DOI: 10.22146/jnteti.v14i2.18985

Comparative Analysis of MVVM and MVP Patterns Performance on Android Dashboard System

Fajar Pradana[1], Raziqa Izza Langundi[1], Djoko Pramono[1], Nur Ida Iriani[2]

[1] Information System Department, Faculty of Computer Science, Universitas Brawijaya, Malang, Jawa Timur 65145, Indonesia
[2] Management Program Study, Faculty of Economics, Universitas Tribhuwana Tunggadewi, Malang, Jawa Timur 65144, Indonesia

[Received: 23 January 2025, Revised: 7 March 2025, Accepted: 16 April 2025]
Corresponding Author: Fajar Pradana (email: fajar.p@ub.ac.id)

ABSTRACT — The rapid growth of the Android market in various developing countries has driven the demand for higher-quality applications. Developing Android-based applications presents specific challenges, such as the need for responsive designs and optimization for devices with diverse specifications. Design patterns like model-view-controller (MVC), model-view-presenter (MVP), and model-view-viewmodel (MVVM) have become popular approaches to address these issues. However, studies on the performance of design patterns in Android applications, especially in modern programming languages like Kotlin, remain limited. This research aims to compare the performance of the MVP and MVVM design patterns in an Android-based boarding house management application, KosGX. This application utilized Kotlin and featured an interactive dashboard requiring significant device resources. Testing was conducted by measuring performance across three key aspects: central processing unit (CPU) usage, memory usage, and system response time. The results of the study showed that MVVM outperformed in CPU efficiency, with an average usage of 8.92% compared to 11.15% for MVP. In terms of memory usage, MVVM was also slightly more efficient, with an average usage of 121.48 MB compared to 121.55 MB for MVP. However, MVP excelled in response time, averaging 236.88 ms, whereas MVVM reached 252.68 ms. This study underscores that the choice of design pattern affects application performance. MVVM is more efficient in CPU and memory usage, while MVP offers better response times. These findings provide valuable insights for developers in selecting the optimal design pattern based on the specific needs of their applications.

KEYWORDS — Kotlin, CPU Efficiency, Response Time, MVP, MVVM, Memory Usage, Android Profiling.

I. INTRODUCTION

The Android market is currently experiencing positive growth [1], driven by increasing demand in developing markets and technological innovations such as the integration of artificial intelligence (AI) in mobile devices [2]. According to a report by the International Data Corporation (IDC), global smartphone shipments were projected to grow by 6.2% year-over-year (YoY) in 2024, reaching 1.24 billion units. Android's rapid growth of 7.6% YoY is particularly evident in the Asia-Pacific region (excluding Japan), Latin America, the Middle East, Africa, and China, primarily in the lower-tier device segment. Conversely, iOS was only expected to grow by 0.4% in 2024.

The Android platform is utilized not only by mobile device users but also by software developers and manufacturers across various types of devices, including smart TVs, tablets, wearables, or automobiles [3]. Reflecting this diversity, the official Android app store hosts over 2 million applications across 60 different categories. These applications range from education [4], banking [5], games, medical [6], travelling, and health monitoring [7].

Android application development differs significantly from desktop or web development. Applications must be optimized for a wide variety of devices and specifications compared to desktop or web, which typically involve more stable environments with greater resources [8]. Additionally, Android applications require responsive designs to accommodate varying screen sizes and resolutions, unlike the relatively uniform resolutions and screen sizes in desktop and web development [9], [10]. While desktop and web application development have been established for a longer time, Android application development is a relatively newer field, with many developers still in the beginner phase [11]. Novice developers often face challenges that can impact application quality, such as inadequate architectural planning, excessive memory usage, poor feature implementation, significant bugs, and difficulty in making improvements. Clear guidance in the form of software design pattern implementation is therefore necessary to produce higher-quality software that is reusable, maintainable, and easier to evolve [12], [13].

Design patterns offer solutions to common problems in software development [14]. By applying design patterns, developers can accelerate their workflow, improve code quality, and create systems that are easier to maintain. Currently, the most widely used design patterns on the Android platform are model-view-controller (MVC), model-view-presenter (MVP), and model-view-viewmodel (MVVM) [15]. MVC is the most frequently adopted design pattern due to its simplicity in the development process, but it has a notable drawback in the form of tight coupling between the controller and the view. MVP and MVVM, on the other hand, provide different approaches to managing data and interactions between application components. In MVP, the presenter acts as the intermediary between the view and the model, enabling modifications to both components. In contrast, MVVM's viewmodel provides a stream of data that the view can consume, eliminating the need for the view to update data as in MVP [16]. By separating these components, these patterns reduce potential issues and enhance the testability of the application. Although MVP and MVVM are claimed to be superior to MVC, limited research has examined the performance differences between these two patterns. Previous studies on mobile device performance

regarding design pattern usage in Android applications have reported that MVVM offers better CPU utilization and faster response times compared to MVP [17]. However, MVP performs better in memory management. These tests were conducted on a point of sale (PoS) application developed in Java, leaving a research gap for performance evaluations in more modern programming languages.

This study introduced a new approach by evaluating the performance of an application named KosGX, built using Kotlin, through a comparative analysis of the MVP and MVVM design patterns. Unlike previous studies that focused solely on Java programming language with limited use cases, this study leveraged Kotlin—a modern programming language officially supported by Android—to address the lack of research evaluating performance in newer programming environments. KosGX, a boarding house management application with complex interactive elements, provides a comprehensive testing platform to assess design pattern performance in real-world scenarios. The application serves as a general example of a dashboard-based system, presenting data akin to typical dashboard systems. Memory usage reflects the amount of RAM allocated to the application, making it critical for low-end devices where excessive RAM allocation can degrade performance [18]. System response time (SRT) is a key factor influencing user satisfaction [19].

This study offers three main contributions: empirical evidence, practical implications, and actionable insights. Empirical evidence provides empirical evidence on the performance trade-offs between MVP and MVVM in Kotlin-based Android applications, addressing gaps in previous research on contemporary software development. Meanwhile, practical implications highlight the significance of choosing design patterns for resource-constrained software, ensuring more optimal memory usage and shorter response times. Actionable insights offer developers to improve application quality and user experience through informed architectural decisions.

