Journal of Technology and Informatics (JoTI)
Vol. 7, No. 2, October 2025
P-ISSN 2721-4842
E-ISSN 2686-6102

Performance Analysis of Provider and Riverpod State Management Library on Flutter Applications

Jonathan Aditya Puryanto[1*], Habibullah Akbar[2]
[1*]Informatics Engineering, Faculty of Computer Science, Esa Unggul University, Jakarta, Indonesia
[2]Master of Computer Science, Faculty of Computer Science, Esa Unggul University, Jakarta, Indonesia
e-mail: jonathanadityapuryanto@student.esaunggul.ac.id[1*], habibullah.akbar@esaunggul.ac.id[2]

Article Information

Article History:
Received : July 23th 2025
Revised : September 16th 2025
Accepted : Oktober 20th 2025
Published : Oktober 28th 2025

***Correspondence:**
jonathanadityapuryanto@student.esaunggul.ac.id

Keywords:
Application Performance Testing, Flutter, Provider, Riverpod, State Management Library

10.37802/joti.v7i2.1164

Journal of Technology and Informatics (JoTI)
P-ISSN 2721-4842
E-ISSN 2686-6102
https://e-journals.dinamika.ac.id/index.php/joti

Abstract:

State management libraries are essential components in Flutter app development. This research aims to compare the performance of the state management library Provider and its successor, Riverpod, to assist Flutter developers in choosing the right solution. Two versions of the MovieDB app were built, each utilizing Provider and Riverpod. Performance testing was conducted using three metrics: CPU Utilization, Memory Usage, and Execution Time, across three data volumes (1,000, 5,000, and 10,000). The results showed that CPU Utilization varied by only 0.1–0.2% with Riverpod being slightly more efficient at 1,000 and 10,000 data volumes. Execution Times also showed minimal differences, with Riverpod being marginally faster by approximately 0.01 seconds at 5,000 and 10,000 data volumes. Riverpod excelled in Memory Usage, demonstrating an average reduction of about 3–6% across all data volumes, particularly at higher data volumes. In conclusion, the performance of both libraries is fundamentally similar, but Riverpod is offers better memory efficiency and architectural flexibility. Therefore, Riverpod is recommended for new projects, while Provider remains a viable option for stable existing applications that already use it.

INTRODUCTION

The rapid advancement of mobile technology has led to a significant increase in mobile application development for both Android and iOS platforms [1]. This growth is driven by enhancements in hardware performance and software capabilities. As internet access becomes widespread, users increasingly rely on mobile apps for daily activities, creating a highly competitive app market [2]. This environment pushes developers to innovate continuously and deliver high-quality user experiences, with users expecting apps to be fast, stable, and efficient.

Building native applications separately for each platform making significant challenges. Different tools and programming languages for Android and iOS mean that developers often must maintain two codebases, which increases both development time and cost. To solve this, Google introduced Flutter, an open source cross-platform framework. Flutter allows developers to create applications from a single codebase that runs on multiple platforms, including Android and iOS [3]. Flutter has become one of the most popular choices for cross-platform mobile application development [4].

Performance is a critical factor in mobile app development, including efficiency, responsiveness, and stability, all of which significantly influence user experience [5]. Despite technological advancements, mobile apps continue to face performance challenges. For example, a survey by AppDynamics revealed that slow loading times, crashes, and freezes are major sources of user frustration, with 55% of respondents reporting negative impacts from these issues [6]. These findings highlight that optimizing performance is essential for maintaining user satisfaction.

In Flutter, user interface components are called widgets. These widgets are immutable, meaning their properties cannot be modified at runtime [7]. Therefore, Flutter relies on the concept of state. State can change based on user interactions and can be accessed synchronously during widget construction. It may also change over the widget's lifetime [8]. When state changes, Flutter rebuilds affected widgets by creating new instances based on new the state. The built-in setState() method initiate this rebuild process. setState() calls the widget's build method and reconstructs the UI based on the new state. However, setState() can cause unnecessary rebuilds across all widgets in the current tree, even those unrelated to the changed state [9]. As an application's widget hierarchy becomes more complex, this inefficiency can waste resources and degrade application performance.

To address these limitations, Flutter developers use state management techniques to control which UI parts are rebuilt upon state changes. State management decouples state from UI components, allowing only dependent widgets to update. Using state management libraries can reduce resource consumption and improve app performance efficiency [9]. One of the most commonly used state management library is Provider [10]. Provider works by wrapping InheritedWidget, developers typically manage state with ChangeNotifier, and use a Consumer to listen for updates for widgets that need that state. However, there is a successor to Provider called Riverpod. Riverpod improves this approach by eliminating the dependency on InheritedWidgets and classes from the Flutter SDK. Allowing to define state as a global variables and access them using specialized widgets such as ConsumerWidget and WidgetRef.

