

Memoization for Saving Energy in Android Applications

When and how to do it

Adriano Pinto
NOVA LINCS, DI, FCT
Univ. Nova de Lisboa, Portugal
ab.pinto@campus.fct.unl.pt

Marco Couto
HASLab/INESC TEC
Universidade do Minho, Portugal
marco.l.couto@inesctec.pt

Jácome Cunha
NOVA LINCS, DI, FCT
Univ. Nova de Lisboa, Portugal
jacome@fct.unl.pt

ABSTRACT

Over the last few years, the interest in the analysis of the energy consumption of Android applications has been increasing significantly. Indeed, there are a considerable number of studies which aim at analyzing the energy consumption in various ways, such as measuring/estimating the energy consumed by an application or block of code, or even detecting energy expensive coding patterns or API's.

Nevertheless, when it comes to actually improving the energy efficiency of an application, we face a whole new challenge, which can only be achieved through source code improvements that can take advantage of energy saving techniques. However, there is still a lack of information about such techniques and their impact on energy consumption.

In this paper, we analyze the impact of the memoization technique in the energy consumption of Android applications. We present a systematic study of the use of memoization, where we compare implementations of 18 method from different applications, with and without using memoization, and measure the energy consumption of both of them. Using this approach, we are able to characterize Android methods that should be memoized.

Our results show that using memoization can clearly be a good approach for saving energy. For the 18 tested methods, 13 of them decreased significantly their energy consumption, while for the remaining 5 we observed unpredictable behavior in 3 of them and an overall increase of energy consumption in the last 2. We also included a discussion about when is actually beneficial to use memoization for saving energy, and what is the expected percentage of gain/loss when memoization works and when it does not.

CCS CONCEPTS

• **Software and its engineering** → **Empirical software validation; Software performance; Software evolution;**

KEYWORDS

Android, Energy Analysis, Empirical Analysis

ACM Reference Format:

Adriano Pinto, Marco Couto, and Jácome Cunha. 2018. Memoization for Saving Energy in Android Applications: When and how to do it. In *Proceedings of 5th IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft'18)*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

The Android ecosystem is evolving at an impressive pace. Since the operative system can run on a wide variety of devices, from smartphones to tablets of several brands, to wearables, its widespread usage in the last decade was significantly notorious: 12 major versions were launched in the last 9 years [1], while the number of available applications in the Google Play Store raised from 30K to 3.5M in the last 7 years [3].

This proliferation helped with the increasing interest in a particular research area: energy consumption analysis in software. It has been a hot research topic in the last few years, greatly motivated not only by the mobile development area, characterized by powerful, yet energy-harvesting, hardware components and software systems, which run over batteries with limited capacity, but also by the growing interest of developers in knowing more about how to develop software in a energy-saving manner [29].

Most of the research works appearing in this area in the last few years focused on detecting or predicting [17, 18] the energy consumption that is triggered by a piece of software. For instance, some of them presented techniques for monitoring and classifying energy consumption of blocks of code, such as lines of code [23], methods [13], API calls [25–28], or even code patterns [22, 32]. Even the energy consumed at the testing phase can be a concern [24], as well as the display of visual elements in the application views [14].

Nevertheless, the amount of information provided about an application, that can be used by developers to reduce its energy consumption at the development phase, is still very low. For instance, we could not find, to the best of our knowledge, any technique that shows how to improve energy efficiency by means of memory usage optimization.

In this paper, we present a systematic study that shows the energy gains of applying memoization techniques to Android applications. We based our approach in a study which characterizes what kind of methods are “pure” (i.e., prone to memoization) [34]. After filtering such methods in an Android application, we applied the memoization technique to them and created a new “memoized” application. Finally, we ran a few tests to compare both the raw and “memoized” application.

Our results showed that memoization can greatly improve the energy efficiency of an Android application, considering that such

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MOBILESoft'18, May 2018, Gothenburg, Sweden

© 2018 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

application is prone to it, without threatening its normal functioning and/or efficiency. At this point, we would like to highlight that energy efficiency and better performance are not synonymous in the context of mobile applications [9].

To summarize, with this work we intend to answer the following research questions:

RQ1 Can memoization be used to reduce energy consumption in Android applications?

RQ2 Can we determine when is it worthy to use memoization to save energy?

The remaining of this paper is organized as follows. In Section 2 we introduce the concept of memoization for Android applications and describe in detail our experimental setting. We then present the results of the experiment in Section 3. The findings are discussed in Section 4, where we also answer our research questions. Section 5 presents the threats to the validity of this work. Related approaches are compared in Section 6. Finally, in Section 7 we present our conclusions and future work directions.

