PERFORMANCE ANALYSIS OF IMAGE CLASSIFICATION BETWEEN EDGE AND CLOUD COMPUTING

Natthasak Vechprasit

A Thesis Submitted in Partial Fulfillment of the Requirements
For the Degree of Master of Science Program in Information Technology
Graduate Studies
Thai-Nichi Institute of Technology
Academic Year 2024

Thesis Topic	Performance Analysis of Image Classification between Edge and Cloud Computing
By	Natthasak Vechprasit
Field of Study	Information Technology
Thesis Advisor	Dr. Pramuk Boonsieng

The Graduate Studies of Thai-Nichi Institute of Technology has been approved and accepted as partial fulfillment of the requirement for the Master's Degree.

………………………………………… Vice President for Academic Affairs
(Assoc. Prof. Dr. Warakorn Srichavengsup)
Month……… Date ……… Year ………

Thesis Committees

………………………………………… Chairperson
(Assoc. Prof. Dr. Annop Monsakul)

………………………………………… Committee
(Dr. Sarayut Nonsiri)

………………………………………… Committee
(Acting Sub Lt. Dr. Pichitchai Kamin)

………………………………………… Advisor
(Dr. Pramuk Boonsieng)

NATTHASAK VECHPRASIT : PERFORMANCE ANALYSIS OF IMAGE CLASSIFICATION BETWEEN EDGE AND CLOUD COMPUTING. ADVISOR : DR. PRAMUK BOONSIENG, 181 PP.

Image classification has become a major application in the AI era for connecting the physical and digital worlds. However, it requires intensive graphic processing power. IoT and Edge Computing have become popular approaches for distributing and offloading the workload from the cloud to the edge. Many edge devices are powered by an energy-efficient processor that can't execute intensive workloads, but some may be able to. In this thesis, we studied the overview of image classification implementation on edge and cloud computing and analyzed the performance to reveal the opportunity to implement a proper system architecture. The edge and cloud computing environments studied in this paper are a smartphone, a personal computer, and a cloud GPU instance with sample applications to simulate real-world scenarios. The performance is based on the datasets and the processing environment comprising three factors: ML runtime, hardware, and network. Resulting in six factors: inference time, end-to-end execution time (including network delays), accuracy, confidence score, resource usage, and data transfer.

Graduate Studies	Student's signature ……………………
Field of Information Technology	Advisor's signature …………………..
Academic Year 2024	

Acknowledgement

I would like to express gratitude to the thesis committee, Assoc. Prof. Dr Annop Monsakul, Dr. Pramuk Boonsieng, Dr. Sarayut Nonsiri, and Acting Sub Lt. Pichitchat Kam-in for their helpful guidance and constructive criticism that significantly influenced my thesis. Without their insightful comments, this thesis would not have been possible.

Furthermore, I would like to express my deepest appreciation to my family and my partner for their love and support, which motivated me to complete my master's degree.

Additionally, I would like to thank the IT faculty staff and TNI staff for their facilitation and for providing the necessary resources to carry out this research.

Finally, I hope this thesis will be useful to future researchers and contribute to the progress of technology and science.

Natthasak Vechprasit

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

List of Tables

List of Figures

List of Figures (Continued)

Chapter 1
Introduction

1.1 Background

Image classification is the adoption of Deep Learning techniques to enable computers to learn and analyze images automatically. It plays an important role in connecting the computer with the physical world. It advances the development of new knowledge, techniques, and methods in many areas such as agriculture [1,2,3], industrial, manufacturing, security [4], smart things [4,5], healthcare [6,7,8], and retail [9,10].

Image classification processes require significant processing power, and that should be a specific processing unit, such as a sufficient GPU (Graphics Processing Unit).

Most applications were designed to execute data processing on the cloud, including the image classification process. Thus, this leads to the expensive subscription cost for GPU instances or computer vision API on the public cloud, and the cost will increase on demand. The network bandwidth is also needed for uploading the image, which is larger than the text, even though the image was pre-processed by resizing and compression. By using cloud computing to process, although cloud computing is a high-availability cluster, the budget of the application owner may be limited, so the resources that the application owner acquired may also be limited, resulting in a slow computation process, or causing a bottleneck or interruption when high-demand throughput occurs. And, in some use cases, the image should not be uploaded to the internet because of privacy concerns.

With the emergence of Edge Computing, offloading the image classification process from the cloud to the edge devices is a viable idea that mitigates the budget needs, reduces cloud computing workloads, reduces network bandwidth, and privacy oriented. Thus, resulting in lower running costs or lower subscription costs, and more privacy for the end user.

Nowadays, there is various energy-efficient hardware that can be used as an edge device. Examples of them are smartphones, tablets, and single-board computers.

Flagship or high-end devices integrated a GPU on the SoC (System on a Chip), which is, by the provided technical specification, the number of GPU cores seems to have the potential for processing image classification. For instance, the iPhone 12 [11] and iPhone 12 Pro [12] series have 4 GPU cores, an iPad Pro 11” 3rd generation [13] has 8 GPU cores, and an Nvidia Jetson Orin Nano [14] has 1024 CUDA cores. The number of cores in mobile devices tends to increase in future generations, as seen in the iPhone 13 Pro [15] and iPhone 14 Pro [16] series.

Even though the technical specifications of mobile devices reveal their potential computing resources, the study of performance between processing on mobile devices, single-board computers, and cloud environments is needed to reveal the practical difference in terms of inference time, accuracy, network data transfer, and resource utilization.

In this research, the performance of the image classification process on viable devices was studied, and the practical performance was evaluated. The result will reveal the future trends in using edge devices to perform image classification in software architecture design.

The performance evaluation method was proposed. It consists of the construction of edge computing and cloud computing environments. Image Classification model training, optimization for each environment, deployment of the predictive model to each environment, and result gathering and analysis.

The environments are separated into two categories (i.e., edge and cloud). For edge computing environments, an Apple iPhone 15 Pro series [17] with an Apple A17 Pro SoC was chosen as a representative of edge devices to be evaluated, and a computer with an NVIDIA GeForce RTX 3070 GPU [18] was chosen as a representative of edge servers to be evaluated. For cloud computing environments, a public cloud GPU instance, EC2 G5.xlarge Instance, from Amazon Web Services (AWS) [19] with NVIDIA A10G Tensor Core GPU [20] was chosen to be evaluated. The details of the environments are explained in Chapter 3.3 Environment Setup.

The model training will be based on the deployment environment. For Apple mobile devices, Apple Create ML [21,22] will be used for generating an image classification model, and Apple Core ML [23,24] will be used for on-device implementation. For Linux-based operating system environments (i.e., edge server and

GPU instance), TensorFlow [25,26,27] and Keras [28] will be used for generating an image classification model, then the model will be converted to TensorRT [29] to run on the environment.

The performance outcomes were generated by running test scripts repeatedly, and the results and performance value metered by IDEs (Integrated Development Environments) were gathered. Then, the results were statistically analyzed to summarize the findings.

1.2 Objectives

1.2.1 To study the overview of image classification implementation on edge and cloud environments.

1.2.2 To study, visualize, and analyze the result of the image classification performance on edge and cloud environments.

1.3 Contributions

1.3.1 Overview of image classification implementation on edge and cloud environments.

1.3.2 Performance results of image classification on edge and cloud environments.

1.4 Scope

1.4.1 Data for image classification model training and testing

Any labeled or classified photos

1.4.2 Technical

To study the practical way of implementing image classification applications, the techniques below are applied.

1) Create ML [21,22] for image classifier model training on macOS and Core ML [23,24] for running image classifier model deployment on iOS

2) TensorFlow [25,26,27] and Keras [28] for image classifier model training on a Linux-based OS.

3) Sample of edge-cloud application software.

1.4.3 Hardware

1) Edge Device – A GPU-integrated smartphone.

2) Edge Server – A computer with an integrated or discrete GPU located in the edge network.

3) Cloud Server – A computer with a discrete GPU located in a remote network.

1.4.4 Computer Network

FTTX broadband or 5G cellular network connection

1.5 Research Questions

What is the practical difference between processing at the edge and the cloud in terms of:

1) Inference Time
2) End-to-End Execution Time
3) Accuracy
4) Network Data Transfer
5) Resource Utilization
6) Confidence Score

1.6 Hypothesis

Processing image classification on edge devices or edge servers is faster than sending an image over the internet to process on the cloud because of the elimination or reduction of network delays, together with the power of GPUs on edge devices.

1.7 Definitions

1.7.1 Edge Computing

Edge Computing refers to the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of cloud services, and upstream data on behalf of IoT services [30].

1.7.2 Graphics Processing Unit (GPU)

A Graphics Processing Unit (GPU) is a specialized electronic circuit designed to manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device. GPUs are used in embedded systems, mobile phones, personal computers, workstations, and game consoles [31].

1.7.3 System on a chip (SoC)

A system on a chip (SoC) is an integrated circuit that combines most or all key components of a computer or electronic system onto a single microchip. Typically, an SoC includes a central processing unit (CPU) with memory, input/output, and data storage control functions, along with optional features like a graphics processing unit (GPU), Wi-Fi connectivity, and radio frequency processing. This high level of integration minimizes the need for separate, discrete components, thereby enhancing power efficiency and simplifying device design. [32]

Chapter 2
Literature Review

Theories, concepts, and related works used as a principle of this research have been grouped into five categories listed below.

2.1 Edge Computing

2.2 Image Classification

2.3 ML Kit and Related Libraries

2.4 Related Works

2.5 Related Theories

2.1 Edge Computing

2.1.1 Concept of Edge Computing

Shi et al. [30] described the concept of Edge Computing as the enabling of technologies allowing computation to be performed at the edge of the network. such as processing on a smartphone which is the edge between the body and the cloud. For example, a smart phone is the edge between body things and cloud, a gateway in a smart home is the edge between home things and cloud.

Edge can perform computing offloading, data storage, caching and processing, as well as distribute requests and delivery services from cloud to user.

Shi et al. [30] also provided an illustration of the edge computing paradigm as shown in Figure 2.1.

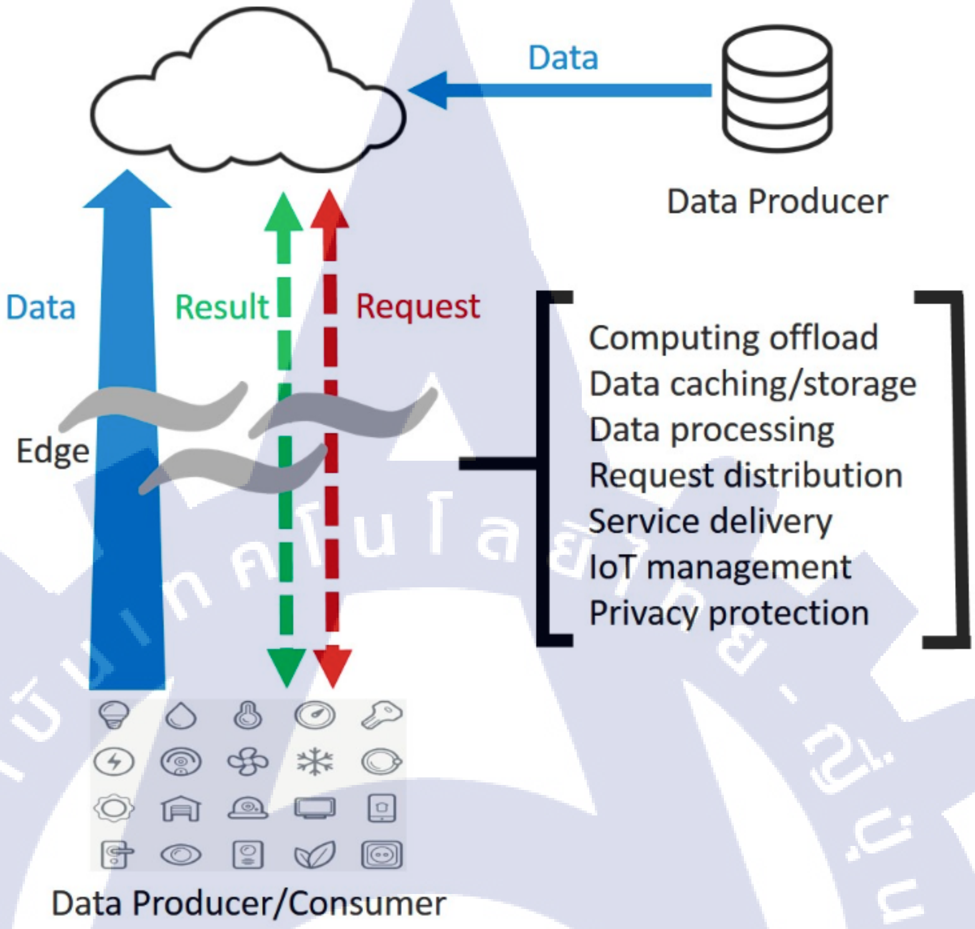


Figure 2.1 Edge Computing Paradigm [30]

2.1.2 Use Cases of Edge Computing

Shi et al. [30] reveal that cloud offloading, video analytics, smart home, smart city and collaborative edge could shine.

J. Chen and X. Ran. [33] proposed that computer vision, natural language processing, network functions, the internet of things, virtual reality, and augmented reality are examples of applications where deep learning on edge devices is useful.

2.1.3 Benefits of Edge Computing

The benefits of Edge Computing mostly involve the reduction of cloud computing workloads, the reduction of network bandwidth, which will save the cost of operating the workload on the cloud.

2.2 Image Classification

Image classification is a computer vision task that involves assigning a label or a category to an image based on its content [34]. The goal of image classification is to teach a machine learning model to recognize and classify images accurately and automatically.

Image classification typically involves a training phase, during which a deep learning model is trained on a large dataset of labeled images. The model learns to identify patterns and features in the images that are associated with different classes or categories. For example, a model might learn to recognize the shape of a car, the texture of fur, or the color of a flower.

Once the model is trained, it can be used to classify new, unseen images. During the classification phase, the model takes an input image and produces a probability distribution over the possible classes or categories. The class with the highest probability is then assigned as the label for the image.

Image classification has many practical applications, including object recognition, face detection, medical image analysis, and self-driving cars. It is a fundamental task in computer vision and has been the subject of much research and development in recent years.

2.3 ML Kit and Related Libraries

2.3.1 CoreML

Apple's Core ML [23,24] is a framework for developing machine learning models and integrating them into iOS, macOS, watchOS, and tvOS applications. It allows developers to easily build and deploy machine learning models on Apple devices, taking advantage of the device's hardware acceleration to run models quickly and efficiently.

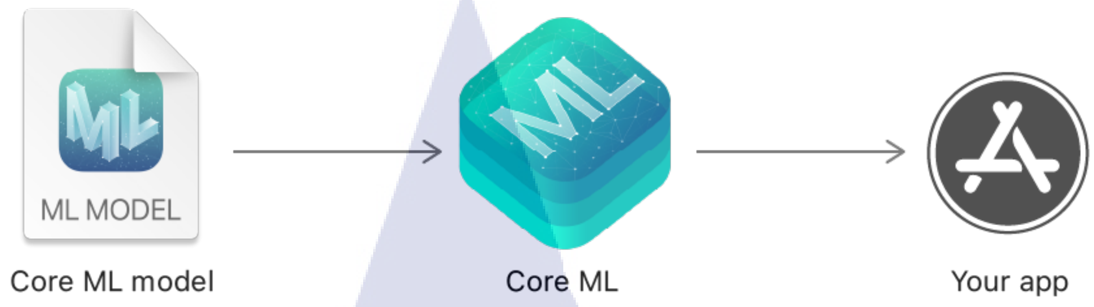


Figure 2.2 Apple Core ML pipeline with the Core ML logo in the middle

Core ML provides a simple and unified API for accessing pre-trained machine learning models from popular machine learning libraries such as TensorFlow and Keras, as well as custom models created using tools like Create ML. This allows developers to quickly integrate machine learning capabilities into their apps without having to worry about the underlying implementation details.

Core ML supports a wide range of machine learning tasks, including image and text analysis, natural language processing, and even custom tasks through the use of custom layers. It also includes features like model quantization and compression, which allow models to be optimized for deployment on mobile devices with limited resources.

Overall, Core ML makes it easier for developers to incorporate machine learning into their apps, making it possible to build more powerful and engaging experiences for users.

2.3.2 Create ML

Apple's Create ML [21,22] is a framework for developing custom machine learning models for use in iOS, macOS, watchOS, and tvOS applications. It is designed to be easy to use, even for developers without extensive experience in machine learning.

Create ML provides a simple and intuitive interface for training machine learning models using a variety of data types, including images, text, tabular data, and more. It includes pre-built models for common tasks like image classification and object detection, as well as the ability to create custom models using a drag-and-drop interface.

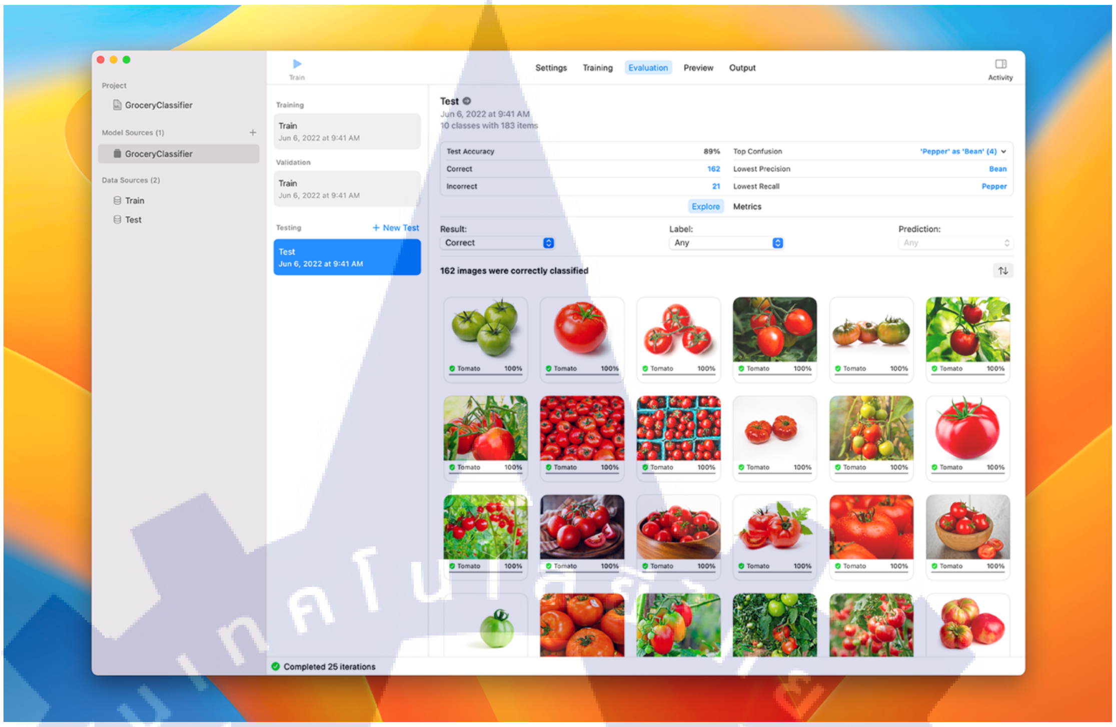
Figure 2.3 Create ML logo

Create ML provides a simple and intuitive interface for training machine learning models using a variety of data types, including images, text, tabular data, and more. It includes pre-built models for common tasks like image classification and object detection, as well as the ability to create custom models using a drag-and-drop interface.

With Create ML, developers can quickly build and train models on their own data sets, without the need for extensive knowledge of machine learning algorithms or programming languages like Python. The framework also includes features like automated machine learning, which can help developers find the best model architecture and hyperparameters for their data set.

Once a model is trained in Create ML, it can be exported as a Core ML model for use in iOS, macOS, watchOS, and tvOS applications. This integration with Core ML allows developers to easily incorporate custom machine learning models into their apps, taking advantage of the device's hardware acceleration for fast and efficient inference.

Overall, Create ML makes it easier for developers to create and deploy custom machine learning models, enabling them to build more intelligent and engaging apps for Apple's platforms.



Test
Jun 6, 2022 at 9:41 AM
10 classes with 183 items

Test Accuracy	89%	Top Confusion	'Pepper' as 'Bean' (4)
Correct	162	Lowest Precision	Bean
Incorrect	21	Lowest Recall	Pepper

Result: Correct
Label: Any
Prediction: Any

162 images were correctly classified

Figure 2.4 User Interface of Create ML application on macOS Ventura [21]

2.3.3 TensorFlow

TensorFlow [25,26,27] is an open-source framework for building and deploying machine learning models. Developed by Google, it was initially released in 2015 and has since become one of the most popular machine learning frameworks used by developers and researchers worldwide.

TensorFlow allows users to build and train machine learning models for a wide range of tasks, including image and speech recognition, natural language processing, and more. It uses a data flow graph to represent the computations in a model, allowing for efficient parallel processing and distributed computing.

One of the key features of TensorFlow is its flexibility. It allows users to build models using a variety of programming languages, including Python, C++, and Java, and supports a wide range of hardware, from mobile devices to high-performance computing clusters. It also includes a vast library of pre-built models and tools, including TensorFlow Hub, which provides access to a large collection of pre-trained models.

TensorFlow also includes LiteRT [35] (formerly known as TensorFlow Lite), which allows models to be deployed on mobile and embedded devices with limited resources.

Overall, TensorFlow provides a powerful and flexible platform for building and deploying machine learning models, and has become a key tool in the machine learning community.

Figure 2.5 TensorFlow Logo

2.3.4 Keras

Keras [28] is an open-source high-level neural network API, written in Python and capable of running on top of popular deep learning libraries such as TensorFlow, Theano, and CNTK. It was developed with a focus on enabling fast experimentation and prototyping of deep learning models.

Keras provides a simple and user-friendly interface for building and training deep learning models, allowing developers to quickly prototype and iterate on different architectures and hyperparameters. It supports a wide range of neural network architectures, including convolutional networks, recurrent networks, and combinations of the two.

One of the key features of Keras is its modularity, which allows users to easily mix and match different layers, loss functions, and optimizers to build custom models. It also includes a large library of pre-built models and tools, including the Keras Applications module, which provides access to a collection of pre-trained models.

Keras is widely used in both industry and academia, and has become a popular choice for building deep learning models due to its ease of use, flexibility, and extensive community support.

Overall, Keras provides a powerful and user-friendly platform for building and training deep learning models, and has played a significant role in making deep learning more accessible to a wider audience.

Figure 2.6 Keras Logo

2.3.5 TensorRT

TensorRT [29] is a deep learning inference optimizer and runtime library developed by NVIDIA. It is designed to improve the performance and efficiency of deep learning inference on NVIDIA GPUs and other accelerators.

TensorRT works by optimizing the computation graph of a trained deep learning model for execution on NVIDIA GPUs. It does this by performing a variety of optimizations, including layer fusion, precision calibration, and kernel auto-tuning, to reduce the memory footprint and computational workload of the model.

By optimizing the model for execution on NVIDIA GPUs, TensorRT is able to significantly accelerate the inference process and improve the throughput and latency of deep learning applications. It supports a wide range of deep learning frameworks, including TensorFlow, PyTorch, and ONNX, and is compatible with both cloud and edge deployments.

TensorRT is widely used in a variety of industries, including automotive, healthcare, and finance, to accelerate and optimize deep learning inference. It has become a key tool in the NVIDIA ecosystem and has helped to drive the adoption of deep learning in industry and academia.

Overall, TensorRT provides a powerful and efficient platform for optimizing and accelerating deep learning inference on NVIDIA GPUs, and has become an essential tool for many deep learning applications.

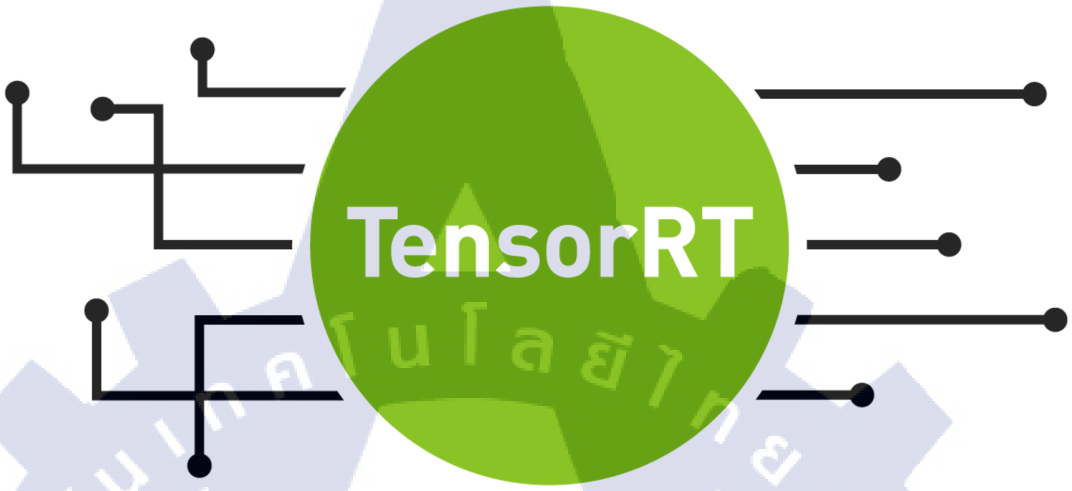


Figure 2.7 NVIDIA TensorRT Logo

2.4 Related Works

Table 2.1 show summarizes research related to the performance analysis, comparison, evaluation, benchmarks, and applications of image classification on edge computing.

Table 2.1 Related Works

No.	Authors	Title
1	N. Monburinon, et al. [1]	A Novel Hierarchical Edge Computing Solution Based on Deep Learning for Distributed Image Recognition in IoT Systems
2	I. Zualkernan, et al. [2]	An IoT System Using Deep Learning to Classify Camera Trap Images on the Edge

Table 2.1 Related Works (Continued)

No.	Authors	Title
3	E. Charteros and I. Koutsopoulos. [6]	Edge Computing for Having an Edge on Cancer Treatment: A Mobile App for Breast Image Analysis
4	S. R. Reza, et al. [36]	Inference Performance Comparison of Convolutional Neural Networks on Edge Devices
5	A. Ignatov, et al. [37]	AI Benchmark: All About Deep Learning on Smartphones in 2019

N. Monburinon et al. [1] proposed the topic "A Novel Hierarchical Edge Computing Solution Based on Deep Learning for Distributed Image Recognition in IoT Systems". They proposed a system for detecting animals that intrude into the agriculture field (Figure 2.8). They localized the training set for each geographical area (Figure 2.9). Then, they deploy the trained model on the edge server (Figure 2.8 and Figure 2.10), which is a Raspberry Pi 3 Model B (Table 2.2).

The prediction results generated from the edge server are federated to the higher-level edge server or to the cloud. The result is their proposed edge computing system performs faster than cloud computing in most cases and has greater accuracy.

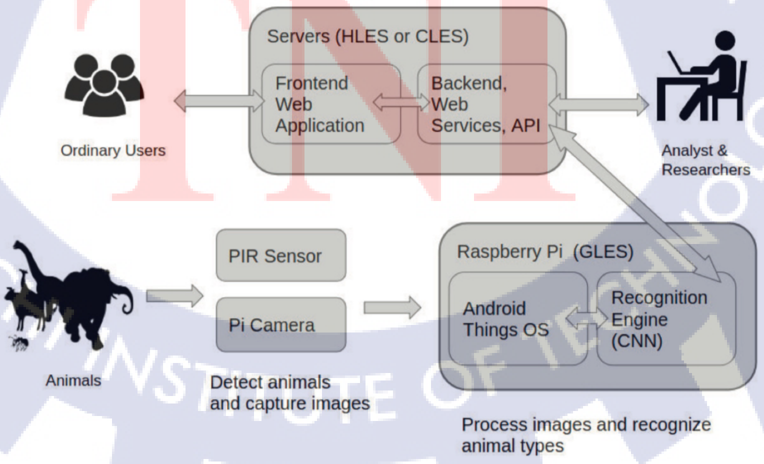


Figure 2.8 IoT Kakashi (Smart Scarecrow) System Overview [1].

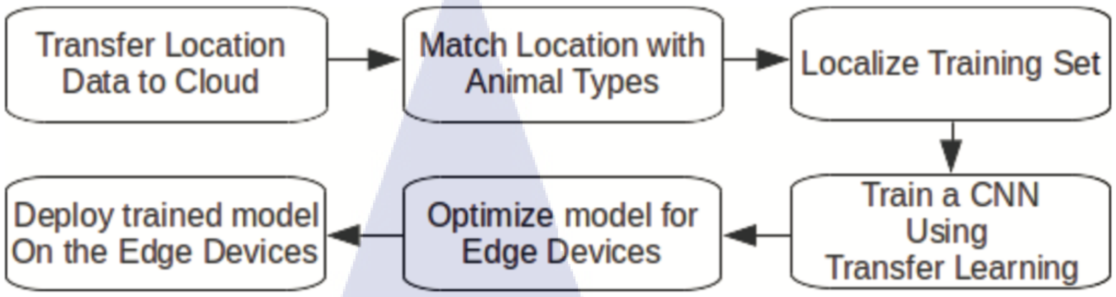

Transfer Location Data to Cloud
Match Location with Animal Types
Localize Training Set
Train a CNN Using Transfer Learning
Optimize model for Edge Devices
Deploy trained model On the Edge Devices

Figure 2.9 Deployment Environment Aware Learning (DEAL) Process for CNN-based model [1].

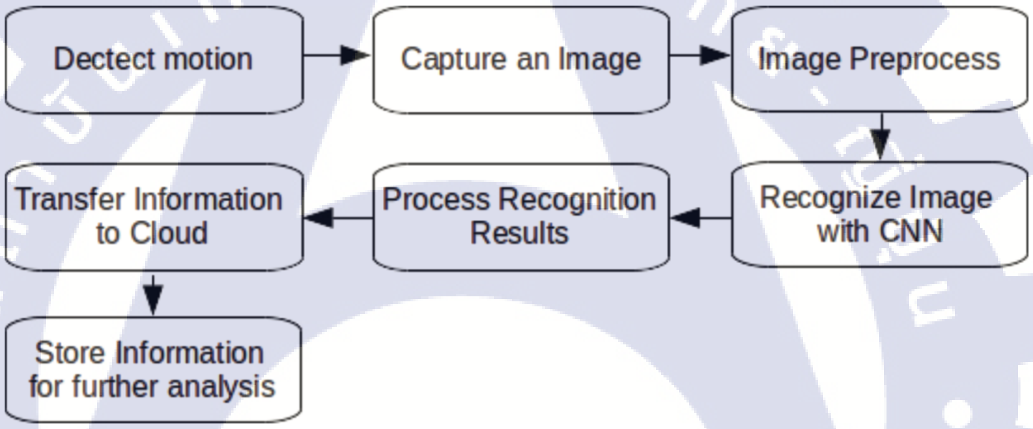

Dectect motion
Capture an Image
Image Preprocess
Recognize Image with CNN
Process Recognition Results
Transfer Information to Cloud
Store Information for further analysis

Figure 2.10 Animal Recognition Process Overview [1]

Table 2.2 Edge Device Information [1]

Raspberry Pi 3 Model B	
CPU	1.2 GHz Quad-Core ARM Cortex A53
GPU	Broadcom VideoCore IV @ 400 MHz
Memory	1 GB LPDDR2-800 SDRAM
Network	10/100 Mbps Ethernet 802.11n Wireless LAN

Table 2.2 Edge Device Information [1] (Continued)

Raspberry Pi 3 Model B	
OS	Android Things 1.0

The results shown in Table 2.3 and Table 2.4 show that the researcher proposed method have of 0.319 seconds as a minimum evaluation time of 0.319 seconds and 1.640 seconds as a maximum evaluation time whereas 0.723-1.284 seconds as a minimum time among cloud providers and 1.377-2.577 seconds as a maximum time among cloud providers. The execution time of their proposed method is 0.962 seconds as a minimum and 3.039 as a maximum whereas among cloud providers, 2.147 is a minimum, and 8.446 is the lowest maximum. While their proposed method is maintaining an accuracy of 90% in Top-1 accuracy and 96% in Top-3 accuracy whereas general purpose image recognition APIs have the highest of 57% in Top-1 accuracy and 77% in Top-3 accuracy as shown in Table 2.5.

