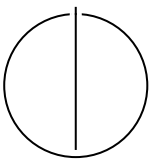
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

A Comparative Analysis of a CNN Model Deployment on Amazon Web Services

Hongyu Guo



SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

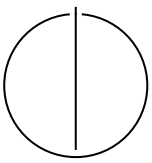
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

A Comparative Analysis of a CNN Model Deployment on Amazon Web Services

Eine Vergleichende Analyse der Bereitstellung eines CNN-Modells auf Amazon Web Services

Author:	Hongyu Guo
Examiner:	Prof. Dr.-Ing. Jörg Ott
Supervisor:	Giovanni Bartolomeo, Navidreza Asadi, Patrick Laufer
Submission Date:	15.06.2025



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15.06.2025 Hongyu Guo

Acknowledgments

Firstly, I am deeply thankful to my three supervisors for their invaluable guidance and support throughout my master's thesis. They gave me the freedom to work at my own pace while offering meaningful and constructive advice whenever I encountered challenges.

I am also deeply grateful to my family—my parents, my sister, and my boyfriend—for their unwavering support during my studies in Germany. This has been an incredible journey filled with challenges and growth, and I could not have done it without their support and companionship.

Finally, I would like to thank my university, TUM, and all the professors and lecturers who have taught me.

This thesis marks both a meaningful conclusion and a hopeful beginning. I am truly grateful for everything I have learned and for everyone who supported me along the way.

Abstract

Many organizations are migrating their on-premise Artificial Intelligence (AI) applications to the cloud due to several advantages such as the availability of large infrastructure with high scalability, low operational management cost and a wide range of software and platform services. Cloud vendors, like Amazon Web Services (AWS), provide various tools, services, and deployment frameworks to facilitate this migration. However, identifying the most appropriate service and deployment strategy to achieve optimal performance with minimal cost remains a challenge.

This work presents the methodology and strategies for deploying a pre-defined Convolutional Neural Network (CNN)-based human age and gender prediction system on the AWS cloud platform. We design and implement deployment strategies across cloud nodes, edge nodes, and hybrid edge-cloud architectures, utilizing both serverless and server-based computing models. Furthermore, we explore the feasibility to integrate the advanced techniques such as Split Computing (SC) and Early Exiting (EE) within hybrid architectures.

We conduct a comprehensive evaluation of five deployment strategies based on latency, accuracy, and cost. Deploying on edge nodes with the help of Wavelength Zone (WZ) achieves the lowest latency but incurs about 1.3 times the cost of equivalent Elastic Compute Cloud (EC2) instances in the cloud. Within cloud deployments, AWS Lambda offers lower compute latency but higher communication latency than EC2, resulting in comparable overall performance. Lambda is more cost-effective for workloads under around 150,000 monthly requests, while EC2 becomes more economical for higher traffic volumes. Although the hybrid edge-cloud setup does not provide significant performance improvements in our experiments, it demonstrates the feasibility with split computing and EE strategies. For large-scale scenarios, it may significantly reduce the latency by combining edge proximity with cloud high-performance resources, while at around 2.3 times the cost of cloud-only deployment. Accuracy remains consistent across deployment strategies unless the model architecture is modified, such as with the integration of EE.

The architecture design and experimental findings in this study offer valuable guidance for selecting appropriate deployment strategies when deploying AI applications on the AWS cloud.

Contents

1 Introduction

This chapter introduces the problem statement and motivation, research contribution, and overall structure of the thesis.

1.1 Problem Statement

In recent years, AI has achieved remarkable progress in a wide range of fields, including Natural Language Processing (NLP), image classification, and speech recognition [1, 2, 3]. Despite the AI applications can achieve high performance, these systems typically require substantial computational and storage resources. However, such resource requirements bring significant challenges for deployment on lightweight devices, such as Internet of Things (IoT) devices, which lack sufficient capacity [4]. What's more, some enterprises often find it difficult to equip the necessary high-performance infrastructure, as it requires much upfront investment [5]. This investment not only increases financial risk due to uncertain Return on Investment (ROI), but also introduces complexity in system maintenance, failover handling, and scalability.

To address these challenges, cloud computing has emerged as a important solution. Cloud providers offer a wide range of services involving computation, storage, networking, security and so on. Furthermore, the pay-as-you-go pricing model allows users to avoid heavy initial investment and scale as needed. However, this flexibility introduces new challenges. Given the large number of services provided by cloud providers such as AWS, with varying pricing models for different services and configurations, a key challenge arises: how can users effectively choose the right services and deployment architecture for their AI applications?

While it is infeasible to evaluate every possible cloud service, this work aims to systematically evaluate and compare representative deployment strategies from two critical dimensions, using AWS as an example: (1) compute models—comparing server-based (e.g., EC2) and serverless (e.g., Lambda) models; and (2) architectural topology—evaluating centralized cloud deployments, edge deployments, and hybrid edge-cloud solutions. This study focuses on identifying trade-offs in performance, like latency and accuracy, and cost under these deployment scenarios.

1.2 Contribution

The main contribution of this work lies in addressing the lack of comprehensive investigation and evaluation of deployment strategies for AI applications on the AWS cloud platform. Specifically, the main contributions of this work are as follows:

- We evaluate five different deployment strategies across cloud, edge, and hybrid edge-cloud architectures. These strategies also consider both serverless and server-based compute options. The deployment architectures and methodologies are discussed in detail in chapter 3.
- We introduce the use of *SC* and *EE* to enable the implementation of hybrid edge-cloud architectures. We explore the feasibility and effectiveness of these methods through detailed system design and experiments.
- We conduct a comprehensive evaluation of each deployment strategy based on three key metrics: latency, accuracy, and cost. Through empirical analysis, we provide evidence-backed insights and practical recommendations for selecting appropriate deployment strategies for cloud-based AI systems.

1.3 Thesis Structure

The following chapter 2 introduces the background of the study, starting with a use case that presents a potential application scenario supported by this research. Then it presents some fundamental concepts such as the attributes and application domains of image classification tasks, the role and structure of CNNs, and some typical CNN architectures. Then this chapter also covers some other basic knowledge for the thesis, such as AWS introduction and relevant services, SC and EE strategies, 5G network with its benefits and limitations. The chapter concludes with a summary of related work, showing the context and novelty of this research.

Chapter 3 presents the system design. It begins by introducing the pre-defined CNN model used throughout the experiments, explaining its architecture, data pre-processing steps, training datasets, methods, and results. This chapter then elaborates on five deployment strategies on AWS: using Lambda in cloud nodes, EC2 in cloud nodes, EC2 in edge nodes, EC2 in hybrid nodes with straightforward split computing, and EC2 in hybrid nodes with EE-introduced split computing. Special attention is given to the implementation of split computing, including splitting point selection based on data transmission and user privacy considerations, the design of branch classifiers for EE, training methods, and threshold selection for fast inference.

Chapter 4 presents experimental results of each deployment strategy from three perspectives: latency, accuracy, and cost. Latency evaluations include overall latency distribution analysis and the latency components analysis like the computation and communication latency. Cost evaluations estimate monthly cost, potential request volumes, and analyze the pricing model of serverless Lambda function. Accuracy evaluations focus on age and gender classification accuracy across different deployments.

Finally, Chapter 5 concludes the thesis by summarizing the key findings and discussing potential research topics for future work.

2 Background

This chapter primarily introduces the background of the study, including the application context of the research topic, basic background knowledge, and a review of related work.

2.1 Use Case

This study focuses on the process, methods, and outcomes of deploying a pre-trained AI model on the public cloud platform AWS. In this section, we describe a practical use case to illustrate the potential application scenarios of the proposed system and to define the scope of our research. Since our discussion is centered on a 5G-enabled application, and the performance of 5G networks can vary significantly across regions and time of day. Additionally, AWS services are region-specific in terms of both infrastructure and pricing, which may affect the cost, latency, and overall performance of the deployment strategy. Therefore, specifying the scope of this study is necessary.

Our application is based on an in-cabin system designed to enhance the comfort and personalization of services offered to vehicle occupants [6]. One of the core technologies behind this system is the use of AI models for facial recognition and attribute extraction, such as the prediction of gender and age. These attributes can then be used to personalize in-cabin services more effectively, such as automatic adjustment of seating, lighting, and media based on individual occupant profiles.

In this use case, we assume a nationwide deployment across Germany. As shown in Figure 2.1, vehicles equipped with the system are distributed throughout the country. When a person enters a car, the AI model is triggered to detect and analyze facial data in real time. The captured facial image is then transmitted to the cloud for inference. The estimated prediction results, which are generated by a pre-trained CNN model, are sent back to the vehicle client, enabling real-time service personalization.

In the Germany scope, AWS maintains its primary data center in Frankfurt, which handles cloud-based processing. What's more, AWS also offers edge computing capabilities at three Wavelength Zones within Germany: Berlin, Munich, and Dortmund [7]. When a user is located near one of these zones and the system is configured for edge or hybrid edge-cloud deployment, requests can be processed locally at the edge. This configuration reduces communication latency and improves flexibility. In areas not

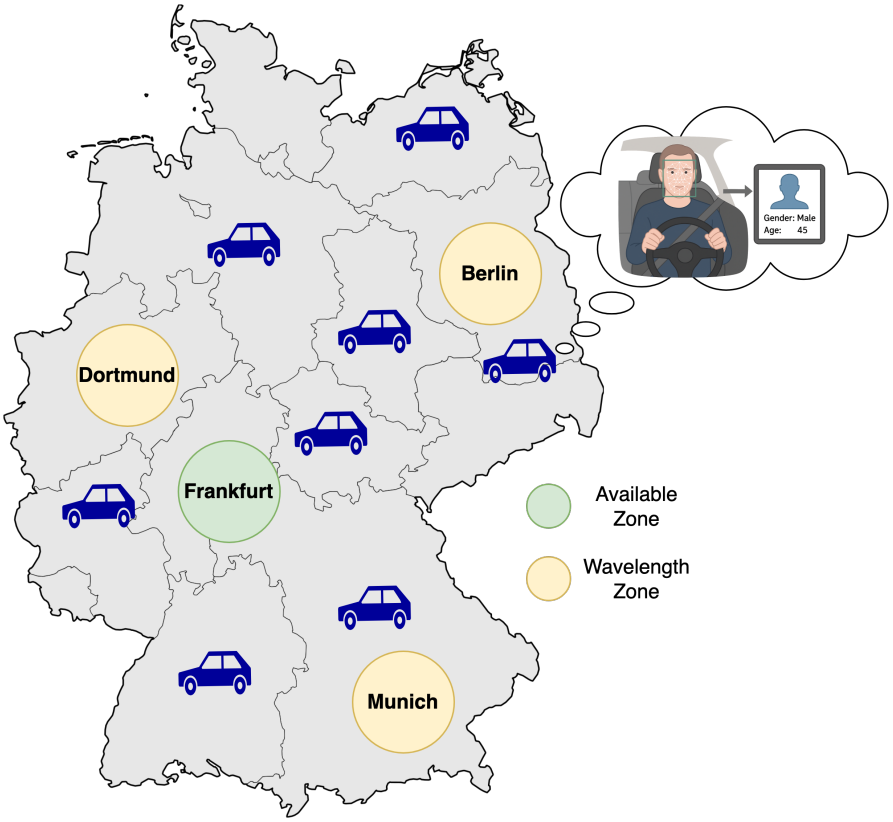


Figure 2.1: A Use case illustration for in-cabin system deployments across Germany with the integration of AWS available zones and wavelength zones for cloud and edge computing.

covered by Wavelength Zones, data must be sent directly to the available zone for processing.

2.2 Image Classification

Image classification is a fundamental task in computer vision, playing a significant role in many real-world applications, including medical diagnostics [8, 9], intelligent surveillance [10], autonomous driving [11], and human-computer interaction [12]. It learns how to extract and interpret visual features and assign categorical labels to input images. A typical pipeline usually involves image preprocessing, segmentation, feature extraction, and classification.

Specifically, facial image classification makes it possible to infer an individual's

demographic attributes, such as age, gender, ethnicity, and emotional state. These attributes can be extracted from a single facial image and are particularly useful in applications such as access control systems, personalized advertising, and intelligent in-cabin systems. In the automotive domain, such systems can use real-time demographic inference to provide personalized and more comfortable services to in-car occupants.

One of the biggest challenges in this task is extracting effective features from facial images. CNNs have shown remarkable success in this area, thanks to their ability to learn hierarchical representations of visual data. CNNs outperform traditional machine learning methods in a variety of image-based tasks, including object detection, face recognition, and classification [13].

2.2.1 Core Components of CNNs

CNNs are composed of three main types of layers: convolutional layers, pooling layers, and fully connected layers. Figure 2.2 shows the structure of LeNet-5, an early CNN architecture proposed by Yann LeCun [14].

Convolutional Layer

The most fundamental building block of CNNs is the convolutional layer. It computes local operations with learnable filters (also referred to as kernels) that slide over the input image or feature maps to generate output feature maps. Each filter is used for detecting particular patterns or features like edges, corners, and textures. After convolution, a non-linear activation function (for example, ReLU, sigmoid, or tanh) is used to bring non-linearity into the model.

Pooling Layer

Pooling layers (also referred as downsampling or subsample layers) decrease the spatial dimensions of the feature maps and thus reduce the number of parameters as well as the amount of computation. Pooling makes the layers translation-invariant and robust for feature extraction. Some of the popular pooling operations are max pooling and average pooling.

Fully Connected Layer

The Fully Connected (FC) layers are generally located towards the end of a CNN architecture. Every neuron within an FC layer is connected with all the neurons of the layer before it. These layers act as a classifier, mapping the high-level extracted features to the output labels. The last layer commonly uses a softmax activation to

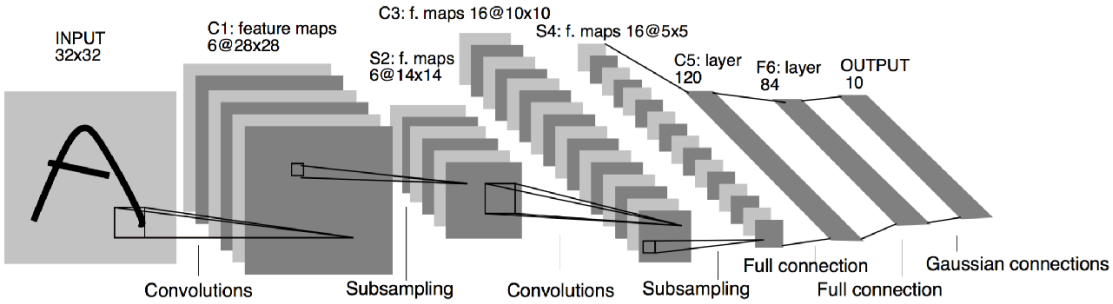


Figure 2.2: The architecture of LeNet-5 network [14].

produce a probability distribution over classes. Other classifiers like Support Vector Machine (SVM)s can also be used for certain tasks.

2.2.2 Popular CNN Architectures

There have been numerous CNN architectures proposed and used in literature with differences in depth, principles of design, and corresponding performance trade-offs:

AlexNet

AlexNet is introduced by Krizhevsky et al. [15]. It marked a significant deep learning breakthrough with an award in the ImageNet Large Scale Visual Recognition Challenge 2012. The architecture consists of eight layers, including five convolutional and three fully connected layers. The innovation involved that they used the ReLU as an activation function which can prevent vanishing gradients and introduced the concept of dropout as a regularization technique that can prevent overfitting by randomly disabling the neurons. The process of generalization was enhanced with the utilization of data augmentation techniques such as random cropping and horizontal flipping. AlexNet also took advantage of Graphics Processing Unit (GPU) acceleration for deeper training. These innovations together reduced the top-5 error rate of the ImageNet to 15.4%, significantly outperforming previous methods.

