 75 fps, and the lens mark visibility model at around 59 fps. Running all models simultaneously resulted in a frame rate of approximately 15 fps. The keyer functionality was not implemented due to hardware compatibility issues with the AJA board. 4K video was successfully processed in the system with a lower frame rate compared to 1080p video. Overall, the system performed satisfactorily, achieving real-time performance.

6 Conclusions

This dissertation addressed the challenge of implementing real-time AI models for arthroscopic surgery in an edge computing environment, contributing to advancing surgical navigation technologies. Through the integration of cutting-edge hardware like the NVIDIA Jetson AGX Orin and innovative software frameworks such as Holoscan, the research explored novel approaches to enhance precision and efficiency in minimally invasive surgeries. The results of this work demonstrate the potential of AI to improve real-time decision-making in complex medical settings, successfully showing that the proposed system have the potential to be tested in real clinical settings.

6.1 Main results

This work successfully developed and deployed real-time AI models for surgical navigation in arthroscopy, focusing on femur segmentation, inside/outside knee classification, and lens mark visibility. By utilizing the NVIDIA Holoscan framework and edge computing platforms, the system achieved real-time performance with notable accuracy and efficiency. The deployed models demonstrated relatively good frame rates and latency for the individual models and acceptable when running all models concurrently. Significant optimizations in preprocessing using CUDA reduced latency to approximately 41 ms, and the femur segmentation model achieved near-perfect accuracy with a Dice score close to 1, showcasing the practical application of the research and its potential to improve surgical procedures.

Despite technical challenges, such as issues with the AJA board and minor differences between Python and CUDA preprocessing outputs, the system met the stringent demands of real-time surgical applications. These results offer a compelling demonstration of how AI can be effectively implemented in real-time medical applications, potentially improving the accuracy and safety of arthroscopic procedures. Holoscan has been proven to have the potential to be used in a clinical setting, but it still needs improvements. When we started

this work, the software was at version 0.6.0, and seven realises later, it is at version 2.4.0, showing that the software is still in development and has a lot of room for improvement.

6.2 Future Work

Future work should further optimize the Holoscan pipeline, particularly in enabling the simultaneous processing of multiple frames, which remains a performance bottleneck. Expanding the AI models to other surgical procedures and anatomical regions is another promising direction to enhance the system's applicability. Improving the robustness of the models to handle variable real-world conditions, such as different lighting or occlusion scenarios, is critical for real-world deployment. Furthermore, clinical validation is essential, requiring rigorous testing of the system's safety, reliability, and integration into existing surgical workflows. Scalability is another crucial area for future research, with the potential to extend the system's capabilities by deploying on more powerful platforms like NVIDIA IGX Orin and Clara AGX for medical environments. The foundation laid by this research opens the door to future advancements, with substantial potential for AI-driven surgical navigation in clinical practice.

7 Bibliography

[1] Filippo Pesapane, Marina Codari, and Francesco Sardanelli. Artificial intelligence in medical imaging: threat or opportunity? radiologists again at the forefront of innovation in medicine. *European radiology experimental*, 2:1–10, 2018.

[2] Rohan Shad, John P. Cunningham, Euan A. Ashley, Curtis P. Langlotz, and William Hiesinger. Designing clinically translatable artificial intelligence systems for high-dimensional medical imaging. *Nature Machine Intelligence*, 3(11):929–935, November 2021.

[3] Karen A Cullen, Margaret J Hall, and Aleksandr Golosinskiy. Ambulatory surgery in the united states, 2006. *Natl Health Stat Report*, 1(11):1–25, 2009.

[4] Reed A.C. Siemieniuk, Ian A. Harris, Thomas Agoritsas, et al. Arthroscopic surgery for degenerative knee arthritis and meniscal tears: a clinical practice guideline. *BMJ*, 357:j1982, 2017. Published 2017 May 10.

[5] Muyibat A Adelani, Alex H Harris, Thomas R Bowe, and Nicholas J Giori. Arthroscopy for knee osteoarthritis has not decreased after a clinical trial. *Clinical Orthopaedics and Related Research*, 474:489–494, 2016.

[6] Megan A Bohensky, Vijaya Sundararajan, Nick Andrianopoulos, and et al. Trends in elective knee arthroscopies in a population-based cohort, 2000-2009. *Medical Journal of Australia*, 197:399–403, 2012.

[7] David F Hamilton and Colin R Howie. Knee arthroscopy: influence of systems for delivering healthcare on procedure rates. *BMJ*, 351:h4720, 2015.

[8] Jonas Bloch Thorlund, Kristoffer Borbjerg Hare, and L Stefan Lohmander. Large increase in arthroscopic meniscus surgery in the middle-aged and older population in denmark from 2000 to 2011. *Acta Orthopaedica*, 85:287–292, 2014.

[9] Micah S Ngatuvai et al. Epidemiological comparison of acl injuries on different playing surfaces in high school football and soccer. *Orthopaedic journal of sports medicine*, 10(5):23259671221092321, 2022.

[10] Carolina Raposo, Cristovao Sousa, Luis Ribeiro, Rui Melo, Joao P. Barreto, Joao Oliveira, Pedro Marques, and Fernando Fonseca. Video-based computer aided arthroscopy for patient specific reconstruction of the anterior cruciate ligament, 2018.

[11] Michael J Strobel. *Manual of arthroscopic surgery*. Springer Science & Business Media, 2013.

[12] Robert F LaPrade, Tim Spalding, Iain R Murray, Jorge Chahla, Marc R Safran, Christopher M Larson, Scott C Faucett, Richard von Bormann, Robert H Brophy, Rodrigo Maestu, Aaron J Krych, Ponky Firer, and Lars Engebretsen. Knee arthroscopy: evidence for a targeted approach. *British Journal of Sports Medicine*, 55(13):707–708, 2021.