The results in Table 2.6 show that the researchers proposed model uses only 4.3 kb/s whereas at least about 70 kb/s for processing on the public cloud. The model uses less bandwidth because it did not send the image data over the internet. And the concern of using a low-power consumption device for processing image recognition is eliminated in this scenario due to the results that show the execution time is significantly less than sending an image over the internet to process on the cloud.

Table 2.3 Proposed System Evaluation Time Compared to Cloud-Based System [1]

Model	**Min. Evaluation Time**	**Max. Evaluation Time**
Proposed System	0.3190	1.6400
Google Vision	0.7233	1.3768
AWS Rekognition	1.2835	2.5770
Clarifai	0.9491	1.8757

Table 2.4 Proposed System Execution Time Compared to Cloud Based System [1]

Model	Min. Execution Time	Max. Execution Time
Proposed System	0.962	3.039
Google Vision	2.456	12.571
AWS Rekognition	2.596	8.446
Clarifai	2.147	8.803

Table 2.5 Proposed System Top-1 and Top-3 Accuracy Compared to Cloud-Based System [1]

Model	Top-1 accuracy	Top-3 accuracy
Proposed System	0.9	0.96
Google Vision	0.57	0.77
AWS Rekognition	0	0.7
Clarifai	0.13	0.2

Table 2.6 Bandwidth Usage Comparison [1]

Model	Bandwidth Usage (KiB/s)
Proposed System	4.3
Google Vision	71.8
AWS Rekognition	70.7
Clarifai	72.5

I. Zualkernan, et al. [2] proposed the topic "An IoT System Using Deep Learning to Classify Camera Trap Images on the Edge", they described it as "an IoT architecture that uses deep learning on edge devices to convey animal classification results to a mobile app using the LoRaWAN, a low-power wide area network" (Figure 2.11). They review various applications on CNNs that can be used for animal image classification. The models InceptionV3, MobileNetV2, ResNet18, EfficientNetB1, DenseNet121, Xception were evaluated. They also evaluate the model and hardware power consumption and latency, their results show that in the model that utilizes GPU, despite twice the power consumption, the inference time is significantly lower (0.276 seconds when utilizing GPU compared to >= 4.316 seconds when not utilizing GPU on Nvidia Jetson Nano), as shown in Table 2.7 and Table 2.8.

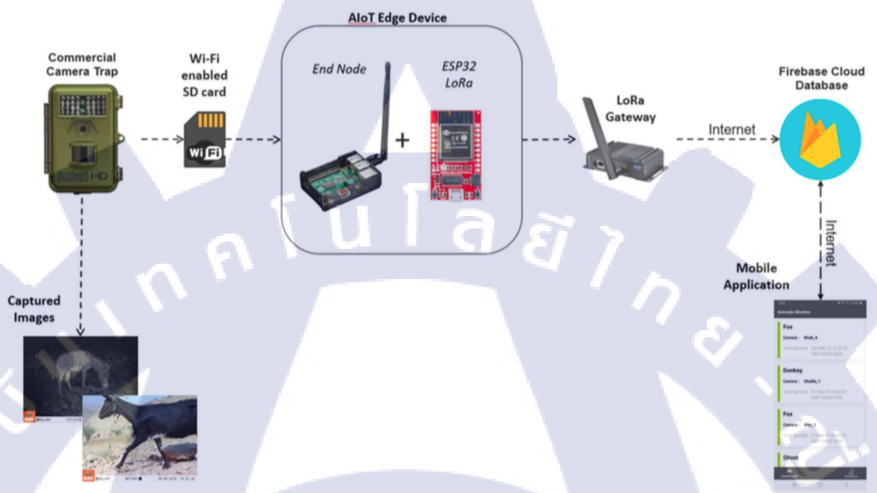


Figure 2.11 IoT system architecture for IoT camera trap system [2]

Table 2.7 Power Consumption and CPU Utilization by Edge Device (1000 inferences) [2]

Device (Model)	**Average Current (mA)**	**Max Current (mA)**	**Average CPU Util. (%)**	**Max CPU Util. (%)**
Raspberry Pi (TFLite)	838.99	977	23.62	57.3
Google Coral (TFLite)	1074.88	1140	50.41	56.8
Jetson Nano (TFLite)	874.37	1057	34.85	60.6

Table 2.7 Power Consumption and CPU Utilization by Edge Device (1000 inferences) [2] (Continued)

Device (Model)	Average Current (mA)	Max Current (mA)	Average CPU Util. (%)	Max CPU Util. (%)
Jetson Nano (TensorRT)	1665.21	2005	27.14	53.7

Table 2.8 Average Latency and Standard Deviation per Image by Edge Device (1000 inferences) [2]

Device (Model)	**Capture Time (s)**	**Pre-Process Time (s)**	**Inference Time (s)**	**Total Time (s)**
RPI-TFLite	0.518 (0.014)	0.011 (0.002)	2.835 (0.036)	3.365 (0.039)
Coral-TFLite	0.013 (0.001)	0.014 (<0.000)	2.739 (0.002)	2.765 (0.002)
Nano-TFLite	0.001 (<0.000)	0.006 (<0.000)	4.316 (0.026)	4.324 (0.026)
Nano-TensorRT	0.002 (<0.000)	0.006 (<0.000)	0.276 (0.002)	0.283 (0.002)

E. Charteros and I. Koutsopoulos. [6] proposed a topic "Edge Computing for Having an Edge on Cancer Treatment: A Mobile App for Breast Image Analysis". They created a mobile application prototype (Figure 2.12) that let the patient take a photo of themselves to perform breast analysis in the app. Face detection was used for removing the patient's face to ensure the patient's privacy and a machine learning model (MaskRCNN) was implemented inside the app to analyze the health of the breast from taken photos. Their proposed system has 98% accuracy in breast identification and the mask prediction finding correctly about 90-95% of the breast surface in both cases. They evaluated the speed of their machine learning application on a low-end mobile device (Huawei Y6 Prime) and a high-end mobile device

(Huawei Mate 20 Pro). They found that it took 20 seconds on the low-end mobile device whereas on the high-end mobile device, took 12-14 seconds. They also reported that energy consumption is medium when using camera and face detection, and high when analyzing a breast image, based on the metrics provided by Android Studio.

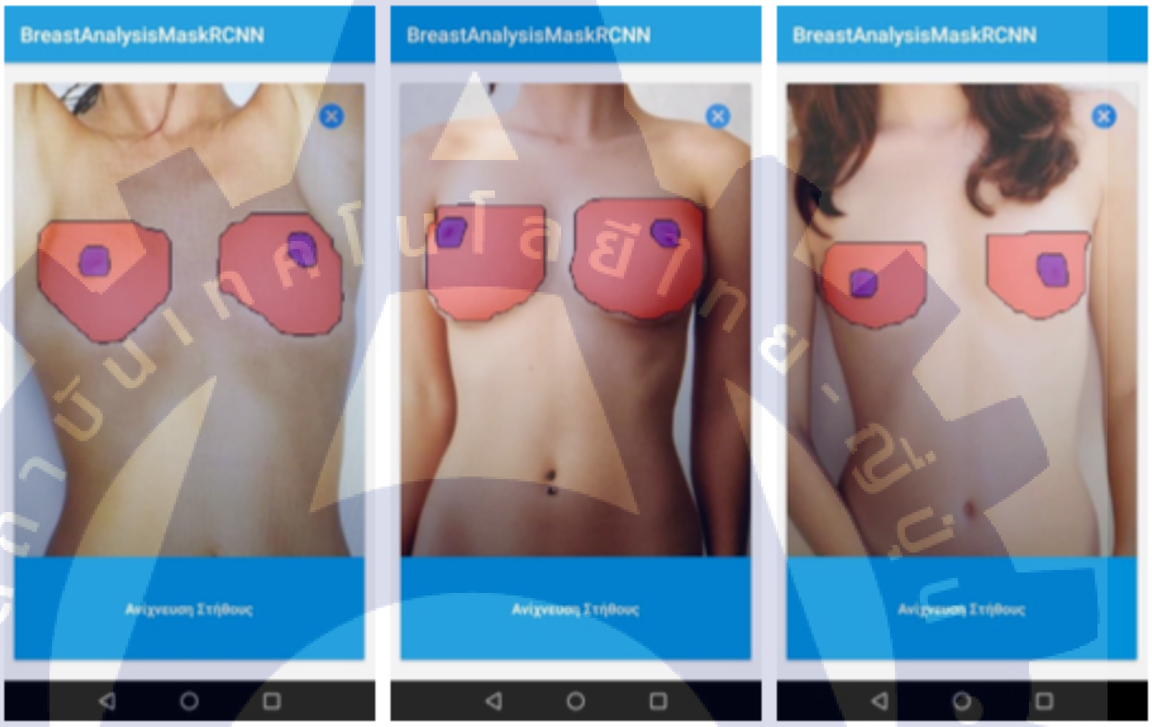


Figure 2.12 Mobile Application (Breast Detection Results) [6]

S. R. Reza, et al. [36] evaluated the inference performance of several popular pre-trained convolutional neural networks (CNN) models, namely MobileNet V1, MobileNet V2 [32], and Inception V3, on three edge computing devices: NVIDIA Jetson TX2, NVIDIA Jetson Nano, and Google Edge TPU. MobileNet V1 and MobileNet V2 were found as candidate models for speed, while Inception V3 was found to be more accurate.

A. Ignatov, et al. [37] published an article that contains tables representing benchmarks of Android smartphones in the market in 2019. The benchmarks consist of the inference time performance of MobileNet v2, Inception-ResNet, SRCNN, VGG-19, and DPED.

2.5 Related Theories

2.5.1 Evaluating a machine learning model

It is important to evaluate a machine learning model to see how well the model is. The techniques that will be used for observing and evaluating the model created in this research are listed below.

2.5.1.1 Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a classification model. It shows the number of true positives, true negatives, false positives, and false negatives, which can be used to calculate various evaluation metrics such as precision, recall, and accuracy. [38,39]

1) **True Positive** is a type of accurate prediction where a model predicted that an object belongs to a class, and it actually belongs to a class.

2) **True Negative** is a type of accurate prediction where a model predicted that an object does not belong to a class, and it actually not belongs to a class.

3) **False Positive** is a type of inaccurate prediction where a model predicted that an object belongs to a class but actually not belongs to a class.

4) **False Negative** is a type of inaccurate prediction where a model predicted that an object does not belong to a class but actually belongs to a class.

2.5.1.2 Accuracy

Accuracy is the percentage of correct prediction, which can be calculated by using the equation (1) below.

$$Accuracy = \frac{Correct\ Prediction}{All\ Prediction} \qquad (1)$$

2.5.1.3 Precision

Precision is a percentage of the model's accuracy when predicting positive samples, which can be calculated by using the equation (2) below.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \qquad (2)$$

2.5.1.4 Recall

Recall is a percentage of the model's ability to find all positive samples, which can be calculated by the equation (3) below.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \qquad (3)$$

2.5.1.5 F1 Score

The F1 score is a metric that combines precision and recall to provide a single measure of a model's performance, represented in percentage.

A higher score indicates that the model is correctly identifying positive samples while minimizing false samples.

The F1 score can be calculated by the simplified equation (4) below.

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} \qquad (4)$$

2.6 Related Technologies

2.6.1 Swift

Swift [40] is a powerful and intuitive programming language developed by Apple for building apps on Apple platforms such as iOS, macOS, watchOS, and tvOS.

Figure 2.13 Swift Logo

2.6.2 FastAPI

FastAPI [41] is a modern, high-performance web framework for building APIs with Python 3.7+. It is designed to be fast, easy to use, and based on standard Python type hints.

Figure 2.14 FastAPI Logo

Chapter 3
Methodology

Following a literature review and technology trends, it was found that edge devices, especially smartphones, and edge servers have improved processing power over the years, enabling the feasibility of the image classification process. Additionally, offloading the application to process at the edge significantly reduces or eliminates the network delay and may increase privacy for the end user. The researcher then questioned whether performing image classification on edge devices or edge servers may be faster than sending data over the internet to the cloud.

Therefore, the researcher proposed experimental research to study and evaluate the performance and analyze the results using statistical methods, as outlined below.

3.1 Conceptual Framework
3.2 Proposed Method
3.3 Environment Setup
3.4 Dataset Preparation
3.5 Create Image Classifier
3.6 Applying Image Classifier
3.7 Result Gathering
3.8 Result Analysis

3.1 Conceptual Framework

The conceptual framework in Figure 3.1 shows the variables and the design of this experimental research. The conceptual framework consists of independent variables, dependent variables, and moderating variables as described below.

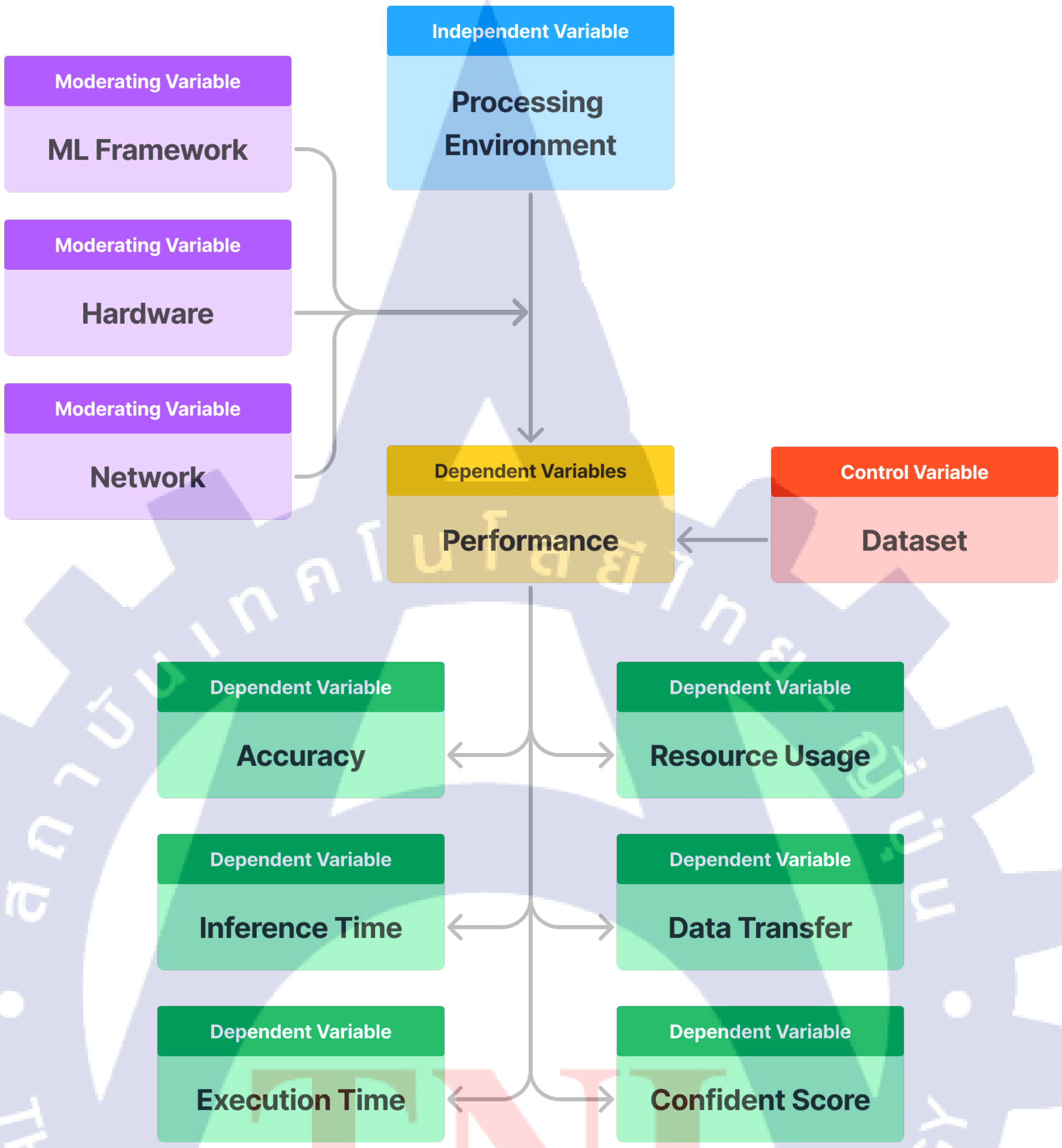


Figure 3.1 Conceptual Framework

3.1.1 Independent Variable

There is only one independent variable studied in this research, i.e., the processing environment. The selected values that will be evaluated in this research are Edge Device, Edge Server, and Cloud Server.

3.1.2 Dependent Variable

There are six dependent variables observed in this research topic, i.e., Accuracy, Inference Time, End-to-End Execution Time, Resource Usage, Data Transfer, and Confident Score. The dependent variables will be used to conclude the findings and provide an answer to the research questions and hypothesis.

3.1.3 Moderating Variables

There are three moderating variables that will influence the outcome, i.e., ML Framework, Hardware, and Network. The processing environment will be moderated by these variables.

3.1.4 Control Variables

There are three moderating variables that will influence the outcome, i.e., ML Framework, Hardware, and Network. The processing environment will be moderated by these variables.

3.2 Proposed Method

The proposed method shown in Figure 3.2 consists of 6 steps listed below. The details of each step are described in Sections 3.3 to 3.8.

1) Environment Setup
2) Dataset Preparation
3) Create Image Classifier
4) Applying Image Classifier
5) Result Gathering
6) Result Analysis

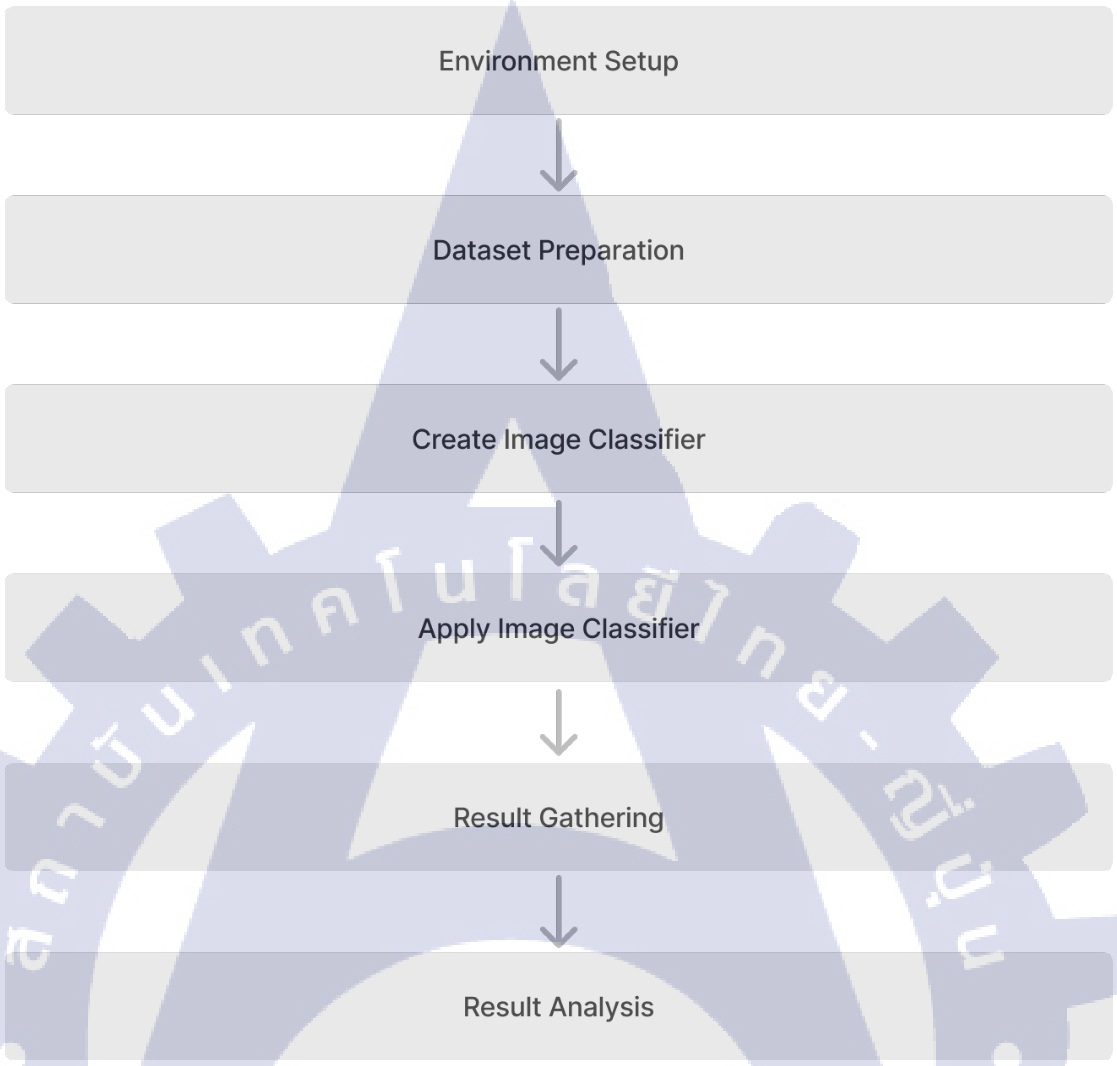


Figure 3.2 Proposed Method

3.3 Environment Setup

To test the hypothesis, three environments were set up as shown in Table 3.1. iPhone 15 Pro series [17] with an Apple A17 Pro SoC was selected to represent an edge device, A computer with a discrete GPU, Nvidia GeForce RTX 3070 [18], was selected to represent an edge server. A public cloud GPU instance (virtual machine with GPU), EC2 G5.xlarge Instance, from Amazon Web Services (AWS) [19] with an NVIDIA A10G Tensor Core GPU [20] was selected to represent a cloud GPU. Three primary criteria for each environment representative selection are listed below.

1) The device must have a GPU that has sufficient processing power according to the official technical specification and literature review. The GPU is required to ensure the speed and efficiency of the process.

2) Popularity

3) Affordability

Table 3.1 Environment Setup

Component	**Environment**		
	Edge Device	**Edge Server**	**Cloud Server**
Hardware / Device	iPhone 15 Pro series	Personal Computer	AWS EC2 G5 g5.xlarge
Processor / SoC	Apple A17 Pro	Intel Core i5-12400F	AMD EPYC 7R32
CPU Cores / vCPU	2 Performance Cores 4 Efficiency Cores	6 Cores, 12 Threads	4 vCPU
Memory	8 GB Unified	16 GB	16 GB
GPU	Apple A17 Pro 6 GPU Cores	NVIDIA GeForce RTX 3070	NVIDIA A10G
GPU Memory	8 GB Unified	8 GB	24 GB
Operating System	iOS 17	Ubuntu Linux 24.04 LTS	Ubuntu Linux 22.04 LTS
Application	Swift	Python (FastAPI)	Python (FastAPI)
ML Runtime	CoreML	TensorRT	TensorRT
Network	Not Required	LAN - Ethernet	Internet - FTTx

3.3.1 Edge Device Environment

This environment is represented by a popular, well-known, and widely used flagship smartphone in the market, the Apple iPhone 15 Pro series, which comes with an Apple A17 Pro System-on-Chip with 2 performance cores and 4 efficiency cores, 8 GB of Unified Memory, and 6 GPU cores. According to its specs, it found that there is a potential processing power to process image classification.

Additionally, Apple has an ecosystem for machine learning called CoreML, which is a proprietary optimization of the execution of machine learning on Apple Silicon devices. Thus, CoreML was selected as an ML runtime for this environment.

The image classification application for this environment will be an application developed using Swift to run the CoreML image classifier model. No network is required to classify an image.

3.3.2 Edge Server Environment

This environment is represented by a personal computer that has hardware specs that are viable for processing image classification. It comes with NVIDIA GeForce RTX 3070, a mid-range and affordable price GPU, together with Intel Core i5-12400F, a mid-range, popular, and affordable price GPU. The RTX 3070 comes with 8 GB of GDDR6 256-bit memory, 448 GB/s memory bandwidth, 5,888 CUDA Cores, and 184 Tensor Cores. The Intel Core i5 CPU comes with 6 cores, 12 threads (Hyperthreading enabled), 2.5 GHz base clock speed, 4.4 GHz Turbo Boost clock speed, 16 GB of RAM, and up to 1 Gbps of network bandwidth.

Additionally, NVIDIA has TensorRT, which is a proprietary optimization for running a machine learning model. Thus, TensorRT was selected as an ML runtime for this environment.

The image classification application for this environment will be an API application developed using FastAPI to classify an image using the TensorRT image classifier model. An API client will be run on a different machine, but within the same network. A Local Area Network (LAN) connection and Ethernet are required for this environment because edge computing requires processing to be executed at the same network or the closest location to the user. In this case, the LAN connects the API application and the API client (which acts as a user).

3.3.3 Cloud Computing Environment

This environment is represented by an affordable cloud GPU instance or a virtual machine from AWS (Amazon Web Services), a G5.xlarge instance type, launched in the Tokyo region. It comes with an NVIDIA A10G with 24 GB GPU memory, 4 virtual CPUs of AMD EPYC 7R32, 16 GB of RAM, and up to 10 Gbps of network bandwidth.

TensorRT was selected as an ML runtime for this environment because of the proprietary optimization for running a machine learning model with TensorRT.

The image classification application for this environment will be an API application developed using FastAPI to classify an image using the TensorRT image classifier model. An API client will run on a different machine and network. An internet connection is required to simulate cloud computing and connect the API client and the API application.

3.4 Dataset Preparation

In this research, two datasets featuring images of dogs and cats were used, one for training and validation and one for testing.

The first dataset is the Kaggle: Cats VS Dogs dataset from Microsoft and PetFinder.com. It is used for training and validation. It contains 12,491 images of cats and 12,470 images of dogs. This dataset was chosen for training and validation because its photo consists of mixed backgrounds, postures, and image sizes. It also has plenty of images for training and validation. The sample data of this dataset is shown in Figure 3.3.

The second dataset is the Animal Faces-HQ (AFHQ) dataset. It is used for testing. It consists of high-quality images. The images are 512-by-512 pixels in resolution. Images of cats and dogs in the training subset were chosen. Each class contains about 5,000 images. This dataset was chosen because there was a high number of photos to simulate and gain a stable statistic, and its quality is consistently good. The sample data of this dataset is shown in Figure 3.4.

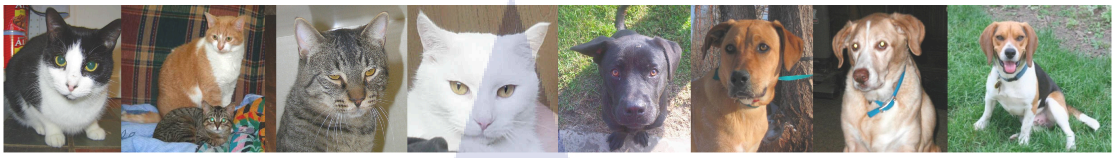


Figure 3.3 Sample Data of Kaggle Cats VS Dogs Dataset

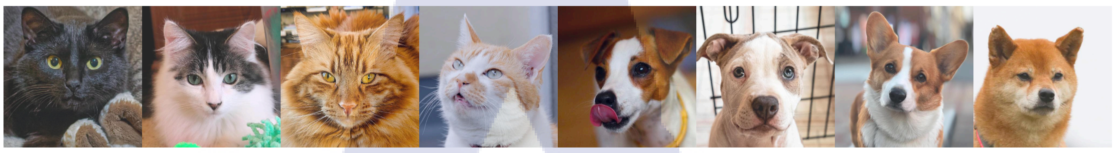


Figure 3.4 Sample Data of Animal Faces-HQ Dataset

3.5 Create Image Classifier

The model varies according to the environment setup. Core ML was used in the edge environment, while TensorRT was used in the cloud environment because of the proprietary optimization of hardware and ML framework.

3.5.1 Create an image classifier model for an edge device

This section describes the procedure to create an image classifier model for the edge device environment. macOS is required to run Create ML. The steps are listed below and represented in a flow as shown in Figure 3.5.

1) Split the images prepared in Section 3.4 into two sets:

- Training Set – A set for training data
- Testing Set – A set for testing

2) For each set, store classified images in the sub-folders, the sub-folder name will be the label of the classified images. The example is shown in Figure 3.6.

3) Install Xcode from App Store

4) Create a new Create ML project, then drag the folder of the set of images and drop them on Training, Validation, and Testing.

5) Configure training properties as follows:

- Validation Data: Automatic (Split from training data)

- Target Iteration: 25
- Augmentation: None

6) Start training by clicking on “Train”.
7) Observe the output of Machine Learning.
8) Export the trained model to a .mlmodel file.