Previous research has shown that the choice of state management library can affect Flutter app performance. For example, research [9] demonstrated that, for the same application, using Provider resulted in better performance than using the BLoC State Management Library or setState(). Similarly, research [4] found that Provider outperformed another popular library, GetX. Although Provider is efficient in many cases, research [7] assume that Provider is suitable only for small to medium-sized apps, as managing multiple ChangeNotifier instances in larger apps can cause performance bottlenecks. Riverpod aims to address some of Provider limitations and potentially offers better performance. However, there are no direct experimental comparisons betweeen Provider and Riverpod have been conducted recently. This represents a gap in the research, it is not yet clear whether Riverpod design improvements translate into measurable performance gains over Provider.

To fill this gap, this study conducts a performance analysis of Provider and Riverpod within a Flutter application. A simple movie catalog app named MovieDB is developed in two

versions, one using Provider and the other using Riverpod. Both versions share the same functionality and UI design. Their performance is evaluated under different test scenarios and varying data volumes, representing application complexity [11]. Key performance metrics, including CPU Utilization, Memory Usage, and Execution Time, are collected for each scenario [4,5,9,10,11]. By analyzing the results, this study identifies which state management approach is more efficient based on the application scale, providing recommendations for Flutter developers when choosing between Provider and Riverpod based on their applications needs.

METHOD

This study employs an experimental approach to evaluate the performance of two state management libraries, within a controlled environment. Two identical versions of a movie catalog app named MovieDB were developed, differing only in their state management implementation. This design allows the performance evaluation to focus specifically on the influence of the state management library used.

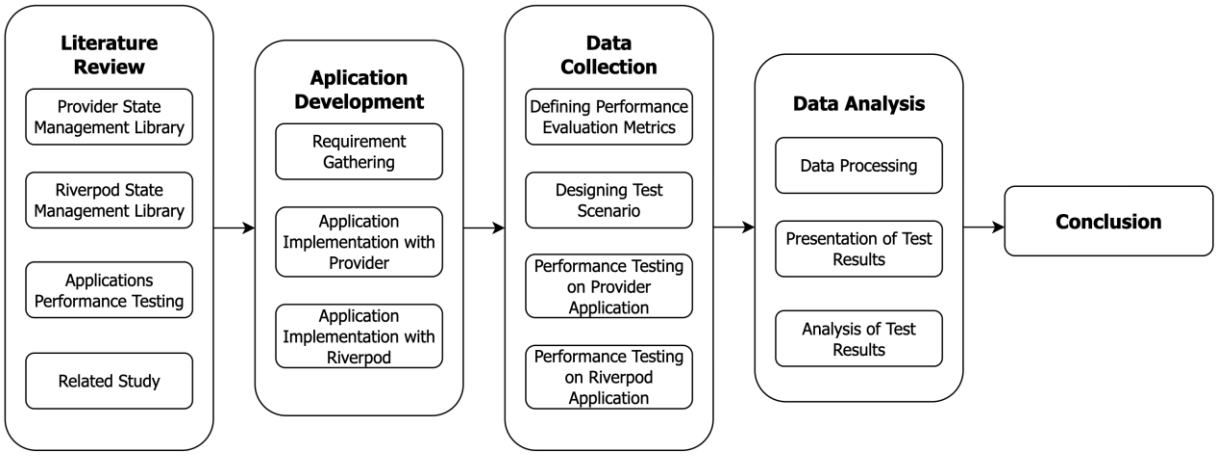

Figure 1. Research Methodology

Literature Review

The literature review phase is carried out to study the sources and materials needed to conduct research. This analysis of literature and study results is done to identify the factors that affect performance or the performance metrics like CPU utilization, memory usage, and execution time. The literature review also supports the design of test scenarios and the selection of tools such as Snapdragon Profiler and Flutter DevTools. Additionally, it justifies the choice of a movie catalog application as a testing medium, given its simplicity, scalability, and common UI and data interaction patterns in Flutter apps.

Application Development

The application development process starts with gathering requirements to define the application's needs. This research aims to show how different state management approaches affect Flutter application performance. To ensure an objective comparison, both app versions will have same appearances and functionalities, differing only in state management implementation.

Previous studies on state management performance have often utilized movie catalog applications as test subjects. For example, the MovDB application [9] and ShowTime application [4] provide relevant benchmarks. These applications demonstrate that the movie catalog format is suitable for testing various performance aspects, as their structure allows for easy

modification of the displayed movie data. This capability enables simulations of heavier application loads and more complex scenarios by adjusting the volume of movie data shown.