[13] Danyal H Nawabi, Nabil Mehta, Peter Chamberlin, Chisa Hidaka, Yile Ge, Ting Jung Pan, and Stephen Lyman. Learning curve for hip arthroscopy steeper than expected. *Journal of Hip Preservation Surgery*, 3(suppl_1):hnw030–007, 2016.

[14] Gonzalo Samitier, Alejandro I Marcano, Eduard Alentorn-Geli, Ramon Cugat, Kevin W Farmer, and Michael W Moser. Failure of anterior cruciate ligament reconstruction. *Archives of bone and joint surgery*, 3(4):220, 2015.

[15] Justin D Krogue, Kaiyang V Cheng, Kevin M Hwang, Paul Toogood, Eric G Meinberg, Erik J Geiger, Musa Zaid, Kevin C McGill, Rina Patel, Jae Ho Sohn, et al. Automatic hip fracture identification and functional subclassification with deep learning. *Radiology: Artificial Intelligence*, 2(2), 2020.

[16] Saeid Asgari Taghanaki, Kumar Abhishek, Joseph Paul Cohen, Julien Cohen-Adad, and Ghassan Hamarneh. Deep semantic segmentation of natural and medical images: a review. *Artificial Intelligence Review*, 54:137–178, 2021.

[17] E.S. Deol, G. Henning, S. Basourakos, et al. Artificial intelligence model for automated surgical instrument detection and counting: an experimental proof-of-concept study. *Patient Saf Surg*, 18:24, 2024.

[18] Huanhuan Zhang and Yufei Qie. Applying deep learning to medical imaging: A review. *Applied Sciences*, 13(18), 2023.

[19] Deya Chatterjee. The rise of deep learning in radiology: An overview of recent research. *International Journal for Research in Applied Science and Engineering Technology*, 7:2353–2361, 06 2019.

[20] Xiaofei Wang, Yiwen Han, Victor C. M. Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(2):869–904, 2020.

[21] NVIDIA holoscan. `https://www.nvidia.com/en-us/clara/holoscan/`. Acessed: 2024-01-15.

[22] Nvidia clara holoscan: Real-time ai computing for medical devices. `https://developer.download.nvidia.com/CLARA/Endoscopy-Whitepaper-Clara-Holoscan.pdf`. Accessed: 2024-02-13.

[23] Monai: Medical open network for ai. `https://monai.io/`. Accessed: 2024-01-16.

[24] NVIDIA holoscan. `https://docs.nvidia.com/holoscan/sdk-user-guide/index.html`. Acessed: 2024-01-22.

[25] NVIDIA holoscan. `https://github.com/nvidia-holoscan/.github`. Acessed: 2024-01-22.

[26] Vikas Khanduja. Arthroscopy: Past, present, and the future. *J Arthrosc Surg Sport Med*, 1(1):3–4, 2020.

[27] Eirik Solheim, Torbjørn Grøntvedt, Anders Mølster, Gisle Uppheim, Caryl Gay, and Sigbjørn Dimmen. Milestones in the early history of arthroscopy. *Journal of Orthopaedic Reports*, 1(3):100060, 2022.

[28] T. Baptista, M. Marques, C. Raposo, and et al. Structured light for touchless 3d registration in video-based surgical navigation. *Int J CARS*, 19:1429–1437, Jul 2024. Received 08 March 2024; Accepted 07 May 2024; Published 30 May 2024.

[29] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[30] Nahian Siddique, Sidike Paheding, Colin P. Elkin, and Vijay Devabhaktuni. U-net and its variants for medical image segmentation: A review of theory and applications. *IEEE Access*, 9:82031–82057, 2021.

[31] Hitesh Chopra, Annu, Dong K. Shin, Kavita Munjal, Priyanka, Kuldeep Dhama, and Talha B. Emran. Revolutionizing clinical trials: the role of ai in accelerating medical breakthroughs. *International Journal of Surgery*, 109(12):4211–4220, 2023.

[32] James R Hawkins, Marram P Olson, Ahmed Harouni, Ming Melvin Qin, Christopher P Hess, Sharmila Majumdar, and Jason C Crane. Implementation and prospective real-time evaluation of a generalized system for in-clinic deployment and validation of machine learning models in radiology. *PLOS Digital Health*, 2(8):e0000227, 2023.

[33] Pietro Mascagni, Deepak Alapatt, Alfonso Lapergola, Armine Vardazaryan, Jean-Paul Mazellier, Bernard Dallemagne, Didier Mutter, and Nicolas Padoy. Real-time artificial intelligence assistance for safe laparoscopic cholecystectomy: Early-stage clinical evaluation, 2022.

[34] Jasper Hofman, Pieter De Backer, Ilaria Manghi, Jente Simoens, Ruben De Groote, Hannes Van Den Bossche, Mathieu D'Hondt, Tim Oosterlinck, Julie Lippens, Charles Van Praet, Federica Ferraguti, Charlotte Debbaut, Zhijin Li, Oliver Kutter, Alexandre Mottrie, and Karel Decaestecker. First-in-human real-time ai-assisted instrument deocclusion during augmented reality robotic surgery. *Healthcare Technology Letters*, 11:33–39, 2023.

[35] Amir H Sadeghi, Quinten Mank, Alper S Tuzcu, Jasper Hofman, Sabrina Siregar, Alexander Maat, Alexandre Mottrie, Jolanda Kluin, and Pieter De Backer. Artificial intelligence–assisted augmented reality robotic lung surgery: Navigating the future of thoracic surgery. *JTCVS Techniques*, 26:121–125, 2024.

[36] Soham Sinha, Shekhar Dwivedi, and Mahdi Azizian. Towards deterministic end-to-end latency for medical ai systems in nvidia holoscan. *arXiv preprint arXiv:2402.04466*, 2024.

[37] Nvidia multi-process service (mps). `https://docs.nvidia.com/deploy/mps/index.html`. Accessed: 2024-02-10.