From a societal perspective, these findings benefit both developers and end-users. For developers, the study equips them with deeper insights into design pattern performance, enabling them to create more efficient and maintainable applications. For end-users, especially those in developing regions who rely on low-end Android devices, optimized applications contribute to smoother user experiences and longer device lifespans. Furthermore, as digital solutions continue to address critical societal needs such as education, healthcare, and financial inclusion, developing high-performance applications becomes a cornerstone for technology-driven progress.

This research aims to identify the design pattern that offered better data presentation and optimal resource utilization for the KosGX application. Significant performance differences indicate that the choice of design pattern is an essential factor in performance testing. Design pattern selection impacts device performance when running applications, making it essential for developers to choose the most suitable pattern to ensure a positive user experience. Application performance plays a vital role in user satisfaction, and poor performance can adversely affect users' perceptions of the application [20]. Consequently, developers must consider system performance as an integral aspect of user experience design.

II. RELATED WORKS

A. DESIGN PATTERN

A design pattern is a solution to common problems encountered during system development, particularly those related to design, code organization, and system efficiency [14]. In mobile applications, system development also requires design patterns to ensure architectural structures are more organized and that the function and purpose of every line of code written by the developer can be easily identified. Officially, Android recommends that applications consist of two layers: the presentation layer and the data layer, with an additional layer acting as an intermediary to facilitate interaction between these two layers [15]. Design patterns are essential in Android system development to create efficient, maintainable applications with high scalability [21]. Various types of design patterns can be applied in Android development, each with its unique characteristics, advantages, and disadvantages. For instance, applying the flyweight design pattern in Android has proven to enhance awareness of memory consumption during mobile application development [22]. One study compared the flyweight pattern with traditional object-oriented programming, showing that the flyweight pattern does not negatively impact memory usage, enabling professional software design without sacrificing efficiency. Another study discussed the importance of improving software quality and reusability in Android systems [23]. As a result, a paper proposed PatRoid, a framework for automatically detecting the presence of design patterns in source code. Preliminary evaluations demonstrated that PatRoid successfully detected 23 gang of four (GoF) design patterns in Android applications.

B. MODEL VIEW VIEWMODEL (MVVM) dan MODEL VIEW PRESENTER (MVP)

The MVVM and MVP architectural patterns are widely used in software development, particularly in Android applications. Both patterns aim to decouple concerns, improving maintainability and testability, but they differ in their implementation approaches and data handling. MVVM is a variation of the MVC architecture designed to achieve a complete separation between the model and the view components [24]. The model is a class containing data, the view represents the application's user interface (UI) and is responsible for displaying information, and the viewmodel handles the application's business logic, providing data streams to the view component without being directly tied to it. In other words, the viewmodel has no knowledge of the existence of a View.

In contrast, MVP is an architecture similar to MVC but with some differences. The workflow begins with the view capturing user input, which is then passed to the presenter [25]. The model contains the data to be displayed, the view represents the application's interface, and the presenter manages all interactions between the model and the view [26]. The presenter retrieves data from the model and delivers it to the view. MVP leverages interface classes, which are empty function definitions that can be extended by other classes. These interfaces are used in both the presenter and view components to implement functions as required, whether to retrieve or send data [27]. Figure 1 illustrates the workflow of the MVP and MVVM architectures.

MVVM excels in terms of modifiability, featuring the lowest modification index, while MVP performs better in terms of raw performance. Both architectures are part of the clean

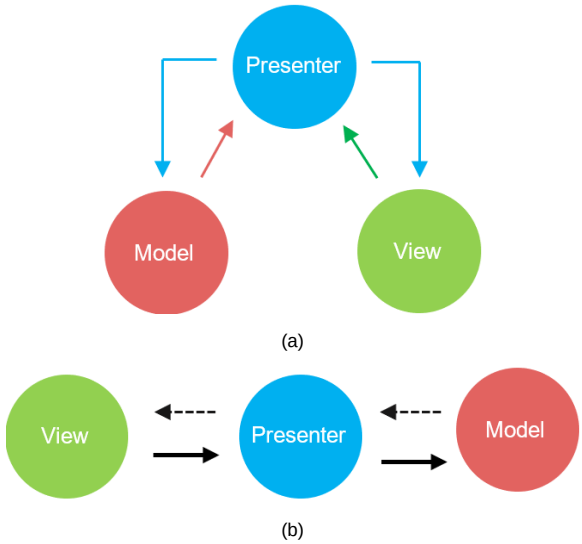

Figure 1. Illustration of how pattern architecture works, (a) MVP and (b) MVVM.

architecture paradigm, which enhances Android application development by ensuring minimal dependencies and improved usability [26].

A study has reported that MVVM is the best in modifiability, boasting the lowest modification index, while MVP outperforms in performance and model-view-intent (MVI) excels in test coverage [28]. Due to the limited research exploring the comparative analysis of MVP and MVVM in native Android platforms, particularly in systems developed using Kotlin, this study aimed to address the existing gap.

C. SYSTEM PERFORMANCE

Modern smartphone users are no longer limited to preinstalled applications provided by device manufacturers. They can also access third-party applications downloaded through various app distribution platforms [21]. In this ecosystem, the use of third-party applications often leads to significant memory consumption, which can result in insufficient available memory to run these applications optimally. This issue is particularly common in lower-tier mobile devices with limited memory capacity, making memory shortages a more frequent problem [29].

Application performance reflects how quickly an application runs, how fast it loads data, and its overall connectivity with various operations. Several key aspects need to be considered to evaluate the performance of Android applications. These include the central processing unit (CPU) usage, execution time, and memory consumption, which are the primary metrics for measuring application performance [30]. The demand for high-performance applications on the Android platform continues to grow with advances in technology and rising user expectations. Thus, this study aimed to compare design patterns to identify the one with the best performance on the Android platform.

In this research, Android Profiler was utilized to measure performance. During the testing process, the device was connected to Android Profiler to record CPU and memory usage while opening the dashboard. For response time, logs from Android Studio were analyzed to determine the time taken by the application to display data.