2 MEMOIZATION OF ANDROID APPLICATIONS

In this section we describe in detail our experimentation with memoization of Android application, its results, and its analysis.

We start by explaining which kind of Java methods can actually be memoized.

2.1 What Can be Memoized?

Yang et al. [35] proposed a set of three pre-conditions to classify a Java method as memoizable:

- (1) The method must be pure, that is, it must be a function (note the return value cannot be void);
- (2) The arguments must be immutable (such as Java primitive types);
- (3) The return value cannot depend on static fields, public member fields nor publicly exposed member fields.

In our particular case, we have simplified the application of pre-condition (2) and used only methods that have as arguments primitive types and in a very specific situation the Context object of an application.

In the next section we will describe in detail the experimental study we conducted: the methods analyzed, the conditions under where such methods were tested, and the full procedure to actually run the tests and obtain results.

2.2 Experimental Study

The empirical study we conducted was based on three critical points:

- (1) Android applications and methods analysis
- (2) Methods refactoring
- (3) Experiments execution

Android applications and methods analysis: We used two Android applications from the MUSE repository, an extension of the sourcerer repository [7] (Pixate Freestyle [4] and android-demos)

and another one from Fdroid (Chanu [2]). With this set of applications we intend to represent the Android corpus of applications. Indeed, we have small and big applications, from different kinds, etc.

Pixate Freestyle [4] is a free framework that lets the user to style his native Android views with stylesheets and is very much based on the graphical component. This application contains 219 classes and it can be found on github. In this application we made memoization of 4 methods as described in 1.

android-demos is a very specific application from the repository and we only know that it contains 34 classes. In this application we made memoization of 4 methods as described in 1.

Chanu [2] is a well-known application with 538 classes. Basically, it's the 4chan site application where you can browse images of various contents. This is the largest application among the three and so it was where we most applied the technique of memoization. 10 methods were modified as you can see at 1

However, this range of applications made it quite difficult to compile and to run (we found incompatibilities between the SDK version of Android, the level of the Android API of the mobile phone and the build of the application in Android Studio.). Indeed, we could not compile and run all the applications in the same setting. To avoid this issue, we created our own application only with the methods we wanted to test. Thus, for each Android application we copied the methods that could be memoized to our own application. After this process, we have duplicated the application, and in the second version, memoized all the methods in it.

Method	Input	Output	Application
createIntent	Context,String,String	Intent	Chanu
countLines	String	int	Chanu
planifyText	String	String	Chanu
join	List<String>,String	String	Chanu
threadSubject	String,String	String	Chanu
quoteText	String	String	Chanu
textViewFilter	String,boolean	String	Chanu
getUrl	Context,String	String	Chanu
exifText	String	String	Chanu
getNumeral	String,String	String	Pixate Freestyle
removeLocaleInfoFromFloat	String	String	Pixate Freestyle
addNegativeSign	String	String	Pixate Freestyle
addPositiveSign	String	String	Pixate Freestyle
isMobile	String	boolean	android-demos
readableFileSize	long	String	android-demos
dip2px	Context,float	int	android-demos
px2dip	Context,float	int	android-demos

Table 1: Characterization of methods used in experiments.

We have than executed the application. For that, we created a class of tests and each method tested was isolated in order to obtain the measurements with a finer granularity. Each method was called 50 times with different parameters, so that its behavior was not trivial. These insertions were carried out cyclically 10,20,30,40 and 50 times. In the case of the versions of the application with memoization this allowed the insertion of 50 values in the map and consequent consultation of the same ones as several iterations were made.

Methods Refactoring: First of all, the applications that respected the conditions already mentioned were extracted. Only

then was it possible to apply the technique of memoization. Three applications were selected for which this spectrum was clearly visible. Thus, their methods were analyzed manually and after we proceeded with their modification. In the end, we created an application to encompass all the methods that were studied. We have thus created two versions of this same application, one with all methods in their original form and another with these memoized ones.

To do memoization we used the traditional technique, that is, we saved the input of the methods as key in a HashMap and its output as value. So, if the method had already been called for that particular input, then we just accessed its value on the map and returned. Otherwise, a new entry was created on the map and returned. In cases where the methods only received an input parameter, this was directly the key of the map. However, there were cases as can be seen from ref table: 1 in which more than one argument was passed as parameter and for this a library called javatuples as in [35] was used. In this way, the input values were saved in tuples and this became the key in our HashMap.

