Workload Characterization of Commercial Mobile Benchmark Suites

Victor Kariofillis*, Natalie Enright Jerger†
University of Toronto
*viktor.karyofyllis@mail.utoronto.ca
†enright@ece.utoronto.ca

***Abstract*—Mobile devices are essential in our daily lives. The hardware and software of these devices differs from their desktop and server counterparts and they require special benchmarks to fairly and accurately evaluate them. However, the computer architecture community lacks studies analyzing the performance characteristics of mobile benchmarks. This paper presents an extensive workload characterization of commercial mobile benchmark suites using several hardware performance counters. We analyze the temporal behaviours of benchmarks. Our findings show the diverse load patterns across CPU core clusters. Our clustering analysis reveal a benchmark subset that offers a reduction in evaluation time close to 75%, while preserving the richness of benchmark coverage. This work contributes insights for refining mobile benchmarking methodologies, providing a valuable compass for researchers navigating the landscape of mobile system architecture assessments.**

***Index Terms*—Mobile benchmark suites, workload characterization, performance evaluation**

I. Introduction

Mobile devices, ranging from smartphones and tablets to smartwatches, are projected to exceed 18 billion units by 2025 [1]. In comparison, the total number of desktop, laptop and server computers worldwide is estimated at over 2 billion [2]. Despite this, a mere 1% of top computer architecture conference papers in 2018 delved into mobile computing, highlighting a research gap [3].

Mobile System-on-Chips (SoCs) are distinct from their desktop and server counterparts. They feature tight integration, significant heterogeneity [4] and rapid evolution. Accelerators in mobile SoCs have more than quadrupled in the last decade, while multiple new CPU designs are released each year [5]. Mobile operating systems (OS) evolve at a comparable pace to take advantage of new hardware capabilities. One challenge faced by architects studying mobile SoCs is the absence of benchmarks specifically designed for evaluating mobile hardware.

Benchmarks commonly used in the architectural community, like *SPEC CPU* [6] and *PARSEC* [7], often fall short in representing real-world mobile applications [8]. They lack the interactivity, heterogeneity, and reliance on shared libraries typical of mobile workloads [9], [10]. Academic benchmark suites targeting the mobile space are often narrowly focused on specific aspects or domains (e.g., web browsing [10], [11], augmented reality [12], race events [13], deep learning [14], etc.), thus limiting their utility. Furthermore, code changes introduced by new OS and API versions can render them incompatible [15].

Commercial benchmarks are commonly employed in industry [16]. However, challenges arise when incorporating them into academic settings. In contrast to desktop benchmark suites, these benchmarks lack prior characterization, impeding computer architects' understanding of their impact on system performance. Additionally, these benchmarks are not tailored for optimal use in system simulations, a common practice in hardware studies. Their lengthy execution times pose challenges for the efficient evaluation of new hardware designs. Closing this gap in mobile benchmark characterization is essential for fair evaluations of mobile platforms.

In this study, we conduct a comprehensive analysis of the execution and performance characteristics inherent in commercial benchmarks. Our objective is to provide researchers with in-depth insights into the behavioural patterns of these benchmarks and their effect on mobile hardware. Additionally, we offer perspectives on the overall similarities and differences among benchmark suites. This nuanced understanding empowers researchers to judiciously select benchmarks aligned with their specific requirements, thereby streamlining the evaluation process.

In summary, we make the following contributions:

- Analysis of the performance characteristics of widely used commercial mobile benchmark suites.
- Analysis of the temporal behaviours and heterogeneity exhibited by the workloads.
- Assessment of the similarity among individual benchmarks.
- Proposal of a reduced benchmark set, allowing researchers to reduce evaluation time by up to 75%.

II. Motivation

Commercial mobile benchmark suites provide insights into how users interact with their mobile phones, offering a realistic representation of real-world usage scenarios. These benchmarks, widely adopted by the industry to showcase the capabilities of new devices, remain up-to-date with the latest features integrated into mobile System-on-Chips (SoCs). As industry standards, they mirror evolving user behaviours and mobile hardware.

Existing benchmarks such as SPEC and PARSEC are designed for traditional computing environments. They fall short when evaluating the rapidly evolving and heterogeneous components of modern mobile SoCs. Beyond conventional CPUs and GPUs, these SoCs include digital signal processors (DSPs) for accelerated vector instructions, GPUs dedicated to

TABLE I: Commercial mobile benchmark suites analyzed.

Benchmark Suite	Benchmark Names	Targeted HW / Workload
3D Mark v2	Slingshot	GPU
	Slingshot Extreme	
	Wild Life	
	Wild Life Extreme	
Antutu v9	CPU	CPU
	GPU	GPU
	Mem	Memory subsystem
	UX	Everyday tasks (e.g., data/image processing, video decoding)
Aitutu v2	-	AI-related tasks
Geekbench 5	CPU	CPU
	Compute	GPU
Geekbench 6	CPU	CPU
	Compute	GPU
GFXBench v5	High Level	GPU (overall graphics performance)
	Low Level	GPU (specific graphics performance, e.g., tessellation)
	Stress Test	GPU (render quality performance)
PCMark	Storage 2.0	Storage subsystem
	Work 3.0	Everyday activities (e.g. browsing, video/photo editing)

tasks like media processing, and AI accelerators tailored for machine learning duties. The iPhone XS's A12 chip, for instance, integrates 42 accelerators [17] with the number steadily rising in later models [18]. The limitations of traditional benchmarks highlight the necessity of turning to commercial mobile benchmark suites. For researchers, understanding these benchmarks is crucial. There's presently no comprehensive workload characterization study on these benchmark; thus, emphasizing the need for one.

III. Benchmark Suites: Brief Overview

In our analysis, we include some of the most popular mobile benchmark suites [19]–[30] (Table I). In this section, we provide an overview of them.

3DMark Android [31], published by UL, is a benchmark suite that measures the CPU and GPU performance of mobile devices. It is composed of two benchmarks, *Wild Life* and *Sling Shot*. Both have *Extreme* versions with higher resolutions, resulting in a total of four sub-benchmarks. *Wild Life* runs for approximately one minute and measures a device's ability to provide high levels of performance for short periods of time. It mirrors mobile games that have short bursts of intense activity. *Sling Shot* tests a range of graphics API features, such as volumetric lighting (i.e., adding lighting effects to rendered scenes) and instanced rendering (i.e., rendering multiple instances of a model in a single draw call).

Antutu [32], published by Cheetah Mobile, is an all-around benchmark suite stressing various hardware components. It is composed of 4 parts; however, the user cannot execute those parts individually. *Antutu GPU* contains five GPU benchmarks. Three of them, *Swordsman*, *Refinery* and *Terracotta*, have mobile game-like high-end graphics. The other two, *Fisheye* and *Blur*, are simpler image processing tests. *Antutu CPU* contains mathematical operations (e.g., GEMM), common algorithms (e.g., PNG decoding) and multi-core tests. *Antutu Mem* stresses the RAM and the storage subsystem. *Antutu UX* includes data processing and data security workloads, image and video processing, as well as a scroll delay test. *Aitutu* is a standalone benchmark, from the creators of *Antutu*. It focuses on AI workloads like image classification and object detection.

