International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. #, No.# (Mon-20..)

Offloading of Indoor Positioning System Using Cloudlet Framework

Sinjiru Setyawan[1], Gede Putra Kusuma[2]

[1,2] Computer Science Department, BINUS Graduate Program - Master of Computer Science, Bina Nusantara University, Jakarta, Indonesia, 11480

E-mail address: sinjiru.setyawan@binus.ac.id[1], inegara@binus.edu[2]

Received ## Mon. 20##, Revised ## Mon. 20##, Accepted ## Mon. 20##, Published ## Mon. 20##

Abstract: This study evaluates the performance of Indoor Positioning System (IPS) by combining offloading and fingerprinting techniques using Convolutional Neural Network (CNN) implementing the cloudlet framework. The aim of the research are to measure the prediction time and battery consumption at various prediction execution location of the CNN model. The execution locations are mobile device (MD), cloudlet, cloud, and using automatic prediction execution location selection. The Received Signal Strength Indication (RSSI) data from 8 Bluetooth Low Energy (BLE) beacons are collected using an Android application and will be used to train the CNN model. The trained CNN model is then used as radio map to predict the coordinates of the MD. Evaluation is conducted by measuring the battery consumption and prediction speed over 30 minutes while continuously running predictions at four execution locations. The prediction speed is measured from the start of the prediction until the end of the prediction. The study results show that the prediction approach using the cloudlet is more efficient on battery consumption compared to other execution locations. Additionally, the proposed automatic selection method of selecting prediction execution location demonstrates faster execution speed compared to performing the prediction s at a single location. These findings are important for understanding the balance between prediction speed and battery consumption efficiency.

Keywords: Indoor Positioning System, Bluetooth Low Energy, Cloudlet Framework, Offloading Method, Convolution Neural Network

1. INTRODUCTION

The increasing development of wireless technology has made indoor positioning technology an interesting topic to discuss. Indoor Positioning System (IPS) technology has various applications in fields such as asset tracking, animal tracking, and table number tracking in the restaurant industry [1]. IPS technology addresses the limitations of existing location tracking technologies like GPS. The Global Positioning System (GPS) utilizes signals from multiple GPS satellites to determine the receiver's position. However, GPS has limitations, such as low accuracy indoors due to obstacles like building walls, signal interference, diverse building structures, and unreliable indoor communication [2]. IPS technology becomes essential to determine the position of objects in indoor environment where GPS signals cannot reach.

To overcome the challenges of indoor tracking, several technologies have been developed, including Frequency Modulation (FM), Ultra-wideband (UWB), Radio Frequency Identification (RFID), WiFi, Bluetooth, Zigbee, and cellular networks (LTE & 5G) [3]. Bluetooth, particularly Bluetooth Low Energy (BLE), is commonly used as an IPS infrastructure due to its relatively high accuracy (±1-3 meters)[3], affordability, ease of access, fast data transfer speeds, and widespread infrastructure, such as BLE-enabled devices in smartphones.

In IPS, one method to determine the location of a target device is by utilizing Received Signal Strength Indication (RSSI). RSSI is an indicator of the signal strength received by a device. It can be used to estimate the location of a device by measuring the signal strength from a transmitter or base station to a receiving device. This method is known as proximity localization[3]. However, this method can only measure the distance between the device and the signal transmitter, not the actual direction of the location. Another method utilizing RSSI is fingerprinting, which requires multiple fixed base stations. The device measures the RSSI at various points, creating a radio map used for location tracking based on received RSSI.

E-mail:author's email

Fingerprinting in IPS requires processing the collected RSSI data from various points to create a radio map for tracking. In some studies, deep learning methods, such as Convolutional Neural Networks (CNN), are employed for fingerprinting [4]. CNNs generate a training model that is then used on the target device to estimate its location based on received BLE signals. The use of CNNs in model creation and location prediction demands significant computational power. When implemented on devices with limited computing capabilities, like smartphones (as seen in [5]), it may lead to noticeable delays in location tracking and increased battery consumption.

To address the challenges of devices with limited computing power or limited power sources, one applicable method is offloading. Offloading involves transferring computational tasks to devices with higher computing capabilities. Various offloading frameworks, such as cloudlet and cloudclone, can be used in IPS, each with its own advantages and disadvantages. Offloading illustration can be seen in Figure 1.