VGGNet

VGGNet [16], built by the Oxford's Visual Geometry Group, investigated the impact of depth in performance by using stacks of 3×3 convolutional filters with fixed stride and padding. VGG16, which is the 16-layer variant, proved particularly popular due to its simplicity and excellent performance. Even with the identical architecture, the

composition of small filters allowed the network to approximate large receptive fields (e.g., three 3×3 layers result in an 7×7 effective receptive field), while introducing additional non-linearities and reducing the number of parameters compared to large filters. Such depth enabled more complex hierarchical representations of features and reached a 7.3% top-5 error rate on ImageNet. The model's regular structure also made it well-suited for transfer learning in downstream tasks.

GoogLeNet (Inception)

GoogLeNet, or Inception v1 [17], proposed the Inception module, which is a new structure that enabled multi-scale feature extraction simultaneously. Inside every module, 1×1, 3×3, and 5×5 convolutions run in parallel and their outputs are concatenated. The 1×1 convolutions not only added more non-linearity but also acted as a tool for reducing dimensionality and computational cost. This architecture enabled deeper depth without an explosion in parameters, and it achieved an error rate of 6.7% on the ImageNet's top-5 rankings. The Inception's modular architecture also inspired later advances in network architecture design.

ResNet

Residual Networks (ResNet) [18] solved the problem of the degradation in very deep networks, when the accuracy saturates or even drops with an increase in layers. ResNet brought in the idea of residual learning using the concept of shortcut (skip) connections that enabled the flow of gradients to propagate more easily through identity mappings. This enabled the training of extremely deep networks (e.g., 152 layers), which otherwise used to experience the vanishing gradient problem. The residual connections maintained the low-level information and enhanced convergence and ended up giving a superior performance. ResNet has a top-5 error rate of 3.6% on the ImageNet and is one of the most popular architectures of deep learning.

DenseNet

DenseNet [19] extended the connectivity concept even further with the inclusion of dense blocks, where each layer receives input from all preceding layers. This approach facilitates the reuse of features, enables efficient gradient propagation, and reduces the number of parameters. By directly connecting all the layers in a block, DenseNet extracts rich and complementary features and eliminates redundancy. This architecture gives a boost in terms of accuracy using fewer parameters when compared with similar deep networks.

Summary

In the current work, we opted for a VGG16-like model for age and gender classification due to the following compelling factors:

- **Transferability:** VGG16 is well suitable for transfer learning with the help of a large collection of publicly accessible pre-trained weights trained on large-scale datasets like Internet Movie Database and Wikipedia Dataset (IMDB-WIKI) and ImageNet. The pre-trained models enable efficient adaptation to new tasks with limited data.
- **Simplicity:** VGG16 architecture is fairly straightforward and uniform involving solely 3×3 convolutional filters and 2×2 max-pooling layers. Such a uniform structure makes it convenient for implementation, debugging, and architecture modifications.
- **Performance:** Although deeper architectures such as ResNet often outperform VGG16 in certain benchmarks, VGG16 is still a good and reliable baseline with comparable accuracy on age and gender classification tasks. As shown in [20], VGG16 is excellent for facial feature extraction and hence is a good choice for demographic attribute prediction.

Thus, using a pre-trained VGG16-based network enables accurate and efficient facial demographic attribute prediction and is a strong candidate for the proposed system.

2.3 Amazon Web Services

AWS is a widely used, highly reliable cloud-based computing platform that allows users to develop, deploy, and manage applications without the need to invest in physical infrastructure. Its pay-as-you-go pricing model eliminates the upfront costs of computation power, storage and other resources and makes them accessible when needed. AWS offers more than 200 full-featured services across the data centers worldwide and offers solutions for a wide range of industries and user types, such as governments, educational institutions, non-profit organizations, startups as well as large businesses [21].

One of the most impressive technologies in recent years is machine learning, particularly with the rapid rise of Large Language Model (LLM) and other deep learning models. However, their development and deployment come with significant challenges, especially in terms of the massive computational and storage resources they require [22]. To train and run these models is far beyond the scope of on-premises infrastructure.

Cloud providers such as AWS present a compelling solution in this regard by providing scalable, on-demand access to high-performance computing, distributed storage, and specialized machine learning services.

2.3.1 Global Infrastructure: Regions and Zones

AWS operates its infrastructure worldwide with 36 geographic regions and more than 114 Availability Zone (AZ)s. This enables users to deploy the applications closer to users so that reduce latency and enhances performance. AWS infrastructure is designed to meet the highest standards for availability, fault tolerance, and data protection.

Regions

An AWS region is a geographic area across the world where AWS clusters its data centers. Each region is divided into several AZs that are geographically distinct but networked connected with low-latency links. In contrast with other cloud providers that tend to treat a region as a singular data center, AWS's multi-AZ infrastructure makes it possible for clients to design fault-tolerant and highly available applications. Each AZ has its own power, cooling, and physical security.

Availability Zones

An AZ is one or more standalone physical data centers with redundant power supply, networking, and connectivity within an AWS Region. AZs allow customers to run more highly available, fault-tolerant, and scalable applications compared to a single data center. AZs are linked with high-bandwidth, low-latency regional fiber networks that provide synchronous replication and secure communications between zones.

AWS Wavelength

AWS Wavelength extends AWS services to the 5G network edge, allowing developers to build ultra-low latency applications for mobile and edge devices. By integrating compute and storage services directly into telecommunication providers' data centers at the 5G edge, Wavelength reduces the number of hops over the network and can achieve single-digit millisecond latencies. In this context, these telecommunication providers act as Communications Service Providers (CSPs)—a broad category that includes companies offering voice, data, and connectivity services. Internet Service Providers (ISPs), which specifically provide internet access to end users, are considered a subset of CSPs. By partnering with such CSPs, AWS is able to deploy infrastructure closer to end users, leveraging the CSPs' network coverage to deliver cloud services

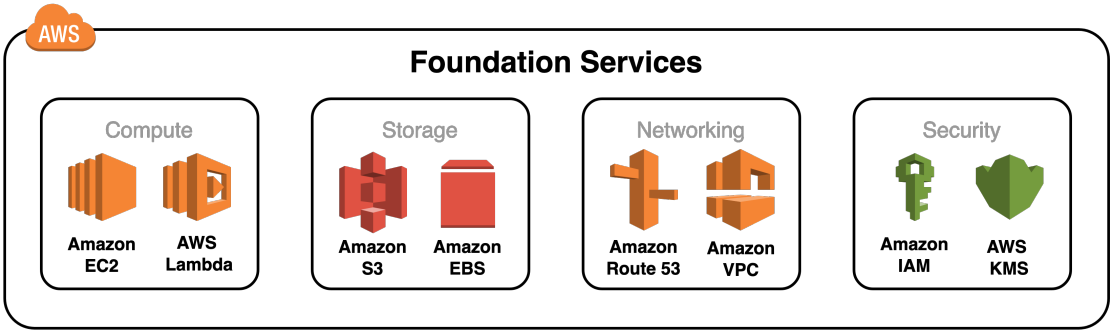


Figure 2.3: Overview of representative AWS services across computing, storage, networking, and security.

with minimal latency. This architecture is particularly useful for applications like real-time gaming, live video streaming, AR/VR, and edge machine learning inference.

AWS Local Zones

AWS Local Zones extend AWS infrastructure closer to end-users in target cities, allowing applications with ultra-low-latency requirements such as real-time video, gaming, simulation, and machine learning. Local Zones offer access to fundamental services like EC2, Elastic Block Store (EBS), Virtual Private Cloud (VPC), and ELB and offer the same secure and high-bandwidth connectivity as the parent AWS Region.

2.3.2 AWS Services

AWS offers a wide range of services covering various aspects such as computing, storage, networking, security, monitoring and so on. Figure 2.3 illustrates some of its representative services.

Compute Services

AWS has a broad set of compute services that support various deployment patterns and workload requirements:

- **Amazon EC2:** Provides resizable compute capacity in the cloud. Users can launch virtual machines or instances that meet specific processing, memory, or GPU requirements.
- **AWS Lambda:** Enables serverless computing, allowing users to run code in response to events without provisioning or managing servers. Lambda is suitable

for event-driven applications and backend logic execution. AWS Lambda supports a variety of programming languages, including Node.js (JavaScript), Python, Java, C#, Ruby, and more. It also allows the deployment using container images.

- **Amazon Elastic Container Service (ECS) and Elastic Kubernetes Service (EKS):** AWS provides managed services by running containerized applications using Docker (ECS) or Kubernetes (EKS), offering scalability, automation, and simplified orchestration.

Storage Services

In order to facilitate scalable, long-term, and cost-efficient data storage, AWS has numerous storage solutions:

- **Amazon Simple Storage Service (S3):** A highly scalable and durable object storage service, which is suitable for backups, data lakes, and static content hosting.
- **Amazon EBS:** Provides block-level storage volumes for persistent data used by EC2 instances.

Networking Services

AWS provides a comprehensive set of networking services to support secure, scalable, and high-performance application delivery:

- **Amazon VPC:** Allows users to provision logically isolated networks within the AWS cloud, it provides full control over IP ranges, subnets, routing, and security.
- **Amazon Route 53:** A scalable and highly available Domain Name System (DNS) web service used to route traffic to AWS resources and external endpoints efficiently.

2.3.3 Summary

AWS provides a robust and globally distributed cloud infrastructure with a wide range of services, from general-purpose compute and storage to advanced analytics and machine learning. Its scalable architecture, combined with latency-aware deployment strategies such as Wavelength and Local Zones, makes it particularly suitable for modern applications that require low-latency inference, real-time response, and operational flexibility—such as the in-cabin AI-based systems explored in this study.

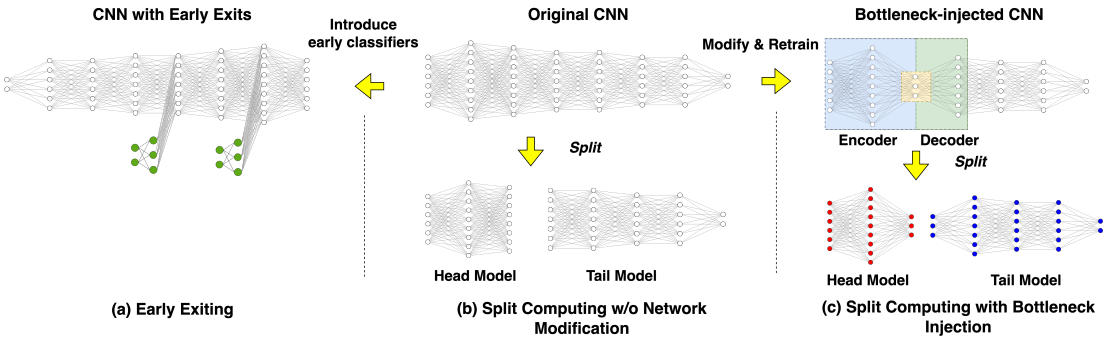


Figure 2.4: Illustration of different split computing strategies: (a) with early exits, (b) without network modification, (c) with bottleneck injection. [4]

2.4 Split Computing

Deep Learning (DL) has enabled significant progress in a number of application domains like Computer Vision (CV), NLP, and wireless communications [23]. Nevertheless, current Deep Neural Network (DNN)s tend to be computationally and memory intensive and hence hard to deploy on resource-limited mobile or edge devices for real-time inference.

To address this, SC has emerged as a promising paradigm. Rather than running the entire model on the mobile or edge device or a cloud server, SC partitions the DNN into several parts, typically a *head* running on the mobile or edge device and a *tail* running on an edge or cloud server. This division aims to reduce computation on the resource-constrained device, offloading computationally expensive tasks, and minimize latency by efficient partitioning the model. Although SC is applicable for mobile-edge-cloud hierarchies, in this work we focus specifically on edge-to-cloud offloading.

In early SC implementations, the model is divided at a specific layer ℓ out of L total layers, so the head and tail comprising the first ℓ layers and the remaining $L - \ell$ layers, respectively. Each segment has the same architecture and weights as the original model. This strategy preserves accuracy but may not reduce total latency, especially when intermediate outputs are large and hence lead to significant communication costs.

To overcome this, recent work introduces bottlenecks at the split point to reduce feature size [24], or adopts adaptive mechanisms such as EE to dynamically get inference results based on input data. Figure 2.4 shows examples of SC without network modification, SC with bottleneck injection, and early exiting structures.

2.4.1 Split Computing without Network Modification

Split computing without network modification is the most straightforward implementation strategy. The front layers are running on the edge device, and the intermediate feature maps will be sent to the cloud for continued inference. This approach is easy to deploy for pre-trained models, but its efficiency mainly depends on the split location. Early layers often produce high-dimensional feature maps, which will lead to the increasement of transmission overhead.

Kang et al. [25] showed that for many DNNs, executing the entire model on either the client or the server often yields better performance in terms of latency and energy than naive splitting, unless the model contains natural bottlenecks (i.e., layers with smaller output tensors than previous input). Moreover, privacy is another motivation for SC. Jeong et al. [26] showed that partial offloading can conceal raw input data from the cloud server, enhancing user privacy in sensitive applications.

2.4.2 Early Exiting

EE is an orthogonal technique that improves inference efficiency by introducing intermediate branch classifiers, or "exits," into the DNN [27]. During inference, if a branch classifier can produce a high-confidence prediction, the computation stops early, avoiding the execution of deeper layers.

EE significantly reduces average latency and energy consumption, particularly for "easy" inputs. It has been extended to hierarchical systems [28], where a lightweight model runs on a mobile device, and imprecise predictions being sequentially forwarded to more advanced models on the edge and cloud servers. It should be kept in mind that the additional branch classifiers should be lightweight otherwise the computation cost of running multiple exits might surpass their advantage.

Together, split computing and early exiting provide adaptive strategies for deploying DNNs under mobile and edge computing constraints, offering trade-offs between latency, accuracy, and resource efficiency.

2.5 5G Networks

Fifth-generation (5G) mobile networks represent a groundbreaking advancement in wireless communication technology. Compared to previous generations, 5G introduces a set of technical innovations that make it particularly suitable for supporting data-intensive, latency-sensitive, and AI-enabled applications.

Enhanced Data Rates. 5G networks offer significantly higher bandwidth, with peak theoretical speeds reaching up to 20 Gbps for downlink and 10 Gbps for uplink [29]. This facilitates smooth high-definition stream delivery, quicker downloads, and optimized transfer of large amounts of data, which is a prime requirement for uses such as real-time video analytics and cloud-based inference.

Ultra-Low Latency. One of the target features of 5G is the ability to reduce end-to-end latency to as low as 1 millisecond [30]. Such low-latency communication is critical for time-sensitive applications such as autonomous driving, remote surgery, and in-cabin AI services, where real-time response is vital for safety and user experience.

Network Slicing. 5G introduces the concept of network slicing, which allows operators to create multiple virtual networks over a shared physical infrastructure. Each of these slices is tailored to the performance, reliability, and security requirements of individual services or applications.

Integration with Edge Computing. 5G is inherently designed to support edge computing through frameworks like Multi-access Edge Computing (MEC) and services like AWS Wavelength. By allowing the placement of compute resources near the user, 5G reduces the dependence on distant cloud data centers and reduces the communication latency as well [31]. This is particularly relevant for split computing architectures where parts of the AI model are deployed at the edge for low-latency inference.

Practical Limitations of Commercial 5G. Despite these capabilities, 5G is still an emerging technology and its performance in real world does not always match theoretical expectations. Hu et al. [32] conducted empirical tests using commercial 5G smartphones in major U.S. cities, and evaluated both Sub-6 GHz and mmWave deployments.

