PartFlow: A Visualization Tool for Application Partitioning and Workload Offloading in Mobile Edge Computing

Boda Li*
University of Washington
Tacoma, WA, United States

Minghao Li†
Nanyang Technological University
Singapore

Jian Zhao‡
University of Waterloo
Waterloo, ON, Canada

Wei Cai§
University of Washington
Tacoma, WA, United States

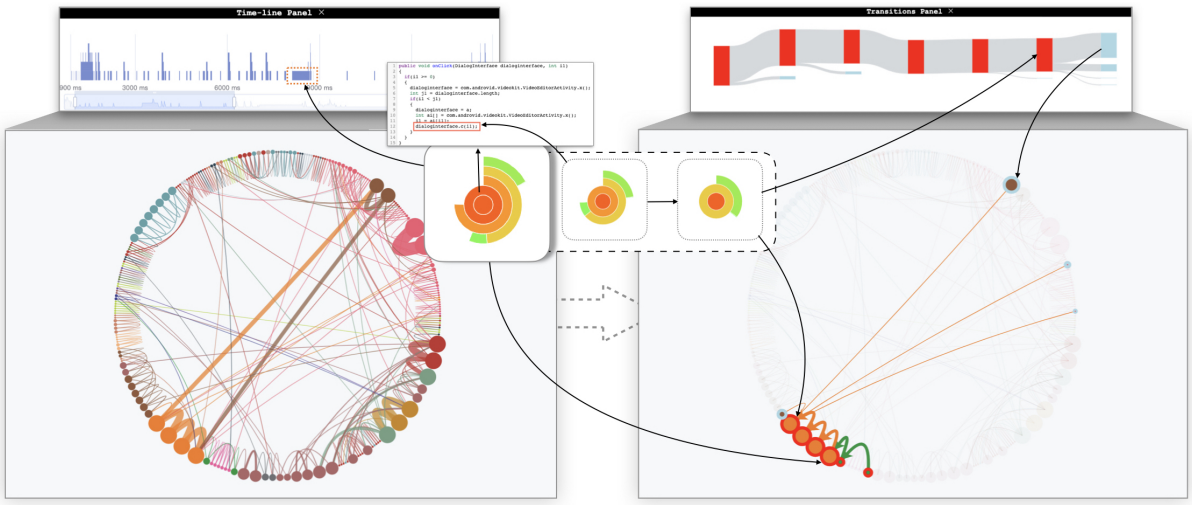


Figure 1: A mobile computing researcher interacts with PartFlow to analyze method-level component sequences within a mobile application. The system dynamically updates various panels, including the Timeline and Transition Panels, providing visual insights into component performance, state transitions, and code structure. Selecting nodes within the interactive digraph reveals detailed component relationships and allows for deeper investigation into application execution patterns.

Abstract

In mobile edge computing (MEC), one optimization strategy for mobile applications is to offload heavy computing tasks to cloud and edge servers. Constructing partitioning algorithms involves modeling individual methods through static code profilers, but exploiting dynamic user-driven execution patterns is also crucial. This paper introduces PartFlow, an interactive visualization system that supports comprehensive analysis of mobile application components and aids researchers in developing partitioning and offloading algorithms using real human behavioral data. PartFlow collects application component data remotely through binary instrumentation of mobile applications. Interactive diagrams are designed to evaluate component performance and illustrate transition patterns using the collected data. Additionally, PartFlow integrates a deep learning (DL)-based approach for multi-step forecasting of component states to improve accuracy and user experience in algorithm design. A case study and user feedback demonstrate PartFlow's effectiveness in assisting researchers and engineers in creating offloading strategies.

Index Terms: Human-centered computing—Visualization—Visualization techniques—Treemaps; Human-centered computing—Visualization—Visualization applicatin domains—information visualization

*e-mail: lib32@uw.edu
†e-mail: minghao002@e.ntu.edu.sg
‡e-mail: jianzhao@uwaterloo.ca
§e-mail: weicaics@uw.edu

1 Introduction

In recent years, mobile applications have proliferated across categories such as entertainment, health, games, business, social networking, travel, and news [13]. Through mobile cloud computing (MCC), mobile users can access rich computational resources. Mobile edge computing (MEC) addresses the latency challenges of MCC by leveraging hardware at the network edge [8]. However, these edge nodes lack the computational power compared to cloud servers, necessitating the offloading of only specific heavy workloads to edge servers.

Most existing partitioning schemes analyze execution paths among application methods using static code analyzers, breaking down applications into fine-grained ***components*** [6, 9, 10, 23]. These methods model applications as graphs with components as vertices and call relations as edges, enabling researchers to make offloading decisions based on heuristic or optimization strategies. While partitioning computational components and executing heavier ones remotely can reduce the burden on mobile devices, frequent switches between local and remote executions can lead to significant energy consumption and response delays [27]. Understanding user-driven execution patterns is critical for enhancing offloading strategies. To optimize MEC systems, minimizing redundant data transmission between end-user devices and edge servers is essential.

Dynamic partitioning approaches acknowledge the importance of runtime execution patterns [16, 30]. However, these strategies face four main challenges: (1) Difficulty in obtaining execution patterns, modeling data, and user behavior insights across diverse tools and environments; (2) Lack of tools for collecting extensive and comprehensive component execution data from applications in

heterogeneous environments; (3) Existing tools focus on static code analysis, missing the stochastic nature of user-driven component execution; (4) The challenge of selecting optimal methods from thousands of components for partitioning and offloading.

To address these challenges, this paper introduces PartFlow, an interactive visualization tool for analyzing user-driven execution patterns to aid in application partitioning and workload offloading. PartFlow provides comprehensive data visualization to help researchers select component sequences for dynamic analysis. The system collects data from application components using binary instrumentation, a technique that inserts additional code into a binary executable at runtime or compile time to monitor and analyze the program's behavior. PartFlow also employs various analytical approaches to assess component performance. Visualizations such as flame graphs [19] and sunburst charts [3] illustrate component statistics, while an interactive digraph displays state transition patterns. PartFlow's Sankey-based decision tree [20] simulates user-driven component sequences, aiding researchers in strategizing component offloading.

This research contributes to the fields of visualization and Mobile Edge Computing (MEC) by introducing an interactive visualization tool, PartFlow, that addresses key challenges in application partitioning and workload offloading. We contribute a comprehensive understanding of the workflow and challenges faced by MEC researchers, offering a visual analytics solution tailored to their needs. PartFlow combines advanced data acquisition methods, predictive analytics, and interactive visualizations—such as flame graphs, sunburst charts, and Sankey-based decision trees—to enable efficient exploration of execution patterns and system performance. Our results demonstrate that PartFlow supports an effective analysis workflow, bridging historical and predictive insights while encouraging interactive decision-making for better optimization strategies. The paper concludes by discussing the broader implications of our findings on visualization's role in supporting dynamic, user-driven system analysis and optimization in MEC research.

2 Related Work

In this section, we review related work on visual analytics techniques and designs for profiler output, methods relationships, and uncertainties in the event sequence.

2.1 Visualization for Profiler Output

Professional performance analysis tools, such as Android Profiler, Solaris Studio, and Java Flight Recorder (JFR), are often used for statistical prediction methods when testing application performance. These tools can collect extensive data, including application stack information, real-time device status, and total power consumption. However, the sheer volume and complexity of this data make it difficult for users to identify specific features of interest. To mitigate this challenge, JFR, a JVM-specific analysis tool, employs line charts and pie charts to display CPU usage and stack information. Similarly, Solaris Studio and Android Profiler have adopted the Flame Graph [19] to visualize the duration and hierarchy of thread execution. Other performance analysis visualization tools include Frequency Trails [18] and Latency Heatmap [19]. Despite their utility, these tools often require users to zoom in to focus on specific components, and they generally lack interactivity and the ability to isolate a single component view. Additionally, they do not indicate the weight of execution time in detail. PartFlow addresses these limitations by enhancing the user experience for single-component analysis. It can be integrated with subsequent visual panels to help users quickly access and analyze component performance data.