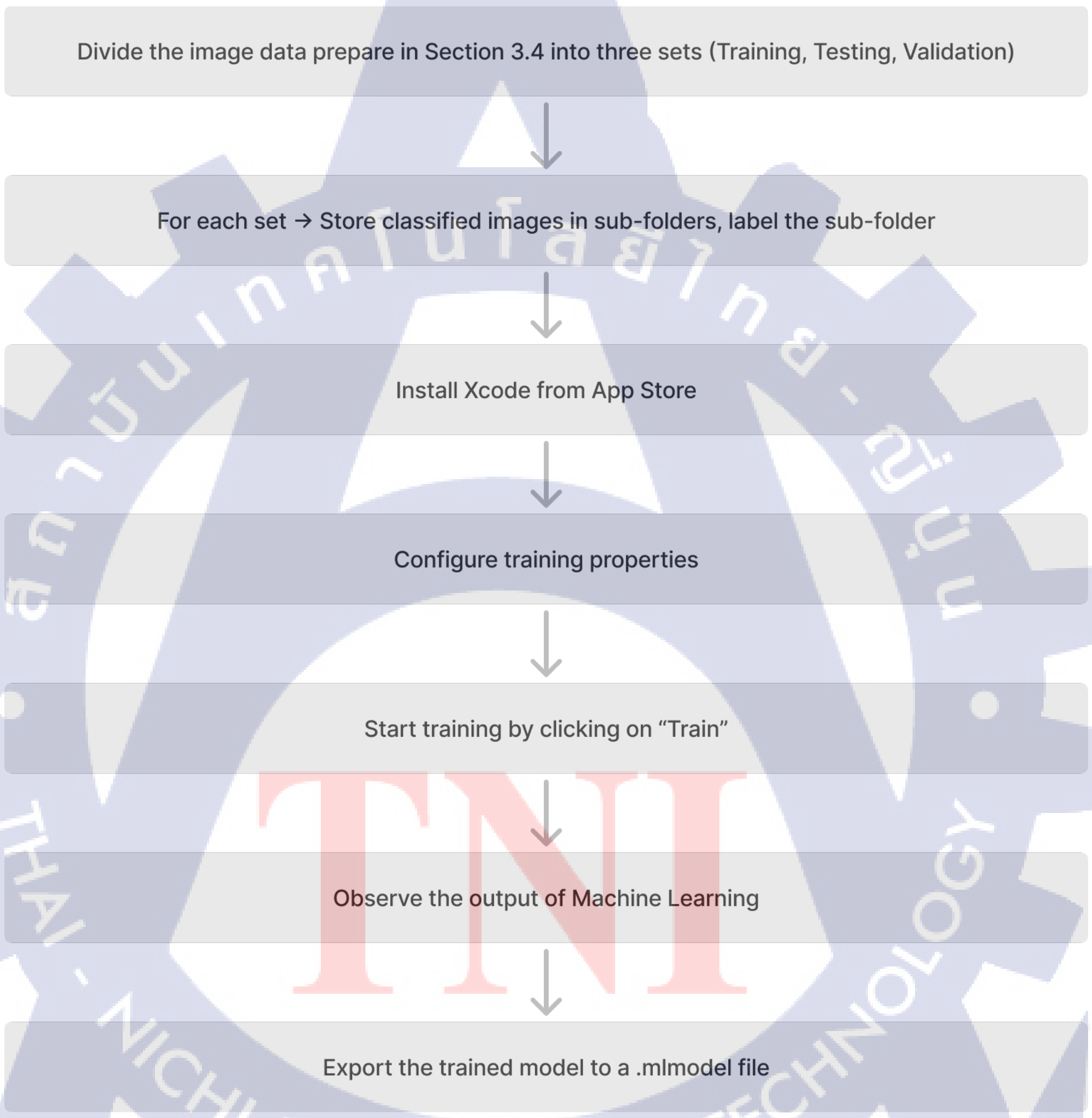
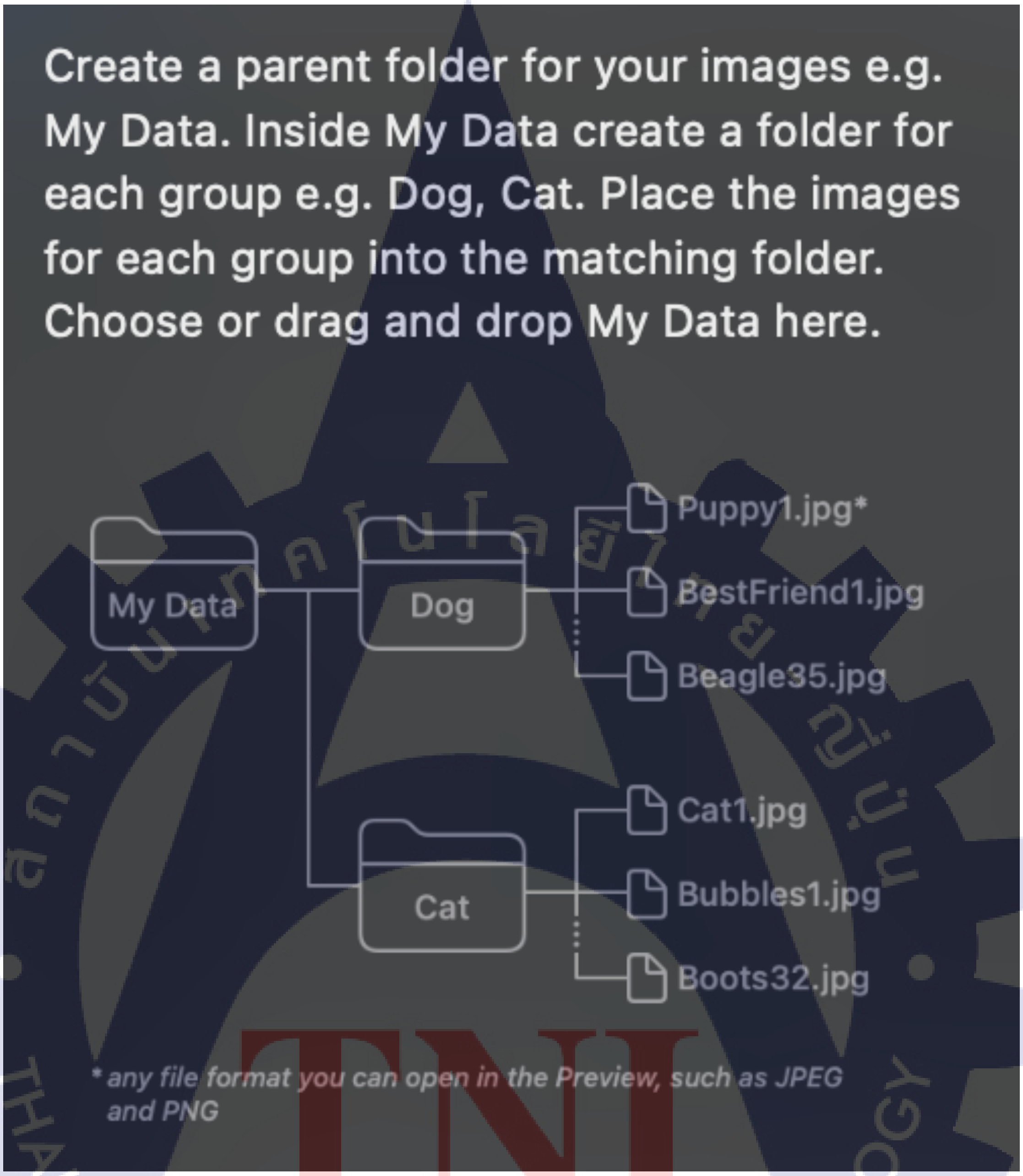

Divide the image data prepare in Section 3.4 into three sets (Training, Testing, Validation)
For each set → Store classified images in sub-folders, label the sub-folder
Install Xcode from App Store
Configure training properties
Start training by clicking on “Train”
Observe the output of Machine Learning
Export the trained model to a .mlmodel file

Figure 3.5 Steps to create an image classifier model using Create ML for iOS (Edge Device Environment).


Create a parent folder for your images e.g. My Data. Inside My Data create a folder for each group e.g. Dog, Cat. Place the images for each group into the matching folder. Choose or drag and drop My Data here.

* any file format you can open in the Preview, such as JPEG and PNG

Figure 3.6 Create ML dataset folder structure instruction

3.5.2 Create an image classifier model for edge server and cloud server

This section describes the procedure to create an image classifier model for the Linux-based environment (for edge server and cloud server). The steps are listed below and represented in a flow as shown in Figure 3.7.

1) Divide the image data prepared in Section 3.4 into two sets:

- Training Set – A set for training data
- Testing Set – A set for testing

2) For each set, store classified images in sub-folders, the sub-folder name will be the label of the classified images.

3) Create a Python project.

4) Import TensorFlow libraries.

5) Configure the training properties as follows:

- Target Iteration: 50
- Early Stop Patient: 8
- Validation Split Rate: 5%
- Augmentations
    - Random Flip
        - Horizontal
    - Random Rotation: 0.1

6) Load Dataset

7) Create the model.

8) Train the model.

9) Visualize and observe the training results.

10) Convert the trained model into a TensorRT model.

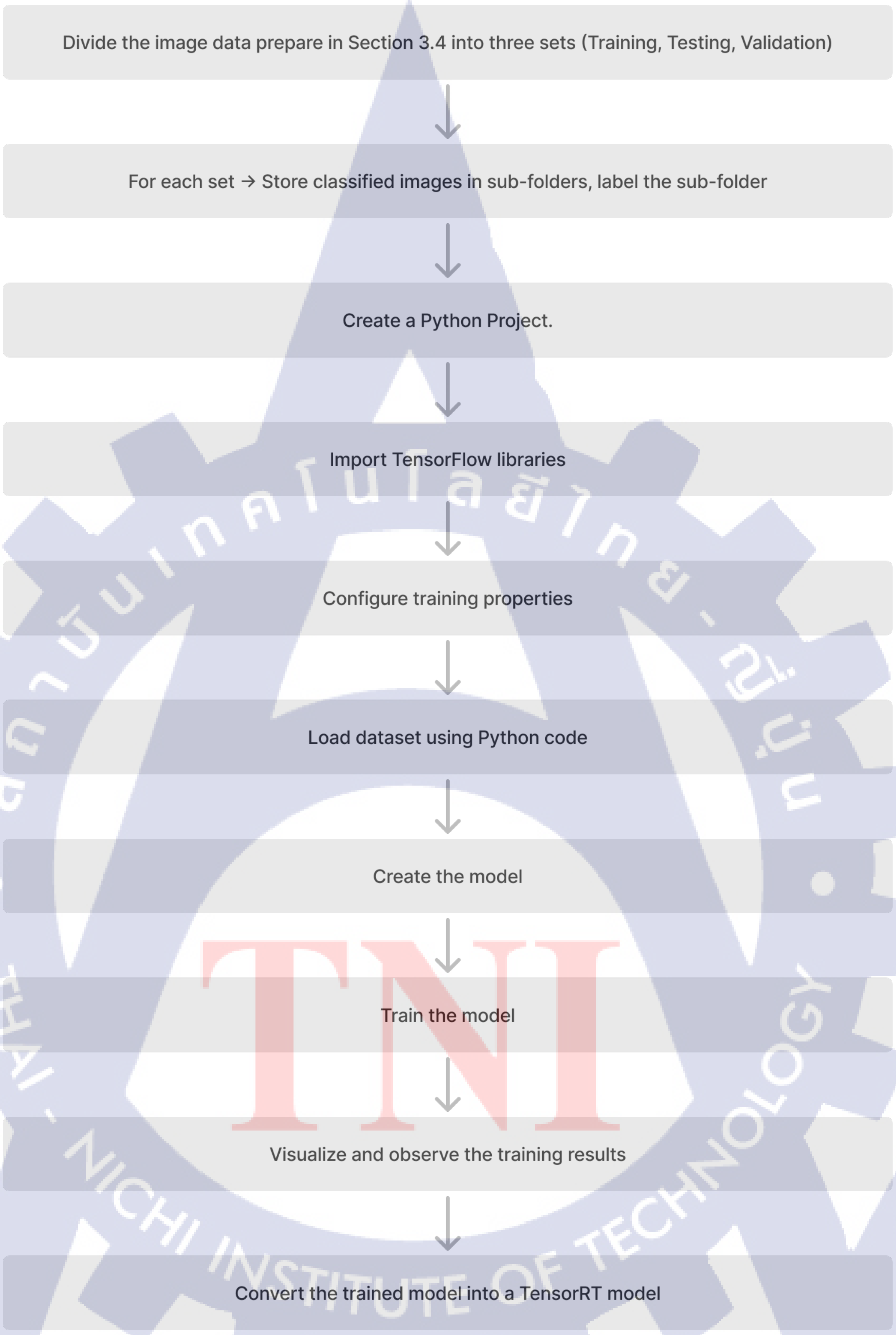

Divide the image data prepare in Section 3.4 into three sets (Training, Testing, Validation)
For each set → Store classified images in sub-folders, label the sub-folder
Create a Python Project.
Import TensorFlow libraries
Configure training properties
Load dataset using Python code
Create the model
Train the model
Visualize and observe the training results
Convert the trained model into a TensorRT model

Figure 3.7 Steps to create an image classifier model using TensorFlow and Keras for Linux-based environment (for edge server and cloud server)

3.6 Apply Image Classifier

3.6.1 Apply the image classifier model for an edge device

This section described the procedure for bundling the exported Core ML model from Section 3.5.1 to a Swift application for iOS. macOS is required to use Xcode for building a Swift application. The steps are described below and represented in Figure 3.8.

The application lets the user choose a photo album that contains the testing dataset. After the user chooses a photo album, all the photos or images in the album and an output handler function are passed to the Core ML model to infer and handle the prediction results. The activity or flow of the iOS Swift application is shown in Figure 3.9.

1) Create an Xcode Project.

2) Import .mlmodel file as an asset of the project.

3) Compose the code for displaying the user interface

4) Create a programming function that listens to the gesture events and accepts an image as input for classification;

5) Make the function call to the Core ML model.

6) Retrieve the list of predictions.

7) Format and store the output.

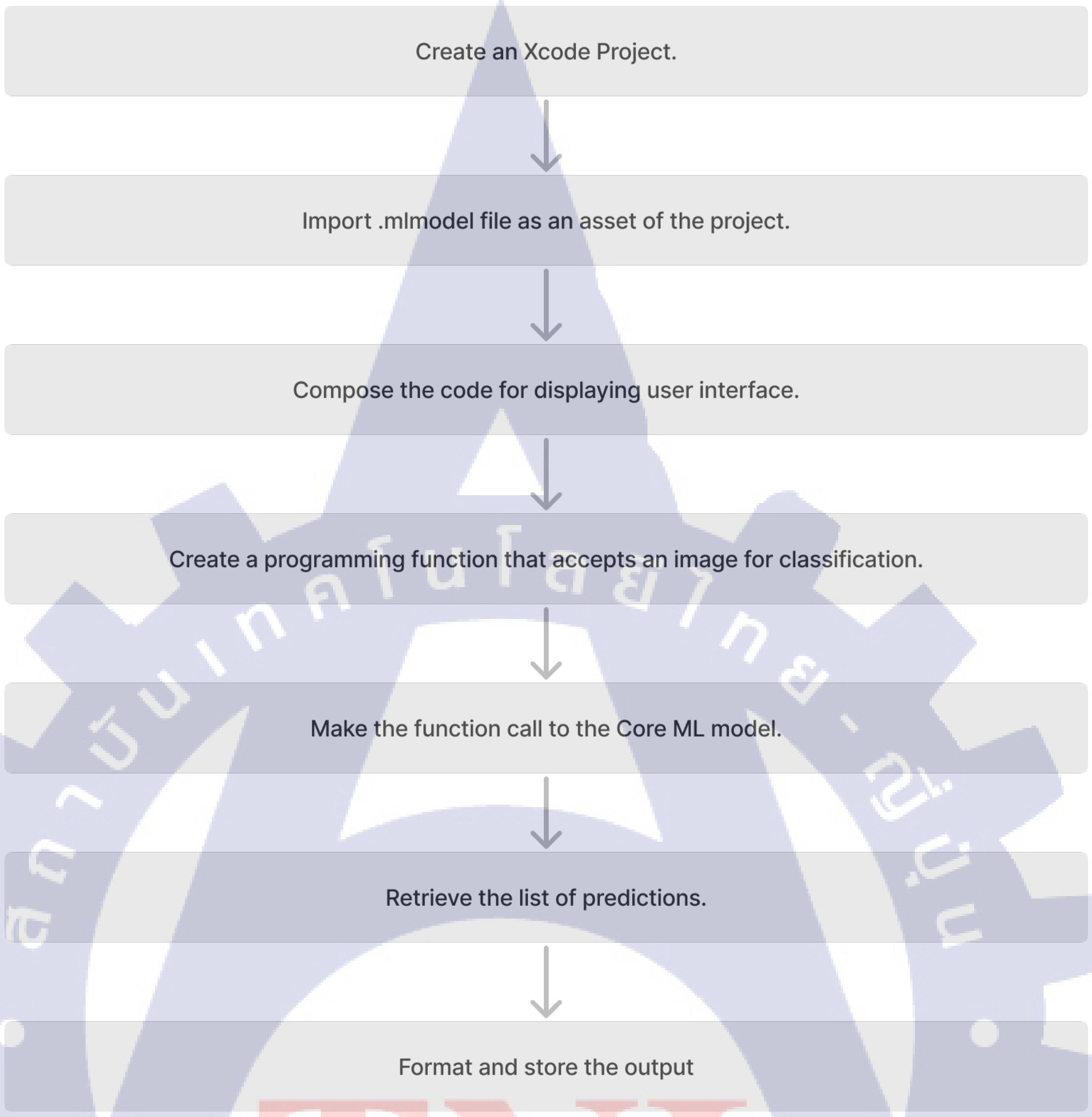

Create an Xcode Project.
Import .mlmodel file as an asset of the project.
Compose the code for displaying user interface.
Create a programming function that accepts an image for classification.
Make the function call to the Core ML model.
Retrieve the list of predictions.
Format and store the output

Figure 3.8. Steps to bundling the Core ML model to an iOS Swift application.

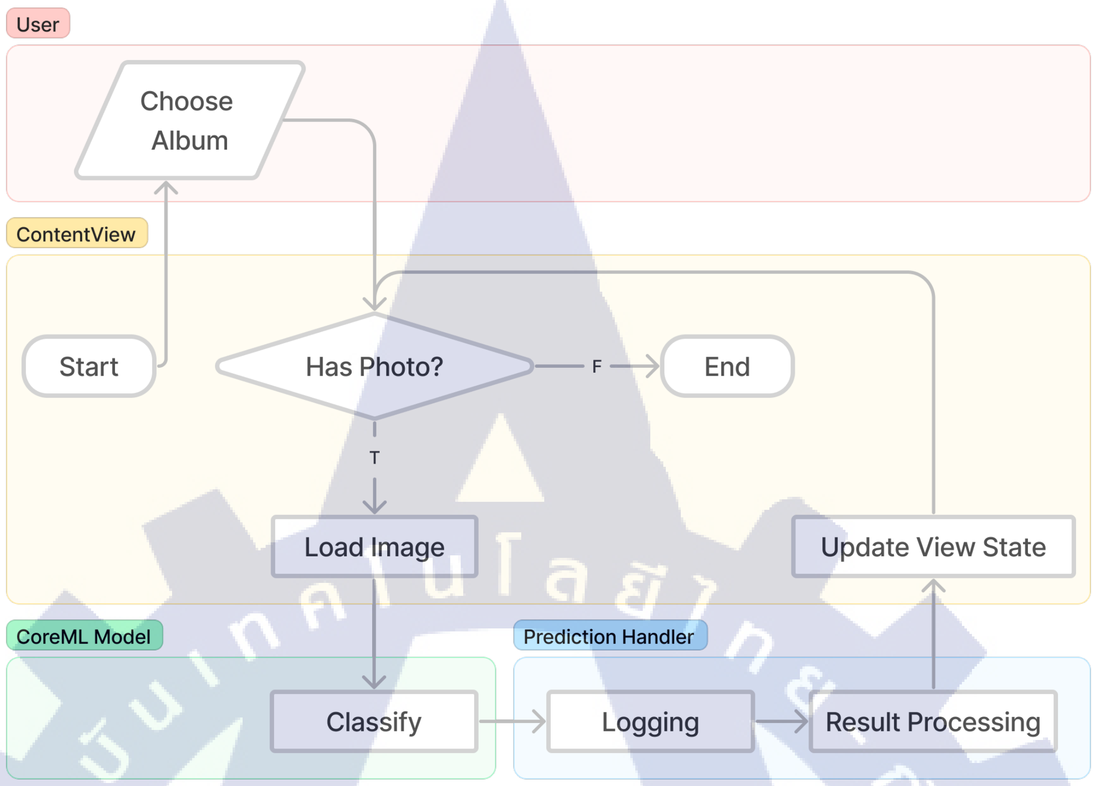


Figure 3.9 Activity diagram of the iOS Swift application

<u>3.6.2 Apply the image classifier model for edge server and cloud server</u>

This section described the procedure for bundling the exported TensorRT model from Section 3.5.2 to a REST API web service application. The creation steps are described below and represented in a flow as shown in Figure 3.10.

The application receives the image uploaded by a user and passes it to the TensorRT model to infer and return the prediction results. The activity or flow of the API application is shown in Figure 3.11.

1) Create a REST API web service application project.

2) Import the model.

3) Create a function to provide the REST API that accepts an image as input for classification.

4) Make the function call to the TensorRT model.

5) Retrieve the list of output.

6) Format and store the output.

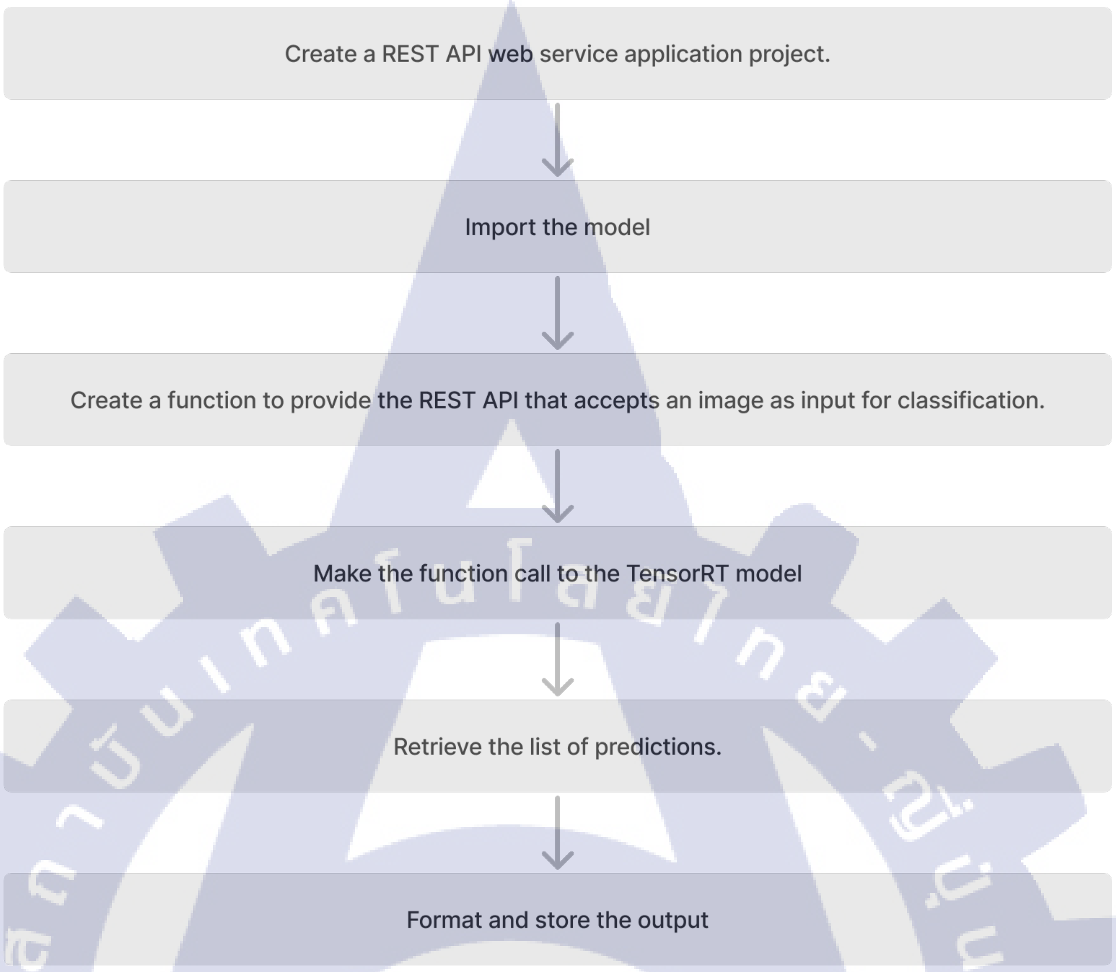

Create a REST API web service application project.
Import the model
Create a function to provide the REST API that accepts an image as input for classification.
Make the function call to the TensorRT model
Retrieve the list of predictions.
Format and store the output

Figure 3.10 Steps to bundling the TensorRT model to a REST API web service application

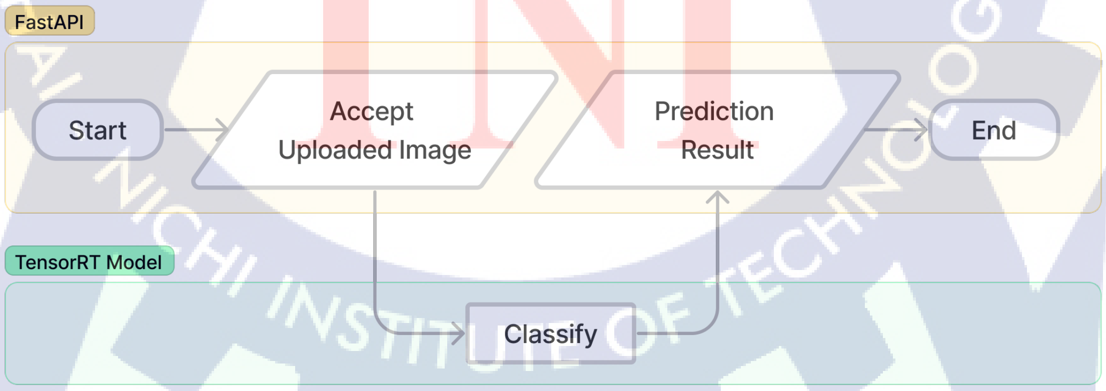


Figure 3.11 Activity diagram of the API application

3.7 Result Gathering

The results of inference time, end-to-end execution time, testing accuracy, and confidence score were gathered by a performance evaluation system represented in Figure 3.12, the resource usage data were obtained through various monitoring tools, and the data transfer results were derived from statistical calculations based on the file sizes of the images in the dataset.

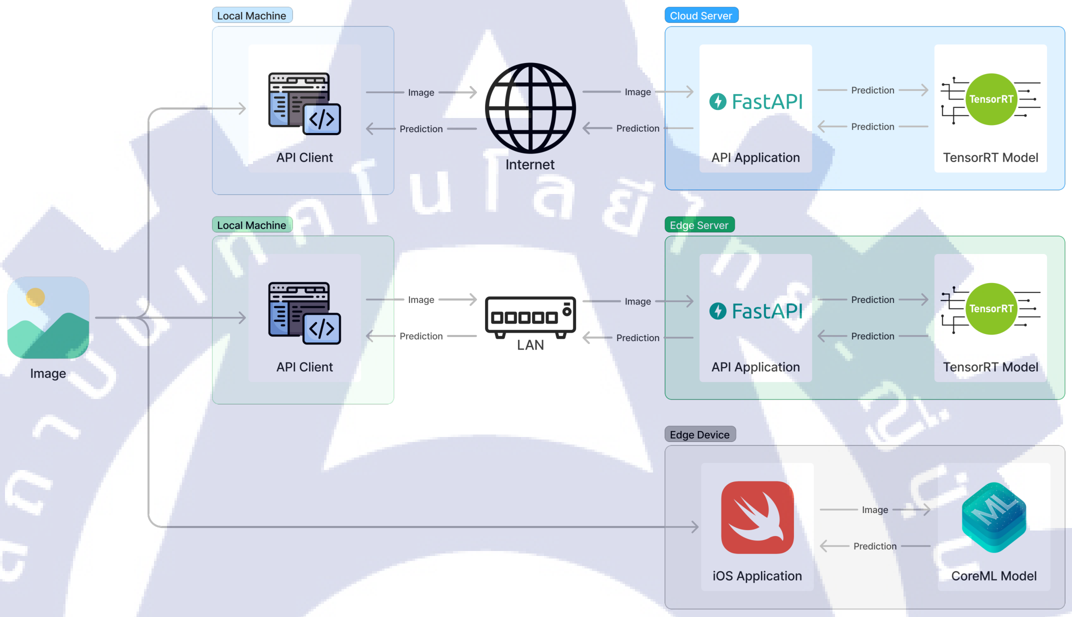


Figure 3.12 Overview of Performance Evaluation System

3.7.1 Inference Time

Inference Time is the duration when the image classifier begins classifying the image and ends when the results are returned.

For the edge environment, a timestamp was captured after an image was loaded and before passing to the classify function, and another timestamp was captured after the model predicted and finished handling the results. Then, the difference between the two captured timestamps was calculated and converted to milliseconds.

For the cloud environment, a timestamp was captured once the API application received the uploaded image, and another timestamp was captured once the model returned the prediction results. Then, the difference between the two captured timestamps was calculated and converted to milliseconds.

The inference time of each image classification will be stored in the database for statistical analysis.

3.7.2 End-to-End Execution Time

End-to-end execution Time is the duration when the system receives the user's input image, passes it to the image classifier to classify, waits for the prediction returns, and ends. This includes the network delay, if applicable.

For the edge device environment, a timestamp was captured once the image began to load, and another timestamp was captured once the results were printed. The difference between the two captured timestamps was calculated and converted to milliseconds.

For the cloud environment, an API client was developed by using Java version 17 (Amazon Corretto 17 aarch64) to run the test. The client invokes the API by making HTTP requests to the API application. This approach simulates the real-world application that submits data to process on the remote server. A timestamp was captured once the API client executed an HTTP request, and another timestamp was captured once the API client received an HTTP response that contained the prediction results. Then, the difference of the two captured timestamps was calculated and converted to milliseconds. All of the images were resized from 512-by-512 pixels to 256-by-256 pixels before being uploaded to the cloud.

3.7.3 Accuracy

The accuracy results include training, validation, testing accuracy, confusion matrix, and model performance. Training accuracy and validation accuracy were obtained from the model training results trained with the training and validation dataset (i.e. Kaggle: Cat VS Dog). Testing accuracy was obtained from the performance evaluation system in Figure 3.12 using the testing dataset (i.e. Animal Faces-HQ). The Confusion matrix was calculated from the accuracy. Model performance including Precision, Recall, and F1-Score was calculated from the accuracy and confusion matrix.

3.7.4 Confidence Score

The confidence score is a factor in the prediction result describing the degree of confidence of the prediction. It returns with each prediction result.

3.7.5 Resource Usage

Resource usage was gathered by utilizing a range of monitoring tools across different platforms: on the Edge Device, **Xcode** was utilized to monitor resource consumption, including CPU utilization, memory usage, and energy impact during the execution of image classification in real-time. For the Edge Server and Cloud Server, **htop** and **Cockpit** were utilized to monitor system performance, including CPU load, and memory usage in real-time. Additionally, **nvidia-smi** was utilized to track GPU performance, providing detailed statistics on GPU usage, and memory allocation in real-time.

3.7.6 Data Transfer

File size of images in the Animal Faces-HQ dataset will be used. The HTTP header is excluded.

3.8 Result Analysis

Results gathered from Section 3.7 will be analyzed and visualized in appropriate charts and tables. Figure 3.13 represents the overview of performance analysis.

Inference Time, End-to-End Execution Time, Confidence Score, and Data Transfer will be analyzed statistically to find the minimum, average (mean), maximum, and standard deviation values.

Accuracy will be calculated and visualized using charts and tables. Resource usage of each environment will be compared in a table

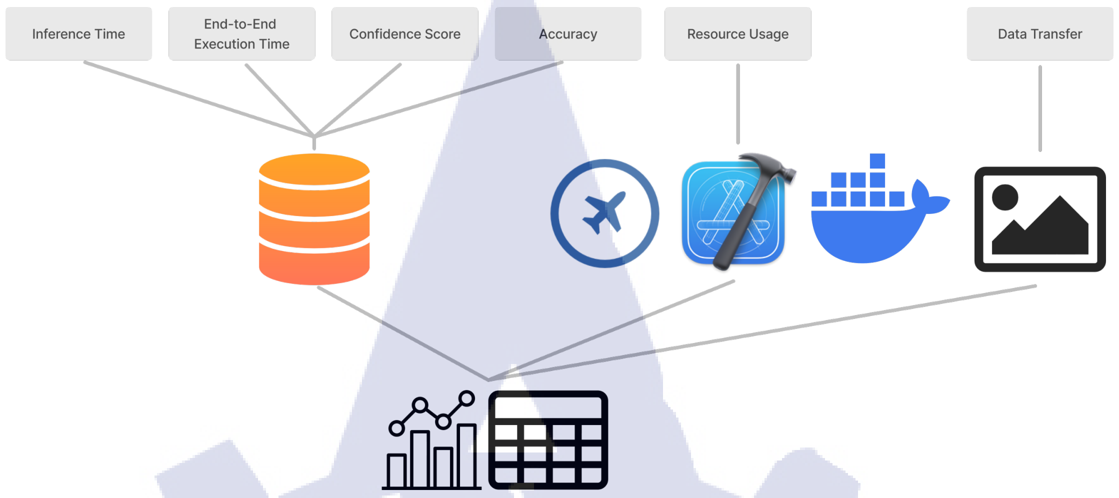


Figure 3.13 Performance Analysis

Chapter 4
Results

The gathered results will be analyzed and visualized in appropriate charts and tables. The results will be presented as follows:

4.1 Inference Time

4.2 End-to-End Execution Time

4.3 Accuracy

4.4 Confidence Score

4.5 Resource Usage

4.6 Data Transfer

4.1 Inference Time

Table 4.1 represents the minimum, average, maximum, and standard deviation of the inference time of image classification on the edge device, edge server, and cloud server, measured in milliseconds; lower is better.