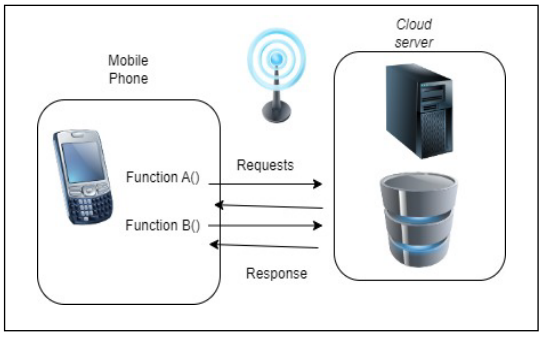


Figure 1 Offloading Illustration

This research aimed on to implement and evaluate the offloading method in the prediction process to determine the position of an individual. The design used consists of microcontroller as BLE beacon, smartphone as BLE signal receiver, a cloudlet server and cloud server as prediction server that use CNN model as the prediction processor. We proposed to use cloudlet as an offloading method because the prediction location is at the edge of local network with lower network latency than cloud. With said benefit, using cloudlet as offloading method was chosen with the hope of better energy efficiency and faster prediction

2. PREVIOUS WORKS

Research related to computation offloading on mobile devices has been conducted using various offloading methods and frameworks. For applications requiring low latency, commonly used frameworks include CloudLet, Mobile Edge Computing (MEC), and Edge Computing. However, for applications not requiring low latency, offloading methods to Cloud Servers are typically employed. Most studies on offloading focus on mobile application offloading, with limited exploration of offloading implementation in Indoor Positioning System (IPS) systems.

A study by Marwa Zamzam, Tallal Elshabrawy, and Mohamed Ashour [6] measured latency and energy consumption by offloading IPS to Mobile Edge Computing and Cloud. Three offloading algorithms (ILLOO, ILLCO, ILLGO) were compared, demonstrating a 70.36% reduction in processing time when using MEC/Cloud compared to local processing on the mobile device. Energy consumption was also 50% more efficient with offloading than with local computation on the mobile device.

In another study by Dimitrios Spatharakis et al. (2020) [7], offloading was applied to a Location-Based Service with a mobile device as the offloading object. Image detection processing using an edge server showed a response time of 5 seconds compared to 30-60 seconds for local processing, indicating a 5-6 times improvement with the edge server as the offloading server.

Yanjun Guo, Liqiang Zhao, Yong Wang, Qi Liu, and Jiahui Qiu [8] implemented fog computing as an offloading solution. Offloading decisions were made at runtime, considering the nearest computational node. Results showed a 1-second delay in cloud computing, 0.5 seconds using fog computing, and 0.2 seconds in distributed fog computing.

Using game theory for offloading determination, Marwa Zamzam, Tallal El-Shabrawy, and Mohammed Ashour [6] achieved a 50% energy saving with the game computation offloading algorithm compared to local computing and random offloading.

Imran A. Zualkernan and Mohammed Towheed [9] compared power consumption and CPU utility for voice prediction using CNN on a smartwatch. Edge computing resulted in a larger battery capacity drop due to network load, with a 14.8% faster battery capacity drop compared to local computation on the smartwatch.

Jude Vivek Joseph, Jeongho Kwak, and George Iosifidis [10] implemented offloading strategies considering parameters such as transmission delay, CPU clock, tasks, and network path, achieving a 50% reduction in power consumption.

Saif U. R. Malik, Hina Akram, Sukhpal Singh Gill, Haris Pervaiz, and Hassan Malik [11] applied the CloudClone framework, showing a 10% difference in battery consumption for tested mobile devices with offloading resulting in lower energy consumption.

Sudip Misra, Bernd E. Wolfinger, Achuthananda M. P., Tuhin Chakraborty, Sankar N. Das, and Snigdha Das [12] used auction systems for offloading with cloudlet and

cloud server architecture. Execution duration for cloudlet was higher than for MD and cloud, measured in milliseconds (ms), while energy consumption in MD was greater when offloading for tasks exceeding 20,000.

Tao Huang, Feng Ruan, Shengjun Xue, Lianyong Qi, and Yucong Duan [13] compared cloud, cloudlet, and NSDE methods in multimedia workflow, achieving smaller power consumption and processing time with cloudlet and NSDE algorithms.

In the final study by S. Erana Veerappa Dinesh and K. Valarmathi [14], direct cloud offloading strategy considering battery, network, CPU, and bandwidth resulted in 33.3% energy savings (Joule).

Based on existing research, several studies compare offloading to the cloud. However, studies in [6], [7], [8], [9], [10], [12], [13] utilize Edge Computing or Mobile Edge Computing, also known as cloudlet, where the architecture is positioned closer to the mobile device (MD). The goal of using this framework is to bring the computing server closer, reducing the time needed for data transportation over the internet to the cloud server.

Among the studies utilizing cloudlet or edge computing, study number [9] focused on smartwatch devices using TensorFlow.js running through a web browser. The results indicated higher battery consumption on the edge computer due to additional load during data transmission to the edge server. Consequently, the performance when running the application through the smartwatch itself became more efficient.

From the available research, cloudlet appears to meet the requirements for IPS research due to its proximity to users, providing faster processing response times. However, cloud technology should not be ignored. Cloud can serve as a backup consideration if the cloudlet is at full capacity, even with potential time penalties.

3. PROPOSED METHOD

This section describes the offloading technique using cloudlet, fingerprinting method, and position prediction method for the IPS.