Their tests revealed that even though mmWave 5G supported a download speed of up to 2 Gbps in ideal situations, overall performance fluctuated greatly based on signal quality, user mobility, and environmental factors. Sub-6 GHz bands offered more consistent coverage but only modest improvements over 4G LTE.

Latency was reduced compared to 4G, with an average 20–30 ms round-trip time, but still far from the 1 ms target. Additionally, mmWave usage led to more than double the power consumption, raising concerns about energy efficiency and battery life. These findings indicate that infrastructure and protocol optimizations are still needed to fully realize 5G's potential.

2.6 Summary of Related Work

Several studies have explored deploying applications (like AI applications) on cloud platforms, especially AWS, evaluating their performance and suitability for various use cases.

Bagai et al. [33] investigates various AWS services such as EC2, SageMaker, and Lambda for machine learning model deployment. The study provides a detailed comparison in terms of ease of deployment, scalability, and inference latency. It emphasizes that SageMaker offers scalable infrastructure at a relatively low cost, while Lambda is well-suited for event-driven and cost-sensitive tasks. Additionally, ECS supports dynamic scaling based on service demand. Bayazitov et al. [34] presented a cloud-based architecture using AWS services including S3, EC2, Lambda, and SageMaker for AI integration. The article states that the system can process audio/video data efficiently, achieving high responsiveness, scalability, and cost-effectiveness. However, the above work lacks detailed quantitative experimental results.

Doukha et al. [35] conduct an empirical study on training several CNN architectures on Amazon EC2 instances. By monitoring GPU utilization, execution time, and associated costs using Amazon CloudWatch, the study identifies scenarios in which upgrading to more powerful GPU instances leads to better efficiency and cost-effectiveness, particularly when GPU usage exceeds 90%. Chahal et al. [36] present a comparative analysis of AWS SageMaker and AWS Lambda for deploying a deep learning-based recommender system. Their evaluation focuses on response time and cost efficiency under varying levels of concurrency. The findings highlight trade-offs between the two services, noting that the cost associated with instance scaling does not grow proportionally with load. Moreover, serverless architectures demonstrate superior autoscaling behavior and overall performance compared to dedicated endpoints. Roy et al. [37] developed a real-time fruit recognition and calorie estimation system using a 5-layer CNN model, achieving 99.45% test accuracy. The model was deployed on AWS EC2 using a Flask-based web service.

The above mentioned articles mainly focus on cloud-only deployment and do not consider edge computing or hybrid architecture. Rao et al. [38] introduce the ECO (Edge-Cloud Optimization) system, a dynamic microservice deployment framework designed for 5G applications. The system adaptively maps services between edge and cloud environments based on inter-service communication patterns and latency requirements. Validated on AWS/Verizon Wavelength edge infrastructure with two real-world video analytics applications, ECO achieves a twofold improvement in application response time compared to static edge-cloud deployments. Rao et al. [39] also introduced ECO-LLM, a hybrid edge-cloud optimization framework using large language models to automate microservice deployment based on natural language

inputs. Evaluated on video analytics tasks, it achieved over 98% decision accuracy and effectively minimized latency by dynamically selecting between AWS EC2 and local edge nodes. Ogden et al. [40] proposed the Layercake system, a hybrid inference architecture combining mobile devices with AWS EC2 cloud resources. It can achieve 20–45% latency reduction and 30–60% cost savings compared to cloud-only approaches, enabling efficient, scalable, and adaptive model serving.

Collectively, these studies underscore the growing importance of cloud and edge platforms in supporting scalable, efficient, and low-latency deployment of deep learning models. Nevertheless, comprehensive investigations involving application deployment and empirical analysis on AWS remain relatively limited. To summarize existing work, table 2.1 presents a comparative overview of recent studies, highlighting their deployment architectures, AWS services used, evaluation metrics, and the presence of empirical results. This paper aims to address this lack by presenting a comparative evaluation of diverse deployment strategies, including serverless and server-based, cloud, edge, and hybrid configurations. Furthermore, we incorporate advanced techniques such as SC and EE for dynamic inference mechanisms. Empirical results on key metrics such as latency, accuracy, and cost are provided to support the analysis and offer actionable insights for real-world implementation.

Table 2.1: Comparison of studies on AWS-based application deployment.

Study	Cloud	Edge	Hybrid	EC2	Lambda	Metrics	Empirical Results
Bagai et al. [33]	✓	✗	✗	✓	✓	Ease of deployment, scalability, latency	✗
Bayazitov et al. [34]	✓	✗	✗	✓	✓	Responsiveness, scalability, cost	✗
Doukha et al. [35]	✓	✗	✗	✓	✗	GPU utilization, latency, cost	✓
Chahal et al. [36]	✓	✗	✗	✗	✓	Latency, cost	✓
Roy et al. [37]	✓	✗	✗	✓	✗	Accuracy	✓
Rao et al. [38]	✓	✓	✓	✓	✗	Latency	✓
Rao et al. [39]	✓	✓	✓	✓	✗	Accuracy, latency	✓
Ogden et al. [40]	✓	✓	✓	✓	✗	Latency, cost	✓
This study	✓	✓	✓	✓	✓	Latency, cost, accuracy	✓

3 System Design

This chapter outlines the overall system design, including the CNN model for age and gender prediction and its deployment on cloud and edge infrastructures. We begin by detailing the dataset, model architecture, and training process, followed by a description of different deployment strategies using AWS. Finally, we introduce SC and EE mechanisms to improve inference efficiency and adaptability.

3.1 CNN Model

3.1.1 Dataset

The CNN model developed in this work aims to predict both age and gender by analyzing facial features. For this purpose, we adopted the UTKFace dataset, as illustrated in Figure 3.1. The UTKFace dataset is a large-scale face dataset with a wide age span, ranging from 0 to 116 years old. It contains over 20,000 facial images annotated with age, gender, and ethnicity information. The images exhibit a high degree of variability in terms of pose, facial expression, illumination, occlusion, and resolution, making it well-suited for real-world facial analysis tasks [41].

Each image file provides ground-truth annotations for both age and gender labels. Specifically, age is an integer ranging from 0 to 116, indicating the actual age of the individual, while gender is either 0 (male) or 1 (female). These annotations allow the model to learn and predict both continuous and categorical outputs.

For age prediction, we employed two strategies: (1) a regression model to estimate a continuous numerical age, and (2) a classification model that categorizes ages into five discrete age groups, defined as follows:

- **Infant**: 0–2 years
- **Child**: 3–9 years
- **Teenager**: 10–19 years
- **Adult**: 20–59 years
- **Elderly**: 60 years and above

Figure 3.1: Sample images from the UTKFace dataset with age and gender annotations. [41]

Tables 3.1a and 3.1b summarize the number of samples in each age category and gender within the training and validation sets, respectively.

Table 3.1: Number of samples per age group and gender in the training and validation sets.

(a) Training Set

Age Group	Male	Female	Total
Infant (0–2)	769	672	1441
Child (3–9)	469	741	1210
Teenager (10–19)	559	684	1243
Adult (20–59)	6966	6171	13137
Elderly (60+)	1261	928	2189
Total	**10024**	**9196**	**19220**

(b) Validation Set

Age Group	Male	Female	Total
Infant (0–2)	192	168	360
Child (3–9)	116	187	303
Teenager (10–19)	138	173	311
Adult (20–59)	1744	1540	3284
Elderly (60+)	316	231	547
Total	**2506**	**2299**	**4805**

3.1.2 Data Pre-processing

In the data pre-processing stage, we first utilize the MediaPipe Holistic module [42] to detect facial landmarks within the input images. Specifically, we extract the contour points of the jawline and use the coordinates between the ears and the chin to define a tight cropping region. This ensures that the main facial features are preserved while minimizing the inclusion of background areas. To meet the input requirements of the

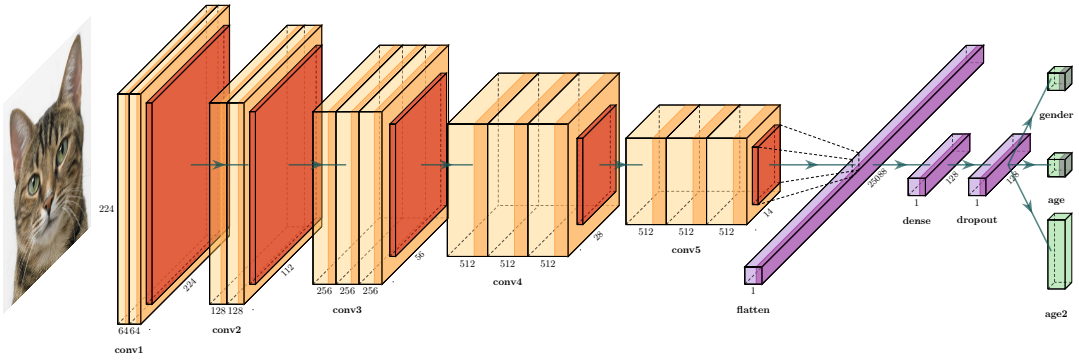


Figure 3.2: The pre-trained CNN model architecture based on VGG16 design.[1]

Visual Geometry Group (VGG)-16-based network, all cropped facial regions are resized to a fixed resolution of 224×224 pixels for training and validation purposes.

3.1.3 Model Architecture

To jointly predict gender and age from facial images, we designed a modified Convolutional Neural Network inspired by the classical VGG-16 architecture, as illustrated in Figure 3.2. The model consists of five convolutional blocks (conv1 to conv5). Each block comprises 2 to 3 consecutive `Conv2D` layers, followed by a `MaxPooling2D` layer for spatial downsampling.

All convolutional layers utilize small kernels (3×3) and progressively increase the number of filters from 64 to 512, following VGG16's "deep and narrow" design strategy, which enhances the feature extraction capability. After the conv5 block, a `Flatten` layer transforms the feature maps into a one-dimensional vector, which is then passed to a fully connected layer with 128 units. A `Dropout` layer is applied thereafter to reduce overfitting.

Following this shared representation, the model branches into three task-specific output layers:

- **gender_output:** Binary classification output for gender prediction.
- **age_output:** Regression output for continuous age estimation.
- **age_output2:** Multi-class classification output for age group prediction.

[1]Figure generated using the open-source tool: `https://github.com/HarisIqbal88/PlotNeuralNet`.

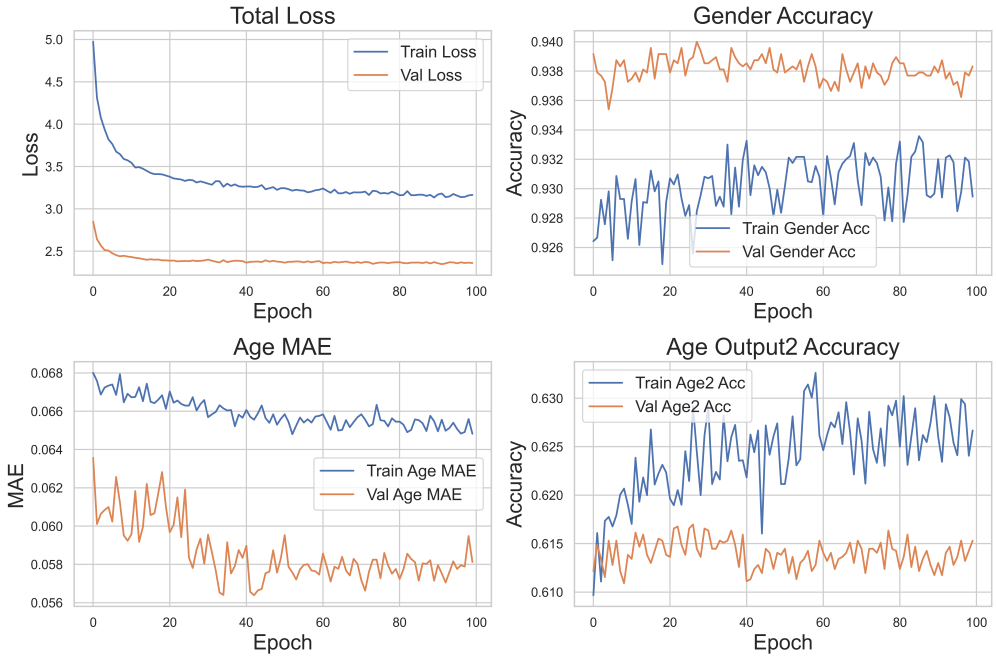


Figure 3.3: Training curves for all tasks.

3.1.4 Model Training

As discussed in Section 3.1.1, the dataset exhibits class imbalance across both age groups and gender. To ensure fairness in training and reduce bias, the pre-trained CNN model is trained using class weighting techniques. This balances the contribution of each class in the loss function. Additionally, data augmentation methods are employed to improve generalization and robustness against overfitting.

Table 3.2 presents the training and validation performance metrics for gender classification, age regression, and age group classification.

For continuous age estimation, the evaluation metric used is the *Mean Absolute Error (MAE)* , which quantifies the average absolute difference between the predicted and ground-truth ages. A lower MAE indicates better predictive accuracy. The MAE is defined as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.1}$$

where y_i denotes the actual age and $\hat{y}_i$ is the predicted value.

Furthermore, visual illustrations of the training and validation accuracy and loss curves are provided in Figures 3.3. An interesting phenomenon that can be observed in

the figure is that, due to the different evaluation methods for age MAE and age output2 accuracy, the two metrics may show different trends. For example, age MAE measures the absolute error, while age output2 accuracy checks whether the predicted age falls within a predefined target age range. As a result, it is possible that the age MAE shows a downtrend, which indicates improved performance, while the age output2 accuracy remains nearly unchanged. This is because even if the predicted age value becomes closer to the true value, the mapped age group may still be the same, leading to the age output2 accuracy remaining unchanged.

Table 3.2: Training and validation performance metrics.

Task	Train Accuracy (%)	Val Accuracy (%)
Gender	92.9	93.8
Age Classification	62.7	61.5
	Train Error (MAE)	Val Error (MAE)
Age	6.5	5.8

3.2 Deployment Using AWS

This chapter outlines the deployment strategies of the proposed CNN model on the AWS platform. Given the vast array of services provided by AWS, it is impractical to evaluate each service individually. Therefore, this study focuses on two main comparative dimensions: compute model (server-based vs. serverless) and architectural topology (cloud, edge, and hybrid).

On the compute model level, the comparison is made between server-based and serverless models. Serverless computing has gained significant popularity due to its simplified management and billing model, where users do not need to be concerned with the underlying infrastructure [43]. In AWS, Lambda service represents the serverless model, while Amazon EC2 serves as the traditional server-based counterpart.

From an architectural perspective, the advent of 5G networks with their promise of ultra-low latency has reinforced the importance of edge computing in modern system designs [44]. AWS Wavelength enables deployment standard AWS compute and storage services to the edge of CSPs' networks. Therefore, comparisons are made between cloud-only deployments, edge-only deployments, and hybrid edge-cloud deployments. For this study, deployments are assumed to occur within Germany. So the central cloud infrastructure is located in Frankfurt, while Wavelength Zones are available in Berlin,

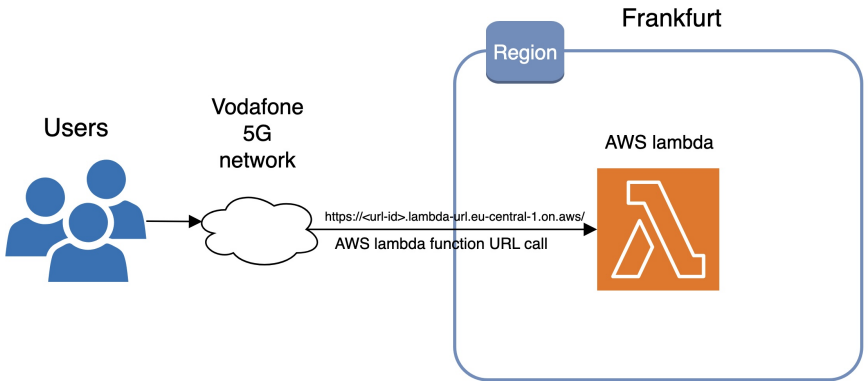


Figure 3.4: System architecture of the cloud-based deployment using AWS lambda.