III. METHODOLOGY

This study employed a methodology inspired by the research and development (R&D) methodology, which is commonly used to develop new products or systems through research, design, development, and evaluation processes [31]. Research methodology was designed to support a comparative performance analysis of the MVVM and MVP design patterns in an Android dashboard system. The study began with the software development phase, where an application implementing both MVVM and MVP design patterns was developed using the Kotlin programming language. Once the application was completed, the designing test case phase was conducted to create testing scenarios covering various performance parameters, such as CPU usage, memory usage, and response time. The subsequent phase, experiment, involved executing the test scenarios on the application to collect performance data. The data obtained from the experiments were gathered during the data collection phase to ensure their completeness and validity. Subsequently, the data were analyzed during the data analysis phase using relevant statistical methods, such as the independent sample *t*-test, in order to identify significant differences between the two design patterns.

A. SOFTWARE DEVELOPMENT

The first step in this research was the development of a dashboard-based application named KosGX, built using Kotlin, a modern programming language officially supported by Google for Android development. KosGX is a boarding house management application with a dashboard feature that includes various visual components such as text, images, graphs, and interactive elements requiring significant CPU and memory resources.

The application was developed using MVP and MVVM design patterns. The application interfaces were made identical, displaying data such as payment visual diagrams, the number of residents, rooms, a list of residents, a list of needs for the boarding house, and a summary of payment income. The data were stored in a local database using Room, a library built on SQLite for local data storage on Android. Figure 2 illustrates the UI of the KosGX application, which serves as the experimental object in the performance comparison study of the MVVM and MVP design patterns on Android applications. The application is designed to assist in managing boarding houses with key features such as monitoring payment status, tracking room availability, revenue management, and a to-do list.

On the left side of the interface, the main dashboard displays a summary of important data, including payment status represented as a pie chart, room availability with total capacity, total revenue, and a high-priority task list for facility maintenance. Meanwhile, on the right side, a detailed view of room occupants is shown, featuring information such as occupant names, professions, payment amounts, and payment statuses (paid or unpaid), marked with visual icons.

This application was developed to support data collection related to application performance in real-world usage scenarios. The UI components are designed to accommodate the needs of boarding house managers, focusing on efficiency, clear information presentation, and simple interactions. In the experiment, this application is used to compare CPU performance, memory usage, and response time between the implementations of the MVVM and MVP design patterns.

B. DESIGNING TEST CASE

The test scenarios in this research focus on the application's performance, which includes evaluating responsiveness,

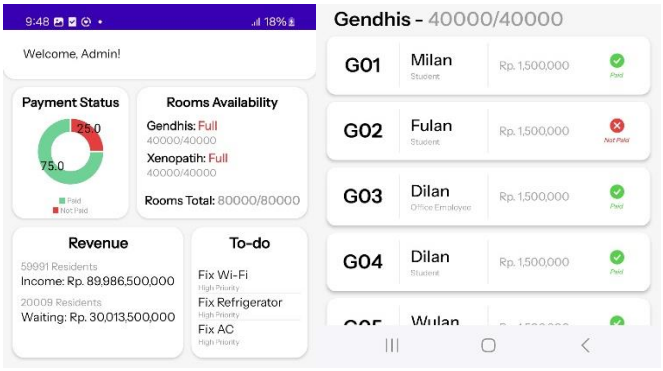

Figure 2. Example of KosGX application UI.

scalability, stability, and resource usage [32]. Dummy data, or synthetic data, are displayed on the application's dashboard for testing. Each test case is identified using a specific code format: "TC-X-Y-Z," where X represents the design pattern, Y represents the test type, and Z indicates the test case number. Data for each test case consists of 10 data points. Tables I and II show the test cases for each design pattern.

C. EXPERIMENT

The testing involves running the application populated with dummy data according to the designated test cases. During testing, the device is connected to Android Profiler to record CPU and memory usage while opening the dashboard. For response time, logs from Android Studio are used to measure the time the application takes to display data. Before testing begins, it is ensured that no other applications are running on the device. The testing was conducted on an Android device, specifically a Samsung Galaxy A52 with a Snapdragon 778G processor, 8GB of RAM, and Android 14 operating system.

D. DATA COLLECTION

CPU and memory performance were recorded using Android Profiler, while dashboard response times were collected using Android activity's built-in methods. Data collection occurs exclusively during the dashboard activity. When the dashboard was opened, CPU and memory usage appeared in the Android Profiler. CPU usage data were summed and averaged, while memory usage was measured at the start, peak, and end of the activity, then averaged.

E. DATA ANALYSIS

Quantitative data were analyzed using parametric or nonparametric statistical techniques, depending on the characteristics of the data. Parametric statistics are typically used when the data meet certain assumptions, including normal distribution, linearity, and homogeneity of variance. These assumptions are crucial as they ensure the reliability and validity of tests such as *t*-tests or ANOVA, which rely on precise mathematical models to evaluate differences or relationships between variables. For instance, the normality of data distribution is often assessed using tools such as the Shapiro-Wilk test, while homogeneity of variance can be evaluated using Levene's test.

If the data fail to meet these assumptions, nonparametric statistical techniques are used as an alternative. Methods such as the Mann-Whitney U test or Kruskal-Wallis test do not rely on strict assumptions about the underlying data distribution, making them more robust for analyzing skewed or nonlinear data. While nonparametric tests are less sensitive to outliers and

TABLE I
TEST CASE MVVM

Data Range	CPU	Memory	Response Time
10,000– 19,000	TC-VM-CPU-01	TC-VM-MEM-01	TC-VM-RT-01
20,000– 29,000	TC-VM-CPU-02	TC-VM-MEM-02	TC-VM-RT-02
30,000– 39,000	TC-VM-CPU-03	TC-VM-MEM-03	TC-VM-RT-03
40,000– 49,000	TC-VM-CPU-04	TC-VM-MEM-04	TC-VM-RT-04

TABLE II
TEST CASE MVP

Data Range	CPU	Memory	Response Time
10,000– 19,000	TC-P-CPU-01	TC-P-MEM-01	TC-P-RT-01
20,000– 29,000	TC-P-CPU-02	TC-P-MEM-02	TC-P-RT-02
30,00 – 39,000	TC-P-CPU-03	TC-P-MEM-03	TC-P-RT-03
40,000– 49,000	TC-P-CPU-04	TC-P-MEM-04	TC-P-RT-04

irregularities, they may lack the statistical power of parametric tests, meaning that detecting significant effects might require larger sample sizes or more pronounced differences.