A. Fingerprinting Method

To predict the position of the mobile device being tracked, it is necessary to obtain RSSI values of the BLE beacons. These obtained RSSI values will be used to make the radio map.

The dataset is collected by obtaining RSSI values at specific coordinates using the Android application developed exactly for RSSI collection purpose. This phase is also referred to as the online phase in the regular fingerprinting method.

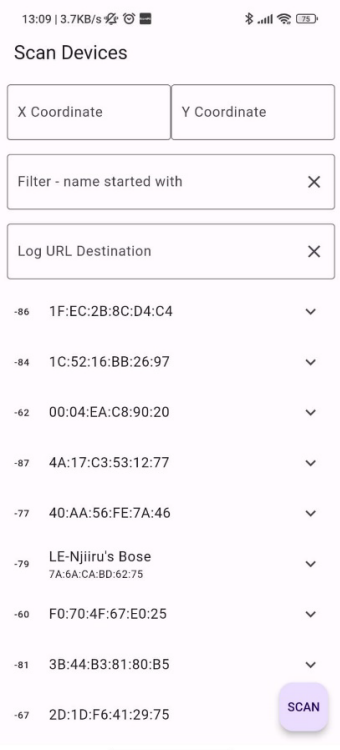


Figure 2 BLE Scanner Application

Data collection is performed at 36 points within the indoor environment using the application in Figure 2, documenting 2000 RSSI readings of 8 BLE signals for every coordinate. Figure 3 contains information about the BLE beacons location and the points where BLE RSSI is captured.

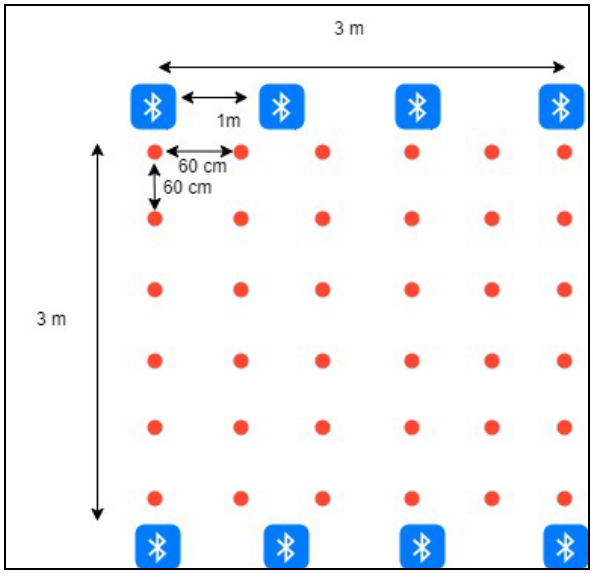


Figure 3 BLE Beacon position and measurement points

The measurement points consists of 8 BLE beacons using ESP32-C3 as beacons and 6 by 6 grid with 60cm space between the measurement points.

While collecting RSSI data, the android application actively filter the received RSSI data so only beacon with advertising name starting with SJ will be scanned. After successfully scan the beacons, the RSSI data is saved on the mobile device and later used for making the radio map.

After the dataset is collected, the next step before training is to evaluate the dataset to clean the data from incomplete or poor-quality entries, such as null values.

B. Position Prediction Model

Before predicting the position of the mobile device (MD), the collected data needs to be prepared. The received RSSI data needs to be cleaned from null values and bad records.

To create a radio map from the collected RSSI data, fingerprinting step is needed. CNN models will be used to do the fingerprinting. Before choosing the suitable CNN model as the model in later experiments, the CNN models needs to be compared.

There are 3 CNN models to compare. They are ResNet50, VGG16 and Custom architecture created in [15]. To compare these models, first the received dataset needs to be split into 3 parts: training data, validation data, and test data. These data then used to train the CNN models.

The model training conducted with implementation of callbacks for each training epoch. The earlystopping callback will be used to prevent overfitting of the model training. The results of the training model then tested using the test dataset prepared on previous step.

After the training is done, the comparison between the models are conducted. The comparison consists of comparing the train accuracy, train validation accuracy and the test accuracy result. These parameters then used in comparing the model. Model with the best accuracy will be used in the offloading experiment.