Based on this context, the criteria for the new MovieDB application designed for state management performance testing are as follows:

1. Multi-page Navigation
   a. Movies Page: Displays a scrollable list of movies, featuring an AppBar that shows the amount of displayed movies and popup button to adjust the quantity of the movies.
   b. Search Page: Allows users to search for movies by title.
   c. Settings Page: Allows users to toggle dark mode, affecting the entire application, including the Movies and Search pages.
2. Remote API Calls
   The application will implement remote API calls to fetch movie data. The TMDB API, specifically the Popular Movies endpoint, will be used to retrieve a list of currently popular movies. This API was selected for its ease of implementation and comprehensive documentation, ensuring a fast data-fetching process, which is critical for testing.

Data Collection

Performance testing is essential for assessing application performance during software development. This involves evaluating the speed, effectiveness, and resource utilization of software and hardware [13]. Based on previous research, several performance evaluation metrics are relevant for assessing state management efficiency in Flutter applications:

1. CPU Utilization
   CPU utilization means the percentage of CPU resources consumed by the application. CPU performance may be affected by the efficiency of state management. Complex or inefficient management can lead to excessive overhead, increasing CPU resource usage. Optimizing state management is essential for maintaining good application performance [9].
2. Memory Usage
   Memory usage refers to the amount of memory consumed by the application. Inefficient state management can negatively impact memory usage. Poor management of state data may result in excessive memory consumption or unnecessary data accumulation. Selecting an efficient state management method is vital for optimizing memory [9].
3. Execution Time
   Execution time is the duration from when a user interaction triggers a state update to when new widgets are rebuilt. State management influences execution time, state updates can be time-consuming if not well organized, potentially increasing execution time [14].

The Performance Metrics will be evaluated across three test scenarios, this experimental scenario is structured based on the factors identified in the literature analysis, as detailed in Table 1 below:

Table 1. Test Scenario

Scenario ID	Description
Sc-01	Scrolling the movie cards on the movie page from first to last.
Sc-02	Searching a movie with the keyword “dark” and scrolling the results.
Sc-03	Changing the app theme (light/dark) and navigating to Search page and Movie page

Data Analysis

Analysis of test data involves interpreting results from experiments to draw conclusions [15]. The analysis of test data in this study involved categorizing application complexity by data

volume into three levels: Low Data (1,000 records), Medium Data (5,000 records), and High Data (10,000 records). These varying data volumes illustrate the increasing complexity of the application, allowing for an evaluation of how state management performs as the data volume increases and the application becomes more complex. Each scenario is tested 20 times across both application versions. The experiment was conducted on a Xiaomi Mi 8 with Android 15, 6GB RAM, and a Qualcomm Snapdragon 845 processor. Performance monitoring tools, including Snapdragon Profiler and Flutter DevTools used for measuring performance data. The data analysis process consists of two stages:

1. Average per Testing Scenario
   The values from the 20 runs of each scenario are averaged. The graph will illustrate the app performance of each scenario (Sc-01, Sc-02, Sc-03) across different data volumes, facilitating comparative analysis.
2. Overall Average Based on Data Volume
   All averaged data from the three scenarios are combined to calculate the overall average for each metric based on data volume. This stage identifies the most efficient state management method for small, medium, or large data scales, forming the basis for conclusions regarding the optimal state management approach for a flutter application.

The analysis will provide interpretations of the quantitative data from the tests, leading to recommendations for Flutter developers on selecting the optimal state management approach. Key findings will address the research objectives, determining which method is more efficient overall. These insights will guide flutter developers in making informed choices for different application scales.