In this study, hypotheses were formulated to examine whether there are significant differences in application performance metrics—CPU usage, memory usage, and response time—between the MVVM and MVP design patterns. These hypotheses were tested using appropriate statistical techniques based on the processed research data. For example, an independent sample *t*-test was applied if the data met parametric criteria, allowing for accurate comparisons of mean differences between the two design patterns. Conversely, if assumptions were violated, the Mann-Whitney U test was used to compare the distributions of performance metrics without relying on normality.

The completion of the data analysis phase provided critical insights into the performance characteristics of each design pattern. The results determined whether observed differences in CPU usage, memory consumption, or response time were statistically significant or merely due to random variation. These findings not only validated the hypotheses but also offered actionable conclusions regarding the suitability of MVVM and MVP for various application scenarios, contributing to a deeper understanding of their performance trade-offs in Android development.

By employing a rigorous and adaptive statistical approach, this study ensured that the analysis was scientifically robust and capable of adapting to the nuances of the data, providing reliable evidence to support the conclusions drawn. This meticulous methodology reinforced the validity of the research findings and emphasized the importance of selecting appropriate statistical techniques tailored to the characteristics of the data.

F. RESULTS AND DISCUSSION

After statistical analysis is completed, the results will yield conclusions about whether there is a significant difference between the MVVM and MVP design patterns in displaying

data on an Android application's dashboard. The following criteria must be satisfied to determine the optimal design pattern.

1. The design pattern is considered better if it has a lower average CPU usage.
2. The design pattern is considered better if it has a lower average memory usage.
3. The design pattern is considered better if it demonstrates faster response times.

IV. RESULTS AND DISCUSSION

A. RESULTS

The testing was conducted on an Android device, specifically a Samsung Galaxy A52 with a Snapdragon 778G processor, 8GB of RAM, and Android 14 operating system. The tests were divided into four test cases (TC) based on the range of data used in each test. For TC-1, which used a data range of 10,000–19,000 entries in the application's database, the average CPU usage was 7.24% for MVVM and 9.54% for MVP. In TC-2 (20,000–29,000 data), average CPU

usage was 8.49% for MVVM and 10.90% for MVP. In TC-3 (30,000–39,000 data), it was 9.37% for MVVM and 11.76% for MVP. Finally, in TC-4 (40,000–49,000 data), CPU usage averaged 10.57% for MVVM and 12.39% for MVP. Figure 3 illustrates the rising trend in CPU usage for each design pattern, with MVVM represented by a blue line and MVP by a red line. In the CPU category, MVVM consistently outperformed MVP by using less CPU in all test cases.

The distribution of CPU usage data can be visualized using a boxplot in Figure 4. The lowest CPU usage is represented by the bottommost point, at 6% for MVVM and 7.43% for MVP. The highest CPU usage is represented by the topmost point, at 11.86% for MVVM and 14% for MVP. The box itself represents the interquartile range (IQR), which is the difference between the first and third quartiles, illustrating the variability of the data around the median. The median CPU usage is marked by the line inside the box, at 9% for MVVM and 11% for MVP. Visually, MVVM appears to be more efficient in CPU usage compared to MVP.

In TC-1, with a data range of 10,000–19,000, the average memory usage was 113.03 MB for MVVM and 113.35 MB for MVP. In TC-2, with a range of 20,000–29,000 data, the average memory usage was 119.84 MB for MVVM and 120.39 MB for MVP. For TC-3, with 30,000–39,000 data, the average memory usage was 124.10 MB for MVVM and 123.37 MB for MVP. Finally, in TC-4, with 40,000–49,000 data, the average memory usage was 128.96 MB for MVVM and 129.10 MB for MVP. The visualization of the increasing average memory usage is shown in Figure 5, with MVVM represented by a blue line and MVP by a red line. In the memory performance category, MVVM and MVP were closely matched, with MVP performing better in TC-1, TC-2, and TC-4, while MVVM outperformed in TC-3.

The distribution of memory usage data can be visualized using a boxplot, as shown in Figure 6, with MVVM represented by a blue color and MVP by a red color. The bottommost point represents the lowest memory usage at 107.37 MB for MVVM and 108.63 MB for MVP. The highest memory usage is represented by the topmost point at 137.83 MB for MVVM and 133.97 MB for MVP. The median memory usage is indicated by the line within the box, at 122.70 MB for MVVM and 122.23 MB for MVP. Visually, the MVVM design pattern

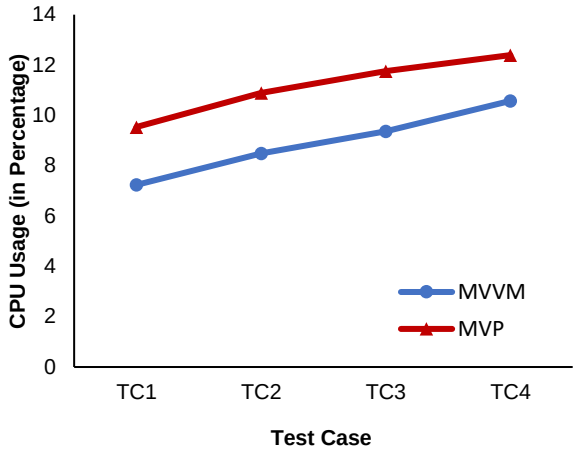

Figure 3. CPU performance comparison.