[38] S.A. Alowais, S.S. Alghamdi, N. Alsuhebany, et al. Revolutionizing healthcare: the role of artificial intelligence in clinical practice. *BMC Medical Education*, 23:689, 2023.

[39] Medtronic gi genius. https://www.medtronic.com/covidien/en-us/products/gastrointestinal-artificial-intelligence/gi-genius-intelligent-endoscopy.html. Acessed: 2024-01-22.

[40] Fierce Biotech. Medtronic doubles down on endoscopy ai partnership, inking $200m deal with cosmo pharmaceuticals, 2023. Accessed: 2023-10-01.

[41] Medtronic. Medtronic unveils next-generation software for gi genius intelligent endoscopy system, 2023. Accessed: 2024-08-14.

[42] Odin Vision caddie. https://odin-vision.com/caddie-2/. Acessed: 2024-01-22.

[43] Moon Surgical. Moon surgical - maestro robotic system, 2023. Accessed: 2024-08-18.

[44] PR Newswire. Moon surgical receives second fda clearance covering its commercial maestro robotic surgery system, 2023. Accessed: 2024-08-14.

[45] Surgical Robotics Technology. Moon surgical announces over 200 patients treated with maestro system powered by nvidia holoscan, 2023. Accessed: 2024-08-14.

[46] Xuesong Han, Junxuan Yu, Xin Yang, Chaoyu Chen, Han Zhou, Chuangxin Qiu, Yan Cao, Tianjing Zhang, Meiran Peng, Guiyao Zhu, Dong Ni, Yuanji Zhang, and Nana Liu. Artificial intelligence assistance for fetal development: evaluation of an automated software for biometry measurements in the mid-trimester. *BMC Pregnancy and Childbirth*, 24:158, 2024.

[47] Trusted Reviews. What is an igpu?, 2024. Accessed: 2024-03-03.

[48] NVIDIA. Nvidia jetson agx orin technical brief, 2024. Accessed: 2024-03-03.

[49] NVIDIA jetson agx orin developer kit. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/. Acessed: 2024-01-14.

[50] NVIDIA clara para dispositivos médicos. https://www.nvidia.com/pt-br/clara/medical-devices/. Acessed: 2024-01-15.

[51] AJA Video Systems aja corvid 44 12g bnc. https://sdksupport.aja.com/. Acessed: 2024-01-20.

[52] Blackmagic Design micro converter bidirectional sdi/hdmi 12g. https://www.blackmagicdesign.com/pt/products/microconverters/techspecs/W-CONU-16. Acessed: 2024-01-17.

[53] Blackmagic Design micro converter bidirectional sdi/hdmi 12g. https://www.blackmagicdesign.com/support/. Acessed: 2024-01-17.

[54] ONNX. Onnx documentation, 2023. Accessed: 2024-08-14.

[55] NVIDIA. Nvidia tensorrt documentation, 2023. Accessed: 2024-08-14.

[56] PREMAI. State of open source ai: Model formats, 2023. Accessed: 2024-08-14.

[57] NVIDIA. Speeding up deep learning inference using tensorflow, onnx, and tensorrt, 2023. Accessed: 2024-08-14.

[58] NVIDIA Corporation. *NVIDIA Holoscan SDK User Guide*, 2023. Accessed: 2023-10-01.

[59] OpenCV team. Opencv: Open source computer vision library, 2024. Accessed: 2024-09-10.

[60] J.L. Gustafson. Amdahl's law. In D. Padua, editor, *Encyclopedia of Parallel Computing*. Springer, Boston, MA, 2011.

[61] Deep Learning Toolbox Converter for ONNX Model Format matlab. https://www.mathworks.com/matlabcentral/fileexchange/67296-deep-learning-toolbox-converter-for-onnx-model-format. Acessed: 2024-04-10.

[62] T. Chai and R. R. Draxler. Root mean square error (rmse) or mean absolute error (mae)? – arguments against avoiding rmse in the literature. *Geoscientific Model Development*, 7(3):1247–1250, 2014.

[63] Jeroen Bertels, Tom Eelbode, Maxim Berman, Dirk Vandermeulen, Frederik Maes, Raf Bisschops, and Matthew Blaschko. Optimizing the dice score and jaccard index for medical image segmentation: Theory & practice. *arXiv preprint arXiv:1911.01685*, 2019. MICCAI 2019.

[64] Statistically Relevant. Confusion matrix and roc curves, 2023. Accessed: 2024-09-01.

Appendix A

Scripts

This appendix provides detailed information on the scripts developed for the system, including the Python scripts for converting models to ONNX format, converting images to GXF format, and the MATLAB script for converting models to ONNX format.

A.1 convert_image_to_gxf.py (adapted from NVIDIA's SDK convert_video_to_gxf.py)

```
class ImageToGxfConverter inherits from UtilityScript

    Method iter_input_images(directory)
        - Fetches all TIFF image files from the specified directory
        - Reads each image file using OpenCV and returns the image frame

    Method main()
        - Parses command line arguments for input directory, output
          directory, basename, and framerate
        - Initializes an entity writer with the provided configurations
        - Iterates through the input images and writes them to GXF format
        - Saves a copy of each frame in PNG format

    Method start()
        - Initializes frame count and writes the images into GXF format
```

A.2 graph_surgeon.py

```
class GraphSurgeon inherits from ModelConversionUtility

    Method update_graph_input_output(graph)
        - Renames the input and output of the graph by appending "_old" to
            their names

    Method insert_transpose_node(graph)
        - Inserts a transpose node at the input layer to convert from NHWC
            to NCHW format
        - Rebinds the graph input to this new node

    Method convert_pytorch_to_onnx(graph, input_shape)
        - Updates the graph to adapt to the new input shape
        - Sorts the graph topologically and cleans up unused nodes
        - Saves the updated ONNX graph to the specified path
```