Geekbench [33], published by Primate Labs, is one of the most popular mobile benchmark suites. It is designed to evaluate and compare the performance of devices across different platforms and operating systems (OSs). It is split into two components, one for testing the CPU and the other for the GPU. *Geekbench 5 CPU* contains three parts: integer, floating point (FP) and cryptography workloads. *Geekbench 6 CPU* is split into five subsections: productivity, developer, machine learning, image editing and image synthesis workloads. *Geekbench GPU Compute* benchmarks evaluate GPU performance. *Geekbench 5 Compute* contains 11 workloads, while *Geekbench 6 Compute* contains 8 workloads divided in four categories: Machine Learning, Image Editing, Image Synthesis and Simulation. Despite the different categorizations, there are multiple workloads that are shared between the two versions.

GFXBench [34], published by Kishonti, is a graphics-based benchmark suite aimed at testing the GPU. It is split into three categories: *High-Level*, *Low-Level* and *Special* tests. *High-Level* tests stress the GPU in a mobile game-like manner. There are four graphics scenes: *Aztec Ruins*, *Car Chase*, *Manhattan* and *T-Rex*. Each scene is executed with tweaked settings (e.g., resolution, API used) resulting in 19 separate benchmarks. The *Low-Level* category consists of 8 benchmarks. They measure specific performance aspects, like tessellation and texturing. *Special* tests measure visual fidelity.

PCMark Android [35], published by UL, is a benchmark suite used for measuring the performance and battery life of Android phones and tablets. It comprises of two benchmarks, *Work* and *Storage*. *Work* is composed of web browsing, data manipulation and video, document and photo editing workloads. *Storage* measures IO performance in internal and external storage, as well as database performance.

IV. Workload Characterization Methodology

In this section, we present our experimental setup as well as explain our approach in characterizing existing commercial mobile benchmark suites.

A. Experimental Setup

Table II presents our experimental system configuration. We use a Qualcomm Snapdragon 888 Mobile Hardware Development Kit [36]. The Snapdragon 888 features a tri-cluster octa-core Kryo 680 CPU. The top core is a single Kryo 680 Prime processor. The second cluster comprises of three Kryo 680 Gold processors. The third cluster includes four Kryo 680 Silver processors. We refer to them as CPU Big, CPU Mid and CPU Little, respectively. There are 4 MBs of L3 cache memory that is shared across all clustered cores and 3 MBs of system-level cache memory.[1] It also features an

[1]This is an SoC-wide accessible cache (i.e., accessible by all components).

Adreno 660 GPU and an AI engine (AIE) with a Hexagon 780 Processor. AIE is intended for compute-intensive multimedia applications (e.g., video, audio, image processing), neural-network-related calculations and communications. There are 12GBs of LPDDR5 RAM and 256GBs of flash storage. The hardware board has Android 11 installed and is connected to an external display with a 1920×1080 pixels (Full HD) resolution.

We use Qualcomm's Snapdragon Profiler [37] to capture various metrics. The tool's real-time view option enables us to capture over 190 hardware performance metrics that cover the following categories: 1) CPU-related including cores, cache, and branch predictor information, 2) GPU-related including cores, shaders, GPU memory and GPU stalls, and 3) metrics about the AIE, system memory and temperature.

Benchmark Suites. We have split each suite into individual benchmarks that users can execute independently, except for *Antutu* and *GFXBench*. The *Antutu* benchmark consists of four components, *GPU*, *Mem*, *CPU* and *UX*. However, users are unable to execute them individually; instead, they must run the entire suite. During *Antutu*'s execution, micro-benchmarks from each component are bundled together and executed consecutively. Given the benchmark's extended runtime, we've organized the collected statistics into four segments, aligning with its constituent parts. Regarding *GFXBench*, we have grouped its 29 micro-benchmarks into three categories, aligning with the classification by the benchmark designers, as outlined in Section III. We ran all benchmarks thrice and averaged their metrics across runs.

Limitations. We briefly describe the limitations of our current evaluation setup:

1) Our evaluation platform and the tools available preclude the inclusion of power or thermal information in our analysis. The absence of a battery and casing in the development board limits the representativeness of thermal readings for a mobile platform. Furthermore, conducting power readings necessitates external hardware, which is not within the scope of our current capabilities.
2) We are limited in our analysis by the available metrics. As such, we are unable to broaden our analysis to other IP components.
3) Snapdragon Profiler provides information about total system memory usage, including the Android OS and the services running. We gathered statistics with the system being idle and computed the average memory usage. We then deducted this amount from all process specific information.

TABLE II: Hardware platform for experiments

Development Board	Qualcomm Snapdragon 888 Mobile Hardware Development Kit
CPU	1x Kryo 680 Prime processor (ARM Cortex-X1 based) @ up to 3.0GHz (CPU Big)
	3x Kryo 680 Gold processors (ARM Cortex-A78 based) @ up to 2.42 GHz (CPU Mid)
	4x Kryo 680 Silver processors (ARM Cortex-A55 based) @ up to 1.8 GHz (CPU Little)
Cache	CPU Big Core: 64 KB L1 Inst., 64 KB L1 Data & 1 MB L2
	Per CPU Mid Core: 512 KB L2
	Per CPU Little Core: 128 KB L2
	4 MB L3 (for CPU cores)
	3 MB System-level
GPU	Adreno 660
AI Engine (AIE)	Hexagon 780 Processor
RAM	12GB LPDDR5
Storage	56GB
Manufacturing Process	Samsung 5nm low power early (LPE)
OS	Android 11
External Display	1920×1080 pixels

TABLE III: Correlation values between metrics.

	IC	IPC	Cache MPKI	Branch MPKI	Runtime
IC	1				
IPC	0.400	1			
Cache MPKI	-0.228	-0.845	1		
Branch MPKI	-0.174	-0.672	0.867	1	
Runtime	0.588	-0.242	0.460	0.350	1

V. Evaluation

A. Metrics

Figure 1 shows the average values of a few important performance metrics for all benchmarks. These metrics are the Dynamic Instruction Count (IC), Instructions per Cycle (IPC), Cache Misses per Kilo Instructions (MPKI), Branch MPKI and Runtimes. We split and colour the benchmarks into 5 groups, based on the clustering in Section VI. Table III shows the correlation between metrics, calculated using the Pearson correlation coefficient [38]. The correlation coefficient is an important measure that quantifies the extent of interconnection among these metrics. Correlation values of above 0.8 or below -0.8 are strong positive and negative associations, respectively. Values between $|0.4-0.8|$ are moderate associations, while lesser values indicate that there is no association present.

The **dynamic instruction counts** of the included benchmarks are in the order of billions. There is an order of magnitude difference between the smallest benchmark (*GFXBench Special Tests* at 1 billion) and the largest one (*Geekbench 6 CPU* at 57 billion). The average IC is 14 billion. For comparison, *SPEC CPU 2017* benchmarks are in the order of trillions. [39]. We observe that newer benchmarks tend to have higher instructions counts (e.g., *Geekbench 6* vs *Geekbench 5*, *3DMark Wild Life* vs *3DMark Slingshot*).