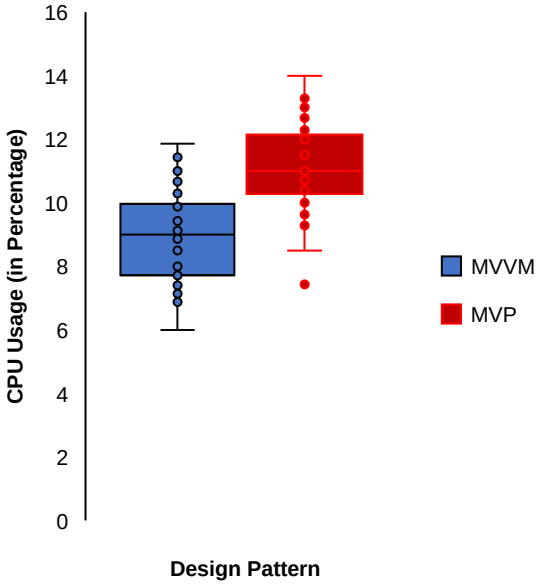

Figure 4. Boxplot diagram of CPU usage.

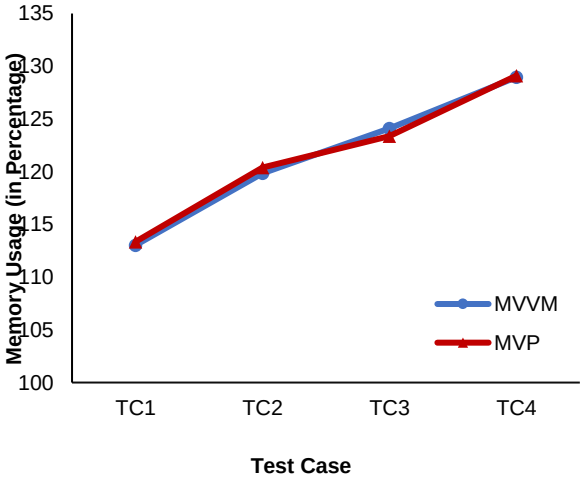

Figure 5. Memory performance comparison.

exhibits a wider data spread and the lowest memory usage, while the MVP design pattern has a narrower data spread.

In TC-1, with a data range of 10,000–19,000, the average response time was 144.79 ms for MVVM and 134.19 ms for MVP. In TC-2, with a data range of 20,000–29,000, the average

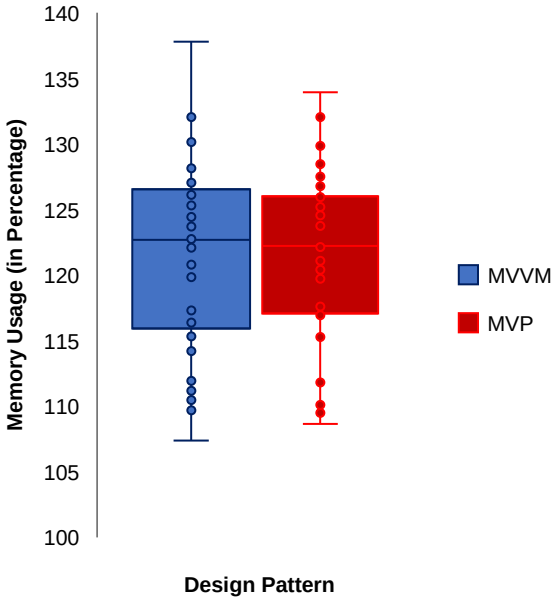

Figure 6. Boxplot diagram of the memory usage.

response time was 217.16 ms for MVVM and 202.28 ms for MVP. In TC-3, with a data range of 30,000–39,000, the average response time was 285.17 ms for MVVM and 270.49 ms for MVP. Finally, in TC-4, with a data range of 40,000–49,000, the average response time was 363.61 ms for MVVM and 340.57 ms for MVP. The visualization of the increasing average response time for each design pattern is shown in Figure 7, with MVVM represented by a blue line and MVP by a red line. MVVM outperformed in TC-1, while MVP performed better in TC-2, TC-3, and TC-4, with a relatively close gap between the two design patterns. Graphically, MVP is faster at displaying data compared to MVVM, though the difference in speed between the two design patterns is quite narrow.

The distribution of response time data can be visualized using a boxplot, as shown in Figure 8, with MVVM represented in blue and MVP in red. The lowest response time is at the bottommost point, 116.30 ms for MVVM and 107.70 ms for MVP. The highest response time is at the topmost point, 407.50 ms for MVVM and 390.80 ms for MVP. The median response time is indicated by the line within the box, at 256.50 ms for MVVM and 231.65 ms for MVP. Visually, both design patterns have boxes of the same size, but MVP is slightly more optimal than MVVM.

Once the experimental data were collected, the next step was to analyze it. The first analysis involved testing the normality of the data, which determines the type of difference test to use. If the data are normally distributed, a parametric test is applied. Otherwise, a nonparametric test is used.

For CPU usage with MVVM, the *p*-value was 0.986, which is greater than 0.05, indicating a normal distribution. Similarly, CPU usage for MVP had a *p*-value of 0.8933, which is also greater than 0.05, confirming normal distribution. For the memory usage, MVVM had a *p*-value of 0.674, while and MVP had a *p*-value of 0.1883. These results indicate normal distribution. For response time, MVVM and MVP had *p*-values of 0.1768 and 0.1809, respectively, both greater than 0.05, confirming normality.

Thus, all data categories were found to be normally distributed, allowing for parametric testing. Table III summarizes the Shapiro-Wilk normality test results for each performance category.

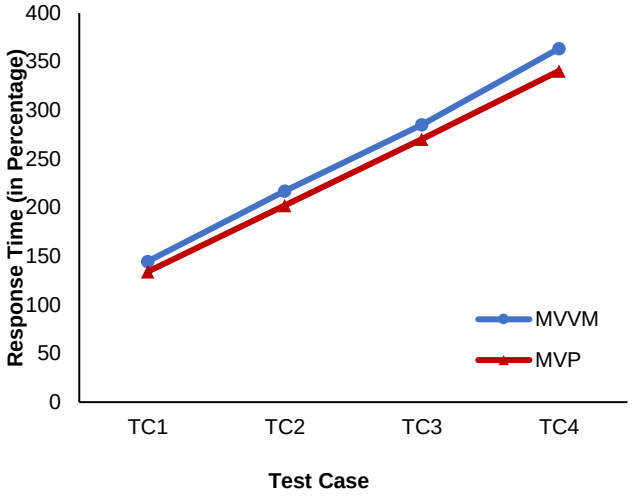

Figure 7. Response time comparison.