A.3 convert_unet_to_onnx.py

```
class UnetToOnnxConverter inherits from PyTorchModelExporter

    Method check_model_existence(model_path)
        - Verifies if the saved model exists in the specified directory
        - Raises a FileNotFoundError if the model is not found

    Method load_model(model_path)
        - Loads the UNet PyTorch model and processes the state dictionary
            to remove any "module." prefixes

    Method export_to_onnx(model, input_example)
        - Exports the loaded UNet model to ONNX format with input example
            of size (1, 3, 1024, 1024)
        - Performs constant folding optimization during export

    Method run_conversion()
        - Main function that checks model, loads it, defines example input
            , and exports to ONNX
```

A.4 Matlab model conversion script

```
class MatlabModelConverter inherits from ModelUtility

    Method load_model(model_path)
        - Loads a saved model from a .mat file and initializes the network

    Method process_images(image_folder, output_size)
        - Loads image files from the specified folder and resizes them to
          the given output size
        - Saves the processed images to the output folder

    Method convert_weights_to_float32(layers)
        - Converts the weights of fully connected and convolutional layers
            to FLOAT32 format

    Method export_to_onnx(network, output_filename)
        - Exports the neural network to an ONNX file with a specified
          OpsetVersion of 13
        - Displays a message indicating successful export

    Method load_onnx_model(model_name)
        - Imports the ONNX model for inference and makes predictions on
          the preprocessed images
```

Appendix B

Evaluation Results

This appendix presents the detailed evaluation results of the system, including the Dice score comparison between the original PyTorch model and the ONNX model for femur segmentation, and the comparison of accuracy, precision, and recall for the inside/outside knee model and lens mark visibility model across all datasets. The evaluation results demonstrate the performance and accuracy of the system in real-time surgical navigation applications.

B.1 Dice Score Comparison Between Original PyTorch Model and ONNX Model for Femur Segmentation

Dataset	1	2	3	4	5	6	7
Dice Score	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Table B.1: Dice Score Comparison Between Original PyTorch Model and ONNX Model for Femur Segmentation. A Dice score of 1 across all datasets indicates that the segmentation outputs from the ONNX model are identical to those from the original PyTorch model, validating the conversion process.

B.2 Comparison of Accuracy, Precision, and Recall for Inside/Outside across all Datasets

	Dataset						
	1	2	3	4	5	6	7
Accuracy	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Precision	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Recall	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Table B.2: Comparison of Accuracy, Precision, and Recall for Inside/Outside Model across all Datasets

B.3 Comparison of Accuracy, Precision, and Recall for Holoscan Inside/Outside Model across all Datasets

	Dataset						
	1	2	3	4	5	6	7
Accuracy	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Precision	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Recall	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Table B.3: Comparison of Accuracy, Precision, and Recall for Holoscan Inside/Outside Model across all Datasets

B.4 Comparison of Accuracy, Precision, and Recall for Lens Mark Model across all Datasets

	Dataset						
	1	2	3	4	5	6	7
Accuracy	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Precision	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Recall	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Table B.4: Comparison of Accuracy, Precision, and Recall for Lens Mark Model across all Datasets

Appendix C

System Specifications

This appendix provides detailed specifications of the hardware components used in the development and deployment of the system. The specifications include the NVIDIA Jetson AGX Orin Developer Kit, AJA Corvid 44 12G BNC, and SDI/HDMI 12G Bidirectional Micro Converter. These specifications are essential for understanding the capabilities and performance of the system components.

C.1 Jetson AGX Orin Specifications

Specification	Details
GPU	2048-core NVIDIA Ampere architecture GPU with 64 Tensor Cores
GPU Max Frequency	1.3 GHz
CPU	12-core Arm® Cortex®-A78AE v8.2 64-bit CPU 3MB L2 + 6MB L3
CPU Max Frequency	1.3 GHz
Memory	64GB 256-bit LPDDR5 204.8GB/s
Storage	64GB eMMC 5.1
Video Encode	2x 4K60 (H.265), 4x 4K30 (H.265), 8x 1080p60 (H.265), 16x 1080p30 (H.265)
Video Decode	1x 8K30 (H.265), 3x 4K60 (H.265), 7x 4K30 (H.265), 11x 1080p60 (H.265), 22x 1080p30 (H.265)
CSI Camera	16-lane MIPI CSI-2 connector
USB	USB Type-C connector: 2x USB 3.2 Gen2, USB Type-A connector: 2x USB 3.2 Gen2, 2x USB 3.2 Gen1, USB Micro-B connector: USB 2.0
Display	1x DisplayPort 1.4a (+MST) connector
Power	15W - 60W
Mechanical	110mm x 110mm x 71.65mm (Height includes feet, carrier board, module, and thermal solution)

Table C.1: NVIDIA Jetson AGX Orin Developer Kit System Specifications

C.2 AJA Corvid 44 12G BNC Specifications

Specification	Details
PCI Interface	Eight-lane gen 3 PCI Express, 2 DMA engines
Inputs and Outputs	4 × 12G-SDI bi-directional (under software control), one × reference/LTC input, 1 × secondary LTC-only input (2-pin Molex connector onboard header), 1 × RS-422 serial port (full-height board version has DB-9 connector)
Frame Buffer	4GB SDRAM, 512 x 8MB frames, 256 × 16MB frames for 2K (1080×2048), 1920×1080 VANC, and 16-bit RGB
Video Formats	Supports all SD, HD, 2K, 4K, and UHD2 video formats (8K coming soon), Multiple, independent video formats supported (playout requires same clock family), SMPTE 425 2si mux supported

Table C.2: General specifications of the AJA Corvid 44 12G BNC

C.3 SDI/HDMI 12G Bidirectional Micro Converter Specifications

Connections:	• SDI Video Inputs: 1 • SDI Video Outputs: Automatically match the HDMI input. • SDI Rates: 270Mb, 1.5G, 3G, 6G, 12G. • Multi-rate Support: Automatic detection of SD, HD, 2K, Ultra HD, and 4K. • HDMI 2.0 Video Inputs: 1 x type A. • HDMI 2.0 Video Outputs: 1 x type A.