2.2 Visualization for Methods Relationship

In the study of dynamic partitioning, it is essential to analyze not only individual components but also the components that precede or follow their execution, as well as related components with call relationships. The current approach to addressing this issue involves visualizing method call graphs. For instance, FlowDroid[1] and TaintDroid [12] use Gephi[2] to visualize digraphs of numerous nodes. However, FlowDroid, which relies on Soot[3], primarily focuses on static bytecode call relationships without taking into account the influence of user behavior and device performance on component state timing during the execution of Android programs. On the other hand, the more dynamic TaintDroid [12] operates at the variable level, tracing untrusted app code through the virtual machine (VM) interpreter. This reverse engineering tool is mainly used for taint analysis and static call diagrams, showcasing static function dependencies as directed graphs. These tools tend to prioritize static code analysis and lack the capability to dynamically assess the sequence of component states in conjunction with actual user interactions with the applications. PartFlow differentiates itself from these tools by focusing on user-driven component states and illustrating their state transition relationships using a stochastic matrix. With PartFlow, users can gain insights into component state transition patterns, aiding in the development of collaborative dynamic partitioning strategies for multiple components.

2.3 Visualization of Uncertainty and Event Sequence

Application partitioning involves the entire sequence of components, implying that adjacent components are offloaded together across one or several edges. Therefore, effective visualization of event sequences is essential. Many existing visualization tools can analyze how different event sequences lead to various outcomes, which can help analysts generate hypotheses about causation. Examples include DecisionFlow [17], OutFlow [31], CareFlow [26], and MatrixWave [34], which aggregate similar event sequences into progression pathways and visually encode the correlations between these pathways and potential outcomes. Moreover, when selecting sequences of components for analysis, state transitions often rely not only on the internal logic of the application but also on program structures and user behavior. Therefore, presenting uncertainty effectively in dynamic partitioning is crucial for enhancing user comprehension and decision-making quality. Extensive surveys [4] indicate that certain techniques incorporate uncertainty visualization using extra visual components such as glyphs [24], geometric elements like contour lines and isosurfaces [28], or annotations [7]. However, these approaches typically focus on the state of each node without considering the probability of the entire sequence occurring. PartFlow addresses this gap by allowing users to select component queues with high occurrence probabilities and develop tailored dynamic partitioning strategies based on these component sequences.

3 Designing PartFlow

As the initial stage of our iterative user-centered design process, we conducted a literature survey and interview study to gather essential design requirements for PartFlow.

3.1 Research Study

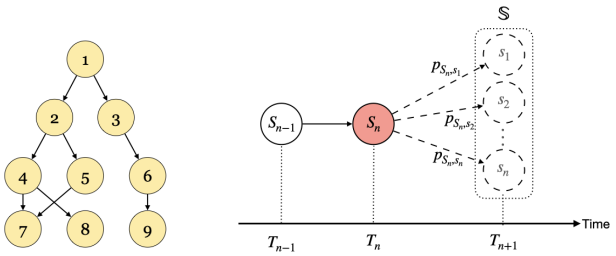
We conducted an investigation to review the challenges addressed in existing academic research. The findings are summarized in three main areas:

Model Analysis and Statistical Prediction of Methods: Evaluating method-level components is essential for constructing partitioning algorithms and generally involves two main approaches: model analysis [9, 10] and statistical prediction [23, 33]. The model analysis approach typically incorporates factors such as time and space complexity, arguments, and return values of method invocations to calculate component metrics using static code analysis. On

[1] https://github.com/secure-software-engineering/FlowDroid
[2] https://gephi.org/
[3] https://github.com/soot-oss/soot

the other hand, the statistical prediction approach leverages profiling tools to directly record the execution times of components by capturing their entry and exit points, thus providing a practical measure of component execution efficiency.

Call Tree of an Individual Method: Constructing the call graph of an application is fundamental for partitioning algorithms [1, 22]. Building call trees for each method-level component requires decomposing the application into individual methods and analyzing their execution patterns. By merging these trees, static analysis can create a control call graph, where vertices represent components and edges depict the call relationships between them. When partitioning an individual method, it is crucial to focus on its specific call tree, as demonstrated in Figure 2(a).



(a) Method's call tree. (b) Method states transition

Figure 2: Two different relations among methods. Subfigure (a) illustrates the call tree of an application. Subfigure (b) shows the transition relationship among method states.

User-driven Execution Patterns of a Set of Methods: Understanding the execution patterns of a set of methods within an application is crucial for offloading systems to save unnecessary energy consumption and reduce latency [16]. When offloading heavy computational tasks, the data processed and returned is transmitted between mobile devices (MDs) and edge servers over networks [5]. Figure 3 illustrates the data transmission flow in a mobile cloud/edge application. Although optimal scheduling can help minimize response time and energy costs, there can still be significant latency due to frequent switching between the device and the edge. To mitigate this inefficiency, it is essential to incorporate execution patterns driven by stochastic characteristics of user behavior [32]. Figure 2(b) shows the data transmissions before and after executing a component state $S_n \in \mathbb{S}$. The invocation graph highlights that each potential alternative s_i for S_n has an associated probability p_{S_n,s_i}, leading to different execution paths.

To model the relationships among method invocations, a Markovian model provides a foundational approach for calculating transition probabilities [16].

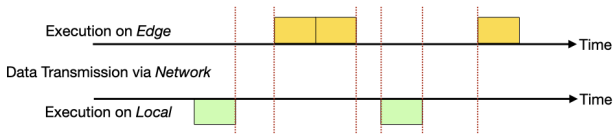


Figure 3: Data transmissions among methods executed on edges and the cloud.

3.2 Interview Study

To understand current concerns and perspectives on partitioning, we conducted four semi-structured interviews with experts in edge computing, including two Ph.D. candidates and two professors with backgrounds in edge computing and software systems. Each interview lasted approximately thirty minutes and covered a range of topics.

We started with a general question: *"What are your thoughts on dynamic partitioning?"* (**Q1**). We then focused on the tools and approaches they had previously applied, asking: *"Can you share any recent approach or tool you have used to solve partitioning challenges?"* (**Q2**). Finally, we explored the experts' specific professional concerns related to partitioning by asking: *"If tasked with performing dynamic partitioning, what issues would be most relevant from your professional perspective?"* (**Q3**). Detailed notes were taken during the interviews, which were also recorded and reviewed for comprehensive analysis.

3.3 Design Requirements

Q1 served as the primary question to uncover the fundamental views of the experts on dynamic partitioning. We summarized their insights as: *"Dynamic partitioning should seamlessly adapt to varying environments and workloads by dynamically selecting appropriate partition strategies between mobile devices and edge servers."* Using this insight as a guiding principle, we established the following design requirements based on the experts' feedback.

R1: Data Collection and Advanced Data Filtering. A recurring need among partitioning researchers is the ability to intuitively and effortlessly inspect user activity data—organized in flexible time sequences and easily accessible. One expert highlighted their dissatisfaction with using Android Profiler, noting that it required connecting the Android device to a computer via a data cable and navigating through numerous item-based records. This process was frustrating when searching for specific data, as locating an activity without remembering the exact time was challenging. This issue stems from Android Profiler's lack of a visual interface that can manage user activity data in a scalable, time-based manner.