C. Offloading using Cloudlet

The technique to decide where the offloading will occur will be determined by the comparison of prediction speed on 3 execution places : MD, cloudlet, and cloud. This comparison can be seen on Figure 4

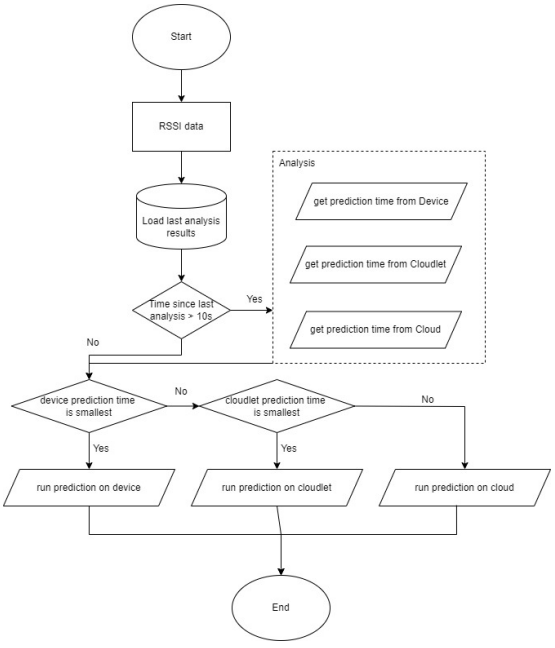


Figure 4 Automatic Offloading execution location selection

The application flow in predicting the location of the target begins with the smartphone receiving all the BLE RSSI and storing the data. Next, prediction will start with comparing when the analysis last taken place. If the analysis takes place after 10 seconds then the app will do the analysis again. This 10 seconds pause is designed to prevent the analysis to take place every time the prediction occur. This will leads to increased prediction time and faster battery drain. The analysis consists of comparing all the time taken to predict in each prediction location. After the comparison finished, the computation then takes place on the location with the shortest prediction time. The prediction diagram can be seen on Figure 5

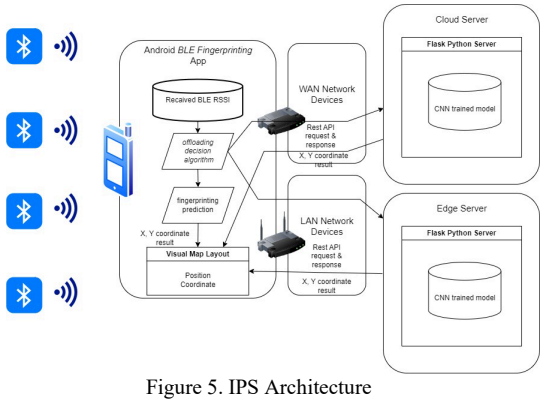


Figure 5. IPS Architecture

The decision decided on which prediction place uses the least prediction time. Then for the next 10 seconds, the offloading will be conducted on selected location. The resource will then perform the prediction

and send back a response containing the prediction class and prediction confidence.

4. EVALUATION

The evaluation conducted can be split into two major parts. The first one is to evaluate the prediction model using the CNN. After the models are trained, every model will be tested using the test dataset. Every model testing results then saved into tables consisting of validation test accuracy and test accuracy.

The first phase of the evaluation focuses on the prediction model developed using CNN. This involves a detailed process starting from the training phase, where multiple CNN models are trained on the training dataset. Each model undergoes training involving numerous epochs and iterations to fine-tune the weights and biases, ensuring the model learns the underlying patterns and features from the dataset.

The results of each model's performance are meticulously recorded in tables, which include detailed metrics for each epoch and the final validation and test accuracy. This comprehensive recording allows for a thorough comparison between different models. By analyzing these results, the model with the highest test accuracy and acceptable validation accuracy is selected for the next phase of evaluation.

After the results of each model have been recorded, performance measurements and comparisons between models can be carried out to determine which model will be used for offloading.

The selected CNN model is then converted into a TensorFlow Lite (TFLite) format. TFLite is designed to be lightweight and optimized for mobile and embedded devices, making it suitable for deployment in various offloading locations such as cloudlets, mobile devices (MD), and the cloud. This conversion is crucial as it ensures the model can run efficiently on different hardware platforms with minimal resource consumption.

With the TFLite model ready, the next phase involves evaluating the offloading techniques. This involves deploying the model to different offloading locations and measuring the performance metrics. The offloading locations considered for this evaluation are cloudlets, mobile devices, and the cloud. Each location presents unique characteristics and challenges in terms of latency, computational power, and energy consumption.

The second evaluation will evaluate the offloading techniques. After the data is uploaded to offloading locations such as cloudlet, MD, and cloud. So experiments on offloading methods can be carried out, The testing will be carried out 4 times according to the location where the offloading is carried out using the Android application that has been created.

The results that will be obtained from this experiment will be the length of prediction time at the location used and the battery percentage every minute for 30 minutes starting with 100% battery condition for each method. The prediction time is measured from the start of the prediction until the prediction produces a coordinate class. In addition, battery measurements will be measured every minute while the application is running.

The first test will be carried out with prediction runs only on MD. In this setup, predictions are made entirely on the mobile device. This setup tests the model's performance when leveraging the device's local computational resources. Key metrics recorded include prediction time and battery consumption over a specified period.

The second test will be carried out with predictions running only on cloudlets. redictions are offloaded to a nearby cloudlet. This setup aims to balance between the low latency of edge computing and the computational capabilities of cloud resources. Metrics recorded include the time taken to offload the data, the prediction time, and the resultant battery consumption.

