CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Edge vs Cloud Models in App-Based AI for Orthodontic Assessment

An Analysis of Architecture, Performance Metrics, and Clinical Effectiveness in AI-driven Malocclusion Detection on Edge and Cloud Platforms

Bachelor's thesis in Computer Science and Engineering

CARL ANDRÉN
ALI AWADA
ISAK GENBERG
OSKAR MEYER
KEVIN NORLIN
BERTIL VARENHORST

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Bachelor's thesis 2025

Edge vs Cloud Models in App-Based AI for Orthodontic Assessment

An Analysis of Architecture, Performance Metrics, and Clinical Effectiveness in AI-driven Malocclusion Detection on Edge and Cloud Platforms

CARL ANDRÉN
ALI AWADA
ISAK GENBERG
OSKAR MEYER
KEVIN NORLIN
BERTIL VARENHORST

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Edge vs Cloud Models in App-Based AI for Orthodontic Assessment
An Analysis of Architecture, Performance Metrics and Clinical Effectiveness in AI-driven Malocclusion Detection on Edge and Cloud Platforms
CARL ANDRÉN
ALI AWADA
ISAK GENBERG
OSKAR MEYER
KEVIN NORLIN
BERTIL VARENHORST

Supervisor: Miroslaw Staron, Interaction Design and Software Engineering, Computer Science and Engineering
Co-supervisor: Emina Čirgić
Examiner: Arne Linde, Department of Computer Science and Engineering
Graded by teacher: Irum Inayat, Interaction Design and Software Engineering, Computer Science and Engineering

Bachelor's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Edge vs Cloud Models in App-Based AI for Orthodontic Assessment
An Analysis of Architecture, Performance Metrics, and Clinical Effectiveness in AI-driven Malocclusion Detection on Edge and Cloud Platforms

CARL ANDRÉN
ALI AWADA
ISAK GENBERG
OSKAR MEYER
KEVIN NORLIN
BERTIL VARENHORST
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

This thesis investigates the trade-offs between edge and cloud deployment of CNN models for AI-driven malocclusion detection in a mobile application. The study explores the CNNs VGG-19, YOLOv8 and ResNet-50 and their latency, accuracy and efficiency in Cloud and Edge environments. A mobile prototype was developed to capture dental images, preprocess them using ARKit-based alignment and validation mechanisms, and classify the severity of malocclusion according to the Skåneindex scale. Each model was evaluated using performance metrics such as F1-score, specificity, and inference latency, while also considering computational resource usage. The results indicate that edge deployment reduces latency, improving user responsiveness, while cloud deployment offers marginally higher classification accuracy, particularly with VGG-19. YOLOv8 demonstrated strong overall performance and robustness across environments. Additionally, expert stakeholder validation confirmed the clinical potential of the application in streamlining orthodontic screening and referral processes. These findings highlight key considerations in selecting an appropriate deployment strategy for mobile AI applications in healthcare.

Sammandrag

Detta examensarbete undersöker avvägningarna mellan edge- och cloud-distribution av konvolutionella neurala nätverk (CNN) för AI-baserad diagnostik av malocklusion i en mobilapplikation. Studien fokuserar på modellerna VGG-19, YOLOv8 och ResNet-50, och utvärderar deras latens, noggrannhet och effektivitet i både molnbaserade och edge-miljöer. En mobil prototyp utvecklades för att ta tandbilder, förbehandla dem med hjälp av ARKit-baserad bildjustering och validering, samt klassificera malocklusionens grad enligt Skåneindex-skalan. Varje modell analyserades med prestandamått såsom F1-score, specificitet och inferenslatens, samt resursförbrukning. Resultaten visar att edge-distribution minskar latens och förbättrar användarrespons, medan molndistribution ger något högre klassificeringsnoggrannhet, särskilt med VGG-19. YOLOv8 uppvisade stark och konsekvent prestanda i båda miljöerna. Dessutom bekräftade expertgranskning den kliniska potentialen för applikationen att effektivisera ortodontisk screening och remisshantering. Studien belyser viktiga aspekter vid val av distributionsstrategi för mobila AI-applikationer inom hälso- och sjukvård.

Keywords: Artificial Intelligence, Machine learning, Edge, Cloud, Orthodontics, Malocclusion, Clinical

Acknowledgements

We owe a heartfelt thanks to our supervisor, Prof. Miroslaw Staron, whose guidance and support carried us through every stage of this project. His willingness to answer our questions significantly contributed to the completion of this bachelors thesis. The authors would also like to extend their gratitude to Dr. Emina Čirgić for her medical expertise in orthodontics, particularly in demonstrating the diagnostic process of crooked teeth and clarifying the criteria used in her assessments. Furthermore, her insights into the evaluation of dental images helped with understanding the difficulties in creating usable images for orthodontic review.

Carl Andrén, Gothenburg, June 2025
Ali Awada, Gothenburg, June 2025
Isak Genberg, Gothenburg, June 2025
Oskar Meyer, Gothenburg, June 2025
Kevin Norlin, Gothenburg, June 2025
Bertil Varenhorst, Gothenburg, June 2025

List of Acronyms

AI	Artificial Intelligence. vi, xi, xiv, xviii, 1–6, 13–18, 27, 28, 35, 49
API	Application Programming Interface. xi, 13, 28, 34, 35
ARKit	Augmented Reality Kit. xi, 27, 30
AWS	Amazon Web Services. xi, xix, 14, 34, 35
Certbot	Let's Encrypt certificate automation tool. xi, 34
CNN	Convolutional Neural Network. vi, xi, 1, 4, 6, 9, 10, 13, 15, 27–29
CPU	Central processing unit. xi, 2
DSLR	Digital Single-Lens Reflex. xi, 5
EC2	A virtual server in Amazon's Elastic Compute Cloud. xi, xix, 14, 35
FC	Fully Connected Layer. xi, 9, 28
HTTPS	Hypertext Transfer Protocol Secure. xi, 19, 34
IoMT	Internet of Medical Things. xi, 1
IoU	Intersection over Union. xi, xix, 31, 39, 40
JWT	JSON web token. xi, 35
mAP	Mean Average Precision. xi, xix, 5, 39, 40
ML	Machine Learning. xi, 4, 15
nginx	Reverse proxy server. xi, 34
NMS	Non-Max Suppression. xi, 31
OHRQoL	Oral Health-Related Quality of Life. xi
ONNX	Open Neural Network Exchange. xi, 15, 47
ORT	ONNX Runtime. xi, xviii, 15, 19, 28, 29
OS	Operating System. xi
postgreSQL	an object-relational database. xi, 34–36
ReLU	Rectified Linear Unit. xi, 9
ResNet-50	Residual neural network 50. xi, xvii, xxi, 10–12, 23, 24, 37–39, 47, 48
ROI	Region of interest. xi, 33
S3	Amazon Simple Storage Service. xi, xix, 14, 34–36, 48
SSL	Socket Security Layer. xi
UI	User Interface. xi, 30
URL	Uniform Resource Locator. xi
VGG-19	Visual Geometry Group. xi, xxi, 10, 12, 23, 24, 37–39, 47, 48
VM	Virtual Machine. xi
YOLO	You Only Look Once. xi, xxi, 5, 6, 10, 12, 31, 37, 39, 46–50

Contents

List of Figures

List of Tables

1
Introduction

In an effort to improve the healthcare systems, countries seek to optimize their medical service [1]. One key approach to achieving this is the introduction of remote medical services, which is dependent on utilization of the Internet of Medical Things (IoMT) - referring to a network of connected healthcare technologies that explains the collection, processing and transmission of health information [1], [2]. IoMT enables more accessible treatment methods and allows for remote execution of various medical tasks, including disease monitoring and diagnosis [1]. This does not only result in reduced medical costs, but also enhances the quality of medical services.

Among the various medical conditions that can be targeted through remote services, dental conditions are one of the main areas. Among these, a common cause for patients to seek orthodontic care is malocclusion, which is a highly prevalent orofacial condition, characterized by misaligned teeth or an incorrect relation between the upper and lower dental arch, or both [3], [4]. Current clinical practice relies on thorough clinical evaluation for diagnosis and prediction of optimal treatment, with complex cases requiring treatment from dental specialists, which can be costly and time consuming [5].

In order to ensure that this thesis has clinical relevance and provides value for patients and medical professionals, development was conducted in collaboration with an experienced odontologist. Furthermore, a previous thesis working with the same odontologist successfully demonstrated that a Convolutional Neural Network (CNN) can achieve satisfactory performance when trained on dental images for malocclusion detection [6]. This finding suggests that AI-based diagnostics in orthodontics are not only theoretically viable but also practically implementable.

Building on these insights, this thesis explored the development of a mobile application capable of analyzing dental images to detect crooked teeth. The app provides users with a preliminary classification and, when necessary, informs them that professional evaluation may be required. The goal was to provide an accessible, efficient, and clinically relevant tool for preliminary screening, helping patients determine whether further orthodontic assessment is needed.

However, AI deployment strategies pose a key challenge. Should inference be performed on the edge or in the cloud? Both approaches present trade-offs in accuracy, latency, resource efficiency, and privacy, all of which could significantly impact real-world usability in a clinical setting.

In this thesis, we systematically examined the trade-offs between edge-based and cloud-based machine learning models for malocclusion diagnosis. By comparing key performance metrics, we determined which deployment strategy is the most suitable for clinical applications. This investigation is crucial to understanding how AI-driven orthodontic screening can be implemented effectively while maintaining accessibility, efficiency, and compliance with healthcare standards.

1.1 Problem

Our central problem statement for investigation is: "What trade-offs exist between cloud-based and on-device ML models in malocclusion diagnosis?". In order to better understand and quantify the main problem, it was divided into several subproblems across different domains. The product iterations were then compared with each other according to these criteria.

Model Performance:
Metrics such as the F1-score, accuracy, sensitivity, specificity, and precision are crucial for evaluating the overall performance of the model. Therefore, the two deployment strategies were analyzed and compared in this regard.

Latency:
The processing time (latency) of the model for generating a diagnostic result from an input image between on-device and cloud-based inference was measured. The end-to-end latency was measured by recording the time from when the user submits an image to when they receive the evaluation response. This was done using high-resolution timestamps within the app, with multiple trials conducted to calculate the average processing time in milliseconds.

Resource Efficiency:
The resource efficiency of the application was measured in terms of Central processing unit (CPU) usage and memory allocation to determine which architecture is more efficient.

1.2 Purpose

The purpose of this project was to develop a diagnostic system powered by AI, using convolutional neural networks, to provide an initial assessment of the patients teeth and explore trade-offs between cloud-based and on-device machine learning models in malocclusion diagnoses. A key aspect was the development of a test framework to systematically evaluate and compare the deployment of cloud-based and on-device AI models. This enabled a structured assessment of model metrics, latency, and resource efficiency, providing valuable information on the trade-offs between the

two deployment strategies. The project followed Roel J. Wieringa's Design Science Methodology [7] to develop the diagnostic system.

The system consists of a mobile application for end users and a cloud infrastructure for data storage and further processing. The mobile app allows patients to photograph their teeth and receive an instant initial suggestion on whether further treatment or a specialist consultation might be necessary. The cloud infrastructure facilitates storage of diagnostic results for users who choose to seek further treatment. This allows medical professionals to access relevant information when evaluating a referred patient. By allowing potential patients to perform initial self-assessments using the mobile app and maintaining a backend for follow-up cases, the system could improve referral efficiency, reduce unnecessary delays in the healthcare system, and optimize orthodontic care.

1.3 Limitations

Early on in the process it was decided due to time constraints that the mobile application would only be available on iOS, since it has native support for the Core ML runtime for AI models. Furthermore, the size of the dataset available to train the AI models was significantly limited to 626 pictures. Consequently, predictions were unlikely to achieve a satisfactory high accuracy.

Limitations also extend to the evaluation of the models and application. Due to the time-consuming nature of development, testing, and evaluation, it was not feasible to validate all three of the trained models in a real-world screening scenario. Because of this, a small amount of screenings were made, and only YOLOv8s-cls was selected for the stakeholder validation. This choice was made due to YOLOv8s-cls demonstrating the highest accuracy in prior evaluations.

2
Related Work

Advancements in medical imaging technologies, particularly when combined with AI applications such as diagnostic algorithms and CNNs, offer significant potential to enhance patient care, diagnostic accuracy, and personalized treatment plans [8]. As the medical industry increasingly digitizes its diagnostic processes, the underlying computational infrastructure of healthcare organizations becomes more critical. While cloud computing has been widely used for healthcare, with a 2017 survey from Taiwan reporting that 76.2% of hospitals were using some form of cloud computing [9], and another study from Germany [10] found that approximately 89% had adopted cloud-based applications. Furthermore, edge computing offers benefits such as data locality, which can enhance patient security and provide lower latency. Nevertheless, cloud deployments continue to be used, largely due to their ability to dynamically scale resources and manage compute-intensive tasks, such as running large AI models.

The choice of computational infrastructure significantly impacts the deployment of AI models in healthcare. Comparative studies [11], [12], highlight that both edge and cloud computing offers both disadvantages and benefits. Edge computing is often favored for its lower latency and enhanced security through local data storage, rendering it suitable for real-time data processing applications. In contrast, cloud computing provides better reliability, scalability, and flexibility in data storage and hardware maintenance [11], [12]. Moreover, there seems to be a lack of standardization in both edge and cloud computing [11]. As a result, it becomes difficult to develop applications across various devices and platforms.

Beyond architectural considerations, the application of Machine Learning (ML) to specific diagnostic tasks is an ever-increasing field. For example, in a study by Mallishery et al. [13], the researchers tried to predict the referral decision and treatment difficulty in endodontics. Their study utilized two ML algorithms: a custom deep-neural-network and a support-vector machine, one of which achieved a sensitivity of 94.94%. Model performance was evaluated using metrics including accuracy, specificity, sensitivity, and precision. The data to train the models comprised of 500 patient forms collected via an Android app, with data being stored in Firebase, a cloud-based database. The study used a train-test split, noting that variations in the random splits resulted in fluctuations within two percentage points, across all performance metrics.

The application of AI extend to orthodontics, particularly in image analysis. Rubiu

et al. [14], for instance, developed a segmentation model capable of identifying the type of tooth from periodontal radiographs with a 98.4% accuracy. The model was trained on a dataset comprising of 1,000 periodontal radiographs, annotated with masks created by both students and clinical experts. To assess the performance of the trained model, the researchers used a train-test-validation split and evaluated the predictions using the dice index metric and accuracy. The researchers concluded that while the model performed well, reaching an accuracy of 98.4% and a dice index of 0.81, other research had shown that segmentation performance could greatly decrease with radiographs that had missing teeth. Additionally, another study [15] used a YOLO model, another object-segmentation model. The model was trained on intraoral photographs captured on a phone and achieved a sensitivity of 85.6%, precision of 90.7%, and an F1-score of 88.0%, even outperforming junior dentists at detecting tooth decay. The model was trained on a dataset comprising of 7,465 intraoral photographs annotated by experienced dentists.

The YOLO architecture pivotal in the aforementioned study by Adnan et al. [15], has undergone continued improvements. The recent iteration, YOLOv12, reportedly offers even faster inference speeds. These improvements stem from a better attention module, called area-attention, and a residual efficient layer-aggregation backbone designed to minimize attention mechanism overhead. As a result, YOLOv12 achieves a mAP 40.6% on the COCO dataset, an improvement of 1.2 percentage points over YOLOv11n, while preserving comparable latency [16]. Making it even better for real-time object detection.

Despite the advancements in AI models like YOLO, their application in orthodontics has predominantly relied on radiographic data or numerical measurements taken from clinics. Digital images have, by comparison, received less attention even though they are an integral part of the initial diagnosis procedure [17]. Focusing on the practical side of taking normal images instead of radiographs, Shahrul et al. [18] assessed smartphone photos for orthodontic review. Their findings suggest that while Digital Single-Lens Reflex (DSLR) cameras gives superior image quality, standard smartphones can still provide images clear enough for diagnosis. Identifying lighting as the most critical for image quality. Moreover, the study underscores the feasibility of using smartphones for remote orthodontic consultations, making assessments possible outside traditional clinical environments.

Dental Monitoring has in recent time grew into a leading company in AI-driven orthodontic treatment. Their product enables monitoring of the treatments remotely, through utilization of currently available smartphones, and offer a less invasive method for diagnosis [19]. The company's AI model is constructed to identify various dental pathologies or factors, including monitoring of tooth movement and displacement of aligners, from images or 3D scans of the patients teeth taken by themselves. In order to facilitate scans of the teeth, the DM ScanBox was introduced - enabling consistent image quality throughout the scannings [19]. The resulting assessments are then transferred to a web-based interface, where clinicians monitor and assess the ongoing treatment.

On the other hand, the need for dedicated hardware such as a DM Scanbox, may present a barrier to widespread consumer adoption. Supporting this claim, Craven et al. [20] observed in their study of health tracking apps that the introduction of additional procedural steps led to a decrease in weekly diary completion rates from 56% to 45%. Similarly, Fazio et al. [21] developed a mobile app to support dental diagnostics during the COVID-19 pandemic. The app was made to operate exclusively with the phone's native camera, trading external optical hardware giving lower on-device computation for maximum compatibility and accessibility across devices. Moreover, a study by Zhukovska et al. [22] shows the global AI healthcare market is expect to grow to $1.591 trillion by 2030. As this market increases, making these systems more accessible and easy-to-use becomes even more important.

2.1 Summary

Recent advancements in technology have demonstrated the feasibility of AI-assisted, phone-based orthodontic consultations, as exemplified by platforms such as Dental Monitoring. However, this platform requires specialized equipment, which limits their accessibility and adoption. Concurrently, new research highlights the capability of smartphone cameras to capture good orthodontic images, while lightweight models like YOLOv12 have improved real-time performance. Despite these developments, many existing orthodontic AI systems continue to rely on radiographic imaging and numerical measurements from clinics. This reliance may not align with the growing demand for mobile healthcare. Furthermore, there is limited exploration of deployment strategies for such systems, particularly the implications of selecting edge versus cloud-based architecture on factors such as latency, accuracy, and computational efficiency. This presents an opportunity to integrate phone-based orthodontics with advanced AI models while analyzing the trade-offs associated with edge- and cloud-based deployment strategies. A study focused on analyzing these trade-offs could yield significant insights into orthodontic AI systems for clinical adoption, while also giving insights into how mobile technologies can make orthodontic care more accessible.

Key findings

- Existing solutions for remote dental monitoring relies on dedicated external measurement instruments.
- Smartphones are adequate for orthodontics.
- The current body of research is disproportionately centered on radiographic data. Revealing a bias that sidelines photographic images.
- Deployment strategies are unexplored in orthodontic AI systems.
- Distinct advantages:
  - Edge: Low latency and secure local storage for real-time usage.
  - Cloud: Reliable, scalable, and more computing power.
- The latest YOLO model offers improved real-time performance.
- CNNs have been shown to accurately diagnose a variety of oral health problems.

3
Background

This section serves to introduce the medical terminology, in combination with the technologies that lay the foundation for the application and research.