R2: Model and Analyze Component. As the three interviewees emphasized, their research aimed to model the performances of every single component in different mobile devices with various hardware capacities. Based on the feedback from a researcher, he considered the times the components were called, the response time of the component, and how much energy they consumed between each start-end life circle, as the indicators analyze the performance of each component and judge whether they were healthy enough in their current platforms. Apart from analyzing the direct indicator mentioned in the previous requirement, the algorithmic complexities in the source code also matter to model every single component's overall performance emphasized by one of the interviewees.

R3: Retrieve Component Source Code. In addition to the direct indicators mentioned in prior requirements, the algorithmic complexities within the source code are essential for modeling each component's overall performance. One interviewee highlighted that both analyzing the structure of the function code and calculating the algorithm's complexity are crucial for estimating a component's efficiency. Moreover, the return values of components, as revealed in the source code, serve as key markers for constructing the call tree among components.

R4: Reveal Transition Patterns of Component State. User behaviors can vary significantly, leading to different interaction patterns with their mobile devices. Two interviewees emphasized that analyzing components in isolation overlooks the relationships between them, resulting in suboptimal performance for specific user activities. Additionally, one expert noted that understanding the relationships between all component states during user activity would enable more rational management of the distribution process.

R5: Provide Simulation of Component Sequence. While revealing component states is crucial for understanding user activity, effective partitioning involves a complex decision-making process for distributing components. For researchers, having insight into potential event sequences is essential for successful partitioning. One interviewee suggested that being able to preview transition sequences would allow them to predict possible user activity scenarios,

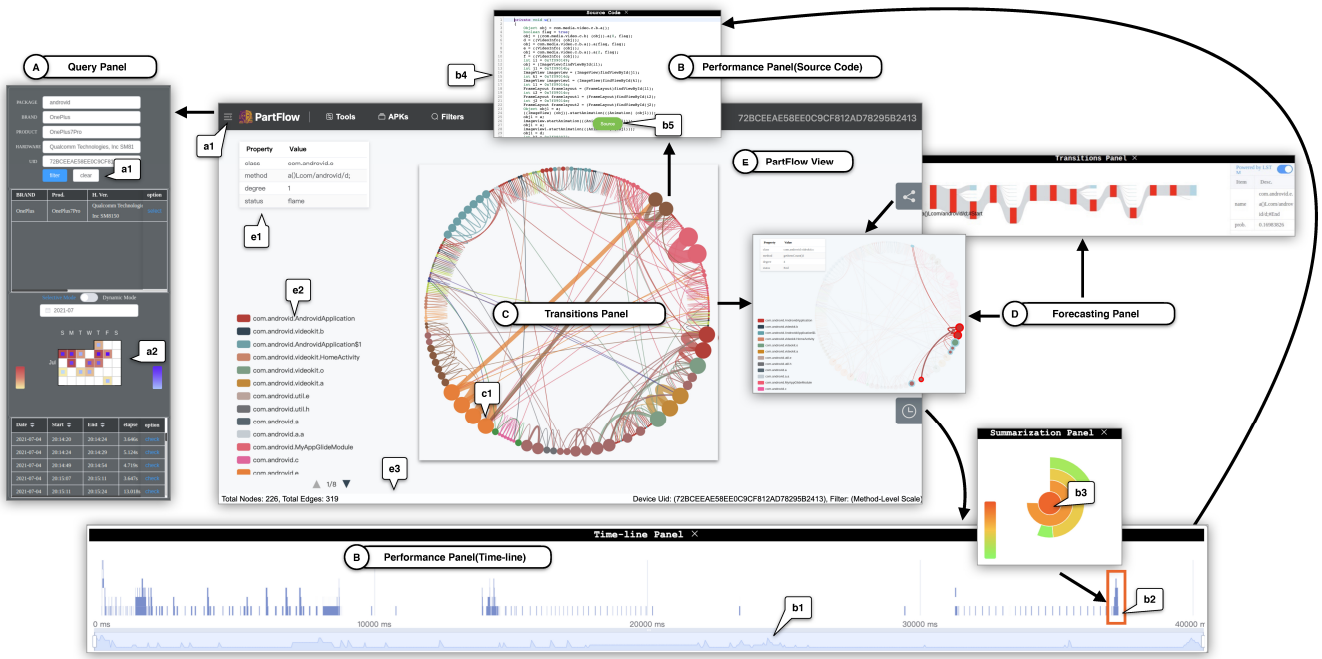


Figure 4: The PartFlow interface. (A) The Query Panel for retrieving user activities. (B) The Performance Panel for evaluating the performance of components. (C) The Transition Panel for indicating component transitions is based on the stochastic matrix. (D) The Forecasting Panel for selecting the high-probability sequence with the interactive Sankey diagram and LSTM model forecasting results.

facilitating a more informed partitioning strategy. Incorporating a simulation sequence for each component would provide researchers with a clearer roadmap for partitioning.

4 PartFlow System

In this section, we illustrate PartFlow as a visual analytics system integrating a series of analytical methods. Guided by the aforementioned design requirements, PartFlow incorporates a variety of analytical methods tailored to data generated by individual user devices. Users can access PartFlow via the GitHub repository[4] for further exploration and commitments.

4.1 PartFlow Overview

To meet the interviewees' requirements for monitoring the component state across multiple devices (**R1**), PartFlow incorporates an instrumentation function alongside visualization. Figure 5 illustrates the PartFlow architecture, showing the interplay between front-end visualization, back-end analytical processes, data storage, and external user devices.

Users can upload their target applications and download processed versions from the server storage. By launching and using these processed applications on external devices, method invocation data is transmitted and stored in a MongoDB database. The back-end conducts three main analytical processes using the stored component data: 1) performance estimation, 2) stochastic matrix calculation, and 3) series forecasting. These analyses support four main visualization panels: 1) the Query Panel (B), 2) the Performance Panel (C), 3) the Transition Panel (D), and 4) the Forecasting Panel (E). Each panel serves to visualize method execution patterns as needed, with their sub-panels depicted in Figure 4.

4.2 Inspecting User Activities

Recording activity data remotely and simultaneously across multiple devices and users is essential for researchers collecting data (**R1**).

[4]https://github.com/liminghao0914/PartFlow

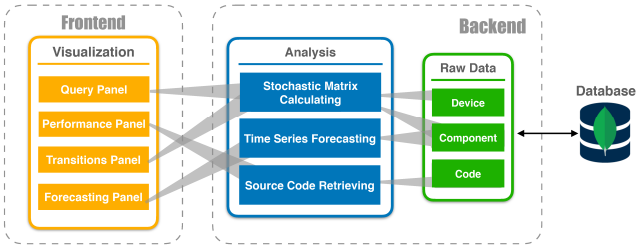


Figure 5: PartFlow consists of three major parts, frontend, backend, and database. Supported by three analytical modules, PartFlow jointly visualizes four panels.

To facilitate this, we implemented a heat map calendar that allows users to access activity records at any time, enabling them to answer questions such as *"How many times was the application launched?"* and *"What is the volume of component invocation data?"*.

The Query Panel, shown in Figure 4(A), extends on the left side (a1). The goal of this user activity map is to display overall user activity over a month. Users can launch instrumented applications multiple times over a set period on various devices (b1). They can then filter the data by month and select specific dates using the calendar heat map (b2) to view individual activities. Drawing on the design in [2] and T-Cal [14], each day cell is divided into two nested squares that represent two key metrics: the number of times an application is launched (in red) and the volume of method call log data (in purple). For instance, Figure 6 shows that on July 4th, a user's device launched an application many times, generating significant call log data. In contrast, on July 9th, the same device produced an equivalent volume of call log data but with fewer application launches, suggesting a more complex task was performed that day. This dual-metric approach enables users to filter daily activities efficiently and identify the data they wish to explore further.