Munich, and Dortmund. The experiments were conducted in Munich.

Considering both compute and architecture model, we evaluate the following four strategies:

- Lambda in Cloud
- EC2 in Cloud
- EC2 in Edge
- EC2 in Edge-Cloud Hybrid

Lambda was not deployed in the edge-cloud hybrid architecture because (1) AWS wavelength zone does not support Lambda. Currently, it only supports services such as EC2, EBS, VPC, as well as services that can coordinate or work with EC2, EBS and VPC, such as Amazon EC2 Autoscaling, Amazon EKS cluster, Amazon ECS cluster. Therefore, lambda was not deployed on the edge node, but EC2 was deployed on the edge node. (2) Lambda is deployed in the private VPC defined by AWS. The most direct way for EC2 in the wavelength zone to access lambda is to access lambda's public URL through the internet. Therefore, there is no ultra-low latency provided between Lambda and EC2; their cooperative use does not have any advantages in terms of latency, especially for global applications. So we do not choose Lambda as a cloud node or edge node in the hybrid architecture.

The following subsections describe each of the designed strategies in detail.

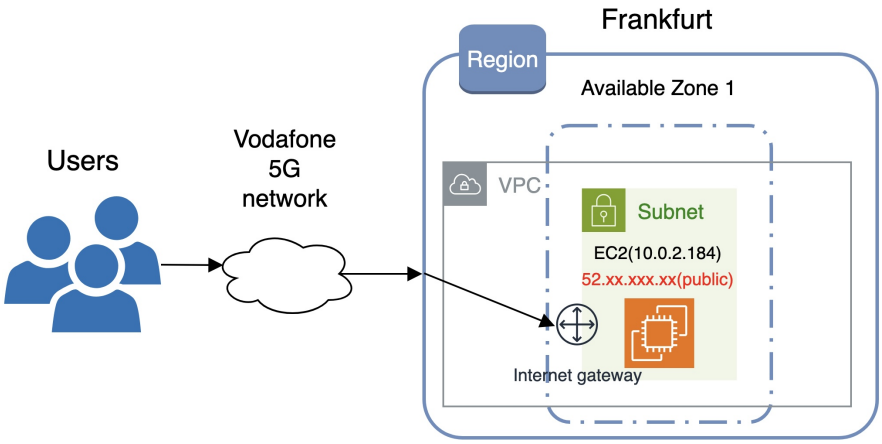


Figure 3.5: System architecture of the cloud-based deployment using AWS EC2. [2]

3.2.1 Lambda in Cloud

In this study, the Lambda function, as a representative serverless computing platform, was deployed via a container image, offering greater control over the runtime environment.

Since Lambda functions are event-driven, the function entry point is defined in a special handler function, commonly named `lambda_handler`. AWS invokes this function whenever the handler event is triggered. To ensure proper functionality, it is essential to implement this handler and define it as the container's startup command.

To enable external access to the Lambda function, a Function URL was configured with the authentication type set to `NONE`, allowing public access without AWS Identity and Access Management (IAM) credentials. This is suitable for public-facing serverless applications.

The invocation mode is set to `BUFFERED` (the default), where the entire payload is processed before a response is returned. This mode is appropriate for lightweight inference tasks.

Cross-Origin Resource Sharing (CORS) was enabled to ensure compatibility with browser-based clients. The 'Allow origin' was set to `*`, allowing requests from any domain, and the 'Allow headers' included `content-type`. The allowed HTTP method was `POST`.

The overall system architecture is illustrated in Figure 3.4. This configuration enables users to send HTTP requests to the Lambda function URL from any location and receive responses without restrictions.

[2]The internet gateway is attached to the VPC and then the route table is used to indicate which subnets

3.2.2 EC2 in Cloud

Amazon EC2 provides a server-based computing environment in which users can launch and manage virtual machines with fine-grained control over the underlying infrastructure. In this study, EC2 was employed to deploy a cloud-based inference server for the CNN model. The configuration details are as follows:

Instance Type Selection:

In this deployment, the `t3.medium` instance type was chosen. This type provides a balance of compute, memory, and network performance, suitable for moderate workloads. The `t3.medium` instance offers 2 vCPUs and 4 GiB of memory and supports burstable performance.

Storage Configuration:

To ensure persistent data storage, a 24 GiB General Purpose SSD (`gp2`) EBS volume was attached. `gp2` volumes are cost-effective and offer reliable performance for a wide range of applications.

Network and Security:

The EC2 instance was launched in a custom VPC with dedicated subnets and security groups to ensure isolation and security. To enable users to access the EC2 instance from the internet, a route to the Internet Gateway was added to the subnet's route table. The security group rules were configured as allowed:

- SSH access via port 22 for instance configuration
- ICMP for network latency testing (`ping`)
- TCP access on port 5201 for `iperf3` bandwidth testing
- TCP access on port 5000 for running the CNN model via Flask

S3 Integration:

To facilitate deployment, the source code was uploaded to Amazon S3. An IAM role with the `AmazonS3FullAccess` policy was created and attached to the EC2 instance, enabling it to fetch and execute the application code.

Flask Application:

Flask is a micro web framework for Python used to expose the CNN model on port 5000. Unlike full-stack frameworks, Flask focuses on minimalism and flexibility, making it well-suited for lightweight inference APIs.

use it. In the diagram, the route table is omitted for simplicity, and the internet gateway is shown as directly connected to the subnet to indicate that the subnet has a route enabling internet access. The same applies to the carrier gateway described later.

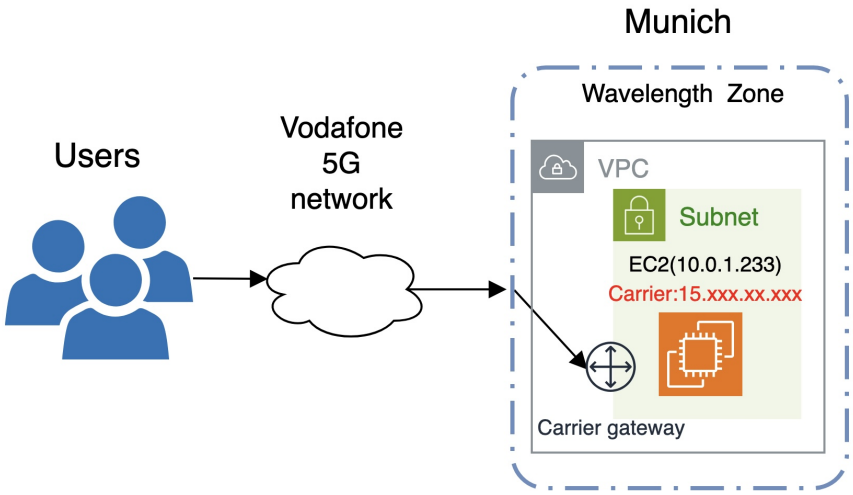


Figure 3.6: System architecture of the edge-based deployment using AWS EC2 and Wavelength Zones.

As shown in Figure 3.5, after completing the above configurations, users can send HTTP requests to the EC2 instance's public IP address on port 5000 using the `/predict` endpoint, and receive valid responses from the CNN model.

3.2.3 EC2 in Edge

To enable low-latency inference at the network edge, AWS EC2 instances can be deployed in Wavelength Zones, which brings compute resources closer to end users. The configuration for EC2 in the Wavelength Zone is largely consistent with the cloud-based EC2 deployment, with the following additional configurations:

- **Enable the Wavelength Zone:** In order to use the WZ, we need to opt in to the desired WZ from the EC2 dashboard and update the zone group accordingly.
- **Establish network connectivity:** After creating a VPC, we need to set up a carrier gateway and specify the subnets that will route traffic through it, which can enable the connectivity between the WZ and the telecommunications network infrastructure.
- **Define routing behavior:** We need a route table that includes a route for local traffic, a route for non-local traffic directed to the carrier gateway, and an association with the relevant subnets.

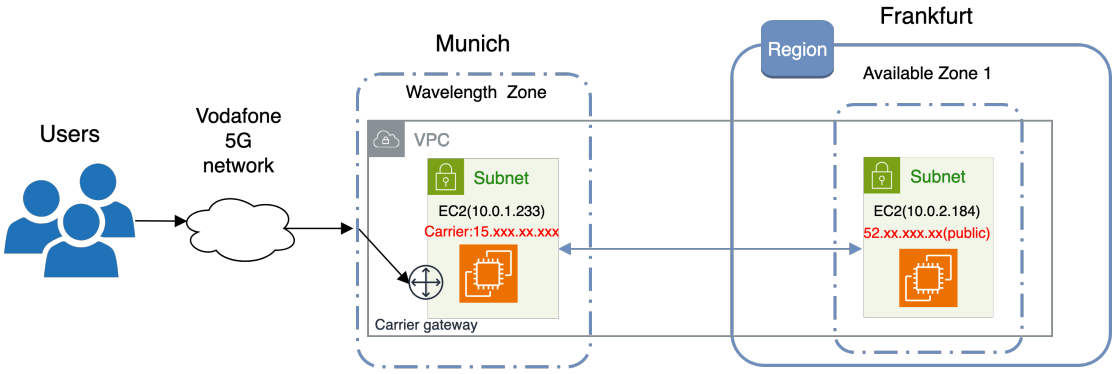


Figure 3.7: Hybrid deployment architecture combining cloud and edge EC2 instances.

Once the network infrastructure is fully set up, EC2 instances can be launched within the designated subnets located in the selected WZ, enabling ultra-low-latency access to edge-based compute resources.

The system architecture is illustrated in Figure 3.6. This setup allows users to send requests to the EC2 carrier IP address on port 5000 using the */predict* endpoint in the WZ and receive timely inference responses.

3.2.4 EC2 in Edge-Cloud Hybrid

To combine the low-latency benefits of edge computing and processing power of the cloud, a hybrid deployment strategy can be employed using both edge and cloud-based EC2 instances. For hybrid deployments, both cloud and edge EC2 instances are created. These instances must reside in the same VPC but in different subnets, as shown in Figure 3.7. In this configuration, users send requests to the EC2 instance at the edge, which may forward them to the cloud-based EC2 for processing. The processed results are then returned back through the edge instance to the user.

3.3 Split Computing

DNNs are frequently used to perform complex inference tasks such as image classification and speech recognition, which often require substantial computational resources and energy consumption. This poses a challenge for edge computing environments [45]. Edge-cloud hybrid models provide a promising solution by combining the low-latency characteristics of edge computing with the high computational capacity of cloud servers. One notable strategy support this hybrid model is *split computing*, which enhances flexibility by partitioning the DNN into multiple sequential segments. A common

implementation involves a two-part structure: a *head* model that runs on the edge device, and a *tail* model that executes in the cloud. By offloading only a portion of the computation to the cloud, this approach can effectively reduce both end-to-end latency and energy consumption.

Another related approach is *early exits*, which trains models to embed multiple "exits" at intermediate layers of the network. These exits can produce predictions with varying levels of accuracy, thereby allowing the tradeoff between accuracy and inference time to be dynamically adjusted based on current system conditions or application requirements.

In this section, we describe the design and implementation of split computing, including two strategies: *Split Computing without Network Modification* and *Split Computing with Early Exits*.

3.3.1 Splitting Point

One of the key challenges in split computing is determining the optimal point at which to divide the CNN model. In this study, this decision is guided by two primary considerations: *Data Transmission* and *User Privacy*. First, to minimize latency, the amount of data transmitted between the edge and the cloud should be as small as possible. Second, given that the task involves age and gender prediction from facial images—containing sensitive biometric information—transmitted data should ideally avoid including identifiable facial features in order to enhance user privacy.

Data Transmission

To maximize performance, it is important to identify whether there are any *natural bottlenecks* within the model architecture—i.e., intermediate layers whose output tensor size is significantly smaller than the input. Such bottlenecks can lead to reductions in latency, bandwidth usage, and transmission overhead.

Table 3.3 presents the output shapes and corresponding tensor sizes of all layers in the CNN model, including the initial input image. As observed, the output of the `block4_pool` layer is 100 352, which is smaller than the input tensor size of 150 528. This makes it a natural bottleneck and a strong candidate for the model splitting point.

User Privacy

In addition to transmission efficiency, privacy is a critical consideration when selecting a splitting point in split computing, particularly for applications involving sensitive data such as facial images. Since the inference task in this study involves age and gender

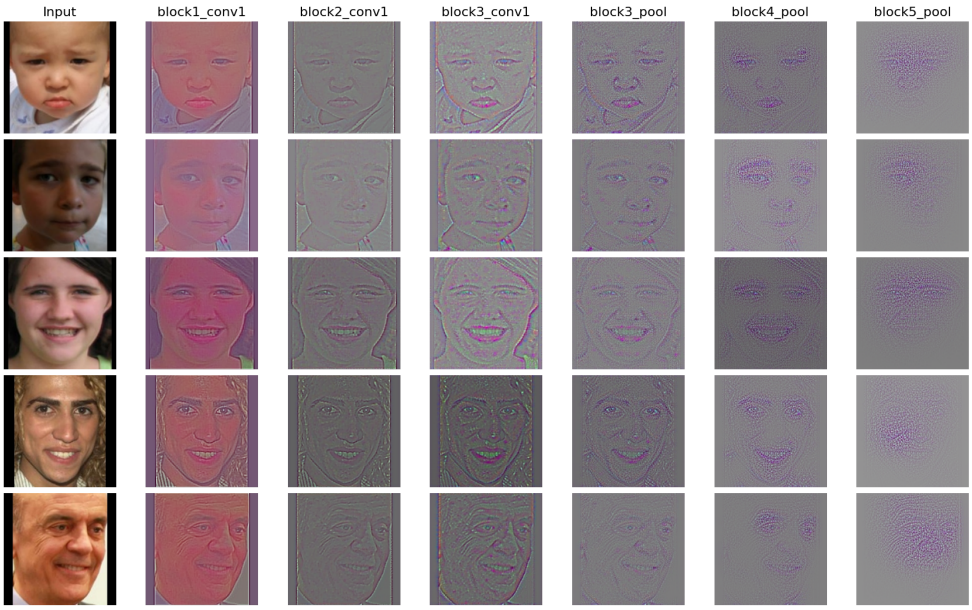


Figure 3.8: Reconstructed face images from different CNN intermediate layers using a deconvolutional approach. Deeper layers yield more abstract and less identifiable reconstructions.

prediction from facial inputs, it is essential to minimize the exposure of Personally Identifiable Information (PII) during transmission between edge and cloud nodes.

One strategy to preserve privacy is to select a splitting point at which the intermediate feature representations no longer retain human-interpretable visual characteristics. This helps reduce the risk of reconstructing the original face image from the transmitted data. In CNNs, early layers tend to capture low-level spatial features such as edges and textures, which are often sufficient to reconstruct recognizable images. In contrast, deeper layers extract more abstract, semantic features that are less likely to reveal detailed personal identity.

To visualize this phenomenon, we adopted the deconvolution-based reconstruction method described in [46]. Specifically, several face images from the dataset were reconstructed from the outputs of intermediate layers: `block1_conv1`, `block2_conv1`, `block3_conv1`, `block3_pool`, `block4_pool`, and `block5_pool`. One layer was selected approximately every three layers, and starting from `block3_pool`, pooling layers were used due to their reduced tensor sizes—making them strong candidates for splitting points.