3.1 Healthcare Workflow in Orthodontics

Traditionally, orthodontic diagnosis include both manual assessments by specialists and treatment planning based on expertise by clinicians [23]. The initial assessments do typically occur with general dental practitioners, where malocclusion is screened for during dental check-ups [24]. Patients are subsequently referred to an orthodontist if necessary. The diagnostic phase also includes acquisition of clinical records consisting of cephalometric radiographs for skeletal analysis, intraoral scans for arch evaluation, and both intraoral and extraoral photographs to document dental morphology [25].

The clinical evaluation phase encompasses a detailed assessment of dental relationships, including occlusion and dental alignment [25], [26]. Malocclusion is generally evaluated using standardized systems designed to guide clinical prioritization. One such system is the Skåneindex, which offers a structured framework for evaluating and ranking orthodontic treatment needs. Moreover, the functional performance of the patients is examined - including the ability to chew and articulate speech.

Upon completion of the clinical assessment, a full diagnosis is established where the type and severity of the malocclusion is specified [26]. When indicated, referrals to specialists are made to address specific needs. The treatment process proceeds with routine follow-up consultations, during which the progress is assessed and the treatment plan is refined [25], [26].

An AI-based orthodontic workflow begins with AI algorithms, often powered by computer vision, analyzing photographs of patients teeth. Such advanced AI models have achieved greater accuracy in classifying malocclusion types than general dentists and specialists, while also identifying anatomical landmarks [27]. Based on the assessment from the models, a treatment plan is constructed - which can be reviewed by specialists.

A representation of the described traditional orthodontic workflow compared to an AI-based workflow is shown in Figure 3.1.

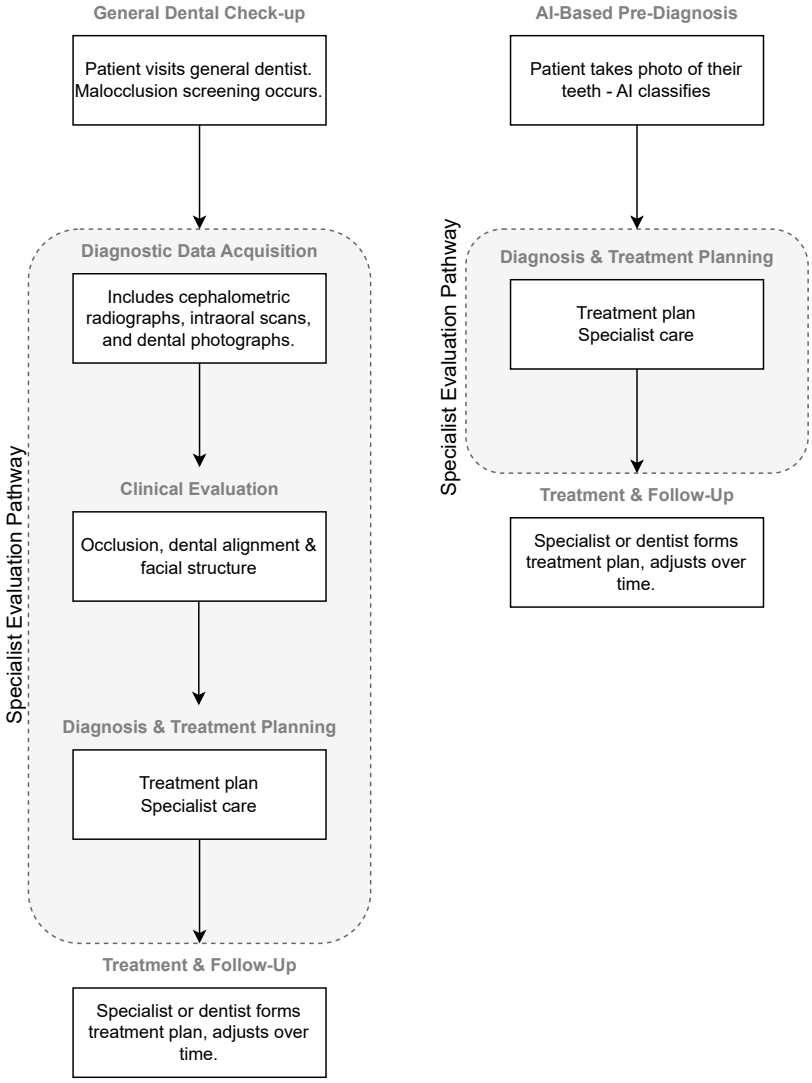


Figure 3.1: The traditional orthodontic workflow encompasses screening by a dentist, referral to a specialist, diagnostic record acquisition, clinical evaluation, and subsequent treatment planning with follow-up. An AI-based workflow begins with classification of photographs, followed by the development of a treatment plan and the execution of treatment.

3.1.1 Skåneindex

In the complex landscape of orthodontic workflow, where clinicians must systematically evaluate and prioritize treatment needs, the Skåneindex is a Swedish prioritization index developed by Folktandvården Skåne, the public dental care unit in the Skåne region. The Skåneindex serves as a standardized method for assessing and prioritizing orthodontic patients.

Upon examination by an odontologist, patients in need of care are divided into the following three groups:

1. Very great need
2. Great need
3. Moderate or low need

Some factors used to assess the patient are, but not limited to: jaw and bite abnormalities, malocclusion, trauma injuries and aplasia. The differences in treatment urgency levels, as seen in Figure 3.2, outline the prioritization process in practice.

Figure 3.2: Visual examples illustrating the application of the Skåneindex for orthodontic priority assessment.

3.2 Convolutional Neural Networks

The technical foundation of the image classification part in this work relies on machine learning architectures for image analysis. As mentioned in Section 1, CNNs can be used to classify dental images with satisfactory results. CNNs are a subgroup of deep learning, commonly used for image processing [28]. This technique involves various fields, such as object detection and image segmentation. Thus, such networks have transformed computer vision and emerged as the leading standard for image classification.

A typical CNN structure consists of several components [28]. The first component is the input layer, followed by the convolution layer equipped with multiple convolution kernels. A fixed-size window is moved by the kernels across a feature map, extracting various feature tiles. Each tile is then multiplied by the learned convolution kernel, resulting in a vector that is reshaped to form a new tensor. The pooling layer introduces down-sampling, and consists of methods such as max-pooling and average-pooling. A Fully Connected Layer (FC) is typically positioned at the final stage in a CNN structure [28]. Its role is to convert the high-dimensional features from the previous layer into a one-dimensional output, making it suitable for classification or prediction.

Activation functions enable the network to identify complex nonlinear patterns effectively [28]. Such functions are applied in convolutional layers and FC layers, where they influence how features are transformed. A commonly used activation function in CNNs is Rectified Linear Unit (ReLU), used because of its ability to alleviate the vanishing gradient problem [28]. Thus, this type allows for more effective training

of deep architectures.

3.2.1 CNN Architectures

There are multiple relevant CNN architectures to consider. VGG-19 depicted in Figure 3.3 is a widely used CNN architecture for image classification, renowned for its remarkable performance - notably in medical image analysis [29]. Due to its straightforward design and ease of training, the VGG-19 - a deep CNN with 19 weighted layers - has excelled in such tasks. Hence, such networks have been utilized for extracting features in various medical applications, such as eye disease detection and Alzheimer's disease prediction [30], [31]. Through pre-training on extensive databases like ImageNet, VGG-19 inherits pre-learned weights, allowing for enhanced performance in transfer learning applications [32], [33].

While VGG networks demonstrate strong performance, they face challenges when scaled to greater depths due to the vanishing gradient problem. ResNet-50, depicted in Figure 3.4, is a type of CNN that has introduced several key innovations and advantages over traditional CNN architectures. ResNet-50 has demonstrated high performance in image classification tasks. For example, ResNet-50 achieved an accuracy of 97.8% in cotton leaf disease classification, outperforming other models like VGG-16 in a study [34]. ResNet-50 introduces residual connections, which help in accelerating weight convergence and preserving patterns and information contained in images. This leads to better accuracy in various tasks, considering dataset size, quality, and task complexity [35], [36].

Beyond classification architectures like VGG and ResNet, object detection networks represent another important category of CNNs. YOLOv8 is a state-of-the-art object detection algorithm that has revolutionized real-time computer vision tasks by enabling fast and accurate object detection in images and videos. Unlike traditional object detection algorithms that involve multiple stages, YOLOv8 approaches object detection as a regression problem, predicting class and bounding box probabilities in a single pass directly from the image pixels [37]. Beyond object detection, YOLOv8 integrates a classification model (YOLOv8-cls), which achieves performance comparable to standard classification models such as VGG and ResNet. Unlike the latter models, YOLOv8-cls include Cross Stage Partial 2-Fused (C2F) module as part of its architectural foundation, which facilitates the processes of feature extraction ad fusion - resulting in an increased ability to identify complex data structures [38]. The architecture of the YOLOv8s-cls model is presented in Figure 3.5, which in contrast to the YOLOv8 object detection model, includes a classification head - comprising of a Global Average Pool followed by a fully connected layer and a softmax activation function to generate class probability distributions.

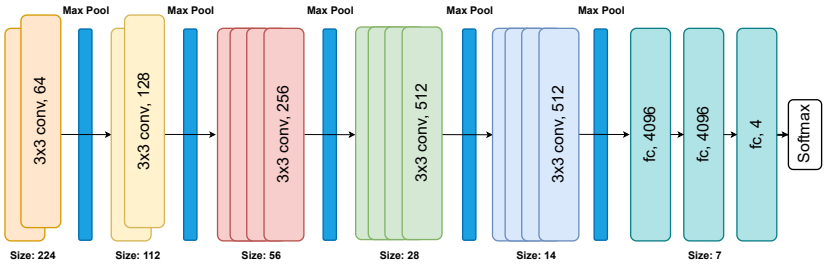


Figure 3.3: VGG-19 containing five convolutional blocks with 3×3 Conv layers and ReLU activations, each followed by 2×2 max pooling. The spatial resolution is reduced from 224×224 to 7×7. The feature maps are then flattened and passed through two fully connected layers with 4096 units and a final layer with 4 outputs for classification.

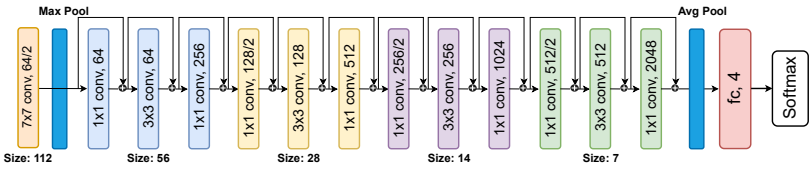


Figure 3.4: ResNet-50 with a 7×7 convolutional layer (64 filters, stride 2) and max pooling, reducing resolution from 224×224 to 56×56. Four residual blocks follow with bottleneck structures (1×1, 3×3, 1×1 convolutions): 64-64-256 filters (stride 1), 128-128-512 (stride 2), 256-256-1024 (stride 2), and 512-512-2048 (stride 2). Spatial dimensions progressively reduce to 7×7, followed by global average pooling, a fully connected layer with 4 outputs, and softmax activation.

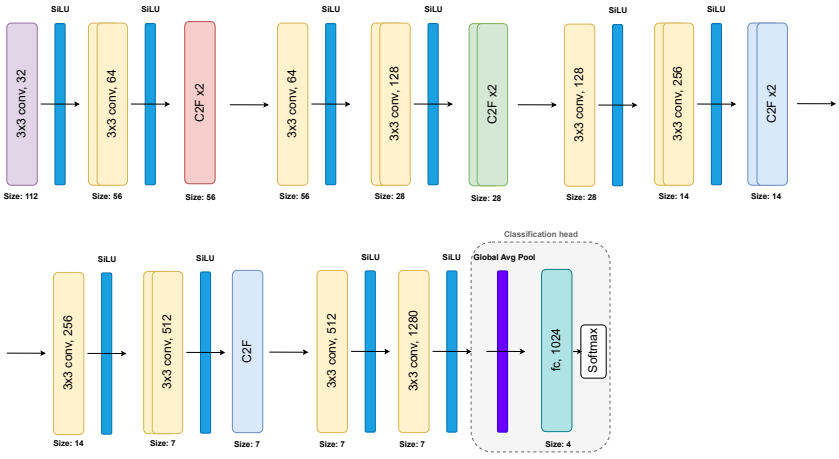


Figure 3.5: Overview of the YOLOv8s-cls model architecture. The model consists of convolutional layers, C2F layers, SiLU activations, Global Average Pool, a fully connected layer and lastly softmax activation.

YOLO11n-seg is an advanced variant of the YOLO family designed for instance segmentation, extending the capabilities of YOLOv8 by integrating pixel-level segmentation masks alongside bounding box predictions and class probabilities [39]. The architecture, as shown in Figure 3.6, processes input images through a backbone, neck, and head to produce both bounding boxes and segmentation masks. The YOLO11n-seg model is more complex as compared to the YOLOv8s-cls model, and it has new layers types: the C3k2 layer, SPPF, and C2PSA [40]. The C3k2 layer, an advanced Cross Stage Partial Bottleneck with 2x2 kernels, enhances feature extraction efficiency by splitting and recombining feature maps. The SPPF module, employing fast max-pooling across multiple scales and concatenation, and the C2PSA block that integrates CSP with parallel spatial attention, focusing on critical image regions to improve accuracy.

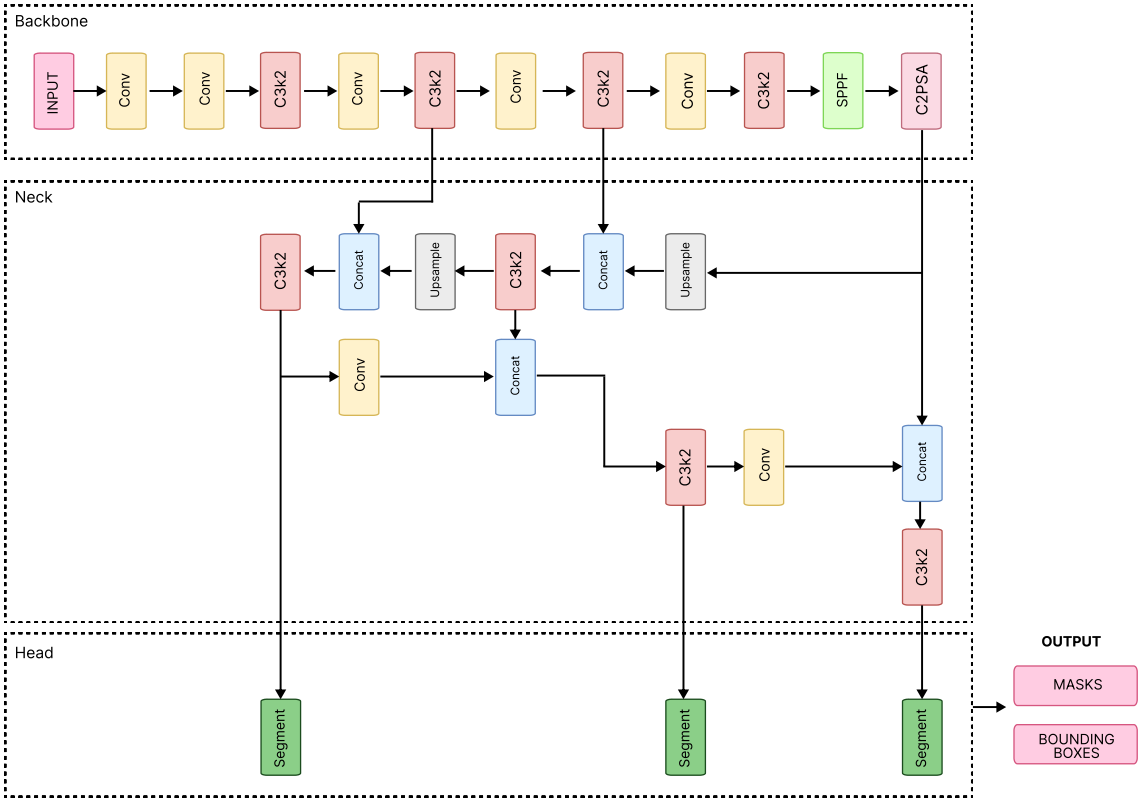


Figure 3.6: Overview of the YOLO11n-seg segmentation model architecture, consisting of the backbone, neck and head, resulting in an output with both bounding boxes and masks.

3.2.2 Transfer Learning and Fine-tuning

Although models like VGG-19, ResNet-50, and YOLOv8 achieve impressive results in classification tasks, training such deep architectures from scratch often demands extensive labeled data and computational power. To address these limitations, transfer learning has emerged as a practical and efficient alternative (see Figure 3.7).

Transfer learning is a crucial technique in the field of deep learning, especially for

image classification problems where labeled data is limited [41]. Rather than training a CNN from scratch, the CNN uses randomly initialized weights and then is trained on a large, corpus such as ImageNet. This pre-training allows the model to learn generic features in early layers and within its deeper layers learn high-level concepts. Next, as shown in Figure 3.7, the CNN's ImageNet output layer is discarded, and replaced with a new output layer based on the target domain. Since ImageNet has about 1,000 classes [42] but the model should only classify four, the CNNs original 1,000 class output layer is replaced with a new four-way classification layer. Then the CNN is fine-tuned, the simplest strategy is to freeze all but the top layers of the pre-trained base model along with the newly added classification layer, and then jointly train them on the target domain [43]. This allows the model to retain the higher-level concepts learned by the pre-training, while the output layers are adapted to the target domain.

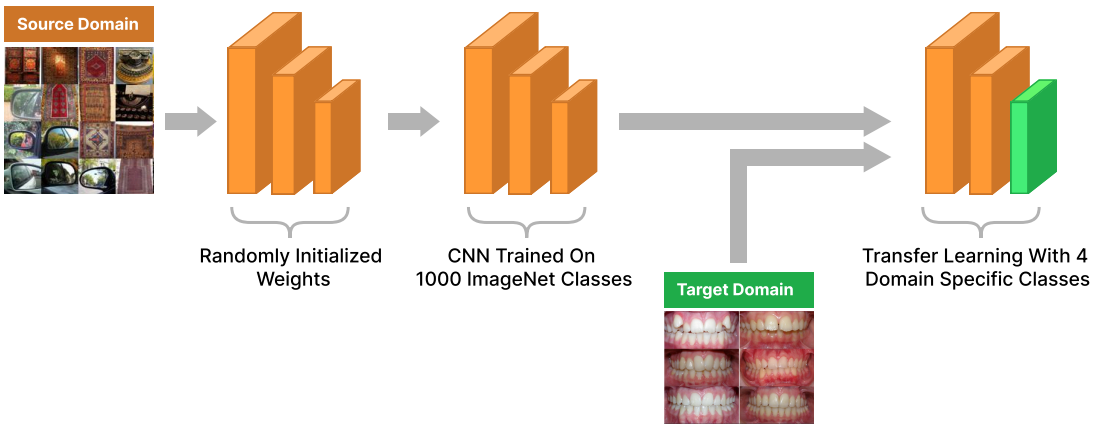


Figure 3.7: High-Level Diagram illustrating the transfer learning process, where a CNN is pre-trained on a large corpus of classified images (source domain), and later fine-tuned on a task specific dataset (target domain). The green layer represents the final non-frozen layers, which is fine-tuned during transfer learning.