Experiments execution: The tests were performed 25 times for each method in both the original and the memoized versions, in a Nexus 4 running Android version 5.1.1 - Api level 22 (reported by Qualcomm for having reliable results regarding energy consumption while using Treprn application [5]). For this, the following steps were as shown in Figure 1. In each experimentation, if there were apks of the application and old tests, then they were uninstalled and the new ones installed. When the instrumented tests were carried out, the elaborated test class was executed and the necessary Treprn calls were made. Thus, at the beginning of the tests the Treprn application was started, a warm-up of 5 seconds was done and the measurements were performed. When the method to be tested ended, the tests stopped and the respective results were saved. To try to guarantee the maximum absence of external factors the mobile phone was in flight mode with the screen and wi-fi turned off and still with no other application installed except the testing one.

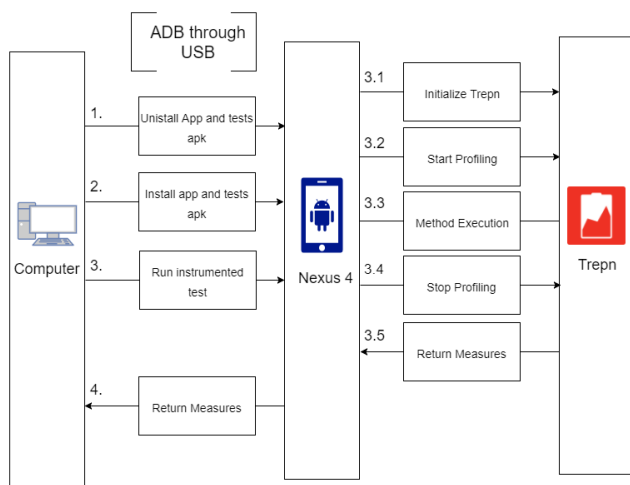


Figure 1: Experimental procedure.

3 RESULTS

In this section we present in detail the results obtained from the experiment introduced in the previous section.

Figure 2 displays a box plot for each of the 13 methods we could find a statistically significant difference between the energy consumption of the original method and the energy consumption of the memoized one in favor of the memoization, that is, when the memoized method spent less energy than the original version¹. Each box plot represents the energy measurements, in millijoules (mJ), of the 25 executions of the corresponding method presenting the minimum and maximum values (marked by the whiskers), possible outliers (marked by small circles), the 1st and 3rd quartiles (bottom and top of the box), the median (marked by the red horizontal filled in line), and the average (marked by the blue horizontal dashed line). Each yellow/green pair represents the original/memoized version of the method. We also show the notch because it can be seen as an informal test of the null hypothesis that the medians are equal, that is, if two notches overlap, then it is not possible to reject the null hypothesis with a 95% confidence [10]. In Section 4 we will present formal verification of the statistical differences of the energy measurements.

Note this charts show the energy values for the 10 executions of the test suite, the first being with new values, and the remaining with repeated ones. In Section 4 we will detail more the analysis of different numbers of executions.

In Figure 3 we show the box plots with the energy consumption of the only two methods where the energy spent by the memoized versions is statistically different from the original methods, but in favor of the original ones. Once more, we refer to Section 4 to test executions greater than 10 times.

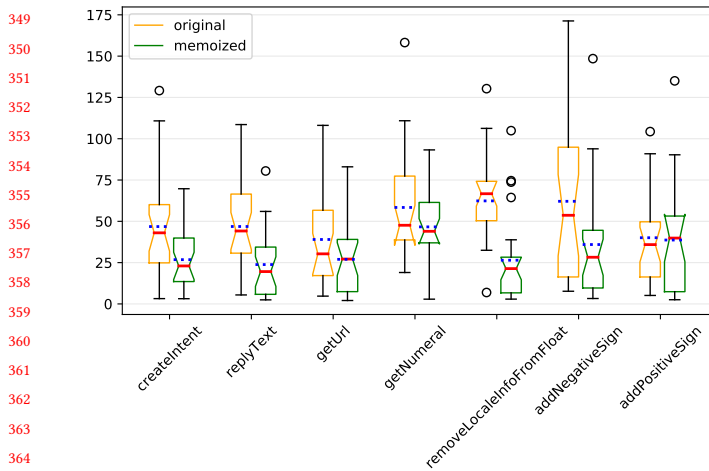
Figure 4 presents once more box plots, but now for the three methods where no statistical difference was found for original and memoized methods.

For these three methods we executed the same test suite, but now running it 10, 20, 30, 40, and 50 times. The goal was to take more advantage of the memoization and understand if this would turn the method better or worst the original one. Each increment of 10 executions only makes the memoized method read the stored values, while the original version needs to run entirely. However, what we recorded for these methods is that sometimes the memoized version is better, but sometimes the original one also is. We will come back to this discussion in Section 4.