RESULTS AND DISCUSSION

Aplication Development

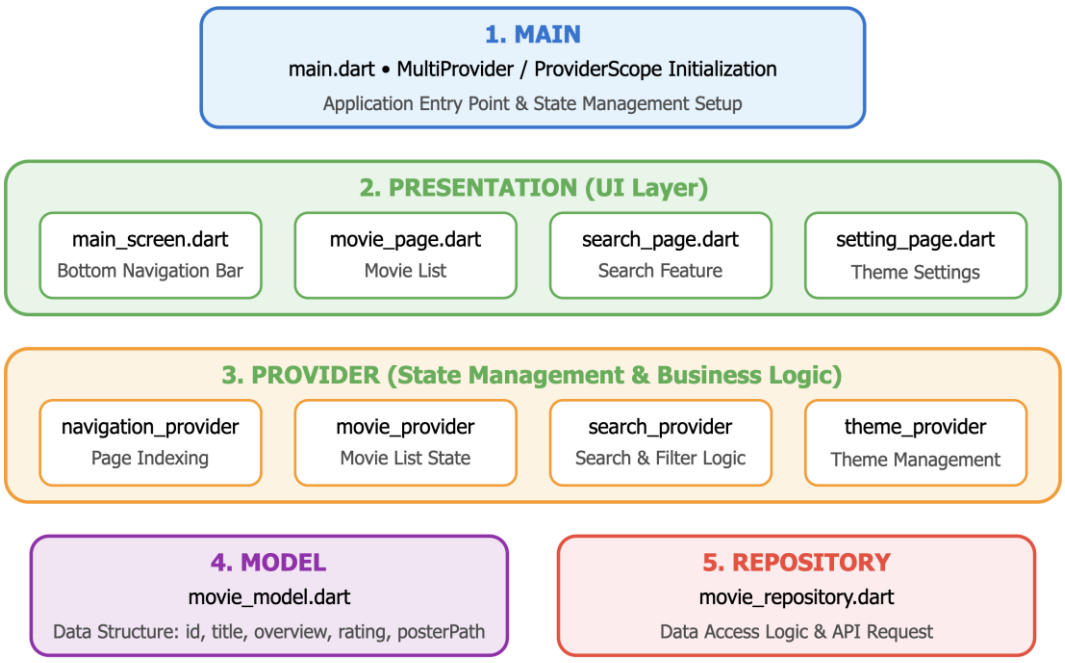

Figure 2. MovieDB Application Architecture

The MovieDB application features a modular clean architecture with clear separation of concern. The architecture consists of five key components:

1. main: Contains the entry point (main.dart), initializing the app with runApp() and global configurations like theme and routing.
2. presentation: Serves the user interface with various pages and widgets.
3. model: Stores data structures, like the Movie model.
4. repository: Acts as the data source, providing static movie data from TMDb API.
5. provider: Manages state and application logic, bridging data to the UI and vice-versa.

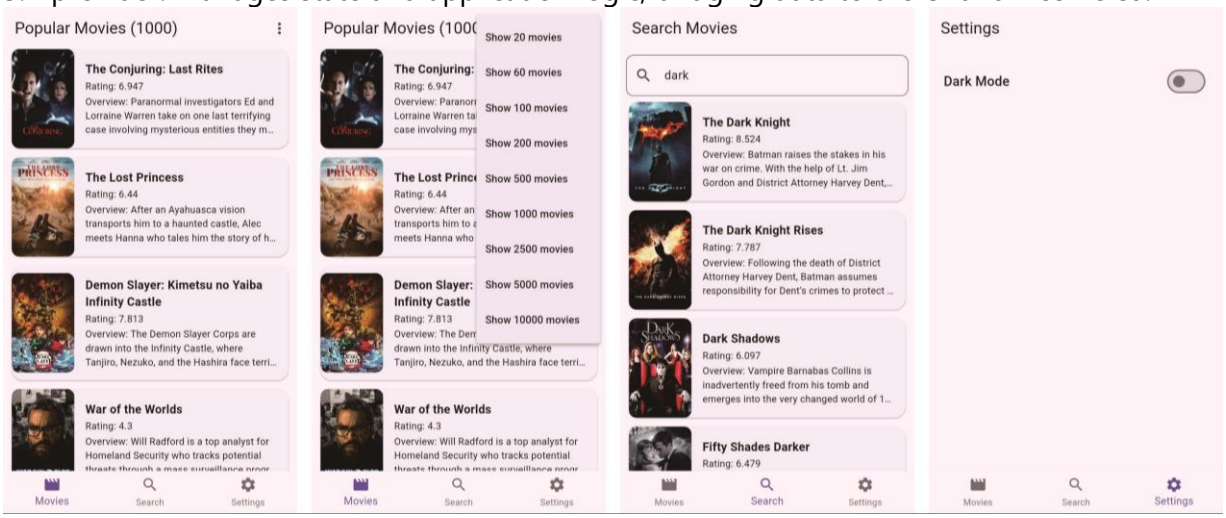

Figure 3. MovieDB User Interface Implementation

Data Collection

Performance testing was conducted to evaluate the efficiency of the state management libraries, across various application resource metrics. For CPU utilization and memory usage, tests were conducted by building application in release mode and then using the Realtime Performance Analysis feature of the Snapdragon Profiler. Meanwhile, execution time were measured by building application in profile mode to use logging features in Flutter DevTools.

CPU Utilization and Memory Usage

CPU utilization is recorded as a percentage (%), while memory usage is measured in bytes and then converted to megabytes (MB). The performance data was exported to CSV format for analysis. To calculate averages for both metrics, the start and end timestamps of the test are identified, and the values between these timestamps are averaged for each scenario.