The third test will be carried out with predictions running only on the cloud. Predictions are offloaded to the cloud, which typically offers substantial computational power but at the cost of higher latency compared to local or edge computing. Metrics include the time taken for data transmission to and from the cloud, the prediction time, and battery usage.

The fourth test will be carried out with running predictions using the location selection method proposed This method employs a dynamic offloading strategy based on certain predefined criteria such as current network conditions, latency, and battery levels. The system dynamically selects the optimal offloading location (MD, cloudlet, or cloud) to balance performance and resource consumption.

The test results then recorded into two tables. Table I captures the battery percentage over time for each offloading method. The battery consumption is monitored every minute for 30 minutes, starting with a fully charged battery (100%). This data provides insights into the energy efficiency of each offloading method.

TABLE I. BATTERY CONSUMPTION OVER TIME

Time elapsed	MD	Cloudlet	Cloud
5 minutes	95%	98%	98%
10 minutes	70%	87%	85%
N minutes	A%	B%	C%

TABLE II. PREDICTION TIME FOR PREDICTING LOCATION

No	Method	Prediction time
1	Cloudlet	θ ms
2	Local (MD)	θ ms
n	…	…

After the battery consumption has been recorded, the received result will be taken from the percentage of battery used over a 30-minute period starting from a full charge, recording the percentage every minute. The resulting data will depict data points of battery percentage. Subsequently, this data will be input into a linear regression to obtain the battery consumption rate for each method. The formula for linear regression is as follows:

$$Y = a + bX \quad (1)$$

The linear regression model in (1) will tell the battery consumption rate for each prediction location. The best battery consumption is measured by the slope of the regression model.

For measuring the duration of the prediction time, measurements are taken from 36 measurement points spaced 60 centimeter apart. The prediction time duration will be measured from the moment the user presses the prediction button and will stop when the user receives the coordinates.

Table II records the prediction time for each method. Prediction time is measured from the start of the prediction process until the prediction produces a coordinate class. This metric is crucial in understanding the latency and responsiveness of the system under different offloading conditions.

The best prediction speed will be chosen from the lowest mean of prediction time on every prediction execution location.

5. RESULTS AND DISCUSSION

This section will shows the results of the model prediction and the offloading evaluation results.

A. Prediction Model Results

To choose what model will be used in the offloading evaluation, there are three CNN model that need to be tested and evaluated. There are ResNet50, VGG16, and Custom architecture. The prediction CNN model architecture can be seen on Figure 6, 7, and 8

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23581440

 flatten_2 (Flatten)         (None, 2048)              0

 dense_2 (Dense)             (None, 36)                73764

=================================================================
Total params: 23,655,204
Trainable params: 23,602,084
Non-trainable params: 53,120
_________________________________________________________________
None
```

Figure 6. Resnet50 model

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 64)        640
 conv2d_1 (Conv2D)           (None, 32, 32, 64)        36928
 max_pooling2d (MaxPooling2D)  (None, 16, 16, 64)      0
 conv2d_2 (Conv2D)           (None, 16, 16, 128)       73856
 conv2d_3 (Conv2D)           (None, 16, 16, 128)       147584
 max_pooling2d_1 (MaxPooling2D)(None, 8, 8, 128)       0
 conv2d_4 (Conv2D)           (None, 8, 8, 256)         295168
 conv2d_5 (Conv2D)           (None, 8, 8, 256)         590080
 conv2d_6 (Conv2D)           (None, 8, 8, 256)         590080
 max_pooling2d_2 (MaxPooling2D)(None, 4, 4, 256)       0
 conv2d_7 (Conv2D)           (None, 4, 4, 512)         1180160
 conv2d_8 (Conv2D)           (None, 4, 4, 512)         2359808
 conv2d_9 (Conv2D)           (None, 4, 4, 512)         2359808
 max_pooling2d_3 (MaxPooling2D)(None, 2, 2, 512)       0
 conv2d_10 (Conv2D)          (None, 2, 2, 512)         2359808
 conv2d_11 (Conv2D)          (None, 2, 2, 512)         2359808
 conv2d_12 (Conv2D)          (None, 2, 2, 512)         2359808
 max_pooling2d_4 (MaxPooling2D)(None, 1, 1, 512)       0
 flatten (Flatten)           (None, 512)               0
 dense (Dense)               (None, 4096)              2101248
 dense_1 (Dense)             (None, 4096)              16781312
 dense_2 (Dense)             (None, 36)                147492
=================================================================
Total params: 33,743,588
Trainable params: 33,743,588
Non-trainable params: 0
```