As illustrated in Figure 3.8, reconstructions from deeper layers contain increasingly abstract and localized features, making it more difficult to visually identify the original

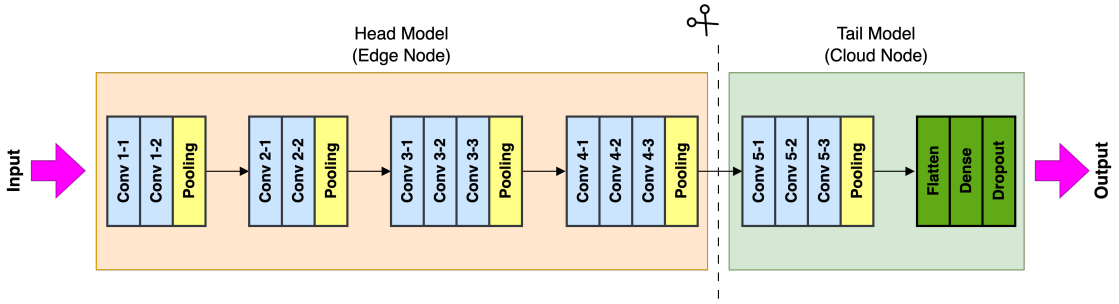


Figure 3.9: An illustration of the split computing architecture

subject. Therefore, selecting a splitting point after multiple convolutional and pooling operations, such as after `block4_pool`, ensures that the transmitted data carries semantically meaningful but anonymized representations. This aligns with the principle of data minimization in privacy-preserving machine learning, where only the minimum required information is shared across system boundaries.

Based on the above considerations regarding both data transmission and user privacy, the `block4_pool` layer was chosen as the model splitting point. All layers prior to this point are executed on the edge node, while the remaining layers run on the cloud node. The output tensor from the `block4_pool` layer is transmitted between the two nodes for inference continuation.

3.3.2 Split Computing without Network Modification

In the baseline split computing approach, the original CNN architecture is retained without any structural modifications. As illustrated in Figure 3.9, the model is partitioned into two sequential segments: a *head model*, executed on the edge node within the WZ, and a *tail model*, executed on the cloud node in the AZ. Based on the discussion in earlier sections, the `block4_pool` layer is selected as the splitting point.

During inference, the edge device processes the input image through the head model and generates an intermediate feature representation. This feature map is then transmitted to the cloud, where the tail model resumes computation to produce the final prediction. This approach preserves the original model architecture and serves as a reference baseline for evaluating the performance gains introduced by more advanced architectural modifications, such as early exits.

This method is straightforward to implement and remains fully compatible with existing pretrained models. However, all inputs must still be propagated through the entire model pipeline, which may result in redundant latency and energy consumption, especially for easily classifiable samples. Compared with traditional cloud-only deploy-

ment, this approach offers certain advantages from a privacy perspective: since only intermediate representations rather than raw data are transmitted to the cloud, sensitive personal data (such as facial features) remain on the edge. This benefit becomes more pronounced when the edge node is a trusted device under the control of the user or organization.

3.3.3 Split Computing with Early Exits

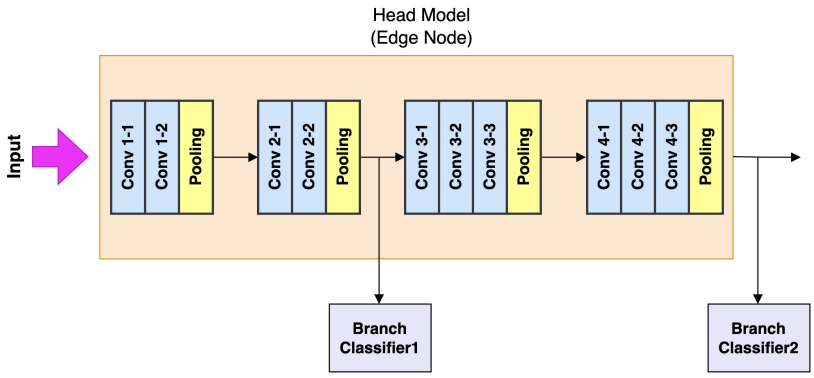
To enhance the efficiency and adaptability of split computing systems, we introduce early exits into the CNN architecture. The overall structure remains similar to the baseline approach, with the model split at the `block4_pool` layer. The head model is executed on the edge node in the Wavelength Zone, while the tail model is deployed on the cloud node in the AZ. The key difference lies in the integration of lightweight classifiers—referred to as *branch classifiers*—at selected intermediate layers.

These branch classifiers allow the system to produce predictions at earlier stages of the model, enabling early termination of the inference process when a sufficient level of confidence is achieved. This is particularly advantageous in scenarios with limited computational resources or where latency is a critical concern. When confidence thresholds are not met, inference continues in the traditional manner, ensuring accuracy is preserved.

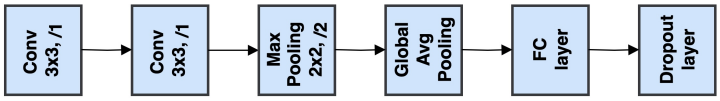
Branch Classifier Design

As depicted in Figure 3.10a, two branch classifiers are added at strategically chosen points: the first one after the `block2_pool` layer (roughly one-third through the model), and the second one immediately after the `block4_pool` layer (approximately two-thirds through). The design of each branch classifier is illustrated in Figure 3.10b and Figure 3.10c. The general principle for branch classifier design is that the number of added layers should not exceed the remaining layers, in order to avoid introducing additional latency. In our case, each branch classifier consists of one or more convolutional layers, followed by a max pooling layer, a global average pooling layer, a fully connected dense layer, and a dropout layer for regularization. For the shallower branch (Exit 1), two additional convolutional layers are included to enhance feature abstraction due to the limited depth of the backbone at that point. In contrast, the deeper branch (Exit 2) uses only one convolutional layer to balance prediction accuracy with computational efficiency.

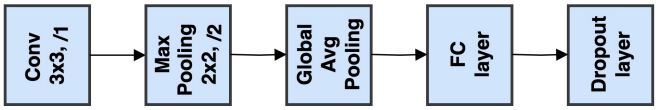
The use of global average pooling, instead of flattening the full feature map, significantly reduces the number of parameters and computational overhead, making



(a) Branch Classifier Positions in the Backbone



(b) Design of Branch Classifier 1



(c) Design of Branch Classifier 2

Figure 3.10: Overview of the branch classifiers. (a) shows the insertion points of early exits in the CNN backbone; (b) and (c) illustrate the architectural details of Branch Classifier 1 and Branch Classifier 2, respectively.

the branch classifiers lightweight and suitable for deployment on edge devices. This architectural choice ensures efficient early inference capabilities.

Training Method

Two primary training strategies are commonly used for models with early exits: *joint training* and *separate training*.

- **Joint Training:** Joint training simultaneously optimizes all classifiers—both the early exits and the final output. This is typically achieved by defining a loss function for each classifier and minimizing the weighted sum of the individual

losses. However, selecting appropriate values for these weights remains a challenging task and often requires empirical tuning. Poorly chosen weights may bias the model toward over-optimizing either the early exits or the final classifier, potentially degrading overall performance.

- **Separate Training:** An alternative approach is to adopt a two-stage training paradigm. In the first stage, the backbone CNN model is trained independently to convergence using standard training procedures. Once the backbone is pretrained, all of its parameters are frozen. In the second stage, the early exit classifiers are inserted into the network at selected intermediate layers and trained independently, while keeping the backbone weights fixed. This allows the early classifiers to adapt to the feature representations already learned by the backbone without perturbing its parameters.

  This strategy offers several benefits. It simplifies optimization by reducing parameter coupling, as early exits can be added to any pretrained model without requiring full retraining. However, it may lead to suboptimal global performance compared to joint training, since the backbone is not updated to accommodate the needs of the early classifiers.

- **Summary:** In this work, we adopt the separate training strategy for simplicity. After pretraining the full CNN model, we freeze the parameters and train the early exit branches individually using the same dataset. This ensures that the original model performance is preserved while enabling early predictions.

Fast Inference

Once the CNN model embedded with early exits is trained, it can be deployed for fast inference. The key idea is that if the classifier at an intermediate exit point is confident enough about the label of a test sample x, the sample can be exited early, thus avoiding further computation by the deeper layers of the network.

A common way to quantify classifier confidence is through confidence selection, typically using one of the following two methods:

- **Probability-based:** If the maximum predicted class probability exceeds a predefined threshold, the prediction is deemed reliable.

- **Entropy-based:** Measures the *uncertainty* of the predicted probability distribution. Entropy is defined as:

$$H(\hat{y}) = -\sum_{i=1}^{C} \hat{y}_i \log \hat{y}_i, \tag{3.2}$$

where $\hat{y}$ is the output probability vector after softmax, and C is the number of classes.

Entropy provides an intuitive measure of confidence:

- **Low entropy (close to 0):** Indicates high confidence—most of the probability mass is concentrated in a single class.
- **High entropy:** Indicates uncertainty—probability mass is spread across multiple classes, which often correlates with incorrect predictions.

In this work, we adopt the entropy-based approach to determine whether a sample should exit early. Since the gender classification task is binary and would otherwise require two separate thresholds to evaluate predictions for males and females, we opt to use a unified entropy-based threshold across all tasks to reduce complexity and maintain consistency.

To perform fast inference, we follow the process outlined in Algorithm 1.

Algorithm 1 Fast Inference with Early Exits

procedure FastInference(x, **T**)
  for $n = 1$ to N **do**
    $\hat{y}_{\text{gender}} \leftarrow f^{\text{gender}}_{\text{exit}_n}(x)$
    $\hat{y}_{\text{age}} \leftarrow f^{\text{age}}_{\text{exit}_n}(x)$
    $e_{\text{gender}} \leftarrow H(\hat{y}_{\text{gender}})$
    $e_{\text{age}} \leftarrow H(\hat{y}_{\text{age}})$
    if $e_{\text{gender}} < T_{n,1}$ **and** $e_{\text{age}} < T_{n,2}$ **then**
      return $\arg\max \hat{y}_{\text{gender}}, \arg\max \hat{y}_{\text{age}}$
    end if
  end for
  return predictions at final exit
end procedure

In this algorithm:

- x is the input sample;
- $f^{\text{gender}}_{\text{exit}_n}$ and $f^{\text{age}}_{\text{exit}_n}$ are the gender and age classifiers at the n-th early exit;
- $\hat{y}_{\text{gender}}$ and $\hat{y}_{\text{age}}$ are the predicted probability distributions for gender and age;
- $H(\cdot)$ denotes the entropy of the predicted distribution, used as a measure of confidence;

- $\mathbf{T} \in \mathbb{R}^{N\times 2}$ is a matrix of entropy thresholds, where $T_{n,1}$ and $T_{n,2}$ correspond to the thresholds for gender and age at the n-th exit.

At each early exit, the algorithm checks if both predicted distributions are confident enough (i.e., both entropy values fall below their respective thresholds). Only when this condition is satisfied, the model terminates early and returns the predicted labels. Otherwise, the inference proceeds to deeper layers, and the final prediction is made at the last exit.

The key challenge lies in selecting suitable thresholds that balance inference efficiency and prediction accuracy. Setting thresholds too high may lead to early exits with incorrect predictions, while overly conservative thresholds reduce efficiency gains.

To address this, we manually define thresholds for each early exit branch based on empirical evaluation. The principle of selecting the threshold is to allow as many samples as possible to obtain prediction results from the early exit, which areduce the additional delay caused by further reasoning. In other words, the selected threshold should be as large as possible, but at the same time, the prediction accuracy should not be significantly affected as much as possible. This depends on different service requirements. For example, in this experiment, we expect the age prediction accuracy to be around 60% or higher, and the gender prediction accuracy to be around 80% or higher. Figure 3.11 shows the relationship between prediction correctness and entropy for the First Early Exit (EE1) on 500 test samples. For gender classification, we observe that predictions with entropy below 0.6 are more likely to be correct. Similarly, for age classification, predictions with entropy below 0.8 tend to be more reliable.

Figure 3.12 illustrates similar results for the Second Early Exit (EE2). For gender classification, the 0.6 entropy threshold still appears effective, while for age classification, a slightly lower threshold of 0.8 better captures confident predictions.

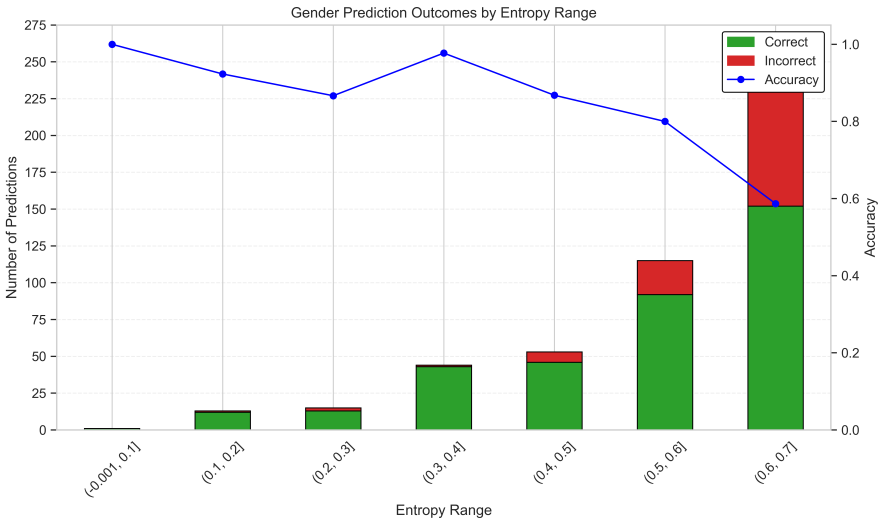
Based on these observations, we define the thresholds for our system as follows:

- **EE1:** Gender: 0.6, Age: 0.8
- **EE2:** Gender: 0.6, Age: 0.8

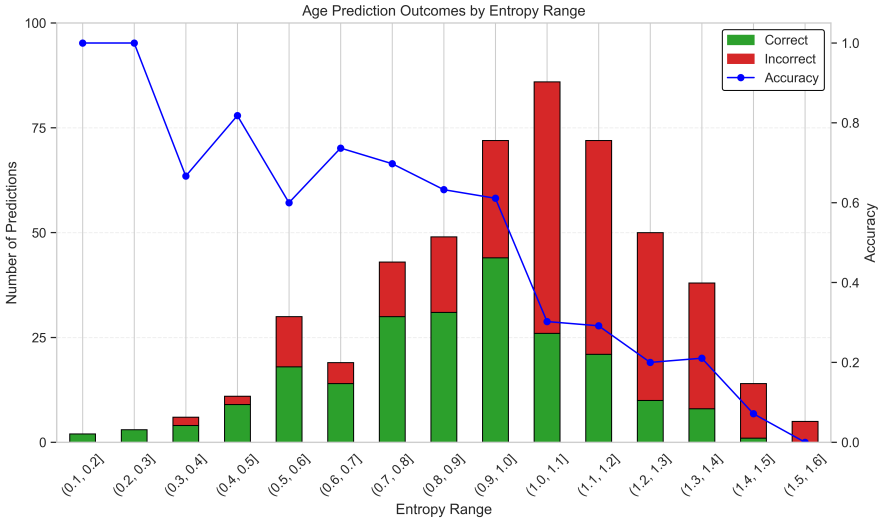
These thresholds are used throughout the remaining experiments to implement early exits without significantly sacrificing model accuracy.

Table 3.3: Output shapes and tensor sizes of each layer in the CNN model.