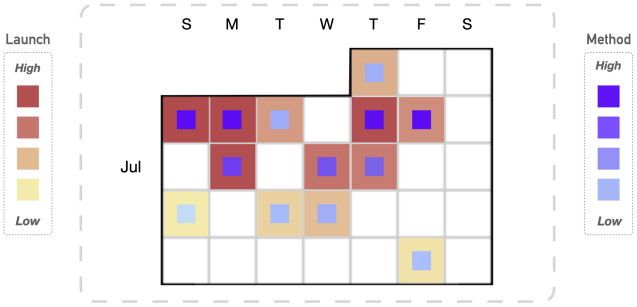


Figure 6: Query Panel displaying user activity for an application. Red indicates the number of launches, and blue indicates the method trace volume.

4.3 Evaluating Component Performance

Evaluating component performance involves two key steps. First, modeling the component structure based on its complexity, which relies on analyzing the source code (**R3**). Second, recording the execution schedule of components and calculating their execution time using entry and exit points (**R2**). This evaluation answers questions like *"What is the execution time of the component?"*, *"What is the component's source code?"*, and *"What does the timeline of a launch period look like?"*.

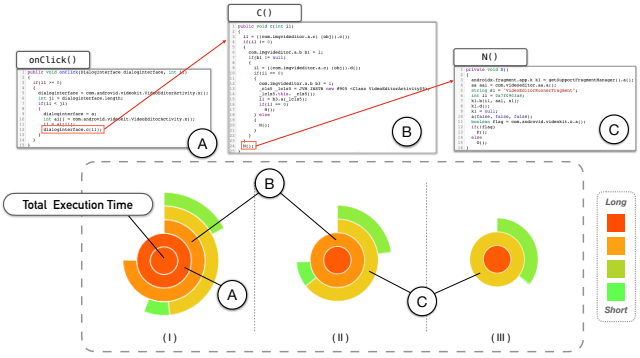


Figure 7: Each diagram presents nested method-level components. In detail, (a()) calls (b()), and (b()) calls (c()). The sunburst diagrams indicate their nested relations.

The Performance Panel, illustrated in Figure 4(B), comprises three main parts: the Source Code Panel, the Timeline Panel, and the Tree Panel. By clicking on a node (c1) or a rectangle (b2) in the Timeline Panel, users can activate the Source Code Panel (b4) to view the source code snippet of the selected component. For examining the entire class structure, users can access the complete Java code by clicking the button (b5).

We use a flame graph in Figure 4(B) to visualize the method execution timeline and a sunburst graph to represent the internal structure and execution time of a particular component. Clicking on rectangle (b2) in the Timeline Panel displays the Tree Panel (b3) above the rectangle. The sunburst diagram is interactive: the selected method is depicted as a central circle, surrounded by layered annulus sectors representing different levels of child methods within the parent method. The angle of each annulus sector corresponds to the proportion of the child method's execution time relative to its parent. The color gradient from green to red indicates short to long execution times. For instance, in Figure 7, the central round in red, encircled by annulus (A), represents the execution time of a parent method `onClick()`, with red indicating a duration exceeding 200ms. The first annulus sector (B) around annulus (A) depicts a first-level child method, `c()`. Clicking on annulus sector (B) expands it to 360 degrees, focusing on the child method `c()` and hiding annulus (A). As shown in Figure 7(b), method `N()` accounts for the major portion of execution time within `c()`. Users can then examine the inner structure and execution time of the third-level child method `N()` by selecting it, as depicted in Figure 7(c). Overall, this sunburst diagram enables users to explore the internal call tree of each component, aiding in the identification of components requiring optimization.

4.4 Summarizing Component Transition

Understanding transition patterns between component states is essential for researchers aiming to synthesize call sequences for targeted dynamic partitioning (**R4**). The Transition Panel addresses questions such as *"How frequently is this component called?"*, *"What are the preceding and subsequent steps of this component state?"*, and *"What is the probability of this transition occurring?"*

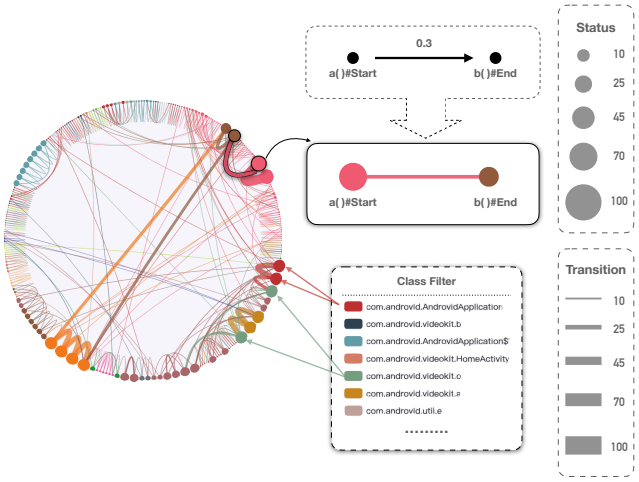


Figure 8: The visualization of transition patterns of one certain call log set. The size of the node implies the call frequency, while the width of the edge shows the transition probabilities.

PartFlow's Transition Panel visualizes these transition patterns using a digraph based on a stochastic matrix, as shown in Figure 4(D). Each node in the digraph represents a component state, such as when method `a()` begins execution (recorded as `a()#Start`), which creates a corresponding node in Figure 8. The node color indicates its class, allowing users to filter nodes by interacting with class legends. Node size reflects the frequency of the corresponding component state, signifying how often that state occurs. Edges illustrate transitions between component states, with edge width proportional to the transition frequency, meaning wider edges represent higher transition probabilities. Unlike traditional Markov Chain graphs with arrows, edge colors in PartFlow indicate the direction of transitions, maintaining visual clarity. For example, the edge connecting `a()#Start` to `b()#End` shares the color of `a()#Start`. This circular digraph serves a similar purpose to a Markov Chain but presents the information in a more streamlined and visually efficient format.

4.5 Forecasting Sequence of Component States

To optimize the overall functionality and services of an application, it is often essential to analyze a sequence of components to achieve a comprehensive and integrated optimization procedure (**R5**). These sequences frequently exhibit patterns that can be used to forecast the next executing method and predict future behavior in mobile applications.

The Forecasting Panel in PartFlow is designed to help answer critical questions such as *Is this sequence of components likely to occur?"* and *What is the next stage in this execution sequence?"*

This panel leverages historical data and sequential patterns to assist researchers in understanding and predicting component transitions and their likelihoods, thereby facilitating better-informed decision-making in application partitioning and workload optimization.

4.5.1 Markov-based Forecasting Visualization

Inspired by Guo *et al.* [21], we use a state transition graph based on Sankey diagrams to visualize component sequences Figure 9. Selecting a node generates a red-highlighted starting point in the Forecasting Panel, with adjacent light blue nodes representing potential next states.

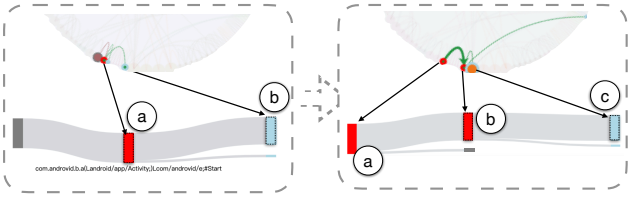


Figure 9: Two visual successive states of the Forecasting Panel.