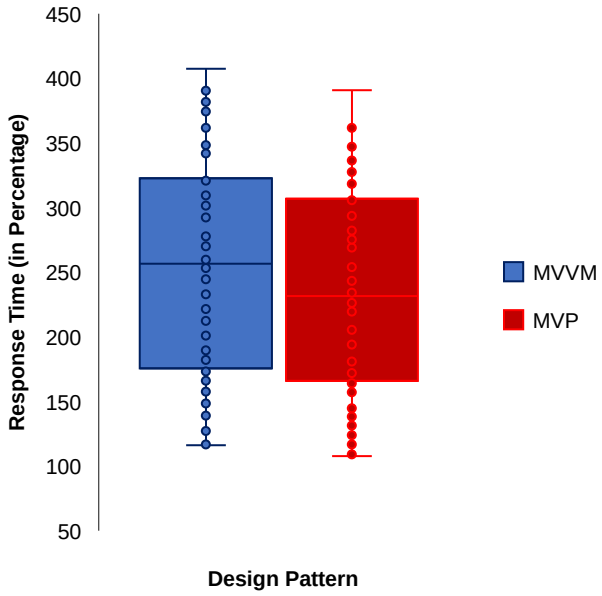

Figure 8. Boxplot diagram response time.

TABLE III
NORMALITY TESTING

Design Pattern	CPU	Memory	Response Time
MVVM	0.986	0.674	0.1768
MVP	0.8933	0.1883	0.1809

Homogeneity testing was conducted to determine whether the variance between groups is equal or homogeneous, which is required for an independent *t*-test. For CPU, the *p*-value was 0.5931, greater than 0.05, indicating homogeneous variance. For memory, the *p*-value was 0.9689, also greater than 0.05, confirming homogeneity. For response time, the *p*-value was 0.756, greater than 0.05, indicating homogeneous variance. Since all data categories had homogeneous variances, an independent *t*-test was conducted.

With the prerequisites fulfilled, the final testing was conducted to determine whether there was a significant difference between the MVVM and MVP design patterns in

each performance category. The hypotheses for the independent *t*-test were as follows:

1. null hypothesis (H0): If $p > 0.05$, no significant difference exists;
2. alternative hypothesis (H1): If $p \leq 0.05$, a significant difference exists.

For CPU usage, the *p*-value was $1.866e^{-10}$ (less than 0.05), indicating a significant difference between the design patterns. For memory usage, the *p*-value was 0.9608, greater than 0.05, indicating no significant difference. For response time, the *p*-value was 0.756, also greater than 0.05, indicating no significant difference. Thus, a significant difference was observed in CPU usage between the design patterns, while no significant difference was found for memory usage and response time. Additionally, the analysis results indicate that in terms of CPU usage, MVVM is more efficient than MVP, with Cohen's d = -1.64, representing a very large and significant effect. Conversely, in response time, MVP demonstrates better performance than MVVM, with Cohen's d = 1.07, which is also a large effect. Meanwhile, in memory usage, the difference between the two design patterns is not significant (Cohen's d = -0.03) with a high *p*-value, indicating that memory usage between MVVM and MVP is relatively the same. Regarding potential biases, the results of the normality test (Shapiro-Wilk test) and homogeneity of variance test (Levene's test) confirm that the statistical assumptions are met, ensuring that the *t*-test results are reliable and free from significant bias.

B. DISCUSSION

Among the three performance categories—CPU, memory, and response time—only CPU usage showed a significant difference between the MVVM and MVP design patterns. MVVM was superior in CPU usage, with an average usage of 8.02%, compared to 11.15% for MVP. For memory usage, MVVM slightly outperformed MVP with an average of 121.48 MB versus 121.55 MB. However, MVP excelled in response time, with an average of 236.88 ms compared to 252.68 ms for MVVM. This finding differs from previous research [17] which reported that MVVM was superior in both CPU usage and response time, while MVP excelled in memory usage. This discrepancy could be attributed to differences in testing environments, application complexity, or the specific use of Kotlin in this study, which may influence performance metrics. Additionally, variations in how the UI rendering pipeline is managed between different versions of Android could also contribute to these differences, as certain optimizations in UI thread execution may favor one architecture over the other.

Based on the results of the independent sample *t*-test, only the CPU category showed a significant difference in application performance between the design patterns for Android-based applications developed using Kotlin. This difference is assumed to arise from the distinct ways each design pattern manages data flow and presents data to the user. The key distinction lies in the intermediary component used to handle requests to the model and deliver data to the view: the viewmodel in MVVM and the presenter in MVP. The viewmodel facilitates a reactive data-binding mechanism that reduces the overhead of frequent updates, thereby optimizing CPU usage. This finding is consistent with [16], which highlighted the efficiency of reactive programming paradigms in reducing CPU load. Furthermore, since MVVM leverages LiveData and flows in Kotlin, it offloads computation-heavy operations to background threads more efficiently than MVP, where the Presenter actively controls UI updates and might keep unnecessary UI-bound computations within the main thread.

Conversely, MVP demonstrated superior response times, likely due to its simpler data flow architecture, where the presenter acts as a direct channel between the model and the view. This aligns with [28], which observed that minimizing the number of intermediary layers in the data flow can result in faster response times. However, this advantage comes at the cost of higher CPU usage, as the presenter requires more frequent interactions with the View and Model components, especially in complex applications.

Interestingly, MVVM's slight advantage in memory usage compared to MVP contradicts previous findings [17], where MVP was reported to excel in this category. A possible explanation is MVVM's more efficient handling of lifecycle-aware components, which reduces the likelihood of memory leaks—a common issue in MVP when managing long-lived Presenters. This aligns with [20], which emphasizes the importance of lifecycle-aware components in effectively managing memory consumption.