Layer	**Output Shape**	**Tensor Size**
input_1	(224, 224, 3)	150,528
block1_conv1	(224, 224, 64)	3,211,264
block1_conv2	(224, 224, 64)	3,211,264
block1_pool	(112, 112, 64)	802,816
block2_conv1	(112, 112, 128)	1,605,632
block2_conv2	(112, 112, 128)	1,605,632
block2_pool	(56, 56, 128)	401,408
block3_conv1	(56, 56, 256)	802,816
block3_conv2	(56, 56, 256)	802,816
block3_conv3	(56, 56, 256)	802,816
block3_pool	(28, 28, 256)	200,704
block4_conv1	(28, 28, 512)	401,408
block4_conv2	(28, 28, 512)	401,408
block4_conv3	(28, 28, 512)	401,408
block4_pool	(14, 14, 512)	100,352
block5_conv1	(14, 14, 512)	100,352
block5_conv2	(14, 14, 512)	100,352
block5_conv3	(14, 14, 512)	100,352
block5_pool	(7, 7, 512)	25,088
flatten	(25088, 1, 1)	25,088
dense128	(128, 1, 1)	128
dropoutShared	(128, 1, 1)	128
gender_output	(1, 1, 1)	1
age_output	(1, 1, 1)	1
age_output2	(9, 1, 1)	9

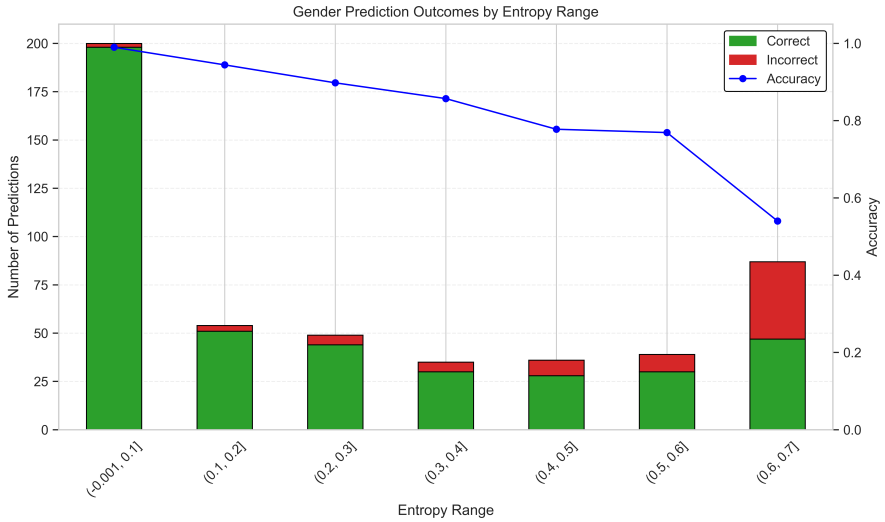


(a) Gender

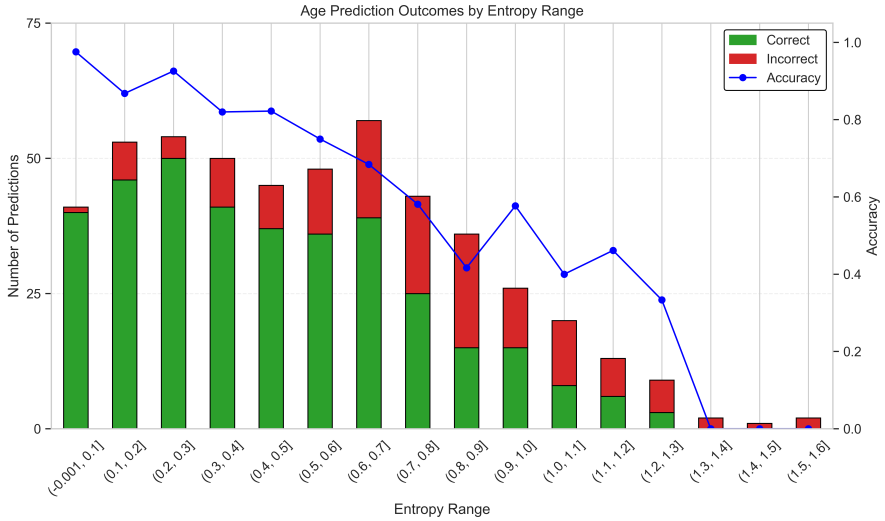


(b) Age

Figure 3.11: Entropy distributions at the EE1.



(a) Gender



(b) Age

Figure 3.12: Entropy distributions at the EE2.

4 Evaluation

In this chapter, we evaluate and compare the proposed deployment approaches from three key perspectives: latency, accuracy, and cost.

4.1 Experimental Setup

To simulate the behavior of user-generated traffic, we utilize a Python script executed on a local machine to generate HyperText Transfer Protocol (HTTP) requests directed toward the designated deployment endpoint. Each test consists of 100 samples, and we collect the results of all 100 requests for later comparison and analysis. The details of the experimental setup are as follows:

Client Side: The Python script used for sending requests runs on a MacBook Air with an 8-core Central Processing Unit (CPU) and 16 Gigabyte (GB) of memory, using Python 3.12. Since the evaluation focuses on 5G-enabled applications, we used a 5G network connection provided via an Askey O_2 HomeSpot 5G router (Model: RTL6319W-0112) with Vodafone 5G SIM card.

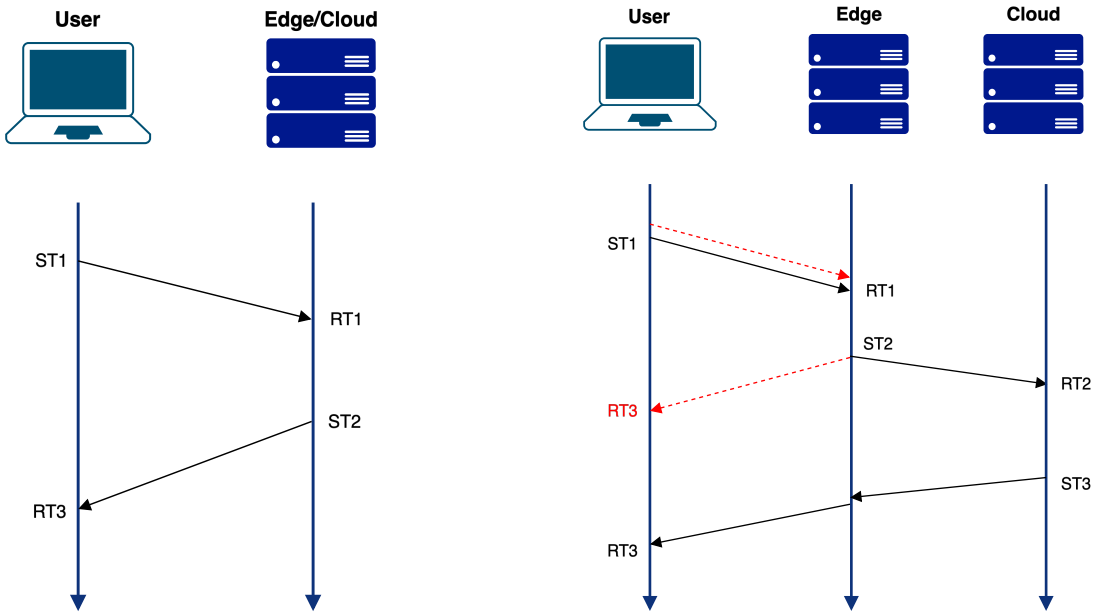
EC2 Instances: To ensure a fair comparison, EC2 instances in both the availability zone and the wavelength zone are configured identically:

- OS: Ubuntu Server 24.04 LTS (HVM)
- Architecture: 64-bit (x86)
- Instance Type: t3.medium (2 vCPUs, 4 GB memory)
- Storage: General Purpose Solid State Drive (SSD) (gp2), 24 GB

Lambda in Availability Zone:

- Memory: 4 GB
- Storage: 4 GB

Time Synchronization: To accurately measure latency and analyze its components, precise timekeeping is essential. Therefore, all backend servers in each deployment approach synchronize time using the Network Time Protocol (NTP).



(a) Single Architecture

(b) Hybrid Architecture

Figure 4.1: Illustration of timestamp propagation in single and hybrid architectures: In the hybrid architecture, if the prediction confidence level of the branch classifier in the early exits strategy is high enough, the result is returned directly to the user from the edge node, as illustrated by the red dashed arrow in the figure. Otherwise, the intermediate output generated at the edge node is forwarded to the cloud node for further inference, as indicated by the black solid arrow.

Timestamps: To accurately measure latency and analyze its components, a timestamp is recorded at each sending and receiving point of the request and response cycle. These timestamps are included in the payload of the HTTP requests and responses for further tracing.

As illustrated in Figure 4.1a, for deployments that adopt a simple edge architecture (EC2 instances in the Wavelength Zone) or a pure cloud architecture (EC2 or Lambda in the Availability Zone), four timestamps are recorded:

1. **ST1** (Sending Time 1): The time when the user device sends the request.
2. **RT1** (Receiving Time 1): The time when the edge or cloud server receives the request.

3. **ST2** (Sending Time 2): The time when the server sends the response after inference and computing.
4. **RT3** (Receiving Time 2): The time when the user device receives the response. We use RT3 instead of RT2 here to maintain consistency with the timestamp notation in hybrid architectures.

For hybrid architectures, as illustrated in Figure 4.1b, six timestamps are involved to capture the request-response cycle:

1. **ST1** (Sending Time 1): The time when the user device sends the request.
2. **RT1** (Receiving Time 1): The time when the edge server receives the request.
3. **ST2** (Sending Time 2): The time when the edge server either forwards the request to the cloud or sends the response back to the user (in early exit cases).
4. **RT2** (Receiving Time 2): The time when the cloud server receives the forwarded request.
5. **ST3** (Sending Time 3): The time when the cloud server sends the final response.
6. **RT3** (Receiving Time 3): The time when the user device receives the response.

In the *Split Computing* scenario, since the second stage of the inference model is hosted in the cloud, the edge server must forward the request to the cloud, and thus all six timestamps are consistently recorded.

However, in the *Early Exits* scenario, if the inference result from the edge server satisfies a predefined threshold condition, the edge directly returns the result to the user without forwarding it to the cloud. In this case, only four timestamps are involved: ST1, RT1, ST2, and RT3. If the threshold condition is not met, the intermediate output generated at the edge node is forwarded to the cloud node for further inference, and all six timestamps are recorded accordingly.

4.2 5G and WAN network performance

In this section, we present raw network performance data that is independent of any specific application deployment strategy.

To better understand real-world conditions, we conducted latency and bandwidth measurements in Munich using a Vodafone 5G network:

- **Location:** 48.1833326, 11.533497866

These measurements provide a baseline understanding of the underlying 5G and WAN network performance, without being affected by the deployment architecture of the application.

Table 4.1 summarizes the observed latency, upload bandwidth, and download bandwidth for the following communication paths:

(a) From a user device located in a Wavelength Zone to an EC2 instance in the Wavelength Zone.

(b) From a user device (not in the Wavelength Zone) to an EC2 instance in the Availability Zone.

(c) Between an EC2 instance in the Availability Zone and another in the Wavelength Zone.

We used the `ping` to measure round-trip latency and `iperf3` [47] to measure upload and download bandwidth. Each experiment was repeated 10 times, and for each metric, we report the minimum, average, maximum, and standard deviation values.

These measurements serve as important references when evaluating end-to-end application performance, especially in architectures where network performance can be a bottleneck.

Table 4.1: Network performance metrics.

Metrics		**Min.**	**Avg.**	**Max.**	**Std. dev.**
Latency between (ms)	*(a) Device and Wavelength*	19.61	24.16	47.48	6.211
	(b) Device and Availability	25.74	31.38	50.58	6.720
	(c) Wavelength and Availability	6.505	6.636	6.706	0.037
Upload Bandwidth (Mbits/s)	*(a) Device to Wavelength*	24.10	34.50	48.10	7.298
	(b) Device to Availability	8.380	30.43	53.70	13.97
	(c) Wavelength to Availability	1180	1356	1700	165.9
Download Bandwidth (Mbits/s)	*(a) From Wavelength to Device*	-	-	-	-
	(b) From Availability to Device	-	-	-	-
	(c) From Availability to Wavelength	208.0	239.7	273.0	21.29

From the data presented in Table 4.1, it can be observed that the WZ, owing to its geographical proximity to end devices and its integration with 5G networks, achieves both lower latency and higher bandwidth compared to the AZ. However, given the specific locations used in this experiment—Frankfurt for the AZ and Munich for the WZ—the performance advantage of the WZ is relatively minor.

Additionally, the network performance between the WZ and AZ demonstrates remarkably low latency and substantial bandwidth. The average upload bandwidth from WZ to AZ reaches approximately 1.35 GBps, even though the AZ is accessed via its public Internet Protocol (IP). Conversely, data transfer from AZ to WZ must use private IP communication, as the AZ does not reside within the designated 5G network. Despite this, the average download bandwidth from AZ to WZ still reaches 239.7 Mbps.

These high-performance metrics between WZ and AZ indicate that computational offloading from the Wavelength Zone to the Availability Zone is feasible, with the response being returned to the WZ efficiently and with minimal latency.

4.3 Latency

In Section 4.2, we present measurements that are application-independent and purely reflect the underlying network performance. In this section, we evaluate latency from the perspective of the specific application we deployed. These results are directly impacted by the type of AWS service used, the deployment strategy, and the nature of the model itself.

4.3.1 Latency Definition

As discussed in Section 4.1, we record multiple timestamps during each request and response cycle. These timestamps enable the computation of different latency components.

The **total latency** refers to the end-to-end delay from the moment the user device sends the request to the moment it receives the final response:

$$\text{Total Latency} = \texttt{RT3} - \texttt{ST1} \tag{4.1}$$

Compute latency on the server side represents the time taken from when the server receives the request to when it completes the processing and returns the response:

$$\text{Compute Latency 1} = \texttt{ST2} - \texttt{RT1} \tag{4.2}$$

$$\text{Compute Latency 2} = \texttt{ST3} - \texttt{RT2} \tag{4.3}$$

For deployments using a single compute location (i.e., non-hybrid architectures) or models with Early Exits where the fast inference result satisfies a predefined condition, Compute Latency 2 is zero.

Since communication latency involves timestamps from both ends and we can't guarantee precise synchronization under 1 ms across servers despite using NTP, we estimate **communication latency** as the residual delay after excluding compute times:

$$\text{communication Latency} = \text{Total Latency} - \text{Compute Latency 1} - \text{Compute Latency 2} \tag{4.4}$$

4.3.2 Overall Latency

Figure 4.2 illustrates the distribution of total latency across 100 test samples for each deployment strategy, with the median highlighted for each case. The results clearly demonstrate that deploying the application on an EC2 instance within the Wavelength Zone yields the lowest latency. Lambda and EC2 exhibit similar overall latency performance when deployed in AZ. For hybrid architectures such as SC and EE, which utilize both Wavelength and Availability Zones, the medium value of the latency falls between that of purely edge-based and purely cloud-based deployments. Among these hybrid strategies, EE achieves a lower average latency than SC, despite showing a wider latency distribution. This broader spread occurs because some samples exit early at the edge, achieving ultra-low latency, while others are forwarded to the cloud, incurring higher latency. Nevertheless, introducing Early Exits strategy still manages to reduce the average latency compared to simply Split Computing.

It is worth noting that serverless platforms such as AWS Lambda typically experience cold start issues, where the first request after a period of inactivity can incur significant latency. In our experiments, we observed that the end-to-end latency of requests affected by cold start was approximately 30 seconds. This overhead can be mitigated by enabling features like provisioned concurrency [48]. However, cold start scenarios were excluded from the latency analysis in this study.

4.3.3 Latency Composition

Figure 4.3 illustrates the average breakdown of latency into compute and communication components for each deployment. For hybrid deployments, compute latency is further split into `Compute1` (at edge) and `Compute2` (at cloud).