As the component state sequence is time-discrete, we assume this event sequence in Figure 10 as

$$S = \{X_1, X_2, X_3, \ldots, X_n\}$$

Assuming that the value of the start node is $\Pr(X_1 = x_1) = 1$, when x_1 occurs, users are supposed to obtain the probability of flowing to each component state. When x_j occurs, the value of node j would be equal to the probability of $S' = \{x_1, x_2, \ldots, x_j\}$ occurring given the occurrence of $X_1 = x_1$ would be

$$\begin{aligned}\Pr(S'|X_1 = x_1) = &\Pr(X_j = x_j|X_{j-1} = x_{j-1}, \ldots, X_1 = x_1)\cdot \\ &\Pr(X_{j-1} = x_{j-1}|X_{j-2} = x_{j-2}, \ldots, X_1 = x_1)\cdot \\ &\ldots \cdot \Pr(X_2 = x_2|X_1 = x_1)\end{aligned}$$

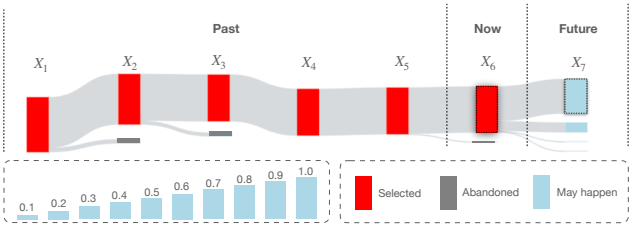


Figure 10: A Sankey-based decision tree for component state forecasting indicates its joint probability.

Node size in the Sankey diagram represents the probability of the sequence. For example, the starting node X_1 has a value of 1, while subsequent nodes decrease in size to reflect diminishing probability as the sequence progresses from left to right, creating a flow-like visualization.

4.5.2 DL-based Sequential State Prediction

Markov models provide reasonable prediction results due to their simplicity and interpretability. However, they assume that future states depend only on the current state, ignoring the potential impact of past states. This assumption can lead to inaccuracies, especially when dealing with errors or missing data, as Markov models will base predictions on prior potentially incorrect states.

To address these limitations, we introduce a DL-based model for sequential state prediction. Specifically, we implement a type of recurrent neural network (RNN) called long short-term memory (LSTM) as the prediction model. LSTMs are well-suited for handling data with temporal dynamics and can manage missing data, making them highly effective for time-series forecasting [29]. Prior to training, the raw sequential data are encoded into a one-hot format. Given that each component has two states (start and end), the input data will contain $2 \times n_{method}$ features. Users can define their preferred input sequence length, with n_{input} set to 10 by default. The LSTM-based model architecture consists of a 120-unit LSTM layer as the first hidden layer, followed by a densely connected output layer shaped as $\langle 2 \times n_{method} \rangle$. The output layer applies a softmax function to predict the most likely next method to be executed. A detailed description of the LSTM model architecture and its source code is available on the GitHub page[5].

4.5.3 Component State Forecasting Evaluation

Multi-step forecasting differs from single-step forecasting, where the latter predicts only the next value in a sequence [15]. In multi-step forecasting, a model is trained on historical data and utilized to make predictions over several future steps. This approach allows the model to account for patterns and trends in past data to forecast upcoming values. To assess the performance of the LSTM model for multi-step forecasting, we compared it with four baseline models: a basic Markov Model (MM), support vector machine (SVM), naive Bayes (NB), and logistic regression (LR). We used the dataset[6] collected in [25] for this evaluation.

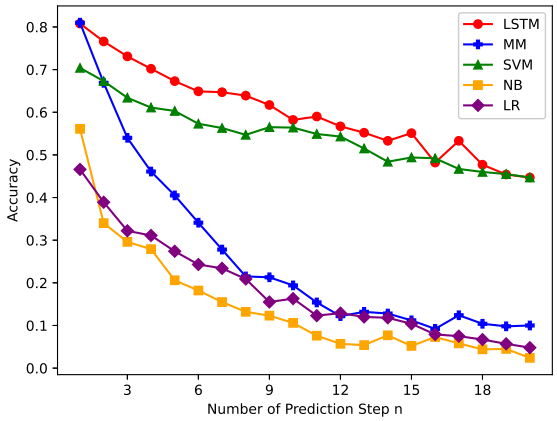


Figure 11: Prediction results of LSTM, MM, SVM, NB, and LR across different prediction steps n.

The results, shown in Figure 11, indicate that the LSTM model performs reliably in multi-step forecasting of component states. Here, the variable n represents the number of prediction steps ahead in the forecasting process. When $n = 1$, the MM and LSTM models achieve similar high accuracy, over 80%. However, as n increases, MM accuracy drops significantly. In contrast, SVM shows a better performance trend for increasing n, though it starts with relatively low accuracy at $n = 1$. NB and LR, on the other hand, show low accuracy for $n = 1$, similar to MM's performance as n increases beyond 9, due to their limitations in handling complex state inputs.

Based on these findings, we suggest users consider both the transition probabilities and the LSTM model predictions when arranging component sequences for improved user experience.

[5]https://github.com/liminghao0914/PartFlow/tree/master/models
[6]https://github.com/liminghao0914/user-component-dataset

5 Case Studies

In this section, we demonstrate the usefulness and effectiveness of PartFlow from two aspects, data provision for algorithms and insights on evaluation and optimization for engineers and researchers with a real-world application.

5.1 Data Provision

Here, we demonstrate the utility of PartFlow in providing data for partitioning and scheduling algorithms. We examined two research studies, one by Gao *et al.* [16] and another by Cai *et al.* [5], which introduced dynamic partitioning algorithms within edge-cloud environments. Our objective was to test whether PartFlow could ensure data provision for the parameters outlined in these algorithms. Table 1 illustrates how PartFlow meets the data requirements for parameters in both modeling processes by facilitating the acquisition of real-time data. The horizontal line in Table 1 separates variables for application methods (upper section) from those for component-based resource optimization (lower section).

Table 1: Data Provision Checklist

Notation	Provision	Acquisition Process
$\mathbb{M}$	✓	a2, a4 and a3 in A
M_n	✓	E
T_n	✓	c1 in C
$\widetilde{M}_n$	✓	E
$\mathbf{T}$	✓	D
$H_{ij}(t)$	✓	c1 and c2 in C
C_L^j, C_R^j	✗	-
$E_{M_n}(t)$	✓	c3 in C
$C_{M_n}(t)$	✓	c3 in C
r_i	✓	c4 and c5 in C
s_i	✓	c4 and c5 in C
f_i	✓	D
p_i	✓	D
$o_{i,j}$	✓	c4 and c5 in C, D and E
$f_{i,j}$	✓	c4 and c5 in C, D and E

Markovian factors in workload offloading. Several variables in the model [16] are illustrated in the upper part of Table 1. The set of application methods being executed $\mathbb{M}$ can be extracted from e1, e2, and e3 in Figure 4 (E). For retrieving the invoked time T_n and its Local Invocation Graph (LIG) of individual method M_n in the set, researchers can get the execution timeline of methods and the LIG from c1 in the Performance Panel (B) and the Transition Panel (C), respectively. Based on invoked time T_n, we can calculate the modeling $E_{M_n}(t)$ and $C_{M_n}(t)$ of energy consumption of the local execution and transmissions of M_n. $H_{ij}(t)$ refers to the time-dependent data transmission overhead between components i and j, retrievable from c1 and c2 in the Transition Panel (C). C_L^j, C_R^j represent the computational resources for local execution and remote transmission of component i, respectively, shown in the Transition Panel (C). Then, from the Forecasting Panel (D) we can arrange method M_n to have a composite Markovian state $\tilde{M}_n$. Further, without loss of generality, through the Transition Panel (C) and the Forecasting Panel (D), researchers obtain the state transition probability $p_{\tilde{M}_n,j}$, which indicates the possibility for M_n to invoke every other application method $m_j \in \mathbb{M}$. Through the above process, the amount (G_{M_n}) of energy saving is computed, which depends the offloading strategy of M_n. Thus, at time T_n when M_n is about to be invoked by M_{n-1}, G_{M_n} is computed by aforementioned variables, which all can be obtained with PartFlow.