3.3 Edge vs Cloud Deployment

To utilize an AI model, it must be deployed within a suitable computing infrastructure. Two common and effective deployment paradigms are cloud computing and edge computing.

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [44]. Thus, clients can utilize an AI model hosted in the cloud by communication through Application Programming Interface (API) calls, requiring minimal computational power of the client.

To implement the cloud computing model, organizations typically rely on established cloud service providers. Among these providers, AWS is one of the most widely adopted [45]. AWS offers a broad set of services, including computing power, storage options, and networking capabilities. Two of the services provided by AWS are EC2 and S3. EC2 is a web service that provides resizable compute capacity in the cloud. An EC2 instance is a virtual server that can be used to run applications in AWS [46]. S3 is an object storage service offering industry-leading scalability, data availability, security, and performance [47]. Files uploaded to Amazon S3 are stored within a container, called an S3 bucket. Each object is associated with a unique key, which allows users to access a specific object on demand. Buckets and the objects in them are private and can only be accessed with explicitly granted access permissions [47]. Thus, Amazon S3 can be set up in a way that ensures secure storage of confidential information, such as patient pictures.

In contrast to the centralized approach of cloud computing, edge computing is a networking technology that enables devices in remote locations to process data and perform actions in real-time. It works by minimizing network latency through processing most data at the edge of the network, such as by the device itself or by a nearby server [48]. Running an AI model on a local device has the potential to reduce network latency; however, the computational power on edge is subject to the hardware capabilities of the device. These hardware restrictions will require a smaller model size compared to running the model in the cloud, which could introduce a diminishing performance and accuracy. The two computing architectures can be seen in Figure 3.8.

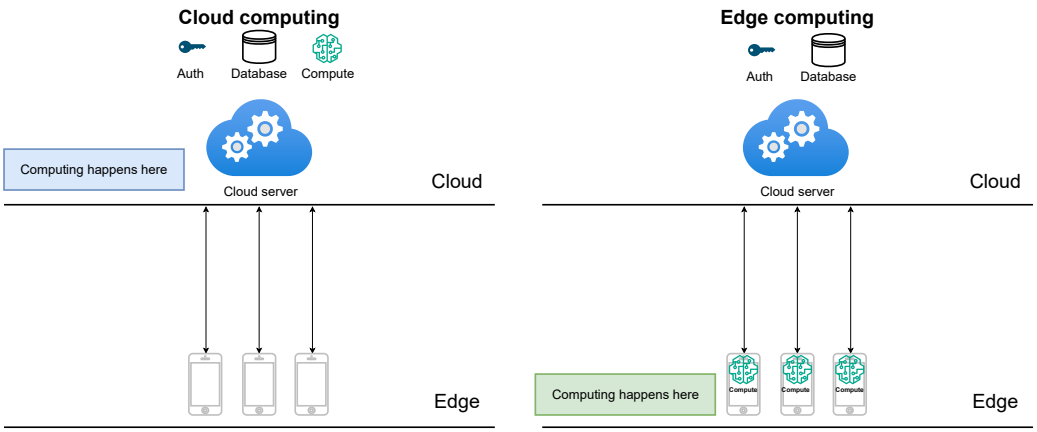


Figure 3.8: Comparison of Cloud Computing and Edge Computing Architectures. On the left side, cloud computing is depicted with data sent from user devices to a centralized cloud server where computation occurs. On the right side, edge computing shows data being processed locally at the edge of the network.

3.4 AI Inference Runtime

Inference runtime refers to the phase in the ML life cycle where a trained model is used to make predictions on new data. During inference, the model applies the learned parameters to new inputs to generate outputs, typically using forward propagation [49]. Depending on whether the AI model is deployed in the cloud or on edge, different inference runtimes are required.

For cloud deployments, the ORT is a high-performance inference engine for deploying machine learning models, including CNNs, across various hardware platforms. The Open Neural Network Exchange (ONNX) exchange format provides a common format to represent deep learning models, enabling developers to move models between different frameworks and tools easily [50].

For edge deployments, the Core ML framework provides a robust solution tailored for running machine learning models directly on Apple devices. The Core ML framework is a powerful tool designed to integrate machine learning models into applications, particularly on Apple platforms. It allows developers to leverage pre-trained models for various tasks, such as image recognition, natural language processing, and more, directly within their apps. By allowing models to run directly on the device, Core ML ensures high performance and low latency, which is crucial for real-time applications. This on-device processing also enhances user privacy, as data does not need to be sent to external servers for analysis [51].

4
Method

This section describes the methodological approach of the project by defining the working process. The research methodology, problem investigation, data collection and prototyping will be presented and discussed.

4.1 Research Methodology

This project adheres to the Design Science Methodology outlined by Roel J. Wieringa [7], which provides a structured, iterative approach to solving design problems and answering knowledge questions through the development and evaluation of artifacts. In this study, the artifact is the AI-powered diagnostic system. This methodology is particularly suitable for our research, as it aligns with our goal of systematically comparing cloud-based and edge-based AI deployment for malocclusion diagnosis. The methodology emphasizes the importance of an iterative process, stakeholder validation, and empirical evaluation, all of which are central to our investigation of AI deployment strategies in orthodontics.

Specifically, we follow "The Design and Engineering Cycle", an iterative framework consisting of problem investigation, treatment design, evaluation, and validation. This cycle allows for systematic evaluation of model performance based on key metrics such as latency, accuracy, and resource efficiency. Through this process, we ensure that the AI system is optimized iteratively, validated with an expert, and assessed for real-world applicability. By adopting this methodology, we ensure a structured and repeatable research process that balances technical feasibility with practical usability.

4.2 Problem Investigation

The deployment of AI-powered diagnostic systems in healthcare requires a balance between performance, usability, and clinical feasibility. In this context, three key areas were identified to assess the system's effectiveness and to support engineering decisions. The definitions and the rationale for selecting each metric is provided in Section 4.2.1.

- **Latency**, defined as the total time elapsed from the moment an image is submitted by the user to when a diagnostic result is presented. This metric captures the responsiveness of the application and its suitability for real-time

usage. Further details and measurement procedures are presented in Section 4.3.3.

- **Model performance**, referring to the model's ability to correctly identify the level of malocclusion. This is quantified using standard classification metrics including F1-score, accuracy, sensitivity, specificity, and precision. More details and measurement procedures are presented in Sections 4.3.2 and 4.3.5.

- **Resource Efficiency**, evaluated based on CPU and memory utilization during inference. These metrics are crucial to understand the feasibility of running models in constrained environments such as mobile devices. More details and measurement procedures are presented in Section 4.3.4.

According to Wieringa's *Design Science Methodology*, problem investigation focuses on understanding the factors that affect the artifact's performance in its intended context before developing a solution. The chosen metrics were selected for their direct relevance to stakeholder goals such as diagnostic reliability, accessibility, and real-time usability.

4.2.1 Justification for Selected Metrics

Previous research in AI-driven healthcare applications highlights the critical role of latency, accuracy, and resource efficiency in real-world deployments. Studies on medical AI inference suggest that:

Latency: Low latency is crucial for maintaining user satisfaction, especially in real-time applications. For example, a study on AI-enabled IoT healthcare solutions highlighted that low latency is essential for timely patient care, which directly impacts user satisfaction [52]. To meaningfully assess latency, this thesis adopts recognized thresholds based on research on human-computer interaction by Nielsen [53], who identified 0.1 and 1.0 seconds as critical perceptual boundaries for interactive system performance. A latency of 0.1 seconds is considered the upper limit for the system to feel instantaneous, while 1.0 second marks the threshold for maintaining the users flow of thought without noticeable interruption.

AI Metrics: AI systems are increasingly used for diagnostic purposes, such as medical imaging and disease detection. High accuracy in these systems is crucial to avoid misdiagnoses, which can lead to inappropriate treatments and adverse patient outcomes [54]. A strong F1-score, such as the 88.0% achieved in one of the studies referenced in the related work section, demonstrates the potential of AI models in diagnostic application. However, given the limited size of the dataset and class imbalance, the objective is not to reach benchmark-level performance, but to compare the relative trade-offs between edge and cloud deployment strategies. Model performance is therefore interpreted in the context of deployment impact, rather than as an absolute measure of diagnostic capability.

Resource Efficiency: Efficient resource management is vital due to limited computing power and the need to optimize energy consumption, especially in edge computing [55]. Optimizing resource usage in cloud computing is essential for upholding a reliable service quality, and the high energy demands of such infrastructure highlight the need for strategies to manage resource consumption [56], [57]. According to performance thresholds defined by BrowserStack [58], a third party app performance testing platform, iOS applications should aim for CPU usage below 20% and memory usage below 250 MB.

By evaluating these metrics, this study aims to quantify the trade-offs between edge and cloud-based inference, ensuring that the AI system is not only technically viable but also clinically usable and sustainable in practice.

4.3 Data Collection

To evaluate the trade-offs between the various architectures, the group collected data for all the three different phenomena.

4.3.1 Training Data

The amount of data used to train the model is integral to its prediction quality. Consequently, the group were given access to an unlabeled dataset of 477 close-up images of individuals' smiles, taken by the stakeholder. The group gathered 149 additional images for a total of 626 images. Three members of the group independently labeled the images by following the Skåneindex guidelines as explained in section 3.1.1, with the addition of a fourth category "normal", where malocclusion is not prevalent. Note that Skåneindex is used to assess multiple orthodontic phenomena, however, the group solely based the decisions on the degree of malocclusion as requested by the odontologist. The pictures in which the three members of the group independently decided on the same label were kept. In the instances where a picture was assigned more than one label, the picture in question got forwarded to the odontologist to assert a correct label. The distribution of the 626 images can be seen in Table 4.1:

Table 4.1: Class distribution between the four classifications; Normal, Low Priority, Medium Priority and High Priority. The percentages are rounded to one decimal.

Classification	**Images**	**Percentage of dataset**
Normal	429	68.5 %
Low Priority	72	11.5 %
Medium Priority	77	12.3 %
High Priority	48	7.7 %

Additionally, the group collected and processed data for a teeth visibility validation model. A dataset consisting of 2495 segmented images of teeth [59] was obtained

for training this instance segmentation model.

4.3.2 Metrics

To evaluate the deployed models, we ran inference on all available images in the validation set using each model in both edge and cloud environments. The same image dataset was used across both environments to ensure consistency. In the cloud configuration, the model was hosted on an AWS EC2 instance, with image data transmitted with Hypertext Transfer Protocol Secure (HTTPS) requests. For the edge deployment, models were integrated into the mobile application using Core ML.

During inference, we recorded the predicted class and compared it with the true class label for each image. Additionally, the inference latency (in seconds), CPU usage (percentage), and memory usage (megabytes) was logged for each run. These metrics were collected per image, allowing for direct, paired comparison between the two deployment configurations. The collected data was used to compute average latency, accuracy, CPU usage, and memory usage per model, per environment. These metrics were visualized using bar plots, box plots, and summary tables. Paired t-tests were conducted to determine whether differences in performance between the edge and cloud environments were statistically significant. A confidence level of $\alpha = 0.05$ was used for all statistical tests. In addition to deployment metrics, we evaluated each model's performance using standard classification metrics, including F1-score, accuracy, sensitivity, specificity, and precision. These results are summarized in a comparative table to highlight differences between edge and cloud configurations. The number of epochs are also included to provide context.

To further illustrate performance differences, confusion matrices are shown for the different models. The evaluation pipeline for image classification is presented in Figure 4.1. Each model is evaluated on the validation set, yielding the confusion matrix and classification metrics described above. Emphasis is placed on sensitivity, as minimizing false negatives is critical to the application's domain. Specificity is also considered a priority to ensure true negatives are correctly identified. This focus enables a balanced assessment of the diagnostic performance of the models.

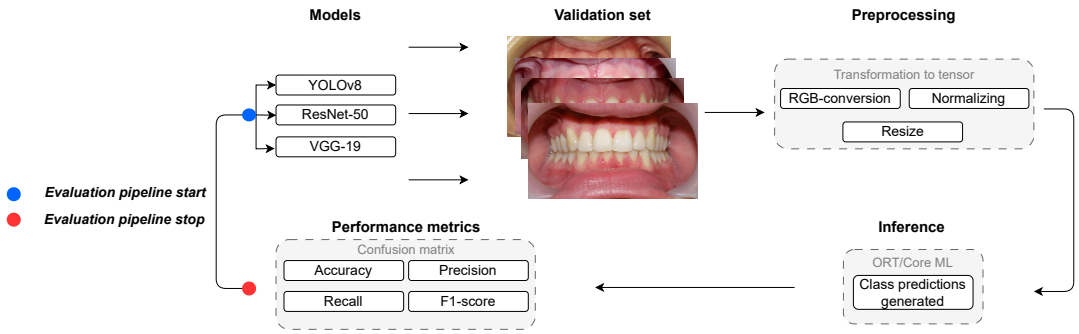


Figure 4.1: The evaluation workflow begins with preprocessing the validation set, followed by class predictions being generated using ORT/Core ML. Subsequently, a confusion matrix is constructed and performance metrics are computed.

4.3.3 Latency

In order to measure the difference in latency between running the model on the edge versus in the cloud, timestamps were recorded during runtime for each processed image. As shown in Figure 4.2, the timestamps that were recorded were the following:

- When the mobile application sends an image for model prediction.
- When the result is received by the client.

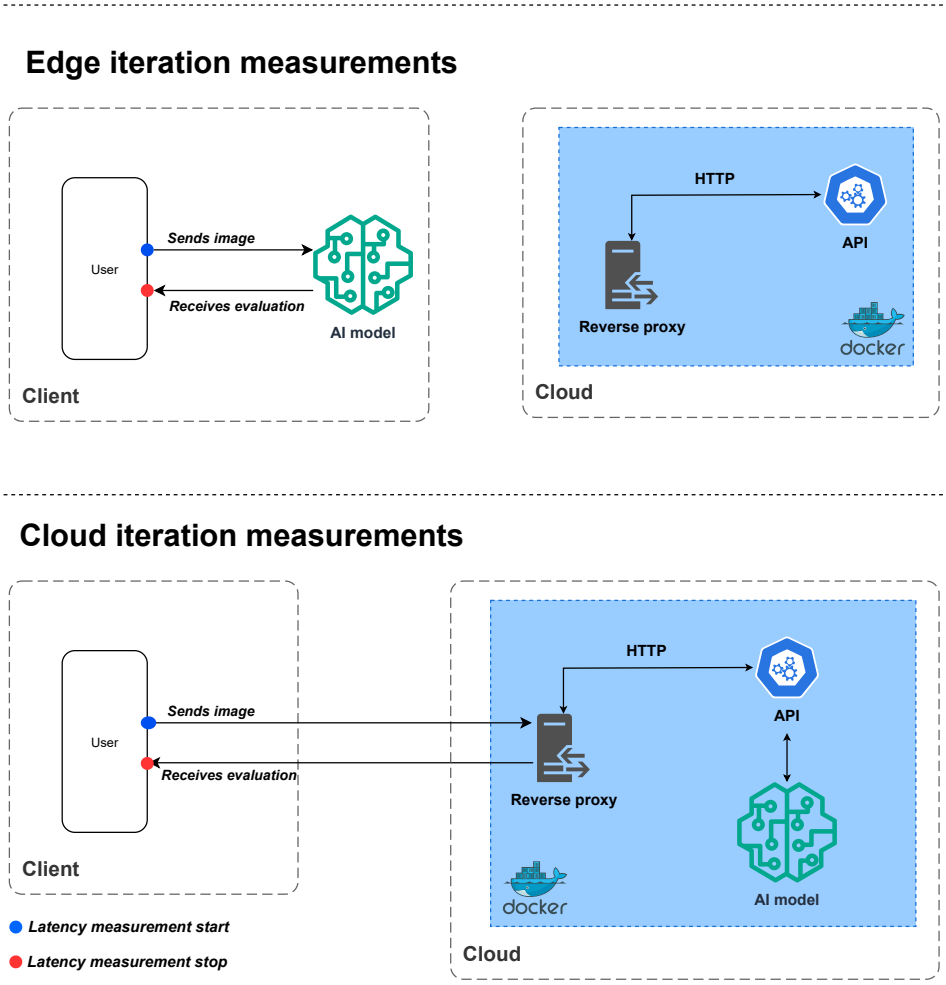


Figure 4.2: End-to-end latency measurement in the edge iteration (top), end-to-end latency measurement in the cloud iteration (bottom).

This procedure was carried out for every image in the validation dataset across all three models, deployed in both cloud and edge configurations. The total latency for each image was calculated as the difference between these two timestamps.

4.3.4 System Resources

To evaluate resource efficiency, CPU and memory usage measured during model inference on both edge and cloud systems. A custom performance monitoring class was implemented in Swift for the edge deployment. This monitor periodically sampled the system's CPU and memory usage using low-level iOS APIs, providing the application's footprint in real time.

The core CPU usage monitoring logic shown in Figure A.1 uses the Darwin framework, which provides kernel-level APIs for resource monitoring. The full implementation can be seen in Appendix A.3, where the key aspects of this implementation are:

- The use of the `host_processor_info()` function, which is a Darwin kernel API that retrieves processor information.
- The processing of different CPU state ticks (user, system, idle, nice).
- The averaging of CPU usage across all available cores to provide a consolidated metric.

The memory monitoring implementation shown in Figure 4.3 also uses the Darwin framework to efficiently capture the application's physical memory footprint. The key parts of this implementation are:

- The `task_vm_info_data_t()` function, which is a Mach kernel API that provides memory information about the current process.
- The use of the `TASK_VM_INFO` parameter to specifically request virtual memory statistics.
- The extraction of the `phys_footprint` field, which represents the actual physical memory consumed by the application, including all the allocated memory regions (which is converted from bytes to MB for easier analysis).

```swift
func getMemoryUsage() -> Double {
    var taskInfo = task_vm_info_data_t()
    var count = mach_msg_type_number_t(MemoryLayout<task_vm_info>.size) / 4
    let result: kern_return_t = withUnsafeMutablePointer(to: &taskInfo) {
        $0.withMemoryRebound(to: integer_t.self, capacity: Int(count)) {
            task_info(mach_task_self_, task_flavor_t(TASK_VM_INFO), $0,
            ↪ &count)
        }
    }
    if result == KERN_SUCCESS {
        return Double(taskInfo.phys_footprint) / (1024.0 * 1024.0) // Convert
        ↪ to MB
    }
    return 0
}
```

Figure 4.3: Function for retrieving the application's physical memory footprint.

Container-level resource usage in the backend (CPU and memory) was also monitored using Docker's statistics API during the classification tasks.

4.3.5 Classification Model Training

A visual overview of the complete training pipeline, including dataset preprocessing, model training and export, is shown in Figure 4.4. This diagram provides an overview of the methodology described below, and is intended to provide clarity on the progression of steps involved, from preprocessing stages to model deployment.