The results illustrated in the figure allow us to derive several key insights into the system's latency characteristics under different deployment scenarios:

- **Lambda exhibits lower compute latency than EC2 in the same region:** According to [49], Lambda allocates CPU resources proportionally to the configured memory. With 4 GB of memory, a Lambda function receives computing power equivalent to approximately 2.5 vCPUs, which is slightly more than the 2 vCPUs provided by

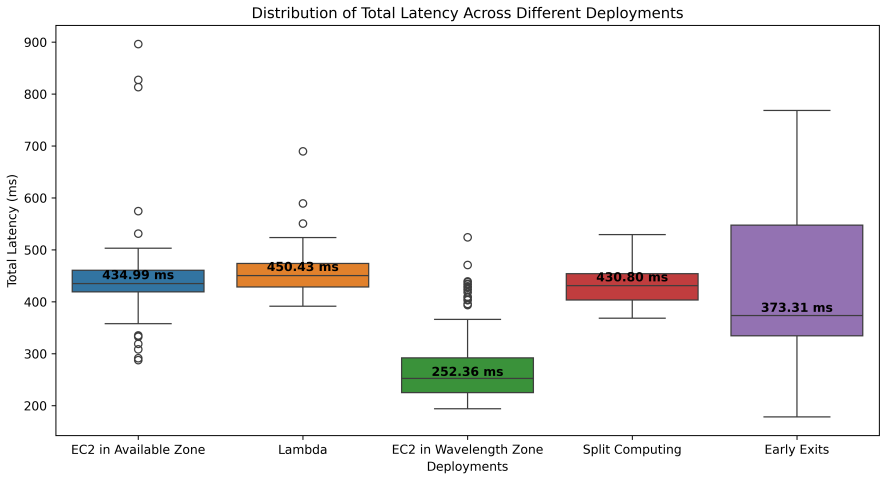


Figure 4.2: Total latency distribution.

an EC2 `t3.medium` instance. As a result, Lambda achieves lower compute latency compared to EC2 when both are deployed in the same region.

- **Lambda suffers from the longest communication latency:** Despite efficient computation, the network transfer to and from Lambda contributes significant delay.
- **EC2 in the Wavelength Zone processes faster than in the Availability Zone:** We observe that even when using EC2 instances with identical configurations, the compute time in the WZ is significantly lower compared to that in the AZ.
- **Early Exit's edge compute time (`Compute1`) is longer than in simply Split Computing:** This is because Early Exit includes two potential exit points. The front portion of the original model deployed at the edge is further divided into two sub-models. Consequently, a single inference operation includes multiple stages, and additional operations such as threshold evaluation and exit decision further increase edge-side compute time. However, since some samples exit early at the edge, the compute time on the cloud side (`Compute2`) is significantly reduced.

It is worth mentioning that since the EC2 instances used in this study are all of the "Shared" tenancy type, the underlying physical hardware may vary with each instance start. This can lead to differences in performance characteristics, such as CPU frequency. Additionally, because the instances are not deployed on dedicated hosts, the resource utilization of the underlying server can fluctuate over time, further contributing to variability in performance measurements. Nevertheless, the conclusions presented are based on common patterns observed through repeated measurements.

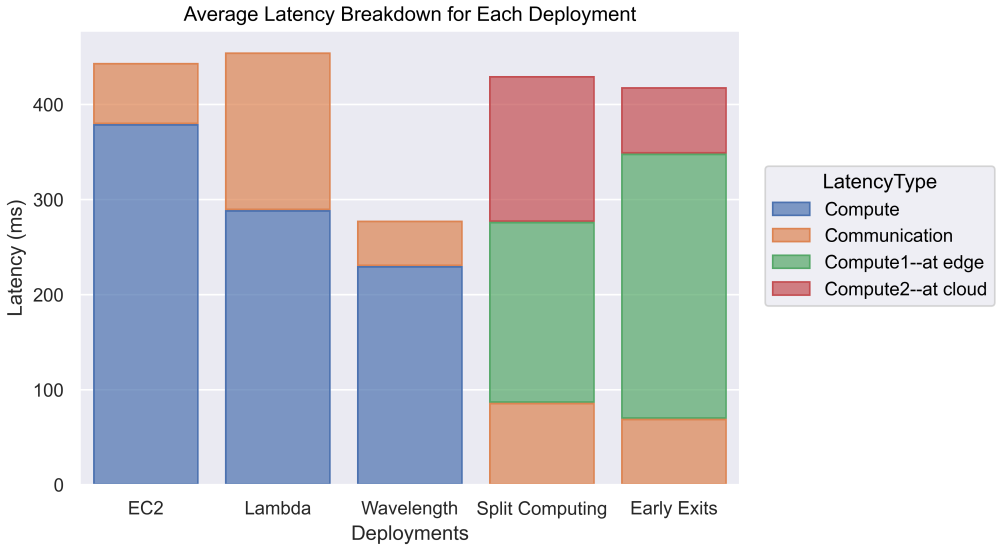


Figure 4.3: Latency breakdown.

4.4 Cost

Cost is a key factor that must be considered in practical production environments. Based on Amazon's pricing model [50], different services and configurations can significantly impact the overall cost. The experiment primarily involves two AWS services: `EC2` and `Lambda`.

For `EC2`, Table 4.2 lists the major cost-influencing factors along with the selected configurations.

- The base price for EC2 in the **Available Zone** is **$0.048 per hour**, with additional EBS storage charged at **$0.119 per GB-month** for General Purpose SSD (gp2). Since 24 GB of storage is configured, the monthly EBS cost is calculated as:

$$0.119 \times 24 = \textbf{\$2.856}$$

- The base price for EC2 in the **Wavelength Zone** is **$0.06 per hour**, with additional EBS storage charged at **$0.155 per GB-month** for gp2. Given the same 24 GB configuration, the monthly EBS cost is:

$$0.155 \times 24 = \textbf{\$3.72}$$

Table 4.3 outlines the key factors affecting `Lambda` pricing along with the chosen configuration. Each Lambda function invocation incurs three primary cost components:

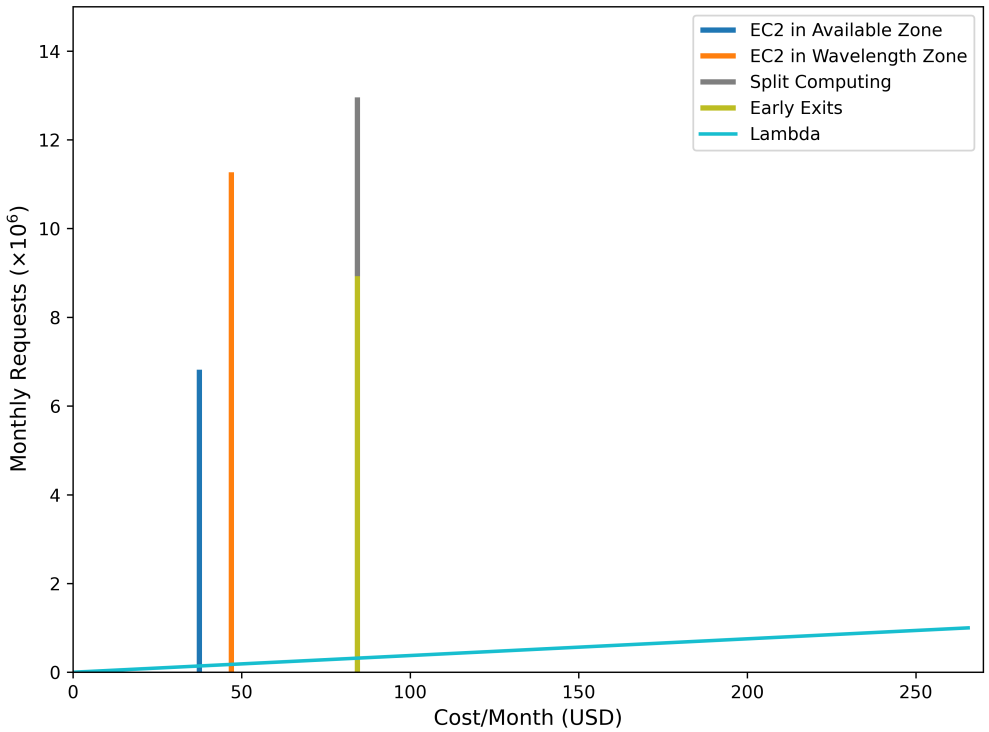


Figure 4.4: Cost comparison of different deployment strategies.

1. **Request Cost**: $0.20 per 1 million requests.
2. **Duration Cost**: Depends on the allocated memory size. In our case, we allocate **4096 MB**, corresponding to a cost of **$0.0000000667 per ms**, which is equivalent to **$0.004002 per minute**.
3. **Ephemeral Storage Cost**: Based on the allocated ephemeral storage and execution duration. We allocate **4096 MB** of storage, which results in a cost of **$0.0000008808 per ms**, or **$0.052848 per minute**.

From a production perspective, even during periods when no requests are received, the EC2 server must remain active to handle any possible incoming traffic, and thus incurs costs. Therefore, instead of calculating the cost per request, it is more meaningful to estimate the cost over a period of time, such as daily or monthly. Figure 4.4 illustrates the average monthly cost of different deployment strategies. Based on the per-request compute time discussed in Section 4.3.3, we roughly estimate the maximum number of requests that can be handled per month for each strategy.

Since AWS Lambda supports automatic scaling by design, we do not specify a fixed upper bound for its request rate. Instead, we present a linear cost-to-request

Table 4.2: EC2 configuration and prices [50].

Instance	EC2 in Available Zone	EC2 in Wavelength Zone
Instance Type	t3.medium	t3.medium
Location	Frankfurt	Munich
Tenancy Type	Shared Instances	Shared Instances
Operating System	Linux	Linux
Base Price	$0.048/hour	$0.060/hour
EBS Type	General Purpose SSD (gp2)	General Purpose SSD (gp2)
EBS Price	$0.119 per GB-month	$0.155 per GB-month

Table 4.3: Lambda Pricing Configuration [50].

Cost Component	Price
Requests	$0.20 per 1M requests
Memory Duration	$0.004002 per min (4096 MB)
Ephemeral Storage	$0.052848 per min (4096 MB)

relationship, reflecting the pay-per-request pricing model and its inherent scalability. The intersection point between the Lambda trend line and each of the other deployment strategies represents the break-even point. If the request rate remains below this point, the serverless approach is more cost-effective, as EC2 instances are underutilized during that time yet still incur full billing. However, if the request rate is between the break-even point and the respective maximum capacity, other deployment strategies become more economical due to better resource utilization.

Additionally, the EC2 deployment in the available zone is generally cheaper than in the Wavelength Zone. However, because of its longer compute time, its maximum served request number is significantly lower. For hybrid deployment strategies, both the cloud and edge EC2 instances are used, making the total cost the sum of both. Since each EC2 instance processes part of the workload, the overall capability is constrained by the slower part. In this case, Early Exits result in lower request rate than Split Computing, primarily due to higher compute latency at edge node for Early Exits.

It should be noted that all experiments were conducted in a test environment. Hence, all prices shown are based on on-demand pricing, which is typically the highest. In production environments, AWS offers various saving plans and reserved mode that can significantly reduce costs.

Additionally, for EC2-based deployments, data transfer costs must also be taken into account in real-world applications. Specifically, data transfer **into** Amazon EC2 from the Internet is free, and **out** from EC2 to the Internet incurs charges. AWS provides

Figure 4.5: Difficulty-based prediction path distribution in the EE-enhanced CNN model

100 GB of data transfer out to the Internet free of charge per month, aggregated across all AWS services and regions (excluding China and GovCloud). Beyond that, the first 10 TB per month is charged at $0.09 per GB. For data transfers between Availability Zones and the Wavelength Zone in Munich (supported by Vodafone), the cost is $0.02 per GB. While this cost was not included in the comparative analysis of deployment strategies in this study, it should be carefully considered in real-world applications based on actual data usage patterns.

4.5 Accuracy

This section discusses the impact of different deployment strategies on the model's prediction accuracy.

For deployments where the entire application model is hosted either in the cloud or at the edge, the prediction accuracy remains unchanged. This is because although the deployment location or service type differs, the model itself is fully migrated without modification. In the case of hybrid deployment strategies, the situation varies.

For *Split Computing*, although the original model is partitioned into two segments, the overall network architecture remains the same, so the prediction accuracy is not affected. However, the *Early Exits* strategy introduces a branch network at intermediate layers, which is trained to provide early outputs. As a result, the accuracy can be influenced by several factors, including the choice of intermediate layer, branch network

design, training strategy, and the threshold used during fast inference.

As shown in Table 4.4, the *Early Exits* strategy implements dynamic inference by introducing additional intermediate output points. The outputs are distributed across Early Exit 1, Early Exit 2, and the final output, accounting for 20%, 36%, and 44% of the predictions, respectively. This approach achieves the same gender prediction accuracy (85%) as other strategies. However, it leads to a decrease in age prediction accuracy from 61% to 57%.

Figures 4.6 and 4.7 illustrate the confusion matrices for gender and age prediction using the original CNN model and the CNN model with EE, respectively. Figure 4.5 presents the classification outcomes of the EE-enhanced CNN model based on the difficulty level of input images. In this context, *easy* refers to images where the model obtains sufficient confidence at EE branch 1 to produce a prediction; *medium* indicates sufficient confidence at EE branch 2; and *hard* denotes cases where inference must be forwarded to the cloud node for full computation.

From the results, it can be observed that the majority of age prediction errors in the EE model occur in the 60+ age group. In these cases, the model tends to assign overly high confidence scores at early exits, resulting in premature termination and incorrect predictions. This highlights a potential issue of overconfident inference on edge-case samples within the EE framework.

Additionally, it is evident that images of children are more frequently classified into the *hard* category, indicating that predicting gender and age for younger individuals remains particularly challenging. These observations suggest the need for a more diverse and balanced training dataset, especially with increased representation of edge demographic groups such as elderly individuals and children. Enhancing data diversity can help improve the model's generalization capabilities and robustness across different population segments.