Overall, these findings underscore the nuanced trade-offs between MVVM and MVP, particularly when applied to Android development using Kotlin. While MVVM demonstrates superior CPU efficiency and marginally better memory usage, MVP offers a more responsive user experience. The choice between the two should consider the application's performance priorities and complexity, as well as the development team's familiarity with each design pattern. Future research could further explore these trade-offs by including more complex scenarios or integrating additional performance metrics, such as energy consumption or maintainability.

This study provides valuable insights into the comparative performance of the MVVM and MVP design patterns in Android applications developed with Kotlin. One of the main advantages of this research is its empirical evaluation of real-world application scenarios, ensuring that the findings are relevant and applicable to modern Android development. Additionally, the use of statistical analysis strengthens the validity of the conclusions, offering developers concrete data to support design pattern selection based on specific performance needs. However, the study also has limitations. The testing was conducted on a single device model, which may not fully capture variations in hardware performance across different Android devices. Furthermore, the study primarily focuses on CPU usage, memory consumption, and response time, while other important factors such as energy efficiency, maintainability, and scalability were not explored in depth. Future research could address these limitations by expanding the scope to include a broader range of devices and additional performance metrics to provide a more comprehensive evaluation of these design patterns.

V. CONCLUSION

This study compared the performance of MVVM and MVP design patterns in an Android application built with Kotlin, focusing on CPU usage, memory consumption, and response time. The results indicate that MVVM outperforms MVP in CPU efficiency, with an average usage of 8.92% compared to 11.15%. MVVM also demonstrated slightly better memory efficiency (121.48 MB vs. 121.55 MB), though the difference was not statistically significant. In contrast, MVP exhibited

faster response times, averaging 236.88 ms compared to 252.68 ms for MVVM.

These findings suggest that the choice of design pattern should be based on the application's performance priorities. MVVM is more suitable for applications requiring optimized CPU and memory management, particularly those with complex data-binding scenarios or resource constraints. MVP, on the other hand, is preferable for applications demanding real-time responsiveness with minimal latency.

Future research could expand on these findings by incorporating additional metrics such as energy consumption, maintainability, and scalability. Furthermore, testing on a wider range of devices and real-world scenarios could provide deeper insights into the performance of these design patterns under various conditions, ensuring broader applicability of the conclusions drawn from this study.

CONFLICTS OF INTEREST

The authors declare that there is no conflict of interest in the research and preparation of this paper.

AUTHORS' CONTRIBUTIONS

Conceptualization, Fajar Pradana and Raziqa Izza Langundi; methodology, Fajar Pradana; software, Raziqa Izza Langundi; validation, Djoko Pramono; formal analysis, Nur Ida Iriani; investigation, Fajar Pradana; resources, Raziqa Izza Langundi; data curation, Raziqa Izza Langundi; writing—original draft preparation, Fajar Pradana; writing—reviewing and editing, Fajar Pradana; visualization, Raziqa Izza Langundi; supervision, Fajar Pradana; project administration, Nur Ida Iriani; funding acquisition, Nur Ida Iriani.

ACKNOWLEDGMENT

This research was funded by the Faculty of Computer Science, Universitas Brawijaya, Malang.

REFERENCES

[1] International Data Corporation (IDC) "Worldwide smartphone market forecast to grow 6.2% in 2024, fueled by Robust growth for Android in emerging markets and China, according to IDC." Access date: 23-Jan-2025. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS52757624

[2] A. Karapantelakis *et al.*, "Generative AI in mobile networks: A survey," *Ann. Telecommun.*, vol. 79, no. 1–2, pp. 15–33, Feb. 2024, doi: 10.1007/s12243-023-00980-9.

[3] D. Rimawi and S. Zein, "A static analysis of Android source code for design patterns usage," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, no. 2, pp. 2178–2186, Mar./Apr. 2020, doi: 10.30534/ijatcse/2020/194922020.

[4] S. Papadakis, M. Kalogiannakis, and N. Zaranis, "Educational apps from the Android Google Play for Greek preschoolers: A systematic review," *Comput. Educ.*, vol. 116, pp. 139–160, Jan. 2018, doi: 10.1016/j.compedu.2017.09.007.

[5] J.B. Jorgensen *et al.*, "Variability handling for mobile banking apps on iOS and Android," in *2016 13th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA)*, 2016, pp. 283–286, doi: 10.1109/WICSA.2016.29.

[6] F.M. Kundi, A. Habib, A. Habib, and M.Z. Asghar, "Android-based health care management system," *Int. J. Comput. Sci. Inf. Secur. (IJCSIS)*, vol. 14, no. 7, pp. 77–87, Jul. 2016.

[7] M. Prakash, U. Gowshika, and T. Ravichandran, "A smart device integrated with an Android for alerting a person's health condition: Internet of things," *Indian J. Sci. Technol.*, vol. 9, no. 6, pp. 1–6, Feb. 2016, doi: 10.17485/ijst/2016/v9i6/69545.

[8] G.H. Prakash *et al.*, "Development and validation of Android mobile application in the management of mental health," *Clin. Epidemiol. Glob. Health*, vol. 31, pp. 1–7, Jan./Feb. 2025, doi: 10.1016/j.cegh.2024.101894.

[9] W. Li, Y. Zhou, S. Luo, and Y. Dong, "Design factors to improve the consistency and sustainable user experience of responsive interface design," *Sustainability*, vol. 14, no. 15, pp. 1–26, Aug. 2022, doi: 10.3390/su14159131.

[10] D. Amalfitano, M. Júnior, A.R. Fasolino, and M. Delamaro, "A GUI-based metamorphic testing technique for detecting authentication vulnerabilities in Android mobile apps," *J. Syst. Softw.*, vol. 224, pp. 1–17, Jun. 2025, doi: 10.1016/j.jss.2025.112364.

[11] N. Hoshieah, S. Zein, N. Salleh, and J. Grundy, "A static analysis of Android source code for lifecycle development usage patterns," *J. Comput. Sci.*, vol. 15, no. 1, pp. 92–107, Jan. 2019, doi: 10.3844/jcssp.2019.92.107.