Table C.3: General specifications of the SDI/HDMI 12G Bidirectional Micro Converter

Appendix D

Operators

This chapter provides detailed information on the configuration parameters for the Holoscan's built-in operators and custom operators developed for the application. At last is described the full workflow of the application.

D.1 Operator Parameter Configuration

D.1.1 `VideoStreamReplayerOp` Configuration Parameters

The `VideoStreamReplayerOp` class includes the following configurable parameters:

Parameter	**Description**	**Default Value**
`directory`	Specifies the directory path for reading the GXF files.	N/A
`basename`	The user-specified file name without an extension.	N/A
`batch_size`	The number of entities (frames) to read and publish in one tick.	1
`ignore_corrupted_entities`	Controls the handling of corrupted entities. If an entity could not be deserialized, it is ignored.	`True`
`frame_rate`	Specifies the frame rate for replaying the video. A value of 0.0 allows replay based on timestamps.	0.0
`realtime`	When set to `True`, the video will play back in real-time, based on either the frame rate or timestamps.	`True`

Table continued from previous page

Parameter	Description	Default Value
`repeat`	Controls whether the video stream will loop indefinitely.	`False`
`count`	Limits the number of frames to playback. If set to 0, all frames will be played.	0

D.1.2 `AJASourceOp` Configuration Parameters

The `AJASourceOp` operator provides various configuration options to fine-tune the video capture process. Below are the key parameters:

Parameter	Description	Default Value
`device`	Specifies which AJA device to target.	`"0"`
`channel`	The NTV2Channel to be used for output (e.g., `NTV2_CHANNEL1`).	`NTV2_CHANNEL1`
`width`	Width of the video stream.	1920
`height`	Height of the video stream.	1080
`framerate`	Frame rate of the video stream.	60
`rdma`	Boolean indicating whether RDMA is enabled.	`False`
`enable_overlay`	Boolean indicating whether a separate overlay channel is enabled.	`False`
`overlay_channel`	The NTV2Channel to use for overlay output.	`NTV2_CHANNEL2`
`overlay_rdma`	Boolean indicating whether RDMA is enabled for the overlay.	`False`

The `AJASourceOp` operator has two key output ports:

- **video_buffer_output**: This port emits the video frame captured by the AJA card. If RDMA is enabled, the video buffer resides on the device; otherwise, it is stored in pinned host memory.

- **overlay_buffer_output** (optional): This port emits a video buffer when overlay functionality is enabled. Like the main video buffer, it can reside on the device or in pinned host memory, depending on RDMA configuration.

The operator can also accept an optional input, `overlay_buffer_input`, to mix the incoming overlay with the captured video when overlay functionality is activated.

D.1.3 `FormatConverterOp` Configuration Parameters

The `FormatConverterOp` class includes the following configurable parameters:

Parameter	**Description**	**Default Value**
`fragment`	The fragment that the operator belongs to.	N/A
`pool`	Memory pool allocator used by the operator.	N/A
`out_dtype`	Destination data type. Available options are: `"rgb888"`, `"uint8"`, `"float32"`, `"rgba8888"`, `"yuv420"`, `"nv12"`.	N/A
`in_dtype`	Source data type. Available options are: `"rgb888"`, `"uint8"`, `"float32"`, `"rgba8888"`, `"yuv420"`, `"nv12"`.	N/A
`in_tensor_name`	The name of the input tensor.	`""`
`out_tensor_name`	The name of the output tensor.	`""`
`scale_min`	Minimum value for output clipping.	0.0
`scale_max`	Maximum value for output clipping.	1.0
`alpha_value`	Unsigned integer [0, 255] indicating the alpha channel value for RGB to RGBA conversion.	255
`resize_height`	Desired height for resizing the output. Unchanged if set to 0.	0
`resize_width`	Desired width for resizing the output. Unchanged if set to 0.	0
`resize_mode`	Resize mode enum value corresponding to NPP's `NppiInterpolationMode`.	0

Table continued from previous page

Parameter	Description	Default Value
`channel_order`	Sequence of integers describing the channel value permutation.	`[0, 1, 2]` for 3-channel, `[0, 1, 2, 3]` for 4-channel
`cuda_stream_pool`	`holoscan.resources.CudaStreamPool` instance for allocating CUDA streams.	`None`

Named Inputs:

- `source_video`: The input video frame to process, which can be either `nvidia::gxf::Tensor` or `nvidia::gxf::VideoBuffer`. The input must be in a supported format such as `GXF_VIDEO_FORMAT_RGBA`, `GXF_VIDEO_FORMAT_RGB`, or `GXF_VIDEO_FORMAT_NV12`.

Named Outputs:

- `tensor`: The output video frame after processing, transmitted as a `nvidia::gxf::Tensor`. The shape, data type, and number of channels will depend on the parameters set for this operator.