In Table 4.1, it was found that the minimum inference time of the edge device is 5.02 milliseconds, the edge server is 1.76 milliseconds, and the cloud server is 2.32 milliseconds. The average inference time of the edge device is 16.30 milliseconds, the edge server is 3.51 milliseconds, and the cloud server is 3.15 milliseconds. The maximum inference time of the edge device is 1,175.30 milliseconds, the edge server is 9.52 milliseconds, and the cloud server is 131.63 milliseconds. The standard deviation of the inference time of the edge device is 11.74 milliseconds, the edge server is 1.31 milliseconds, and the cloud server is 1.43 milliseconds.

Figure 4.1. represents the comparison chart of inference time between the edge device, edge server, and cloud server. The histograms of each environment's inference time results are shown in Figures 4.2 to 4.4.

Table 4.1. Inference Time

Environment	Inference Time Results (milliseconds – lower is better)			
	Min	**Average**	**Max**	**S.D.**
Edge Device	5.02 ms	16.30 ms	1,175.30 ms	11.74 ms
Edge Server	1.76 ms	3.51 ms	9.52 ms	1.31 ms
Cloud Server	2.32 ms	3.15 ms	131.63 ms	1.43 ms

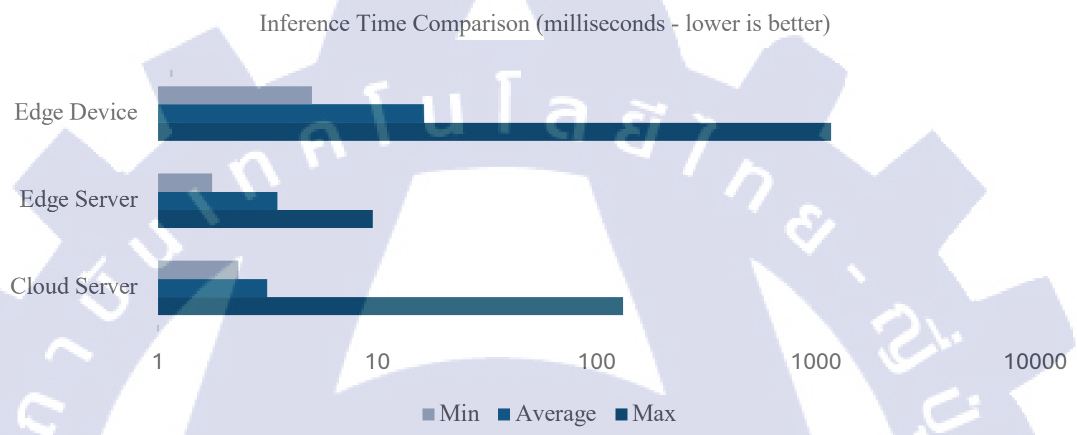


Figure 4.1 Inference Time Comparison

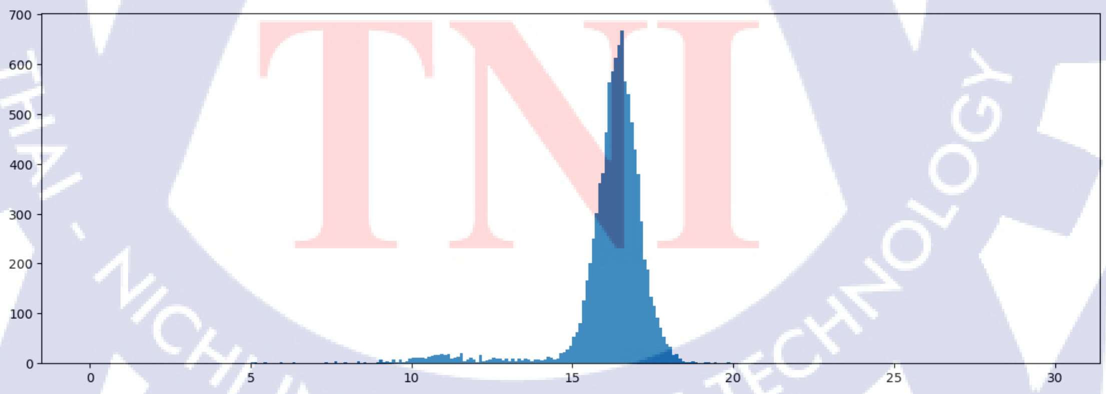


Figure 4.2 Histogram of the inference time results of the edge device

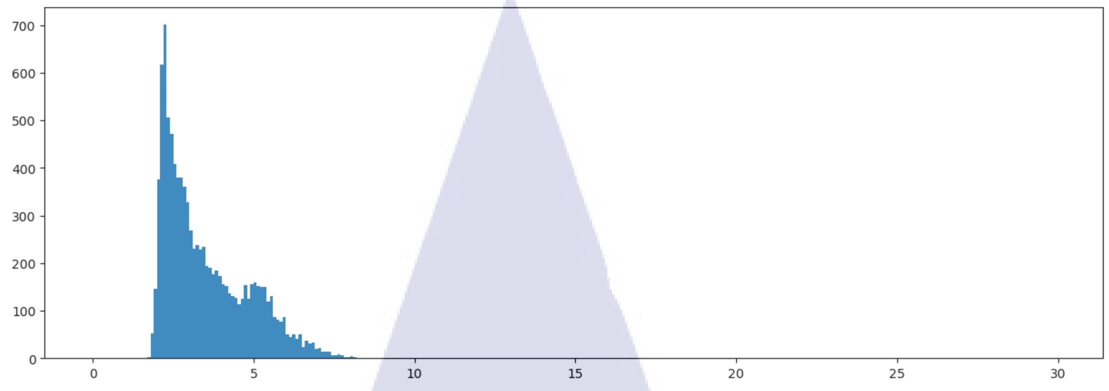


Figure 4.3 Histogram of the inference time results of the edge server

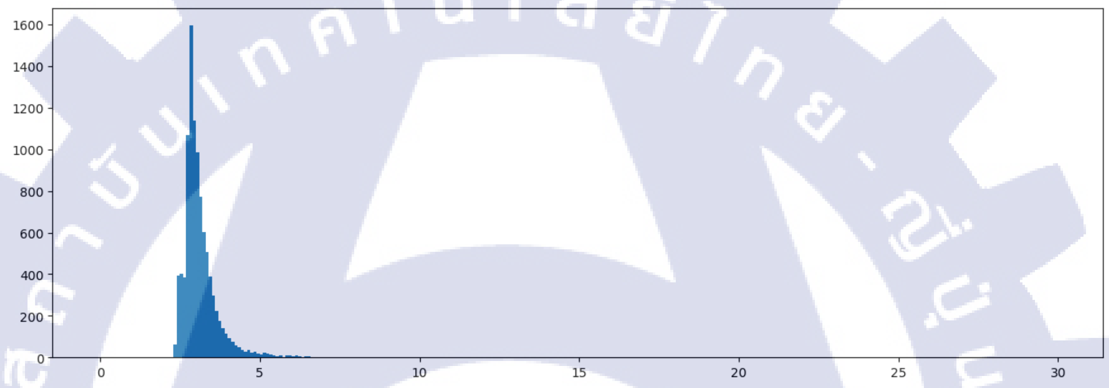


Figure 4.4 Histogram of the inference time results of the cloud server

4.2 End-to-End Execution Time

Table 4.2 represents the minimum, average, maximum, and standard deviation of the end-to-end execution time of image classification on the edge device, edge server, and cloud server, measured in milliseconds; lower is better.

In Table 4.2, it was found that the minimum end-to-end execution time of the edge device is 5.05 milliseconds, the edge server is 11 milliseconds, and the cloud server is 197 milliseconds. The average end-to-end execution time of the edge device is 16.47 milliseconds, the edge server is 28.23 milliseconds, and the cloud server is 280.54 milliseconds. The maximum end-to-end execution time of the edge device is 1,175.35 milliseconds, the edge server is 98 milliseconds, and the cloud server is 493 milliseconds. The standard deviation of the end-to-end execution time of the edge device is 11.75 milliseconds, the edge server is 8.17 milliseconds, and the cloud server is 47.88 milliseconds.

Figure 4.5. represents the comparison chart of end-to-end execution time between the edge device, edge server, and cloud server. The histograms of each environment's inference time results are shown in Figures 4.6 to 4.8.

Table 4.2. End-to-End Execution Time

Environment	**End-to-End Execution Time Results (milliseconds – lower is better)**			
	Min	**Average**	**Max**	**S.D.**
Edge Device	5.05 ms	16.47 ms	1,175.35 ms	11.75 ms
Edge Server	11 ms	28.23 ms	98 ms	8.17 ms
Cloud Server	197 ms	280.54 ms	493 ms	47.88 ms

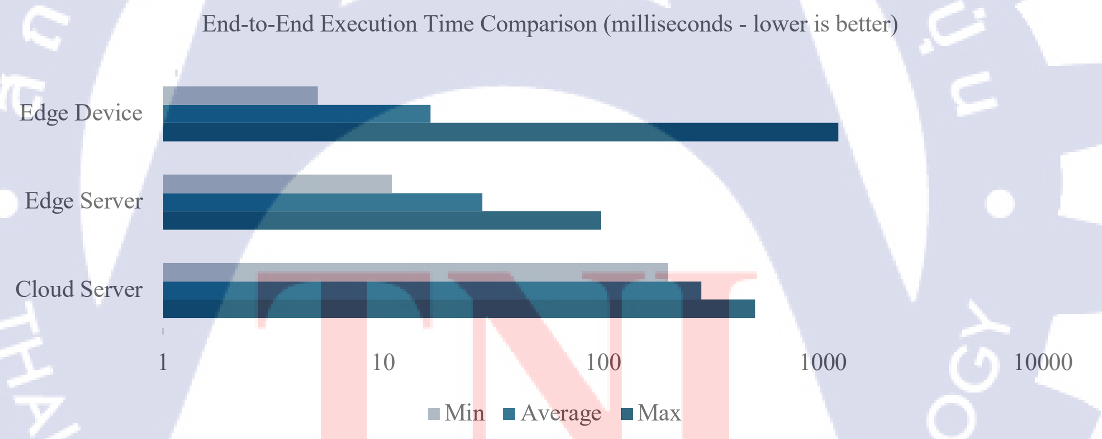


Figure 4.5. End-to-End Execution Time Comparison

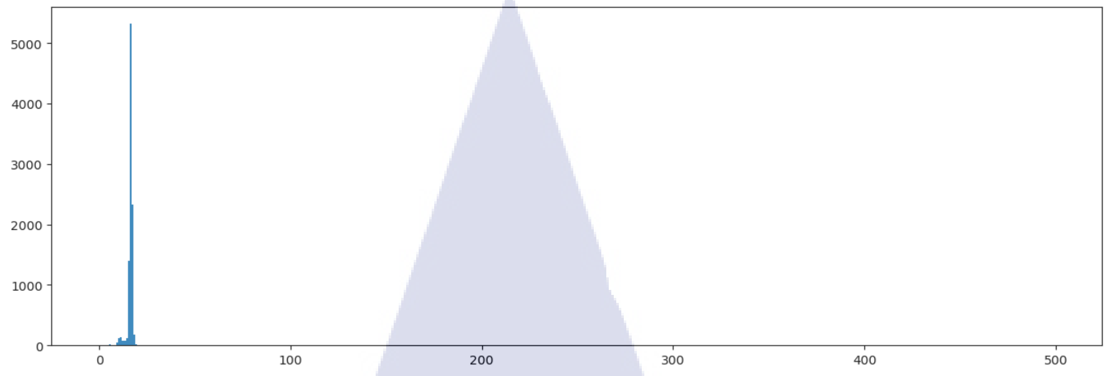


Figure 4.6 Histogram of the end-to-end execution time results of the edge device environment (iPhone + Core ML)

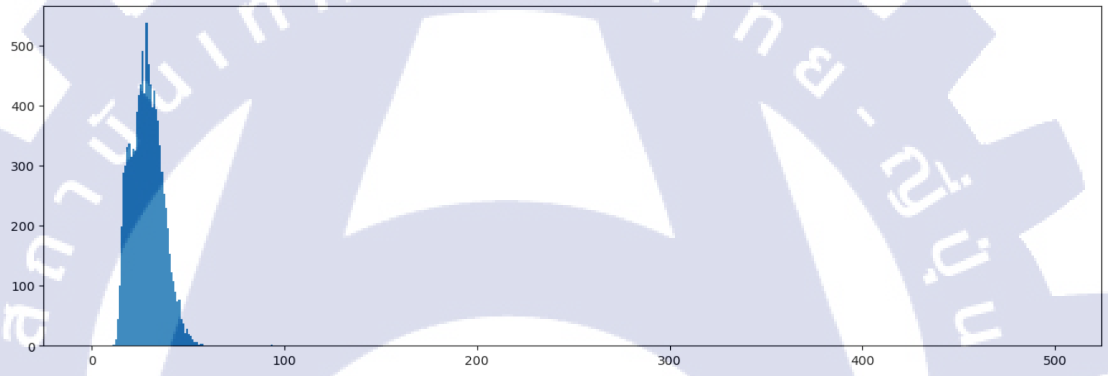


Figure 4.7 Histogram of the end-to-end execution time results of the edge server (NVIDIA GeForce RTX 3070 + TensorRT + LAN)

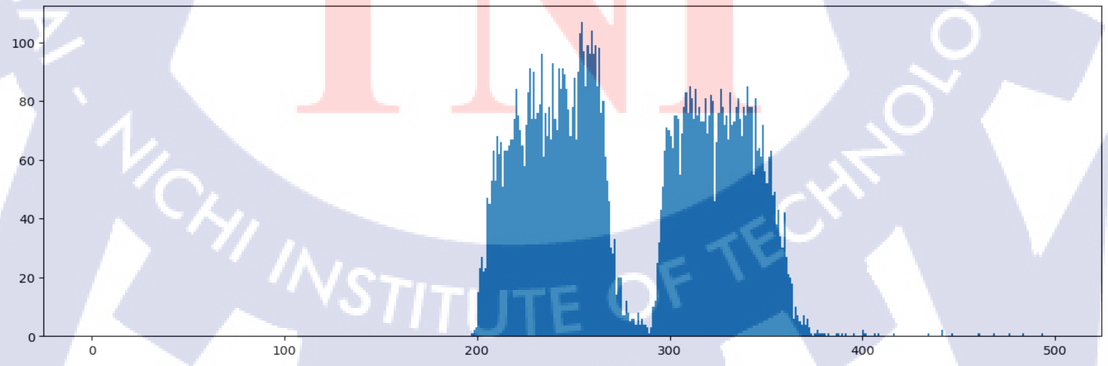


Figure 4.8 Histogram of the end-to-end execution time results of the cloud server (NVIDIA A10G + TensorRT + Internet)

Even though the standard deviation (S.D.) of both inference time and end-to-end execution time are higher, The histogram shows that the end-to-end execution time of the edge device environment is the most stable and predictable, followed by the edge server environment, and the cloud environment. The reason that S.D. of the edge device environment is higher than the edge server and cloud environment and not going the same direction as the histogram is because the edge device environment has a maximum record that was an outline.

4.3 Accuracy

Table 4.3 shows the training, validation, and testing accuracy of both image classifier models. In the table, it was found that Edge Device has 99% training accuracy, 98.4 validation accuracy, and 99.75% testing accuracy, Edge Server has 97.7% training accuracy, 96.7% validation accuracy, and 98.27% testing accuracy, and Cloud Server has 97.7% training accuracy, 96.7% validation accuracy, and 95.46% testing accuracy.

Table 4.3. Accuracy

ML Environment	**Accuracy (Percentage)**		
	Training	**Validation**	**Testing**
Core ML	99.0%	98.4%	99.74%
Keras	97.7%	96.7%	98.27%

Table 4.4 shows the maximum iteration configuration of the Core ML and Keras image classifier models and the convergence iteration. Training in Core ML was configured to 25 max iterations but the model converged at iteration 11 as visualized in Figure 4.9. Training in Keras was configured to 50 max iterations, and the early stop patience was configured to observe for 8 continuously unimproved iterations, the model converged at iteration 34 as visualized in Figure 4.10.

Table 4.4 Training Results

ML Framework	**Iterations / Epochs**	
	Maximum	**Convergence**
Core ML	25	11
Keras	50	34

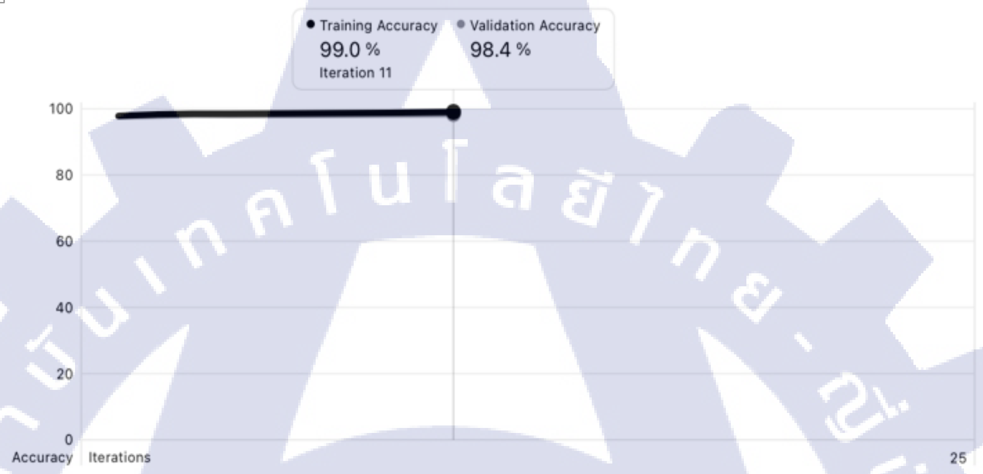


Figure 4.9. Core ML Model Training and Validation Accuracy

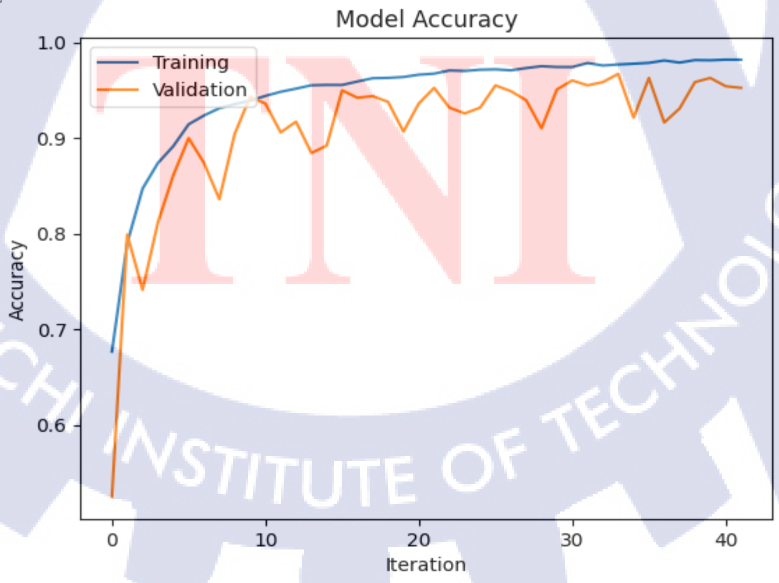


Figure 4.10. Keras Model Training and Validation Accuracy

Table 4.5 shows the confusion matrix of the Core ML model tested on the edge device, the Keras model converted to TensorRT tested on the edge server, and the Keras model converted to TensorRT tested on the cloud server.

Table 4.5. Confusion Matrix

Actual Class	**Predicted Class**		**Total**
	Dog	**Cat**	
Edge Device – Core ML			
Dog	4718	21	4739
Cat	4	5148	5153
Edge Server and Cloud Server – Keras – TensorRT			
Dog	4600	139	4739
Cat	32	5121	5153

Table 4.6 shows the precision, recall, and F1-score calculated from the confusion matrix. In the table, it was found that Precision for

Table 4.6. Model Performance

Actual Class	**Model Performance**		
	Precision	**Recall**	**F1-Score**
Edge Device – Core ML			
Dog	99.8%	100%	0.999
Cat	100%	100%	1
Edge Server and Cloud Server – Keras – TensorRT			
Dog	99.3%	97.1%	0.982
Cat	97.4%	99.4%	0.984

4.4 Confidence Score

Table 4.7 represents the minimum, average, maximum, and standard deviation of the confidence score of the inferences. It was found that the edge device has the lowest standard deviation and highest average. The minimum confidence score is around 50% across all environments. The histogram of confidence score results is shown in Figure 4.11.

Table 4.7. Confidence Score Stats

Environment	**Confidence Score Stats (Percentage)**			
	Min	**Average**	**Max**	**S.D.**
Core ML	50.34%	99.9%	100%	1.67%
Keras	50.01%	98.98%	100%	5.16%

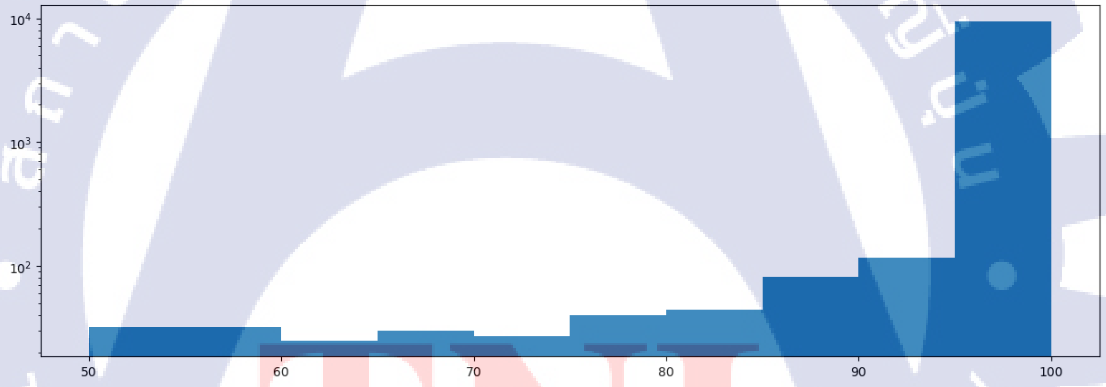


Figure 4.11. Histogram of confidence score results

4.5 Resource Usage

Table 4.8 shows the resource usage of image classification on the edge device, edge server, and cloud server. On the edge device, 14% to 17% of CPU was utilized, about 100 MB to 400 MB of unified memory was utilized, and it has a high to very high energy impact. On the edge server, an average of 1% of 12 vCPU was utilized, 5.74 GB memory was utilized, 5,328 MB of GPU memory was utilized, and 42 watts of energy was used. On the cloud server, about 8% to 13% of 4 vCPU was utilized, 8.98 GB of memory was utilized, and 12,283 MB of GPU memory was utilized.

Table 4.8. Resource Usage

Resource	**Usage**		
	Edge Device	**Edge Server**	**Cloud Server**
CPU	14 ~ 17%	1~3% of 12 vCPU	8~13% of 4 vCPU
Memory	100 ~ 400 MB	5.74 GB	8.96 GB
GPU Memory	Unified	5,328 MB	12,283 MB
Energy Impact / Usage	High ~ Very High	42 Watts	65 Watts

4.6 Data Transfer

The data transfer was statistically measured by the size of the images used for testing (the Animal Faces-HQ dataset) but did not include the HTTP payload and other overheads. The images uploaded to the cloud are high-quality JPEGs resized to 256-by-256 pixels, but the original size was 512-by-512 pixels. The color profile is sRGB IEC61966-2.1.

Table 4.9 shows the minimum, average, maximum, and standard deviation of the size of the images of the Animal Faces-HQ dataset. The histograms of the size of the images are shown in Figure 4.12 and Figure 4.13.

Table 4.9. Data Transfer

Resolution	**Image Size (Kilobytes)**			
	Min	**Average**	**Max**	**S.D.**
256 x 256	3.49	13.74	32.46	3.42
512 x 512	13.95	42.32	112.68	12.05

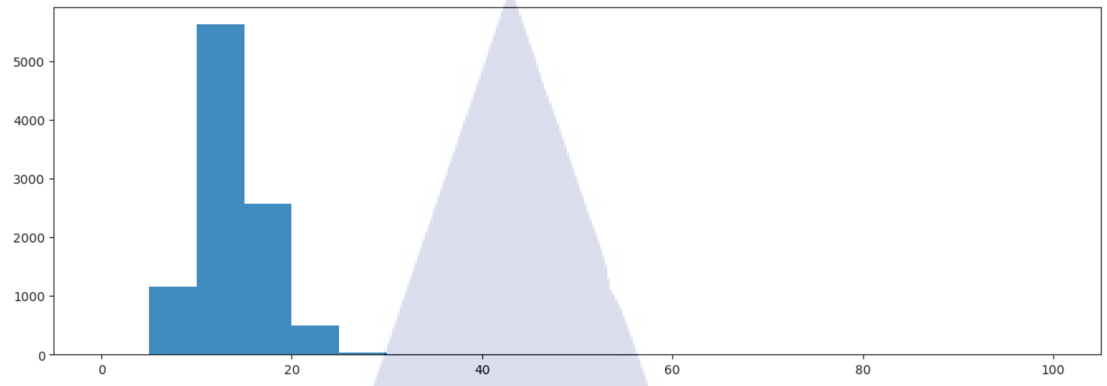


Figure 4.12. Histogram of file size of test images in resolution 256-by-256 px sRGB IEC61966-2.1 JPEG

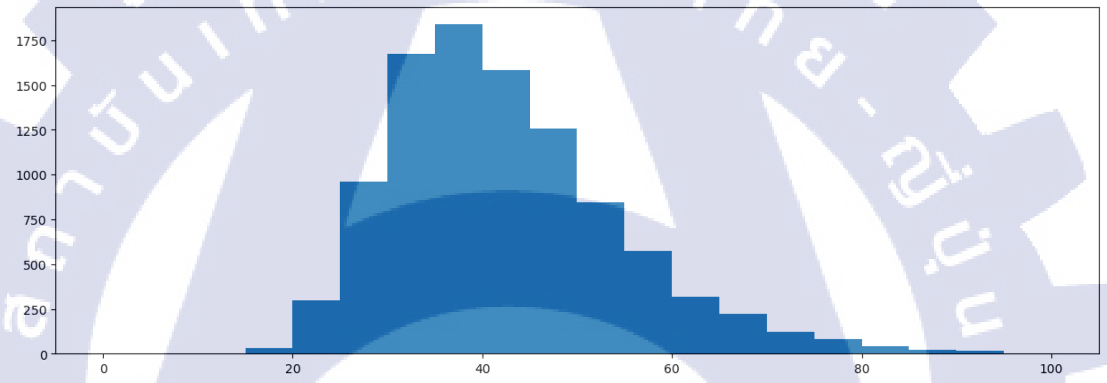


Figure 4.13. Histogram of file size of test images in resolution 512-by-512 px sRGB IEC61966-2.1 JPEG

Chapter 5
Conclusion and Discussion

The research study on performance analysis of image classification between Edge and Cloud Computing presented the following conclusion as outlined below.

5.1 Conclusion

5.2 Discussion

5.1 Conclusion

In this work, the researcher simulated the environments and application of image classification on edge and cloud computing based on hardware, ML framework, and network to evaluate and analyze the performance including inference time, end-to-end execution time, accuracy, confidence score, resource usage, and data transfer.

It was found that inferences on the cloud were done faster compared to at the edge. However, when considering the end-to-end execution time, the network delays were significantly affecting the end-to-end execution time of image classification. In our case, the distance between the application and the cloud has a significant impact on the propagation delay, which leads to higher execution delay in the cloud environment. The cloud GPU instance was running in Tokyo while the application was running in Bangkok. The network delay measured by the ping test was 100 (±1) milliseconds. Thus, executing image classification at the edge is significantly faster. The size of data uploaded to the cloud has no significant impact on high-speed internet connections such as FTTx, broadband, and 4G/5G cellular networks (without a Fair-Usage Policy or a bandwidth limit policy applied).

The accuracy of the image classifier model created using CoreML is slightly greater than the model created using Keras/TensorFlow. Based on testing, CoreML has 99.75% accuracy, and Keras/TensorFlow has 95.46%. The average confidence score of both models was 99.9% and 98.0%, respectively. This is because CoreML works best with images of real-world objects since it has an image feature extractor pre-trained by millions of images, and our datasets are also composed of common real-world objects.

5.2 Discussion

Offloading image classification to the edge eliminates the network delays required for federating data to process on the cloud (or the remote server). This makes image classification faster, which is impactful for time-critical applications. It also reduces cloud resources and subscription costs.

Even though the cloud is close to the edge, for instance, if the cloud and the edge were placed in the same city and connected via FTTx, the network propagation delay might range from 1 to 10 ms, leading to an insignificant difference of end-to-end execution time between the edge and the cloud. However, offloading image classification to the edge offers the benefit of cloud resource reduction while still maintaining the application's speed and experience.

When considering offloading the image classification to the edge, the performance and availability of the edge devices, the scenarios, and the use cases must be considered.

Besides the execution speed, the model accuracy must also be considered. Our experiment dataset was controlled by real-world dog and cat images, so CoreML has an advantage. However, the model accuracy must be re-evaluated every time the dataset changes.

Inference time and end-to-end execution time have a major impact on the speed of the image classification system or application and on user experiences. Accuracy impacts the correctness of prediction. The confidence score could be used as a threshold in the application to display the prediction label confidently. Resource usage shows how much resource the image classification consumed, including CPU, memory, and energy, which will be related to the power consumption. Data transfer impacts how much network bandwidth will be used.

References

References

[1] N. Monburinon et al., “A novel hierarchical edge computing solution based on deep learning for distributed image recognition in IoT systems,” *2019 4th International Conference on Information Technology (InCIT)*, Bangkok, Thailand, October 24-25, 2019, pp. 294-299, doi: 10.1109/INCIT.2019.8912138.

[2] I. Zualkerman et al., “An IoT system using deep learning to classify camera trap images on the edge,” *Computers 2022,* vol.11, no.1, pp. 13, January 2022, doi: 10.3390/computers11010013.

[3] A. Olsen et al., “DeepWeeds: A multiclass weed species image dataset for Deep Learning,” *Scientific Reports,* vol. 9, no. 2058, pp. 1-12, February 2019, doi: 10.1038/s41598-018-38343-3.

[4] G. P. A. Stalin and S. Anand, “Intelligent smart home security system: A deep learning approach,” *2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC)*, Hyderabad, India, September 16-18, 2022, pp. 438-444, doi: 10.1109/R10-HTC54060.2022.9929516.

[5] S. Khan et al., “Pedestrian traffic lights classification using transfer learning in smart city application,” *2021 13th International Conference on Communication Software and Networks (ICCSN)*, Chongqing, China, June 4-7, 2021, pp. 352-356, doi: 10.1109/ICCSN52437.2021.9463615.

[6] E. Charteros and I. Koutsopoulos, “Edge computing for having an edge on cancer treatment: A mobile app for breast image analysis,” *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, Dublin, Ireland, June 7-11, 2020, pp. 1-6, doi: 10.1109/ICCWorkshops49005.2020.9145219.