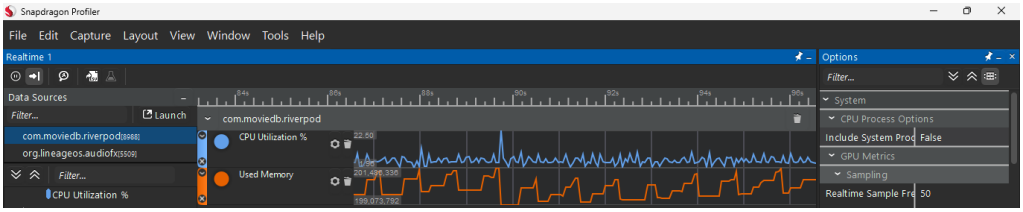

Figure 4. Snapdragon Profiler Realtime Performance Analysis

Execution Time

Execution time were measured using Flutter DevTools. Execution time is determined by recording timestamps at the beginning and end of the code segment being evaluated. The difference between these timestamps is calculated to express the total duration in seconds.

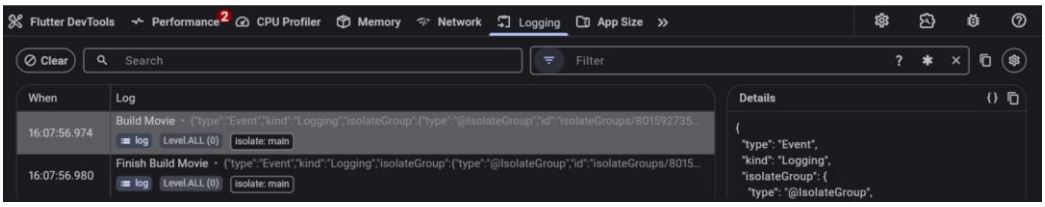

Figure 5. Flutter DevTools Logging

Data Analysis

This section assesses the performance of Provider and Riverpod in a Flutter application. Key metrics such as CPU Utilization, Memory Usage, and Execution Time are analyzed across varying data volumes to identify efficiency differences.

Cpu Utilization Analysis

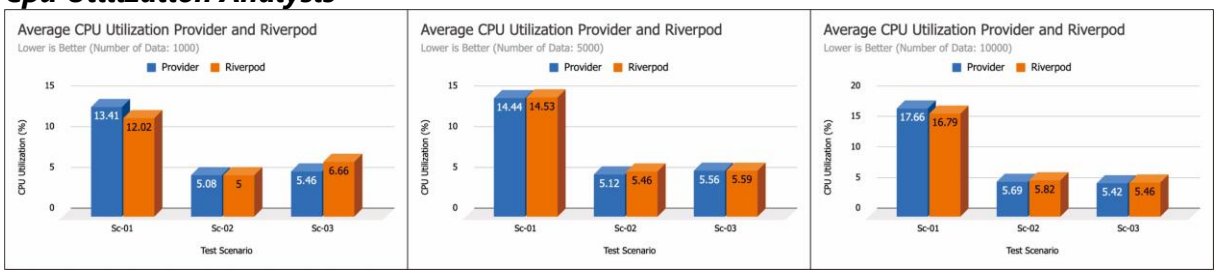

Figure 8. CPU Utilization Measurement Results

For a data volume of 1,000, CPU utilization varied across scenarios, with Provider peaking at 13.41% in Sc-01 compared to Riverpod 12.02%. In Sc-02, both libraries showed lower CPU utilization around 5%, with Provider at 5.08% and Riverpod at 5.00%. During the Sc-03, Riverpod consumed more CPU at 6.66% compared to Provider's 5.46%. As the data volume increased to 5,000, CPU utilization rise, especially in Sc-01, where Provider reached 14.44% and Riverpod 14.53%. In Sc-02, Riverpod's CPU usage increased slightly to 5.46%, while Provider was at 5.12%. At a volume of 10,000, CPU utilization peaked at 17.66% for Provider and 16.79% for Riverpod in Sc-01, with both libraries maintaining similar values in Sc-02, 5.69% for Riverpod vs. 5.82% for Provider and Sc-03 5.42% for Provider and 5.46% for Riverpod.