Figure 7. VGG16 model

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 16, 16, 32)        320
 conv2d_5 (Conv2D)           (None, 16, 16, 64)        18496
 max_pooling2d_3(MaxPooling2D)(None, 8, 8, 64)         0
 conv2d_6 (Conv2D)           (None, 8, 8, 64)          36928
 max_pooling2d_4(MaxPooling2D)(None, 4, 4, 64)         0
 conv2d_7 (Conv2D)           (None, 4, 4, 64)          36928
 max_pooling2d_5(MaxPooling2D)(None, 2, 2, 64)         0
 flatten_1 (Flatten)         (None, 256)               0
 dense_3 (Dense)             (None, 2176)              559232
 dense_4 (Dense)             (None, 1024)              2229248
 dense_5 (Dense)             (None, 36)                36900
=================================================================
Total params: 2,918,052
Trainable params: 2,918,052
Non-trainable params: 0
```

Figure 8.. Custom Model model

After training the models using the acquired dataset, the results then written on Table III

TABLE III. MODEL RESULT COMPARISON

Model	Epoch	Validation Accuracy	Validation Loss	Test Accuracy
Resnet50	86	0,9927	0,0441	0,9877
VGG16	17	0,0268	3,5841	0,0250
Custom	84	0,9800	0,1250	0,9836

From the model training results, model with the best validation accuracy and test are Resnet50 with lowest validation loss. With this result, the model that will be used on offloading testing is ResNet50. The selected model then converted into tflite format and distributed to the prediction execution location.

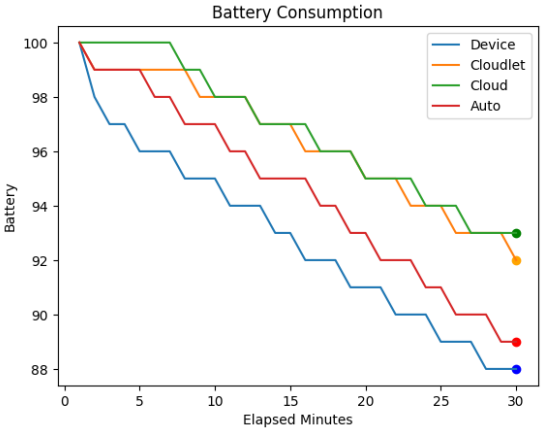
B. Offloading Evaluation Result

Prediction or task were done on mobile device, cloudlet server, cloud server directly and using the proposed offloading technique. The comparison will compare the battery percentage every minute for 30 minutes span.

TABLE IV. BATTERY CONSUMPTION OVER TIME

Time elapsed	MD	Cloudlet	Cloud	Proposed technique
5 min	97%	99%	100%	99%
10 min	95%	98%	98%	97%
15 min	93%	97%	97%	95%
20 min	91%	95%	95%	93%
25 min	89%	94%	94%	91%
30 min	88%	92%	93%	89%

Table IV presents a comparison of the average battery consumption among devices performing the task. Based on the results, the best energy efficiency is reflected on the cloudlet method. This result will be best presented in graph form.



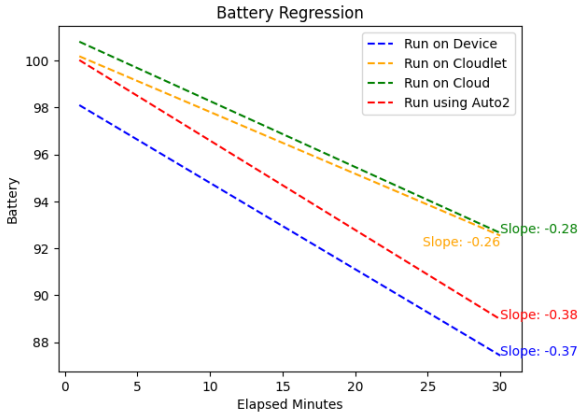


Figure 9. a) Battery Percentage over time b) linear regression on battery percentage

As shown in Figure 9, the highest battery consumption is observed when tasks are performed directly on mobile devices. This significant consumption can be attributed to the intensive computational requirements of running the prediction model locally, which heavily taxes the device's processor and other resources. Over a span of 30 minutes, the battery percentage drops from a full charge of 100% to 88%, indicating a substantial 12% decline. This steep drop highlights the energy demands and potential impracticality of relying solely on mobile devices for such intensive tasks, particularly in scenarios where battery life is crucial, such as in the field or during long-duration use without access to charging facilities.

In contrast, when the task is offloaded to a cloud server, the battery consumption on the mobile device is notably less severe. The offloading process involves transmitting data to the cloud for processing, which, while still consuming energy, is far less demanding than executing the task locally. As a result, the battery drop is not as steep as when the task is performed on the mobile device. This indicates that cloud offloading can be a viable strategy for conserving battery life, especially in applications where preserving device longevity is important. The reduced energy drain also suggests that cloud offloading can enhance the user experience by prolonging device usability before recharging is necessary.

The optimal battery performance is observed when tasks are offloaded to a cloudlet. Cloudlets, which are essentially small-scale cloud servers located closer to the mobile devices, offer a balanced approach by providing computational resources with lower latency compared to distant cloud servers. As illustrated in Figure 9, when tasks are executed on cloudlets, the battery percentage drops from 100% to 92% over the course of 30 minutes. This represents a mere 8% decline, making it the most energy-efficient offloading method among those evaluated. The cloudlet's proximity reduces the energy

cost of data transmission and leverages powerful yet accessible processing capabilities, thus minimizing the overall battery drain.