[7] K. Muhammad et al., “Deep learning for multigrade brain tumor classification in smart healthcare systems: A prospective survey,” in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 507-522, February 2021, doi: 10.1109/TNNLS.2020.2995800.

[8] K. Iqtidar et al., “Image pattern analysis towards classification of skin cancer through dermoscopic images,” *2020 First International Conference of Smart Systems and Emerging Technologies (SMARTTECH)*, Riyadh, Saudi Arabia, November 3-5, 2020, pp. 208-213, doi: 10.1109/SMART-TECH49988.2020. 00055.

[9] P. Rujakom et al., “Retail management on mobile application using product classification,” *2022 19th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, Prachuap Khiri Khan, Thailand, May 24-27, 2022, pp. 1-5, doi: 10.1109/ECTI-CON54298.2022.9795504.

[10] A. Savit and A. Damor, “Revolutionizing retail stores with computer vision and edge AI: A novel shelf management system,” *2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, Salem, India, May 4-6, 2023, pp. 69-74, doi: 10.1109/ICAAIC56838.2023.10140947.

[11] Apple, *“iPhone 12 – Technical Specification,”* support.apple.com [Online]. Available: https://support.apple.com/kb/SP830. [Accessed: February 22, 2023].

[12] Apple, *“iPhone 12 Pro – Technical Specification,”* support.apple.com [Online]. Available: https://support.apple.com/kb/SP831. [Accessed: February 22, 2023].

[13] Apple, *“iPad Pro, 11-Inch (3rd Generation) - Technical Specifications,”* support.apple.com [Online]. Available: https://support.apple.com/en-us/111897. [Accessed: May 14, 2025].

[14] NVIDIA, *“Jetson Orin Nano Super Development Kit,”* nvidia.com [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/nano-super-developer-kit. [Accessed: May 1, 2025].

[15] Apple, *“iPhone 13 Pro – Technical Specification,”* support.apple.com [Online]. Available: https://support.apple.com/kb/SP852. [Accessed: February 22, 2023].

[16] Apple, *“iPhone 14 Pro – Technical Specification,”* support.apple.com [Online]. Available: https://support.apple.com/kb/SP875. [Accessed: February 22, 2023].

[17] Apple, *“iPhone 15 Pro – Tech Specs,”* support.apple.com [Online]. Available: https://support.apple.com/en-us/111829. [Accessed: May 14, 2025].

[18] NVIDIA, *"GeForce RTX 3070 Family,"* nvidia.com [Online]. Available: https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3070-3070ti. [Accessed: May 14, 2025].

[19] Amazon, *"Amazon EC2 G5 Instances,"* Amazon Web Services [Online]. Available: https://aws.amazon.com/ec2/instance-types/g5. [Accessed: May 14, 2025].

[20] NVIDIA, *"NVIDIA A10 Tensor Core GPU,"* nvidia.com [Online]. Available: https://www.nvidia.com/en-us/data-center/products/a10-gpu. [Accessed: May 14, 2025].

[21] Apple, *"Create ML Overview – Machine Learning,"* developer.apple.com [Online]. Available: https://developer.apple.com/machine-learning/create-ml. [Accessed: Febuary 22, 2023].

[22] Apple, *"Create ML,"* developer.apple.com [Online]. Available: https://developer.apple.com/documentation/createml. [Accessed: Febuary 22, 2023].

[23] Apple, *"Core ML Overview – Machine Learning,"* developer.apple.com [Online]. Available: https://developer.apple.com/machine-learning/core-ml/. [Accessed: February 22, 2023].

[24] Apple, *"Core ML,"* Apple Developer Documentation [Online]. Available: https://developer.apple.com/documentation/coreml. [Accessed: February 22, 2023].

[25] M. Abadi el al., *"TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,"* arxiv.org [Online]. Available: https://doi.org/10.48550/arXiv.1603.04467. [Accessed: Febuary 22, 2023].

[26] TensorFlow, *"TensorFlow,"* tensorflow.org [Online]. Available: https://www.tensorflow.org. [Accessed: February 22, 2023].

[27] TensorFlow, *"Tensorflow/Tensorflow: An Open Source Machine Learning Framework for Everyone,"* GitHub [Online]. Available: https://github.com/tensorflow/tensorflow. [Accessed: February 22, 2023].

[28] Keras, *"Keras: Deep Learning for Humans,"* keras.io [Online]. Available: https://keras.io/. [Accessed: February 22, 2023].

[29] NVIDIA, *"NVIDIA TensorRT,"* developer.nvidia.com [Online]. Available: https://developer.nvidia.com/tensorrt. [Accessed: February 22, 2023].

[30] W. Shi et al., "Edge computing: Vision and challenges," in *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637-646, October 2016, doi: 10.1109/JIOT.2016.2579198.

[31] Wikipedia, *"Graphic Processing Unit,"* wikipedia.org [Online]. Available: https://en.wikipedia.org/wiki/Graphics_processing_unit. [Accessed: February 22, 2023].

[32] Wikipedia *"System on a Chip,"* wikipedia.org [Online]. Available: https://en.wikipedia.org/wiki/System_on_a_chip. [Accessed: May 9, 2025].

[33] J. Chen and X. Ran, "Deep learning with edge computing: A review," in *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655-1674, August 2019, doi: 10.1109/JPROC.2019.2921977.

[34] GeeksForGeeks, *"What is Image Classification?,"* geeksforgeeks.org [Online]. Available: https://www.geeksforgeeks.org/what-is-image-classification. [Accessed: May 14, 2025].

[35] Google AI for Developers, *"LiteRT,"* ai.google.dev [Online]. Available: https://ai.google.dev/edge/litert. [Accessed: May 9, 2025].

[36] S. R. Reza et al., "Inference performance comparison of convolutional neural networks on edge devices," in *6th EAI International Conference, SmartCity360°*, Virtual Event, December 2-4, 2020, pp. 323-335, doi: 10.1007/978-3-030-76063-2_23.

[37] A. Ignatov et al., *"AI Benchmark: All about Deep Learning on Smartphones in 2019,"* arxiv.org [Online]. Available: https://doi.org/10.48550/arXiv.1910.06663. [Accessed: February 22, 2023].

[38] H. Smolic, *"AI and Machine Learning Glossary,"* Graphic Note [Online]. Available: https://graphite-note.com/comprehensive-ai-and-machine-learning-glossary. [Accessed: February 22, 2023].

[39] Omardonia, *"What is a Confusion Matrix and How is it Used in Evaluating Model Performance,"* Medium [Online]. Available: https://levelup.gitconnected.com/what-is-a-confusion-matrix-and-how-is-it-used-in-evaluating-model-performance-b3f01143c981. [Accessed: February 22, 2023].

[40] Apple, *"Swift,"* developer.apple.com [Online]. Available: https://developer.apple.com/swift. [Accessed: May 14, 2025].

[41] Tiangolo, *"FastAPI,"* fastapi.tiangolo.com [Online]. Available: https://fastapi.tiangolo.com. [Accessed: May 14, 2025].

Appendices

Appendix A
Source Code – iOS Swift App

ContentView.swift

```
import SwiftUI
import UIKit

struct ContentView: View {

    var body: some View {
        ImageClassifierView()
    }
}

#Preview {
    ContentView()
}
```

Views/ImageClassifierView.swift

```
import Foundation
import SwiftUI
import Photos
import os

struct ImageClassifierView2: View {

    private let logger = Logger(
        subsystem: Bundle.main.bundleIdentifier!,
        category: String(describing: ContentView.self)
    )

    /// A predictor instance that uses Vision and Core ML to generate prediction
strings from a photo.
    private let imageClassificationService = ImageClassificationService()
```

```
let encoder = JSONEncoder()

@ObservedObject var testScenarioViewModel = TestScenarioViewModel()

@State private var albumName: String = ""

@State private var albums: [PHAssetCollection] = []
@State private var openPicker = false

@State private var inferenceTimeElapsed: Double = 0
@State private var inferenceTimeStart: CFAbsoluteTime? = nil

@State private var e2eTimeElapsed: Double = 0
@State private var e2eTimeStart: CFAbsoluteTime? = nil

@State private var successCount: Int = 0
@State private var sumInferenceTime: Double = 0
@State private var minInferenceTime: Double?
@State private var maxInferenceTime: Double?
@State private var showPerformanceResult: Bool = false

@State private var correctPredictionCount: Int = 0
@State private var incorrectPredictionCount: Int = 0

@State private var sumConfidenceScore: Double = 0

@State private var fileName: String = ""

var body: some View {
  VStack {
      NavigationView {
        Form {
          Section("Config") {
            Picker("Album", selection: $albumName) {
              Text("Recents").tag("")
              ForEach(albums, id: \.self) { album in
                Text(album.localizedTitle ?? "")
                  .tag(album.localizedTitle ?? "")
              }
            }
            .pickerStyle(.navigationLink)
          }
          Section(header: Text("Test Information")) {
            Text("ID: " + getTestId())
            Text("Name: " + getTestName())
            Button(action: testScenarioViewModel.createTestScenario) {
              Label("Create Test Scenario", systemImage: "plus")
            }
          }
          Section("Run") {
            Button(action: submit) {
              Text("Run")
            }
          }
          if (showPerformanceResult) {
            Section("Result") {
              Text("Done \(successCount) images.")
              if (successCount > 0) {
                Text("Average Inference Time \(sumInferenceTime / 
Double(successCount)) ms.")
                Text("Max Inference Time: \(maxInferenceTime ?? 0) ms")
```

```
                        Text("Min Inference Time: \(minInferenceTime ?? 0) ms")
                        Text("Correct Prediction: \(correctPredictionCount)")
                        Text("Incorrect Prediction: \(incorrectPredictionCount)")
                        Text(String(format: "Accuracy: %.2f%%",
Double(correctPredictionCount) / Double(successCount) * 100))
                     }
                  }
               }
            }
            .navigationTitle("Classify Images")
         }
      }
      .onAppear() {
         loadAlbums()
      }
   }

   func getTestId() -> String {
      guard let createdTestScenario = testScenarioViewModel.createdTestScenario
else {
         return "Undefined"
      }
      return String(createdTestScenario.id)
   }

   func getTestName() -> String {
      return testScenarioViewModel.createdTestScenario?.name ?? "Undefined"
   }

   func submit() {
      showPerformanceResult = true
```

```
        loadImagesFromAlbum(albumName: albumName)
    }

    func loadImagesFromAlbum(albumName: String) -> Void {
        let fetchOptions = PHFetchOptions()
        fetchOptions.predicate = NSPredicate(format: "title = %@", albumName)

        let albums = PHAssetCollection.fetchAssetCollections(with: .album,
subtype: .any, options: fetchOptions)

        if let album = albums.firstObject {
            let assets = PHAsset.fetchAssets(in: album, options: nil)

            for index in 0..<assets.count {
                let asset = assets[index]

                // Now you can do something with each asset, like load its image
                loadImage(for: asset)
            }
        }
    }

    func loadImage(for asset: PHAsset) {
        e2eTimeStart = CFAbsoluteTimeGetCurrent()
        let imageManager = PHImageManager.default()
        let requestOptions = PHImageRequestOptions()
        requestOptions.deliveryMode = .highQualityFormat
        requestOptions.isSynchronous = false

        imageManager.requestImage(for: asset, targetSize: CGSize(width: 100, height:
100), contentMode: .aspectFill, options: requestOptions) { (image, _) in
```

```
            if let image = image {
                let resources = PHAssetResource.assetResources(for: asset)
                if let originalFilename = resources.first?.originalFilename {
                    logger.info("Image file name: \(originalFilename)")
                    fileName = originalFilename
                } else {
                    logger.info("Image file name not available")
                }
                // Do something with the image, e.g., display it
//                print("Loaded image: \(image)")
                classifyImage(image: image)
                usleep(200 * 1000) // Sleep for milliseconds
            }
        }
    }

    func classifyImage(image: UIImage) {
        logger.info("Begin Classification")
        logger.info("Image Selected")
        do {
            logger.info("Start Inference")
            inferenceTimeStart = CFAbsoluteTimeGetCurrent()
            try imageClassificationService.classify(
                for: image,
                completionHandler: imageClassificationHandler
            )

        } catch {
            logger.error("Vision was unable to make a
prediction...\n\n\(error.localizedDescription)")
        }
    }

    func imageClassificationHandler(_ predictions: [Prediction]?) {
        if let predictions {
//            for prediction in predictions {
//                print(prediction)
//            }

            if let prediction = predictions.first {
                let inferenceTimeInMillisec = (CFAbsoluteTimeGetCurrent() -
inferenceTimeStart!) * 1000
                logger.info("\(prediction.classification.capitalized)
(\(prediction.confidence * 100)%)")
                logger.info("Done Inference, Time Elapsed: \(inferenceTimeInMillisec)
ms")
                let e2eTimeInMillisec = (CFAbsoluteTimeGetCurrent() -
inferenceTimeStart!) * 1000
                successCount += 1
                if (albumName.lowercased() == prediction.classification.lowercased()) {
                    correctPredictionCount += 1
                }
                else {
                    incorrectPredictionCount += 1
                }
                if let test = testScenarioViewModel.createdTestScenario {
                    createRecord(record: Record(
                        test: test.id,
                        inferenceTime: inferenceTimeInMillisec,
                        executionTime: e2eTimeInMillisec,
                        imageFileName: fileName,
                        actual: albumName,
                        prediction: prediction.classification.capitalized,
                        accurate: albumName.lowercased() ==
prediction.classification.lowercased(),
                        confidence: prediction.confidence,
                        dataTransfer: 0)
                    )
                }
                else {
                    logger.warning("Test ID not defined.")
                }
                submitInferenceTimeElapsed(timeElapsed: inferenceTimeInMillisec)
            }

        }
    }

    func submitInferenceTimeElapsed(timeElapsed: Double) {
        sumInferenceTime += timeElapsed
        if let max = maxInferenceTime {
            if (timeElapsed > max) {
                maxInferenceTime = timeElapsed
            }
        }
        else {
            maxInferenceTime = timeElapsed
        }
        if let min = minInferenceTime {
            if (timeElapsed < min) {
                minInferenceTime = timeElapsed
            }
        }
        else {
            minInferenceTime = timeElapsed
        }
    }

    func loadAlbums() {
        openPicker = true
        let fetchOptions = PHFetchOptions()
        let albums = PHAssetCollection.fetchAssetCollections(with: .album,
subtype: .any, options: fetchOptions)
        albums.enumerateObjects { (collection, _, _) in
            self.albums.append(collection)
        }

    }

    func createRecord(record: Record) {
        if let test = testScenarioViewModel.createdTestScenario {
            print(test)
            let request = ApiService.post(path: "records", data: [
                "test": test.id,
                "inferenceTime": record.inferenceTime,
                "executionTime": record.executionTime,
                "imageFileName": record.imageFileName,
                "actual": record.actual,
                "prediction": record.prediction,
                "accurate": record.accurate,
                "confidence": record.confidence,
                "dataTransfer": record.dataTransfer,
            ])
        let session = URLSession(configuration: URLSessionConfiguration.default)

        let task = session.dataTask(with: request) { (data, response, error) in
          guard let data = data, error == nil else {
            // handle error
            logger.error("Unexpected Error")
            return
          }

          if let httpStatus = response as? HTTPURLResponse, ![200,
201].contains(httpStatus.statusCode) {
            // handle non-200 status code
            print(data)
            logger.error("Failed 2")
            return
          }

          let responseString = String(data: data, encoding: .utf8)
          // handle response
          logger.info("Successfully Stored Prediction Result")
        }

        task.resume()
      }
    }
}
```

Models/Prediction.swift

```
import Foundation

struct Prediction: Codable {
    /// The name of the object or scene the image classifier recognizes in an image.
    let classification: String

    /// The image classifier's confidence as a percentage string.
    ///
    /// The prediction string doesn't include the % symbol in the string.
    let confidence: Float
}
```

Models/TestScenario.swift

```
import Foundation

struct TestScenario: Codable {
    let id: Int
    let name: String
    let description: String
    let startedTime: String
}
```

Models/Record.swift

```
import Foundation

struct Record {
    let test: Int
    let inferenceTime: Double
    let executionTime: Double
    let imageFileName: String
    let actual: String
    let prediction: String
    let accurate: Bool
    let confidence: Float
```

```
  let dataTransfer: Double
}
```

ViewModels/TestScenarioViewModel.swift

```
import Foundation

class TestScenarioViewModel: ObservableObject {

  @Published var createdTestScenario: TestScenario?

  private func getCurrentTimestampInMilliseconds() -> Int {
    return Int(Date().timeIntervalSince1970 * 1000)
  }

  func createTestScenario() {
    let session = URLSession(configuration: URLSessionConfiguration.default)
    let request: URLRequest = ApiService.post(path: "tests", data: [
      "name": DeviceConfig.getDeviceId().uppercased(),
      "description": DeviceConfig.getDeviceModelName(),
      "device": DeviceConfig.getDeviceId(),
      "startedTime": getCurrentTimestampInMilliseconds(),
      "rate": 1
    ])

    let task = session.dataTask(with: request) { (data, response, error) in
      if let error = error {
        print("Error: \(error)")
      }

      if let data = data {
        DispatchQueue.main.async {
```

```
                    do {
                        self.createdTestScenario = try
JSONDecoder().decode(TestScenario.self, from: data)
                    } catch {
                        print("Error decoding JSON: \(error)")
                    }
                    print(self.createdTestScenario ?? "Default Value for
createdTestScenario")
                }
            }
        }

        task.resume()
    }

}
```

Config/Config.swift

```
import Foundation

struct Config {
    public static let apiBaseUrl = "http://API_BASE_URL"
}
```

Config/DeviceConfig.swift

```
import Foundation
import UIKit

class DeviceConfig {

    public static func getDeviceModel() -> String? {
```

```
    var systemInfo = utsname()
    uname(&systemInfo)
    let model = withUnsafePointer(to: &systemInfo.machine) {
        $0.withMemoryRebound(to: CChar.self, capacity: 1) {
            ptr in String.init(validatingUTF8: ptr)
        }
    }

    return model
}

public static func getDeviceModelName() -> String {
    guard let model: String = getDeviceModel() else {
        return "Unidentified Device Model"
    }
    return [
        "iPhone16,2": "iPhone 15 Pro Max",
        "iPhone16,1": "iPhone 15 Pro",
        "iPhone15,3": "iPhone 14 Pro Max",
        "iPhone15,2": "iPhone 14 Pro",
        "iPhone13,4": "iPhone 12 Pro Max",
        "iPhone13,3": "iPhone 12 Pro Max"
    ][model] ?? "Unlisted Device Model"
}

public static func getDeviceId() -> String {
    guard let model: String = getDeviceModel() else {
        return "ed-03"
    }
    return [
        "iPhone16,2": "ed-03", // iPhone 15 Pro and iPhone 15 Pro Max
        "iPhone15,2": "ed-02", // iPhone 14 Pro and iPhone 14 Pro Max
        "iPhone13,2": "ed-01" // iPhone 12 Pro and iPhone 12 Pro Max
      ][model]  ?? "Unlisted Device Model"
  }

}
```

Services/ImageClassificationService.swift

```
import Foundation
import Vision
import UIKit
import os

class ImageClassificationService {

  private let logger = Logger(
      subsystem: Bundle.main.bundleIdentifier!,
      category: String(describing: ImageClassificationService.self)
  )

  /// A common image classifier instance that all Image Predictor instances use to
generate predictions.
  ///
  /// Share one ``VNCoreMLModel`` instance --- for each Core ML model file ---
across the app,
  /// since each can be expensive in time and resources.
  static let imageClassifier = createImageClassifier()



  /// The function signature the caller must provide as a completion handler.
  typealias ImagePredictionHandler = (_ predictions: [Prediction]?) -> Void
```

```
/// A dictionary of prediction handler functions, each keyed by its Vision request.
private var predictionHandlers = [VNRequest: ImagePredictionHandler]()


private static func createImageClassifier() -> VNCoreMLModel {
    // Use a default model configuration.
    let defaultConfig = MLModelConfiguration()

    // Create an instance of the image classifier's wrapper class.
    let imageClassifierWrapper = try? PetImages2_PetFinderMS(configuration:
defaultConfig)

    guard let imageClassifier = imageClassifierWrapper else {
        fatalError("App failed to create an image classifier model instance.")
    }

    // Get the underlying model instance.
    let imageClassifierModel = imageClassifier.model

    // Create a Vision instance using the image classifier's model instance.
    guard let imageClassifierVisionModel = try? VNCoreMLModel(for:
imageClassifierModel) else {
        fatalError("App failed to create a `VNCoreMLModel` instance.")
    }
    return imageClassifierVisionModel
}
```

```
/// Generates a new request instance that uses the Image Predictor's image
classifier model.
private func createImageClassificationRequest() -> VNImageBasedRequest {

    // Create an image classification request with an image classifier model.
    let imageClassificationRequest = VNCoreMLRequest(
        model: ImageClassificationService.imageClassifier,
        completionHandler: visionRequestHandler
    )

    imageClassificationRequest.imageCropAndScaleOption = .centerCrop

    return imageClassificationRequest
}


/// The completion handler method that Vision calls when it completes a request.
/// - Parameters:
///   - request: A Vision request.
///   - error: An error if the request produced an error; otherwise `nil`.
///
///   The method checks for errors and validates the request's results.
/// - Tag: visionRequestHandler
private func visionRequestHandler(_ request: VNRequest, error: Error?) {
    // Remove the caller's handler from the dictionary and keep a reference to it.
    guard let predictionHandler = predictionHandlers.removeValue(forKey:
request) else {
        fatalError("Every request must have a prediction handler.")
    }

    // Start with a `nil` value in case there's a problem.
    var predictions: [Prediction]? = nil

    // Call the client's completion handler after the method returns.
    defer {
      // Send the predictions back to the client.
      predictionHandler(predictions)
    }

    // Check for an error first.
    if let error = error {
      print("Vision image classification error...\n\n\(error.localizedDescription)")
      return
    }

    // Check that the results aren't `nil`.
    if request.results == nil {
      print("Vision request had no results.")
      return
    }

    // Cast the request's results as an `VNClassificationObservation` array.
    guard let observations = request.results as? [VNClassificationObservation]
else {
        // Image classifiers, like MobileNet, only produce classification
observations.
        // However, other Core ML model types can produce other observations.
        // For example, a style transfer model produces
`VNPixelBufferObservation` instances.
        print("VNRequest produced the wrong result type: \(type(of:
request.results)).")
        return
        }

        // Create a prediction array from the observations.
        predictions = observations.map {
            // Convert each observation into an `Prediction` instance.
            observation in Prediction(classification: observation.identifier, confidence:
observation.confidence)
        }
    }

    // Generates an image classification prediction for a photo.
    /// - Parameter photo: An image, typically of an object or a scene.
    /// - Tag: makePredictions
    func classify(for photo: UIImage, completionHandler: @escaping
ImagePredictionHandler) throws {
        logger.info("Classifying")
        let orientation = CGImagePropertyOrientation(rawValue:
UInt32(photo.imageOrientation.rawValue))

        guard let photoImage = photo.cgImage else {
            fatalError("Photo doesn't have underlying CGImage.")
        }

        let imageClassificationRequest = createImageClassificationRequest()
        //imageClassificationRequest.usesCPUOnly = true
        predictionHandlers[imageClassificationRequest] = completionHandler

        let handler = VNImageRequestHandler(cgImage: photoImage, orientation:
orientation!)
        let requests: [VNRequest] = [imageClassificationRequest]
        // Start the image classification request.
        try handler.perform(requests)
    }
}
```

Services/ApiService.swift

```
import Foundation

class ApiService {

    public static func post(path: String, data: [String: Any]) -> URLRequest {
        var request = URLRequest(url: URL(string: Config.apiBaseUrl + "/" + path)!)
        request.httpMethod = "POST"
        request.setValue("application/json", forHTTPHeaderField: "Content-Type")
        do {
            request.httpBody = try JSONSerialization.data(withJSONObject: data)
        } catch {
            print("Error encoding JSON: \(error)")
        }
        return request
    }
}
```

ImageClassifierApp.swift

```
import SwiftUI

@main
struct ImageClassifierApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

Appendix B

Source Code – Python FastAPI App

main.py

```
import uvicorn
from fastapi import FastAPI, File, UploadFile, HTTPException
from typing import Literal
from PIL import Image
import numpy as np
from io import BytesIO
import tensorflow as tf
from pydantic import BaseModel
import time


gpu_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(gpu_devices[0], True)
tf.config.experimental.set_virtual_device_configuration(
    gpu_devices[0],
    [tf.config.experimental.VirtualDeviceConfiguration(memory_limit=512)])

class PetImagesPrediction(BaseModel):
    dog: float
    cat: float


description = """
Classify cat and dog images.
"""


app = FastAPI(
    title="PetImages API",
    version="1.0.0",
    description=description,
)

TFTRT_MODEL_DIR = "./models/tensorrt/20231129_053542_fp32"
root_tftrt = tf.saved_model.load(TFTRT_MODEL_DIR)
func_tftrt = root_tftrt.signatures["serving_default"]
print("Model loaded.")

def predict_petimages(img: Image.Image, model: Literal["native", "tftrt"]) -> dict:
   image_size = (180, 180)
   img = img.resize(image_size)
   img = np.array(img, dtype=np.float32)
   img = tf.expand_dims(img, 0)
   if model == "native":
      model_func = func_native
   elif model == "tftrt":
      model_func = func_tftrt
   else:
      return "Only native and tftrt are provided."
   prediction = model_func(img)
   prediction = prediction["dense_6"]
   score = float(prediction[0][0])
   return dict(dog=score, cat=1 - score)


def read_image_file(file) -> Image.Image:
   image = Image.open(BytesIO(file))
   return image

@app.post("/predict")
async def predict_petimage_tftrt(file: UploadFile = File(...)) ->
PetImagesPrediction:
   extension = file.filename.split(".")[-1] in ("jpg", "jpeg", "png")
   if not extension:
      return "Image must be jpg or png format!"
   img = read_image_file(await file.read())
   prediction = predict_petimages(img, model="tftrt")
   return prediction

def evaluate(img: Image.Image, model: Literal["native", "tftrt"]) -> dict:
   print("Evaluating Image")
   image_size = (180, 180)
   img = img.resize(image_size)
   img = np.array(img, dtype=np.float32)
   img = tf.expand_dims(img, 0)
   model_func = func_tftrt
   start_time = time.perf_counter()
   prediction = model_func(img)
   print(prediction)
   prediction = prediction["dense"]
   score = float(prediction[0][0])
   inference_time = (time.perf_counter() - start_time) * 1000 # This will output in
milliseconds
   return dict(
      inference_time=inference_time,
      result=dict(dog=score, cat=1 - score)
   )

@app.post("/evaluate/tensorrt")
async def evaluate_tensorrt(file: UploadFile = File(...)) -> PetImagesPrediction:
   print("Recieved HTTP Request - POST /evaluate/tensorrt")
   valid_extension = file.filename.split(".")[-1] in ("jpg", "jpeg", "png")
   if not valid_extension:
      raise HttpException(status_code=400, detail="Image must be jpg or png
format!")
   img = read_image_file(await file.read())
   result = evaluate(img, "tftrt")
   return result



@app.get("/")
def read_root():
   return dict(
      message=""" Visit /docs for more details."""
   )



if __name__ == "__main__":
   uvicorn.run(app)
```

Appendix C
Source Code – API Client

App.java

```
package dev.babebbu.academic.thesis.master;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.hc.client5.http.HttpResponseException;
import org.apache.hc.client5.http.fluent.Request;
import org.apache.hc.client5.http.fluent.Response;
import org.apache.hc.core5.http.ContentType;

import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Date;
import java.util.LinkedList;
import java.util.concurrent.TimeUnit;

public class App
{
    private static LinkedList<String> fileList;

    public static void main(String[] args) {
        run256SquarePixelsImages(8);
    }

    private static void run512SquarePixelsImages(int numThreads) {
        // Call a recursive method to list all files
        fileList = new LinkedList<>();
        String originalPath = "/Users/babebbu/Thesis/afhq/train";
        listFiles(createDirectory(originalPath));
    String url = "http://192.168.1.200:8000/evaluate/tensorrt";
    String deviceId = "EC-02";

    int testId = createTest(deviceId, numThreads, fileList.size(), 512);
    runUsingThreads(originalPath, testId, url, numThreads);
}

private static void run256SquarePixelsImages(int numThreads) {
    fileList = new LinkedList<>();
    String resizedImageDirectory = "/Users/babebbu/Thesis/afhq/resized";
    listFiles(createDirectory(resizedImageDirectory));

    String url = "http://192.168.1.200:8000/evaluate/tensorrt";
    String deviceId = "EC-02";

    int testId = createTest(deviceId, numThreads, fileList.size(), 256);
    runUsingThreads(resizedImageDirectory, testId, url, numThreads);
}

private static File createDirectory(String path) {
    File file = new File(path);

    // Check if the specified path is a directory
    if (!file.isDirectory()) {
        throw new RuntimeException("Specified path is not a directory.");
    }

    return file;
}
```

```
    private static void resizeImage(String directoryPath, String file) {
        String inputImagePath = directoryPath + "/" + file;
        String outputImagePath = "/Users/babebbu/Thesis/afhq/resized/" + file;

        // Set the desired dimensions for the resized image
        int newWidth = 256; // New width in pixels
        int newHeight = 256; // New height in pixels

        try {
            // Read the input image from the file
            File inputFile = new File(inputImagePath);
            BufferedImage inputImage = ImageIO.read(inputFile);

            // Create a new scaled image with the desired dimensions
            BufferedImage outputImage = new BufferedImage(newWidth, newHeight,
inputImage.getType());

            // Perform the resizing
            Graphics2D g2d = outputImage.createGraphics();
            g2d.drawImage(inputImage, 0, 0, newWidth, newHeight, null);
            g2d.dispose();

            // Write the resized image to the output file
            File outputFile = new File(outputImagePath);
            ImageIO.write(outputImage, "jpg", outputFile);

            System.out.println("Image " + outputImagePath + " resized successfully!");
        } catch (Exception ex) {
            System.err.println("Error resizing image: " + ex.getMessage());
        }
    }
```

```
  private static int createTest(String deviceId, int rate, int totalFiles, int resolution)
{
      try {
          ObjectMapper mapper = new ObjectMapper();
          String json = mapper.writeValueAsString(TestRequest.builder()
              .name(deviceId + "-BATCH-" + rate)
              .description(String.format("Nvidia RTX 3070 handling %s requests per
second. JPEG Image %sx%s.", rate, resolution, resolution))
              .device(deviceId)
              .rate(rate)
              .startedTime(new Date().getTime())
              .totalFiles(totalFiles)
              .build()
          );
          Response response = Request.post("http://localhost:8080/tests")
              .bodyString(json, ContentType.APPLICATION_JSON)
              .execute();
          String res = response.returnContent().asString();
          System.out.println(res);
          return mapper.readValue(res, Test.class).getId();
      } catch (HttpResponseException e) {
          System.out.println(e.getStatusCode());
          System.out.println(e.getMessage());
          throw new RuntimeException("...");
      } catch (IOException e) {
          throw new RuntimeException(e);
      }
  }

  private static Test updateTest(int testId, int totalSuccess) {
      try {
          ObjectMapper mapper = new ObjectMapper();
          String json = mapper.writeValueAsString(TestRequest.builder()
              .finishedTime(new Date().getTime())
              .totalSuccess(totalSuccess)
              .totalFailed(fileList.size() - totalSuccess)
              .build()
          );
          Response response = Request.post("http://localhost:8080/tests/" + testId)
              .bodyString(json, ContentType.APPLICATION_JSON)
              .execute();
          return mapper.readValue(response.returnContent().asString(), Test.class);
      } catch (IOException e) {
          throw new RuntimeException(e);
      }
  }

  private static void runUsingThreads(String directoryPath, int testId, String url, int
numThreads) {
      System.out.println("Total Files: " + fileList.size());
      System.out.println("Threads starting...");
      while (!fileList.isEmpty()) {
          for (int i = 0; i < numThreads && !fileList.isEmpty(); i++) {
              String imageFilePath = directoryPath + "/" + fileList.pop();
              new Thread(new PerformanceEvaluationRunnable(testId, url,
imageFilePath)).start();
          }
          try {
              Thread.sleep(TimeUnit.MILLISECONDS.toMillis(1000));
          } catch (InterruptedException e) {
              throw new RuntimeException(e);
          }
      }
  }

  private static void runUsingParallelStream(String directoryPath, int testId, String
url) {
      fileList.parallelStream().forEach(file -> {
          String imageFilePath = directoryPath + "/" + file;
          Runnable runnable = new PerformanceEvaluationRunnable(testId, url,
imageFilePath);
          runnable.run();
      });
  }

  private static void listFiles(File directory) {
      listFiles(directory, null);
  }

  private static void listFiles(File directory, String label) {
      File[] files = directory.listFiles();

      if (files != null) {
          for (File file : files) {
              if (file.isDirectory()) {
                  // Recursive call for subdirectories
                  listFiles(file, file.getName());
              }
              else if (file.isFile()) {
                  String fileName = label + "/" + file.getName();
                  fileList.add(fileName);
              }
          }
      }
  }