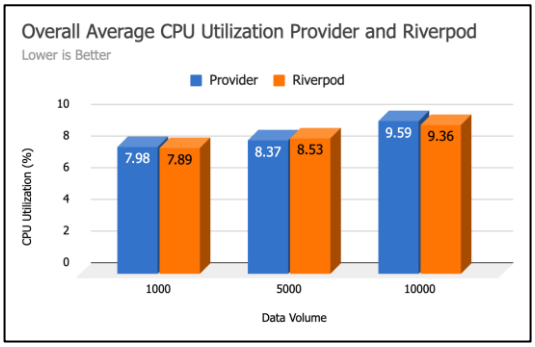

Figure9. CPU Utilization Comparison (%)

The overall average CPU utilization shows very close values across all data volumes, with differences between the two state management libraries around 0.1–0.2%. Overall, there is no significant difference in CPU usage metrics, indicating that both architectures manage state changes with nearly equal efficiency despite minor variations. At 1,000 data, Provider averaged 7.98% while Riverpod was at 7.89%. At 5,000 data, Provider was at 8.37% compared to Riverpod's 8.53%. Finally, at 10,000 data, Provider reached 9.59% versus Riverpod's 9.36%. These small fluctuations may be attributed to architectural differences, as Provider uses

BuildContext while Riverpod uses WidgetRef, potentially reducing state tracking load. Overall, both Provider and Riverpod demonstrate comparable CPU utilization efficiency.

Memory Usage Analysis

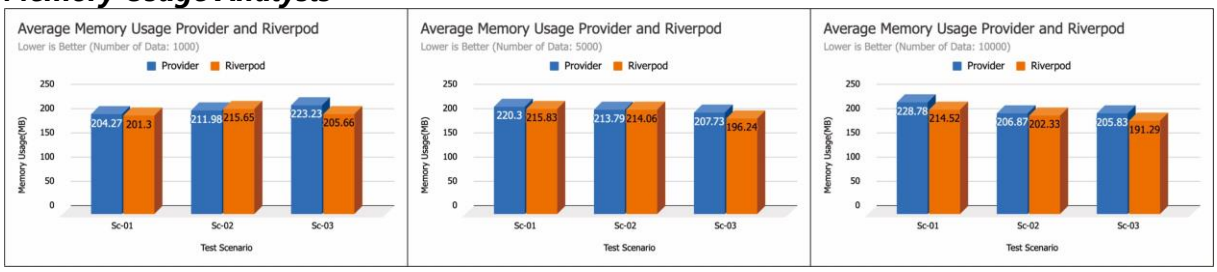

Figure 60. Memory Usage Measurement Results

For a data volume of 1,000, memory usage ranged between 200-220 MB. In Sc-01, Provider used 204.27 MB, while Riverpod consumed 201.30 MB. In Sc-02, Provider increased to 211.98 MB, with Riverpod slightly higher at 215.65 MB. During Sc-03, Provider had the highest memory usage at 223.23 MB, compared to Riverpod 205.66 MB. As the volume increased to 5,000, Provider memory usage rise to 220.30 MB and Riverpod to 215.83 MB in Sc-01, while Sc-02 showed nearly equal usage around 213–214 MB for both libraries. In Sc-03, memory usage decreased to 207.73 MB for Provider and 196.24 MB for Riverpod, with Provider still consuming more memory. At a volume of 10,000, Provider's memory usage was 228.78 MB in Sc-01 compared to Riverpod 214.52 MB, and in Sc-02, Provider used 206.87 MB versus Riverpod 202.33 MB. Overall, Riverpod demonstrated better memory efficiency.

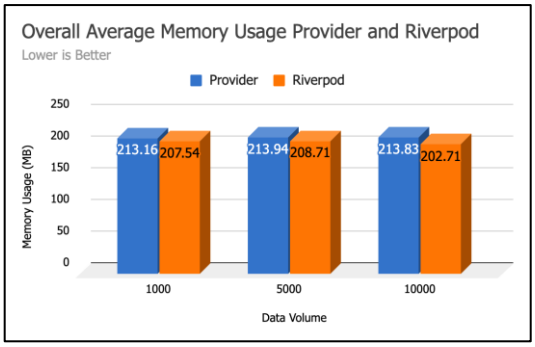

Figure 1. Memory Usage Comparison (MB)

The overall average memory usage indicates that Riverpod consistently consumes slightly less memory than Provider. For instance, at a volume of 10,000 data points, Provider used 213.83 MB while Riverpod only used 202.71 MB, showing a difference of about 5%. A similar pattern appears at 1,000 and 5,000 data points. This difference suggests that applications built with Riverpod are more memory-efficient. Specifically, at 1,000 data points, Provider consumed 213.16 MB compared to Riverpod 207.54 MB, and at 5,000, Provider was at 213.94 MB while Riverpod was 208.71 MB. Riverpod efficiency comes from its use of autoDispose, which automatically releases unused states. In contrast, Provider relies on InheritedWidget and BuildContext, which can result in longer retention of states in memory. This makes Riverpod the more efficient choice for managing memory usage.