The **IPC** metric is a critical measure for assessing the efficiency and performance of processors. A high IPC is indicative of high instruction-level parallelism. In general, a CPU-centric benchmark's IPC value tends to be bigger than one [40]. In our evaluation platform, the CPU Big core can theoretically achieve a maximum IPC value of 8, for reference [41]. All benchmarks explicitly targeting the CPU (i.e., *Antutu CPU*, *Geekbench 5 CPU* and *Geekbench 6 CPU*) have an average IPC of 1.16. We observe that benchmarks with a focus on graphics (e.g., *GFXBench High Tests*) exhibit lower IPC values, averaging at 0.55. Typically, 3D-graphics applications employ a strategy of pre-allocating textures anticipated for

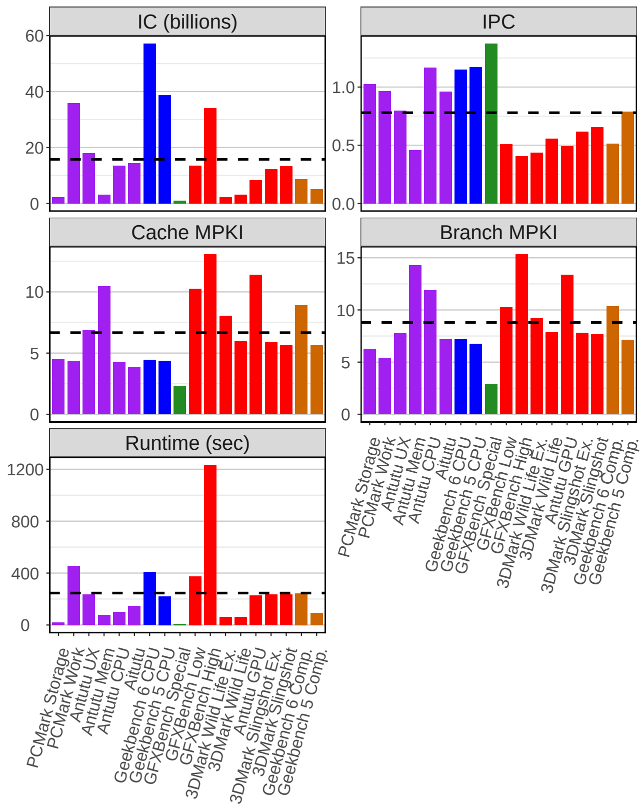

Fig. 1: Benchmark metrics. Dash lines show the average values of each metric.

imminent use, leading to heightened memory bandwidth usage and significant occupancy of cache memory space [42], [43]. Our analysis reveals a notable correlation between IPC values and the utilization of GPU shader cores. We posit that the diminished IPC values can be attributed to cache contention arising from the substantial loading of graphics-related data into the cache. The only outlier is *Antutu Mem* with an IPC of 0.45, affected by its high number of cache misses.

Cache misses per thousand instructions (MPKI) is a strong indicator of data retrieval inefficiency. Similarly, **branch MPKI** reflects the efficiency of the processor's branch prediction mechanism. Both metrics impact performance by introducing delays, necessitating data retrieval from slower memory levels and causing missteps in program execution. Moreover, both metrics are dependent on the micro-architectural design choices and intricacies of the SoC. We capture the misses across all levels of the cache hierarchy. We see that these metrics have similar trends and they exhibit negative correlations with IPC.

It is important for researchers to be cognizant of benchmark **runtimes**, as they impact the time spent on evaluating new designs. The average runtime is slightly over 200 seconds. Dynamic instruction count is frequently used as a predictor for the runtime of a program. However, we see that there is only a moderate correlation of 0.588 with the benchmark runtimes, as shown in Table III.

TABLE IV: Performance Metrics.

Metric	**Explanation**
CPU Load	Load on CPU Core (CPU Frequency × CPU % Utilization)
GPU Load	Load on GPU (GPU Frequency × GPU % Utilization)
% Shaders Busy	Percentage of time that all Shader cores are busy
% GPU Bus Busy	Percentage of time the GPU's bus to system memory is busy
AIE Load	Load on AIE (AIE Frequency × AIE % Utilization)
Used Memory	Percentage of total system memory used

B. Temporal Behaviour

Averaging values across a time series condenses the information and offers a succinct summary. While this simplification aids in clarity, it comes with inherent limitations. It can obscure nuances and variations present in the complete data. Examining the entire time series preserves detailed temporal information, enabling a more granular understanding of workload fluctuations and system behaviour.

Figure 2 shows the normalized values of six metrics across the normalized execution time for all benchmarks. We normalize the values of the metrics to the [0 – 1] range. The highest values recorded for each metric across all benchmarks serve as the normalization's upper bound, while the inverse is true for the lower bound. Coloured regions indicate a value exceeding 0.5. Table IV shows these metrics with an explanation. We opt for CPU Load instead of CPU utilization as it incorporates the frequency the CPU cores are running at. High CPU utilization levels at low frequencies can be misleading in terms of the stress the CPU cores are under. Following the same reasoning, we pick GPU and AIE Load. Percentage Shaders Busy and Percentage GPU Bus Busy show the amount of execution time that the GPU Shaders and the GPU bus are utilized. We chose these metrics because they collectively serve as important indicators of heterogeneous components' behaviour in a mobile SoC. We can make several observations.

Observation #1: Benchmarks that include multi-core or multi-threaded components show high CPU load levels.

Geekbench 5 CPU and *Geekbench 6 CPU* exhibit a spike in CPU load when the multi-core segment of the benchmark is running. The single-core part has a significantly lower CPU load of close to 30% for both benchmarks. Similarly, *Antutu CPU* contains a multi-core micro-benchmark near the end that focuses on multi-threading and multi-tasking performance. The uptick in the beginning of *Antutu CPU* is due to a general matrix multiplication (GEMM) routine, commonly used in benchmarks due to its intensity [44], [45]. Most efficient matrix multiplication routines are multi-threaded. The steep increase in CPU load in *3DMark Slingshot* and *3DMark Slingshot Extreme* is due a physics tests. This test measures CPU performance, while minimizing the GPU workload. It has three levels, successively more intensive, and is highly