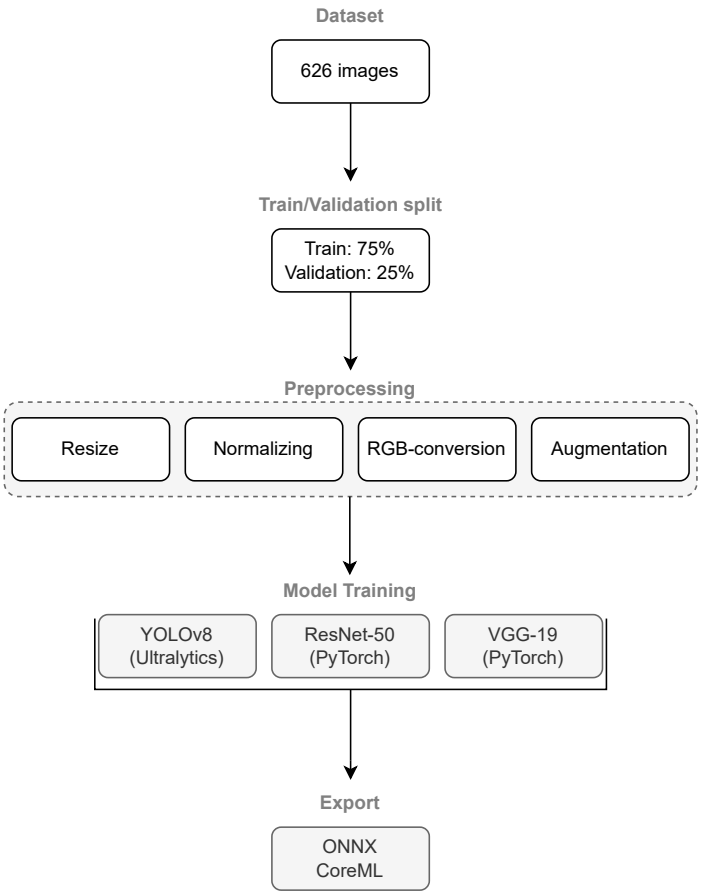


Figure 4.4: Schematic representation of the full training pipeline, illustrating the process from dataset preparation, through train/validation splitting and preprocessing, to model training and export.

As described in Section 4.3.1, the dataset consisted of 626 images, which were divided into training and validation set. The distribution of classes across both subsets are presented in Table 4.2.

Table 4.2: Class distribution in the training and validation sets.

Class	Training	Validation
Normal	318	111
Crooked-low	56	16
Crooked-medium	61	16
Crooked-high	38	10

The models YOLOv8, ResNet-50 and VGG-19 were trained locally, since the dataset could not be uploaded to any cloud service due to privacy reasons. Ahead of training, the images were preprocessed and resized to the desired resolution, 224×224. Furthermore, they were converted to RGB format and normalized using mean and standard deviation values from ImageNet. Data augmentation techniques such as perspective transformation and random horizontal flipping was also applied. These preprocessing steps are visualized in Figure 4.5, which outlines the transition from original input, to resized, augmented and normalized data.

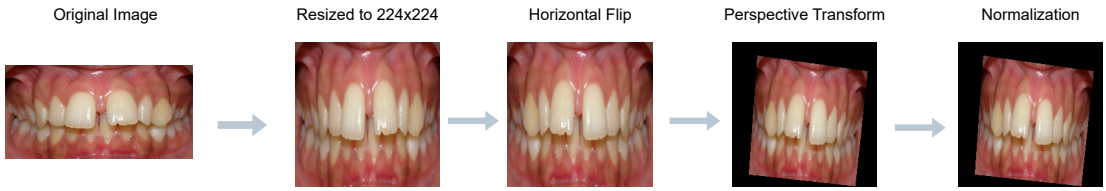


Figure 4.5: Image preprocessing steps: Starting with the original image, followed by resizing, horizontal flipping, perspective distortion and lastly normalization (denormalized for clarity).

Training of the YOLOv8s-cls model was done utilizing the Ultralytics YOLO framework, while both the ResNet-50- and VGG-19-model were trained using the PyTorch framework. To ensure a consistent training procedure, the same core configuration was used across all models. Furthermore, Stochastic Gradient Descent was decided to be employed as the optimizer. Table 4.3 presents the training configuration used in this study, such as number of epochs, batch size, learning rate and momentum. Subsequent to training, the models were exported to ONNX- and Core ML-format for compatibility with both deployment environments.

Table 4.3: Training Parameters for Model Training, including number of epochs, batch size, learning rate and momentum.

Training Parameter	Value
Number of Epochs	10
Batch Size	16
Learning Rate	0.001
Momentum	0.9

In order to convert the ResNet-50 and VGG-19 architectures to ONNX format, representative input tensors, shaped to match the resolution of the dataset, were used to trace the PyTorch models. Moreover, the YOLOv8 model was exported to ONNX format using the functionality provided by Ultralytics. Lastly, the models were validated and visualized in Netron to verify correct input-output nodes.

4.3.6 Instance Segmentation Model Training

In addition to training the classification models, an instance segmentation model was trained in order to detect the number of teeth visible during the image capturing process. While we used YOLOv8s-cls for classification tasks described earlier, we opted for the more advanced YOLO11n-seg model for instance segmentation to detect the number of teeth visible during image capture. YOLO11n-seg was selected over YOLOv8s-cls because it offers pixel-wise segmentation, allowing us to generate precise masks for each individual tooth. The YOLO11n-seg model was trained using a dataset consisting of 2495 segmented images of teeth [59]. As seen in Figure 4.6, the YOLO11n-seg model processes the input image and generates detailed segmentation masks for each detected tooth in addition to bounding boxes.

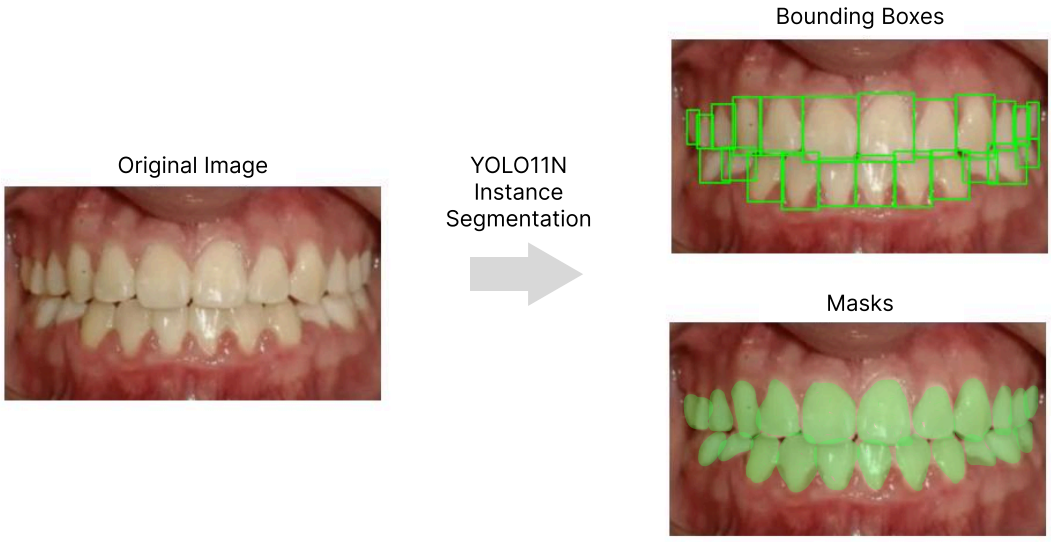


Figure 4.6: Example input and output of the YOLO11n instance segmentation model, producing both masks and bounding boxes.

4.4 Prototyping

The group started the project by developing a Minimum Viable Product (MVP) for each of the two architectures. This MVP was designed to perform the basic functions: receiving images and providing diagnoses. More advanced features such as user authentication, data storage, and image alignment were omitted to focus on

the core functionality. Specifically, the MVP aimed to verify the connection between the phone and the cloud service, as well as to demonstrate that the AI could operate directly on the phone.

Once the MVP was successfully completed, the project transitioned into a phase of incrementally adding features as seen in Figure 4.7. These features, such as image alignment and authentication, were introduced to enhance the user experience and align with stakeholder goals.

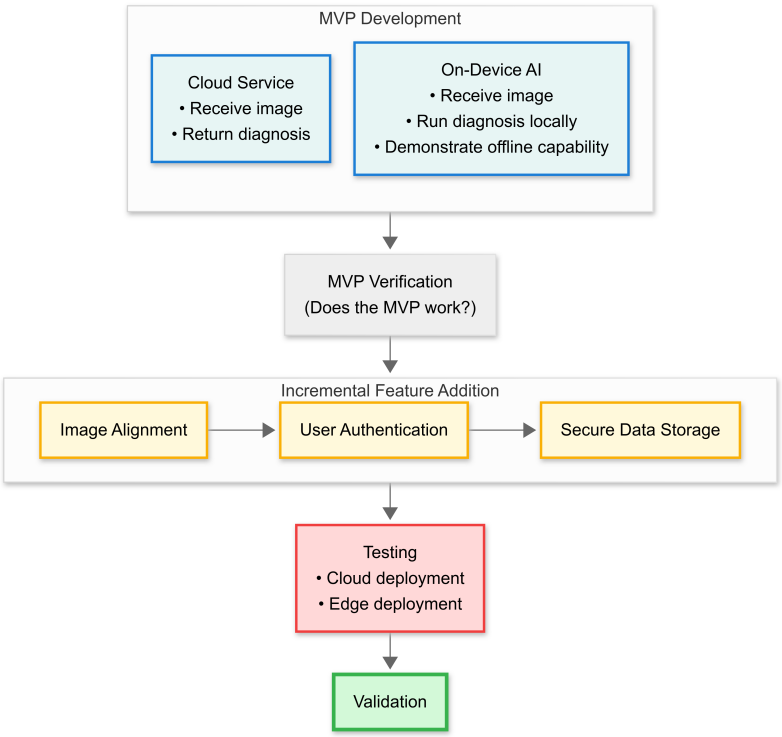


Figure 4.7: High-level diagram showing the transition from MVP to a prototype

4.5 Prototype Evaluation and Stakeholder Validation

The evaluation of the prototypes occurred in two stages: prototype evaluation guided by the project's key performance indicators, and a stakeholder validation involving expert feedback.

4.5.1 Prototype Evaluation

The prototype evaluation compared the performance of the developed models across both deployment environments. This evaluation was structured around the three key performance indicators outlined in Section 4.2: latency, accuracy, and resource efficiency. The goal was to systematically assess the trade-offs between deployment

strategies in terms of technical feasibility and operational efficiency.

To support this comparison, paired t-tests were conducted to evaluate the statistical significance of differences in performance metrics. This quantitative analysis enabled an informed assessment of which deployment approach was better suited to the application's practical requirements.

4.5.2 Stakeholder Evaluation

To validate the practical relevance and usability of the system, a stakeholder evaluation was conducted with the odontologist. The goal was to assess whether the system could be beneficial in the context of clinical screening and to gather feedback on the system's functionality.

A session was organized in which the odontologist interacted directly with the mobile application running the YOLOv8s-cls model on edge. Prior to the session, the system had been used to screen a set of test subjects, and its predictions were recorded without revealing them to the expert. During the session, the expert conducted their own evaluation of each subject, and their assessments were later compared to the predictions of the system.

In addition to this, a semi-structured interview with prepared questions was conducted to gather qualitative insights into the application's usability, clarity, and potential integration into current healthcare practices.

4.5.3 Iteration and Future Improvement

In accordance with Wieringa's Design and Engineering Cycle [7], insights gained through both the prototype evaluation and stakeholder validation were used to identify limitations, highlight areas for improvement, and guide future iterations of the system.

5

Application

The mobile application allows users to receive an AI-powered dental assessment simply by taking a guided photo of their teeth. Using Augmented Reality Kit (ARKit), the app ensures the image is captured with the correct positioning and angles, aligning with the data our model has been trained on. Once the image is submitted, it is processed by a CNN, which extracts relevant features and performs classification. The user then receives an assessment along with a recommendation whether they should visit an odontologist, a dentist, or if no immediate care is needed. Additionally, users can access their past evaluations and remove them if desired.

5.1 Edge vs Cloud configuration

To compare edge and cloud, the mobile application has been developed in two versions: one in which AI inference runs locally on the device (edge) using the Core ML framework, as shown in Figure 5.1, and another in which inference is performed in the cloud via the ONNX runtime, as shown in Figure 5.2.

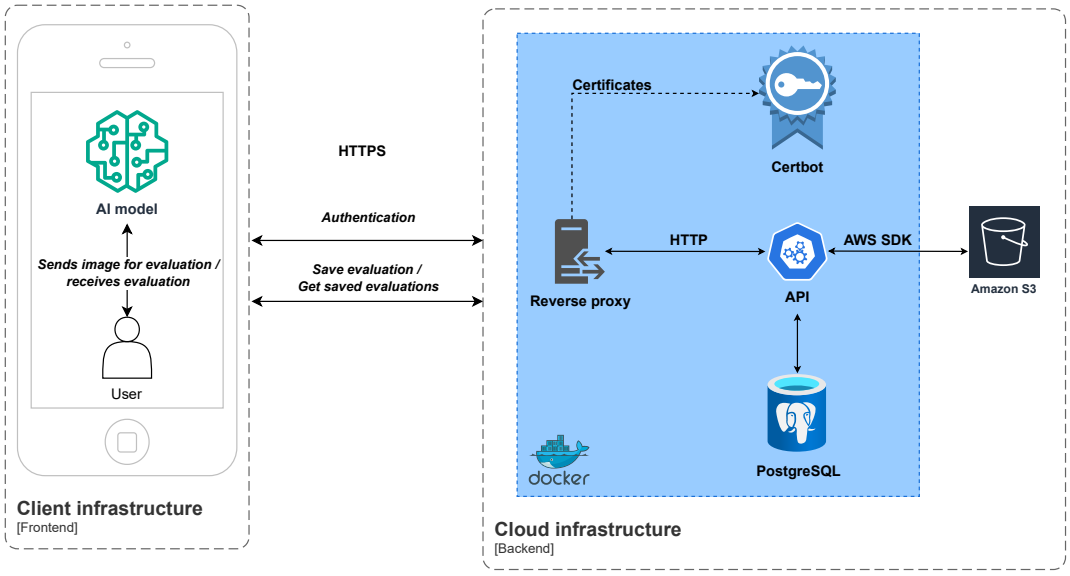


Figure 5.1: Edge iteration system architecture. The client sends an image to an AI model running inference on the edge, receives the evaluation and communicates with the backend for authentication, saving and retrieving evaluations.

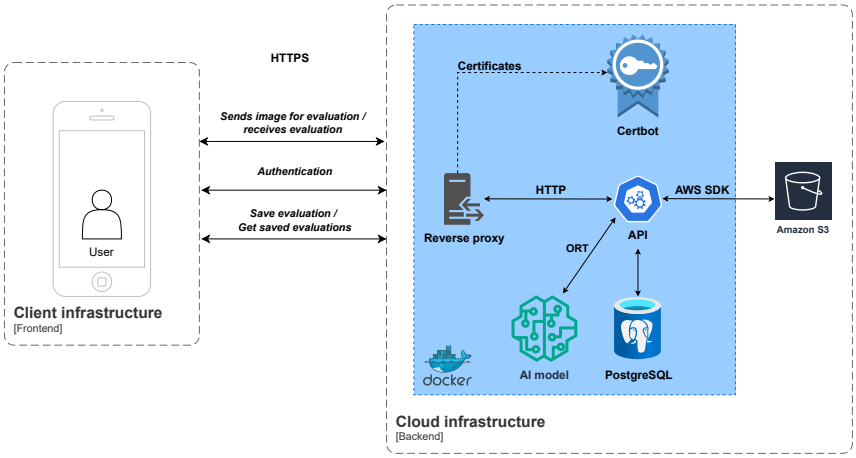


Figure 5.2: Cloud iteration system architecture where a mobile client interacts with a cloud-based AI model. The client sends images to the backend for evaluation, receives results, authenticates and saves/retrieves the evaluations from the cloud.

Both architectures prioritize reproducibility and encapsulation through containerization which isolates each component of the system. As a result, performance measurements remain consistent across product iterations. Both configurations share a common cloud infrastructure for authentication and user data storage.

5.1.1 AI Inference

The cloud configuration runs a CNN within the same container as the API. This is enabled by using ORT along with the `onnxruntime_go` package. The endpoint */ai/predict* accepts a multipart payload containing the image to be classified. The image is first converted to a tensor (see Figure 5.3), which is a multi-dimensional array representing the image's raw pixel values in matrix form for the CNN to use. Inference is then performed using ORT (see Figure 5.4), the input tensor inside of the model is transformed into an embedding i.e. a fixed-length vector that describes which features are present, and how strongly. A final FC layer followed by a softmax transforms this embedding into a class-probability vector returned as `outputData`.

```
1 func (as *aiService) imageToTensor(file multipart.File)
  ↪  (*ort.Tensor[float32], error) {
2   img, _, err := image.Decode(file) // Decode image
3   inputShape := ort.NewShape(1, 3, int64(as.config.InputHeight),
    ↪  int64(as.config.InputWidth)) // Define input tensor shape [1, 3, 224,
    ↪  224]
4   inputData := as.imagePixelsToFloatArray(img)
5   inputTensor, err := ort.NewTensor(inputShape, inputData) // ONNX tensor
6   // ---ERROR HANDLING---
7   return inputTensor, nil
8 }
```

Figure 5.3: Function to convert received image to a tensor.

```go
// Run inference
  if err := as.modelSession.Session.Run(); err != nil {
    return dto.PredictionResponse{}, fmt.Errorf("failed to run inference:
    ↪ %w", err)
  }
  // Retrieve output tensor data
  outputData := as.modelSession.Output.GetData()

  maxIndex := findMaxIndex(outputData) // Find class with highest probability
  predictedClass := as.config.Labels[maxIndex]

  return dto.PredictionResponse{
    Prediction: predictedClass,
  }, nil
```

Figure 5.4: Code segment showing how inference is run in the backend, passing a tensor to ORT.

In the edge configuration, the CNN is run on device with Core ML framework in combination with Apple's Vision framework for preparing input (see Figure 5.5).

```swift
// Classify a UIImage using the loaded model and return the label
func classify(image: UIImage, completion: @escaping (Result<String, Error>)
↪ -> Void) {
    guard let visionModel = self.visionModel,
          let cgImage = image.cgImage else {
        completion(.failure(...))
        return
    }
    let request = VNCoreMLRequest(model: visionModel) { request, _ in
        guard let results = request.results as?
        ↪ [VNClassificationObservation],
              let topResult = results.first else {
            completion(.failure(...))
            return
        }
        let prediction = "\(topResult.identifier) (\(Int(topResult.confidence
        ↪ * 100))%)"
        completion(.success(prediction))
    }
    request.imageCropAndScaleOption = .centerCrop
    let handler = VNImageRequestHandler(cgImage: cgImage)
    try? handler.perform([request])
}
```

Figure 5.5: Image classification on the edge using Core ML and Vision

5.2 Frontend

The frontend consists of a iOS application created in Swift, leveraging several key Apple frameworks, including SwiftUI for user interface, ARKit for real-time face tracking, AVFoundation for camera access, Core ML and Vision for on-device inference. The primary responsibility of the frontend application is to guide the user in correctly positioning their face, and capture a precisely angled and cropped image of the mouth showing the teeth, to perform subsequent analysis. Following the analysis, the user is presented with the predicted priority index and treatment needs.