The rate of battery drain in the cloudlet scenario, calculated at approximately 26% per minute, underscores the efficiency of this method. The relatively minor battery depletion compared to local and cloud offloading demonstrates that cloudlets can effectively balance computational demand and energy consumption. This efficiency makes cloudlets particularly suitable for applications requiring frequent, intensive computational tasks where maintaining battery life is critical. Additionally, the reduced latency associated with cloudlets can enhance the performance and responsiveness of mobile applications, further contributing to a positive user experience.

In summary, the comparative analysis of battery consumption across different offloading methods reveals that while local processing on mobile devices leads to the highest energy drain, offloading to cloud servers and cloudlets significantly mitigates this issue. Cloudlets emerge as the most advantageous option, offering the best balance between conserving battery life and maintaining computational efficiency. This finding underscores the potential of cloudlets to improve the sustainability and user satisfaction of mobile applications that require robust processing capabilities. By strategically leveraging offloading techniques, developers can optimize both performance and energy usage, leading to more efficient and user-friendly mobile solutions.

The second comparison is comparing the prediction time across different execution locations. Prediction or task will be done on mobile device, cloudlet server, cloud server and using the proposed offloading technique. The comparison will compare the duration of the task between task execution locations. Then the prediction time comparison can be seen on Figure 10

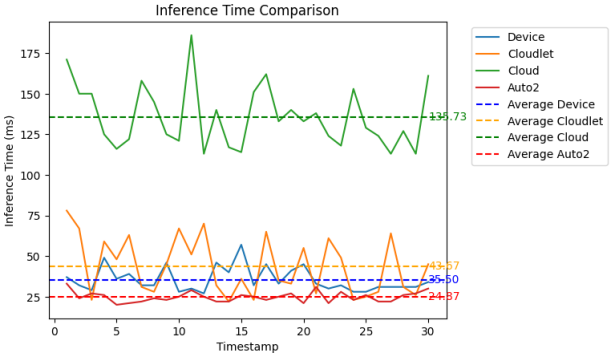


Figure 10 Execution time on all locations

The execution time on mobile device averaging around 35ms with maximum of 57ms and minimum of 27ms. The execution time on cloudlet averaging around 43ms with maximum of 78ms and minimum of 27ms. The execution time on cloud averaging around 135ms with maximum of 186ms and minimum of 27ms. The execution time using proposed technique averaging around 24ms with maximum of 33ms and minimum of 27ms.

The differences of the execution duration between the task execution duration can be seen clearly on the Figure 10. The highest average duration of task execution were on the cloud. The second highest were taken on the cloudlet and the fastest execution time were executed on the using the proposed method.

6. Conclusion and Future Works

In summary, the shortcomings of the current indoor positioning systems (IPS), especially in difficult situations with bad GPS signals, were solved by implementing a novel strategy. For location prediction, a convolutional neural network (CNN)-based fingerprinting method in conjunction with offloading techniques was used. The experimental findings provided important new information about the system's performance, particularly with prediction time and power consumption at different execution sites.

Offloading duties to cloudlet servers resulted in significant improvements in energy efficiency, as demonstrated by reduced power usage when compared to mobile devices and direct cloud server execution. But using the proposed technique the time taken to process the data is significantly decreased. The improvement of the suggested strategy is highlighted by the alignment of these results with the original goal of overcoming location tracking technology limitations, particularly in indoor environments with spotty GPS signals.

In the future, optimization solutions for the execution of mobile devices should be investigated. Possible segmentation of tasks or algorithmic improvements to lower power consumption should be taken into account. Further system optimization depends on exploring dynamic offloading rules, including edge computing technologies, and improving the CNN-based fingerprinting model. In order to guarantee that the created IPS systems satisfy user expectations and preferences, user-centric research should also be carried out to obtain a comprehensive understanding of the trade-offs between power consumption, prediction time, and user experience.

In summary, the study not only demonstrates the advantages of offloading for performance in IPS but also lays the groundwork for future research and development in the area, tackling the issues mentioned in the introduction..