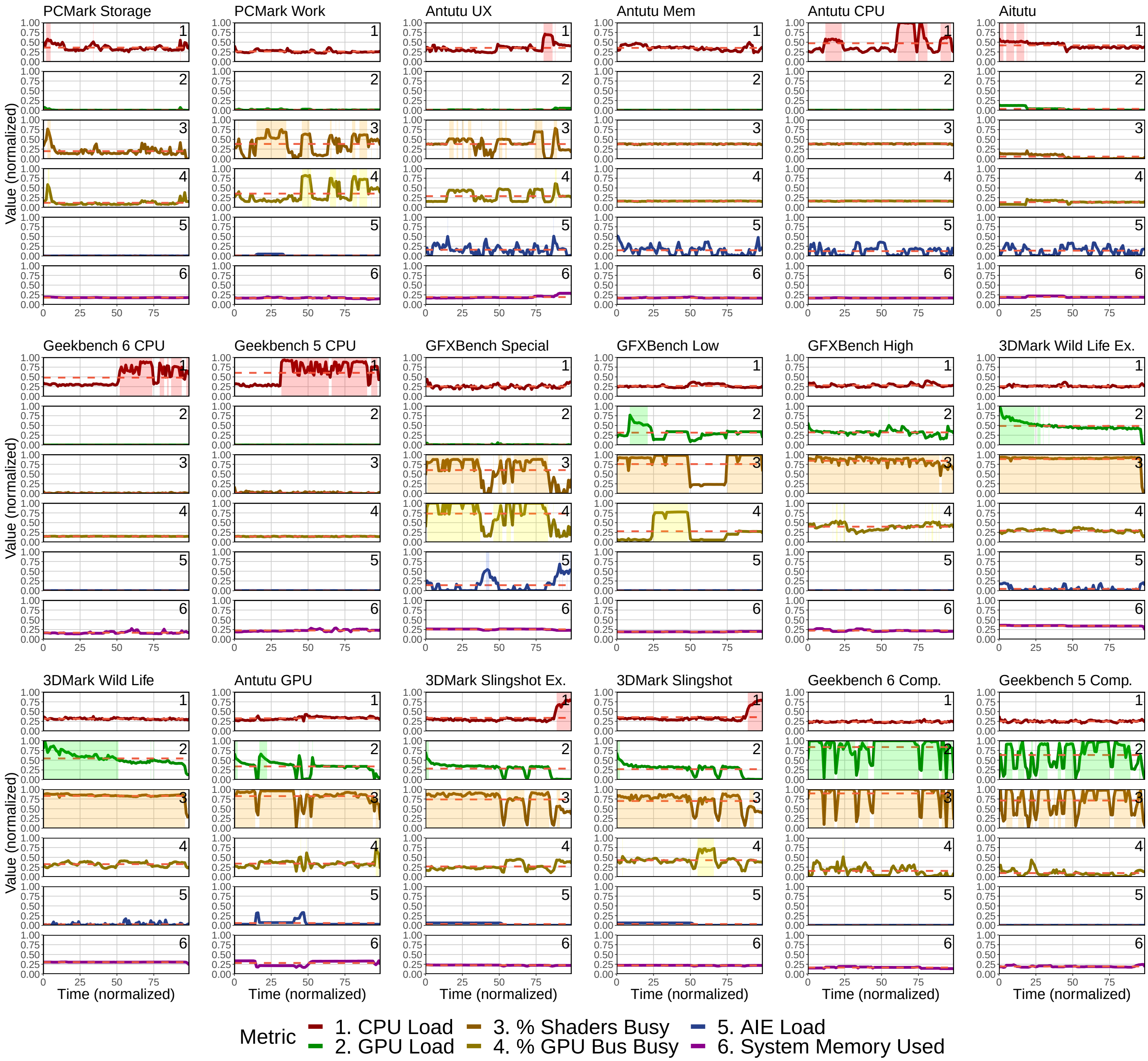

Fig. 2: Values of various metrics across the normalized runtime of all benchmarks. Coloured regions indicate that a metric's normalized value exceeds 0.5. Dash lines show the average values across the entire benchmark execution. [2]

multi-thread.

Observation #2: Benchmarks that use the Vulkan API have lower GPU load than the ones that use OpenGL.

GFXBench benchmarks that use OpenGL have 9.26% higher GPU load compared to Vulkan ones. This is due to Vulkan being a more efficient API [46], [47].

Observation #3: Usage of GPU resources is not limited to GPU-related benchmarks.

We observe that GPU shaders are not used exclusively by benchmarks that include 3D graphics workloads (e.g., *3DMark* and *GFXBench*). *PCMark Work* exhibits sustained periods where the majority of shaders are used. This is attributed to the video and photo editing workloads that it contains. Similarly, we see that the percentage of time the GPU memory bus is busy is not proportional to a workload's graphical intensity.

Observation #4: Newer benchmarks are not always more computationally intensive.

[2] *GFXBench Special Test* appears to have AIE Load over 50% near the end of its execution time. The region does not appear coloured due to the timestamps with high loads not being contiguous.

Antutu version 9 introduced a new GPU-focused micro-benchmark, *Swordsman*, that is executed at the beginning of the benchmark. It is followed by the *Refinery* and *Terracotta Warriors* micro-benchmarks and two short image-processing micro-benchmarks. The normalized running times of the benchmarks are 15%, 30% and 49% of the total duration of *Antutu GPU,* respectively. Figure 2 shows that the CPU load of *Antutu GPU* spikes at 16% and 49% of the execution. Both of these moments are not during the newest benchmark's execution. *Swordsman*, *Refinery* and *Terracotta Warriors* have 28%, 31% and 35% CPU load, respectively.

Observation #5: Benchmarks make little use of AIE.

While there are parts of benchmarks that stress the AIE more, the average load is just 5%. *Antutu UX* exhibits short peaks close to 50% in the scroll delay, webview rendering and video decode tests. In general, *Antutu UX* includes photo- and video-related tests, leading to increased AIE usage. *Antutu CPU*'s mathematical functions (e.g., fast Fourier transform (FFT), MAP) and the PNG decoding test result in increased load. All the previously mentioned workloads are encompassed within the domain of DSP tasks, supported by the AIE. Similarly, *3DMark Wild Life* and *3DMark Wild Life Extreme* use FFT operations as part of the post-processing techniques. *Aitutu* focuses on AI workloads. It has three image-oriented tasks, image classification, object detection and super resolution. We also observe higher AIE load levels on *GFXBench Special tests*. They compare a single rendered frame against a reference frame using a Peak-Signal-to-Noise-Ratio (PSNR) metric, based on the Mean Square Error (MSE). It is split into two sections, with the latter doing computations in higher precision. We theorize that the increase in AIE usage is due to the computation of the PSNR metric at the end of each part. Lastly, *PCMark Work* has an uptick in AIE load due to the video editing part of the test.

Observation #6: The memory footprint of benchmarks is moderate.

The average system memory usage across all benchmarks is 21.6%. This is 2.55GB out of the 11.83GB of system memory. GPU-oriented benchmarks have higher memory usage compared to others. Our analysis reveals a strong correlation between utilized memory, the processing load on GPU shaders, and L1 Texture misses. As highlighted in Section V-A, benchmarks with intensive graphics demands exhibit elevated memory occupancy, attributed to the loading of textures. The highest recorded usage is 4.3GB during the execution of *Antutu GPU*. The highest average memory consumption is 3.8GB (34.5%) during the execution of *3DMark Wild Life Extreme*.

In the rest of section, we focus on some interesting benchmark behaviour characteristics.

- *Antutu UX* has high CPU load near the end due to the video encoding and decoding tests. These micro-benchmarks use common video formats like H264, H265, VP9 and AV1 [48]. All formats except the last are supported by the SoC's AIE component. We speculate that the lack of support for AV1 leads to a considerable increase in CPU load, as it cannot be processed by the AIE.
- *GFXBench High* and *Low* benchmarks have two varieties of tests, on-screen and off-screen. In the former, the GPU drawing operations are performed on a region of memory that goes directly to the display output. This means that the tests run at a Full HD resolution. In the latter, the computations in memory are not directed to the display output. Off-screen rendering is used to generate intermediary images in computer graphics, like post-processing filters (e.g., blur). The off-screen tests have various resolutions. All tests can be executed at Full HD. The Manhattan test can also be executed at 2K QHD, while Aztec Ruins contain all previous options and a 4K one. In our tests, we observe that *High level* off-screen benchmarks result in a 14.5% increase in GPU load. *Low level* off-screen benchmarks exhibit a 62.85% increase.

C. CPU Heterogeneity Analysis

Designing and evaluating mobile SoCs comprehensively is challenging, due to the heterogeneity of both their hardware components and workloads [49]. It is imperative for researchers to be aware of the level of heterogeneity exhibited by mobile benchmarks. Thus, they can ensure the fair and comprehensive testing of proposed techniques by picking appropriate target workloads.

So far, we have looked into CPU load as one metric that is the mean of the load of all cores. However, there are 8 cores in the SoC we are using, divided into 3 clusters. Work is not spread equally among all these cores and clusters.

In this part of the analysis, we examine the degree of CPU load exhibited by the benchmarks. Figure 3 shows the results. We categorize normalized CPU core load metrics into four levels (each covering 25% of the [0 – 1] range) and count their occurrences. This is averaged across CPU core clusters, as the load values for cores belonging to the same cluster are almost identical. The x-axis indicates the load of the three CPU core clusters as defined in Table II, while the y-axis represents the normalized benchmark runtimes. We use colours to represent the four load levels. Table V shows the average percentage of execution time each CPU core cluster spend in each load level across all benchmarks.

We make the following observations:

Observation #7: Bigger, more powerful cores have higher load levels than medium cores.

CPU Big has high load levels (i.e., 50% – 100%) sustained for longer than CPU Mid in all but one of the benchmarks that they are actively used. In *3DMark Wild Life*, *GFXBench Low* and *GFXBench High* both CPU Mid and CPU Big clusters see minimal use. *Aitutu* is the only benchmark where the CPU Mid cluster exhibits high load levels sustained for longer compared to CPU Big.

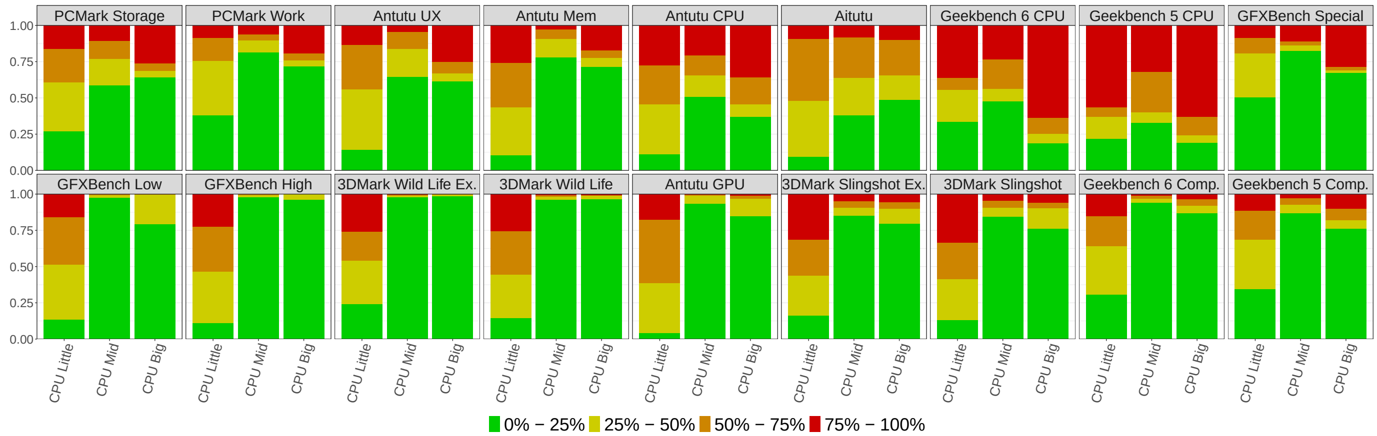

Fig. 3: Load levels of CPU core clusters across the sub-benchmarks.

TABLE V: Percentage of execution time spent by the CPU core clusters in the load levels.

CPU Cluster	0% - 25%	25% - 50%	50% - 75%	75% - 100%
CPU Little	21%	32%	25%	22%
CPU Mid	76%	8%	8%	8%
CPU Big	69%	7%	6%	18%

Observation #8: GPU tests tend to use only the energy-efficient cores.

CPU Big and CPU Mid have fewer instances of high load than CPU Little in all GPU tests (lower row of Figure 3) The computational demands of GPU-oriented benchmarks can mostly be satisfied with just the energy-efficient CPU cores.

Observation #9: Workloads tend not to exploit more than one type of core concurrently [50].

We observe a consistent load on all CPU core clusters only in *Aitutu*, *Antutu CPU*, *Geekbench 6 CPU* and *Geekbench 5 CPU*. As highlighted in Section V-B under Observation #1, these benchmarks are distinct in having workloads explicitly designed for multi-core architectures. Among them, the latter is the only benchmark that exhibits sustained high load levels in CPU Mid for more than half of its execution time.

Overall, we observe that few benchmarks utilize all CPU core clusters, even though heterogeneous mobile architectures have existed commercially since at least 2011 [51], [52]. Using the cores in the CPU Little cluster proves adequate in most cases. There is potential for future benchmark designs to fully utilize all CPU core clusters.

VI. Similarity and Redundancy

The benchmark suites we have included in our analysis contain 41 sub-benchmarks that can be individually executed. Their combined runtime on a real device is over 110 minutes. Architectural research often employs simulators (e.g., gem5 [53]) that are thousand times slower than native execution [54]. While limiting simulation to smaller regions of interest is commonly employed [55], [56], it is not always an option. The closed-source nature of these commercial benchmark suites renders modifications to their source code nearly impossible. Moreover, choosing a Region of Interest (ROI) poses challenges, given our observations that these benchmarks can encompass various types of workloads (e.g., *3DMark Slingshot*). Thus, simulating all benchmarks or even natively running them can result in prohibitively expensive execution times.

In this section, we perform a statistical analysis on the similarity exhibited between benchmarks of different suites. We also discuss methods for subsetting the benchmark set while retaining its behavioural and performance characteristics.

A. Similarity

Clustering is a method for finding subgroups of observations within a larger dataset. In this section, we present some issues that arise with the use of clustering algorithms.

Optimal number of clusters. Selecting the optimal number of clusters is challenging in unsupervised learning. In scenarios like workload characterization with diverse benchmarks, the ideal number of clusters may not be evident. Most algorithms lack prior knowledge, often relying on user input for cluster count. However, real-world workload differences are not always clear-cut, making the number of clusters ambiguous. Consequently, evaluating the produced clusters using validation measures becomes crucial in ensuring meaningful insights.

We use two methods, *Internal* and *Stability* validation [57]. Internal validation measures the compactness, connectedness and separation of the clusters. That is, we want the average distance within a cluster to be as small as possible and the average distance between clusters to be as large as possible. We use two popular measures that demonstrate the above characteristics, the *Dunn Index* [58] and the *Silhouette Width* [59]. Higher values are better for these measures. Stability validation evaluates the consistency of a clustering result by comparing it with the clusters obtained after each column is removed, one at a time. We use two cluster stability measures, the *average proportion of non-overlap (APN)* and *the average distance (AD)* [60]. Lower values are better for these measures.

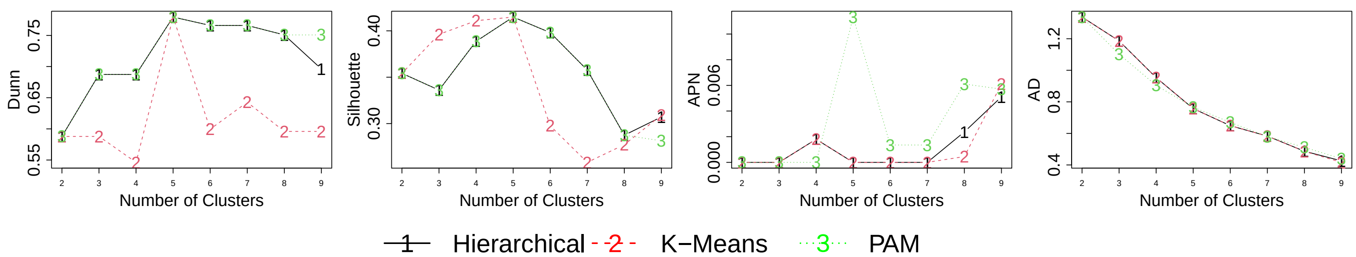

Fig. 4: Techniques validating the number of clusters. In Dunn and Silhouette higher is better, while in APN and AD lower is better.

Clustering Algorithms. There are several clustering algorithms, some of which see widespread use. However, there is no single method that is universally considered the best. Thus, we use three common clustering techniques, *K-means* [61], *Partitioning Around Medoids (PAM)* [62] and *Agglomerative Hierarchical clustering* [63]. Instead of just using a single algorithm, we try to find common patterns that emerge across different techniques.

Figure 4 shows the results of the clustering validation analysis. The two sub-figures to the left show the internal validation measures, while the two right ones show the stability validation measures. We observe that the optimal number of clusters is 5 for both the internal measures, regardless of the clustering technique used. As for the stability measures, APN shows a tie among various number of clusters with a general preference towards the lower range. The AD measure indicates a strong bias for a higher number of clusters. Considering that three out of four measures have 5 clusters as their top pick, we select this number.

Figures 5 and 6 depict the clusters created after applying a Hierarchical and a K-Means clustering algorithm, respectively. We omit the results of a PAM algorithm as they are similar to K-Means. We average the metrics across the benchmarks' runtime. The resulting data are provided as input to all these techniques. We see that all three algorithms group the sub-benchmarks identically. This validates the correctness of our clusters.

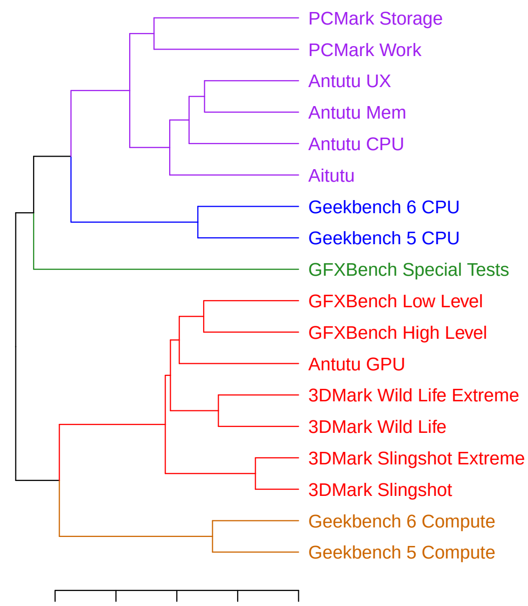

Fig. 5: Benchmark clustering results of a Hierarchical clustering algorithm.

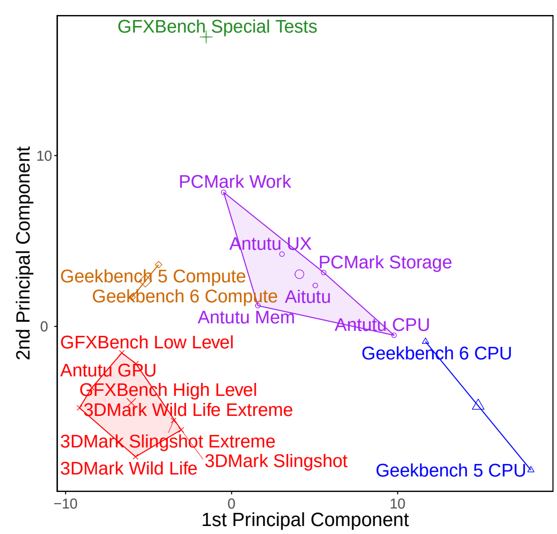

Fig. 6: Benchmark clustering results of a K-Means clustering algorithm.

B. Reduced Benchmark Set

Computer architects frequently subset benchmark suites to streamline evaluations and focus on specific aspects relevant to their research [64]. The goal of a subset should be to provide the same information as the full benchmark suite [65]. Researchers have to navigate a fine balance between efficiency gains and the need for comprehensive coverage.

A commonly employed subsetting technique is to select a single benchmark from every benchmark cluster. However, this method introduces potential challenges, as it does not guarantee coverage of all application domains. Furthermore, it does not take into account the unique circumstances of the benchmarks in our original set. Thus, we choose to showcase and evaluate multiple subsetting techniques.

Naive Subset. We select one benchmark from each cluster. We make the decision based on the benchmarks' execution times, since we want to minimize time spent on evaluation. The Naive subset is comprised of *PCMark Storage*, *Geekbench 5 CPU*, *GFXBench Special*, *3DMark Wild Life* and *Geekbench 5 Compute*.

Select Subset. As we explain in Section IV-A, the different parts of the *Antutu* benchmark cannot be executed individually. The user can only run it in its entirety. Thus, we begin our selection with *Antutu*. All of *Antutu*'s segments (i.e., *Antutu CPU*, *Antutu Mem*, *Antutu UX*) are grouped in the same cluster except *Antutu GPU*. Our goal is to ensure that the reduced set stresses all of the SoC components. Thus, we include *GFXBench Special Tests* as it provides the highest AIE load. Finally, we select *Geebench 5 CPU* as it covers the need for stressing all CPU core clusters, while having a shorter execution time than *Geekbench 6 CPU*.

Select + GPU Subset. The previous reduced set included *Antutu GPU* for stressing the GPU subsystem. However, it does not provide the highest GPU load. We present a third reduced set option that includes *Geekbench 6 CPU*, as it has the highest average GPU load exhibited among all benchmarks.

Table VI shows the total running times for all benchmarks and the proposed reduced sets. It also shows the running time reduction percentages compared to executing all benchmarks. We observe that even the subset with the slowest running time (Select + GPU) results in a close to 75% reduction in execution time.

Careful consideration is crucial when creating a benchmark subset. The objective is to choose a subset of benchmarks that, when executed, yields equivalent information as the full suite but in considerably less time. We measure the representativeness and coverage of our presented subsets by utilizing a technique presented by Yi et al. [66]. We follow these steps:

1) Create a vector containing the values of all performance metrics of each benchmark.
2) Normalize the performance metrics to the maximum recorded value of each.
3) Compute the Euclidean distance between each benchmark vector *not* in the subset to each benchmark vector in the subset.
4) For each benchmark vector *not* in the subset, we pick the minimum Euclidean distance with a benchmark vector in the subset.
5) Sum the Euclidean distances for all benchmark vectors and assign that value as the total minimum Euclidean distance for that subset.

A smaller total minimum Euclidean distance means that the benchmarks that are not in the subset are very close to a benchmark that is in the subset. This means that every benchmark not in the subset is accurately represented by one in the subset. Also, it means that the coverage extends to all benchmarks in the original set.

Figure 7 shows the total minimum Euclidean distances for the three subsets. We start by having a single benchmark in each subset. For example, the Naive subset begins with *PCMark Storage*, while the Select subset has *Antutu CPU*. We continue by adding one of the benchmarks that belongs in each subset at each step. After all benchmarks that belong in a subset have been added, we add the rest of the benchmarks. We measure the total minimum Euclidean distance at each step. The dashed lines indicate the total minimum Euclidean distance of each subset as they were presented in Section VI-B. Notably, the Select + GPU subset, comprising 7 benchmarks, exhibits a total minimum Euclidean distance of 11. This positions it towards the lower end of the range of Euclidean distances, specifically at the 32.5% percentile. This signifies a reduction of 22.96% and 9.78% when compared to the Naive subset with 5 and 7 benchmarks, respectively.

TABLE VI: Running times and percentage reductions for all proposed subsets.

	Original Set	Naive Set	Select Set	Select + GPU Set
Running Time (sec)	4429.5	401.7	865.2	1108.36
Running Time Reduction	-	90.93%	80.47%	74.98%

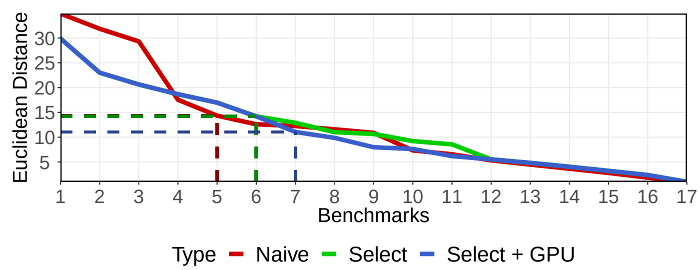

Fig. 7: Normalized Euclidean distances for reduced benchmark subsets.

VII. Conclusion

In summary, our research thoroughly explored and explained commonly used commercial mobile benchmark suites, offering important insights for the computer architecture community. We analyzed hardware performance metrics, uncovering the details of workload similarities and their impact on different parts of the system-on-chip (SoC). Our study not only looked at how workloads change over time but also presented the complexities in how CPU core clusters are used.

Moreover, we found a smaller set of benchmarks, the Select + GPU combination, which significantly reduces the time needed for evaluations by almost 75%. This reduction in time doesn't compromise the thorough assessment needed for robust designs. Our discoveries highlight the importance of choosing the right benchmarks for efficient evaluations, especially considering the ever-changing landscape of mobile system architectures.

We believe that these insights contribute to the ongoing discourse on benchmarking methodologies, providing valuable considerations for researchers engaged in mobile system architecture assessments.

References

[1] Mar 2023. [Online]. Available: https://www.statista.com/statistics/245501/multiple-mobile-device-ownership-worldwide/

[2] N. de Loisy, *Transportation and the Belt and Road Initiative: A Paradigm Shift*. Amazon Digital Services LLC, 2020.

[3] V. J. Reddi, H. Yoon, and A. Knies, "Two Billion Devices and Counting," *IEEE Micro*, vol. 38, no. 1, pp. 6–21, 2018.

[4] M. Hill and V. Janapa Reddi, "Gables: A Roofline Model for Mobile SoCs," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 317–330.

[5] M. Halpern, Y. Zhu, and V. J. Reddi, "Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 64–76.

[6] J. Bucek, K.-D. Lange, and J. v. Kistowski, "SPEC CPU2017: Next-Generation Compute Benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 41–42. [Online]. Available: https://doi.org/10.1145/3185768.3185771

[7] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 72–81. [Online]. Available: https://doi.org/10.1145/1454115.1454128

[8] M. Hayenga, C. Sudanthi, M. Ghosh, P. Ramrakhyani, and N. Paver, "Accurate System-Level Performance Modeling and Workload Characterization for Mobile Internet Devices," in *MEmory performance: DEaling with Applications, systems and architecture (MEDEA)*. New York, NY, USA: Association for Computing Machinery, 2008, p. 54–60. [Online]. Available: https://doi.org/10.1145/1509084.1509092

[9] N. Chidambaram Nachiappan, P. Yedlapalli, N. Soundararajan, M. T. Kandemir, A. Sivasubramaniam, and C. R. Das, "GemDroid: A Framework to Evaluate Mobile Platforms," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, p. 355–366, jun 2014. [Online]. Available: https://doi.org/10.1145/2637364.2591973

[10] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *2011 IEEE International Symposium on Workload Characterization (IISWC)*, 2011, pp. 81–90.

[11] D. Pandiyan, S.-Y. Lee, and C.-J. Wu, "Performance, energy characterizations and architectural implications of an emerging mobile platform benchmark suite - MobileBench," in *IEEE IISWC*, 2013.

[12] S. Chetoui, R. Shahi, S. Abdelaziz, A. Golas, F. Hijaz, and S. Reda, "ARBench: Augmented Reality Benchmark For Mobile Devices," in *IEEE ISPASS*, 2022.

[13] N. Salehnamadi, A. Alshayban, I. Ahmed, and S. Malek, "A Benchmark for Event-Race Analysis in Android Apps," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 466–467. [Online]. Available: https://doi.org/10.1145/3386901.3396602

[14] Q. Zhang, X. Li, X. Che, X. Ma, A. Zhou, M. Xu, S. Wang, Y. Ma, and X. Liu, "A Comprehensive Benchmark of Deep Learning Libraries on Mobile Devices," in *Proceedings of the ACM Web Conference 2022*, ser. WWW '22. New York, NY, USA: ACM, 2022, p. 3298–3307.

[15] G. Bavota, M. Linares-Vásquez, C. E. Bernal-Cárdenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk, "The Impact of API Change- and Fault-Proneness on the User Ratings of Android Apps," *IEEE Transactions on Software Engineering*, vol. 41, no. 4, pp. 384–407, 2015.

[16] "Snapdragon Summit 2023: Keynote," https://www.youtube.com/live/h_vh7_n_OPs?si=id9VBiR6J2V7D4sV.

[17] M. Hill and V. Janapa Reddi, "Accelerator-level Parallelism," 6 2020. [Online]. Available: https://research.cs.wisc.edu/multifacet/papers/ALP_technion2020_06.pdf

[18] Wikipedia contributors, "Apple silicon — Wikipedia, the free encyclopedia," 2023, [Online; accessed 13-December-2023]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Apple_silicon&oldid=1189613601

[19] Y. Guo, Y. Xu, and X. Chen, "Freeze it if you can: Challenges and future directions in benchmarking smartphone performance," ser. HotMobile '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 25–30. [Online]. Available: https://doi.org/10.1145/3032970.3032979

[20] H. Cho, C. Eun, and O. Jeong, "Design of adaptive threshold control algorithm based on real-time cpu operation patterns for mobile devices," *Electronics Letters*, vol. 55, no. 12, pp. 688–690, 2019. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/el.2018.8160

[21] B. Egilmez, G. Memik, S. Ogrenci-Memik, and O. Ergin, "User-specific skin temperature-aware DVFS for smartphones," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015, pp. 1217–1220.

[22] X. Yuan, O. Setayeshfar, H. Yan, P. Panage, X. Wei, and K. H. Lee, "DroidForensics: Accurate Reconstruction of Android Attacks via Multi-Layer Forensic Logging," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 666–677. [Online]. Available: https://doi.org/10.1145/3052973.3052984

[23] Y. Xue, X. Zhang, X. Yu, Y. Zhang, Y. Tan, and Y. Li, "Isolating Host Environment by Booting Android from OTG Devices," *Chinese Journal of Electronics*, vol. 27, no. 3, pp. 617–624, 2018. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cje.2018.03.017

[24] G. Brunner, "Android Benchmark," *Distributed Computing Group*, 2016.

[25] S.-R. Ohk, S. M. Hong, and Y.-J. Kim, "Phase-based predicting the battery remaining time for Android mobile devices," in *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, 2021, pp. 942–944.

[26] Y. Wang, V. Lee, G.-Y. Wei, and D. Brooks, "Predicting New Workload or CPU Performance by Analyzing Public Datasets," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 4, jan 2019. [Online]. Available: https://doi.org/10.1145/3284127

[27] Y. Lin, A. Barker, and J. Thomson, "Modelling VM Latent Characteristics and Predicting Application Performance using Semi-supervised Non-negative Matrix Factorization," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, 2020, pp. 470–474.

[28] J.-G. Park, C.-Y. Hsieh, N. Dutt, and S.-S. Lim, "Co-cap: Energy-efficient cooperative cpu-gpu frequency capping for mobile games," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, ser. SAC '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1717–1723. [Online]. Available: https://doi.org/10.1145/2851613.2851671

[29] P. Mercati, F. Paterna, A. Bartolini, L. Benini, and T. S. Rosing, "WARM: Workload-Aware Reliability Management in Linux/Android," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 9, pp. 1557–1570, 2017.

[30] K. Lee, S.-R. Ohk, S.-G. Lim, and Y.-J. Kim, "Phase-based accurate power modeling for mobile application processors," *Electronics*, vol. 10, no. 10, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/10/1197

[31] "3DMark Android." [Online]. Available: https://benchmarks.ul.com/3dmark-android

[32] "AnTuTu Benchmark." [Online]. Available: https://www.antutu.com/en/index.htm

[33] "Geekbench 6." [Online]. Available: https://www.geekbench.com/

[34] "Gfxbench." [Online]. Available: https://gfxbench.com/benchmark.jsp

[35] "PCMark Android." [Online]. Available: https://benchmarks.ul.com/pcmark-android

[36] "Snapdragon 888 Mobile Hardware Development Kit." [Online]. Available: https://developer.qualcomm.com/hardware/snapdragon-888-hdk

[37] "Snapdragon Profiler." [Online]. Available: https://developer.qualcomm.com/software/snapdragon-profiler

[38] K. Pearson and F. Galton, "Vii. note on regression and inheritance in the case of two parents," *Proceedings of the Royal Society of London*, vol. 58, no. 347-352, pp. 240–242, 1895. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rspl.1895.0041

[39] R. Panda, S. Song, J. Dean, and L. K. John, "Wait of a decade: Did spec cpu 2017 broaden the performance horizon?" in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 271–282.

[40] W. Lee, J. Lee, B. K. Park, and R. Y. C. Kim, "Microarchitectural characterization on a mobile workload," *Applied Sciences*, vol. 11, no. 3, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/3/1225

[41] "Arm Cortex-X1: The First From The Cortex-X Custom Program." [Online]. Available: https://fuse.wikichip.org/news/3543/arm-cortex-x1-the-first-from-the-cortex-x-custom-program/

[42] A. Sembrant, T. E. Carlson, E. Hagersten, and D. Black-Schaffer, "A graphics tracing framework for exploring CPU+GPU memory systems," in *2017 IEEE International Symposium on Workload Characterization (IISWC)*, 2017, pp. 54–65.

[43] A. Sharma, S. Kondguli, and M. Huang, "Irrelevant data traffic in modern low power gpu architectures," in *2022 IEEE International Conference on Networking, Architecture and Storage (NAS)*, 2022, pp. 1–7.

[44] J. J. Dongarra, "The LINPACK Benchmark: An Explanation," in *Proceedings of the 1st International Conference on Supercomputing*. Berlin, Heidelberg: Springer-Verlag, 1988, p. 456–474.

[45] A. Lokhmotov, "GEMMbench: a framework for reproducible and collaborative benchmarking of matrix multiplication," 2015.

[46] O. Ferraz, P. Menezes, V. Silva, and G. Falcao, "Benchmarking vulkan vs opengl rendering on low-power edge gpus," in *2021 International Conference on Graphics and Interaction (ICGI)*, 2021, pp. 1–8.

[47] M. Lujan, M. McCrary, B. W. Ford, and Z. Zong, "Vulkan vs opengl es: Performance and energy efficiency comparison on the big.little architecture," in *2021 IEEE International Conference on Networking, Architecture and Storage (NAS)*, 2021, pp. 1–8.

[48] Wikipedia contributors, "Antutu — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=AnTuTu&oldid=1162767853, 2023, [Online; accessed 4-December-2023].

[49] "big.LITTLE Technology: The Future of Mobile." [Online]. Available: https://armkeil.blob.core.windows.net/developer/Files/pdf/white-paper/big-little-technology-the-future-of-mobile.pdf

[50] A. K. Singh, K. R. Basireddy, A. Prakash, G. V. Merrett, and B. M. Al-Hashimi, "Collaborative Adaptation for Energy-Efficient Heterogeneous Mobile SoCs," *IEEE Transactions on Computers*, vol. 69, no. 2, pp. 185–197, 2020.

[51] Wikipedia contributors, "Arm big.little — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=ARM_big.LITTLE&oldid=1186095396, 2023, [Online; accessed 6-December-2023].

[52] A. Phillips, "ARM Unveils Its Most Energy Efficient Application Processor Ever; Redefines Traditional Power and Performance Relationship with big.LITTLE Processing," https://www.businesswire.com/news/home/20111019006329/en/ARM-Unveils-Its-Most-Energy-Efficient-Application-Processor-Ever-Redefines-Traditional-Power-and-Performance-Relationship-with-big.LITTLE-Processing, 10 2011.

[53] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, aug 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718

[54] A. Sandberg, N. Nikoleris, T. E. Carlson, E. Hagersten, S. Kaxiras, and D. Black-Schaffer, "Full speed ahead: Detailed architectural simulation at near-native speed," in *2015 IEEE International Symposium on Workload Characterization*. IEEE, 2015, pp. 183–192.

[55] A. Sabu, H. Patil, W. Heirman, and T. E. Carlson, "Roiperf: A framework to rapidly validate workload sampling methodologies."

[56] P. Prieto, P. Abad, J. A. Herrero, J. A. Gregorio, and V. Puente, "SPECcast: A Methodology for Fast Performance Evaluation with SPEC CPU 2017 Multiprogrammed Workloads," in *Proceedings of the 49th International Conference on Parallel Processing*, ser. ICPP '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3404397.3404424

[57] J. Handl, J. Knowles, and D. B. Kell, "Computational cluster validation in post-genomic data analysis," *Bioinformatics*, vol. 21, no. 15, pp. 3201–3212, 08 2005. [Online]. Available: https://doi.org/10.1093/bioinformatics/bti517

[58] J. C. Dunn†, "Well-separated clusters and optimal fuzzy partitions," *Journal of Cybernetics*, vol. 4, no. 1, pp. 95–104, 1974. [Online]. Available: https://doi.org/10.1080/01969727408546059

[59] L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction To Cluster Analysis*, 01 1990.

[60] S. Datta and S. Datta, "Comparison and validation of statistical clustering techniques for microarray gene expression data," *Bioinformatics (Oxford, England)*, vol. 19, pp. 459–66, 04 2003.

[61] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[62] *Partitioning Around Medoids (Program PAM)*. John Wiley & Sons, Ltd, 1990, ch. 2, pp. 68–125. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470316801.ch2

[63] Springer Singapore, 2022. [Online]. Available: https://link.springer.com/book/10.1007/978-981-19-0420-2#bibliographic-information

[64] D. Citron, "MisSPECulation: Partial and Misleading Use of SPEC CPU2000 in Computer Architecture Conferences," in *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ser. ISCA '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 52–61. [Online]. Available: https://doi.org/10.1145/859618.859625

[65] H. Vandierendonck and K. De Bosschere, "Experiments with subsetting benchmark suites," in *IEEE International Workshop on Workload Characterization, 2004. WWC-7. 2004*, 2004, pp. 55–62.

[66] J. J. Yi, R. Sendag, L. Eeckhout, A. Joshi, D. J. Lilja, and L. K. John, "Evaluating Benchmark Subsetting Approaches," in *2006 IEEE International Symposium on Workload Characterization*, 2006, pp. 93–104.