Partition algorithm in Cognitive Resource Optimization. In the cognitive cloud gaming platform, games consist of interdependent components that work collaboratively to provide gaming services for players. The partitioning algorithm for the game needs several variables that are shown in the lower part of Table 1. By utilizing c4 and c5 in Performance Panel (B), researchers can construct the resource consumption and the size of compiled code of component i, r_i and s_i, through its static code. Through the shape of edges and nodes in Transition Panel (C), the execution frequency f_i and probability p_i of component i are able to be retrieved. As for the frequency $f_{i,j}$ and probability $o_{i,j}$ that component i sends data to component j, it can be calculated via Forecasting Panel (D). PartFlow helps researchers derive the minimum cost for different graphs, including the minimum spanning tree (MST), complete graph, and general graph, where the optimal cut achieves the general goal of a partition for cognitive resource optimization.

5.2 Insights on Evaluation and Optimization of Multimedia Application

Now we illustrate how PartFlow can be used in the overall evaluation and optimization of a multimedia application. We recruited a software engineer (SE) who had five years of working experience from a game lab at a university. We conducted preliminary instructions with the SE, lasting about half an hour. During the instruction, we first introduced the purpose and function of each interface and panel provided by PartFlow. When the instruction was finished, the SE was asked to employ PartFlow as the tool to investigate strategy-making insights during partitioning and offloading optimization. The SE chose a mobile device-based prototype of the digital twin [11]. With the original application he provided, we recompiled it with PartFlow to insert the data acquisition functions. After confirming the normal functioning of the recompiled application, the SE was informed to start the exploration. At the beginning of the exploration, he asked a volunteer student, who used a *OnePlus 7 Pro*, an Android device, to try out the digital campus for one day. After that, PartFlow visualized the collected data and presented the comprehensive results we intended to draw for the SE. During the case study, we discussed with the SE the insights gained during the exploration of PartFlow and took notes when necessary and recorded the entire session for later analysis.

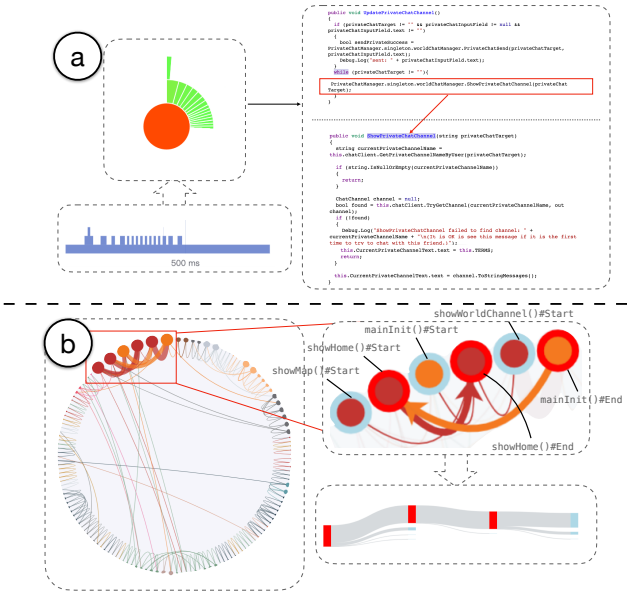


Figure 12: The case studies of PartFlow: (a) Estimating the Performance of the Component and (b) Selecting the Objective Sequence of Components.

According to our record, the first action he took was to check

the Timeline Panel and the Tree Panel shown in Figure 12(a), where he noticed that the component of `UpdatePrivateChatChannel()` lasted a relatively long time. Then he opened the Source Code Panel to inspect the realization of this component. He found that the issue was caused by an inefficient traversal algorithm.

Inspired by the former insight, the SE switched his attention to the transitions of component states revealed in the Transition Panel shown in Figure 12(b). He selected the node of `mainInit#Start` as it was the entrance of the digital twin campus and he wanted to know what users would do after this operation. After that, the SE interacted with the Forecasting Interface to seek answers to his question. When he selected the node of `mainInit#End`, which was executed 32 times according to the record, as the start node of the prediction flow, the interface illustrated the probability distribution of the possible components as the next node. The forecasting result showed that there existed two high-probability states as the second node, which were `showHome()#Start` and `showWorldChannel()#Start`. He continued the forecasting procedure by assigning the `showHome()#Start` component as the new start node. The new forecasting result showed only two components that were possibly being executed in the next node. The top high-probability states were the `showHome()#End`, which was quite obvious. The SE then repeated the forecasting procedure until he gained enough details. Finally, he maintained three forecasting flows from the node of `showWorldChannel()#Start`, the node of `showMap()#Start` and the node of `mainInit()#Start`, which was the most possible to be the next state. Figure 12(b) showed a part of the process. With the forecasting information provided by the flows, the SE obtained the next optimization direction for the application.

6 User Study

We conducted a user study to evaluate the unique features of PartFlow. The primary questions guiding this study were: (**S1**) What impact does PartFlow have on participants' strategy-making? (**S2**) What are participants' attitudes towards the features of PartFlow? (**S3**) How do participants interact with and utilize these features?

6.1 Participants Recruitment and Apparatus

We recruited twelve participants (5 females and 7 males, aged 22-40) via mailing lists and social media, including 9 graduate students and 3 professors with backgrounds in edge computing. The study was conducted using a 27-inch AOC monitor (3920×2180 resolution) connected to a Dell PC running Windows 10 OS, equipped with a mouse and keyboard. PartFlow was accessed via Google Chrome in full-screen mode.

6.2 Procedure

Initially, each participant was provided with a hands-on tutorial on the PartFlow interface. An administrator demonstrated the system's features and interactions in detail, allowing participants to ask questions and explore the tool until they felt comfortable proceeding with the study. This step ensured that all participants began the trial with a similar level of understanding.

Next, we uploaded data from a commercial video editing Android application, *androvid*[7], for experimental use. Participants announced the start of their trial so that the administrator could observe their interactions. Upon completing the trial, participants were asked to share their experience, rate the effectiveness of PartFlow's functions (**S1**), assess the usefulness of its features (**S2**), and provide feedback on the overall quality of the tool (**S3**) to the administrator.

[7]https://play.google.com/store/apps/details?id=com.androvid

6.3 Results and Analysis

6.3.1 Questionnaire Ratings

To answer **S1**, **S2** and **S3**, we conducted a questionnaire towards participants' about PartFlow and calculated its ratings (Figure 13).

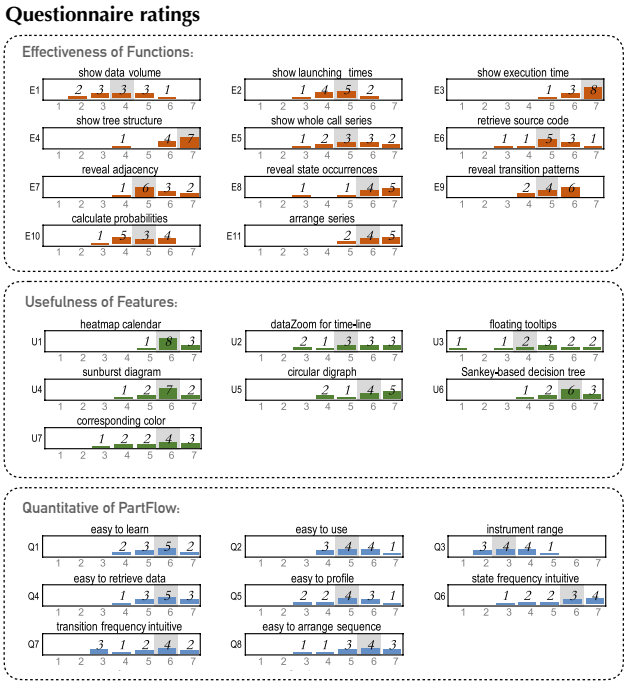


Figure 13: Participants' ratings on various system aspects, including the effectiveness of function, usefulness of features, and qualitative of PartFlow.