Acknowledgment

The writers thank Computer Science Department, BINUS Graduate Program - Master of Computer Science, Bina Nusantara University for supporting the research.

REFERENCES

[1] K. Szyc, M. Nikodem, and M. Zdunek, "Bluetooth low energy indoor localization for large industrial areas and limited infrastructure," *Ad Hoc Networks*, vol. 139, p. 103024, Feb. 2023, doi: 10.1016/J.ADHOC.2022.103024.

[2] S. Bian, P. Hevesi, L. Christensen, and P. Lukowicz, "Induced magnetic field-based indoor positioning system for underwater environments," *Sensors*, vol. 21, no. 6, pp. 1–25, Mar. 2021, doi: 10.3390/s21062218.

[3] S. M. Asaad and H. S. Maghdid, "A Comprehensive Review of Indoor/Outdoor Localization Solutions in IoT era: Research Challenges and Future Perspectives," *Computer Networks*, vol. 212, p. 109041, Jul. 2022, doi: 10.1016/j.comnet.2022.109041.

[4] T. Jian *et al.*, "Radio Frequency Fingerprinting on the Edge." [Online]. Available: https://github.com/neu-spiral/RFonEdge

[5] K. N. Alinsavath, L. E. Nugroho, Widyawan, and K. Hamamoto, "Indoor location tracking system based on android application using bluetooth low energy beacons for ubiquitous computing environment," *Journal of Communications*, vol. 15, no. 7, pp. 572–576, Jul. 2020, doi: 10.12720/jcm.15.7.572-576.

[6] M. Zamzam, T. Elshabrawy, and M. Ashour, "A minimized latency collaborative computation offloading game under mobile edge computing for indoor localization," *IEEE Access*, vol. 9, pp. 133861–133874, 2021, doi: 10.1109/ACCESS.2021.3115157.

[7] D. Spatharakis *et al.*, "A scalable Edge Computing architecture enabling smart offloading for Location Based Services," *Pervasive Mob Comput*, vol. 67, Sep. 2020, doi: 10.1016/j.pmcj.2020.101217.

[8] Y. Guo, L. Zhao, Y. Wang, Q. Liu, and J. Qiu, "Fog-Enabled WLANs for Indoor Positioning," in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, IEEE, Apr. 2019, pp. 1–5. doi: 10.1109/VTCSpring.2019.8746592.

[9] I. A. Zualkernan and M. Towheed, "Computational Offloading of Convolutional Neural Network on a Smart Watch," in *2020 International Conference on Artificial Intelligence in Information and Communication, ICAIIC 2020*, Institute of Electrical and Electronics Engineers Inc., Feb. 2020, pp. 203–207. doi: 10.1109/ICAIIC48513.2020.9064970.

[10] J. V. Joseph, J. Kwak, and G. Iosifidis, "Dynamic Computation Offloading in Mobile-Edge-Cloud Computing Systems," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, IEEE, Apr. 2019, pp. 1–6. doi: 10.1109/WCNC.2019.8885461.

[11] S. U. R. Malik, H. Akram, S. S. Gill, H. Pervaiz, and H. Malik, "EFFORT: Energy efficient framework for offload communication in mobile cloud computing," in *Software - Practice and Experience*, John Wiley and Sons Ltd, Sep. 2021, pp. 1896–1909. doi: 10.1002/spe.2850.

[12] S. Misra, B. E. Wolfinger, M. P. A. Achuthananda, T. Chakraborty, S. N. Das, and S. Das, "Auction-Based Optimal Task Offloading in Mobile Cloud Computing," *IEEE Syst J*, vol. 13, no. 3, pp. 2978–2985, Sep. 2019, doi: 10.1109/JSYST.2019.2898903.

[13] T. Huang, F. Ruan, S. Xue, L. Qi, and Y. Duan, "Computation offloading for multimedia workflows with deadline constraints in cloudlet-based mobile cloud," *Wireless Networks*, vol. 26, no. 8, pp. 5535–5549, Nov. 2020, doi: 10.1007/s11276-019-02053-z.

[14] S. Erana Veerappa Dinesh and K. Valarmathi, "A novel energy estimation model for constraint based task offloading in mobile cloud computing," *J Ambient Intell Humaniz Comput*, vol. 11, no. 11, pp. 5477–5486, Nov. 2020, doi: 10.1007/s12652-020-01903-5.

[15] Sinha and Hwang, "Comparison of CNN Applications for RSSI-based Fingerprint Indoor Localization," *Electronics (Basel)*, vol. 8, no. 9, p. 989, Sep. 2019, doi: 10.3390/electronics8090989.

Sinjiru Setyawan born in Indonesia on year 2000. The writer finished bachelor degree on Bina Nusantara University Indonesia on year 2022 and now studying master of computer science on Bina Nusantara University Indonesia

Gede Putra Kusuma received PhD degree in Electrical and Electronic Engineering from Nanyang Technological University (NTU), Singapore, in 2013. He is currently working as a Lecturer and Head of Department of Master of Computer Science, Bina Nusantara University, Indonesia. Before joining Bina Nusantara University, he was

working as a Research Scientist in I2R – A*STAR, Singapore. His research interests include computer vision, deep learning, face recognition, appearance-based object recognition, gamification of learning, and indoor