5.2.1 User Interface

The application's User Interface (UI) is designed to intuitively guide the user through the process of capturing a correctly framed photograph of their teeth. As presented in Figure 5.7, the user is first informed to remove any dental accessories, be in a well-lit room and sit down comfortably to get ready for the scan. Then the camera feed is presented along with instructions to guide the framing correctly. Once the scan is complete the user is presented with the resulting predicted priority assessment, along with information to further guide their choice of dental care. This flow of actions is illustrated below in Figure 5.6.

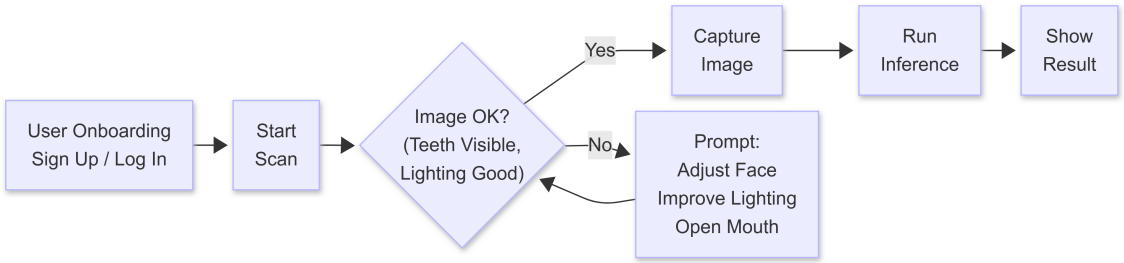


Figure 5.6: User workflow for starting the application and initiating a new dental scan.

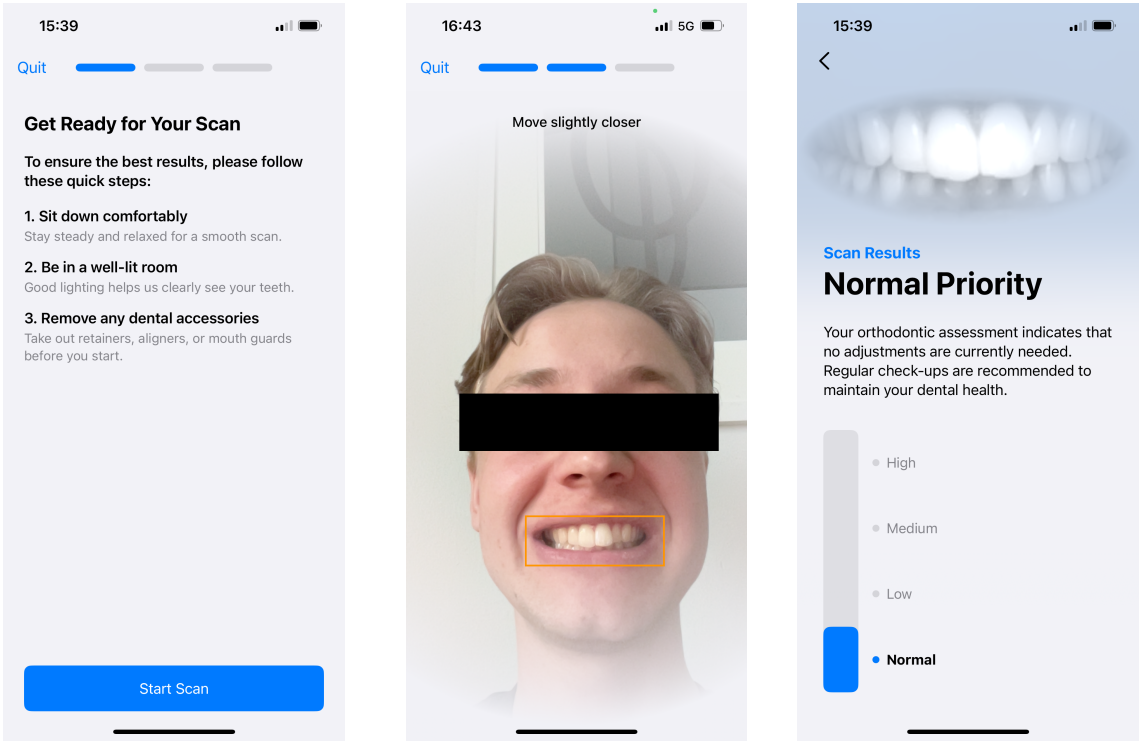


Figure 5.7: Screenshots of each view presented during the image capture process, first informing the user how to prepare prior to capturing a photo, then showing the camera feed and guidance text, ultimately showing the resulting priority assessment scale and patient information.

5.2.2 Instance Segmentation for Teeth Detection

To ensure the captured image of the user's teeth is correctly aligned, taken directly from the front with visible teeth in good lighting, multiple criteria are enforced during the photo capture process. These include relative face position, facial expression, and teeth detection using an on-device YOLO11n-seg instance segmentation model. The lightweight YOLO11n-seg instance segmentation model runs on the device using the Core ML and Vision frameworks.

To prevent performance degradation, inference is performed asynchronously on a background queue using a DispatchSourceTimer, processing frames periodically once a second. The models output is parsed, as seen in Figure 5.8, to find potential teeth detections above a confidence threshold of 0.55. Non-Max Suppression (NMS) with IoU threshold of 0.1 is applied to filter redundant detections. The final count of valid detections is stored and compared against target values.

```
func extractBoundingBoxes(from multiArray: MLMultiArray) -> [(rect: CGRect,
    score: Float)] {
    let confidence = objectness * maxClassScore
    if confidence > 0.55 {
        let boundingBox = CGRect(x: CGFloat(xCenter - width / 2), y:
            CGFloat(yCenter - height / 2), width: CGFloat(width), height:
            CGFloat(height))
        boundingBoxesWithScores.append((rect: boundingBox, score:
            confidence))
    }
    ...
    return boundingBoxesWithScores
}
```

Figure 5.8: Function for extracting the bounding boxes from the YOLO11n-seg output.

5.2.3 Face Tracking and Alignment

When the capture process begins, an ARFaceAnchor instance that tracks face geometry is initialized, from which the relative face position to the device is calculated using their respective transform matrices as $T_{relative} = T_{camera}^{-1} \times T_{face}$ as seen in Figure 5.9. ARFaceAnchor is a class in ARKit that represents a detected face in the camera view, providing access to face geometry, position, and tracking state. The resulting matrix $T_{relative}$ is compared against target values to ensure proper alignment.

```
let relativeTransform = simd_mul(simd_inverse(frame.camera.transform),
    faceAnchor.transform)
let position = SCNVector3(relativeTransform.columns.3.x,
    relativeTransform.columns.3.y, relativeTransform.columns.3.z)
let positionMatch = abs(position.z - (-0.2)) <= 0.05
```

Figure 5.9: Function for enforcing face alignment criteria during capturing process.

The ARFaceAnchor also exposes blend shape coefficients that can be used to track facial expressions, which is employed in the capture guidance to track the degree of which the jaw is open as seen in Figure 5.10, ensuring the teeth are visible.

```
let jawOpenAmount = faceAnchor.blendShapes[.jawOpen]?.floatValue ?? 0.0
let mouthMatch = abs(jawOpenAmount - 0.2) <= 0.1 && detectedObjectCount > 5
...
```

Figure 5.10: Function for checking face expression during capturing process.

5.2.4 Image Capture and Processing

To isolate the specific Region of interest (ROI) containing the teeth for analysis, a precise mouth cropping mechanism was implemented. This process uses the facial landmark tracking of ARKit to accurately identify and extract the relevant oral region.

The cropping algorithm makes use of key facial vertex points from the ARFaceAnchor's geometry to define a bounding rectangle around the mouth, specifically four critical landmarks are tracked: the upper lip (vertex index 21), lower lip (vertex index 27), left mouth corner (vertex index 173), and right mouth corner (vertex index 622). The vertices are first transformed from local face coordinates to world coordinates using the face node's transformation matrix as shown in Figure 5.11.

```
worldPosition = faceNodeTransform * vertexPosition
```

Figure 5.11: Code to transform from local face coordinates to world coordinates.

The resulting 3D world coordinates are then projected onto the 2D camera plane using ARSCNView's projection method. This yields a set of four screen-space coordinates that define the outer bounds for the mouth region. The final bounding rectangle is calculated by finding the minimum and maximum values among the projected points as seen in Figure 5.12.

```
minX = min(upperLipX, lowerLipX, leftCornerX, rightCornerX)
maxX = max(upperLipX, lowerLipX, leftCornerX, rightCornerX)
minY = min(upperLipY, lowerLipY, leftCornerY, rightCornerY)
maxY = max(upperLipY, lowerLipY, leftCornerY, rightCornerY)
```

Figure 5.12: Logic for defining the bounding box from the screen space face vertex coordinates.

When all the above mentioned framing criteria are satisfied for one second, the application proceeds to capture and crop the image, whilst accounting for the difference between the viewport display size and the actual camera image resolution by applying the scaling factor: `scaleFactorH = imageHeight / viewportHeight`.

5.2.5 Authentication

To ensure secure access to user data and maintain privacy compliance, the application implements a comprehensive token-based authentication system. The system

manages user authentication through a combination of access and refresh tokens, enabling persistent sessions while maintaining security. The main authentication logic in the frontend can be seen in Figure 5.13.

Following successful initial authentication, the application securely stores the received access token, refresh token, and their associated expiration times using the Keychain service. Before executing authorized operations, the application verifies the validity of the access token by checking its expiration status. If the access token has expired the application utilizes the stored refresh token, if it is still valid, to automatically request a new access token from the backend endpoint. If the refresh token also expired or the renewal process fails, the current session is terminated. Part of this authentication logic can be seen in Figure 5.13.

```
func getValidAccessToken(completion: @escaping (String?) -> Void) {
    guard let accessToken = keychain.get(forKey: "access_token") else {
        completion(nil)
        return
    }
    if let expiry = parseISO8601DateToTimestamp(...),
       Date().timeIntervalSince1970 < expiry {
        completion(accessToken)
        return
    }
    // Additional token validation logic omitted for brevity

    if Date().timeIntervalSince1970 < refreshExpiry {
        refreshAccessToken(refreshToken: ...) { newToken in
            completion(newToken)
    // Error handling and cleanup logic omitted
}
```

Figure 5.13: Swift authentication function that validates existing access tokens, checks token expiration against current time, and initiates token refresh when needed.

5.3 Backend

The backend, deployed on AWS, follows the cloud infrastructure depicted in Figure 5.14. All incoming requests are first received by a Reverse proxy server (nginx), configured with HTTPS using Let's Encrypt certificate automation tool (Certbot) for certificate automation, which forwards the requests to the API. The API then communicates with an object-relational database (postgreSQL) and S3 for reliable data storage and retrieval.

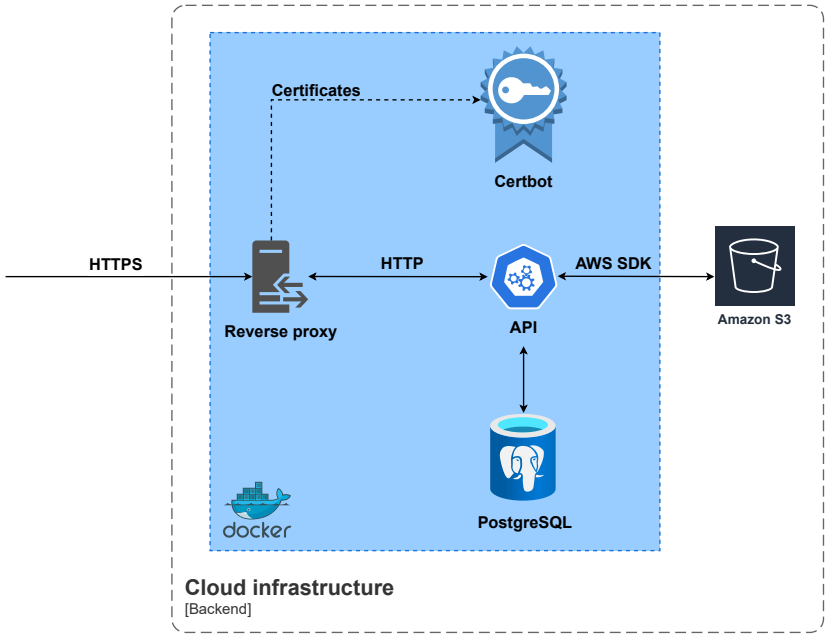


Figure 5.14: The diagram illustrates the cloud infrastructure, highlighting EC2 running Docker containers, secure communication via Nginx, automated certificate management, and integration with AWS S3 storage.

The API is developed in Golang and is designed to support several key functionalities: storing evaluation history, managing user authentication via JSON web token (JWT), and executing AI inference operations. The application follows a handler-service-repository architectural pattern: handlers process incoming client requests, the service layer implements the business logic, and the repository layer manages data access and interaction with the postgreSQL database. An overview of all available endpoints is provided in Table A.2 in Appendix A.2.

5.3.1 Authentication

Protected endpoints require the user to be logged in. Authentication is implemented using JWT, as defined in RFC 7519 [60], a widely adopted standard for securely transmitting claims between parties.

Upon successful login, the server issues a signed access token and a refresh token, and creates a session in the database. To authenticate subsequent API calls, the client must include the access token in the request headers. The protected routes listed in Table A.2 are secured by middleware responsible for verifying the access token. Token verification is performed in two steps. First, the `Authorization` header is extracted and parsed. Then, the token is verified using the JWT secret.

5.3.2 Data storage

The postgreSQL database holds user credentials (emails, names, and hashed passwords using the bcrypt algorithm), session data, and evaluation records.

Each evaluation entry associates a user ID with its classification result and a corresponding image key stored in the S3 bucket, which contains all submitted images. When users request their evaluation history, the backend generates presigned URLs for each image, providing secure, time-limited access restricted to the authenticated user.

6
Result

This section presents the results of the study, comparing edge and cloud deployments for VGG-19, ResNet-50 and YOLOv8. In addition to the performance metrics and comparative analysis, it integrates feedback from a key stakeholder gathered through a semi-structured interview.

6.1 Classification Model Accuracy

In the following, the VGG-19, ResNet-50 and YOLOv8 model accuracies are assessed and presented through their confusion matrices, displayed in Figure 6.1, and the key performance metrics: accuracy, precision, recall, F1-score, and specificity for each model are shown in Table 6.1.

Table 6.1: Comparison of YOLOv8, VGG-19 and ResNet-50 on key performance metrics.

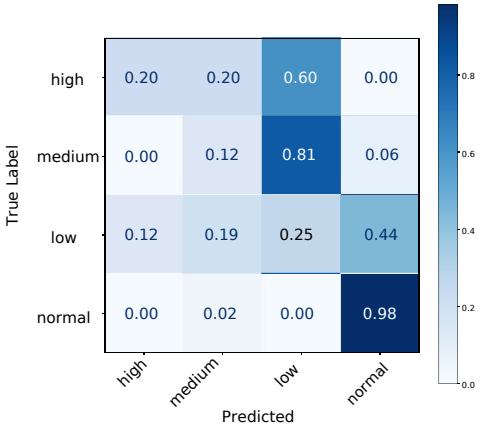
Metric	**YOLOv8**	**VGG-19**	**ResNet-50**
Accuracy	**0.8301**	0.7386	0.7386
Precision	**0.8025**	0.5854	0.6034
Recall	**0.8301**	0.7386	0.7386
F1-score	**0.8072**	0.6490	0.6586
Specificity	**0.8485**	0.7604	0.7693

The obtained confusion matrix for YOLOv8 is shown in Figure 6.1a, revealing balanced predictions across the classes, with confusion among the crooked categories. Moreover, the computed YOLOv8 metrics are presented in Table 6.1, highlighting the balanced performance across all key metrics, including an accuracy exceeding 83%. Furthermore, it is evident that the model achieves high specificity and F1-score - 0.8485 and 0.8072 respectively. The obtained precision was 80.25%, indicating the fraction of true positives to the total number of instances classified as positive by the model. With a recall of 83.01%, the model effectively captures the majority of relevant positive cases, whereas the achieved F1-score underscores a balanced performance between precision and recall. Furthermore, the specificity reflects the model's ability to accurately identify true negatives.

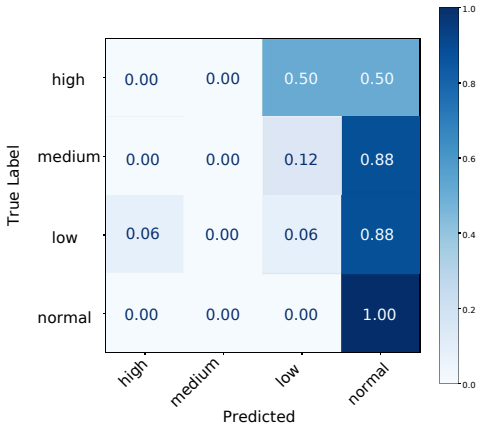
Moreover, the resulting confusion matrix for VGG-19 is illustrated in Figure 6.1b, demonstrating that the "normal" category is classified with high consistency while

misclassifications are primarily observed within the other classes. The corresponding metrics is shown in Table 6.1. The VGG-19 model achieved an overall accuracy of 73.86 %, indicating that the model correctly classified approximately three out of four instances. However, the precision score of 58.54 % suggests that among the positive classifications, a notable portion were incorrect. In terms of sensitivity, the recall was also 73.86 % - meaning most of the actual positives were identified. Lastly, the specificity was measured at 76.04 %, while the F1-score stood at 64.90 %.

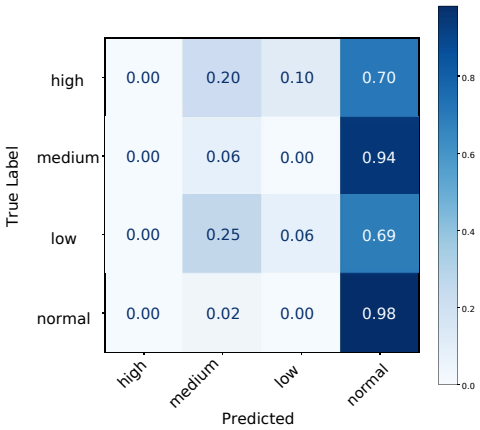
The confusion matrix of ResNet-50, depicted in Figure 6.1c, indicates that the model exhibits challenges with distinguishing the degrees of misalignment, with somewhat accurate classification of "normal" cases. Despite challenges in distinguishing the degrees of misalignment, the models overall performance metrics still shows consistent classification capabilities. Achieving an accuracy exceeding 73 % and a reasonable balance between recall and precision. The precision of 60.34 % indicates that it is able to detect a large proportion of positive cases, though the recall of 73.86 % suggests that most actual positive cases were successfully identified. With an F1-score of 65.86 %, derived from the model's precision and recall, the result indicates a reasonable overall effectiveness in distinguishing between the classes.