Effectiveness of Functions. Figure 13 illustrates the ratings of the effectiveness of each function in PartFlow. From the results, it is evident that participants expressed a clear preference for functions related to component performance (E3-E6), method relationships (E7-E9), and forecasting (E10, E11). Notably, the Tree Panel (E3, E4) received particularly high effectiveness ratings from participants, indicating that this feature was well-received and valuable for their tasks. On the other hand, the feature for suggesting data volume (E1) was rated as less effective in enhancing the users' efficiency for making partitioning strategies. Despite the varied levels of attention and feedback on individual functions, participants unanimously expressed overall satisfaction with PartFlow as a tool for facilitating partitioning and offloading strategies.

Usefulness of Features. Figure 13 highlights participants' feedback on PartFlow's features. The bi-colored heatmap calendar was praised for its clarity in displaying user activity data (U1). The sunburst-based performance tree and Sankey-based decision tree were highly valued for analyzing component structure and visualizing execution paths, respectively (U4, U6). Other features, such as the flame graph and interactive digraph (U2, U5, U7), were useful but rated slightly lower. Floating tooltips received moderate ratings, viewed more as supplementary for quick information checks than for in-depth analysis. Overall, PartFlow's features were seen as collectively beneficial for analysis and decision-making.

Quality of PartFlow. As the listed results of the quality in Figure 13 from the questionnaire, it was rated as easy to learn (Q1), but not as easy as it to use (Q2). As the entrance to the application partitioning, most of the participants admitted PartFlow could not monitor such comprehensive components as they expected (Q3). After accessing the data, easy-to-retrieve data intuitively became

the consensus for most of the participants (Q4). For visualizing transition and occurrence (Q6-Q8), many participants acknowledged PartFlow as a useful tool to indicate series patterns.

6.3.2 Interview Results

To evaluate PartFlow, we gathered several participant comments regarding their experiences with the tool.

Feedback on Query Panel. Participants praised the Query Panel for allowing them to retrieve data conveniently. One participant noted, *"Usually, I forget which one is the objective experiment data. In that case, I have to inspect data by memory. Fortunately, with the help of PartFlow, I could see a beautiful heatmap calendar indicating the volume of data, which gave great help to me."* While some participants mentioned that it might not always be necessary to retrieve data through a heatmap since they could remember specific details, they acknowledged the importance of being able to *"reveal the overall situation of user activities within a day."*

Feedback on Performance Panel. The Performance Panel was praised for evaluating component performance across three dimensions. Participants valued knowing exact execution times, with one noting, *"PartFlow highlights complex methods by coloring sectors red."* The panel clarified component structure, as another participant said, *"It's like the flame graph but more specific and interactive. By clicking, I could focus on the objective component."* Additionally, the source code panel enabled time complexity calculations, helping participants evaluate computing complexity comprehensively. One participant remarked, *"While I could calculate complexity in Visual Studio Code, it was difficult to inspect execution and code structure simultaneously."*

Feedback on Transition Panel. The Transition Panel was seen as a useful aid for dynamic partitioning in two primary ways. First, it helped participants understand the next and previous steps of the current component state. One participant noted, *"This tool reveals multi-dimensional information about functions in an intuitive way—I can see the numerous edges of varying widths across the circular digraph."* Additionally, the panel highlighted the most frequently executed components, with one participant mentioning, *"A large node caught my attention, making me aware of this frequently used component with relatively high resource consumption."*

Feedback on Forecasting Panel. Participants expressed strong approval for the Forecasting Panel, which allowed them to explore dynamic partitioning strategies by easily simulating possible series of components that could be offloaded or optimized. One participant noted, *"Choosing several objective components and their possible adjacencies instead of inspecting stack timelines in Android Profiler was incredibly easy to use."* The panel also helped participants maintain objective sequences with a high probability of occurrence while making selections. One participant said, *"The river-like decision tree made me naturally pay more attention to the size of its downstream and avoid making it too thin."*

7 Discussion

In this section, we reflect on the insights gained from our study and outline the limitations and potential future directions for enhancing the PartFlow system.

7.1 Value of PartFlow in Mobile Edge Computing

Data Acquisition. PartFlow has demonstrated significant value as an entry point for rigorous scientific analysis by providing a remote approach for collecting data across multiple users and devices. This capability simplifies the process and reduces the cost of analyzing user activities within the targeted applications. Participants in the user study expressed satisfaction with the ease of data access, which supports both model analysis and statistical prediction of method-level components.

Data Handling and Management. PartFlow offers a robust, visualized framework for handling and managing data, allowing participants to locate and extract relevant data from extensive component datasets efficiently. By filtering data based on devices and application packages, users can model components through statistical data that reflect their execution on various heterogeneous resources. Furthermore, applying machine learning techniques enables data aggregation and training to model the interactions between components and their operational environments.

Dimensional Analysis. The multi-dimensional analysis capabilities of PartFlow fulfilled the requirements of most study participants. Users praised the system's innovative visual representations that facilitated performance evaluation of components. These visualizations empowered users to conduct comprehensive dimensional analysis of user-driven method relationships, helping them identify optimization solutions relevant to their fields of expertise, such as cloud gaming, artificial intelligence, and multimedia applications.

7.2 Limitations and Future Directions

One limitation of PartFlow is the restricted number of available components for inspection. The current tools for manipulation and decomposition focus on Java byte-code, which cannot handle native methods within APKs. In many cases, particularly for applications developed by large companies, essential methods run as native methods in Shared Object (so) files. PartFlow is unable to monitor or retrieve data from these native method components. Additionally, PartFlow faces challenges in forecasting due to the complexity of component interactions. When components call each other across multiple steps, the sequence of component states loses the memoryless property, rendering the stochastic matrix insufficient for accurate forecasting. Another limitation is related to the use of hues for indicating class and edge directions in the legends, as shown in Figure 4(e2). Human perception struggles with accurately distinguishing colors as the number of categories increases, which limits the volume of component data that can be effectively visualized in the system.

From a software engineering perspective, several enhancements can be made. First, we need to guide users in applying their own or open-source Android applications on a larger scale. Secondly, incorporating an LSTM-based deep neural network into PartFlow for sequential forecasting would improve prediction accuracy. Finally, to complement the use of colors for distinguishing classes, we plan to implement additional differentiation measures, such as clustering, to better represent components within the same class.

8 Conclusion

We have introduced PartFlow, an interactive visualization system designed to help researchers dynamically analyze user-driven execution patterns for application partitioning and workload offloading. PartFlow displays multi-dimensional data on application component states gathered from multiple users and devices, effectively revealing user activities, component performance metrics, and transition patterns. Additionally, leveraging a stochastic matrix, PartFlow offers an innovative time-series forecasting feature that simulates the execution flow of specific components.

Through engagement with expert users, we identified key design requirements and iteratively refined PartFlow through a human-centered design process. To assess its effectiveness and usability, we conducted two case studies and a user study involving realistic user-driven data from a single application. Feedback from the evaluation phase was overwhelmingly positive, with users highly rating PartFlow and emphasizing the value and functionality of its features.

REFERENCES

[1] K. Ali and O. Lhoták. Application-only call graph construction. In *European Conference on Object-Oriented Programming*, pp. 688–712. Springer, 2012.