[12] B.S. Panca, S. Mardiyanto, and B. Hendradjaya, "Evaluation of software design pattern on mobile application based service development related to the value of maintainability and modularity," in *2016 Int. Conf. Data Softw. Eng. (ICoDSE)*, 2016, pp. 1–5, doi: 10.1109/ICODSE.2016.7936132.

[13] B.B. Mayvan, A. Rasoolzadegan, and Z.G. Yazdi, "The state of the art on design patterns: A systematic mapping of the literature," *J. Syst. Softw.*, vol. 125, pp. 93–118, Mar. 2017, doi: 10.1016/j.jss.2016.11.030.

[14] A. Naghdipour, S.M.H. Hasheminejad, and M.R. Keyvanpour, "DPSA: A brief review for design pattern selection approaches," in *2021 26th Int. Comput. Conf. Comput. Soc. Iran (CSICC)*, 2021, pp. 1–6, doi: 10.1109/CSICC52343.2021.9420629.

[15] D. Panchal, "Comparative study on Android design patterns," *Int. Res. J. Eng. Technol.*, vol. 7, no. 9, pp. 833–840, Sep. 2020.

[16] R.L.B. Baptista, "Framedrop-Mobile Client," M.S. thesis, University of Coimbra, Coimbra, Portugal, 2023.

[17] B. Wisnuadhi, G. Munawar, and U. Wahyu, "Performance comparison of native Android application on MVP and MVVM," in *Proc. Int. Semin. Sci. Appl. Technol. (ISSAT 2020)*, 2020, pp. 276–282, doi: 10.2991/aer.k.201221.047.

[18] M. Willocx, J. Vossaert, and V. Naessens, "Comparing performance parameters of mobile app development strategies," in *MOBILESoft '16, Proc. Int. Conf. Mob. Softw. Eng. Syst.*, 2016, pp. 38–47, doi: 10.1145/2897073.2897092.

[19] R.A. Doherty and P. Sorenson, "Keeping users in the flow: Mapping system responsiveness with user experience," *Procedia Manuf.*, vol. 3, pp. 4384–4391, 2015, doi: 10.1016/j.promfg.2015.07.436.

[20] F. Rösler, A. Nitze, and A. Schmietendorf, "Towards a mobile application performance benchmark," in *ICIW 2014, 9th Int. Conf. Internet Web Appl. Serv.*, 2014, pp. 55–59.

[21] G. Lim, C. Min, and Y.I. Eom, "Enhancing application performance by memory partitioning in Android platforms," in *2013 IEEE Int. Conf. Consum. Electron. (ICCE)*, 2021, pp. 649–650, doi: 10.1109/ICCE.2013.6487055.

[22] W. Ngaogate, "Applying the Flyweight design pattern to Android application development," *ASEAN J. Sci. Technol. Rep. (AJSTR)*, vol. 26, no. 2, pp. 49–57, Apr.-Jun. 2023, doi: 10.55164/ajstr.v26i2.247607.

[23] D. Rimawi and S. Zein, "A model based approach for Android design patterns detection," in *2019 3rd Int. Symp. Multidiscip. Stud. Innov. Technol. (ISMSIT)*, 2019, pp. 1–10, doi: 10.1109/ISMSIT.2019.8932921.

[24] R.F. García, "MVVM: Model–view–viewmodel," in *iOS Architecture Patterns*. Berkeley, CA, USA: Apress, 2023, pp. 145–224.

[25] X. Li, S. Wang, Z. Liu, and G. Wu, "Design and implementation of enterprise web application common framework based on model-view-viewmodel architecture," in *5th Int. Conf. Mechatron. Comput. Technol. Eng. (MCTE 2022)*, 2022, pp. 1-4, doi: 10.1117/12.2661040.

[26] M.I. Alfathar *et al.*, "Penerapan MVVM (model view viewmodel) pada pengembangan aplikasi bank sampah digital," *J. Ris. Apl. Mhs. Inform. (JRAMI)*, vol. 5, no. 2, pp. 406–414, Apr. 2024, doi: 10.30998/jrami.v5i2.11071.

[27] C.J. Sampayo-Rodríguez, R. González-Ambriz, B.A. Gonzalez-Martinez, and J. Aldana-Herrera, "Processor and memory performance with design patterns in a native Android application," *J. Appl. Comput.*, vol. 6, no. 18, pp. 53–61, Jun. 2022, doi: 10.35429/JCA.2022.18.6.53.61.

[28] F.F. Anhar, M.H.P. Swari, and F.P. Aditiawan, "Analisis perbandingan implementasi clean architecture menggunakan MVP, MVI, dan MVVM pada pengembangan aplikasi Android native," *Jupiter, Publ. Ilmu Keteknikan Ind. Tek. Elekt. Inform.*, vol. 2, no. 2, pp. 181–191, Mar. 2024, doi: 10.61132/jupiter.v2i2.155.

[29] A. Moreno-Azze, D. López-Plaza, F. Alacid, and D. Falcón-Miguel, "Validity and reliability of an iOS mobile application for measuring change of direction across health, performance, and school sports contexts," *Appl. Sci.*, vol. 15, no. 4, pp. 1–11, Feb. 2025, doi: 10.3390/app15041891.

[30] L. Corral, A. Sillitti, and G. Succi, "Mobile multiplatform development: An experiment for performance analysis," *Procedia Comput. Sci.*, vol. 10, pp. 736–743, 2012, doi: 10.1016/j.procs.2012.06.094.

[31] F. Pradana, P. Setyosari, S. Ulfa, and T. Hirashima, "Development of gamification-based e-learning on web design topic," *Int. J. Interact. Mob. Technol. (iJIM)*, vol. 17, no. 3, pp. 21–38, Feb. 2023, doi: 10.3991/ijim.v17i03.36957.

[32] S. Pargaonkar, "A comprehensive review of performance testing methodologies and best practices: Software quality engineering," *Int. J. Sci. Res. (IJSR)*, vol. 12, no. 8, pp. 2008–2014, Aug. 2023, doi: 10.21275/SR23822111402.