(a) YOLOv8



(b) VGG-19



(c) ResNet-50

Figure 6.1: Confusion matrices, illustrating the distribution of predicted vs actual classes, for the evaluated models: (a) YOLOv8, (b) VGG-19, and (c) ResNet-50.

6.2 Instance Segmentation Model Accuracy

To evaluate the performance of the trained instance segmentation model, validation metrics were monitored throughout the training process. The primary metrics used were mAP for both bounding box detection and mask segmentation.

Figure 6.2 illustrates the evolution of the validation mAP_{50} score, calculated at a fixed IoU threshold of 0.50. Both the bounding box and mask mAP_{50} scores show a clear upward trend during training, indicating successful learning. The mask mAP_{50} is slightly lower than the bounding box mAP_{50}, reflecting the increased difficulty of precise pixel-level segmentation compared to object localization.

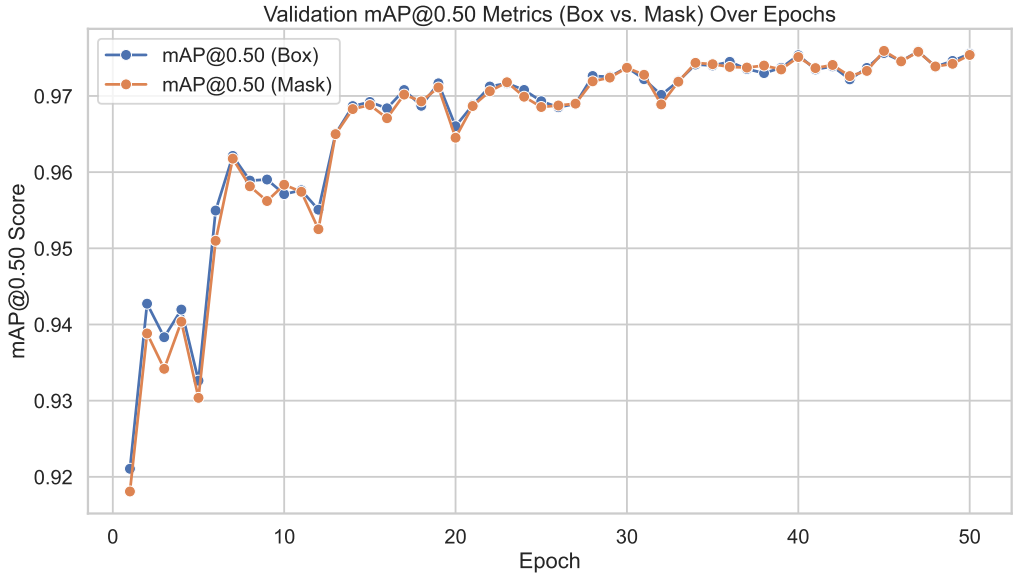


Figure 6.2: Validation mAP at 50% IoU threshold (mAP_{50}) over training epochs.

Figure 6.3 presents the validation $mAP_{50:95}$ scores which represent the average mAP calculated over multiple IoU thresholds ranging from 0.50 to 0.95, which provides a more comprehensive assessment of the model's localization accuracy. Similar to the mAP_{50} results, the $mAP_{50:95}$ scores for both boxes and masks increase steadily throughout training before stabilizing. The final $mAP_{50:95}$ value of about 0.86 for the masks demonstrates a robust capability of the model to accurately delineate tooth boundaries across various overlap criteria.

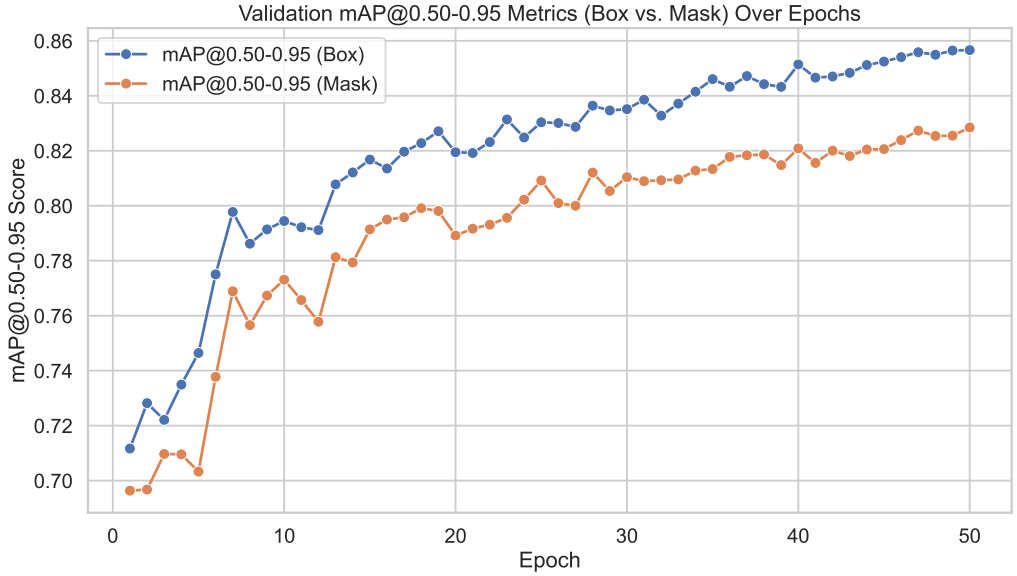


Figure 6.3: Validation mAP averaged over IoU thresholds from 0.50 to 0.95 ($mAP_{50:95}$) over training epochs.

6.3 Cloud vs Edge

This section presents the performance comparison between edge and cloud deployment across the three models recorded from the device. Metrics evaluated include inference time, accuracy, memory usage, and CPU usage. Paired t-tests were conducted to assess the significance of observed differences.

Figure 6.4 shows the mean inference times for deployments in edge and cloud environments for each model. On average, edge deployment resulted in faster inference.

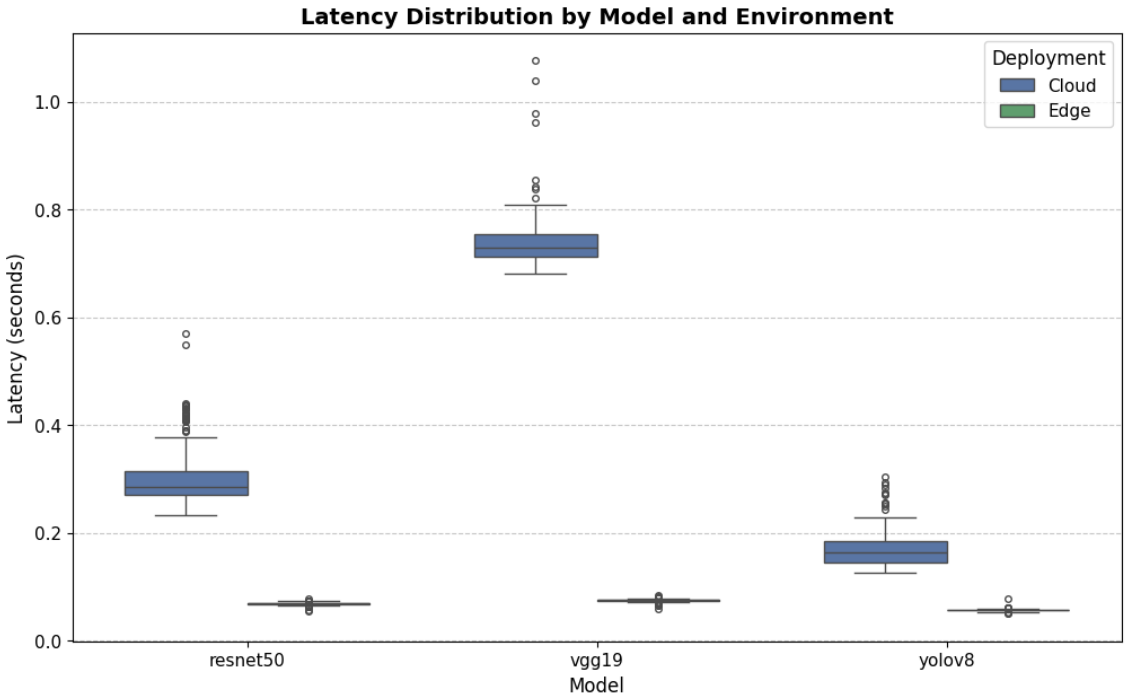


Figure 6.4: Visualization of the differences in latency by model and environment.

Paired t-tests revealed statistically significant reductions in latency for all models when deployed on edge devices (see Table 6.2).

Table 6.2: Paired t-test results for latency (Cloud vs Edge)

Model	Mean Latency (Cloud)	Mean Latency (Edge)	t-value	p-value
ResNet-50	307.3 ms	69.1 ms	-47.41	0.0000
VGG-19	742.9 ms	97.9 ms	-29.78	0.0000
YOLOv8	171.7 ms	65.0 ms	-13.50	0.0000

Figure 6.5 illustrates the classification accuracy in edge and cloud deployment. Accuracy was slightly higher on the cloud.

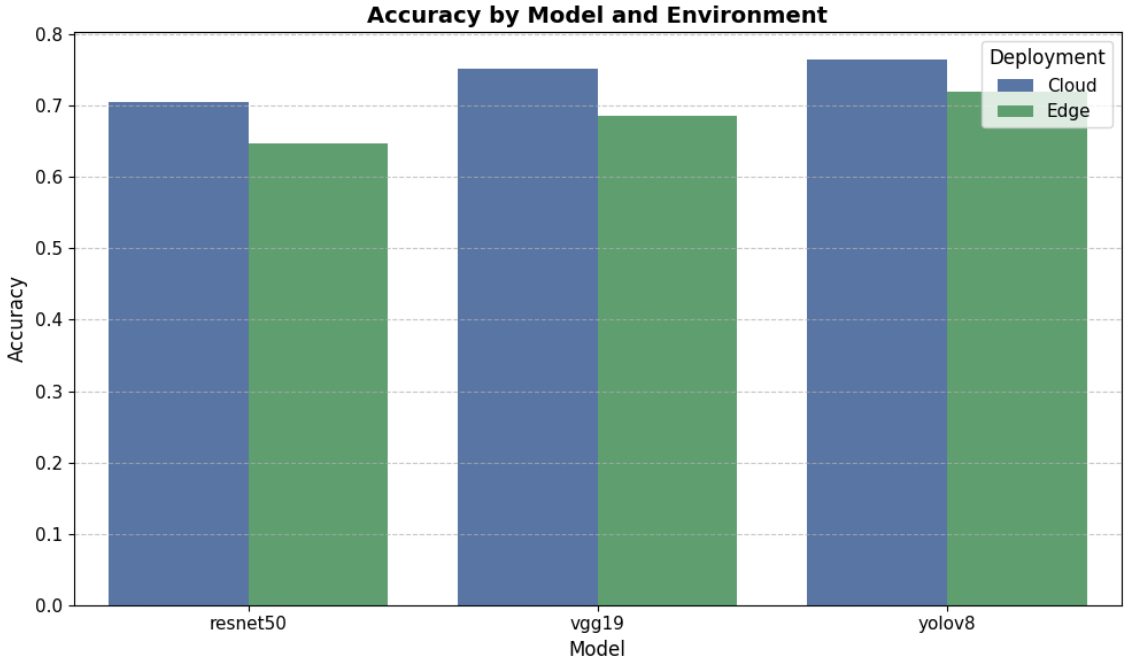


Figure 6.5: Visualization of the differences in accuracy by model and environment.

Paired t-tests, seen in Table 6.3, revealed that only VGG-19 showed a statistically significant difference at the $\alpha = 0.05$ level. For ResNet-50 and YOLOv8, the differences were not statistically significant.

Table 6.3: Paired t-test results for accuracy (Cloud vs Edge)

Model	Mean Accuracy (Cloud)	Mean Accuracy (Edge)	t-value	p-value
ResNet50	0.706	0.647	-1.6251	0.1062
VGG19	0.752	0.686	-2.0628	0.0408
YOLOv8	0.765	0.719	-1.8210	0.0706

Figure 6.6 illustrates the CPU usage in edge and cloud deployments. CPU usage was similar across both environments.

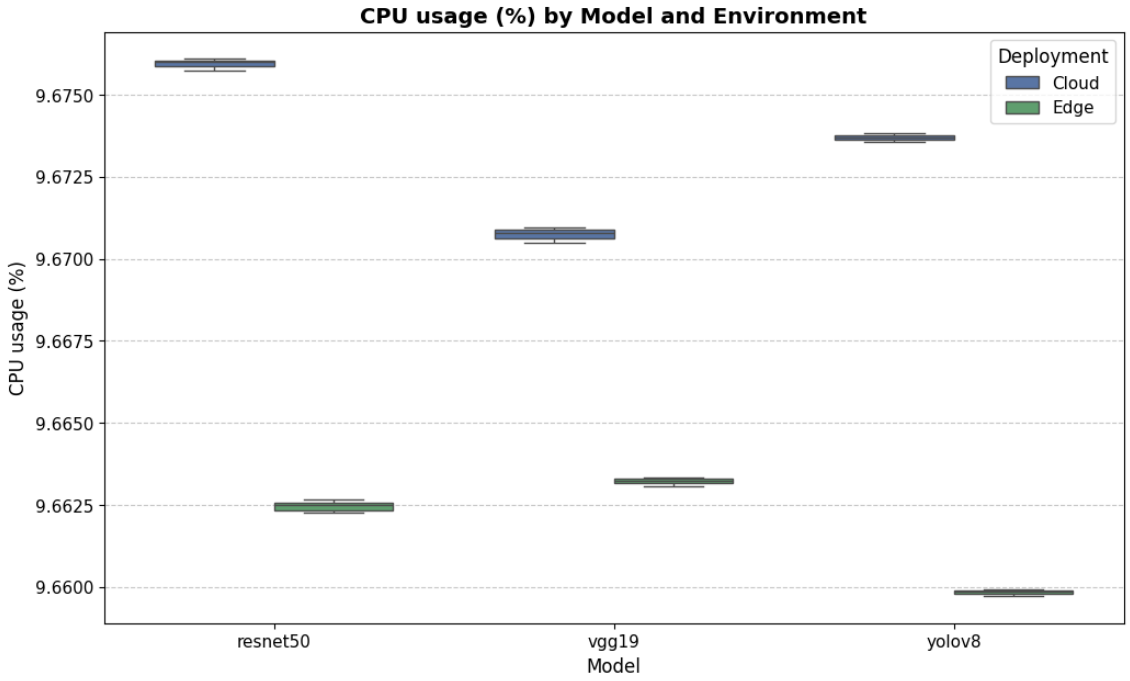


Figure 6.6: Visualization of the differences in CPU usage by model and environment.

Paired t-tests, seen in Table 6.4, revealed that CPU usage in cloud deployments was nearly constant across all models, making statistical comparison with edge deployments unreliable in some cases. Notably, YOLOv8 exhibited higher variance in CPU usage on edge devices, leading to a statistically significant difference. However, the extreme t-statistic observed for VGG-19 ($t = -1336.86$) reflects instability in the test due to minimal variance rather than a meaningful performance difference.

This is likely due to the model leveraging the neural processing unit (NPU) during inference on edge, offloading the computation away from the CPU. Due to lack of official Apple APIs for monitoring the NPU, we were unable to directly measure or confirm this.

Table 6.4: Paired t-test results for CPU usage (Cloud vs Edge)

Model	Mean CPU Usage (Cloud) (%)	Mean CPU Usage (Edge) (%)	t-value	p-value
ResNet-50	9.676	9.599	-1.2137	0.2267
VGG-19	9.671	9.663	-1336.8594	0.0000
YOLOv8	9.674	9.344	-2.3655	0.0193

Figure 6.7 illustrates the memory usage in edge and cloud deployments. Memory usage was substantially higher on edge for ResNet-50 and VGG-19, while YOLOv8 showed similar memory usage across both environments. As mentioned in subsection 4.3.4, edge deployment gathers memory data using `task_vm_info_` which measures the entire memory footprint of the app. Meanwhile, the cloud memory is measured from the docker stats command. Due to the difference in method of measurement

and sampling frequency, we did not perform t-tests, as the datasets are not directly comparable in structure or timing.

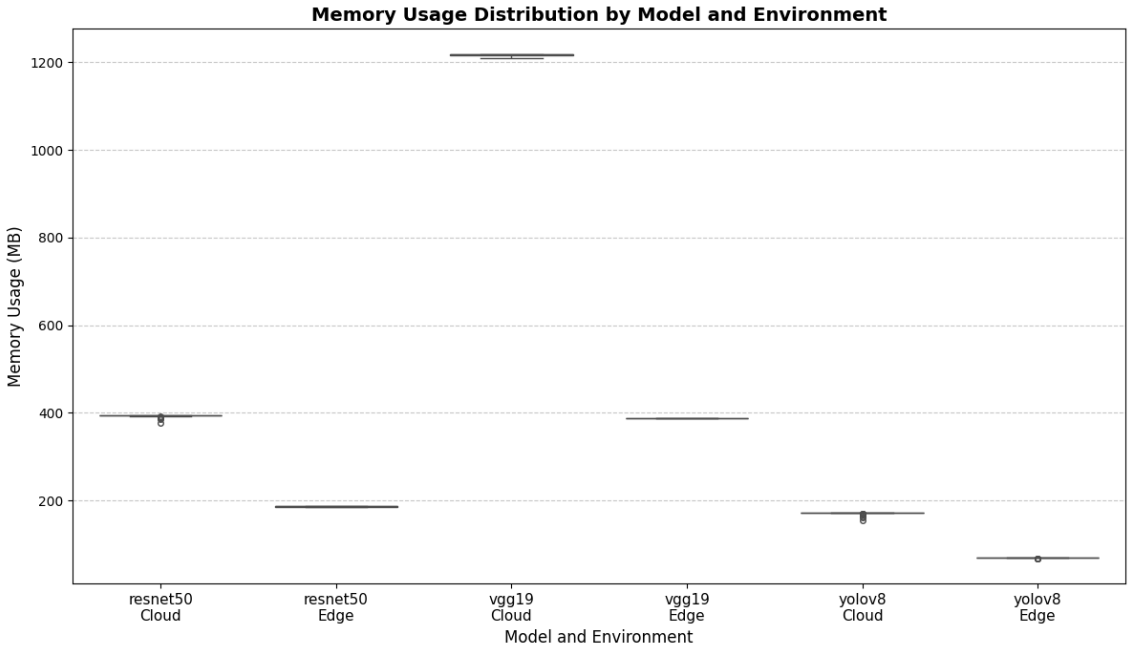


Figure 6.7: Visualization of the differences in memory usage by model and environment.