[2] B. Alper, B. Bach, N. Henry Riche, T. Isenberg, and J.-D. Fekete. Weighted graph comparison techniques for brain connectivity analysis. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 483–492, 2013.

[3] R. AlTarawneh and S. R. Humayoun. Visualizing software structures through enhanced interactive sunburst layout. In P. Buono, R. Lanzilotti, M. Matera, and M. F. Costabile, eds., *Proceedings of the International Working Conference on Advanced Visual Interfaces, AVI 2016, Bari, Italy, June 7-10, 2016*, pp. 288–289. ACM, 2016. doi: 10.1145/2909132.2926066

[4] G. Bonneau, H. Hege, C. R. Johnson, M. M. Oliveira, K. Potter, P. Rheingans, and T. Schultz. Overview and state-of-the-art of uncertainty visualization. In C. D. Hansen, M. Chen, C. R. Johnson, A. E. Kaufman, and H. Hagen, eds., *Scientific Visualization*, Mathematics and Visualization, pp. 3–27. Springer, 2014. doi: 10.1007/978-1-4471-6497-5_1

[5] W. Cai, H. C. Chan, X. Wang, and V. C. Leung. Cognitive resource optimization for the decomposed cloud gaming platform. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(12):2038–2051, 2015.

[6] W. Cai, Y. Chi, C. Zhou, C. Zhu, and V. C. Leung. Ubcgaming: Ubiquitous cloud gaming system. *IEEE Systems Journal*, 12(3):2483–2494, 2018.

[7] A. Cedilnik and P. Rheingans. Procedural annotation of uncertain information. In *11th IEEE Visualization Conference, IEEE Vis 2000, Salt Lake City, UT, USA, October 8-13, 2000, Proceedings*, pp. 77–83. IEEE Computer Society and ACM, 2000. doi: 10.1109/VISUAL.2000.885679

[8] X. Chen, L. Jiao, W. Li, and X. Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM transactions on networking*, 24(5):2795–2808, 2015.

[9] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pp. 301–314, 2011.

[10] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: making smartphones last longer with code offload. In S. Banerjee, S. Keshav, and A. Wolman, eds., *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys 2010), San Francisco, California, USA, June 15-18, 2010*, pp. 49–62. ACM, 2010.

[11] H. Duan, J. Li, S. Fan, Z. Lin, X. Wu, and W. Cai. Metaverse for social good: A university campus prototype. In H. T. Shen, Y. Zhuang, J. R. Smith, Y. Yang, P. Cesar, F. Metze, and B. Prabhakaran, eds., *MM '21: ACM Multimedia Conference, Virtual Event, China, October 20 - 24, 2021*, pp. 153–161. ACM, 2021. doi: 10.1145/3474085.3479238

[12] W. Enck, P. Gilbert, B. Chun, L. P. Cox, J. Jung, P. D. McDaniel, and A. Sheth. Taintdroid: an information flow tracking system for real-time privacy monitoring on smartphones. *Commun. ACM*, 57(3):99–106, 2014. doi: 10.1145/2494522

[13] N. Fernando, S. W. Loke, and W. Rahayu. Mobile cloud computing: A survey. *Future generation computer systems*, 29(1):84–106, 2013.

[14] S. Fu, J. Zhao, H. F. Cheng, H. Zhu, and J. Marlow. T-cal: Understanding team conversational data with calendar-based visualization. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pp. 1–13, 2018.

[15] A. Galicia, R. Talavera-Llames, A. Troncoso, I. Koprinska, and F. Martínez-Álvarez. Multi-step forecasting for big data time series based on ensemble learning. *Knowledge-Based Systems*, 163:830–841, 2019.

[16] W. Gao, Y. Li, H. Lu, T. Wang, and C. Liu. On exploiting dynamic execution patterns for workload offloading in mobile cloud applications. In *2014 IEEE 22nd international conference on network protocols*, pp. 1–12. IEEE, 2014.

[17] D. Gotz and H. Stavropoulos. Decisionflow: Visual analytics for high-dimensional temporal event sequence data. *IEEE Trans. Vis. Comput. Graph.*, 20(12):1783–1792, 2014. doi: 10.1109/TVCG.2014.2346682

[18] B. Gregg. Visualizing system latency: Heat maps are a unique and powerful way to visualize latency data. explaining the results, however, is an ongoing challenge. *Queue*, 8(5):30–42, 2010.

[19] B. Gregg. The flame graph. *Commun. ACM*, 59(6):48–57, 2016. doi: 10.1145/2909476

[20] S. Guo, F. Du, S. Malik, E. Koh, S. Kim, Z. Liu, D. Kim, H. Zha, and N. Cao. Visualizing uncertainty and alternatives in event sequence predictions. In S. A. Brewster, G. Fitzpatrick, A. L. Cox, and V. Kostakos, eds., *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, p. 573. ACM, 2019. doi: 10.1145/3290605.3300803

[21] S. Guo, F. Du, S. Malik, E. Koh, S. Kim, Z. Liu, D. Kim, H. Zha, and N. Cao. Visualizing uncertainty and alternatives in event sequence predictions. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–12, 2019.

[22] B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel computing*, 26(12):1519–1534, 2000.

[23] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *2012 Proceedings IEEE INFOCOM*, pp. 945–953, 2012. doi: 10.1109/INFCOM.2012.6195845

[24] A. Kumpf, B. Tost, M. Baumgart, M. Riemer, R. Westermann, and M. Rautenhaus. Visualizing confidence in cluster-based ensemble weather forecast analyses. *IEEE Trans. Vis. Comput. Graph.*, 24(1):109–119, 2018. doi: 10.1109/TVCG.2017.2745178

[25] M. Li and W. Cai. A blockchain-based profiling system for exploring human factors in cloud-edge-end orchestration. In *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 13–18. IEEE, 2022.

[26] A. Perer and D. Gotz. Data-driven exploration of care plans for patients. In W. E. Mackay, S. A. Brewster, and S. Bødker, eds., *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013, Extended Abstracts*, pp. 439–444. ACM, 2013. doi: 10.1145/2468356.2468434

[27] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3g networks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 137–150, 2010.

[28] P. J. Rhodes, R. S. Laramee, R. D. Bergeron, and T. M. Sparr. Uncertainty visualization methods in isosurface rendering. In M. Chover, H. Hagen, and D. Tost, eds., *24th Annual Conference of the European Association for Computer Graphics, Eurographics 2003 - Short Presentations, Granada, Spain, September 1-5, 2003*. Eurographics Association, 2003. doi: 10.2312/egs.20031054

[29] X. Tang, H. Yao, Y. Sun, C. Aggarwal, P. Mitra, and S. Wang. Joint modeling of local and global temporal dynamics for multivariate time series forecasting with missing values. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 5956–5963, 2020.

[30] L. Tong and W. Gao. Application-aware traffic scheduling for workload offloading in mobile clouds. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9. IEEE, 2016.

[31] K. Wongsuphasawat and D. Gotz. Exploring flow, factors, and outcomes of temporal event sequences with the outflow visualization. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2659–2668, 2012. doi: 10.1109/TVCG.2012.225

[32] L. Xiang, S. Ye, Y. Feng, B. Li, and B. Li. Ready, set, go: Coalesced offloading from mobile devices to the cloud. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 2373–2381. IEEE, 2014.

[33] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs. Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mob. Networks Appl.*, 16(3):270–284, 2011. doi: 10.1007/s11036-011-0305-7

[34] J. Zhao, Z. Liu, M. Dontcheva, A. Hertzmann, and A. Wilson. Matrixwave: Visual comparison of event sequence data. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pp. 259–268, 2015.