Execution Time Analysis

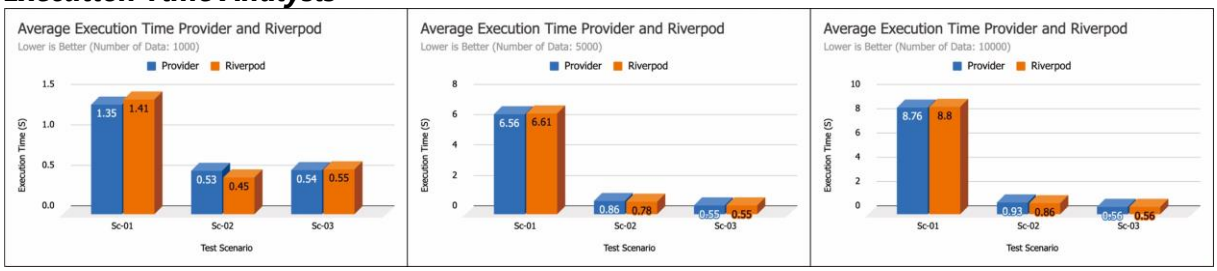

Figure 72. Execution Time Measurement Results

For a data volume of 1,000, execution time was relatively quick, with Sc-01 taking 1.35 seconds for Provider and 1.41 seconds for Riverpod. In Sc-02, execution times were short, at 0.53 seconds for Provider and 0.45 seconds for Riverpod, showing Riverpod advantage. For Sc-03, both libraries were nearly identical, taking around 0.54 seconds. As the volume increased to 5,000, Sc-01 has longer execution times of 6.56 seconds for Provider and 6.61 seconds for Riverpod, while Sc-02 rise to 0.86 seconds for Provider and 0.78 seconds for Riverpod. In Sc-03, both libraries performed similarly. At a volume of 10,000, execution times extended to 8.76 seconds for Provider and 8.80 seconds for Riverpod in Sc-01, while Sc-02 took 0.93 seconds for Provider and 0.86 seconds for Riverpod, with Sc-03 both remaining around 0.56 seconds

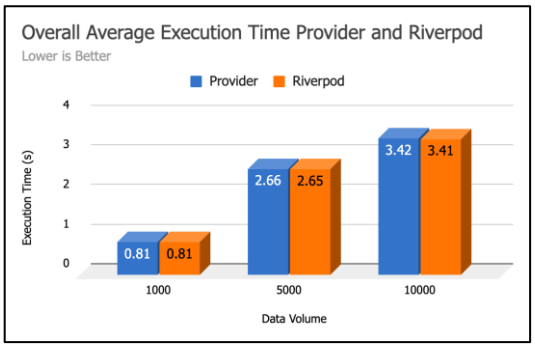

Figure 13. Execution Time Comparison (s)

The overall average execution time for both approaches across various data volumes is nearly identical. For 1,000 data, both libraries took 0.81 seconds. At 5,000 points, Provider was at 2.66 seconds and Riverpod at 2.65 seconds, and at 10,000 points, Provider took 3.42 seconds while Riverpod took 3.41 seconds. The differences are minimal, around 0.01 seconds, are practically insignificant. This similarity is reasonable since the application logic and data processing remain the same regardless of the state management method. Riverpod may have slight overhead from accessing global providers via WidgetRef, while Provider uses Flutter built in BuildContext, but these differences are negligible. With Flutter's compiler optimizations, both architectures achieve nearly identical runtime efficiency. Therefore, it can be concluded that execution time is not significantly affected by the choice of state management library, supporting the finding that both methods provide equivalent runtime performance.

CONCLUSIONS AND SUGGESTIONS

In Flutter application development, selecting a state management library can significantly impacts performance. This study compared Provider and its successor, Riverpod, focusing on performance metrics. Overall, both libraries demonstrated similar performance. CPU utilization showed minimal variation, averaging around 0.1–0.2% across all scenarios, while execution times differed by only 10–30 milliseconds. Riverpod had an advantage in memory

usage, consuming 3–6% less memory, especially with larger data volumes. Based on the findings, Riverpod is recommended for new Flutter projects needing easier state management, greater memory efficiency, and scalability. However, for existing projects using Provider, migration for minor performance differences is unnecessary. Ultimately, the choice of library should depend on each project's specific needs. If memory efficiency and modern architecture are priorities, Riverpod is ideal, but Provider remains a valuable option. Future research should test the performance of additional state management libraries, such as GetX, BLoC, or MobX, to identify optimal solutions for various Flutter contexts. Additionally, testing on iOS platforms is recommended to assess performance in different ecosystems.

REFERENCES

[1] Endah Puspitarini, Roudhotul Hanifa, and Faridatun Nadziroh, "Rancang Bangun Aplikasi Absensi Mahasiswa Pada Platform Android," *Journal of Technology and Informatics (JoTI)*, vol. 2, no. 1, pp. 48–55, Oct. 2020, doi: 10.37802/joti.v2i2.114.