}
```

PerformanceEvaluationRunnable.java

```
package dev.babebbu.academic.thesis.master;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.PropertyNamingStrategies;
import org.apache.hc.client5.http.fluent.Request;
import org.apache.hc.client5.http.fluent.Response;
import org.apache.hc.core5.http.ContentType;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.mime.MultipartEntityBuilder;
import org.apache.http.entity.mime.content.FileBody;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.util.Map;
```

```
public class PerformanceEvaluationRunnable implements Runnable {

    public static int successRequest = 0;

    private final Logger log = LoggerFactory.getLogger(getClass());
    private final ObjectMapper mapper;
    private final ObjectMapper snakeCaseApiMapper;

    private final String url;

    private final int testId;

    private final String imageFilePath;

    private Record record;

    public PerformanceEvaluationRunnable(int testId, String url, String
imageFilePath) {
        this.testId = testId;
        this.url = url;
        this.imageFilePath = imageFilePath;
        this.mapper = new ObjectMapper();
        this.snakeCaseApiMapper = new ObjectMapper();
        this.snakeCaseApiMapper.setPropertyNamingStrategy(new
PropertyNamingStrategies.SnakeCaseStrategy());
    }

    public void run() {
        try (CloseableHttpClient closeableHttpClient = HttpClients.createDefault()) {
            File file = new File(imageFilePath);
            long fileSizeInBytes = file.length();
```

```
String[] path = imageFilePath.split("/");
String label = path[path.length - 2];
String filename = path[path.length - 1];
String imageFileName = label + "/" + filename;

// Invoke Image Classification API
HttpPost httpPost = new HttpPost(url);
HttpEntity requestEntity = MultipartEntityBuilder.create()
    .addPart("file", new FileBody(new File(imageFilePath)))
    .build();
httpPost.setEntity(requestEntity);

long startTime = System.currentTimeMillis();
CloseableHttpResponse response = closeableHttpClient.execute(httpPost);
long elapsedTime = System.currentTimeMillis() - startTime;
InputStreamReader responseContent = new
InputStreamReader(response.getEntity().getContent(), StandardCharsets.UTF_8);

if (response.getStatusLine().getStatusCode() != 200) {
    if (imageFileName.contains("Thumbs.db")) {
        System.out.println("Skipping Thumbs.db file");
        return;
    }
    if (imageFileName.contains(".DS_Store")) {
        System.out.println("Skipping .DS_Store file");
        return;
    }
    return;
}
```

```
            PetImagesAPIResponse res =
snakeCaseApiMapper.readValue(response.getEntity().getContent().readAllBytes(),
PetImagesAPIResponse.class);

            String predictedLabel = res.getResult().get("Dog") >
res.getResult().get("Cat") ? "Dog" : "Cat";
            double confidence = res.getResult().get(predictedLabel);

            if (confidence < 0.5) {
               throw new RuntimeException("Prediction Confidence is below 50%");
            }

            record = Record.builder()
               .test(testId)
               .inferenceTime(res.getInferenceTime())
               .executionTime(elapsedTime)
               .imageFileName(imageFileName)
               .actual(label)
               .prediction(predictedLabel)
               .accurate(label.equalsIgnoreCase(predictedLabel))
               .confidence(confidence)
               .dataTransfer(fileSizeInBytes)
               .build();

            storeRecord();

            closeableHttpClient.close();

            successRequest++;
         } catch (IOException e) {
            throw new RuntimeException(e);
```

```
        }
    }

    private void storeRecord() {
        try {
            String json = mapper.writeValueAsString(record);
            Response response = Request.post("http://localhost:8080/records")
                .bodyString(json, ContentType.APPLICATION_JSON)
                .execute();
            System.out.println(mapper.readValue(response.returnContent().asString(),
StatsAPIResponse.class));
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

PetImagesAPIResponse.java

```
package dev.babebbu.academic.thesis.master;

import lombok.Data;

@Data
public class PetImagesAPIResponse {
    private double inferenceTime;
    private Result result;
}
```

Record.java

```
package dev.babebbu.academic.thesis.master;
```

```
import lombok.Builder;
import lombok.Data;

@Data
@Builder
public class Record {
    private int test;
    private double inferenceTime;
    private double executionTime;
    private String imageFileName;
    private String actual;
    private String prediction;
    private boolean accurate;
    private double confidence;
    private double dataTransfer;
}
```

Result.java

```
package dev.babebbu.academic.thesis.master;

import java.util.HashMap;

public class Result extends HashMap<String, Double> {

    public Double getDog() {
        return get("dog");
    }

    public Double getCat() {
        return get("cat");
    }
    public Double get(String key) {
        return super.get(key.toLowerCase());
    }
}
```

StatsAPIResponse.java

```
package dev.babebbu.academic.thesis.master;

import java.util.HashMap;

public class StatsAPIResponse extends HashMap<String, Object> {
}
```

Test.java

```
package dev.babebbu.academic.thesis.master;

import lombok.Data;

import java.util.Date;
import java.util.Map;

@Data
public class Test {
    private int id;
    private String name;
    private String description;
    private Map<String, Object> device;
    private int rate;
    private Date startedTime;
    private Date finishedTime;
    private int totalFiles;
    private int totalSuccess;
    private int totalFailed;
}
```

TestRequest.java

```
package dev.babebbu.academic.thesis.master;

import lombok.Builder;
import lombok.Data;

@Data
@Builder
public class TestRequest {
    private String name;
    private String description;
    private String device;
    private int rate;
    private long startedTime;
    private long finishedTime;
    private int totalFiles;
    private int totalSuccess;
    private int totalFailed;
}
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
```

```
<groupId>dev.babebbu.academic.thesis.master</groupId>
<artifactId>client</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>

<name>client</name>
<url>http://maven.apache.org</url>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <java.version>21</java.version>
  <maven.compiler.release>21</maven.compiler.release>
  <maven.compiler.source>21</maven.compiler.source>
  <maven.compiler.target>21</maven.compiler.target>
</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.1</version>
  </dependency>
  <dependency>
    <groupId>com.mashape.unirest</groupId>
    <artifactId>unirest-java</artifactId>
    <version>1.4.9</version>
  </dependency>
  <dependency>
    <groupId>org.apache.httpcomponents.client5</groupId>
    <artifactId>httpclient5</artifactId>
    <version>5.2.1</version>
  </dependency>
  <dependency>
    <groupId>org.apache.httpcomponents.client5</groupId>
    <artifactId>httpclient5-fluent</artifactId>
    <version>5.2.1</version>
  </dependency>
  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.15.2</version>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.30</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>2.0.7</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
```

```
      <artifactId>slf4j-simple</artifactId>
      <version>2.0.7</version>
    </dependency>
  </dependencies>
</project>
```

Appendix D
Source Code – Performance Stats API

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.4</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>dev.babebbu.academic.thesis.master</groupId>
    <artifactId>stats</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>ImageClassificationPerformanceStats</name>
    <description>ImageClassificationPerformanceStats</description>
    <properties>
        <java.version>21</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jdbc</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.mariadb.jdbc</groupId>
        <artifactId>mariadb-java-client</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-configuration-processor</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
```

```
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

ImageClassificationPerformanceStatsApplication.java

```
package dev.babebbu.academic.thesis.master.stats;


import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ImageClassificationPerformanceStatsApplication {

   public static void main(String[] args) {
      SpringApplication.run(ImageClassificationPerformanceStatsApplication.class,
args);
   }

}
```

controllers/ApplicationTypesController.java

```
package dev.babebbu.academic.thesis.master.stats.controllers;

import com.fasterxml.jackson.databind.ObjectMapper;
import dev.babebbu.academic.thesis.master.stats.models.entities.ApplicationType;
import
dev.babebbu.academic.thesis.master.stats.models.requests.ApplicationTypeRequest;
import
dev.babebbu.academic.thesis.master.stats.repositories.ApplicationTypesRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Map;
import java.util.Optional;

@RestController
```

```
@RequestMapping("application-types")
@RequiredArgsConstructor
public class ApplicationTypesController {

    private final ApplicationTypesRepository repository;
    private final ObjectMapper objectMapper;

    @GetMapping
    public Object list() {
        return repository.findAll(Pageable.unpaged());
    }

    @GetMapping("/{slug}")
    public Object get(@PathVariable("slug") final String slug) {
        return repository.findById(slug);
    }

    @PostMapping
    public Object create(@RequestBody ApplicationTypeRequest request) {
        ApplicationType entity = getEntityFromRequest(request);
        entity.setId(request.getName().toLowerCase().replace(" ", "-"));
        return repository.save(entity);
    }

    @PutMapping("/{slug}")
    public Object update(@PathVariable("slug") String slug, @RequestBody
ApplicationTypeRequest request) {
        Optional<ApplicationType> record = repository.findById(slug);

        if (record.isEmpty()) {
            return notFoundError();
        }

        ApplicationType entity = getEntityFromRequest(request);
        entity.setId(slug);

        return repository.save(entity);
    }

    @DeleteMapping("{slug}")
    public Object delete(@PathVariable("slug") String slug) {
        if (!repository.existsById(slug)) {
            return notFoundError();
        }
        repository.deleteById(slug);
        return ResponseEntity.ok();
    }

    private ApplicationType getEntityFromRequest(ApplicationTypeRequest
request) {
        return objectMapper.convertValue(request, ApplicationType.class);
    }

    private Object notFoundError() {
        return ResponseEntity
            .status(HttpStatus.NOT_FOUND)
            .body(Map.of("message", "Application Type not found"));
    }

}
```

controllers/DeviceController.java

```
package dev.babebbu.academic.thesis.master.stats.controllers;

import dev.babebbu.academic.thesis.master.stats.models.entities.*;
import dev.babebbu.academic.thesis.master.stats.models.requests.DeviceRequest;
import dev.babebbu.academic.thesis.master.stats.repositories.*;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Map;
import java.util.Optional;

@RestController
@RequestMapping("devices")
@RequiredArgsConstructor
public class DevicesController {

  private final DevicesRepository repository;
  private final MLEnvironmentsRepository mlEnvironmentsRepository;

  @GetMapping
  public Object list() {
    return repository.findAll(Pageable.unpaged());
  }

  @GetMapping("/{slug}")
  public Object get(@PathVariable("slug") final String slug) {
    return repository.findById(slug);
  }
  @PostMapping
  public Object create(@RequestBody DeviceRequest request) {
    Optional<MLEnvironment> environment =
mlEnvironmentsRepository.findById(request.getEnvironment());

    if (environment.isEmpty()) {
      return ResponseEntity.badRequest().body(Map.of(
        "message", "Some of arguments are not exist in the database.",
        "exists", Map.of(
          "environment", false
        )
      ));
    }

    Device entity = Device.builder()
      .id(request.getName().toLowerCase().replace(" ", "-"))
      .name(request.getName())
      .displayName(request.getDisplayName())
      .description(request.getDescription())
      .hardware(request.getHardware())
      .location(request.getLocation())
      .latency(request.getLatency())
      .environment(environment.get())
      .build();

    return repository.save(entity);
  }

  @PutMapping("/{slug}")
  public Object update(@PathVariable("slug") String slug, @RequestBody
DeviceRequest request) {
        Optional<Device> record = repository.findById(slug);

        if (record.isEmpty()) {
            return notFoundError();
        }

        Optional<MLEnvironment> environment =
mlEnvironmentsRepository.findById(request.getEnvironment());

        if (environment.isEmpty()) {
            return ResponseEntity.badRequest().body(Map.of(
                "message", "Some of arguments are not exist in the database.",
                "exists", Map.of(
                    "environment", false
                )
            ));
        }

        Device entity = Device.builder()
            .id(slug)
            .name(request.getName())
            .displayName(request.getDisplayName())
            .description(request.getDescription())
            .hardware(request.getHardware())
            .location(request.getLocation())
            .latency(request.getLatency())
            .environment(environment.get())
            .build();

        return repository.save(entity);
```

```
    }

    @DeleteMapping("{slug}")
    public Object delete(@PathVariable("slug") String slug) {
        if (!repository.existsById(slug)) {
            return notFoundError();
        }
        repository.deleteById(slug);
        return ResponseEntity.ok();
    }

    private Object notFoundError() {
        return ResponseEntity
            .status(HttpStatus.NOT_FOUND)
            .body(Map.of("message", "Device not found"));
    }

}
```

controllers/MLEnvironmentsController.java

```
package dev.babebbu.academic.thesis.master.stats.controllers;

import dev.babebbu.academic.thesis.master.stats.models.entities.ApplicationType;
import dev.babebbu.academic.thesis.master.stats.models.entities.MLEnvironment;
import dev.babebbu.academic.thesis.master.stats.models.entities.NetworkTier;
import dev.babebbu.academic.thesis.master.stats.models.entities.Runtime;
import
dev.babebbu.academic.thesis.master.stats.models.requests.MLEnvironmentRequest;
import
dev.babebbu.academic.thesis.master.stats.repositories.ApplicationTypesRepository;
import
```

```
dev.babebbu.academic.thesis.master.stats.repositories.MLEnvironmentsRepository;
import dev.babebbu.academic.thesis.master.stats.repositories.RuntimesRepository;
import dev.babebbu.academic.thesis.master.stats.repositories.TiersRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Map;
import java.util.Optional;

@RestController
@RequestMapping("ml-environments")
@RequiredArgsConstructor
public class MLEnvironmentsController {

    private final MLEnvironmentsRepository repository;
    private final ApplicationTypesRepository applicationTypesRepository;
    private final TiersRepository networkTiersRepository;
    private final RuntimesRepository runtimesRepository;
    @GetMapping
    public Object list() {
        return repository.findAll(Pageable.unpaged());
    }

    @GetMapping("/{slug}")
    public Object get(@PathVariable("slug") final String slug) {
        return repository.findById(slug);
    }
```

```
  @PostMapping
  public Object create(@RequestBody MLEnvironmentRequest request) {
      Optional<NetworkTier> networkTier =
networkTiersRepository.findById(request.getTier());
      Optional<ApplicationType> applicationType =
applicationTypesRepository.findById(request.getApplicationType());
      Optional<Runtime> runtime =
runtimesRepository.findById(request.getRuntime());

      if (networkTier.isEmpty() || applicationType.isEmpty() || runtime.isEmpty()) {
        return ResponseEntity.badRequest().body(Map.of(
          "message", "Some of arguments are not exist in the database.",
          "exists", Map.of(
            "tier", networkTier.isPresent(),
            "applicationType", applicationType.isPresent(),
            "runtime", runtime.isPresent()
          )
        ));
      }

      MLEnvironment entity = MLEnvironment.builder()
        .id(request.getName().toLowerCase().replace(" ", "-"))
        .name(request.getName())
        .tier(networkTier.get())
        .applicationType(applicationType.get())
        .runtime(runtime.get())
        .build();

      return repository.save(entity);
  }
  @PutMapping("/{slug}")
  public Object update(@PathVariable("slug") String slug, @RequestBody
MLEnvironmentRequest request) {
      Optional<MLEnvironment> record = repository.findById(slug);

      if (record.isEmpty()) {
        return notFoundError();
      }

      Optional<NetworkTier> networkTier =
networkTiersRepository.findById(request.getTier());
      Optional<ApplicationType> applicationType =
applicationTypesRepository.findById(request.getApplicationType());
      Optional<Runtime> runtime =
runtimesRepository.findById(request.getRuntime());

      if (networkTier.isEmpty() || applicationType.isEmpty() || runtime.isEmpty()) {
        return ResponseEntity.badRequest().body(Map.of(
          "message", "Some of arguments are not exist in the database.",
          "exists", Map.of(
            "tier", networkTier.isPresent(),
            "applicationType", applicationType.isPresent(),
            "runtime", runtime.isPresent()
          )
        ));
      }

      MLEnvironment entity = MLEnvironment.builder()
        .id(slug)
        .name(request.getName())
        .tier(networkTier.get())
```

```
            .applicationType(applicationType.get())
            .runtime(runtime.get())
            .build();

        return repository.save(entity);
    }

    @DeleteMapping("{slug}")
    public Object delete(@PathVariable("slug") String slug) {
        if (!repository.existsById(slug)) {
            return notFoundError();
        }
        repository.deleteById(slug);
        return ResponseEntity.ok();
    }

    private Object notFoundError() {
        return ResponseEntity
            .status(HttpStatus.NOT_FOUND)
            .body(Map.of("message", "ML Environment not found"));
    }

}
```

controllers/NetworkTiersController.java

```
package dev.babebbu.academic.thesis.master.stats.controllers;

import com.fasterxml.jackson.databind.ObjectMapper;
import dev.babebbu.academic.thesis.master.stats.models.entities.NetworkTier;
import
dev.babebbu.academic.thesis.master.stats.models.requests.NetworkTierRequest;
```

```
import dev.babebbu.academic.thesis.master.stats.repositories.TiersRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Map;
import java.util.Optional;

@RestController
@RequestMapping("network-tiers")
@RequiredArgsConstructor
public class NetworkTiersController {

    private final TiersRepository tiersRepository;
    private final ObjectMapper objectMapper;

    @GetMapping
    public Object listNetworkTiers() {
        return tiersRepository.findAll(Pageable.unpaged());
    }

    @GetMapping("/{slug}")
    public Object getNetworkTier(@PathVariable("slug") final String slug) {
        return tiersRepository.findById(slug);
    }

    @PostMapping
    public Object createNetworkTier(@RequestBody NetworkTierRequest request) {
        NetworkTier networkTier = getEntityFromRequest(request);
        networkTier.setId(request.getName().toLowerCase().replace(" ", "-"));
        return tiersRepository.save(networkTier);
    }

    @PutMapping("/{slug}")
    public Object updateNetworkTier(@PathVariable("slug") String slug,
@RequestBody NetworkTierRequest request) {
        Optional<NetworkTier> record = tiersRepository.findById(slug);

        if (record.isEmpty()) {
            return networkTierNotFoundError();
        }

        NetworkTier networkTier = getEntityFromRequest(request);
        networkTier.setId(slug);

        return tiersRepository.save(networkTier);
    }

    private NetworkTier getEntityFromRequest(NetworkTierRequest request) {
        return objectMapper.convertValue(request, NetworkTier.class);
    }

    @DeleteMapping("{slug}")
    public Object deleteNetworkTier(@PathVariable("slug") String slug) {
        if (!tiersRepository.existsById(slug)) {
            return networkTierNotFoundError();
        }
        tiersRepository.deleteById(slug);
        return ResponseEntity.ok();
    }
    private Object networkTierNotFoundError() {
        return ResponseEntity
            .status(HttpStatus.NOT_FOUND)
            .body(Map.of("message", "Network Tier not found"));
    }

}
```

controllers/RecordsController.java

```
package dev.babebbu.academic.thesis.master.stats.controllers;

import dev.babebbu.academic.thesis.master.stats.models.entities.Record;
import dev.babebbu.academic.thesis.master.stats.models.entities.Test;
import dev.babebbu.academic.thesis.master.stats.models.requests.RecordRequest;
import dev.babebbu.academic.thesis.master.stats.repositories.RecordsRepository;
import dev.babebbu.academic.thesis.master.stats.repositories.TestRepository;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Map;
import java.util.Optional;

@RestController
@RequestMapping("records")
```

```
@RequiredArgsConstructor
public class RecordsController {

    private final RecordsRepository repository;
    private final TestRepository testRepository;
    private final Logger log = LoggerFactory.getLogger(getClass());

    @GetMapping
    public Object list() {
        return repository.findAllByTestIsNotNull(Pageable.unpaged());
    }

    @GetMapping("/{id}")
    public Object get(@PathVariable("id") final int id) {
        return repository.findById(id);
    }

    @PostMapping
    public Object create(@RequestBody RecordRequest request) {
        log.info("{}", request);

        Optional<Test> test = testRepository.findById(request.getTest());

        if (test.isEmpty()) {
            return ResponseEntity.badRequest().body(Map.of(
                "message", "Some of arguments are not exist in the database.",
                "exists", Map.of(
                    "device", false
                )
            ));
        }
        Record entity = Record.builder()
            .test(test.get())
            .inferenceTime(request.getInferenceTime())
            .executionTime(request.getExecutionTime())
            .imageFileName(request.getImageFileName())
            .prediction(request.getPrediction())
            .actual(request.getActual())
            .accurate(request.isAccurate())
            .confidence(request.getConfidence())
            .dataTransfer(request.getDataTransfer())
            .build();

        return repository.save(entity);
    }

    @PutMapping("/{id}")
    public Object update(@PathVariable("id") int id, @RequestBody RecordRequest
request) {
        Optional<Record> record = repository.findById(id);

        if (record.isEmpty()) {
            return notFoundError();
        }

        Optional<Test> test = testRepository.findById(request.getTest());

        if (test.isEmpty()) {
            return ResponseEntity.badRequest().body(Map.of(
                "message", "Some of arguments are not exist in the database.",
                "exists", Map.of(
                "device", false
            )
        ));
    }

    Record entity = Record.builder()
        .id(id)
        .test(test.get())
        .inferenceTime(request.getInferenceTime())
        .executionTime(request.getExecutionTime())
        .imageFileName(request.getImageFileName())
        .prediction(request.getPrediction())
        .actual(request.getActual())
        .accurate(request.isAccurate())
        .confidence(request.getConfidence())
        .dataTransfer(request.getDataTransfer())
        .build();

    return repository.save(entity);
}

@DeleteMapping("{id}")
public Object delete(@PathVariable("id") int id) {
    if (!repository.existsById(id)) {
        return notFoundError();
    }
    repository.deleteById(id);
    return ResponseEntity.ok();
}

private Object notFoundError() {
        return ResponseEntity
            .status(HttpStatus.NOT_FOUND)
            .body(Map.of("message", "Record not found"));
    }

}
```

controllers/RuntimesController.java

```
package dev.babebbu.academic.thesis.master.stats.controllers;

import com.fasterxml.jackson.databind.ObjectMapper;
import dev.babebbu.academic.thesis.master.stats.models.entities.Runtime;
import dev.babebbu.academic.thesis.master.stats.models.requests.RuntimeRequest;
import dev.babebbu.academic.thesis.master.stats.repositories.RuntimesRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Map;
import java.util.Optional;

@RestController
@RequestMapping("runtimes")
@RequiredArgsConstructor
public class RuntimesController {

    private final RuntimesRepository repository;
    private final ObjectMapper objectMapper;
```

```
  @GetMapping
  public Object list() {
    return repository.findAll(Pageable.unpaged());
  }

  @GetMapping("/{slug}")
  public Object get(@PathVariable("slug") final String slug) {
    return repository.findById(slug);
  }

  @PostMapping
  public Object create(@RequestBody RuntimeRequest request) {
    Runtime entity = getEntityFromRequest(request);
    entity.setId(request.getName().toLowerCase().replace(" ", "-"));
    return repository.save(entity);
  }

  @PutMapping("/{slug}")
  public Object update(@PathVariable("slug") String slug, @RequestBody
RuntimeRequest request) {
    Optional<Runtime> record = repository.findById(slug);

    if (record.isEmpty()) {
      return notFoundError();
    }

    Runtime entity = getEntityFromRequest(request);
    entity.setId(slug);

    return repository.save(entity);
    }

    @DeleteMapping("{slug}")
    public Object delete(@PathVariable("slug") String slug) {
        if (!repository.existsById(slug)) {
            return notFoundError();
        }
        repository.deleteById(slug);
        return ResponseEntity.ok();
    }

    private Runtime getEntityFromRequest(Object request) {
        return objectMapper.convertValue(request, Runtime.class);
    }

    private Object notFoundError() {
        return ResponseEntity
            .status(HttpStatus.NOT_FOUND)
            .body(Map.of("message", "Runtime not found"));
    }

}
```

controllers/TestsController.java

```
package dev.babebbu.academic.thesis.master.stats.controllers;

import dev.babebbu.academic.thesis.master.stats.models.TestStats;
import dev.babebbu.academic.thesis.master.stats.models.entities.Device;
import dev.babebbu.academic.thesis.master.stats.models.entities.Test;
import dev.babebbu.academic.thesis.master.stats.models.requests.TestRequest;
import dev.babebbu.academic.thesis.master.stats.repositories.DevicesRepository;
import dev.babebbu.academic.thesis.master.stats.repositories.TestRepository;
import dev.babebbu.academic.thesis.master.stats.services.StatsService;
import dev.babebbu.academic.thesis.master.stats.services.ToolsService;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageImpl;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Map;
import java.util.Optional;

@RestController
@RequestMapping("tests")
@RequiredArgsConstructor
public class TestsController {

    private final TestRepository repository;
    private final DevicesRepository devicesRepository;

    private final Logger log = LoggerFactory.getLogger(getClass());

    private final StatsService statsService;

    private final ToolsService toolsService;

    @GetMapping
```

```
  public Object list() {
    return repository.findAll(Pageable.unpaged());
  }

  @GetMapping("stats")
  public Object listStatsOfAllTests() {
    Page<TestStats> testStats =
repository.findAll(Pageable.unpaged()).map(statsService::getStatsByTest);
    return new PageImpl<>(toolsService.reverseCollection(testStats.stream()));
  }

  @GetMapping("{id}")
  public Object get(@PathVariable("id") final int id) {
    return repository.findById(id);
  }

  @PostMapping
  public Object create(@RequestBody TestRequest request) {
    Optional<Device> device =
devicesRepository.findById(request.getDevice().toLowerCase());

    if (device.isEmpty()) {
      return ResponseEntity.badRequest().body(Map.of(
        "message", "Some of arguments are not exist in the database.",
        "exists", Map.of(
          "environment", false
        )
      ));
    }

    Test entity = Test.builder()
```

```
                .name(request.getName())
                .description(request.getDescription())
                .rate(request.getRate())
                .device(device.get())
                .totalSuccess(request.getTotalSuccess())
                .totalFailed(request.getTotalFailed())
                .totalFiles(request.getTotalFiles())
                .build();

        entity.setStartedTime(request.getStartedTime());
//        entity.setFinishedTime(request.getFinishedTime());

        log.info("{}", request);

        return repository.save(entity);
    }

    @PutMapping("{id}")
    public Object update(@PathVariable("id") int id, @RequestBody TestRequest
request) {
        Optional<Test> record = repository.findById(id);

        if (record.isEmpty()) {
            return notFoundError();
        }

        Optional<Device> device =
devicesRepository.findById(request.getDevice().toLowerCase());

        if (device.isEmpty()) {
            return ResponseEntity.badRequest().body(Map.of(
                "message", "Some of arguments are not exist in the database.",
                "exists", Map.of(
                    "environment", false
                )
            ));
        }

        Test entity = Test.builder()
            .id(id)
            .name(request.getName())
            .description(request.getDescription())
            .rate(request.getRate())
            .device(device.get())
            .totalSuccess(request.getTotalSuccess())
            .totalFailed(request.getTotalFailed())
            .totalFiles(request.getTotalFiles())
            .build();

        entity.setStartedTime(request.getStartedTime());
        entity.setFinishedTime(request.getFinishedTime());

        return repository.save(entity);
    }

    @PatchMapping("{id}")
    public Object finalize(@PathVariable("id") int id, @RequestBody TestRequest
request) {
        Optional<Test> record = repository.findById(id);

        if (record.isEmpty()) {
            return notFoundError();
        }

        Test entity = Test.builder()
            .id(id)
            .totalSuccess(request.getTotalSuccess())
            .totalFailed(request.getTotalFailed())
            .build();

        entity.setFinishedTime(request.getFinishedTime());

        return repository.save(entity);
    }

    @DeleteMapping("{id}")
    public Object delete(@PathVariable("id") int id) {
        if (!repository.existsById(id)) {
            return notFoundError();
        }
        repository.deleteById(id);
        return ResponseEntity.ok();
    }

    private Object notFoundError() {
        return ResponseEntity
            .status(HttpStatus.NOT_FOUND)
            .body(Map.of("message", "Device not found"));
    }

}
```

entities/ApplicationType.java

```
package dev.babebbu.academic.thesis.master.stats.models.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "application_types")
public class ApplicationType {
   @Id
   private String id;
   private String name;
   private String type;
   private String programmingLanguage;
}
```

entities/Device.java

```
package dev.babebbu.academic.thesis.master.stats.models.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "devices")
public class Device {

    @Id
    private String id;
    private String name;
    private String displayName;
    private String description;
    private String hardware;
    private String location;
    private double latency;

    @ManyToOne
    @JoinColumn
    private MLEnvironment environment;

}
```

entities/MLEnvironment.java

```
package dev.babebbu.academic.thesis.master.stats.models.entities;


import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "ml_environments")
public class MLEnvironment {

    @Id
    private String id;

    private String name;

    @ManyToOne
    private NetworkTier tier;

    @ManyToOne
    @JoinColumn
    private Runtime runtime;

    @ManyToOne
    @JoinColumn
    private ApplicationType applicationType;

}
```

entities/NetworkTier.java

```
package dev.babebbu.academic.thesis.master.stats.models.entities;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "network_tiers")
public class NetworkTier {
    @Id
    private String id;
    private String name;
    private String tier;
    private String type;
    private String description;
}
```

entities/Record.java

```
package dev.babebbu.academic.thesis.master.stats.models.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.annotations.CreationTimestamp;
import java.util.Date;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "records")
public class Record {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @ManyToOne
    @JoinColumn
    private Test test;

    private double inferenceTime;

    private double executionTime;

    private String imageFileName;

    private String prediction;

    private String actual;

    private boolean accurate;
    private double confidence;


    private double dataTransfer;


    @CreationTimestamp
    private Date createdAt;


}
```

entities/Runtime.java

```
package dev.babebbu.academic.thesis.master.stats.models.entities;


import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;


@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "runtimes")
public class Runtime {
    @Id
    private String id;
    private String name;
    private String os;
    private String mlEngine;
}
```

entities/Test.java

```
package dev.babebbu.academic.thesis.master.stats.models.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;
import java.util.concurrent.TimeUnit;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "tests")
public class Test {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String name;

    private String description;

    @ManyToOne
    private Device device;

    private Integer rate;

    private Date startedTime;

    private Date finishedTime;

    private Integer totalFiles;

    private Integer totalSuccess;

    private Integer totalFailed;

    public void setStartedTime(long timestamp) {
        startedTime = new Date(timestamp);
    }

    public void setFinishedTime(long timestamp) {
        finishedTime = new Date(timestamp);
    }

}
```