6.4 Stakeholder Validation

On the 28th of April, a meeting was held where the odontologist got to experience the application hands on. Before the meeting, the group had conducted trials in which 12 different individuals were screened using the YOLOv8 model running on edge. Due to time restrictions we were not able to validate all three models, therefore YOLOv8 was chosen due to having the highest accuracy on edge as seen in Table 6.3. The predictions were hidden from the odontologist, who performed her own examination. The results can be found in Table A.3 in Appendix A.4:

As seen in Table A.3 in Appendix A.4 and Figure 6.8, 5 out of the 12 predictions were the same as that of an odontologist. Furthermore, it can be noted that the model predicted "normal" in 7 out of 12 cases, compared to the odontologist which classified 3 out of 12 cases as "normal".

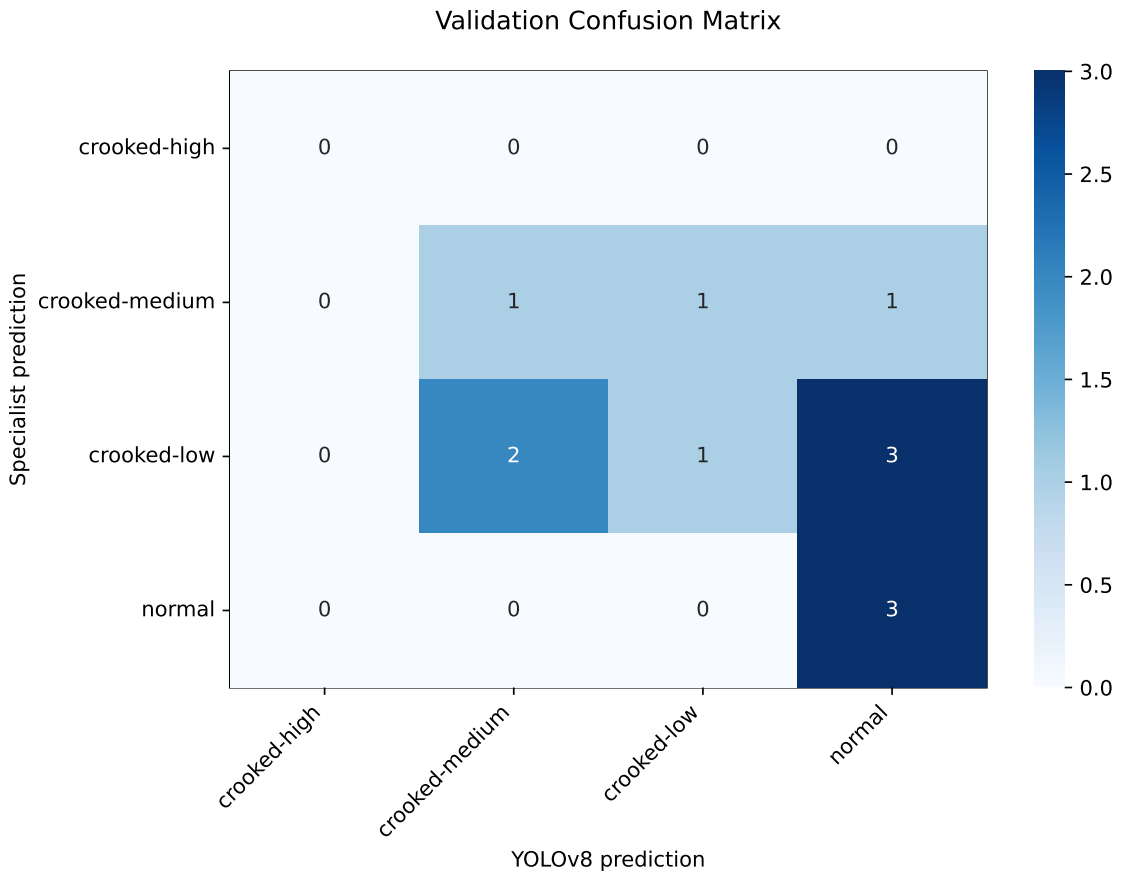


Figure 6.8: Confusion matrix illustrating the performance of YOLOv8 model running on edge in classifying teeth misalignment compared to a specialist's evaluation.

Table A.1 in Appendix A.1 outlines the responses provided by the specialist orthodontist, focusing on the usability and clinical implementation potential. In general, the stakeholder was satisfied with the application - specifically the usability. The orthodontist noted that although the application provided real-time feedback during image acquisition, its limited clarity may negatively affect the user experience.

7
Discussion

This section discusses the main findings of the project in relation to the central research question, along with a detailed analysis of model performance, clinical relevance, usability, and practical considerations.

7.1 Edge vs Cloud Deployment Trade-offs

A central part of this investigation involved comparing the deployment of AI models on the edge versus in the cloud, revealing several key trade-offs, each evaluated against the predefined thresholds and priorities outlined in Section 4.2.1.

7.1.1 Latency

Latency was consistently lower in edge deployment across the three models. For instance, ResNet-50's latency dropped from 307.3 ms in the cloud to 69.1 ms on edge and YOLOv8 being the clear winner with lowest latency for both cloud and edge deployments. Notably, all models and deployment methods, are within the 1.0 second threshold defined by Nielsen [53] for preserving the flow of the application, and the edge deployment for each model also met the stricter threshold of 0.1 seconds, that set the boundary for perceived instantaneity. Therefore, while edge offers a clear latency advantage, both deployment methods meet usability thresholds established for real-time applications and latency alone might not be the deciding factor when choosing between edge and cloud deployment.

7.1.2 Accuracy

Regarding accuracy, the results were marginally higher in the cloud deployments for all models. However, this difference was only found to be statistically significant for the VGG-19 model. This suggests that the potential accuracy degradation when converting and running models on the edge using the Core ML framework was minimal for ResNet-50 and YOLOv8 in this study. A more detailed analysis of model performance and metrics is provided in Section 7.2.

7.1.3 Resource efficiency

CPU usage was broadly similar between both environments and all three models. However, interpreting these results on edge devices is complicated by the likely involvement of the NPU. The consistency seen across both environments, especially the minimal variance in models like VGG-19, suggests that a significant portion of the inference computation may have been offloaded by the NPU. This introduced challenges in measurement, as Apple does not provide official APIs for tracking the NPU activity. As a result, CPU-based comparisons may under-represent the actual computational demands on the device during edge deployment of the model.

In terms of memory usage, the results compare model memory consumption in the backend to memory usage during edge inference. Importantly, cloud deployments do not contribute to memory load on the mobile device itself, as the models are stored remotely. However, backend inference tends to consume more memory overall. For example, VGG-19 in the cloud consumed about 1160.8 MB but only 386.7 MB on edge, an three-fold increase. ResNet-50 showed a similar tendency with an two-fold increase, from 185.35 MB on edge to 386.7 MB in the cloud. Likewise, YOLOv8 required 66.98 MB on edge versus 163.29 MB in the cloud, giving an increase of a 2.4× compared to edge in memory usage.

When compared to performance thresholds set by BrowserStack [58], which recommend staying below 250 MB for memory and 20% CPU usage, only the edge deployments for ResNet-50 and YOLOv8, met these criteria. VGG-19 on edge notably exceeded the memory threshold. Cloud deployments, however, do not contribute to memory usage on the mobile device and therefore would also reach the defined thresholds. Nonetheless, the increased resource demand from edge deployment could negatively impact battery life and performance, especially on lower-end devices.

The increased memory usage in cloud deployments compared to edge was likely due to two factors: the additional overhead introduced by the Docker runtime and the use of different floating point precisions. On edge, models used a 16-bit floating-point tensor, whereas cloud's ONNX runtime operated with a 32-bit tensor. A study [61] examining mixed-precision notes that, although deep neural networks have traditionally used 32-bit floating-point arithmetic, modern systems now offer support for 16-bit precision. Furthermore, the study goes on to say that switching from 32-bit to 16-bit can cut memory usage by around 50% while also delivering faster arithmetic operations.

7.1.4 Privacy Considerations

Finally, privacy and security considerations differ significantly between the two approaches. Edge computing provides inherent privacy advantages by allowing sensitive data, such as facial and dental images, to be processed locally without transmission to external servers. Cloud deployments necessitate robust security protocols

such as HTTPS, token-based authentication, and secure S3 storage with presigned URLs implemented in this project to safeguard patient data during transit and storage. The optimal choice requires balancing the enhanced privacy of edge processing against the potential scalability, centralized management, and marginally higher accuracy observed in cloud solutions.

7.2 Analysis of Classification and Segmentation Models

The performance of the classification models used in this project - YOLOv8, VGG-19 and ResNet-50 - alongside the segmentation model YOLOv11-N are evaluated in this section, with an emphasis on both technical performance and clinical relevance based on stakeholder feedback.

The YOLOv11-N instance segmentation model for teeth visibility validation demonstrated robust performance. Its final validation $mAP_{50:95}$ score reached approximately 0.86 for masks, indicating a strong capability to accurately delineate tooth boundaries across various overlap criteria. This suggests potential for future applications requiring detailed tooth identification.

As presented in section 6, the YOLOv8 model achieved the highest accuracy of 83.01%, outperforming both VGG-19 and ResNet-50, which both reached an accuracy of 73.86%. YOLOv8 also led across all key metrics. Its higher precision, in particular, indicates improved reliability in accurately determining positive cases, thereby reducing false negatives, which is an important factor in clinical screening scenarios where misdiagnosis can lead to unnecessary treatment. In contrast, the lower precision score from VGG-19 and ResNet-50 could increase the risk of misdiagnosis, potentially undermining the user trust in a healthcare setting.

However, it is important to interpret these results with caution. The limited dataset of merely 626 images constrains the model's capacity to generalize, regardless of architectural superiority. While YOLOv8 shows promise, its accuracy levels, though comparatively high, do not yet meet the standard expected for clinical use.

Also, the dataset suffers from significant class imbalance, with the "normal" class comprising over two-thirds of the data. This biases the models toward over-predicting the majority class, something clearly visible in the confusion matrices. While YOLOv8 demonstrates relatively better balance between predictions, it still frequently misclassified "crooked-high" cases as "crooked-medium" and "crooked-low" cases as "normal". In contrast, VGG-19 and ResNet-50 show a much stronger tendency to default to the "normal" class across all categories, which is a critical concern in a clinical screening context.

Originally, the group anticipated access to roughly 5,000 images taken by the odon-

tologist. However, since these images had not been anonymized, their use required ethics approval, which unfortunately had expired. Access to a larger, more representative dataset could greatly improve the generalizability of the AI models, which in-turn would reduce the risk of overfitting to narrow patterns and features present in the small dataset. Therefore, while performance at least for YOLOv8 appears encouraging, further work is needed to improve class balance and mitigate bias before real-world deployment is considered.

7.3 Clinical Validation and Practical Impact

To better understand the clinical relevance and usability of the system, stakeholder validation was conducted with a domain expert. When comparing the model's predictions to expert assessment, only 5 of 12 classifications matched. Notably, the model predicted "normal" in 7 out of 12 cases, while the expert only classified 3 of them as "normal". As seen in Table 4.1, 68.5% of the dataset has the label "normal". The confusion matrices in Figures 6.1a, 6.1b, and 6.1c further highlight that the three models favor the "normal" classification. This discrepancy in classifications further highlights the system's current limitations in diagnostic accuracy and further emphasizes the need for larger, more diverse and balanced training data to align the model prediction with real clinical judgment.

Despite these mismatches, the qualitative feedback from the stakeholder was generally positive. The orthodontist found the app intuitive, trustworthy and with clearly presented results. However, they noted the lack of real-time feedback during image capture, which could negatively affect user experience. Regarding clinical integration, the orthodontist noted that, although the tool may not be a significant aid to specialists, it holds promise for general dentists. The app could serve as an accessible communication aid for explaining misalignment severity to the patient or assisting in deciding whether a referral to a specialist is necessary.

In contrast to Dental Monitoring, which requires the DM ScanBox hardware - our AI diagnostic system operates entirely through a mobile application, eliminating any external device required for diagnostics. This app-only approach simplifies the user experience and broadens accessibility. Additionally, the system features an automated image-capturing process that guides the users to position their cameras correctly and consistently, ensuring that every photograph meets the quality standards necessary for reliable evaluation. By integrating this built-in capture mechanism rather than relying on additional hardware, the system's AI models consistently receive good quality photos, thereby improving diagnostic accuracy.

8

Conclusion

The aim of this this thesis was to explore the deployment of AI models for orthodontic assessment in both edge and cloud environments, and to compare their trade-offs in latency, accuracy, and resource efficiency. Edge deployment consistently delivered lower latency and better privacy, aligning well with real-time application requirements. However, cloud deployments offer slightly higher accuracy and scalability, particularly benefiting from access to more computational resources, without impacting the resource demand of the device.

Among the models evaluated, YOLOv8 was the most promising for all metrics, particularly in the classification task, achieving the highest accuracy, precision, and demonstrating greater resilience to the class imbalance. However, all models were constrained by the limited and unbalanced dataset, highlighting a significant barrier to clinical reliability. The segmentation model YOLO11n-seg demonstrated strong performance, indicating its potential for applications where tooth identification is necessary.

Stakeholder feedback highlighted the app's usability and practical relevance, particularly for general dentistry, though concerns around diagnostic accuracy and the models tendency to overclassify cases as "normal" were noted. The app's independence from external devices improves accessibility and simplifies the user experience, setting it apart from other alternatives on the market today.

Ultimately, while the system is an encouraging step toward an AI-assisted orthodontic diagnostic tool, its potential for clinical use remains limited by dataset size, accuracy requirements, and platform scope. Future work should focus on expanding and diversifying the dataset to improve generalization and reduce bias. The technical implementation was limited to the iOS platform, and while this decision facilitated development within the given time frame, it hinders the deployment scope. To support a broader adoption, future studies should also involve performance evaluations across multiple platforms and hardware configurations, and aim to improve model performance while preserving the practical benefits observed in edge deployment, such as lower latency and better privacy.

Bibliography

[1] K. Kakhi, R. Alizadehsani, H. D. Kabir, A. Khosravi, S. Nahavandi, and U. R. Acharya, "The internet of medical things and artificial intelligence: Trends, challenges, and opportunities," *Biocybernetics and Biomedical Engineering*, vol. 42, no. 3, pp. 749–771, Sep. 2022. DOI: 10.1016/j.bbe.2022.05.008.

[2] C. Huang, J. Wang, S. Wang, and Y. Zhang, "Internet of medical things: A systematic review," *Neurocomputing*, vol. 557, Jan. 2023. DOI: 10.1016/j.neucom.2023.126719.

[3] S. J. Davies, R. M. J. Gray, P. J. Sandler, and K. D. O'Brien, "Orthodontics and occlusion," *British Dental Journal*, vol. 191, no. 10, pp. 539–549, Nov. 2001. DOI: 10.1038/sj.bdj.4801229.

[4] G. Lombardo, F. Vena, P. Negri, *et al.*, "Worldwide prevalence of malocclusion in the different stages of dentition: A systematic review and meta-analysis," *European Journal of Paediatric Dentistry*, vol. 21, no. 2, pp. 115–122, Jun. 2020. DOI: 10.23804/ejpd.2020.21.02.05.

[5] P. Niemi, M. Kortelainen, U. Harjunmaa, and J. Waltimo-Sirén, "Costs and duration of orthodontic-surgical treatment with mandibular advancement surgery," *European Journal of Orthodontics*, vol. 45, no. 5, pp. 558–564, Sep. 2023. DOI: 10.1093/ejo/cjad051.

[6] A. Saghar, H. Dino, M. Mattias, S. Jalal, O. Zoé, and T. Alice, "Ai-based diagnosis of malaligned teeth," Bachelor's thesis, Chalmers University of Technology, 2024, pp. 32–33.

[7] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014, pp. 1–11, 27–34. DOI: 10.1007/978-3-662-43839-8.

[8] L. Pinto-Coelho, "How artificial intelligence is shaping medical imaging technology: A survey of innovations and applications," *Bioengineering*, vol. 10, no. 12, 2023, ISSN: 2306-5354. DOI: 10.3390/bioengineering10121435. [Online]. Available: https://www.mdpi.com/2306-5354/10/12/1435.

[9] J.-W. Lian, "Establishing a cloud computing success model for hospitals in taiwan," *INQUIRY*, vol. 54, p. 0 046 958 016 685 836, 2017. DOI: 10.1177/0046958016685836.

[10] M. Putzier, T. Khakzad, M. Dreischarf, S. Thun, F. Trautwein, and N. Taheri, "Implementation of cloud computing in the german healthcare system," *npj Digital Medicine*, vol. 7, no. 1, p. 12, Jan. 2024. DOI: 10.1038/s41746-024-01000-3. [Online]. Available: https://doi.org/10.1038/s41746-024-01000-3.

[11] M. Barakat, R. A. Saeed, and S. Edam, "A comparative study on cloud and edgeb computing: A survey on current research activities and applications," in *2023 IEEE 3rd International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA)*, 2023, pp. 679–684. DOI: 10.1109/MI-STA57575.2023.10169821.

[12] F. C. Andriulo, M. Fiore, M. Mongiello, E. Traversa, and V. Zizzo, "Edge computing and cloud computing for internet of things: A review," *Informatics*, vol. 11, no. 4, 2024, ISSN: 2227-9709. DOI: 10.3390/informatics11040071. [Online]. Available: https://www.mdpi.com/2227-9709/11/4/71.

[13] S. Mallishery, P. Chhatpar, K. S. Banga, T. Shah, and P. Gupta, "The precision of case difficulty and referral decisions: An innovative automated approach," *Clinical Oral Investigations*, vol. 24, no. 6, pp. 1909–1915, Jun. 2020, ISSN: 1436-3771. DOI: 10.1007/s00784-019-03050-4.

[14] G. Rubiu, M. Bologna, M. Cellina, *et al.*, "Teeth segmentation in panoramic dental x-ray using mask regional convolutional neural network," *Applied Sciences*, vol. 13, no. 13, 2023, Cited by: 18; All Open Access, Gold Open Access, Green Open Access. DOI: 10.3390/app13137947. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85164964165&doi=10.3390%2fapp13137947&partnerID=40&md5=f2ab7702041b47edface24f8189654ed.

[15] N. Adnan, S. M. Faizan Ahmed, J. K. Das, *et al.*, "Developing an ai-based application for caries index detection on intraoral photographs," *Scientific Reports*, vol. 14, no. 1, p. 26 752, Nov. 2024, ISSN: 2045-2322. DOI: 10.1038/s41598-024-78184-x. [Online]. Available: https://doi.org/10.1038/s41598-024-78184-x.

[16] Y. Tian, Q. Ye, and D. Doermann, *Yolov12: Attention-centric real-time object detectors*, 2025. DOI: 10.48550/arXiv.2502.12524. arXiv: 2502.12524 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2502.12524.

[17] J. Ryu, Y.-S. Lee, S.-P. Mo, K. Lim, S.-K. Jung, and T.-W. Kim, "Application of deep learning artificial intelligence technique to the classification of clinical orthodontic photos," *BMC Oral Health*, vol. 22, no. 1, p. 454, Oct. 2022. DOI: 10.1186/s12903-022-02466-x. [Online]. Available: https://doi.org/10.1186/s12903-022-02466-x.