D.1.4 `InferenceOp` Configuration Parameters

The `InferenceOp` class includes the following configurable parameters:

Parameter	Description	Default Value
`fragment`	The fragment that the operator belongs to.	N/A
`backend`	Backend to use for inference. Options: `"trt"` for TensorRT, `"torch"` for LibTorch, and `"onnxrt"` for ONNX runtime.	N/A
`allocator`	Memory allocator used for the output.	N/A
`inference_map`	Dictionary mapping tensor names to models for inference.	N/A

Table continued from previous page

Parameter	Description	Default Value
`model_path_map`	Dictionary mapping model names to their respective file paths.	N/A
`pre_processor_map`	Dictionary mapping pre-processed data to models.	N/A
`device_map`	Dictionary mapping models to GPU IDs for inference.	N/A
`temporal_map`	Dictionary mapping models to frame delays for inference.	N/A
`activation_map`	Dictionary mapping models to their activation states for inference.	N/A
`backend_map`	Dictionary mapping models to their backend types (e.g., `"trt"`, `"torch"`).	N/A
`in_tensor_names`	Sequence of input tensor names.	N/A
`out_tensor_names`	Sequence of output tensor names.	N/A
`infer_on_cpu`	Boolean indicating whether to run inference on the CPU.	`False`
`parallel_inference`	Boolean indicating whether to enable parallel execution.	`True`
`input_on_cuda`	Boolean indicating whether the input buffer is on the GPU.	`True`
`output_on_cuda`	Boolean indicating whether the output buffer is on the GPU.	`True`
`transmit_on_cuda`	Boolean indicating whether to transmit the message on the GPU.	`True`
`enable_fp16`	Boolean indicating whether to use 16-bit floating point computations.	`False`
`is_engine_path`	Boolean indicating whether the input model path mapping is for TensorRT engine files.	`False`
`cuda_stream_pool`	`holoscan.resources.CudaStreamPool` instance to allocate CUDA streams.	`None`

Named Inputs:

- `receivers`: Multi-receiver accepting `nvidia::gxf::Tensor(s)`. This can connect to any number of upstream ports. The operator will search for tensors matching the specified `in_tensor_names` in all incoming messages. These tensors are used by the models in `inference_map`.

Named Outputs:

- `transmitter`: Emits a message containing tensors corresponding to the inference results from all models. The names of the tensors correspond to those in `out_tensor_names`.

D.1.5 `SegmentationPostprocessorOp` Configuration Parameters

The `SegmentationPostprocessorOp` class includes the following configurable parameters:

Parameter	Description	Default Value
`fragment`	The fragment that the operator belongs to.	N/A
`allocator`	Memory allocator used for the output.	N/A
`in_tensor_name`	Name of the input tensor.	`""`
`network_output_type`	Network output type (e.g., `"softmax"`).	`"softmax"`
`data_format`	Data format of network output (e.g., `"hwc"`).	`"hwc"`
`cuda_stream_pool`	`holoscan.resources.CudaStreamPool` instance to allocate CUDA streams.	`None`

Named Inputs:

- `in_tensor`: Expects a message containing a 32-bit floating point device tensor named as specified in `in_tensor_name`. The tensor's data layout can be `HWC`, `NCHW`, or `NHWC` format, as specified by `data_format`.

Named Outputs:

- `out_tensor`: Emits a message containing a device tensor named `"out_tensor"` that holds the segmentation labels. This tensor has an unsigned 8-bit integer data type and a shape of `(H, W, 1)`.

D.1.6 `HolovizOp` Configuration Parameters

The `HolovizOp` class provides visualization capabilities using the Holoviz module, a Vulkan-based visualizer. Below is a detailed description of the parameters and methods for configuring this operator:

Parameter	Description
`fragment`	The fragment that the operator belongs to.
`allocator`	Allocator used to allocate render buffer output. If `None`, defaults to `holoscan.core.UnboundedAllocator`.
`receivers`	Sequence of `holoscan.core.IOSpec`. List of input receivers.
`tensors`	Sequence of dictionaries defining the input tensors. Each tensor is defined by a dictionary with keys such as `"name"` and `"type"`.
`color_lut`	Color lookup table for tensors of type `color_lut`. Should be a list of floats with shape `(n_colors, 4)`.
`window_title`	Title of the window canvas. Default is `"Holoviz"`.
`display_name`	Name of the display in exclusive display or fullscreen mode. Default is `""`.
`width`	Window width or display resolution width in exclusive display or fullscreen mode. Default is `1920`.
`height`	Window height or display resolution height in exclusive display or fullscreen mode. Default is `1080`.
`framerate`	Display framerate in Hz for exclusive display mode. Default is `60.0`.
`use_exclusive_display`	Enable exclusive display mode. Default is `False`.
`fullscreen`	Enable fullscreen window. Default is `False`.
`headless`	Enable headless mode (no window opened). Default is `False`.
`framebuffer_srgb`	Enable sRGB framebuffer for rendering. Default is `False`.
`vsync`	Enable vertical sync. Default is `False`.

Table continued from previous page

Parameter	Description
enable_render _buffer_input	If True, adds an additional input port named "render_buffer_input". Default is False.
enable_render _buffer_output	If True, adds an additional output port named "render_buffer_output". Default is False.
enable_camera _pose_output	If True, adds an additional output port named "camera_pose_output". Default is False.
camera_pose _output_type	Specifies the type of data output for the "camera_pose_output" port. Supported values are "projection_matrix" and "extrinsics_model". Default is "projection_matrix".
camera_eye	Initial camera eye position. Default is (0.0, 0.0, 1.0).
camera_look_at	Initial camera look-at position. Default is (0.0, 0.0, 0.0).
camera_up	Initial camera up vector. Default is (0.0, 1.0, 0.0).
font_path	File path for the font used for rendering text. Default is "".
cuda_stream_pool	holoscan.resources.CudaStreamPool instance for allocating CUDA streams. Default is None.
name	Name of the operator. Default is "holoviz_op".

Named Inputs:

- receivers: Multi-receiver accepting nvidia::gxf::Tensor and/or nvidia::gxf::VideoBuffer. Each input results in a layer, with the operator autodetecting the layer type for certain input types.
- input_specs (optional): A list of InputSpec objects to dynamically update the overlay specification at runtime.
- render_buffer_input (optional): An empty render buffer provided in nvidia::gxf::VideoBuffe format. This input port is available if enable_render_buffer_input is set to True.

Named Outputs:

- render_buffer_output (optional): Outputs a filled render buffer in nvidia::gxf::VideoBuffer format. Available if enable_render_buffer_output is set to True.

- `camera_pose_output` (optional): Outputs the camera pose as either a 4x4 row-major projection matrix or camera extrinsics model, depending on the `camera_pose_output_type` setting.

Notes:

The `tensors` parameter specifies the tensors to display. Each tensor is defined using a dictionary that must include a `"name"` key and, optionally, a `"type"` key. Examples of supported types include `"color"`, `"rectangles"`, `"points"`, and more. Additional parameters, such as `opacity`, `priority`, and `color`, can also be specified to control the visualization.

D.2 ImageProcessingOp Implementation

```
class ImageProcessingOp inherits from Operator

    Method save_image(image, save_filepath)
        - Saves the image in uint8 format at the specified filepath

    Method resize(image, size, is_label)
        - Resizes the image to the specified size
        - If is_label is True, use nearest-neighbor interpolation;
          otherwise, use cubic interpolation

    Method save_mask(mask_array, original_size)
        - Resizes and crops the mask_array to match the original_size
        - Returns the resized and cropped mask

    Method crop_outside_circle(img)
        - Converts the image to grayscale
        - Labels connected regions and sorts by size
        - Crops the image outside the largest labeled region
        - Returns the cropped image and mask

    Method setup(spec)
        - Sets up input and output specifications for the operator

    Method start()
        - Initializes frame count

    Method compute(op_input, op_output, context)
```

```
        - Receives input tensor
        - Preprocesses the image (resize, crop, apply CLAHE, resize to
          final size)
        - Sends the processed image to the output tensor

    Method stop()
        - Placeholder for stopping the operator
```

D.3 PostImageProcessingOp Implementation

```
class PostImageProcessingOp inherits from Operator

    Method resize(image, size, is_label)
        - Resizes the image to the specified size
        - If is_label is True, use nearest-neighbor interpolation;
          otherwise, use cubic interpolation

    Method save_mask(mask_array, original_size)
        - Resizes and crops the mask_array to match the original_size
        - Returns the resized and cropped mask

    Method setup(spec)
        - Sets up input and output specifications for the operator

    Method start()
        - Initializes frame count

    Method compute(op_input, op_output, context)
        - Receives input tensor
        - Converts image data to uint8, processes the mask
        - Saves the processed mask to a file
        - Sends the processed mask to the output tensor

    Method stop()
        - Placeholder for stopping the operator
```

D.4 ImageProcessing_in_out_Op Operator

```
class ImageProcessing_in_out_Op inherits from Operator

    Method setup(spec)
        - Sets up input and output specifications for the operator

    Method start()
        - Initializes frame count

    Method compute(op_input, op_output, context)
        - Receives input tensor
        - Resizes and converts the image
        - Sends the processed image to the output tensor

    Method stop()
        - Placeholder for stopping the operator
```

D.5 ImageProcessing_lens_mark_Op Operator

```
    class ImageProcessing_lens_mark_Op inherits from Operator

        Method setup(spec)
            - Sets up input and output specifications for the operator

        Method start()
            - Initializes frame count

        Method compute(op_input, op_output, context)
            - Annotates the operation with NVTX
            - Records the start time
            - Receives input tensors
            - Converts input tensor to numpy array
            - Prints the shape of the frame
            - Checks the range of the frame values
            - Converts the frame back to a tensor
            - Sends the processed frame to the output tensor

        Method stop()
            - Placeholder for stopping the operator
```

D.6 PostInferenceOp Implementation

```
class PostInferenceOp inherits from Operator

    Method setup(spec)
        - Sets up input and output specifications for the operator

    Method start()
        - Initializes frame count

    Method compute(op_input, op_output, context)
        - Receives input tensor
        - Compares dimensions and determines state (Inside/Outside)
        - Creates and sends output tensors based on state

    Method stop()
        - Placeholder for stopping the operator
```

D.7 visualizerOp Implementation

```
class visualizerOp inherits from Operator

    Method setup(spec)
        - Sets up input and output specifications for the operator

    Method start()
        - Initializes frame count

    Method compute(op_input, op_output, context)
        - Receives input tensors
        - Combines specifications from inputs
        - Sends combined specifications to the output tensor

    Method stop()
        - Placeholder for stopping the operator
```

D.8 PostInference_lens_mark_classification_Op Operator

```
class PostInference_lens_mark_classification_Op inherits from Operator
    Method setup(spec)
        - Sets up input and output specifications for the operator

    Method start()
        - Initializes frame count

    Method compute(op_input, op_output, context)
        - Receives input tensors
        - Combines specifications from inputs
        - Sends combined specifications to the output tensor

    Method stop()
        - Placeholder for stopping the operator
```