requests/ApplicationTypeRequest.java

```
package dev.babebbu.academic.thesis.master.stats.models.requests;

import lombok.Data;

@Data
public class ApplicationTypeRequest {
    private String name;
    private String type;
    private String programmingLanguage;
}
```

requests/DeviceRequest.java

```
package dev.babebbu.academic.thesis.master.stats.models.requests;

import lombok.Data;

@Data
public class DeviceRequest {
    private String id;
    private String name;
    private String displayName;
    private String description;
    private String hardware;
    private String location;
    private double latency;
    private String environment;
}
```

requests/MLEnvironmentRequest.java

```
package dev.babebbu.academic.thesis.master.stats.models.requests;

import lombok.Data;

@Data
public class MLEnvironmentRequest {
    private String name;
    private String tier;
    private String runtime;
    private String applicationType;
}
```

requests/NetworkTierRequest.java

```
package dev.babebbu.academic.thesis.master.stats.models.requests;

import lombok.Data;

@Data
public class NetworkTierRequest {
    private String name;
    private String type;
    private String tier;
    private String description;
}
```

requests/RecordRequest.java

```
package dev.babebbu.academic.thesis.master.stats.models.requests;

import lombok.Data;

@Data
public class RecordRequest {

    private int test;

    private double inferenceTime;
    private double executionTime;

    private String imageFileName;
    private String prediction;
    private String actual;
    private boolean accurate;
    private double confidence;
    private double dataTransfer;


}
```

requests/RuntimeRequest.java

```
package dev.babebbu.academic.thesis.master.stats.models.requests;

import lombok.Data;

@Data
public class RuntimeRequest {
    private String os;
    private String name;
    private String mlEngine;
}
```

requests/TestRequest.java

```
package dev.babebbu.academic.thesis.master.stats.models.requests;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
```

```
public class TestRequest {

    private String name;
    private String description;
    private String device;
    private int rate;
    private long startedTime;
    private long finishedTime;
    private int totalFiles;
    private int totalSuccess;
    private int totalFailed;

}
```

repositories/ApplicationTypesRepository.java

```
package dev.babebbu.academic.thesis.master.stats.repositories;

import dev.babebbu.academic.thesis.master.stats.models.entities.ApplicationType;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ApplicationTypesRepository extends
JpaRepository<ApplicationType, String> {
}
```

repositories/DevicesRepository.java

```
package dev.babebbu.academic.thesis.master.stats.repositories;

import dev.babebbu.academic.thesis.master.stats.models.entities.Device;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface DevicesRepository extends JpaRepository<Device, String> {
}
```

repositories/MLEnvironmentsRepository.java

```
package dev.babebbu.academic.thesis.master.stats.repositories;

import dev.babebbu.academic.thesis.master.stats.models.entities.MLEnvironment;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface MLEnvironmentsRepository extends
JpaRepository<MLEnvironment, String> {
}
```

repositories/RecordsRepository.java

```
package dev.babebbu.academic.thesis.master.stats.repositories;

import dev.babebbu.academic.thesis.master.stats.models.entities.Record;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface RecordsRepository extends JpaRepository<Record, Integer> {
    List<Record> findAllByTestIsNotNull(Pageable pageable);
}
```

repositories/RuntimesRepository.java

```
package dev.babebbu.academic.thesis.master.stats.repositories;

import dev.babebbu.academic.thesis.master.stats.models.entities.Runtime;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface RuntimesRepository extends JpaRepository<Runtime, String> {
}
```

repositories/TestsRepository.java

```
package dev.babebbu.academic.thesis.master.stats.repositories;

import dev.babebbu.academic.thesis.master.stats.models.entities.Test;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.query.Procedure;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.Map;

@Repository
public interface TestRepository extends JpaRepository<Test, Integer> {
}
```

repositories/TiersRepository.java

```
package dev.babebbu.academic.thesis.master.stats.repositories;

import dev.babebbu.academic.thesis.master.stats.models.entities.NetworkTier;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface TiersRepository extends JpaRepository<NetworkTier, String> {
}
```

services/StatsService.java

```
package dev.babebbu.academic.thesis.master.stats.services;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.PropertyNamingStrategy;
import dev.babebbu.academic.thesis.master.stats.models.QuantitativeStats;
import dev.babebbu.academic.thesis.master.stats.models.QueryResultSet;
import dev.babebbu.academic.thesis.master.stats.models.TestStats;
import dev.babebbu.academic.thesis.master.stats.models.entities.Test;
import lombok.RequiredArgsConstructor;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.simple.SimpleJdbcCall;
import org.springframework.stereotype.Service;

import java.util.Map;

@Service
@RequiredArgsConstructor
public class StatsService {
    private final JdbcTemplate jdbcTemplate;
    private final ObjectMapper objectMapper;

    public TestStats getStatsByTest(Test test) {
        return getStatsByTestId(test.getId());
    }

    public TestStats getStatsByTestId(int id) {

        ObjectMapper caseMapper =
objectMapper.copy().setPropertyNamingStrategy(PropertyNamingStrategy.SNAKE_CASE);

        Map<String, Object> queryResult = new SimpleJdbcCall(jdbcTemplate)
            .withProcedureName("GetStatsByTestId")
            .execute(new MapSqlParameterSource().addValue("search_id", id));

        QueryResultSet resultSet = objectMapper.convertValue(
            queryResult.get("#result-set-1"),
            QueryResultSet.class
        );

        Map<String, Object> result = resultSet.get(0);

        TestStats testStats = caseMapper.convertValue(result, TestStats.class);
        testStats.setInferenceTime(QuantitativeStats
            .builder()
            .min(result.get("min_inference_time"))
            .avg(result.get("avg_inference_time"))
            .max(result.get("max_inference_time"))
            .sd(result.get("sd_inference_time"))
            .build()
        );
        testStats.setExecutionTime(QuantitativeStats
            .builder()
            .min(result.get("min_e2e_time"))
            .avg(result.get("avg_e2e_time"))
            .max(result.get("max_e2e_time"))
            .sd(result.get("sd_e2e_time"))
            .build()
        );

        return testStats;
    }

}
```

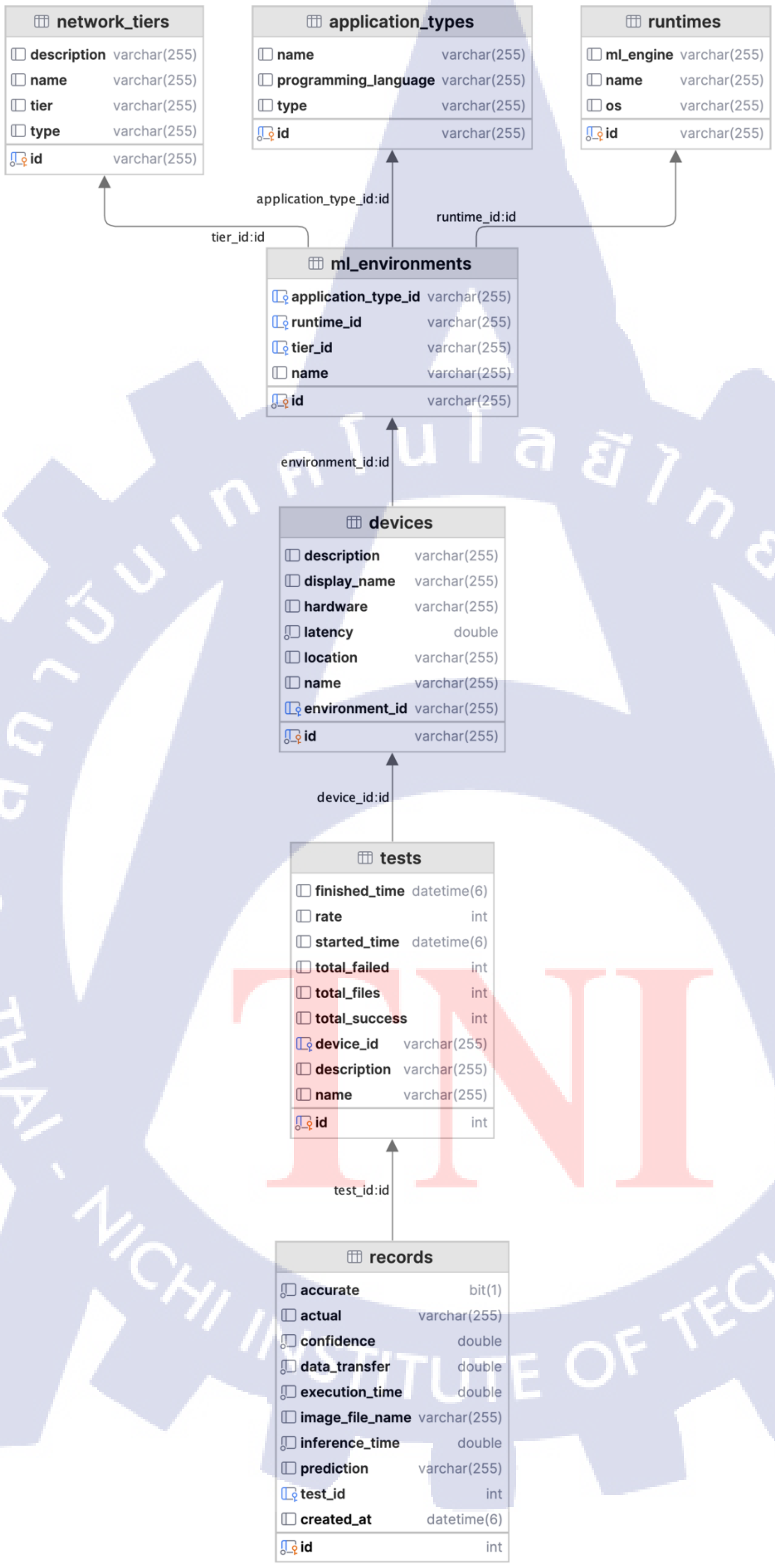
services/ToolsService.java

```
package dev.babebbu.academic.thesis.master.stats.services;

import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.Collections;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

@Service
@RequiredArgsConstructor
public class ToolsService {
  public <T> List<T> reverseCollection(Stream<T> stream) {
    return stream.collect(
      Collectors.collectingAndThen(Collectors.toList(), list -> {
        Collections.reverse(list);
        return list;
      })
    );
  }

}
```

Appendix E

Database

ER Diagram


network_tiers
description varchar(255)
name varchar(255)
tier varchar(255)
type varchar(255)
id varchar(255)
application_types
name varchar(255)
programming_language varchar(255)
type varchar(255)
id varchar(255)
runtimes
ml_engine varchar(255)
name varchar(255)
os varchar(255)
id varchar(255)
application_type_id:id
runtime_id:id
tier_id:id
ml_environments
application_type_id varchar(255)
runtime_id varchar(255)
tier_id varchar(255)
name varchar(255)
id varchar(255)
environment_id:id
devices
description varchar(255)
display_name varchar(255)
hardware varchar(255)
latency double
location varchar(255)
name varchar(255)
environment_id varchar(255)
id varchar(255)
device_id:id
tests
finished_time datetime(6)
rate int
started_time datetime(6)
total_failed int
total_files int
total_success int
device_id varchar(255)
description varchar(255)
name varchar(255)
id int
test_id:id
records
accurate bit(1)
actual varchar(255)
confidence double
data_transfer double
execution_time double
image_file_name varchar(255)
inference_time double
prediction varchar(255)
test_id int
created_at datetime(6)
id int

Procedure: GetStatsByTestId

```
CREATE
   definer = root@`%` procedure GetStatsByTestId(IN search_id int)
BEGIN
   SET SESSION sql_mode = '';
   SELECT
      tests.id AS "test_id",
      tests.description,
      tests.started_time,
      (SELECT created_at FROM records WHERE test_id = search_id ORDER BY
`id` DESC LIMIT 1) AS "finished_time",
      TIMESTAMPDIFF(SECOND, tests.started_time, (SELECT created_at FROM
records WHERE test_id = search_id ORDER BY `id` DESC LIMIT 1)) AS
"elapsed_time",
      CONCAT(devices.latency, " ms") AS "latency",
      count(records.id) AS total_records,
      CONCAT(ROUND(MIN(inference_time), 2), " ms") AS
"min_inference_time",
      CONCAT(ROUND(AVG(inference_time), 2), " ms") AS "avg_inference_time",
      CONCAT(ROUND(MAX(inference_time), 2), " ms") AS
"max_inference_time",
      CONCAT(ROUND(STDDEV(inference_time), 2), " ms")  AS
"sd_inference_time",
      CONCAT(ROUND(MIN(execution_time), 2), " ms") AS "min_e2e_time",
      CONCAT(ROUND(AVG(execution_time), 2), " ms") AS "avg_e2e_time",
      CONCAT(ROUND(MAX(execution_time), 2), " ms") AS "max_e2e_time",
      CONCAT(ROUND(STDDEV(execution_time), 2), " ms")  AS "sd_e2e_time",
      CONCAT(ROUND(AVG(execution_time) - AVG(inference_time), 2), " ms") as
"average_overhead",
      CONCAT(ROUND(AVG(accurate) * 100, 2), " %") as "accuracy",
      CONCAT(ROUND(MIN(confidence) * 100, 2), " %") as
```

```
"min_confidence_score",
        CONCAT(ROUND(AVG(confidence) * 100, 2), " %") as
"avg_confidence_score",
        CONCAT(ROUND(MAX(confidence) * 100, 2), " %") as
"max_confidence_score",
        ROUND(STDDEV(confidence) * 100, 2) as "sd_confidence_score",
        CONCAT(ROUND(MIN(data_transfer) / 1000, 2), " KB") as "min_file_size",
        CONCAT(ROUND(AVG(data_transfer) / 1000, 2), " KB") as "avg_file_size",
        CONCAT(ROUND(MAX(data_transfer) / 1000, 2), " KB") as "max_file_size",
        CONCAT(ROUND(STDDEV(data_transfer) / 1000, 2), " KB") as
"sd_file_size"
    FROM tests
    JOIN records ON tests.id = records.test_id
    JOIN devices ON tests.device_id = devices.id
    WHERE test_id = search_id;
END;
```

Appendix F

Source Code – Keras Image Classifier Training

train.py

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
from keras.callbacks import ModelCheckpoint, EarlyStopping

image_size = (180, 180)
batch_size = 64
epochs = 50

train_ds, val_ds = tf.keras.utils.image_dataset_from_directory(
    "PetImages-Cleaned",
    validation_split=0.05,
    subset="both",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)
import matplotlib.pyplot as plt

classes = ["Cat", "Dog"]

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(classes[int(labels[i])])
        plt.axis("off")
```

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
    ]
)
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
augmented_train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y))
# Apply `data_augmentation` to the training images.
train_ds = train_ds.map(
    lambda img, label: (data_augmentation(img), label),
    num_parallel_calls=tf.data.AUTOTUNE,
)
# Prefetching samples in GPU memory helps maximize GPU utilization.
train_ds = train_ds.prefetch(tf.data.AUTOTUNE)
val_ds = val_ds.prefetch(tf.data.AUTOTUNE)
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)

    # Entry block
    x = layers.Rescaling(1.0 / 255)(inputs)
    x = layers.Conv2D(128, 3, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)
```

```
previous_block_activation = x  # Set aside residual

for size in [256, 512, 728]:
    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(size, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(size, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

    # Project residual
    residual = layers.Conv2D(size, 1, strides=2, padding="same")(
        previous_block_activation
    )
    x = layers.add([x, residual])  # Add back residual
    previous_block_activation = x  # Set aside next residual

x = layers.SeparableConv2D(1024, 3, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)

x = layers.GlobalAveragePooling2D()(x)
if num_classes == 2:
    activation = "sigmoid"
    units = 1
else:
    activation = "softmax"
```

```
        units = num_classes

    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)


model = make_model(input_shape=image_size + (3,), num_classes=2)
keras.utils.plot_model(model, show_shapes=True)
callbacks = [
    ModelCheckpoint("checkpoints/save_at_{epoch}.keras"),
    EarlyStopping(monitor='val_accuracy', patience=8, restore_best_weights=True,
verbose=1)
]
model.compile(
    optimizer=keras.optimizers.legacy.Adam(1e-3),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
history = model.fit(
    train_ds,
    epochs=epochs,
    callbacks=callbacks,
    validation_data=val_ds,
)
from datetime import datetime

current_time =  datetime.now().strftime("%Y%m%d_%H%M%S")

tf.keras.saving.save_model(model, "models/keras/" + current_time + ".keras",
overwrite=True, save_format="keras")
```

```
tf.keras.saving.save_model(model, "models/tensorflow/" + current_time,
overwrite=True, save_format="tf")
tf.keras.saving.save_model(model, "models/hdf5/" + current_time + ".hdf5",
overwrite=True, save_format="h5")
# Verify
img = keras.utils.load_img(
    "PetImages/Cat/6779.jpg", target_size=image_size
)
img_array = keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

predictions = model.predict(img_array)
score = float(predictions[0])
print(f"This image is {100 * (1 - score):.2f}% cat and {100 * score:.2f}% dog.")

# summarize history for accuracy
print(history.history.keys())
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Iteration')
plt.legend(['Training', 'Validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Iteration')
plt.legend(['Training', 'Validation'], loc='upper left')
plt.show()
accuracy_min = None
accuracy_min_iteration = None
accuracy_max = None
accuracy_max_iteration = None
for i, element in enumerate(history.history['accuracy']):
   if i == 0:
      accuracy_min = element
      accuracy_max = element
   else:
      if element < accuracy_min:
         accuracy_min = element
         accuracy_min_iteration = i+1
      if element > accuracy_max:
         accuracy_max = element
         accuracy_max_iteration = i+1
   print(str(i+1) + " " + f"{round(element*100, 1)}%")

print("=========")
print(str(accuracy_max_iteration) + " " + f"{round(accuracy_max*100, 1)}%")

val_accuracy_min = None
val_accuracy_min_iteration = None
val_accuracy_max = None
val_accuracy_max_iteration = None
for i, element in enumerate(history.history['val_accuracy']):
   if i == 0:
      val_accuracy_min = element
      val_accuracy_max = element
   else:
        if element < val_accuracy_min:
            val_accuracy_min = element
            val_accuracy_min_iteration = i+1
        if element > val_accuracy_max:
            val_accuracy_max = element
            val_accuracy_max_iteration = i+1
    print(str(i+1) + " " + f"{round(element*100, 1)}%")

print("=========")
print(str(val_accuracy_max_iteration) + " " + f"{round(val_accuracy_max*100,
1)}%")
```

Appendix G

Source Code – TensorRT Model Conversion

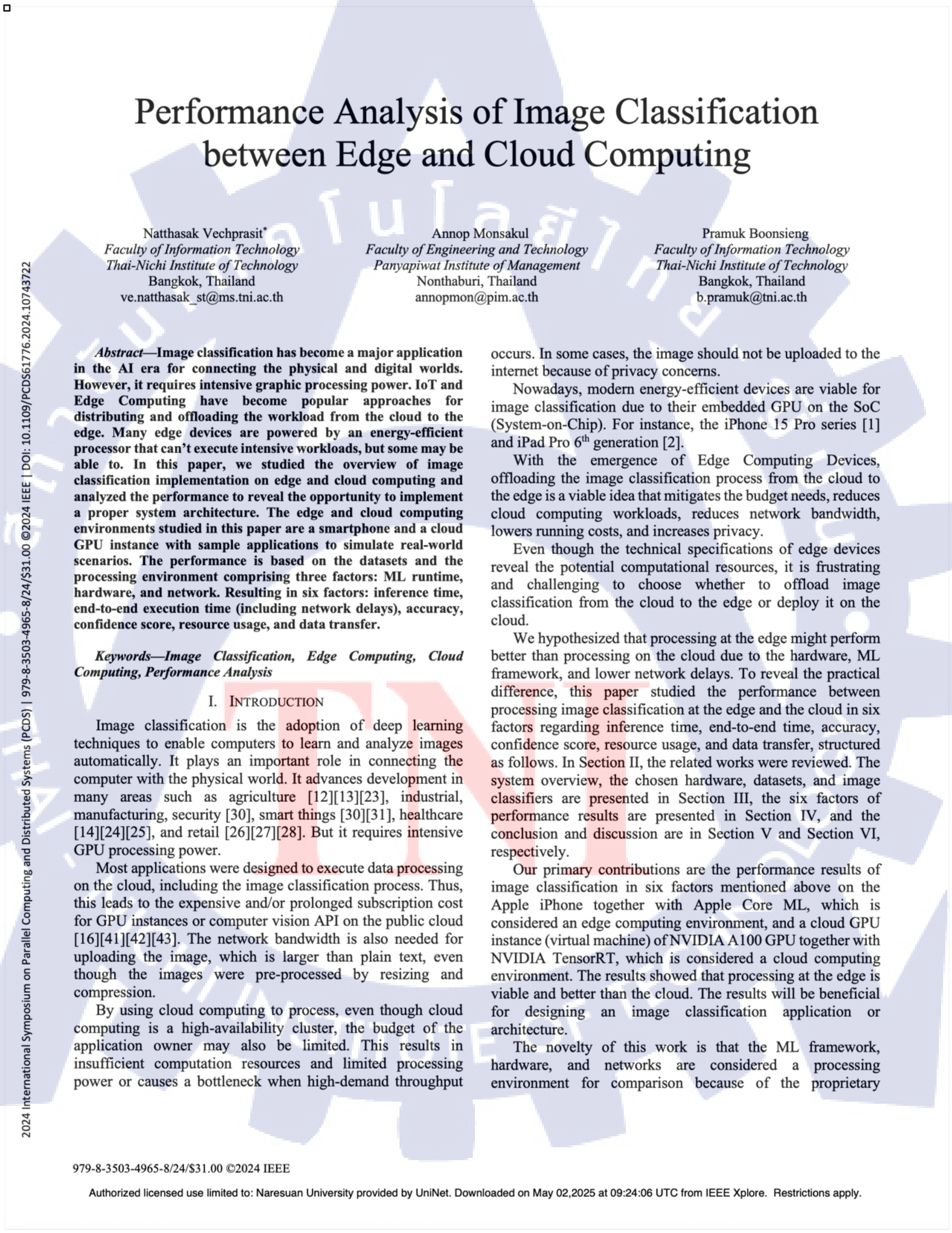
convert.py

```
import tensorflow as tf
from tensorflow.python.compiler.tensorrt import trt_convert as trt
import numpy as np

SAVED_MODEL_DIR = "models/outputs/tensorflow/20231129_053542"
OUTPUT_SAVED_MODEL_DIR =
"models/outputs/tensorrt/20231129_053542_fp32"

# Instantiate the TF-TRT converter
converter = trt.TrtGraphConverterV2(
   input_saved_model_dir=SAVED_MODEL_DIR,
   precision_mode=trt.TrtPrecisionMode.FP32
)

# Convert the outputs into TRT compatible segments
trt_func = converter.convert()
converter.summary()

# Define expected input size
# Input: 1 Image at a time, Image Size and Color Space = 180x180xRGB
def input_fn():
   yield [np.zeros((1, 180, 180, 3)).astype(np.float32)]

converter.build(input_fn=input_fn)
converter.save(output_saved_model_dir=OUTPUT_SAVED_MODEL_DIR)
```

Appendix H

Publication

The research article has been published in 2024 International Symposium on Parallel Computing and Distributed Systems (PCDS) and added to IEEE Xplore with DOI 10.1109/PCDS61776.2024.10743722 accessible via DOI link at https://doi.org/10.1109/PCDS61776.2024.10743722 or IEEE Xplore at https://ieeexplore.ieee.org/document/10743722



2024 International Symposium on Parallel Computing and Distributed Systems (PCDS) | 979-8-3503-4965-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/PCDS61776.2024.10743722

Performance Analysis of Image Classification between Edge and Cloud Computing

Natthasak Vechprasit*
Faculty of Information Technology
Thai-Nichi Institute of Technology
Bangkok, Thailand
ve.natthasak_st@ms.tni.ac.th

Annop Monsakul
Faculty of Engineering and Technology
Panyapiwat Institute of Management
Nonthaburi, Thailand
annopmon@pim.ac.th

Pramuk Boonsieng
Faculty of Information Technology
Thai-Nichi Institute of Technology
Bangkok, Thailand
b.pramuk@tni.ac.th

***Abstract*—Image classification has become a major application in the AI era for connecting the physical and digital worlds. However, it requires intensive graphic processing power. IoT and Edge Computing have become popular approaches for distributing and offloading the workload from the cloud to the edge. Many edge devices are powered by an energy-efficient processor that can't execute intensive workloads, but some may be able to. In this paper, we studied the overview of image classification implementation on edge and cloud computing and analyzed the performance to reveal the opportunity to implement a proper system architecture. The edge and cloud computing environments studied in this paper are a smartphone and a cloud GPU instance with sample applications to simulate real-world scenarios. The performance is based on the datasets and the processing environment comprising three factors: ML runtime, hardware, and network. Resulting in six factors: inference time, end-to-end execution time (including network delays), accuracy, confidence score, resource usage, and data transfer.**

Keywords—Image Classification, Edge Computing, Cloud Computing, Performance Analysis

I. Introduction

Image classification is the adoption of deep learning techniques to enable computers to learn and analyze images automatically. It plays an important role in connecting the computer with the physical world. It advances development in many areas such as agriculture [12][13][23], industrial, manufacturing, security [30], smart things [30][31], healthcare [14][24][25], and retail [26][27][28]. But it requires intensive GPU processing power.

Most applications were designed to execute data processing on the cloud, including the image classification process. Thus, this leads to the expensive and/or prolonged subscription cost for GPU instances or computer vision API on the public cloud [16][41][42][43]. The network bandwidth is also needed for uploading the image, which is larger than plain text, even though the images were pre-processed by resizing and compression.

By using cloud computing to process, even though cloud computing is a high-availability cluster, the budget of the application owner may also be limited. This results in insufficient computation resources and limited processing power or causes a bottleneck when high-demand throughput occurs. In some cases, the image should not be uploaded to the internet because of privacy concerns.

Nowadays, modern energy-efficient devices are viable for image classification due to their embedded GPU on the SoC (System-on-Chip). For instance, the iPhone 15 Pro series [1] and iPad Pro 6th generation [2].

With the emergence of Edge Computing Devices, offloading the image classification process from the cloud to the edge is a viable idea that mitigates the budget needs, reduces cloud computing workloads, reduces network bandwidth, lowers running costs, and increases privacy.

Even though the technical specifications of edge devices reveal the potential computational resources, it is frustrating and challenging to choose whether to offload image classification from the cloud to the edge or deploy it on the cloud.

We hypothesized that processing at the edge might perform better than processing on the cloud due to the hardware, ML framework, and lower network delays. To reveal the practical difference, this paper studied the performance between processing image classification at the edge and the cloud in six factors regarding inference time, end-to-end time, accuracy, confidence score, resource usage, and data transfer, structured as follows. In Section II, the related works were reviewed. The system overview, the chosen hardware, datasets, and image classifiers are presented in Section III, the six factors of performance results are presented in Section IV, and the conclusion and discussion are in Section V and Section VI, respectively.

Our primary contributions are the performance results of image classification in six factors mentioned above on the Apple iPhone together with Apple Core ML, which is considered an edge computing environment, and a cloud GPU instance (virtual machine) of NVIDIA A100 GPU together with NVIDIA TensorRT, which is considered a cloud computing environment. The results showed that processing at the edge is viable and better than the cloud. The results will be beneficial for designing an image classification application or architecture.

The novelty of this work is that the ML framework, hardware, and networks are considered a processing environment for comparison because of the proprietary optimization of the hardware, as represented in the conceptual framework in Fig 1.

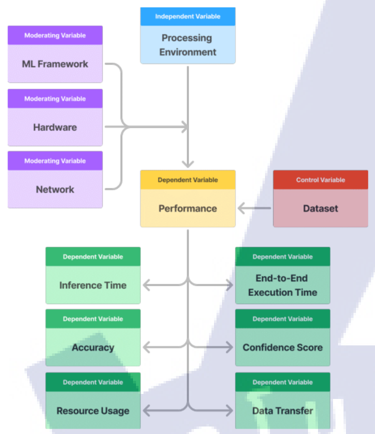


Fig. 1. The Conceptual Framework

II. Literature Review

A. *Edge Computing*

Edge computing is the evolution of technologies that allow computation to be performed at the edge of the network (which might be within the Local Area Network). For example, a smartphone, tablet, single board computer (SBC), and various Internet of Things (IoT) devices can be nodes for edge computing. Edge computing can perform computing offloading, data storage, caching, processing, etc., as much as an edge device or edge node can perform [3].

Examples of edge computing applications are Computer vision, natural language processing (NLP), network functions, Internet of Things (IoT), augmented reality (AR), and virtual reality (VR) [4].

B. *Image Classification*

Image classification is a computer vision task of assigning a label or class to an entire image. It can be done by using a type of artificial neural network that is used for image recognition and tasks that involve the processing of pixel data called Convolutional Neural Network (CNN). An image classifier model can be created by using Create ML (an application from Apple to create a Core ML model) [5], TensorFlow (an end-to-end opensource platform for machine learning) [6], Keras (a deep learning API written in Python and capable of running on top of either TensorFlow, Pytorch) [7], and Pytorch (an optimized tensor library for deep learning using GPUs and CPUs) [8].

C. *Running Machine Learning Model*

Hardware in edge computing is mostly designed for power efficiency as the priority. But as time passed by, the performance has improved over time. However, some of them have a GPU that embeds an optimization for running computer vision tasks and neural network tasks. For instance, Apple has proprietary technology for running machine learning models called “Core ML” [9]. It provides API for software developers to implement machine learning models to their application to run a machine learning model using GPU or NPU (Neural Processing Unit). Google also has an SDK called “ML Kit” [10] for running machine learning models on mobile devices, which supports both iOS and Android, whereas some Nvidia GPUs have a bunch of Tensor Cores for handling machine learning workloads and a software development kit that facilitates high-performance machine learning inference called “TensorRT” [11].

D. *Survey of Image Classification on Edge Computing and Performance Analysis*

Many researchers proposed the application of image classification on edge computing. Edge computing can be any device located at the edge of the network ranging from an energy-efficient device, a low-power device, an energy-efficient device with embedded GPU, a smartphone, to a high-end server.

Monburinon et al. proposed Deployment Environment Aware Learning (DEAL) framework to optimize model accuracy for detecting animals based on the edge device deployment location. They deployed a localized image classification model on Raspberry Pi 3 Model B to prevent animals from intruding into the agriculture field by classifying images from a camera. Their proposed system achieves a minimum of 0.32 seconds of inference time, 90% top-1 accuracy, and 96% top-3 accuracy, and the data transfer is reduced [12].

Zualkernan et al. evaluated the performance of various image classification models, including Inception V3, MobileNet V2 [32], ResNet-18 [34], EfficientNet B1 [35], DenseNet 121 [36]. Their results showed that Xception was the best image classifier model and was chosen to be deployed on edge devices to test the performance. They converted the model to TensorFlow Lite format to deploy on Rasberry Pi, Google Coral, and Nvidia Jetson Nano. And they also converted to TensorRT to deploy on Jetson Nano. The results showed that Nvidia Jetson Nano with TensorRT produced only 0.28 seconds of inference time which is the best in the test and a lot faster compared to other tested devices [13].

Charteros et al. developed an Android application that lets patients take a snapshot of their breast, and the application will analyze the breast cancer by a CNN model integrated within the application. They evaluated their application on Huawei Y6 Prime as a low-end device and Huawei Mate 20 Pro as a high-end device. They achieved around 20 seconds of inference time on Huawei Y6 Prime and around 12-14 seconds of inference time on Huawei Mate 20 Pro. And their model also achieved 98% accuracy in the identification of breasts and nipples [14].