[18] A. I. Shahrul, N. Shukor, and N. H. Norman, "Technique for orthodontic clinical photographs using a smartphone," *International Journal of Dentistry*, vol. 2022, 2022, Cited by: 3; All Open Access, Gold Open Access, Green Open Access. DOI: 10.1155/2022/2811684. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85124041308&doi=10.1155%2f2022%2f2811684&partnerID=40&md5=df057cde81c175b426d6aab929e394da.

[19] K. Homsi, V. Snider, B. Kusnoto, *et al.*, "In-vivo evaluation of artificial intelligence driven remote monitoring technology for tracking tooth movement and reconstruction of 3-dimensional digital models during orthodontic treatment," *American Journal of Orthodontics & Dentofacial Orthopedics*, vol. 164, no. 5, Nov. 2023. DOI: 10.1016/j.ajodo.2023.04.019.

[20] M. P. Craven, K. Selvarajah, R. Miles, *et al.*, "User requirements for the development of smartphone self-reporting applications in healthcare," in *Human-Computer Interaction. Applications and Services*, M. Kurosu, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 36–45. DOI: 10.1007/978-3-642-39262-7_5.

[21] M. Fazio, C. Lombardo, G. Marino, *et al.*, "Linguapp: An m-health application for teledentistry diagnostics," *International Journal of Environmental Research and Public Health*, vol. 19, no. 2, 2022. DOI: 10.3390/ijerph19020822. [Online]. Available: https://www.mdpi.com/1660-4601/19/2/822.

[22] A. Zhukovska, T. Zheliuk, D. Shushpanov, V. Brych, O. Brechko, and N. Kryvokulska, "Management of the development of artificial intelligence in healthcare," in *2023 13th International Conference on Advanced Computer Information Technologies (ACIT)*, 2023, pp. 241–247. DOI: 10.1109/ACIT58437.2023.10275435.

[23] N. Nordblom, M. Büttner, and F. Schwendicke, "Artificial intelligence in orthodontics: Critical review," *Journal of Dental Research*, vol. 103, no. 6, pp. 577–584, 2024, Cited by: 9; All Open Access, Green Open Access. DOI: 10.1177/00220345241235606. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85191695526&doi=10.1177%2f00220345241235606&partnerID=40&md5=9f450a7af9733fcab42fb1b320c28095.

[24] A. Tan, F. Bennani, W. M. Thomson, M. Farella, and L. Mei, "A qualitative study of orthodontic screening and referral practices among dental therapists in new zealand," *Australian orthodontic journal*, vol. 32, no. 2, pp. 155–164, 2016, Cited by: 3. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85045222511&partnerID=40&md5=eba42f1ac5120a9c07ae9e773cf053db.

[25] A. T. Macari, *Orthodontic disorders and diagnosis*. 2021, pp. 73–91, Cited by: 0. DOI: 10.1007/978-3-030-69109-7_5. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85151732524&doi=10.1007%2f978-3-030-69109-7_5&partnerID=40&md5=02a22c100dfac47dee280da313800ee3.

[26] M. Postnikov, A. Seryogin, D. Andriyanov, and P. Voroshnina, "Algorithm for comprehensive diagnosis and treatment of patients with gnathic form of malocclusion class ii (clinical case)," *Clinical Dentistry (Russia)*, no. 1, pp. 114–122, 2021, Cited by: 0; All Open Access, Gold Open Access. DOI: 10.37988/1811-153X_2021_1_114. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85150060628&doi=10.37988%2f1811-153X_2021_1_114&partnerID=40&md5=ad3b0e5a334805f8e09ad10de8311eaa.

[27] E. Bardideh, F. L. Alizadeh, M. Amiri, and M. Ghorbani, "Designing an artificial intelligence system for dental occlusion classification using intraoral photographs: A comparative analysis between artificial intelligence-based and clinical diagnoses," *Original Article*, vol. 166, no. 2, pp. 125–137, Aug. 2024.

[28] S. Cong and Y. Zhou, "A review of convolutional neural network architectures and their optimizations," *Artificial Intelligence Review*, vol. 56, pp. 1905–1969, 2023, Published: 22 June 2022, Issue Date: March 2023. DOI: 10.1007/

s10462-022-10213-5. [Online]. Available: https://doi.org/10.1007/s10462-022-10213-5.

[29] W. He, T. Zhou, Y. Xiang, Y. Lin, J. Hu, and R. Bao, "Deep learning in image classification: Evaluating vgg19's performance on complex visual data," 2024. arXiv: 2412.20345 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2412.20345.

[30] G. Ravi, M. Prasanth, and S. Rajendran, "Eye disease classification using vgg-19 architecture," *Signals and Communication Technology*, vol. Part F2556, pp. 545–552, 2024. DOI: 10.1007/978-3-031-47942-7_46. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85191444562&doi=10.1007%2f978-3-031-47942-7_46&partnerID=40&md5=f7bc4fb3a6f27e8ee0efed58ebf8b4b4.

[31] F. Mohammad and S. Al Ahmadi, "Alzheimers disease prediction using deep feature extraction and optimization," *Mathematics*, vol. 11, no. 17, 2023, Cited by: 6; All Open Access, Gold Open Access. DOI: 10.3390/math11173712. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85176382110&doi=10.3390%2fmath11173712&partnerID=40&md5=aeaebb9a3412f6794458bd33bf40b8bf.

[32] M. Shaha and M. Pawar, *Transfer learning for image classification*, Conference paper, Cited by: 292, 2018. DOI: 10.1109/ICECA.2018.8474802. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85060869694&doi=10.1109%2fICECA.2018.8474802&partnerID=40&md5=5d9fb3fe151c3f2b21b3915946f6b930.

[33] V. Rezende, M. Costa, A. Santos, and R. C. De Oliveira, *Image processing with convolutional neural networks for classification of plant diseases*, Conference paper, Cited by: 10, 2019. DOI: 10.1109/BRACIS.2019.00128. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85077034805&doi=10.1109%2fBRACIS.2019.00128&partnerID=40&md5=89f00a514af99f4c978c798aa2562359.

[34] D. G. Kumar, G. Harish, M. V. Subbarao, G. C. Ram, D. R. Varma, and E. P. Kumar, "A compact cnn architecture for detection and classification of cotton leaf diseases," Cited by: 0, 2024. DOI: 10.1109/ICIICS63763.2024.10859957. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85218184302&doi=10.1109%2fICIICS63763.2024.10859957&partnerID=40&md5=1b0b0028052f3842efe97878793548e4.

[35] G. Guruarunachalam, G. Prabhath Sai, and J. J. Hilda, "Automated kidney stone detection using deep learning technique," Cited by: 2, 2024. DOI: 10.1109/ic-ETITE58242.2024.10493536. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85192531062&doi=10.1109%2fic-ETITE58242.2024.10493536&partnerID=40&md5=17363e7a0d59aaea18df40cead68bf83.

[36] J. Miao, S. Xu, B. Zou, and Y. Qiao, "Resnet based on feature-inspired gating strategy," *Multimedia Tools and Applications*, vol. 81, no. 14, pp. 19 283–19 300, 2022, Cited by: 8. DOI: 10.1007/s11042-021-10802-6. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-

85103187432&doi=10.1007%2fs11042-021-10802-6&partnerID=40&md5=1d96fd5f709ff97d6fa35f3e5d820874.

[37] M. G. Boehme and F. Al-Turjman, "Enhancing object detection capabilities: A comprehensive exploration and fine-tuning of yolov5 algorithm across diverse datasets," *Advances in Science, Technology and Innovation*, pp. 99–104, 2024, Cited by: 0. DOI: 10.1007/978-3-031-63103-0_9. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85201928708&doi=10.1007%2f978-3-031-63103-0_9&partnerID=40&md5=8a4a25b2696f44ae7a88ccf2a694ce8f.

[38] M. Hussain, "Yolov5, yolov8 and yolov10: The go-to detectors for real-time vision," *Preprint*, Jul. 2024.

[39] Ultralytics, *Yolov8 vs yolo11: A technical comparison*, Accessed: May 19, 2025, n.d. [Online]. Available: %5BOnline%5D.%20Available:%20%5Curl%7Bhttps://docs.ultralytics.com/compare/yolov8-vs-yolo11/#ultralytics-yolov8%7D.

[40] R. Khanam and M. Hussain, *Yolov11: An overview of the key architectural enhancements*, 2024. arXiv: 2410.17725 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2410.17725.

[41] T. A. Korzhebin and A. D. Egorov, "Comparison of combinations of data augmentation methods and transfer learning strategies in image classification used in convolution deep neural networks," in *2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*, 2021, pp. 479–482. DOI: 10.1109/ElConRus51938.2021.9396724.

[42] O. Russakovsky, J. Deng, H. Su, *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[43] TensorFlow, [Online]. Available: https://www.tensorflow.org/tutorials/images/transfer_learning, [Accessed: May 18, 2025], Aug. 2024.

[44] National Institute of Standards and Technology (NIST), "The nist definition of cloud computing," U.S. Department of Commerce, Special Publication 800-145, 2011. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf.

[45] J. Varia and S. Mathew, "Overview of amazon web services," in *Cloud Computing: Principles and Paradigms*, R. Buyya, J. Broberg, and A. Goscinski, Eds., Wiley, 2016, pp. 85–110. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/aws-overview.pdf.

[46] S. Xu, S. M. Ghazimirsaeed, J. M. Hashmi, H. Subramoni, and D. K. Panda, "Mpi meets cloud: Case study with amazon ec2 and microsoft azure," *BDJ Open*, vol. 10, no. 1, p. 58, Jul. 2024. DOI: 10.1038/s41405-024-00235-2.

[47] Amazon Web Services Inc, *"amazon s3"*, aws.amazon.com, Accessed: Mar. 21, 2025. [Online]. Available: https://aws.amazon.com/s3/.

[48] Microsoft Azure, *"what is edge computing?"*, azure.microsoft.com, Accessed: Mar. 17, 2025. [Online]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-edge-computing/.

[49] G. Flamis, S. Kalapothas, and P. Kitsos, "Best practices for the deployment of edge inference: The conclusions to start designing," *Electronics*

(Switzerland), vol. 10, no. 16, 2021, Cited by: 6; All Open Access, Gold Open Access. DOI: 10.3390/electronics10161912. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85112599933&doi=10.3390%2felectronics10161912&partnerID=40&md5=fd46c2bca67210b3ff4983fe1c159cc7.

[50] D. Danopoulos, C. Kachris, and D. Soudris, "Utilizing cloud fpgas towards the open neural network standard," *Sustainable Computing: Informatics and Systems*, vol. 30, 2021, Cited by: 17. DOI: 10.1016/j.suscom.2021.100520. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85100488237&doi=10.1016%2fj.suscom.2021.100520&partnerID=40&md5=289edfba61c51753a53a795775939d81.

[51] O. Marques, "Computer vision and image analysis with the vision framework," *SpringerBriefs in Computer Science*, pp. 41–50, 2020, Cited by: 0. DOI: 10.1007/978-3-030-54032-6_5. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85096766206&doi=10.1007%2f978-3-030-54032-6_5&partnerID=40&md5=70a079cf1498c9abfe601c54a7423829.

[52] V. K. Rathi, N. K. Rajput, S. Mishra, *et al.*, "An edge ai-enabled iot healthcare monitoring system for smart cities," *Computers and Electrical Engineering*, vol. 96, 2021, Cited by: 60. DOI: 10.1016/j.compeleceng.2021.107524. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85116919952&doi=10.1016%2fj.compeleceng.2021.107524&partnerID=40&md5=c594cc09c8ed65cc74bb1542e7984de6.

[53] J. Nielsen, *Response times: The 3 important limits*, [Online]. Available: https://www.nngroup.com/articles/response-times-3-important-limits/, Accessed: May 8, 2025, 1993.

[54] S. M. Nour, "Artificial intelligence (ai) for improving performance at the cutting edge of medical imaging," Cited by: 6, 2023, pp. 426–429. DOI: 10.1109/NILES59815.2023.10296694. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85178502547&doi=10.1109%2fNILES59815.2023.10296694&partnerID=40&md5=ce7e215eb02780d8d9ff4149ea924044.

[55] A. R. Nandhakumar, A. Baranwal, P. Choudhary, M. Golec, and S. S. Gill, "Edgeaisim: A toolkit for simulation and modelling of ai models in edge computing environments," *Measurement: Sensors*, vol. 31, 2024, Cited by: 23; All Open Access, Gold Open Access, Green Open Access. DOI: 10.1016/j.measen.2023.100939. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85178244067&doi=10.1016%2fj.measen.2023.100939&partnerID=40&md5=da99affe8cba629dc08bac85a5000499.

[56] A. Mukherjee, D. De, and R. Buyya, "Cloud computing resource management," *Studies in Big Data*, vol. 151, pp. 17–37, 2024, Cited by: 1. DOI: 10.1007/978-981-97-2644-8_2. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85198325304&doi=10.1007%2f978-981-97-2644-8_2&partnerID=40&md5=5d7f67c711d25d8055cc1278bb77a522.

[57] W. Dargie, in *Identification of resource utilisation patterns in data centers using tensor decomposition*, Cited by: 2, vol. 2019-July, 2019. DOI: 10.1109/ICCCN.2019.8846965. [Online]. Available: https://www.scopus.com/

inward/record.uri?eid=2-s2.0-85073160246&doi=10.1109%2fICCCN.2019.8846965&partnerID=40&md5=8ced376d78977a13897a6302e8fd04a0.

[58] BrowserStack, *App performance metrics for ios*, Accessed: May 9, 2025, n.d. [Online]. Available: %5BOnline%5D.%20Available:%20%5Curl%7Bhttps://www.browserstack.com/docs/app-performance/app-performance-guides/ios%7D.

[59] P. Valluri, *Dentalai computer vision project*, 2023. [Online]. Available: https://www.kaggle.com/datasets/pawanvalluri/dental-segmentation.

[60] Internet Engineering Task Force (IETF), *JSON Web Token (JWT)*, RFC 7519, 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519.

[61] J. Hayford, J. Goldman-Wetzler, E. Wang, and L. Lu, *Speeding up and reducing memory usage for scientific machine learning via mixed precision*, 2024. arXiv: 2401.16645 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2401.16645.

A

Appendix 1

A.1 Stakeholder Interview Transcript

Table A.1: Stakeholder interview responses from a specialist orthodontist, evaluating the usability, clarity, and potential clinical integration of the application.

Question	Answer
Navigation and Core Functionality: How intuitive did you find the process of starting the app? Was it easy to understand the instructions, take a photo, and initiate the AI analysis to classify the dental misalignment? Were any steps unclear or hard to find?	No, it was very clear. However, when the camera wasn't close enough, I didn't receive feedback indicating that it was too far away. I was just standing there waiting.
Results Presentation and Clarity: How clearly and understandably was the result of the AI classification presented? Did you immediately understand the type of misalignment identified by the AI and its severity, if applicable? Was there any information you felt was missing in the results view?	I might be biased, but it was very clear and educational when it said "normal." If I were a patient, it could say something like, "Your teeth are well aligned; you don't need treatment." Someone not familiar with dental terminology might wonder, "Are my teeth okay?" There's potential to improve this. I liked the graphics.
Integration in Clinical Workflow: Based on your experience, how well do you think this app would fit into your current workflow for assessing or documenting malocclusion? Would it save time, require extra steps, or replace an existing tool/method?	In my clinical work as an orthodontist, not much for me personally, but it could help patients understand the severity of their misalignment. For general dentists, it would be an easier way to communicate with patients, providing an assessment of the malocclusion severity. Follow-up Explanation: As a specialist, this takes me 2 seconds, but for general dentists who refer patients to me, it can be difficult for them to distinguish between severe and mild cases. This app would help them assess cases on their own and improve communication with patients. I'm specially trained in orthodontics, and the app could help facilitate communication with patients or direct them to the appropriate specialist.
Overall Experience and Confidence: What was your overall impression of using the app? Was there any specific moment that felt particularly smooth or, conversely, frustrating? How does the interface and interaction affect your initial confidence in the AI-generated classification?	Confidence in the app was very high. It felt easy, pleasant, and straightforward. The steps were clear and concise, making it feel trustworthy. Frustrating moment: Taking the photo due to the instructions. Everything else worked perfectly.

A.2 Overview of API Endpoints

Table A.2: Overview of API Endpoints

Method	Path	Summary
PUT	/ai/change-model/{model}	Change model (admin only)
POST	/ai/predict	Run AI inference on an image (protected)
POST	/auth/login	Log in a user
POST	/auth/logout	Log out a user (protected)
GET	/evaluations	Get evaluations for user (protected)
POST	/evaluations	Add a new evaluation (protected)
DELETE	/evaluations/{id}	Delete evaluation (protected)
GET	/health	Health check endpoint
POST	/token/renew	Renew access token (protected)
POST	/token/revoke/{id}	Revoke a session (admin only)
GET	/users/all	Get all users (admin only)
POST	/users/create	Create a new user

A.3 CPU Monitoring Code

```swift
private func getCurrentCPUUsage() -> Double {
    // Variable declarations omitted for brevity

    let result = host_processor_info(
        mach_host_self(),
        PROCESSOR_CPU_LOAD_INFO,
        &processorCount,
        &cpuInfo,
        &cpuInfoCount
    )

    if result == KERN_SUCCESS, let cpuInfo = cpuInfo {
        let cpuLoadInfos = UnsafeBufferPointer(start: unsafeBitCast(cpuInfo,
        ↪  to: UnsafePointer<processor_cpu_load_info>.self), count:
        ↪  Int(processorCount))

        for i in 0..<Int(processorCount) {

            // CPU tick extraction and type casting omitted for brevity

            let totalTicks = userTicks + systemTicks + idleTicks + niceTicks
            if totalTicks > 0 {
                let coreCpuUsage = Double(userTicks + systemTicks) /
                ↪  Double(totalTicks) * 100.0
                totalCpuUsage += coreCpuUsage
            }
        }

        // Average CPU usage across all cores
        if processorCount > 0 {
            totalCpuUsage /= Double(processorCount)
        }

        // Memory cleanup omitted for brevity
    }
    return totalCpuUsage
}
```

Figure A.1: Function for retrieving and processing CPU usage across all available cores using kernel-level APIs.

A.4 Stakeholder Validation Screening

Table A.3: Screening of 12 individuals performed by an odontologist compared to the YOLOv8 model running on edge.

YOLOv8 prediction	**Odontologist**
Normal	Medium
Normal	Normal
Normal	Low
Low	Medium
Medium	Low
Normal	Low
Normal	Normal
Normal	Low
Medium	Medium
Low	Low
Medium	Low
Normal	Normal

A.5 Use of AI Assistance

For programming components of our thesis GitHub Copilot was utilized to help check for vulnerabilities and errors. To help with finding spelling mistakes, and other inconsistencies, Grammarly was used throughout the writing process. Scopus was also used to research and find relevant articles, all of which were thoroughly inspected and read, and evaluated for trustworthiness using Ulrichsweb. The use of AI was done with great caution, mainly for correctional purposes, and all output from AI models was carefully and thoughtfully inspected and adjusted based on our judgment.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY