



# Evaluating .NET MAUI as a replacement for native Android mobile application development with focus on performance

Lukas Palmqvist

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Bachelor of Science in Software Engineering. The thesis is equivalent to 10 weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

**Contact Information:**

Author(s):

Lukas Palmqvist

E-mail: lupa18@student.bth.se

University advisor:

Doctoral student, Julian Frattini

Department of Software Engineering

Faculty of Computing  
Blekinge Institute of Technology  
SE-371 79 Karlskrona, Sweden

Internet : [www.bth.se](http://www.bth.se)  
Phone : +46 455 38 50 00  
Fax : +46 455 38 50 57

---

# Abstract

**Background.** These days when developing applications for mobile platforms, there have been two main branches to pick between as a developer. The first option – denoted *native* development – targets only one specific mobile operating system (OS). The other is *cross-platform development* which can target multiple platforms simultaneously. A common concern with cross-platform development, however, is its performance when compared to native frameworks.

**Objectives.** In this thesis, the objective is to compare the performance of applications created for Android using the native framework and the cross-platform .NET MAUI framework by Microsoft.

**Methods.** The method used in this thesis was a combination of a literature (mapping) study and an experiment. The purpose of the mapping study was to identify relevant performance metrics. The experiment then observed those how those metrics, CPU and memory (RAM) usage, differed between the applications created using both frameworks.

**Results.** Overall, the .NET MAUI framework was significantly worse than the native framework on both measured performance metrics in 9 out of 12 tasks tested. The .NET MAUI framework was significantly better in 1 task, while 2 other tasks ended up with differences that were statistically insignificant.

**Conclusions.** An experiment was conducted to compare CPU usage in % and memory (RAM) usage in MB between the native framework and the .NET MAUI framework for Android development on a variety of tasks. The outcome of the experiment significantly favored the native framework statistically. However, due to of the small sample size in selected tasks as well as specific design choices it is unclear how real-world use would compare.

**Keywords:** .NET MAUI, cross-platform, Android, performance

---

## Acknowledgments

The utmost appreciation goes out to my supervisor, Julian Frattini, who fully guided me in every step of this Bachelor's thesis. Without his timely and detailed feedback, as well as his knowledge, this thesis would not have been even remotely close to what it currently is. My sincerest thanks.

I would also like to thank Systematiq AB for giving me the initial idea for this thesis and for allowing me access to their office for the duration of this thesis.

---

# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgments</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Purpose . . . . .	2
1.3 Scope . . . . .	2
<b>2 Related Work</b>	<b>3</b>
<b>3 Research Questions and Method</b>	<b>6</b>
3.1 Research Questions . . . . .	6
3.2 Mapping Study . . . . .	7
3.2.1 Article Search . . . . .	7
3.2.2 Article Selection . . . . .	8
3.2.3 Data Extraction and Mapping . . . . .	9
3.3 Empirical Study . . . . .	10
3.3.1 Goals . . . . .	10
3.3.2 Experimental Units . . . . .	10
3.3.3 Experimental Materials . . . . .	10
3.3.4 Tasks . . . . .	11
3.3.5 Hypothesis, Parameters and Variables . . . . .	12
3.3.6 Procedure . . . . .	13
3.3.7 Deviations . . . . .	14
<b>4 Results and Analysis</b>	<b>15</b>
4.1 Mapping Study . . . . .	15
4.2 Empirical Study . . . . .	17
4.2.1 CPU usage for Mathematical calculation task . . . . .	17
4.2.2 CPU usage for 1000 item list task . . . . .	18
4.2.3 CPU usage for Geolocation task . . . . .	18
4.2.4 CPU usage for Change Page task . . . . .	19
4.2.5 CPU usage for Download file task . . . . .	19
4.2.6 CPU usage for File system task . . . . .	19
4.2.7 RAM usage for Mathematical calculation task . . . . .	20
4.2.8 RAM usage for 1000 item list task . . . . .	20
4.2.9 RAM usage for Geolocation task . . . . .	21

4.2.10	RAM usage for Change Page task . . . . .	21
4.2.11	RAM usage for Download file task . . . . .	21
4.2.12	RAM usage for File system task . . . . .	22
<b>5</b>	<b>Discussion</b>	<b>23</b>
5.1	Mapping Study . . . . .	23
5.1.1	Interpretation of Mapping Study results . . . . .	23
5.1.2	Chosen metrics and tasks . . . . .	23
5.2	Empirical Study . . . . .	24
5.2.1	Interpretation of Empirical Study results . . . . .	24
5.3	Summary . . . . .	26
5.4	Validity Threats . . . . .	26
5.4.1	Conclusion Validity Threats . . . . .	26
5.4.2	Internal Validity Threats . . . . .	27
5.4.3	Construct Validity Threats . . . . .	27
5.4.4	External Validity Threats . . . . .	28
<b>6</b>	<b>Conclusions</b>	<b>29</b>
<b>7</b>	<b>Future Work</b>	<b>30</b>
	<b>References</b>	<b>31</b>
<b>A</b>	<b>Supplemental Information</b>	<b>34</b>
A.1	Windows PC and Emulator Environment . . . . .	34
A.2	Studies included in Literature Study . . . . .	35

This thesis will evaluate the how the performance differs between the cross-platform framework .NET MAUI and native Android development. .NET MAUI was released by Microsoft as a cross-platform alternative for .NET developers who are already invested in the ecosystem and want to save time when creating cross-platform business related systems. [24]

## 1.1 Background

These days, when developing applications for mobile platforms, there are two main branches to choose from as a developer. The developer can pick the “native” way to build applications, which basically means that the application will be developed specifically for that operating system or platform, and it will not run on any other platforms [2]. For the rest of this thesis, the *native* way of developing things will be referred to as the *native framework*. It is usually considered safer to stick with this approach when building infrastructure-critical applications, partly because of the direct access to underlying system functionality as well as lack of “wrapper” methods or other workarounds [2].

The second way to develop an application is through “cross-platform” development. This means that the majority (if not all) of the codebase used for an application in can be re-used when building the same application but for another platform, i.e., operating system. Due to this, cross-platform development has been getting more and attention from developers in the recent years, especially amongst mobile developers, with the main incentive being to minimize parallel workloads between multiple platforms and not having to create multiple applications which should lead to money saved down the line . Furthermore, it allows for more consistency between different platforms since both platforms access the same functionality abstracted by the cross-platform framework [1]. Another benefit is that instead of hiring developers specialized in developing apps for one platform, they can instead hire developers with much less regard to their specialties.

However, the main concern inhibiting developers to make the switch is the potential lack of performance or hardware utilization that comes with the abstraction of cross-platform development. Basically, because native functionality is abstracted behind the cross-platform framework, one is dependent on how the developers of that framework decided to implement each corresponding native functionality.

These drawbacks or concerns are more or less relevant depending on what framework one decides to use in the end (usually based on the scale of the company behind

it, and their ambitions for the framework in the future). Nevertheless, there have been an increasing number of frameworks trying to mitigate potential drawbacks related to performance and accessing underlying system functionality. [1,17]. Hence, there is a need for research which investigates said mitigative actions, more specifically the performance and accessibility when comparing a native application to an application made utilizing a cross-platform approach.

## 1.2 Purpose

The purpose of this thesis is to empirically determine whether the performance of the applications built with both frameworks significantly differ and, hence, one is preferable over the other. This will be done by developing an application using native and then crating a clone of it using .NET MAUI, allowing for a direct comparison in terms of relevant functionality.

The results of this thesis will benefit organizations which have previous knowledge of a larger ecosystems (in this scenario, .NET) but lack the time, funding, or manpower to investigate frameworks outside of that ecosystem. Taking that into consideration, combined with the fact that the usage of the .NET ecosystem in production has greatly increased [30] over the last 5 years, it would make sense for many smaller- to medium-sized organizations to try and adopt the .NET MAUI framework for mobile development. By doing this, the organization can utilize their previous domain knowledge and potentially drastically reduce development times by removing individual development of applications for specific platforms [26]. Furthermore, by potentially reducing the effort required to build an app, a company could reduce their e-waste such as computing power draw by great margin. This could affect the environment positively by allowing for a more sustainable practices in the long term, as that effort (both by reducing e-waste and manpower requirements for projects).

It is also in the direct interest of the industrial partner collaborating with the author that the implementation of an app in .NET MAUI performs equal to an application developed using the native framework for Android. From this it can be deduced that while performance is not the primary quality of .NET MAUI, it is nonetheless something companies need to be aware of when deciding what platform to utilize in their next project.

## 1.3 Scope

The focus of this thesis will be a comparison between a native Android application and a .NET MAUI application in terms of performance when executing similar tasks with comparable implementations. Because of time constraints, the implemented functionality was limited to a few select tasks and the comparison was only done on the Android operating system. Furthermore, by limiting the scope to one operating system (Android), more time can be spent researching each framework's standards. This in turn allows for a more educated approach when the apps are being developed. This also works in combination with the fact that Android has a bigger market share when compared to iOS (the second largest mobile OS) made the choice relatively one-sided. [31]

When it comes to .NET MAUI, no previous empirical or literature-focused studies have taken place from what could be found during the initial search to further domain knowledge performed on Google Scholar <sup>1</sup>. This showed a huge gap in research when it comes to .NET MAUI, and while it is most likely due to its recency, is still valid due to the potential upsides of .NET MAUI discussed in Section 1.2.

Furthermore, based on this no performance comparisons had thus been made yet compared to more mature cross-platform frameworks such as Flutter [35]. This would however mean that it would be hard to select relevant performance metrics, as well as tasks for testing those metrics without having any related works which directly work with .NET MAUI. The next step was then to attempt to generalize the knowledge of related cross-platform frameworks, to see if there are some form of patterns when constructing studies to these aspects.

Because of this, to make a more educated decision on what research questions to focus on for the thesis and its empirical study, and to also get a better grasp of how the field of more general cross-platform currently looks when it comes to state of the art, a related works study was conducted. During the search for related works, the goal was initially to find out what appropriate preliminary performance metrics could be when constructing the remaining research questions. Because the focus of this thesis is on performance comparison between a cross-platform (more specifically, cross-compiled) and native framework, similar types of papers were preferred during screening. However, since the specific cross-platform framework should not matter when selecting metrics of performance, the criteria were slightly loosened to a more general “cross-platform compared to native application performance” focus. The papers included in the related works study can be found in Table 2.1.

Depending on the type of paper, as well as the type of framework/platforms of focus, the performance metrics of interest varied a lot. Although, it did not vary the point where it was impossible to find any sort of indication towards common metrics of interest. Some reoccurring themes were file reading/writing, app size, CPU usage, memory (RAM) usage, app startup time, and computation. In papers with a more detailed selection criteria for performance metrics, the focus often was closely tied to CPU usage, memory (RAM) usage, FPS/frame duration, and energy consumption.

In most cases the performance metrics appeared to be chosen arbitrarily, or the author simply deduced that a performance metric would be relevant for that paper’s focus without any prior motivations. Furthermore, as previously hinted towards, none of these papers had utilized .NET MAUI to any degree in their testing, which

---

<sup>1</sup><https://scholar.google.com/>

is understandable because of its recency, but still causes some further uncertainty when trying to decide based on the current collection of papers.

Due to limited consensus as well as the quality of the sources not being verified, it was decided that a proper literature (mapping) study would be performed, which through statistical means can verify that the selected performance metrics would be relevant for any selected cross-platform framework being compared to a native one. The main takeaways from this section were the most common reoccurring keywords in the papers currently found. Based on these preliminary results, the most important recurring keywords in these papers were: “cross-platform”, “native”, “mobile”, and “performance”. With these keywords in mind, construction of the mapping study could begin.

Table 2.1: Related Work set of studies

APA Reference	Metric(s)
Grzmil, P., Skublewska-Paszowska, M., Łukasik, E., & Smołka, J. (2017). Performance analysis of native and cross-platform mobile applications. <i>Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska</i> , 7(2), 50-53.	Response times for (Computation, File access, Image downloading, Determining location)
Jia, X., Ebone, A., & Tan, Y. (2018, May). A performance evaluation of cross-platform mobile application development approaches. In <i>Proceedings of the 5th International Conference on Mobile Software Engineering and Systems</i> (pp. 92-93).	Time to build, rendering time, total UI response time, memory (RAM) usage, application size.
Nawrocki, P., Wrona, K., Marczak, M., & Sniezynski, B. (2021). A comparison of native and cross-platform frameworks for mobile applications. <i>Computer</i> , 54(3), 18-27.	Application size, startup time, memory (RAM) usage, CPU usage.
Rösler, F., Nitze, A., & Schmietendorf, A. (2014, July). Towards a mobile application performance benchmark. In <i>International Conference on Internet and Web Applications and Services</i> (Vol. 9, pp. 55-59).	Startup time, memory (RAM) usage, execution (response) time.
Willocx, M., Vossaert, J., & Naessens, V. (2015, June). A quantitative assessment of performance in mobile app development tools. In <i>2015 IEEE international conference on mobile services</i> (pp. 454-461). IEEE.	Startup time, pause and resume times, page load times, memory (RAM) usage, CPU usage, size of application.
Continued on next page	

Table 2.1 – continued from previous page

APA Reference	Metric(s)
Delia, L., Galdamez, N., Corbalan, L., Pesado, P., & Thomas, P. (2017, July). Approaches to mobile application development: Comparative performance analysis. In 2017 Computing conference (pp. 652-659). IEEE.	Computation (execution/response) time
Willocx, M., Vossaert, J., & Naessens, V. (2016, May). Comparing performance parameters of mobile app development strategies. In Proceedings of the International Conference on Mobile Software Engineering and Systems (pp. 38-47).	Response times, CPU usage, Memory (RAM) usage, size of application
Fournier, C. (2020). Comparison of Smoothness in Progressive Web Apps and Mobile Applications on Android.	Smoothness (frame duration), Memory (RAM) usage, CPU usage
Huber, S., & Demetz, L. (2019, July). Performance Analysis of Mobile Cross-platform Development Approaches based on Typical UI Interactions. In ICISOFT (pp. 40-48).	CPU usage, memory (RAM) usage
Wu, W. (2018). React Native vs Flutter, Cross-platforms mobile application frameworks.	Frame rate (FPS), file read/write
Ciman, M., & Gaggi, O. (2017). An empirical analysis of energy consumption of cross-platform frameworks for mobile development. <i>Pervasive and Mobile Computing</i> , 39, 214-230.	Energy consumption

## Chapter 3

---

# Research Questions and Method

The answer of the first research question will aid the development of the remaining research questions and sub-questions. This was due to limited consensus as well as the quality of the sources not being verified during the selection process, hence it was decided that a proper mapping study would be performed. which through statistical means can verify that the selected performance metrics and tasks would be relevant for any selected cross-platform framework being compared to a native one.

### 3.1 Research Questions

**RQ1: How is performance evaluated for mobile app development frameworks in state of the art?** To answer this question, a mapping study was performed which has its own section in this thesis. In that section, the selected metrics and tasks for the remaining research questions were based on the number of occurrences found in papers which passed the inclusion into the study.

**RQ1.1: Which metrics are used to quantify performance?** For this research question, the aim was to identify relevant performance metrics so that the remaining research question **RQ2** and its sub-question could be specified in more detail, while also allowing for more statistically backed claims to be made during evaluation of those questions.

**RQ1.2: Which tasks are used to evaluate an application's performance?** The aim of this research question is to identify relevant tasks which will be used in tandem with the selected performance metrics to evaluate **RQ2**. It is also similar to **RQ1.1** in the sense that it helps answer a sub-question of **RQ2**.

**RQ2: How does the performance of the native and .NET MAUI apps compare?**

For this question to be answered, each framework will be used to build an application for evaluation purposes. However, because the first research question also needed to be answered before this question could add any sub-questions. Once **RQ1** was answered, each sub-question was defined containing one performance metric as its focus, and it was answered after the empirical part of this thesis. To answer this question and its sub-questions, data was gathered through an empirical experiment, and based on the results a conclusion was made regarding to what degree performance differs with the aid of hypothesis testing.

**RQ2.1: How does the CPU usage of native and .NET MAUI applications compare?**

**RQ2.2: How does the memory (RAM) usage of native and .NET MAUI applications compare?**

## 3.2 Mapping Study

The structure of this mapping study adheres to the guidelines by Petersen et al. [27] and Kitchenham et al [19]. For some practical applications for literature studies, the guidelines proposed by Montgomery et al. [25] were also used as a reference. The article search is presented in Section 3.2.1, the article selection in Section 3.2.2, and finally the data extraction and mapping is presented in Section 3.2.3.

### 3.2.1 Article Search

The strategy which was utilized for identification of research was a database search one. The databases of interest for this study were those which publish high quality studies and are commonly used in software engineering literature studies [25]. The selected databases were ACM Digital Library<sup>1</sup> and Scopus<sup>2</sup>.

To utilize the knowledge gathered from the Chapter 2 of this thesis, the search-string which would be used in these databases was based on the frequently appearing concepts and keywords previously mentioned. The keywords in this case were “cross-platform”, “native”, “mobile”, and “performance”, but to create a more detailed search-string, synonymous words as used in the related work were included to increase the recall of the search strategy. The synonyms for each keyword can be seen in Table 3.1.

Concept	Synonyms
Cross Platform	"cross-platform", "cross platform", "multiplatform", "multi-platform"
Native	"native", "single platform", "single-platform"
Mobile	"mobile", "smartphone", "phone"
Performance	"performance"

Table 3.1: Synonyms for keywords

Based on the list of keywords and synonyms in Table 3.1, a search-string was created for both ACM and Scopus (see Table 3.2). The search-strings were designed to only target the title, abstract and keywords of a study to limit the results and minimize false positives. The motivation for only selecting to target those parts during search were the trade-offs in terms of precision and recall, in cases where more targets of the search were added, it produced a lot of false positives for both data bases during piloting of the search strings.

Furthermore, a quasi-gold standard was created from a selected set of relevant and representative studies. This quasi-gold standard could then be used to evaluate the quality of the search-string, by ensuring that each of the studies in the quasi-gold

<sup>1</sup><https://dl.acm.org/search/advanced>

<sup>2</sup><https://www.scopus.com/>

standard appear in the combined results of both databases after the search-string has been applied. Assuming that there were studies in the quasi-gold standard not appearing in the results, the search-string would be re-evaluated.

Database	Search string
ACM Digital Library	((title:(native OR "single platform" OR "single-platform") OR abstract:(native OR "single platform" OR "single-platform") OR Keyword:(native OR "single platform" OR "single-platform")) AND (title:("cross-platform" OR "cross platform" OR multiplatform OR "multi-platform") OR abstract:("cross-platform" OR "cross platform" OR multiplatform OR "multiplatform") OR Keyword:("cross-platform" OR "cross platform" OR multiplatform OR "multi-platform")) AND (title:(mobile OR smartphone OR phone) OR abstract:(mobile OR smartphone OR phone) OR Keyword:(mobile OR smartphone OR phone)) AND (title:performance OR abstract:performance OR Keyword:performance))
Scopus	TITLE-ABS-KEY ( ( "native" OR "single platform" OR "single-platform" ) AND ( "cross-platform" OR "cross platform" OR "multiplatform" OR "multi-platform" ) AND ( "mobile" OR "smartphone" OR "phone" ) AND "performance" )

Table 3.2: Search strings used in databases

In total, the resulting number of studies acquired through both databases were 116. However, the actual number of studies which could move on to the next step differed due to duplicates. The actual final number of studies once duplicates had been removed were 94.

### 3.2.2 Article Selection

Due to the nature of this thesis as well as its goal, inclusion and exclusion criteria were established to ensure only relevant primary studies were considered in the mapping study. Some of the criteria were decided on solely based on the goal of the study, but some had to be established to minimize the number of false positives which moved on the next step of the study, such as ensuring that the language is accessible for the author of this thesis, which is a common strategy in Software Engineering (SE)-related literature studies [25]. The inclusion and exclusion criteria used throughout this entire study can be found in Table 3.3.

Inclusion criteria	Exclusion criteria
Compares the performance of cross-platform mobile application(s) with single-platform mobile application(s) through empirical work	Study is from a year before 2007 due to the Android SDK being released Nov. 2007
Specifies the performance metric(s) used when comparing cross-platform mobile application(s) to single-platform application(s)	Discusses cross-platform applications but does not evaluate their performance through empirical work, instead evaluates accessibility/standards/design choices
	The study is in a completely different language from English (such as an Asian language), and not even the abstract is English
	Study is not peer-reviewed

Table 3.3: Inclusion and exclusion criteria for studies

The next step was to apply the inclusion/exclusion criteria in Table 3.3 to the main studies found during Section 3.2.1. This was done by the author of this thesis, manually going through each study, and excluding any paper which invalidate at least 1 of the inclusion/exclusion criteria. The criteria were cycled through for each study, starting with inclusion criteria 1 and ending with exclusion criteria 4. Finally, all the decisions related to whether or not to include or exclude a study was recorded in a table.

The sections of the studies where the criteria were applied to were limited to the “Abstract”, “Introduction”, “Method” and “Result” sections, to maximize efficiency while minimizing the amount of possible false positives by focusing solely on the title or something of the sort. In case the layout of the paper was irregular, the focus would be solely on the “Abstract” and the “Result” section.

Once the set of studies had been put through the inclusion/exclusion criteria, 14 out of the 94 studies were successfully marked as included in the study.

### 3.2.3 Data Extraction and Mapping

During this stage of the study, data was extracted from the studies which passed the article selection phase. The data which is relevant to extract thus needed to be decided on before continuing towards the finalization of the study where the results are gathered.

The studies which were included were read through twice. During the initial reading groups for extracting performance metrics and tasks will be created to make sure that all relevant groups could be represented properly. Only during the second reading did any data relevant for this section get extracted. To address **RQ1**, the data extracted from an included study had to be labeled according into one of those groups depending on if the data was considered a performance metric or task.

The format for the data related to inclusion or exclusion was simply saved as a single cell only containing the first inclusion/exclusion criteria invalidated. According to the guidelines for the mapping study, the researcher could group this information (or "map" it) with bubble plots for clarity. However, because of the low sample size in this study, it was deemed irrelevant.

Once a performance metric or a task had been identified for an included study, it was manually added to the data sheet in a cell corresponding to the group it was identified as belonging to. **RQ1** was thus to be answered by selecting the most common metrics and a random task from each group of tasks created.

### 3.3 Empirical Study

The empirical study was designed according to the guidelines by Wohlin et al. [34] and reported according to the guidelines of Jedlitschka et al. [18].

#### 3.3.1 Goals

The goals of the study can be summarized as follows:

*Goal 1: Analyze the difference in CPU usage between an application created in .NET MAUI and one created targeting only the native Android framework For the purpose of seeing if the differences are statistically significant With respect to common tasks for performance comparison.*

*Goal 2: Analyze the difference in RAM usage between an application created in .NET MAUI and one created targeting only the native Android framework For the purpose of seeing if the differences are statistically significant. With respect to common tasks for performance comparison.*

#### 3.3.2 Experimental Units

The experimental units in the case of this experiment are the applications developed using the different frameworks. This is because the independent variable - the two treatments (the frameworks .NET MAUI and native) - are applied to these applications. Since the experimental units solely are the applications, the effects of implementing the same tasks with different frameworks can be observed.

#### 3.3.3 Experimental Materials

The tools utilized when creating both applications in this case were Microsoft Visual Studio [23] with C# for the .NET MAUI application, and Android Studio [10] with Kotlin [9] for the native application. The documentation followed when implementing both applications were the standard documentation for .NET MAUI [22] and the native framework [11]. To find similar functioning UI elements, one was selected in the native framework and then cross-referenced for functionality with the one selected for the .NET MAUI framework. Furthermore, each application was evaluated

on the same PC and emulator, as for which the specifications can be found in Appendix A.1. This was done in order to minimize the effect of confounding factors. For evaluation of the performance, both applications were built for release (to minimize the performance overhead caused by debug-properties in a debug build) and profiled using the Android Studio profiler with “low overhead” profiling. For full replication, refer to the replication package of this thesis <sup>3</sup>

### 3.3.4 Tasks

The tasks were selected based on the results from the performed mapping study. These tasks were found to represent one of each task-group identified during the study, allowing for a more complete (and research-backed) coverage of potential task-groups when testing metrics. The details of each task can be seen in Table 3.4.

Name of task	Implementational details of task
Mathematical calculation	Utilize multithreading to approximate the value of PI over 1 000 000 iterations. Perform the same calculation 50 times.
1000 item list	In a separate tab from the start-screen of each application, a list containing 1000 items is initialized with numbers in linear fashion from 1 to 1000. The task is done once scrolling of the list has been performed for 5 continuous seconds.
Geolocation	Using the geolocation manager for each respective framework, the location of the emulator was requested exactly once. The granularity of the location request was set as “fine” for both frameworks.
Change Page	From the initial start-screen, a button is pressed to go to a second page (activity). The task is completed once the page has been initialized and is ready for input (signaled by a Toast).
Download file	A file containing exactly 1 Megabyte (MB) is downloaded asynchronously. To make it more similar for both frameworks, a direct HTTP connection was established instead of using download-managers providing different functionality for each framework.
File system	An empty .txt-file is created. 1MB of data is then written to it. The content of the file is then read (its bytes). Finally, the file is deleted, and the task is finished.

Table 3.4: Task details

All of the selected tasks were implemented in a fashion recommended by the standardized guidelines for performing such a task according to the standard documentations mentioned previously. Each task was accessible through the start-screen

<sup>3</sup><https://github.com/LukasPalmq/MAUINativeAndroidEval>

of each respective application. The flow of accessing the tasks can be seen in Figure 3.1.

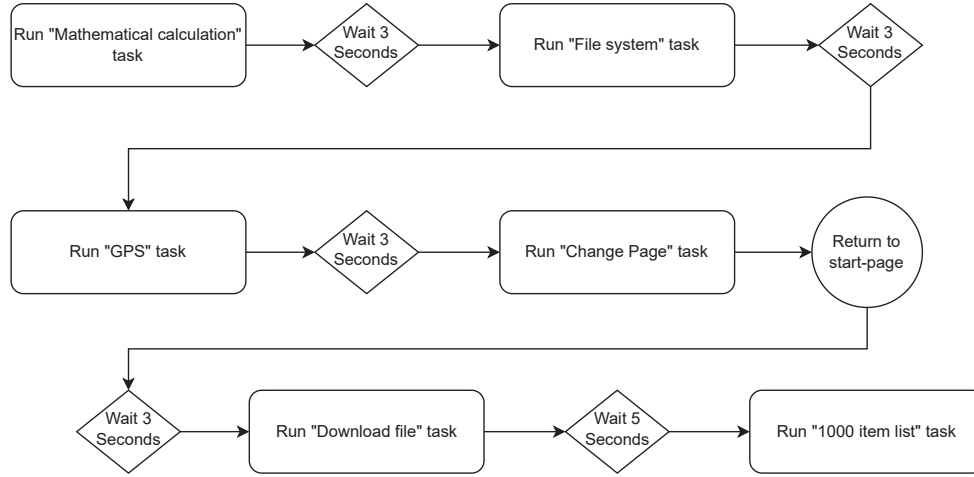


Figure 3.1: Task flow

### 3.3.5 Hypothesis, Parameters and Variables

To answer the goals of the experiment, *Goal 1* and *Goal 2*, several hypotheses had to be established. These null hypotheses were designed according to the guidelines by Wohlin et al. [34]. In the following,  $n$  denotes the native application and  $m$  denotes the .NET MAUI application. Each task related to answering a specific goal will have its own hypothesis and thus result.

Furthermore, the effect size [13,20] of each task will be stated after the associated hypothesis test. The effect size measures the strength/magnitude of a difference for a specific task between the applications. This experiment will use either *Cohen's Delta* or *Cliff's Delta* to calculate the effect programmatically. This selection depends on whether or not the data is normally distributed (parametric, then Cohen's), or not (non-parametric, then Cliff's). The effect size ranges between 1 and -1. The logic behind this varies slightly between Cohen's and Cliff's. Cohen's is calculated by taking the difference between two means and dividing by the pooled standard deviation, hence an effect size of 1 (considered large) means that the first framework's values have a higher mean value than the second group by a whole standard deviation (and -1 would mean the opposite). In Cliff's case, an effect size of 1 means that the first framework's values has higher values than every vaue recorded for the second framework (and -1 means the opposite of this).

In other terms, every task related to answer *Goal 1* will have its own hypothesis established in terms of CPU usage, as well as their own effect size. The same is true for *Goal 2* but for memory (RAM) usage.

For *Goal 1*, the following hypothesis is established:

$$C0 : CPU_n = CPU_m$$

$$C1 : CPU_n! = CPU_m$$

$C0$  assumes that there is no difference between the applications using the different frameworks when it comes to CPU usage (in %). Furthermore,  $C1$  assumes that there is a difference between the different frameworks when it comes to CPU usage (in %).

The same is then done when establishing *Goal 2*:

$$R0 : RAM_n = RAM_m$$

$$R1 : RAM_n! = RAM_m$$

This time the difference being that the RAM usage is measured in MB instead of %. The RAM here will also not refer to not the total amount of RAM used, but rather: *(RAM used in MB after the task has been performed) – (RAM used in MB before the task has been performed)*.

As for the variables of the experiment, they are defined as such:

**Independent:**

The implementations of the frameworks being evaluated, in this case an application made native and one using .NET MAUI.

The hardware used in the experiment (physical, emulator).

**Dependent:**

The result of each task in terms of the metric, or in other words the CPU and memory (RAM) usage for each task while running each application.

The hypotheses are accepted or rejected based on the resulting p-value of a statistical test of independence. Depending on if the sample data is normally distributed or not (determined via the Shapiro-Wilk test), a parametric (unpaired T-test) or a non-parametric (Mann-Whitney U test) was used to answer each hypothesis test. The same is done for the effect size, by using the Shapiro-Wilk test the data can be determined whether it is parametric or not. The level of significance for at what p-value we accept or reject the hypothesis test was set as:  $\alpha = 0.05$ .

### 3.3.6 Procedure

The flow of the procedure utilized during this experiment to collect the data can be seen in Figure 3.2. In the figure, “Perform task flow” refers to the contents of Figure 3.1.

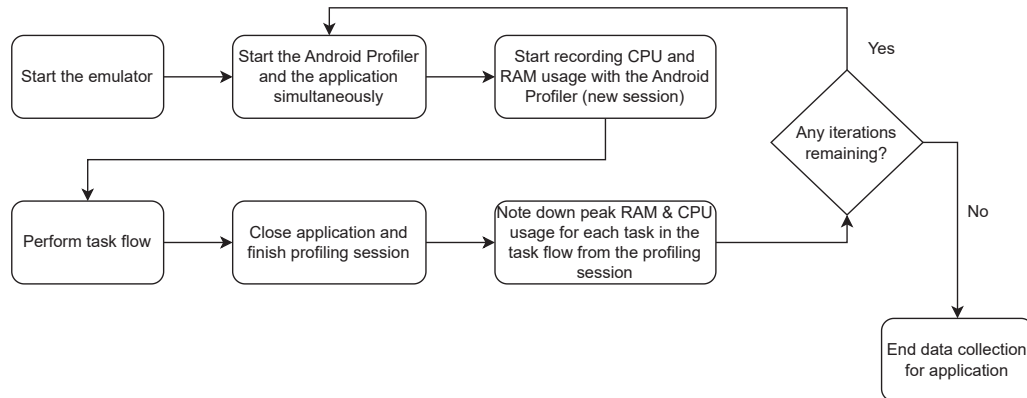


Figure 3.2: Procedure flow

Because the data was collected manually, data transformation happened in the form of the collected data being rounded to a maximum of 1 decimal due to the Android Profiler imposing such limitations in saved sessions.

### 3.3.7 Deviations

The original plan was to automatically collect this data over several hundreds of iterations using a different tool-set. However, because of technical limitations, as well as limited time to finalize the thesis, there was no time to further research the subject and decision to manually collect the data using the Android Profiler. As a byproduct of this decision, only the peak data could be collected for each metric instead of the average being calculated automatically for each profiling session. Nevertheless, the number of observations collected during this experiment (20), resembles what related works using similar tooling report from the mapping study.

In the original setting for this thesis, the plan was to originally include another application targeting the iOS native framework to create a more “complete” sense of cross-platform comparison with .NET MAUI. However, due to technical limitations in terms of hardware and implementation time being an issue, the decision was made to only focus on Android.

The results of the mapping study will be presented in Section 4.1, and the empirical study will in turn be presented in Section 4.2. Because **RQ2** has sub-questions, the results of the empirical study will be grouped depending on their correlation to each question. For the files which hold the results of both the mapping study, as well as the files related to replicating the empirical study, again refer to the replication package <sup>1</sup>.

### 4.1 Mapping Study

The result of this study will aid in answering **RQ1**, as well as formulating the basis for **RQ2.1** and **RQ2.2**. The groupings for all results presented in this section are based on the two groups created to abstract performance metrics and tasks into.

The sub-group in Table 4.1 focuses on the performance metrics, and a study could be labeled as belonging to multiple of these sub-groups depending on what metrics that were evaluated in the sections of interest previously described in Section 3.2.2. The condition for the labeling to happen was that the “Result”-section of the study mentions that data was collected with that specific metric in mind to conclude differences in performance.

The sub-group in Table 4.2 focuses on the tasks which the studies utilized to conclude the results of the performance metric comparisons presented.

CPU	RAM	GPU	NETWORK	SIZE	FPS	BATTERY	RESP. TIMES
-----	-----	-----	---------	------	-----	---------	-------------

Table 4.1: Performance metrics

---

<sup>1</sup><https://github.com/LukasPalmq/MAUINativeAndroidEval>

Task-Calculations	Task-ListScrolling	Task-Sensors	Task-AppRelated	Task-Network	Task-Filesystem
Mathematical calculation [8]	Infinite List [21] [3] [15]	Geolocation [33] [6] [5] [7]	Change page [33] [32] [3] [15] [4]	Download file [16]	File system (get/set data from/of random file) [3] [7] [5]
Animation [4]	1000 item list [14]	Camera [21] [6]	Minimized Usage [32]	Make API request [3] [28]	Get data from Contacts [5] [7]
		Access accelerometer [6] [5] [7]	Launching App [14] [33] [32] [16]		
		Access sound [6] [7]	Overall usage when moving around in-app [33] [32] [15] [4] [16]		
		Access temperature of components [28]	Overall usage when nothing is happening [6] [28]		
			Start the app and leave it for 3 minutes [12]		

Table 4.2: Tasks

The most frequently appearing performance metrics amongst the included studies can be seen in Figure 4.1. The most common performance metric(s) occurring are thus CPU and memory (RAM) usage.

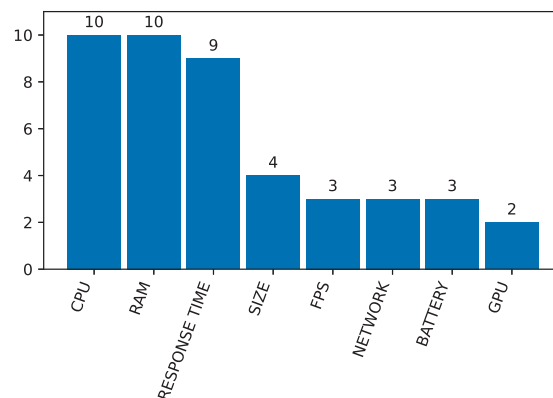


Figure 4.1: Performance metrics by occurrence in included studies.

The most frequently occurring tasks amongst the included studies can in turn be seen in Figure 4.2. From these groups, the most common tasks are put into the “AppRelated” group, followed by the tasks put into the “Sensor” group.

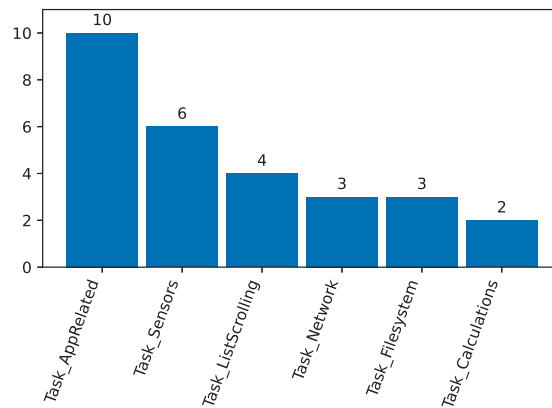


Figure 4.2: Most frequently occurring tasks in included studies.

## 4.2 Empirical Study

The results of this study will aid in answering **RQ2.1** and **RQ2.2**, in turn allowing for a conclusive answer of **RQ2**. For each figure presented in this section, “Native” denotes the native application, and “.NET MAUI” denotes the .NET MAUI application.

The graphs are of the type "matplotlib.pyplot.boxplot"<sup>2</sup>, and what each detail means will be explained shortly: The green triangle refers to the arithmetic mean of the data. The box itself represents the quartiles and median confidence intervals. The "whiskers" or vertical lines at the start and end of each box represents the most extreme (but not outlier-classified) data points. The "fliers" or outliers are represented by circles. For more detailed information, please refer to the documentation for this specific graph-type.

The figure is followed by the result of the hypothesis test connected to that specific task and metric.

### 4.2.1 CPU usage for Mathematical calculation task

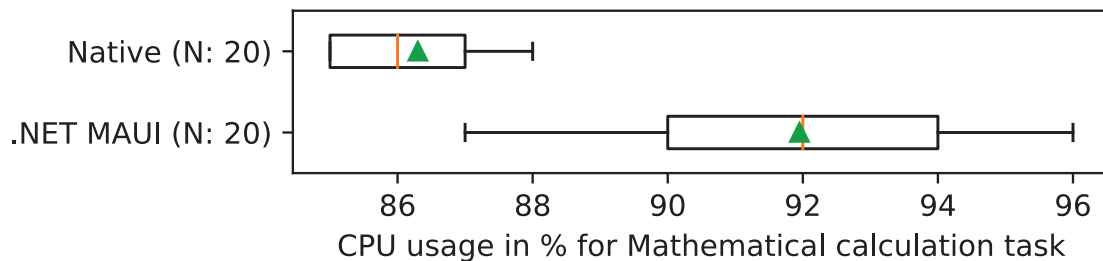


Figure 4.3: The peak CPU usage in % for Mathematical calculation task

According to the hypothesis test (Mann-Whitney U test), there is a statistically significant difference between the CPU usage in the “Mathematical calculation” task

<sup>2</sup>[https://matplotlib.org/3.1.1/api/\\_as\\_gen/matplotlib.pyplot.boxplot.html](https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.boxplot.html)

(p-value: 1.725e-07). The difference has an effect size of 0.96 (Cliffs delta).

### 4.2.2 CPU usage for 1000 item list task

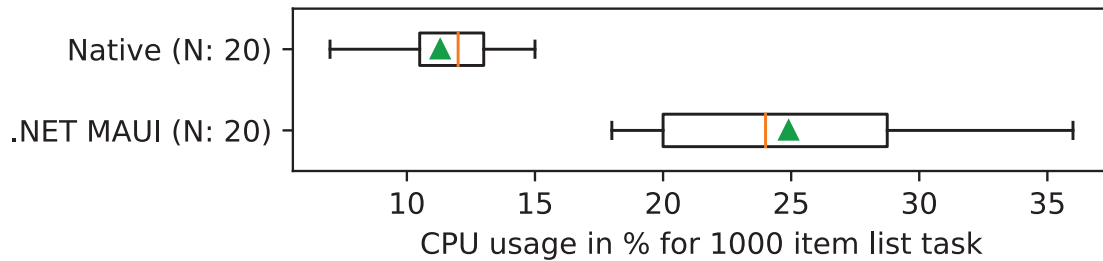


Figure 4.4: The peak CPU usage in % for 1000 item list task

According to the hypothesis test (Mann-Whitney U test), there is a statistically significant difference between the CPU usage in the "1000 item list" task (p-value: 6.215e-08). The difference has an effect size of 1.0 (Cliffs delta).

### 4.2.3 CPU usage for Geolocation task

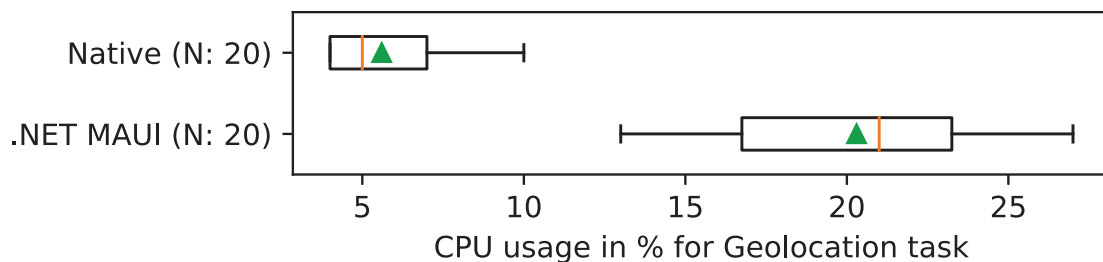


Figure 4.5: The peak CPU usage in % for Geolocation task

According to the hypothesis test (Mann-Whitney U test), there is a statistically significant difference between the CPU usage in the "Geolocation" task (p-value: 5.604e-08). The difference has an effect size of 1.0 (Cliffs delta).

#### 4.2.4 CPU usage for Change Page task

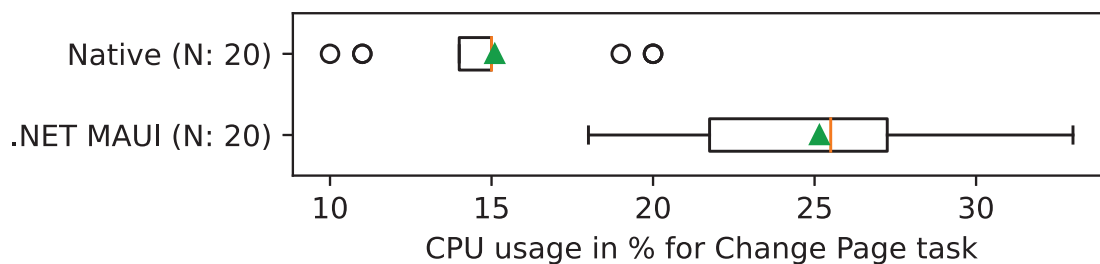


Figure 4.6: The peak CPU usage in % for Change Page task

According to the hypothesis test (Mann-Whitney U test), there is a statistically significant difference between the CPU usage in the "Change Page" task (p-value:  $1.695e-07$ ). The difference has an effect size of 0.96 (Cliffs delta).

#### 4.2.5 CPU usage for Download file task

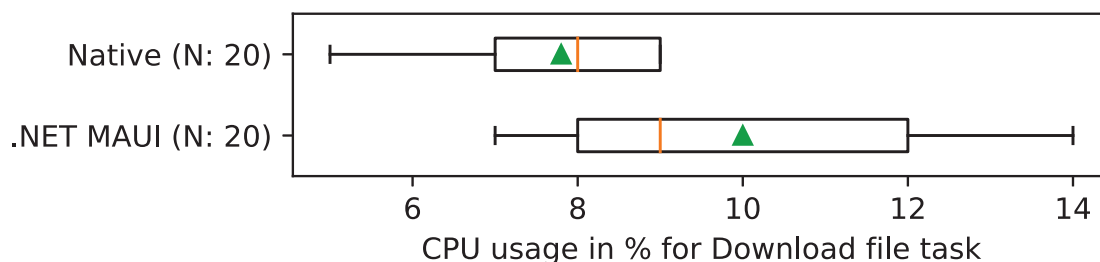


Figure 4.7: The peak CPU usage in % for Download file task

According to the hypothesis test (Mann-Whitney U test), there is a statistically significant difference between the CPU usage in the "Download file" task (p-value:  $0.0007374$ ). The difference has an effect size of 0.61 (Cliffs delta).

#### 4.2.6 CPU usage for File system task

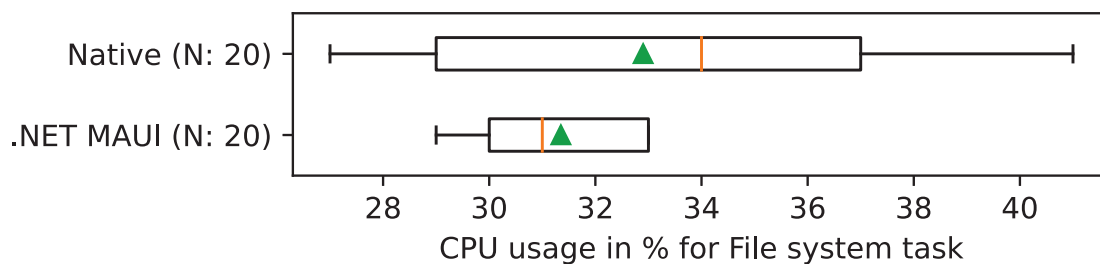


Figure 4.8: The peak CPU usage in % for File system task

According to the hypothesis test (Mann-Whitney U test), there is no statistically significant difference between the CPU usage in the "File system" task (p-value: 0.353). The difference has an effect size of -0.17 (Cliffs delta).

#### 4.2.7 RAM usage for Mathematical calculation task

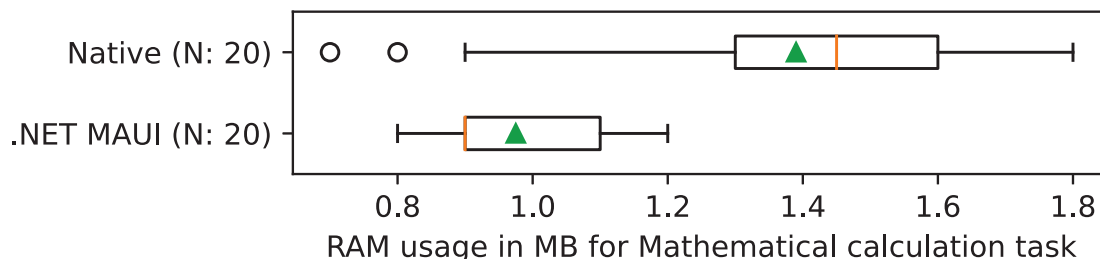


Figure 4.9: The peak RAM usage in MB for Mathematical calculation task

According to the hypothesis test (Mann-Whitney U test), there is a statistically significant difference between the RAM usage in the "Mathematical calculation" task (p-value: 0.0002546). The difference has an effect size of -0.67 (Cliffs delta).

#### 4.2.8 RAM usage for 1000 item list task

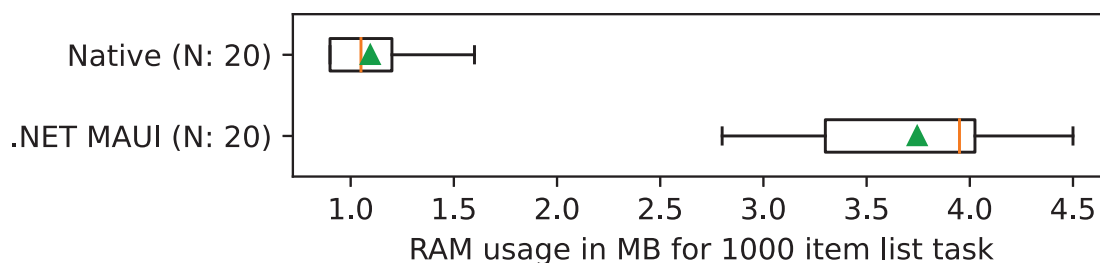


Figure 4.10: The peak RAM usage in MB for 1000 item list task

According to the hypothesis test (Mann-Whitney U test), there is a statistically significant difference between the RAM usage in the "1000 item list" task (p-value: 5.653e-08). The difference has an effect size of 1.0 (Cliffs delta)

### 4.2.9 RAM usage for Geolocation task

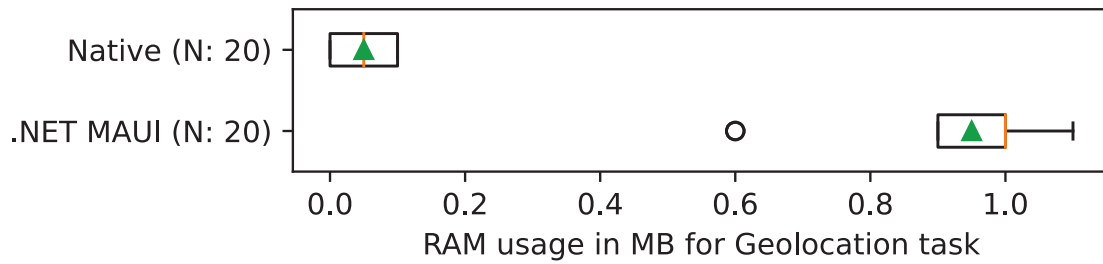


Figure 4.11: The peak RAM usage in MB for Geolocation task

According to the hypothesis test (Mann-Whitney U test), there is a statistically significant difference between the RAM usage in the "Geolocation" task (p-value: 3.452e-08). The difference has an effect size of 1.0 (Cliffs delta).

### 4.2.10 RAM usage for Change Page task

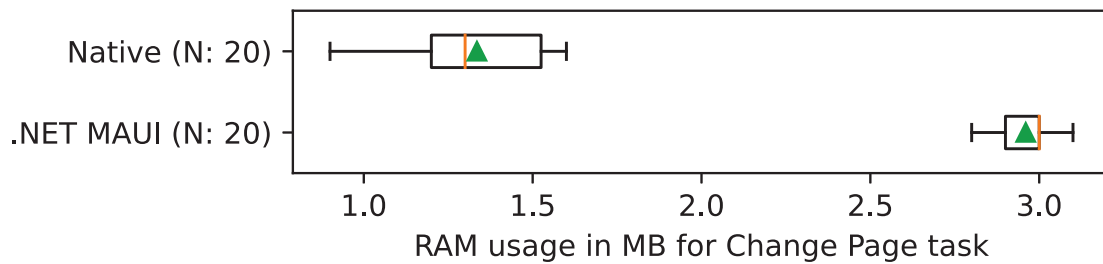


Figure 4.12: The peak RAM usage in MB for Change Page task

According to the hypothesis test (Mann-Whitney U test), there is a statistically significant difference between the RAM usage in the "Change Page" task (p-value: 4.988e-08). The difference has an effect size of 1.0 (Cliffs delta)

### 4.2.11 RAM usage for Download file task

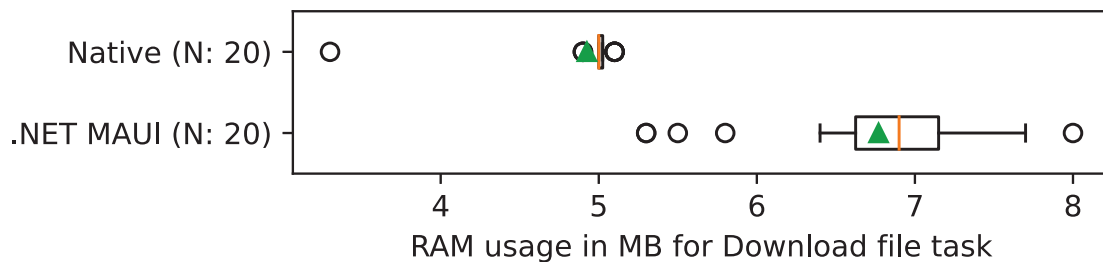


Figure 4.13: The peak RAM usage in MB for Download file task

According to the hypothesis test (Mann-Whitney U test), there is a statistically significant difference between the RAM usage in the "Download file" task (p-value: 4.661e-08). The difference has an effect size of 1.0 (Cliffs delta).

#### 4.2.12 RAM usage for File system task

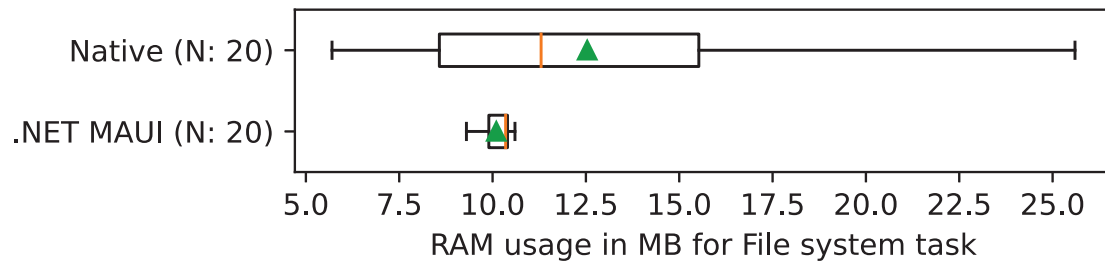


Figure 4.14: The peak RAM usage in MB for File system task

According to the hypothesis test (Mann-Whitney U test), there is no statistically significant difference between the RAM usage in the "File system" task (p-value: 0.3348). The difference has an effect size of -0.18 (Cliffs delta).

### 5.1 Mapping Study

In this section, the results will be interpreted in the context of the research question **RQ1**. Also, any implications found during the study will be evaluated in relation to the preliminary results found in the related works.

#### 5.1.1 Interpretation of Mapping Study results

**RQ1.1: Which metrics are used to quantify performance?** Based on the results presented in Figure 4.1 of Section 4.1, the two most commonly occurring performance metrics are CPU and memory (RAM) usage. As can also be observed, the margin was not necessarily significant when comparing say CPU and memory (RAM) with the Response Time metric, however because of the scope being limited for this study only two metrics were chosen.

The results showed that the most frequently occurring performance metrics were CPU and memory (RAM) usage. Judging by the quasi-gold standard, 3 out of 4 studies were using CPU and memory (RAM) as a performance metric. Thus, the results seem to line up with what the quasi-gold standard was insinuating.

**RQ1.2: Which tasks are used to evaluate an application's performance?** Based on a combination of the results presented in Table 4.2 and Figure 4.2, the most commonly used groups as well as what they consist of can be observed. The majority of the tasks found usually were related to the application directly, such as measuring the size of the application post-installation or measuring the time it took to open a new page inside of said application.

**RQ1: How is performance evaluated for mobile app development frameworks in state of the art?** From the results found in **RQ1.1** and **RQ1.2**, the answer to this research question was found to be a combination of the performance metrics CPU and memory (RAM) usage when measured on a set of arbitrarily selected tasks from the groups identified in Table 4.2.

#### 5.1.2 Chosen metrics and tasks

Based on the results and discussion presented, the most logical choices are the metrics CPU and memory (RAM) usage since both are utilized most while also being used the

same number of times according to the results of the implemented study. Hence, the metrics establishing research sub-questions **RQ2.1** and **RQ2.2** could be confirmed to be relevant based on the results of the mapping study performed.

As for the task selection, one task in each group of tasks were selected arbitrarily based on implementational difficulty for both frameworks and how hard they were to measure using the Android Profiler. The details of the selected tasks can be seen in Section 3.3.4.

Thus, the basis for selection of metric for the research sub-questions **RQ2.1** and **RQ2.2** were established.

## 5.2 Empirical Study

For this section, the results of the empirical study will be discussed in the context of answering **RQ2.1** and **RQ2.2**. Furthermore, the hypothesis tests and effect sizes results for each task will be taken into consideration during this process. Because the data was all non-parametric, only Cliff's delta was used to calculate the effect size. The effect size was calculated by having .NET MAUI as the "first framework", meaning that if the effect size is 1, then every recorded value for that task in .NET MAUI had a higher value than the ones for the native implementation of the task (and the opposite is true for -1).

In the case of the related work measuring a type of mathematical calculation task, there was no comparison in terms of CPU (or RAM) usage [8], so it is difficult to synthesize those results. In addition,

### 5.2.1 Interpretation of Empirical Study results

#### **RQ2.1: How does the CPU usage of native and .NET MAUI apps compare?**

Based on the results presented in Section 4.2, the result presented that in 5 out of 6 tasks for CPU (%) usage, the native implementation outperformed the cross-platform implementation.

In addition, for each task, the effect size calculated in terms of quantitative outcomes was noted, and in the majority of cases it was 1.0 according to Cliff's delta calculation. This further shows that there in fact was a measurable difference between the frameworks.

In general, from what was observed in the accepted mapping studies implementing the same type of task that this thesis did, RAM usage was considerably lower across the board in favor of the native applications [14–16, 33] except for the file related tasks [5] which was also found to be the exception of this thesis which also showed no statistically significant difference.

Possible reasons for this difference between the frameworks could be due to different run-time environments and standards for Kotlin and C. Themes such as concurrency models<sup>1 2</sup> could be working differently behind the scenes, leading to a bit

---

<sup>1</sup><https://kotlinlang.org/docs/multiplatform-mobile-concurrency-and-coroutines.html>

<sup>2</sup><https://learn.microsoft.com/en-us/dotnet/api/system.threading.tasks.task>

more varied and unpredictable results for tasks such as the mathematical calculation one where .NET MAUI performed worse.

In the one case where the .NET MAUI arguably outperformed the native framework, for the file-related task, it could be further theorized that it once again has a similar underlying cause. This being that the underlying implementation of a function used for file-management in Kotlin internally was running blocking, while the .NET MAUI one was asynchronous. This could potentially cause interference from the GUI thread, while the asynchronous one is not affected, also leading to a potentially larger spread in recorded values for the native framework results.

This, combined with the fact that all but one of the hypothesis tests (the one for the File system task) indicated that there was a significant statistical difference between the frameworks in terms of CPU usage, suggests that there is indeed a significant statistical difference, assuming that the significance threshold is as established for this thesis ( $\alpha = 0.05$ ). It is also worth noting that the only test which .NET MAUI came out ahead in was the one not indicating any significant statistical difference.

### **RQ2.2: How does the memory (RAM) usage of native and .NET MAUI apps compare?**

As for memory (RAM) usage, based on the results presented in Section 4.2, the result presented that in 4 out of 6 tasks for memory (RAM) usage, the native implementation outperformed the cross-platform implementation. In addition, only one of the hypothesis tests indicated that any significant statistical difference was found in favor for the .NET MAUI framework, in this case the one correlating with the Mathematical calculation task. This could again be related to the previously discussed differences in concurrency, perhaps showing itself as a trade-off between CPU and RAM usage. Since there are no related works discussing this metric and task combination, it is thus purely speculative.

What was observed in this case from the accepted mapping studies implementing the same type of task that this thesis did, was that RAM usage was considerably lower across the board in favor of the native applications [14–16, 33]. Once again, except for the file related tasks [5] which was also found to be the exception in terms of RAM for this thesis, showing .NET MAUI with a slight (although not statistically significant) advantage.

For the specific tasks where .NET MAUI came out ahead with the mathematical calculation task, a study from the mapping study performed [8] showed tendencies of phone hardware impacting performance when comparing native to a cross-platform framework. While the only metric in that specific study was response time, it could be theorized that similar results could have occurred in this study with these chosen metrics.

Again, there was one test (for the File System task) with no significant statistical difference, and the case was also the same in the sense that .NET MAUI had a very slight advantage here, similar to the CPU hypothesis test for the same task.

## 5.3 Summary

In general, based on the results of both **RQ2.1** and **RQ2.2**, there is a statistically significant difference in performance for both CPU and memory (RAM) usage in favor of the native framework. This can be further supported by the fact that 5/6 hypothesis tests for **RQ2.1** and 4/6 for **RQ2.2** were rejected in favor of the native framework. With this, the Goals of the experiment *Goal 1* and *Goal 2* can safely be considered answered, as these results statistically support that there is a difference in both CPU and memory (RAM) usage.

Furthermore, in Section 4.1, the applications evaluated in the included studies that were built using the native framework were in many cases slightly ahead in terms of the performance metrics used in each respective study when compared to another cross-platform framework, which is in line with the results observed in the experiment of this thesis. However, something that does not align with related works are that the previous studies included in either Chapter 2 or Section 4.1 of the thesis did not directly consider the .NET MAUI framework. Instead, the studies focus more on different cross-platform frameworks, so it is difficult to say whether this is directly related to .NET MAUI or simply a byproduct of newer Android software versions colliding with differently implemented wrappers or similar.

Also, while the hypotheses in most cases were rejected and deemed statistically significant enough, it is also arguable whether the difference in a scale of a few MB or percentage units truly are significant on newer hardware. For instance, as of writing, the average available memory (RAM) on mid- to top-tier smartphones range from 6-12GB, effectively rendering a difference of a few MB to not mean a lot in real world usage. [29]

## 5.4 Validity Threats

Finally, threats to validity following the categorization by Wohlin et al. [34] and Montgomery et al. [25] are acknowledged and discussed.

### 5.4.1 Conclusion Validity Threats

*Low statistical power:* Because the data collection process of the experiment performed was manual, and the tools used did not allow for any form of automation, a low sample size of 20 reading (per task and application) was created. While it could have been mitigated somewhat by doing more readings, the studies found during the mapping study used between 10 and 30 readings per task as well, suggesting that 20 resembles the state of the art.

*Violated assumptions of statistical tests:* This threat was mitigated through Shapiro-Wilk test, which was done programmatically for each observation to see if the data sample is normally distributed or not before selecting the appropriate hypothesis test method.

*Reliability of measures:* Because each observation was recorded manually by the experimenter, there is a chance that the observation was made incorrectly with regards to accuracy as well as the Android Profiler in this case possibly providing the

wrong data. This was mitigated by recording each profiling session and slowly stepping through each recorded datapoint; however, the potentially reduced accuracy of each reading cannot be mitigated further due to the tooling used.

*Reliability of treatment implementation:* There is also the threat that the implementation of each individual task caused the disparity in performance when measuring the performance metrics for each framework of this thesis. To minimize this threat, the task implementations were done by the same individual, and the respective documentation of each framework was consulted when replicating functionality from one application to another. Furthermore, it is possible that optimization was not taken into account enough. This was mostly because the study focused on similarity over optimization for the tasks.

### 5.4.2 Internal Validity Threats

*History:* To minimize the possibility of factors such as time affecting the results of the experiments, the data was collected in one session by the experimenter. However, because the collection also took time, it is possible that something occurred between the data collection for both applications was finished.

*Maturation:* The experiment was designed in such a way that the application and profiler is restarted between each iteration of data collection. This minimized any disparities caused by potentially allowing the application to use more resources over time, influencing the data collected for future iterations of the experiment.

*Selection:* The subjects (or tasks) of the experiment were selected based on the result of the conducted mapping study in Section 4.1, hence the tasks should have adequate motivation behind each one. However, it is possible that the tasks used for assessment have changed in recent years and that the mapping study did not take this into account, as some of the sources were not current at the time of writing. It is also possible that the selected metrics were incompatible with the tasks selected, which was something not measured during the mapping study.

### 5.4.3 Construct Validity Threats

*Inadequate preoperational explication of constructs:* This threat was mitigated by ensuring that the goals of the experiment were outlined before experimentation started, as well as making sure that the intended data could be collected.

*Confounding constructs and levels of constructs:* It is possible that the experimenter's previous knowledge of native Android development could have positively affected the results of the native Android applications' results in the experiment. This was mitigated to the greatest extent possible by doing research on the other framework in the experiment before implementation started of each application.

*Lack of inter-rater agreement:* Since the process of extracting data in the mapping study, as well as including/excluding studies was done by a single researcher, it potentially introduces bias to the subjective tasks of article selection and data extraction. Ideally, several researchers should process each paper to maximize validity and ensure that the extracted results are as close to reality as possible. While a full labeling overlap has not been performed, this threat to validity was minimized by

having the supervisor of this thesis review a select sample of the subjective selection and extraction tasks.

#### 5.4.4 External Validity Threats

*Interaction of setting and treatment:* To ensure that the experiment was performed using up-to-date information as well as proper tooling, the respective standard documentation of each framework was consulted prior to the implementation of any application starting. This could for instance refer to the usage of the Android Profiler (suggested by Google themselves) instead of other tooling suggested in older studies found during the mapping study.

*Interaction of history and treatment:* It is possible that the task utilizing a network connection was affected by the time of day depending on when the data was collected. To mitigate this threat and make it as fair as possible for both applications taking part in the experiment, the data was collected as with as little delay as possible between both applications.

In this thesis, the performance of two frameworks for Android development, native and .NET MAUI, was investigated. The performance metrics measured were CPU usage in % and memory (RAM) usage in MB. The tasks used for measuring those metrics were a mathematical calculation, scrolling through a list consisting of 1000 items, geolocation, changing a page in the application, downloading a file, and finally performing file system (get/set) operations on a text-file. These metrics and tasks were derived through systematically investigating relevant literature and in turn is backed by a mapping study.

Based on the results of the experiment, the .NET MAUI framework performs significantly worse than the native framework of Android in terms of those performance metrics in the context of the experiment. The goals of the experiment, *Goal 1* and *Goal 2*, were answered clearly in favor of a statistically significant difference in both CPU and memory (RAM) usage. The results thus show a significant difference in performance when opting for cross-platform development, which affects decisions related to the trade-off between native and cross-platform applications in practice.

However, it is still unclear if the performance differs to a noticeable extent in real world usage because of the small sample size and implementational details.

As for the takeaways related to relative performance, because .NET MAUI is still in development and a relatively recent addition to available cross-platform frameworks, it is possible that the slight performance overhead observed in this thesis are even more close to negligible in the future. Looking past the significant differences in performance however, .NET MAUI needs further research to assess whether is it viable in terms of other factors.

To extend the comparison, future work should consider adding another platform to the experiment instead of only focusing on the comparison with native Android. The only realistic candidate for this is iOS, and in that case the native application for iOS could have been developed using Swift. By including iOS, the experiment could cover a much larger global mobile user base.

Furthermore, because of the implementational limitations imposed on this thesis, such as implementation time being limited, the applications were developed in a very “barebones” manner and may not represent actual usage in the industry. This is also partially caused by the tasks being chosen from a mapping study consisting only of research level studies. Also, it is possible that doing something as simple as changing the visual design of each application could drastically change the results, which could be interesting to look further into. Due to this, further case studies investigating more representative, industry-strength applications are necessary.

Another thing which could be done in future work is evaluating the differences between .NET MAUI and the native framework in terms of user experience or developmental decisions available for each framework. Alternatively, going even further down the path of performance by looking at optimization strategies for each framework is another option for future work within a similar region of research.

Finally, by finding a way to add automated data collection the accuracy of the experiments could possibly benefit greatly. Currently, the Android Profiler is the state of the art for performance metric collection according to Google (when it comes to Android development). However, the Android Profiler does not allow for automatic data collection, but rather relies on a visual system for performance readings.

---

## References

- [1] “Cross-platform frameworks,” Feb 2023, accessed: 2023-04-28. [Online]. Available: <https://kotlinlang.org/docs/cross-platform-frameworks.html>
- [2] “Native and cross-platform app development: how to choose?” Mar 2023, accessed: 2023-04-28. [Online]. Available: <https://kotlinlang.org/docs/native-and-cross-platform.html>
- [3] L. Barros, F. Medeiros, E. Moraes, and A. F. Júnior, “Analyzing the performance of apps developed by using cross-platform and native technologies.” in *SEKE*, 2020, pp. 186–191.
- [4] A. Biørn-Hansen, T.-M. Grønli, and G. Ghinea, “Animations in cross-platform mobile applications: An evaluation of tools, metrics and performance,” *Sensors*, vol. 19, no. 9, p. 2081, 2019.
- [5] A. Biørn-Hansen, C. Rieger, T.-M. Grønli, T. A. Majchrzak, and G. Ghinea, “An empirical investigation of performance overhead in cross-platform mobile development frameworks,” *Empirical Software Engineering*, vol. 25, pp. 2997–3040, 2020.
- [6] M. Ciman and O. Gaggi, “An empirical analysis of energy consumption of cross-platform frameworks for mobile development,” *Pervasive and Mobile Computing*, vol. 39, pp. 214–230, 2017.
- [7] L. Corral, A. Sillitti, and G. Succi, “Mobile multiplatform development: An experiment for performance analysis,” *Procedia Computer Science*, vol. 10, pp. 736–743, 2012.
- [8] L. Delia, N. Galdamez, L. Corbalan, P. Pesado, and P. Thomas, “Approaches to mobile application development: Comparative performance analysis,” in *2017 Computing conference*. IEEE, 2017, pp. 652–659.
- [9] A. Developers, “Android’s kotlin-first approach,” accessed: 2023-04-28. [Online]. Available: <https://developer.android.com/kotlin/first>
- [10] —, “Android studio,” Apr 2023, accessed: 2023-04-28. [Online]. Available: <https://developer.android.com/studio/>
- [11] —, “Developer guides,” May 2023, accessed: 2023-05-10. [Online]. Available: <https://developer.android.com/docs>
- [12] T. Dorfer, L. Demetz, and S. Huber, “Impact of mobile cross-platform development on cpu, memory and battery of mobile devices when using common mobile app features,” *Procedia Computer Science*, vol. 175, pp. 189–196, 2020.

- [13] C. O. Fritz, P. E. Morris, and J. J. Richler, “Effect size estimates: current use, calculations, and interpretation.” *Journal of experimental psychology: General*, vol. 141, no. 1, p. 2, 2012.
- [14] S. Huber and L. Demetz, “Performance analysis of mobile cross-platform development approaches based on typical ui interactions.” in *ICSOFIT*, 2019, pp. 40–48.
- [15] S. Huber, L. Demetz, and M. Felderer, “Analysing the performance of mobile cross-platform development approaches using ui interaction scenarios,” in *Software Technologies: 14th International Conference, ICSOFIT 2019, Prague, Czech Republic, July 26–28, 2019, Revised Selected Papers 14*. Springer, 2020, pp. 40–57.
- [16] D. Inupakutika, S. Kaghyan, D. Akopian, P. Chalela, and A. G. Ramirez, “Facilitating the development of cross-platform mhealth applications for chronic supportive care and a case study,” *Journal of biomedical informatics*, vol. 105, p. 103420, 2020.
- [17] inVerita, “Flutter vs native vs react-native: Examining performance,” Mar 2020, accessed: 2023-04-28. [Online]. Available: <https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980>
- [18] A. Jedlitschka, M. Ciolkowski, and D. Pfahl, “Reporting experiments in software engineering,” *Guide to advanced empirical software engineering*, pp. 201–228, 2008.
- [19] B. Kitchenham, “Procedures for performing systematic reviews,” *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.
- [20] C. W. LeCroy and J. Krysik, “Understanding and interpreting effect size measures,” *Social Work Research*, vol. 31, no. 4, pp. 243–248, 2007.
- [21] M. Mahendra and B. Anggorojati, “Evaluating the performance of android based cross-platform app development frameworks,” in *Proceedings of the 6th International Conference on Communication and Information Processing*, 2020, pp. 32–37.
- [22] Microsoft, “.net multi-platform app ui documentation,” May 2023, accessed: 2023-05-10. [Online]. Available: <https://learn.microsoft.com/en-us/dotnet/maui/>
- [23] —, “Visual studio,” Apr 2023, accessed: 2023-04-28. [Online]. Available: <https://visualstudio.microsoft.com/vs/>
- [24] B. Microsoft Learn (davidbritch and jconrey), “What is .net maui?” Jan 2023, accessed: 2023-04-28. [Online]. Available: <https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui>
- [25] L. Montgomery, D. Fucci, A. Bouraffa, L. Scholz, and W. Maalej, “Empirical research on requirements quality: a systematic mapping study,” *Requirements Engineering*, vol. 27, no. 2, pp. 183–209, 2022.

- [26] D. Ortinau, “Use .net maui for native no-compromise apps,” Oct 2022, accessed: 2023-02-28. [Online]. Available: <https://www.codemag.com/Article/2211052/Use-.NET-MAUI-for-Native-No-Compromise-Apps>
- [27] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, “Systematic mapping studies in software engineering,” in *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*, 2008, pp. 1–10.
- [28] P. Que, X. Guo, and M. Zhu, “A comprehensive comparison between hybrid and native app paradigms,” in *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, 2016, pp. 611–614.
- [29] G. Sims, “How much ram does your android phone really need in 2023?” Jan 2023, accessed: 2023-05-10. [Online]. Available: <https://www.androidauthority.com/how-much-ram-do-i-need-phone-3086661/>
- [30] StackOverflow, “Stack overflow developer survey 2022,” Jun 2022. [Online]. Available: <https://survey.stackoverflow.co/2022/>
- [31] Statcounter, “Mobile operating system market share worldwide,” April 2023, accessed: 2023-05-10. [Online]. Available: <https://gs.statcounter.com/os-market-share/mobile/worldwide>
- [32] M. Willocx, J. Vossaert, and V. Naessens, “A quantitative assessment of performance in mobile app development tools,” in *2015 IEEE international conference on mobile services*. IEEE, 2015, pp. 454–461.
- [33] —, “Comparing performance parameters of mobile app development strategies,” in *Proceedings of the International Conference on Mobile Software Engineering and Systems*, 2016, pp. 38–47.
- [34] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.
- [35] W. Wu, “React native vs flutter, cross-platforms mobile application frameworks,” 2018.

## Appendix A

---

# Supplemental Information

### A.1 Windows PC and Emulator Environment

<b>CPU</b>	AMD Ryzen 7 5800H
<b>RAM</b>	16GB SO-DIMM DDR4, (3200 MHz)
<b>GPU</b>	GeForce RTX 3070 8GB GDDR6, 115 W, (+15 W with Dynamic Boost 2.0)
<b>HDD/SSD</b>	1TB m.2 PCIe NVMe Gen 3 x4
<b>OS</b>	Windows 10 ver. 19044.2604
<b>Network speeds</b>	10/1

Table A.1: Windows PC Hardware

<b>Phone name</b>	Google Pixel 6 (8 cores)
<b>OS</b>	Android 11, Google API level 30
<b>Chipset</b>	8-core built-in GPU
<b>CPU</b>	Octa-core (2x2.80 GHz Cortex-X1, 2x2.25 GHz Cortex-A76, 4x1.80 GHz Cortex-A55)
<b>GPU</b>	Mali-G78 MP20
<b>RAM</b>	8GB RAM
<b>Positioning</b>	GPS (L1+L5), GLONASS (G1), BDS (B1I+B1c+B2a), GALILEO (E1+E5a), QZSS (L1+L5)

Table A.2: Emulator hardware

## A.2 Studies included in Literature Study

APA Reference
Mahendra, M., & Anggorojati, B. (2020, November). Evaluating the performance of Android based Cross-Platform App Development Frameworks. In Proceedings of the 6th International Conference on Communication and Information Processing (pp. 32-37).
Huber, S., & Demetz, L. (2019, July). Performance Analysis of Mobile Cross-platform Development Approaches based on Typical UI Interactions. In ICISOFT (pp. 40-48).
Willocx, M., Vossaert, J., & Naessens, V. (2016, May). Comparing performance parameters of mobile app development strategies. In Proceedings of the International Conference on Mobile Software Engineering and Systems (pp. 38-47).
Willocx, M., Vossaert, J., & Naessens, V. (2015, June). A Quantitative Assessment of Performance in Mobile App Development Tools. In Proceedings of the 2015 IEEE International Conference on Mobile Services (pp. 454-461).
Ciman, M., & Gaggi, O. (2017). An empirical analysis of energy consumption of cross-platform frameworks for mobile development. <i>Pervasive and Mobile Computing</i> , 39, 214-230.
Barros, L., Medeiros, F., Moraes, E., & Júnior, A. F. (2020). Analyzing the Performance of Apps Developed by using Cross-Platform and Native Technologies. In SEKE (pp. 186-191).
Dorfer, T., Demetz, L., & Huber, S. (2020). Impact of mobile cross-platform development on CPU, memory and battery of mobile devices when using common mobile app features. <i>Procedia Computer Science</i> , 175, 189-196.
Biørn-Hansen, A., Rieger, C., Grønli, T. M., Majchrzak, T. A., & Ghinea, G. (2020). An empirical investigation of performance overhead in cross-platform mobile development frameworks. <i>Empirical Software Engineering</i> , 25, 2997-3040.
Huber, S., Demetz, L., & Felderer, M. (2020). Analysing the performance of mobile cross-platform development approaches using UI interaction scenarios. In <i>Software Technologies: 14th International Conference, ICISOFT 2019, Prague, Czech Republic, July 26–28, 2019, Revised Selected Papers 14</i> (pp. 40-57). Springer International Publishing.
Biørn-Hansen, A., Grønli, T. M., & Ghinea, G. (2019). Animations in cross-platform mobile applications: An evaluation of tools, metrics and performance. <i>Sensors</i> , 19(9), 2081.
Que, P., Guo, X., & Zhu, M. (2016, December). A comprehensive comparison between hybrid and native app paradigms. In <i>2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)</i> (pp. 611-614). IEEE.
Continued on next page

Table A.3 – continued from previous page

APA Reference
Inupakutika, D., Kaghyan, S., Akopian, D., Chalela, P., & Ramirez, A. G. (2020). Facilitating the development of cross-platform mHealth applications for chronic supportive care and a case study. <i>Journal of biomedical informatics</i> , 105, 103420.
Corral, L., Sillitti, A., & Succi, G. (2012). Mobile multiplatform development: An experiment for performance analysis. <i>Procedia Computer Science</i> , 10, 736-743.
Delia, L., Galdamez, N., Corbalan, L., Pesado, P., & Thomas, P. (2017, July). Approaches to mobile application development: Comparative performance analysis. In <i>2017 Computing conference</i> (pp. 652-659). IEEE.

Table A.3: Included Literature Studies