Reza et al. evaluated the inference performance of several popular pre-trained convolutional neural networks (CNN) models, namely MobileNet V1 [33], MobileNet V2 [32], and

Inception V3 [37], on three edge computing devices: NVIDIA Jetson TX2, NVIDIA Jetson Nano, and Google Edge TPU [15].

The researchers from ETH Zurich, Google, Samsung, Huawei, Qualcomm, MediaTek, and Unisoc published an article that contains tables representing benchmarks of Android smartphones in the market in 2019. The benchmarks consist of the inference time performance of MobileNet v2 [32], Inception-ResNet [39], SRCNN [40], VGG-19 [38], and DPED [29].

III. Methodology

The researcher evaluated the performance based on the conceptual framework mentioned in the prior section, developed a simulation system as shown in Fig. 2, and proposed a four-step method as defined in the list below.

1) *Environment Setup*
2) *Dataset Preparation*
3) *Creating Image Classifier Model*
4) *Image Classification Application and Runtimes*

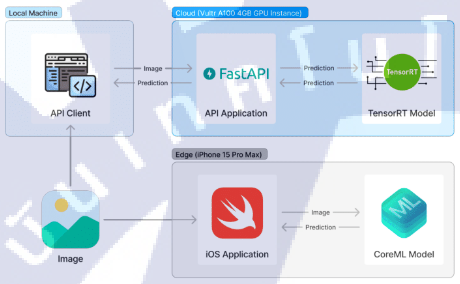


Fig. 2. System Overview of Image Classification on Edge and Cloud Using Customized Model

A. *Environment Setup*

For the edge computing environment, the Apple iPhone 15 Pro series, together with Apple Core ML, was chosen to represent a mobile application scenario in which image classification is executed on the device. It was chosen to represent the smartphone category because its SoC seems to have sufficient graphic processing power and a native ML runtime environment, which the researcher anticipates will demonstrate strong performance because of its proprietary optimization.

For the cloud computing environment, the lowest plan of an Nvidia A100 Cloud GPU instance from Vultr [16] (4 GB out of 80 GB of Nvidia A100 GPU Memory) together with TensorRT was chosen to represent a web service or an API application deployed on a remote server that receives the image from a remote client to classify. It was chosen to represent how well a budget-friendly cloud GPU instance can perform.

Table I compares the configuration and specification of edge and cloud computing environments for image classification performance evaluation.

TABLE I. Environment Setup

Component	Environment	
	Edge	Cloud
Hardware / Device	iPhone 15 Pro Max	Vultr GPU Instance Nvidia A100 (4 GB)
Processor / SoC	Apple A17 Pro	Intel Xeon (Vultr Generic)
CPU Cores / vCPU	2 Performance cores 4 Efficiency cores	1 vCPU
Memory	8 GB (Unified)	6 GB
GPU	Apple A17 Pro	Nvidia A100
GPU Memory	8 GB (Unified)	4 GB
Operating System	iOS 17	Nvidia NGC (Ubuntu Linux 22.04)
Application	Swift	REST API
Network	Not Required	FTTx Broadband
ML Runtime	Core ML	TensorRT

B. *Dataset Preparation*

In this paper, two sets of data featuring images of dogs and cats were used.

The first dataset is the Kaggle: Cats VS Dog dataset [17]. It is used for training and validation, It contains 12,491 images of cats and 12,470 images of dogs. This dataset was chosen for training and validation because its photo consists of mixed backgrounds, postures, and image sizes. It also has plenty of images for training and validation. Fig. 3 represents the sample data of the training and validation dataset.

Another dataset is Animal Faces-HQ (AFHQ) [18]. It is used for testing. It consists of high-quality images. The images are 512-by-512 pixels resolution. Only images of cats and dogs in the training subset were chosen. Each class contains about 5,000 images. This dataset was chosen because there was a high number of photos to simulate to gain a stable statistic, and its quality is consistently good. Fig. 4. represents the sample data of the testing dataset.

Fig. 3. Sample data of Kaggle Cats VS Dogs dataset

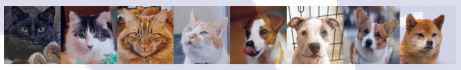


Fig. 4. Sample data of Animal Faces-HQ dataset

C. *Creating Image Classifier Model*

The model varies according to the environment setup. Core ML was used in the edge environment, while TensorRT was used in the cloud environment because of the proprietary optimization of hardware and ML framework.

Our Core ML image classifier model was created using Create ML, an application that allows developers to create a customized model via the user interface. The validation dataset was randomly partitioned by 5% based on Core ML default. Image Feature Print V2 [19], a feature extractor provided by Create ML, was applied. The target iteration was set to 25 iterations. No augmentations were applied.

Our TensorRT image classifier model was created using a small Xception network [23]. The target iteration was set to 50 iterations with an early stop applied. The training will stop if the model performance stops improving within the next 8 iterations and the best iteration is restored to be the final output. The training set was augmented by random horizontal flips and random rotation. The validation dataset was partitioned by 5%, which is similar to our Core ML model. Keras outputs a model in TensorFlow format, and it needs to be converted to a TensorRT model format to fully take the benefits from the Nvidia GPU driver and Tensor Core to accelerate the inference time.

Table II represents the configuration for the training for each environment.

The main goal of both image classifier models is to classify an input image into one of two classes, which are cat and dog.

TABLE II. TRAINING CONFIGURATION

Description	Training Configuration	
	Core ML	Keras
Target Iteration	25	50
Early Stop Patient	Auto	8
Validation Split Rate	Auto (5%)	5%
Augmentation	None	- Random Flip - Random Rotation

D. *Image Classification Applications and Runtimes*

For the edge environment, a Swift [20] application was developed to run the image classification on iOS. Fig. 5. illustrates the activity or flow of the iOS application. The application lets the user choose a photo album that contains the testing dataset. After the user chooses a photo album, all the photos or images in the album and an output handler function are passed to the Core ML model to infer and handle the prediction results.

For the cloud environment, a web service REST API [44] application was developed using FastAPI [21], a fast and lightweight web application server API written in Python. Fig. 6 illustrates the activity or flow of the API application. The application receives the image uploaded by a user and passes it to the TensorRT model to infer and return the prediction results.

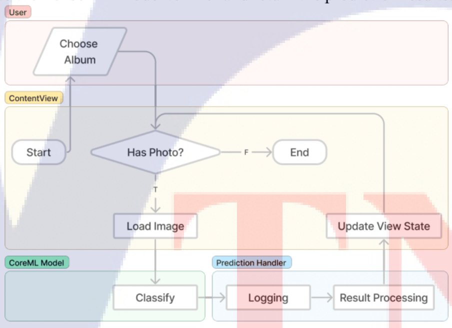


Fig. 5. Activity diagram of the iOS application.

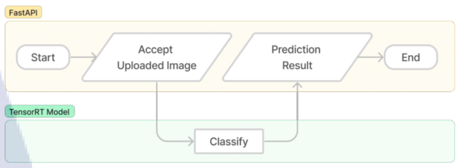


Fig. 6. Activity diagram of the API application

IV. EXPERIMENT, PERFORMANCE EVALUATION, AND RESULTS

There are six factors of performance studied in this paper that would reveal the characteristics of each environment. Those are:

1) *Inference Time*
2) *End-to-End Execution Time*
3) *Accuracy*
4) *Confidence Score*
5) *Resource Usage*
6) *Data Transfer*

Fig. 7. And Fig. 8. illustrate the overview of how the experiment was set up and how the prediction results were sampled.

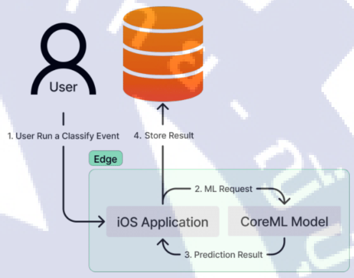


Fig. 7. Result gathering flow for the edge application.

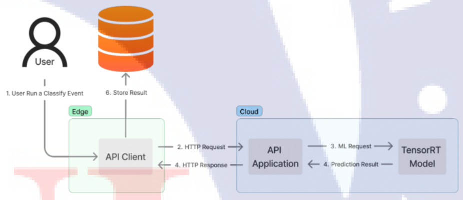


Fig. 8. Result gathering flow for the cloud application.

A. *Inference Time*

Inference Time is the duration when the image classifier begins classifying the image and ends when the results are returned.

For the edge environment, a timestamp was captured after an image was loaded and before passing to the classify function, and another timestamp was captured after the model predicted and finished handling the results. Then, the difference between the two captured timestamps was calculated and converted to milliseconds.

For the cloud environment, a timestamp was captured once the API application received the uploaded image, and another timestamp was captured once the model returned the prediction results. Then, the difference between the two captured timestamps was calculated and converted to milliseconds.

Table III represents the minimum, average, maximum, and standard deviation of edge and cloud computing inference time measured in milliseconds. Lower is better. The image used for the evaluation is Animal Faces-HQ. The standard deviation (S.D.) in the table indicates the amount of variation in the expected inference time. Fig. 9 represents a comparison chart of inference time between edge and cloud.

B. *End-to-End Execution Time*

End-to-end execution Time is the duration when the system receives the user's input image, passes it to the image classifier to classify, waits for the prediction returns, and ends. This includes the network delay, if applicable.

For the edge environment, a timestamp was captured once the image began to load, and another timestamp was captured once the results were printed. The difference between the two captured timestamps was calculated and converted to milliseconds.

For the cloud environment, an API client was developed by using Java version 17 (Amazon Corretto 17 aarch64) to run the test. The client invokes the API by making HTTP requests to the API application. This approach simulates the real-world application that submits data to process on the remote server. The cloud GPU instance was deployed in Tokyo, Japan, based on the availability region of the cloud service provider, while the client initiated HTTP requests from Bangkok, Thailand. The distance between these two cities is around 4,600 km. The network latency was approximately 100 (±1) milliseconds over the FTTx broadband connection. The latency was roughly measured by a ping test with 10 intervals. A timestamp was captured once the API client executed an HTTP request, and another timestamp was captured once the API client received an HTTP response that contained the prediction results. Then, the difference of the two captured timestamps was calculated and converted to milliseconds. All of the images were resized from 512-by-512 pixels to 256-by-256 pixels before being uploaded to the cloud.

Table IV shows the minimum, average, maximum, and standard deviation of the end-to-end execution time of edge and cloud computing measured in milliseconds. Lower is better. The image used for the evaluation is Animal Faces-HQ. The standard deviation (S.D.) in the table indicates the amount of variation in the expected inference time.

Fig. 10 represents a comparison chart of end-to-end execution time between edge and cloud. We also compare it with inference time, represented in Fig. 11.

TABLE III. INFERENCE TIME RESULTS

Environment	**Results (milliseconds – lower is better)**			
	Min	**Average**	**Max**	**S.D.**
Edge	5.02 ms	16.30 ms	1175.30 ms	11.74 ms
Cloud	1.12 ms	1.44 ms	16.59 ms	0.47 ms

TABLE IV. END-TO-END EXECUTION TIME RESULTS

Environment	**Results (milliseconds – lower is better)**			
	Min	**Average**	**Max**	**S.D.**
Edge	5.05 ms	16.47 ms	1175.35 ms	11.75 ms
Cloud	188 ms	289.16 ms	583 ms	53.44 ms

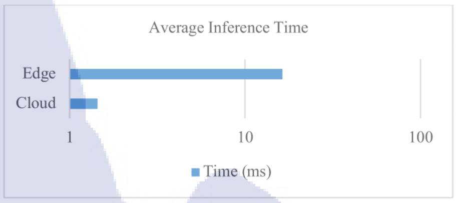


Fig. 9. Comparison of Average Inference Time between Edge and Cloud

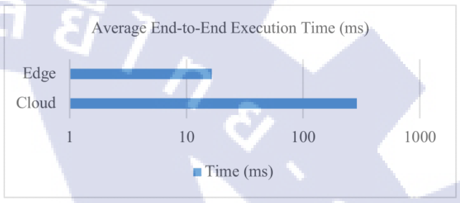


Fig. 10. Comparison of Average End-to-End Execution Time between Edge and Cloud

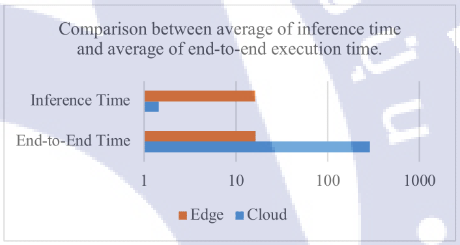


Fig. 11. Comparison of Inference Time and End-to-End Execution Time between Edge and Cloud

C. *Accuracy*

Table V shows the iterations or epochs configuration for image classifier model training. The maximum iterations for the Core ML model were configured to 25 iterations, but the model converged at iteration 11, as also visualized in Fig. 12. The maximum iterations for training with Keras were configured to 50 iterations, and the early stop patience was configured to observe for 8 continuously unimproved iterations. The

Xception model trained using Keras was converged at iteration 34, as also visualized in Fig. 13.

Table VI shows the training, validation, and testing accuracy of both image classifiers, as expressed below in (1).

$$Accuracy = \frac{Correct\ Prediction}{All\ Prediction} \quad (1)$$

The result is Core ML model achieved 99.74% testing accuracy, which is better than the Keras Xception model, which achieved 95.46% testing accuracy.

TABLE V. TRAINING RESULTS

ML Framework	Iterations / Epochs	
	Max	Convergence
Core ML	25	11
Keras	50	34

TABLE VI. TRAINING, VALIDATION, AND TESTING ACCURACY

ML Framework	Training	Validation	Testing
Core ML	99.0%	98.4%	99.74%
Keras	97.7%	96.7%	95.46%

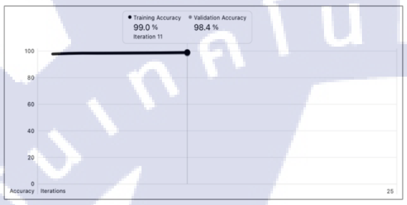


Fig. 12. Core ML Model Training and Validation Accuracy

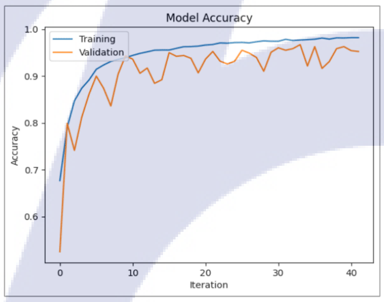


Fig. 13. Keras Model Training and Validation Accuracy

Tables VII, VIII, and IX show the confusion matrix of the models, and Table X shows the model's Precision, Recall, and F1-Score, as expressed below in (2), (3), and (4), respectively.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \times 100 \quad (2)$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \times 100 \quad (3)$$

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} \times 100 \quad (4)$$

TABLE VII. CORE ML MODEL CONFUSION MATRIX

Actual Class	Predicted Class		Total
	Dog	Cat	
Dog	4718	21	4739
Cat	4	5148	5153

TABLE VIII. CORE ML MODEL PERFORMANCE

Class	Core ML Performance		
	Precision	Recall	F1-Score
Dog	99.92%	99.56%	0.997
Cat	99.59%	99.92%	0.998

TABLE IX. KERAS MODEL CONFUSION MATRIX

Actual Class	Predicted Class		Total
	Dog	Cat	
Dog	4421	318	4739
Cat	131	5022	5153

TABLE X. KERAS MODEL PERFORMANCE

Class	Keras Performance		
	Precision	Recall	F1-Score
Dog	97.12%	93.29%	0.952
Cat	94.04%	97.46%	0.972

D. Confidence Score

Table XI represents the minimum, average, maximum, and standard deviation of the confidence score of the inferences.

TABLE XI. CONFIDENCE SCORE STATS

Environment	Confidence Score Stats			
	Min	Average	Max	S.D.
Edge	50.34%	99.9%	100%	1.67%
Cloud	50.03%	94.95%	100%	10.06%

E. Resource Usage

The resource usage of the edge application in Table XII was monitored by Xcode while running the iOS application that runs the Core ML image classifier model. The gathered resource usage that Xcode provided includes CPU, Unified Memory, and Energy Impact.

The resource usage of the cloud application in Table XIII was monitored by monitoring tools available in Ubuntu Linux, such as htop, Cockpit, and nvidia-smi. The CPU, Memory, and GPU Memory usage were gathered. The energy impact value is inaccessible from inside the GPU instance.

TABLE XII. RESOURCE USAGE OF EDGE APPLICATION

Resource	Usage
CPU	14 ~ 17%
Unified Memory	100 ~ 400 MB
Energy Impact	High ~ Very High

TABLE XIII. RESOURCE USAGE OF CLOUD APPLICATION

Resource	Usage
CPU	18 ~ 38% (of 1 vCPU)
Memory	3.1 GB
GPU Memory	1,765 MB
Energy Impact	Not Applicable

F. Data Transfer

The data transfer was statistically measured by the size of the images used for testing (the AFHQ dataset) but did not include the HTTP payload and other overheads. The images uploaded to the cloud are high-quality JPEGs resized to 256-by-256 pixels, but the original size was 512-by-512 pixels. The color profile is sRGB IEC61966-2.1.

Table XIV shows the minimum, average, maximum, and standard deviation of the size of the images of the AFHQ dataset.

TABLE XIV. JPEG FILE SIZE

Resolution (pixels)	Image Size (Kilobytes)			
	Min	Average	Max	S.D.
256 x 256	3.49	13.74	32.46	3.42
512 x 512	13.95	42.32	112.68	12.05

V. CONCLUSION

In this paper, we simulated the environments and applications of image classification on edge and cloud computing based on hardware, ML framework, and network to evaluate and analyze the performance, including inference time, end-to-end execution time, accuracy, confidence score, resource usage, and data transfer.

We found that the inference time of image classification on the cloud was faster than at the edge. However, the end-to-end execution time of image classification at the edge was faster than the cloud because, at the edge, the network delays were eliminated. In our case, the distance between the application and the cloud significantly impacts the propagation delay, which leads to higher execution time in the cloud environment. The cloud GPU instance was running in Tokyo while our application was running in Bangkok. The network delay measured by the ping test was 100 (±1) milliseconds. Thus, executing image classification at the edge is significantly faster. The size of data uploaded to the cloud has no significant impact on high-speed internet connections such as FTTx, broadband, and 4G/5G cellular networks (without a Fair-Usage Policy or a bandwidth limit policy applied).

The accuracy of the image classifier model created using Core ML is slightly greater than the small Xception network model created using Keras. Based on testing, Core ML has 99.75% accuracy, and Keras Xception has 95.46%. The average confidence score of both models was 99.9% and 98.0%, respectively. This is because Core ML works best with images of real-world objects since it has an image feature extractor pre-trained by millions of images, and the datasets used in this work are also composed of common real-world objects.

We also measured the resource usage on edge devices and found that it highly impacts energy.

VI. DISCUSSION

Offloading image classification to the edge eliminates the network delays required for federating data to process on the cloud (or the remote server). This makes image classification faster, which is impactful for time-critical applications. It also reduces cloud resources and subscription costs.

If the cloud is close to the application, for instance, if the cloud and the application were in the same city and connected via FTTx, the network propagation delay might range from 1 to 10 ms, leading to an insignificant difference in end-to-end execution time between the edge and the cloud. However, offloading image classification to the edge offers the benefit of cloud resource reduction while still maintaining the application's speed and experience.

When considering offloading the image classification to the edge, the performance and availability of the edge devices, the scenarios, and the use cases must be considered.

Besides the execution speed, the model accuracy must also be considered. Our experiment dataset was controlled by real-world dog and cat images, so Core ML has an advantage. However, the model accuracy must be re-evaluated every time the dataset changes.

Inference time and end-to-end execution time significantly impact the speed of the image classification system or application and user experiences. Accuracy affects the correctness of prediction. The confidence score could be used as a threshold in the application to display the prediction label confidently. Resource usage shows how much resource the image classification consumed, including CPU, memory, and energy, which will be related to the power consumption. Data transfer impacts how much network bandwidth will be used.

VII. FUTURE WORKS

We look forward to bringing more environments into account to make a wider analysis. Those environments we are willing to analyze include the representative of Android smartphones with ML Kit [10] running TensorFlow Lite [45], the Nvidia Jetson series [46] with TensorRT [11] as a representative of Fog Computing [47][48][49][50] environment, and/or other GPU or devices that may be related. Also, the load testing and stress testing of Image Classification applications on Fog Computing and Cloud Computing are worth evaluating because they will reveal how much the devices can handle and what will be affected.

ACKNOWLEDGMENT

This research work is supported by the Artificial Intelligence & Internet of Things Research Laboratory under the administration of Sarayut Nonsiri, Ph.D., Faculty of Information Technology, Thai-Nichi Institute of Technology.

REFERENCES

[1] iPhone 15 Pro – Technical Specifications. https://support.apple.com/kb/SP903?locale=en_US

[2] iPad Pro 12.9-inch (6th generation) – Technical Specifications. https://support.apple.com/kb/SP883?viewlocale=en_US

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, "Edge Computing: Vision and Challenges", IEEE Internet of Things Journal, Vol. 3, No. 5, p.638, October 2016.

[4] Jiasi Chen and Xukan Ran. "Deep Learning With Edge Computer: A Review". Proceedings of the IEEE, Vol. 107, No.8, p.1655-1674. August 2019.

[5] Create ML. https://developer.apple.com/machine-learning/create-ml

[6] TensorFlow. https://www.tensorflow.org/

[7] Keras. https://keras.io/

[8] PyTorch. https://pytorch.org/

[9] Core ML. https://developer.apple.com/machine-learning/core-ml/

[10] ML Kit. https://developers.google.com/ml-kit

[11] TensorRT. https://developer.nvidia.com/tensorrt

[12] N. Monburinon, S. Zabir, N. Vechprasit, S. Utsumi, N. Shiratori., "A Novel Hierarchical Edge Computing Solution Based on Deep Learning for Distributed Image Recognition in IoT Systems". 4th International Conference on Information Technology (InCIT), p.294-299. October 2019.

[13] I. Zualkerman et al., "An IoT System Using Deep Learning to Classify Camera Trap Images on the Edge". Computers 2022, 11, 13. January 2022.

[14] E. Charteros and I. Koutsopoulos, "Edge Computing for Having an Edge on Cancer Treatment: a Mobile App for Breast Image Analysis". 2020 IEEE International Conference on Communications Workshops (ICC Workshops). June 2020.

[15] S. Reza, Y. Yan, X. Dong, L. Qian, "Inference Performance Comparison of Convolutional Neural Networks on Edge Devices". 6th EAI International Conference," SmartCity360°, LNICST 372, pp.323-355, 2021

[16] Vultr A100. https://www.vultr.com/es/news/Affordable-Cloud-VMs-Accelerated-with-NVIDIA-GPUs/

[17] Microsoft and PetFinder.com Kaggle Cats and Dogs Dataset. https://www.kaggle.com/datasets/karakaggle/kaggle-cat-vs-dog-dataset

[18] Y. Choi, Y. Uh, J. Yoo, J. Ha, "StarGAN v2: Diverse Image Synthesis for Multiple Domains". Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2020.

[19] Image Feature Print V2. https://developer.apple.com/documentation/CreateMLcomponents/imagefeatureprint/revision

[20] Swift. https://developer.apple.com/swift/

[21] FastAPI. https://fastapi.tiangolo.com/

[22] Keras Image Classification from Scratch. https://keras.io/examples/vision/image_classification_from_scratch

[23] A. Olsen et al., "DeepWeeds: A Multiclass Weed Species Image Dataset for Deep Learning". Scientific Reports 9:2058, February 2019

[24] K. Muhammad, S. Khan, J. Ser, V. Albuquerque, "Deep Learning for Multigrade Brain Tumor Classification in Smart Healthcare Systems: A Prospective Survey". IEEE Transactions on Neural Network and Learning Systems, Vol. 32, No. 2. February 2021.

[25] K. Iqtidar, A. Iqtidar, W. Ali, S. Aziz, M. Khan, "Image Pattern Analysis towards Classification of Skin Cancer through Dermoscopic Images". 2020 First International Conference of Smart Systems and Emerging Technologies (SMARTTECH). 2020

[26] P. Rujakon, P. Takam, S. Sinthupuan, P. Khoenkaw, K. Dullayachai, T. Chaiya. "Retail Management on Mobile Application using Product Classification". 19th ECTI-CON, 2020.

[27] L. Liu, B. Zhou, Z. Zou, S. Yeh, L. Zheng, "A Smart Unstaffed Retail Shop Based on Artificial Intelligence and IoT". IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD). 2018.

[28] A. Savit, A. Damor. "Revolutionizing Retail Stores with Computer Vision and Edge AI: A Novel Shelf Management System". The 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC), 2023.

[29] A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. wang, F. Baum, M. Wu, L. Xu, L. Gool, "AI Benchmark: All About Deep Learning on Smartphones in 2019". IEEE/CVF International Conference on Computer Vision workshop (ICCVW), p.3617-3635, 2019.

[30] G. Stalin, S. Anand, "Intelligent Smart Home Security System: A Deep Learning Approach". IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC), 2022.

[31] S. Khan, Y. Teng, J. Cui, "Pedestrian Traffic Lights Classification Using Transfer Learning in Smart City Application". 13th International Conference on Communication Software and Networks. 2021.

[32] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4510–4520, 2018.

[33] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". https://doi.org/10.48550/arXiv.1704.04861

[34] He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.

[35] Tan, M.; Le, Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; Chaudhuri, K., Salakhutdinov, R., Eds.; PMLR: Cambridge, MA, USA, 2019; Volume 97, pp. 6105–6114.

[36] 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 (CVPR), 2017, pp. 2261-2269. DOI: 10.1109/CVPR.2017.243

[37] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2818-2826. DOI: 10.1109/CVPR.2016.308

[38] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1646–1654, 2016.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[40] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2016.

[41] Amazon EC2 P4 Instances. https://aws.amazon.com/ec2/instance-types/p4/

[42] Google Cloud GPU pricing. https://cloud.google.com/compute/gpus-pricing

[43] Azure VM ND A100 v4-series. https://learn.microsoft.com/en-us/azure/virtual-machines/nda100-v4-series

[44] Fielding, Roy Thomas. "Architectural Styles and the Design of Network-based Software Architectures." Doctoral dissertation, University of California, Irvine, 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[45] TensorFlow Lite. https://www.tensorflow.org/lite

[46] NVIDIA Jetson. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/

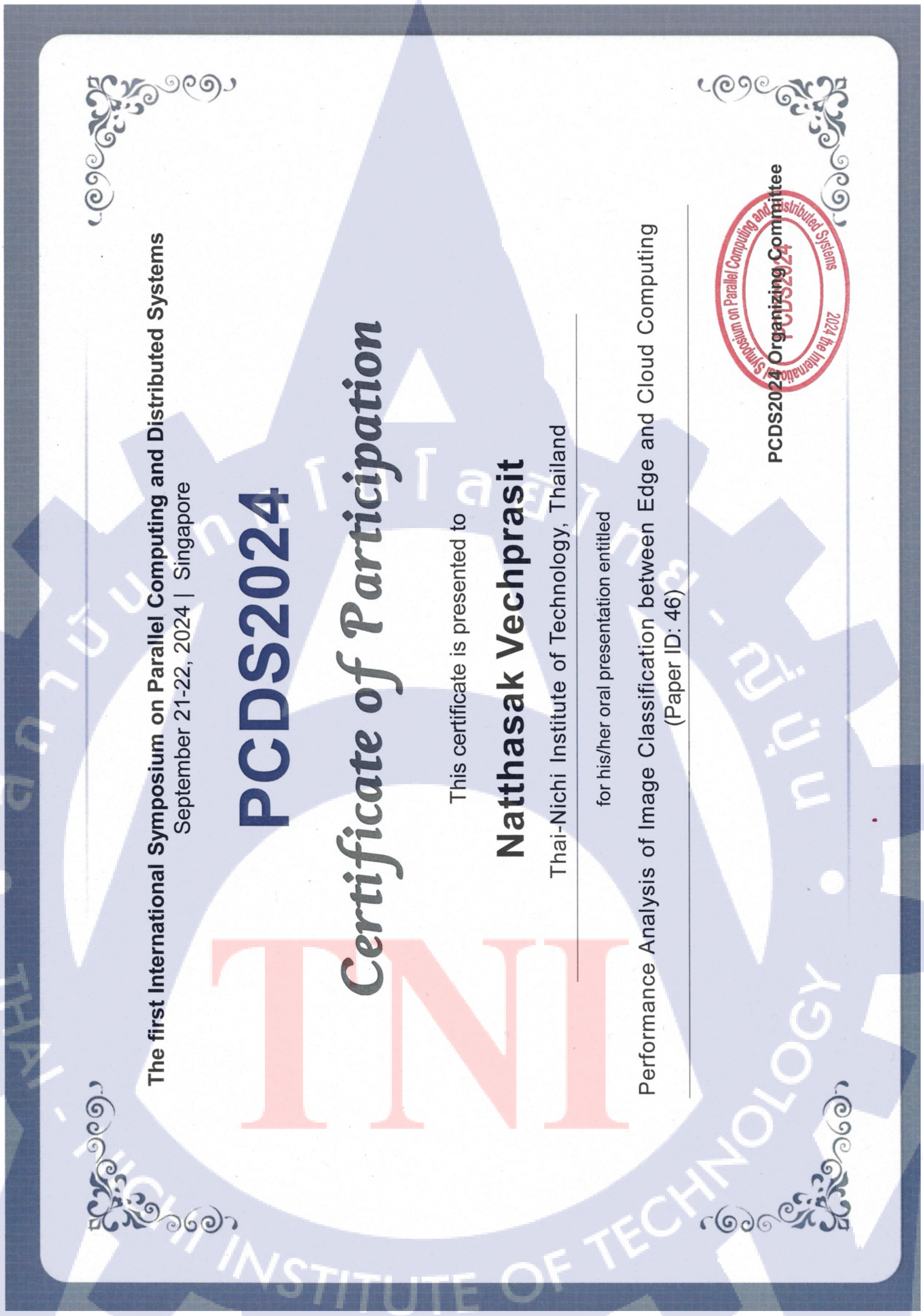
[47] Hathal Salamah A. Alwageed, "FOG Computing: The new Paradigm". Communications on Applied Electronics (CAE), Vol.3, No. 5, November 2015

[48] M. Iorga, L. Feldman, R. Barton, M. Martin, N. Goren, C. Mahmoudi, "Fog Computing Conceptual Model", NIST Special Publication 500-325, March 2018. https://doi.org/10.6028/NIST.SP.500-325

[49] A.V. Dastjerdi, H. Gupta, R.N. Calheiros, S.K. Ghosh, R. Buyya. "Chapter 4 - Fog Computing: Principles, Architectures, and Applications". Internet of Things Principles and Paradigms, ISBN 9780128053959, p. 61-75. 2016. https://doi.org/10.1016/B978-0-12-805395-9.00004-6

[50] R. Mahmud, R. Kotagiri, R. Buyya, "Fog Computing: A Taxonomy, Survey and Future Directions". Internet of Everything. Internet of Things. Springer, Singapore, 2018. https://doi.org/10.1007/978-981-10-5861-5_5

[51] Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015.



The first International Symposium on Parallel Computing and Distributed Systems
September 21-22, 2024 | Singapore

PCDS2024

Certificate of Participation

This certificate is presented to

Natthasak Vechprasit

Thai-Nichi Institute of Technology, Thailand

for his/her oral presentation entitled

Performance Analysis of Image Classification between Edge and Cloud Computing
(Paper ID: 46)

2024 the International Symposium on Parallel Computing and Distributed Systems
PCDS2024

PCDS2024 Organizing Committee