Table 4.4: Prediction accuracy (%) comparison across deployment strategies

Deployment Strategy	**Gender Accuracy (%)**	**Age Accuracy (%)**	**Exit (%)**
EC2 in Available Zone	85	61	100
EC2 in Wavelength Zone	85	61	100
Split Computing	85	61	0, 100
Early Exits	85	57	20, 36, 44
Lambda	85	61	100

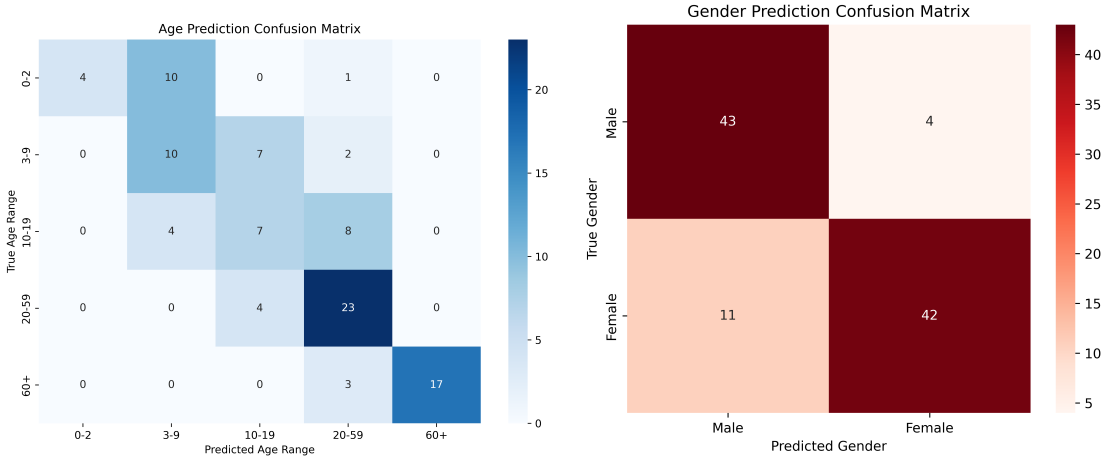


Figure 4.6: Confusion matrix of the original CNN model

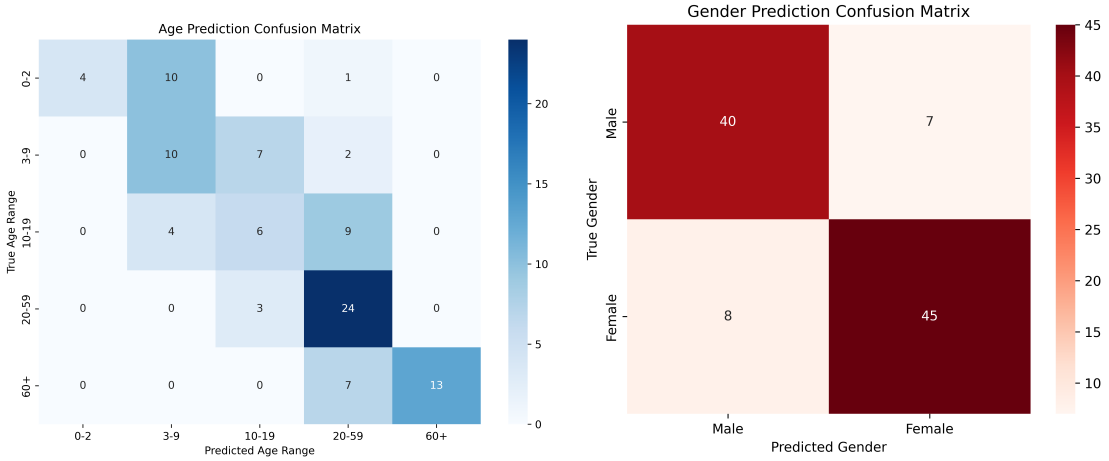


Figure 4.7: Confusion matrix of the CNN model with EE

5 Conclusion

This thesis presents the methodology and strategies for deploying a CNN-based human age and gender prediction system on the AWS cloud platform. We designed and implemented various deployment architectures, including cloud-based, edge-based, and edge-cloud hybrid configurations, as well as serverless and server-based computing paradigms. In particular, we explored the implementation of edge-cloud hybrid architectures using split computing and EE techniques.

The study conducted a comprehensive comparative analysis of these deployment strategies in terms of latency, accuracy, and cost. Experimental results show that, under similar computational capacity, edge-based deployment achieves the lowest latency. However, its cost is approximately 1.3 times that of using an equivalent EC2 instance in a cloud-only deployment. Lambda's cost scales linearly with the number of requests, making it more cost-effective for workloads with fewer than approximately 150,000 monthly requests; beyond this threshold, EC2 becomes more economical.

Although the edge-cloud hybrid deployment did not demonstrate clear advantages in this experiment, it shows the feasibility of implementing such architecture using split computing. Results in section 4.2 indicate that it can achieve ultra-low latency and high bandwidth within edge and cloud nodes. This suggests that for scenarios involving geographically distant cloud data centers or when high-performance EC2 instances are used, hybrid deployments can leverage the low latency of edge nodes and the high computing power of cloud nodes to achieve better overall performance. However, this comes at an estimated cost of approximately 2.3 times that of a cloud-only deployment.

For deployment strategies that do not alter the network architecture (excluding using EE techniques), the model's predictive accuracy remains unaffected. Compared to conventional split computing, incorporating EE can reduce latency by approximately 5%. However, the EE model trained in this work maintains gender prediction accuracy while causing a decline of approximately 4% in age prediction accuracy.

This work provides a solid foundation for further research. Future work can be extended in the following directions:

1. Beyond latency, accuracy, and cost, future comparisons could include factors such as scalability, security and deployment complexity.

2. In addition to EC2 and Lambda, other AWS services can be explored, such

as SageMaker [51], which is designed specifically for machine learning, and container-based services like ECS and EKS.

3. For EE strategies, further improvements can be made by designing more effective branch classifiers and adopting advanced training techniques such as joint optimization and knowledge distillation [52, 53].

Abbreviations

AI Artificial Intelligence

AWS Amazon Web Services

AZ Availability Zone

CNN Convolutional Neural Network

CV Computer Vision

DL Deep Learning

DNN Deep Neural Network

DNS Domain Name System

EBS Elastic Block Store

EC2 Elastic Compute Cloud

EE Early Exiting

ECS Elastic Container Service

EKS Elastic Kubernetes Service

FC Fully Connected

GPU Graphics Processing Unit

IMDB-WIKI Internet Movie Database and Wikipedia Dataset

IoT Internet of Things

LLM Large Language Model

MEC Multi-access Edge Computing

NLP Natural Language Processing

ROI Return on Investment

SC Split Computing

S3 Simple Storage Service

SVM Support Vector Machine

VPC Virtual Private Cloud

MAE Mean Absolute Error

CORS Cross-Origin Resource Sharing

IAM Identity and Access Management

PII Personally Identifiable Information

VGG Visual Geometry Group

WZ Wavelength Zone

EE1 First Early Exit

EE2 Second Early Exit

CPU Central Processing Unit

GB Gigabyte

HTTP HyperText Transfer Protocol

IP Internet Protocol

NTP Network Time Protocol

SSD Solid State Drive

CSPs Communications Service Providers

ISPs Internet Service Providers

List of Figures

List of Tables

Bibliography

[1] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, and G. Wang. "Recent Advances in Convolutional Neural Networks." In: *CoRR* abs/1512.07108 (2015). arXiv: 1512.07108.

[2] A. Torfi, R. A. Shirvani, Y. Keneshloo, N. Tavaf, and E. A. Fox. "Natural Language Processing Advancements By Deep Learning: A Survey." In: *CoRR* abs/2003.01200 (2020). arXiv: 2003.01200.

[3] Z. Zhang. "AI-Powered Intelligent Speech Processing: Evolution, Applications and Future Directions." In: *International Journal of Advanced Computer Science and Applications* 16.2 (2025). DOI: 10.14569/IJACSA.2025.0160291.

[4] Y. Matsubara, M. Levorato, and F. Restuccia. "Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges." In: *ACM Comput. Surv.* 55.5 (Dec. 2022). ISSN: 0360-0300. DOI: 10.1145/3527155.

[5] L. Zavodna, M. Ueberwimmer, and E. Frankus. "Barriers to the implementation of artificial intelligence in small and medium sized enterprises: Pilot study." In: *Journal of Economics and Management* 46 (Jan. 2024), pp. 331–352. DOI: 10.22367/jem.2024.46.13.

[6] InCabin. *InCabin USA 2024: Steering Into The Future – Keynote Announcement*. Accessed: 2025-05-10. Jan. 2024.

[7] Amazon Web Services. *AWS Wavelength Zone locations*. Accessed: 2025-05-10. n.d.

[8] A. Esteva, B. Kuprel, R. A. Novoa, et al. "Dermatologist-level classification of skin cancer with deep neural networks." In: *Nature* 542.7639 (2017), pp. 115–118.

[9] G. Litjens, T. Kooi, B. E. Bejnordi, et al. "A survey on deep learning in medical image analysis." In: *Medical image analysis* 42 (2017), pp. 60–88.

[10] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features." In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* IEEE. 2001.

[11] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. "DeepDriving: Learning affordance for direct perception in autonomous driving." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2722–2730.

[12] M. Pantic and L. J. Rothkrantz. "Toward an affect-sensitive multimodal human-computer interaction." In: *Proceedings of the IEEE* 91.9 (2003), pp. 1370–1390.

[13] W. Rawat and Z. Wang. "Deep convolutional neural networks for image classification: A comprehensive review." In: *Neural computation* 29.9 (2017), pp. 2352–2449.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet classification with deep convolutional neural networks." In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[16] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014).

[17] C. Szegedy, W. Liu, Y. Jia, et al. "Going deeper with convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[18] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[19] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. "Densely connected convolutional networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 4700–4708.

[20] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman. "VGGFace2: A Dataset for Recognising Faces across Pose and Age." In: *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*. 2018, pp. 67–74. DOI: 10.1109/FG.2018.00020.

[21] Amazon Web Services. *AWS Documentation Overview*. Accessed: 2025-05-10. n.d.

[22] F. Yu, D. Wang, L. Shangguan, M. Zhang, X. Tang, C. Liu, and X. Chen. *A Survey of Large-Scale Deep Learning Serving System Optimization: Challenges and Opportunities*. 2022. arXiv: 2111.14247 [cs.LG].

[23] J. Jagannath, N. Polosky, A. Jagannath, F. Restuccia, and T. Melodia. "Machine learning for wireless communications in the Internet of Things: A comprehensive survey." In: *Ad Hoc Networks* 93 (2019), p. 101913. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2019.101913.

[24] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh. "Distilled Split Deep Neural Networks for Edge-Assisted Real-Time Systems." In: HotEdgeVideo'19. Los Cabos, Mexico: Association for Computing Machinery, 2019, pp. 21–26. ISBN: 9781450369282. DOI: 10.1145/3349614.3356022.

[25] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge." In: *ACM SIGARCH Computer Architecture News* 45.1 (2017), pp. 615–629.

[26] H.-J. Jeong, I. Jeong, H.-J. Lee, and S.-M. Moon. "Computation offloading for machine learning web apps in the edge server environment." In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2018, pp. 1492–1499.

[27] S. Teerapittayanon, B. McDanel, and H.-T. Kung. "Branchynet: Fast inference via early exiting from deep neural networks." In: *2016 23rd international conference on pattern recognition (ICPR)*. IEEE. 2016, pp. 2464–2469.

[28] S. Teerapittayanon, B. McDanel, and H.-T. Kung. "Distributed deep neural networks over the cloud, the edge and end devices." In: *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE. 2017, pp. 328–339.

[29] D. Jiang and G. Liu. "An overview of 5G requirements." In: *5G Mobile Communications* (2016), pp. 3–26.

[30] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, and J. C. Zhang. "What Will 5G Be?" In: *IEEE Journal on Selected Areas in Communications* 32.6 (2014), pp. 1065–1082. DOI: 10.1109/JSAC.2014.2328098.

[31] N. Hassan, K.-L. A. Yau, and C. Wu. "Edge Computing in 5G: A Review." In: *IEEE Access* 7 (2019), pp. 127276–127289. DOI: 10.1109/ACCESS.2019.2938534.

[32] A. Narayanan, E. Ramadan, J. Carpenter, Q. Liu, Y. Liu, F. Qian, and Z.-L. Zhang. "A First Look at Commercial 5G Performance on Smartphones." In: *Proceedings of The Web Conference 2020*. WWW '20. Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 894–905. ISBN: 9781450370233. DOI: 10.1145/3366423.3380169.

[33] R. Bagai. "Comparative Analysis of AWS Model Deployment Services." In: *International Journal of Computer Trends and Technology* 72.5 (May 2024), pp. 102–110. ISSN: 2231-2803. DOI: 10.14445/22312803/ijctt-v72i5p113.

[34] D. Bayazitov, K. Kozhakhmet, A. Omirali, and R. Zhumaliyeva. "Leveraging Amazon Web Services for cloud storage and AI algorithm integration: A comprehensive analysis." In: *Applied Mathematics* 18.6 (2024), pp. 1235–1246.

[35] R. Doukha, S. A. Mahmoudi, M. Zbakh, and P. Manneback. "A comparative analysis of Convolution Neural Network models on Amazon Cloud Service." In: *AIIPCC 2022; The Third International Conference on Artificial Intelligence, Information Processing and Cloud Computing*. 2022, pp. 1–7.

[36] D. Chahal, M. Mishra, S. Palepu, and R. Singhal. "Performance and Cost Comparison of Cloud Services for Deep Learning Workload." In: *Companion of the ACM/SPEC International Conference on Performance Engineering*. ICPE '21. Virtual Event, France: Association for Computing Machinery, 2021, pp. 49–55. ISBN: 9781450383318. DOI: 10.1145/3447545.3451184.

[37] V. K. Roy, G. K. Roy, V. Thakur, N. Baliyan, N. Goyal, and Y. Chauhan. "Real Time Fruit Recognition and Calories Estimation Using CNN and Model Deployment on Amazon's AWS." In: *2023 3rd International Conference on Innovative Sustainable Computational Technologies (CISCT)*. 2023, pp. 1–6. DOI: 10.1109/CISCT57197.2023.10351261.

[38] K. Rao, G. Coviello, W.-P. Hsiung, and S. Chakradhar. "ECO: Edge-Cloud Optimization of 5G applications." In: *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 2021, pp. 649–659. DOI: 10.1109/CCGrid51090.2021.00078.

[39] K. Rao, G. Coviello, P. Benedetti, C. Giuseppe De Vita, G. Mellone, and S. Chakradhar. "Eco-llm: Llm-based edge cloud optimization." In: *Proceedings of the 2024 Workshop on AI For Systems*. 2024, pp. 7–12.

[40] S. S. Ogden and T. Guo. "Layercake: Efficient Inference Serving with Cloud and Mobile Resources." In: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 2023, pp. 191–202. DOI: 10.1109/CCGrid57682.2023.00027.

[41] Z. Zhang, Y. Song, and H. Qi. "Age Progression/Regression by Conditional Adversarial Autoencoder." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2017.

[42] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann. *MediaPipe: A Framework for Building Perception Pipelines*. 2019. arXiv: 1906.08172 [cs.DC].

[43] Y. Li, Y. Lin, Y. Wang, K. Ye, and C. Xu. "Serverless Computing: State-of-the-Art, Challenges and Opportunities." In: *IEEE Transactions on Services Computing* 16.2 (2023), pp. 1522–1539. DOI: 10.1109/TSC.2022.3166553.

[44] N. Hassan, K.-L. A. Yau, and C. Wu. "Edge Computing in 5G: A Review." In: *IEEE Access* 7 (2019), pp. 127276–127289. DOI: 10.1109/ACCESS.2019.2938534.

[45] H. Hussain, P. Tamizharasan, and C. Rahul. "Design possibilities and challenges of DNN models: a review on the perspective of end devices." In: *Artificial Intelligence Review* 55.7 (2022), pp. 5109–5167. DOI: 10.1007/s10462-022-10138-z.

[46] L. Lamport. *LaTeX : A Documentation Preparation System User's Guide and Reference Manual*. Addison-Wesley Professional, 1994.

[47] *iPerf - The ultimate speed test tool for TCP, UDP and SCTP*. https://iperf.fr/. Accessed: 2025-03-30. 2025.

[48] Amazon Web Services. *Provisioned Concurrency for AWS Lambda*. https://docs.aws.amazon.com/lambda/latest/dg/provisioned-concurrency.html. Accessed: 2025-06-15. 2024.

[49] Amazon Web Services. *Configuring memory and compute power for Lambda functions*. Accessed: 2025-03-25. 2024. URL: https://docs.aws.amazon.com/lambda/latest/dg/configuration-memory.html.

[50] Amazon Web Services. *AWS Pricing*. https://aws.amazon.com/pricing/. Accessed: 2025-03-30. 2025.

[51] Amazon Web Services. *Amazon SageMaker – Build, train, and deploy machine learning models at scale*. https://aws.amazon.com/de/sagemaker/. Accessed: 2025-06-13. 2024.

[52] H. Li, H. Zhang, X. Qi, R. Yang, and G. Huang. "Improved techniques for training adaptive deep networks." In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1891–1900.

[53] M. Phuong and C. H. Lampert. "Distillation-based training for multi-exit architectures." In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1355–1364.