[2] Nur Moniroh and Rafiq Chasnan Habibi, "Implementation of UAT and Blackbox Methods in the Android-Based Prayer Collection and News Portal Application of PP El-Bayan," *Journal of Technology and Informatics (JoTI)*, vol. 7, no. 1, pp. 76–89, Apr. 2025, doi: 10.37802/joti.v7i1.884.

[3] M. Zulistiyan, M. Adrian, Y. Firdaus Arie Wibowo, and J. Telekomunikasi, "Performance Analysis of BLoC and GetX State Management Library on Flutter," *Journal of Information System Research (JOSH)*, vol. 5, no. 2, pp. 583–591, 2024, doi: 10.47065/josh.v5i2.4698.

[4] I. Husain, P. Purwantoro, and C. Carudin, "Analisis Performa State Management Provider Dan GetX Pada Aplikasi Flutter," *JATI (Jurnal Mahasiswa Teknik Informatika)*, vol. 7, no. 2, pp. 1417–1422, Sep. 2023, doi: 10.36040/jati.v7i2.6867.

[5] M. Abdul Hakeem, M. Abdul Razack Maniyar, and M. Khalid Mubashir Uz Zafar, "Performance Testing Framework for Software Mobile Applications," *Int J Innov Res Sci Eng Technol*, vol. 7, pp. 6225–6234, 2020, [Online]. Available: www.ijirset.com

[6] AppDynamics, "The App Attention Index 2019: The Era of the Digital Reflex," AppDynamics. Accessed: Oct. 11, 2024. [Online]. Available: https://www.appdynamics.com/blog/news/app-attention-index-2019/

[7] D. Slepnev, "State Management Approaches In Flutter," South-Eastern Finland University of Applied Science, 2020. Accessed: Oct. 11, 2024. [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/355086/Dmitrii_Slepnev.pdf

[8] Flutter Team, "State class - widgets library - Dart API." Accessed: Jul. 14, 2025. [Online]. Available: https://api.flutter.dev/flutter/widgets/State-class.html

[9] R. R. Prayoga, G. Munawar, R. Jumiyani, and A. Syalsabila, "Performance Analysis of BLoC and Provider State Management Library on Flutter," *Jurnal Mantik*, vol. 5, no. 3, pp. 1591–1597, 2021, Accessed: Oct. 11, 2024. [Online]. Available: https://iocscience.org/ejournal/index.php/mantik/article/view/1539

[10] pub.dev, "Search results for state management (sorted by downloads)." Accessed: Jul. 14, 2025. [Online]. Available: https://pub.dev/packages?q=state+management&sort=downloads

[11] Mgs. M. F. Abdillah, I. L. Sardi, and A. Hadikusuma, "Analisis Performa GetX dan BLoC State Management Library Pada Flutter untuk Perangkat Lunak Berbasis Android," *LOGIC: Jurnal Penelitian Informatika*, vol. 1, no. 1, p. 73, Sep. 2023, doi: 10.25124/logic.v1i1.6479.

[12] M. Hafid, N. Azis, A. Pinandito, I. Sartika, and E. Maghfiroh, "Analisis Perbandingan Penggunaan State Management pada Aplikasi Ditonton menggunakan Framework Flutter," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 7, no. 1, pp. 148–153, 2023, Accessed: Oct. 11, 2024. [Online]. Available: https://j-ptiik.ub.ac.id/index.php/j-ptiik/article/view/12124

[13] M. A. Putri, H. N. Hadi, and F. Ramdani, "Performance testing analysis on web application: Study case student admission web system," in *2017 International Conference on Sustainable Information Engineering and Technology (SIET)*, IEEE, Nov. 2017, pp. 1–5. doi: 10.1109/SIET.2017.8304099.

[14] A. A. D. Jatnika, M. A. Akbar, and A. Pinandito, "Comparative Analysis of the Use of State Management in E-commerce Marketplace Applications Using the Flutter Framework," *Journal of Information Technology and Computer Science*, vol. 8, no. 2, pp. 111–124, Aug. 2023, doi: 10.25126/jitecs.202382557.

[15] K. Afandi, M. H. Arief, N. Faizatul Laily, and D. Maulana Nugroho, "Analisis Performa Akademik Mahasiswa Menggunakan Social Network Analysis (Studi Kasus: Prodi Bisnis Digital Universitas dr. Soebandi)," *Journal of Technology and Informatics (JoTI)*, vol. 5, no. 2, pp. 64–69, Apr. 2024, doi: 10.37802/joti.v5i2.514.