D.9 Workflow Implementation

```
# Part 1: Arthrosegmentation Model
if arthrosegmentation_model:
    if is_aja:
        - Add flow from source to viz and drop alpha channel
        if do_preprocessing:
            - Add flow from drop_alpha_channel to ImageProcessing
            - Add flow from ImageProcessing to preprocessor
        else:
            - Add flow from drop_alpha_channel to preprocessor
    else:
        if do_preprocessing:
            - Add flow from source to ImageProcessing
            - Add flow from source to preprocessor
        else:
            - Add flow from source to viz
            - Add flow from source to preprocessor
    - Add flow from preprocessor to inference
    - Add flow from inference to postprocessor
```

```
    - Add flow from postprocessor to PostImageProcessing
    - Add flow from PostImageProcessing to viz
    if do_record_output:
        - Add flow from viz to recorder

# Part 2: Inside/Outside Model
if in_out_model:
    if is_aja:
        - Add flow from source to viz and drop alpha channel
        if do_preprocessing:
            - Add flow from drop_alpha_channel to ImageProcessing_in_out
            - Add flow from ImageProcessing_in_out to preprocessor_in_out
        else:
            - Add flow from drop_alpha_channel to preprocessor_in_out
    else:
        if do_preprocessing:
            - Add flow from source to ImageProcessing_in_out
            - Add flow from source to preprocessor_in_out
        else:
            - Add flow from source to viz
    - Add flow from preprocessor_in_out to inference
    - Add flow from inference to PostInference and viz
    if do_record_output:
        - Add flow from viz to recorder

# Part 3: Lens Mark Classification Model
if lens_mark_classification_model:
    if is_aja:
        - Add flow from source to viz and drop alpha channel
        if do_preprocessing:
            - Add flow from drop_alpha_channel to
               ImageProcessing_lens_mark
            - Add flow from ImageProcessing_lens_mark to
               preprocessor_lens_mark
        else:
            - Add flow from drop_alpha_channel to preprocessor_lens_mark
    else:
        if do_preprocessing:
            - Add flow from source to ImageProcessing_lens_mark
            - Add flow from source to preprocessor_lens_mark
```

```
        else:
            - Add flow from source to viz
    - Add flow from preprocessor_lens_mark to inference
    - Add flow from inference to PostInference_lens_mark and viz
    if do_record_output:
        - Add flow from viz to recorder

# Part 4: Both Models
if both_models:
    if is_aja:
        - Add flow from source to viz and drop alpha channel
        if do_preprocessing:
            - Add multiple flows to ImageProcessing,
            - ImageProcessing_lens_mark, ImageProcessing_in_out
        else:
            - Add flow from drop_alpha_channel to preprocessor
    else:
        if do_preprocessing:
            - Add multiple flows to ImageProcessing,
            - ImageProcessing_lens_mark, ImageProcessing_in_out
        else:
            - Add flow from source to viz
    - Add flows from preprocessors to inference and PostInference
       processes
    if do_record_output:
        - Add flow from viz to recorder
```

Appendix E

Preprocessing Pseudocode

This appendix provides pseudocode for the preprocessing functions used in the image processing pipeline. The pseudocode outlines the steps for image preprocessing, including contrast enhancement, resizing, cropping, and mask generation. The functions are implemented in C++ and CUDA for efficient image processing on the GPU.

E.1 C++ Preprocessing Pseudocode

```
# Preprocessing Functions for Image Processing

Function CLAHEWithMask(cv::Mat& rgb_image, const cv::Mat& mask, const int mask_area)
    - Creates a CLAHE object for Contrast Limited Adaptive Histogram Equalization.
    - Applies CLAHE only to the pixels within the largest connected island in the mask.
    - Extracts RGB channels within the mask area.
    - Applies CLAHE to each channel separately.
    - Updates the RGB image with enhanced pixel values inside the mask.
    - Sets pixels outside the mask to zero.

Function largestIslandMask(const cv::Mat& rgb_image, int& largest_area)
    - Converts the input image to grayscale.
    - Thresholds the grayscale image to create a binary mask.
    - Uses connected components to identify distinct regions in the mask.
```

```
15          - Finds the largest connected component (excluding the background)
                .
16          - Returns a binary mask of the largest island.
17
18      Function crop(const cv::Mat& rgb_image)
19          - Calculates the middle size for cropping based on image
                dimensions.
20          - Crops the image to a square region by removing excess pixels on
                the wider side.
21          - Returns the cropped image.
22
23      Function resize(const cv::Mat& rgb_image, const cv::Size& size)
24          - Checks if the image size matches the desired size.
25          - If not, resizes the image using cubic interpolation.
26          - Returns the resized image.
27
28      Function preprocessing(cv::Mat& rgb_image)
29          - Resizes the input image to 1080x1920 resolution.
30          - Crops the image to a square aspect ratio.
31          - Identifies the largest island (connected component) in the image
                .
32          - Applies CLAHE with the mask of the largest island to enhance
                contrast.
33          - Resizes the final image to 1024x1024 resolution.
34          - Returns the processed image.
```

E.2 CUDA Preprocessing Pseudocode

```
1       # Preprocessing Functions for Image Processing with CUDA
2
3       Function CLAHEWithMask(cv::Mat& rgb_image, const cv::Mat& mask, const
            int mask_area)
4           - Starts a timer to measure execution time.
5           - Splits the input RGB image into three separate channels (R, G, B
                ).
6           - Creates a CLAHE (Contrast Limited Adaptive Histogram
                Equalization) object using CUDA.
7           - Sets the clip limit for contrast enhancement (default 40.0 for
                our case).
```

```
        - Uploads each channel to the GPU for CLAHE application.
        - Downloads the processed image back to the CPU.
        - Sets pixels outside the mask to zero for each channel.
        - Merges the processed channels back into a single RGB image.
        - Measures the CLAHE processing time and prints the FPS.

    Function largestIslandMask(const cv::Mat& rgb_image, int& largest_area
        )
        - Converts the input image from RGB to grayscale.
        - Applies a binary threshold to create a mask.
        - Uses connected components to identify distinct regions in the
            image.
        - Finds the largest connected component (excluding the background)
            .
        - Returns a binary mask of the largest region (island).

    Function crop(const cv::Mat& rgb_image)
        - Calculates the middle region based on the difference between the
            image width and height.
        - Crops the image to a square region, removing excess width.
        - Returns the cropped image.

    Function resize(const cv::Mat& rgb_image, const cv::Size& size)
        - Checks if the image is already at the target size.
        - If not, resizes the image using cubic interpolation.
        - Returns the resized image.

    Function preprocessing(cv::Mat& rgb_image)
        - Starts a timer to measure the entire preprocessing time.
        - Resizes the input image to HD resolution (1080x1920).
        - Crops the resized image to a square aspect ratio.
        - Applies CLAHE using the mask of the largest connected island in
            the image.
        - Resizes the final processed image to 1024x1024 resolution.
        - Returns the processed image and prints the total execution time
            and FPS.
```