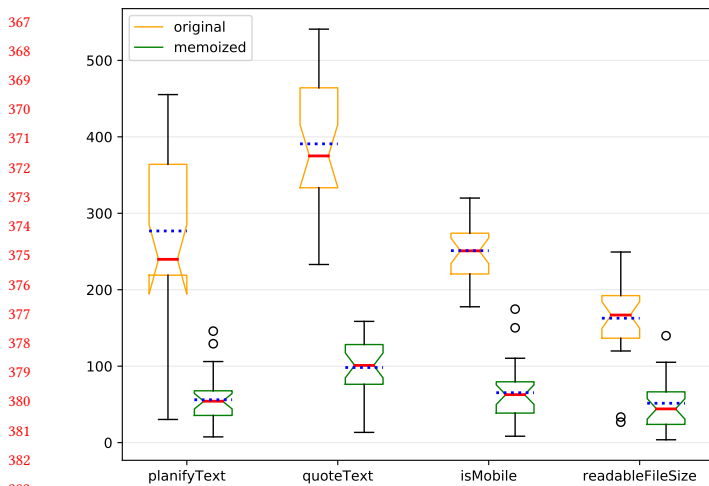
The results presented until now seem to indicate that there is a positive effect in the energy consumption when memoizing methods.

In Figure 5 we present the percentage of losses in the energy spent from the original version of the method to the memoized one, considering the methods where memoization produces positive results (the methods represented in Figure 2). For each method, we calculated the percentage of energy decrease as follows. For each of the 25 executions, we sorted, independently, the original and the memoized values of energy consumption. We then calculated the losses of energy from the use of the original version to the memoized

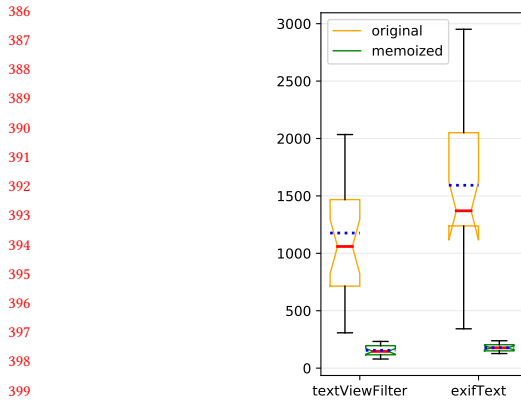
¹We have only divided the methods in different charts because they have different scales of values and to have them all in the same would not allow to properly see all the details.



(a) Methods with consumption between 0 and 175 mJ.



(b) Methods with consumption between 0 and 600 mJ.



(c) Methods with consumption between 0 and 3000 mJ.

Figure 2: Box plots representing the energy spent by the original and memoized methods, considering 10 executions of the tests, where the memoized version spent less energy.

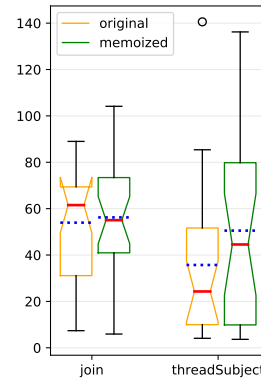


Figure 3: Box plots representing the energy spent by the original and memoized methods, considering 10 executions of the tests, where the original version spent less energy.

one using the formula $\frac{original - memoized}{original} \times 100$. Positive values represent the fact that there is indeed a loss of energy from the original to the memoized, thus supporting the use of memoization, and negative values imply an increase in the energy spent, thus pointing situations where memoization does not spend less energy. As we can see, the gains are fairly positive. Note once more we are only showing the results for running the test suite 10 times, as this already points to quite positive results. We will discuss these gains further on in the next section.

As we seen before, two methods got bad results when memoized. In Figure 6 we present the percentage of losses, which in this case are mostly gains, that is, there is an increase of the energy spent from the original to the memoized version.

Finally, as before, in Figure 7, we present the values for the three methods where there is no clear indication of losses or gains, considering 10 to 50 executions of the test suite.

To increase the reading of these results, and to give a possible summary of the, in Table 2 we present the percentage of the energy not spent comparing the memoized method to the original one of the 1st and 3rd quartiles. This follows from the box plots, giving central values of the losses.

Finally, we show in Table 3 the percentage of times the energy spent by the memoized method is lower than the original one, considering the 25 runs of each method, for 10 to 50 executions of the test suite. Note the underlined values are the ones where we could not find statistical significant difference between the original and memoized measurements. For all the other methods there is statistical significant difference between the energy consumption of the original and the memoized method. Values in **bold** are marked when more than 50% of the 25 runs spend more energy in the memoized methods than the original ones. The values with no special formatting have statistical significance in favor of the memoized versions.

4 DISCUSSION

The results presented so far were obtained considering that the 18 analyzed methods were invoked only 10 times in each of the 25

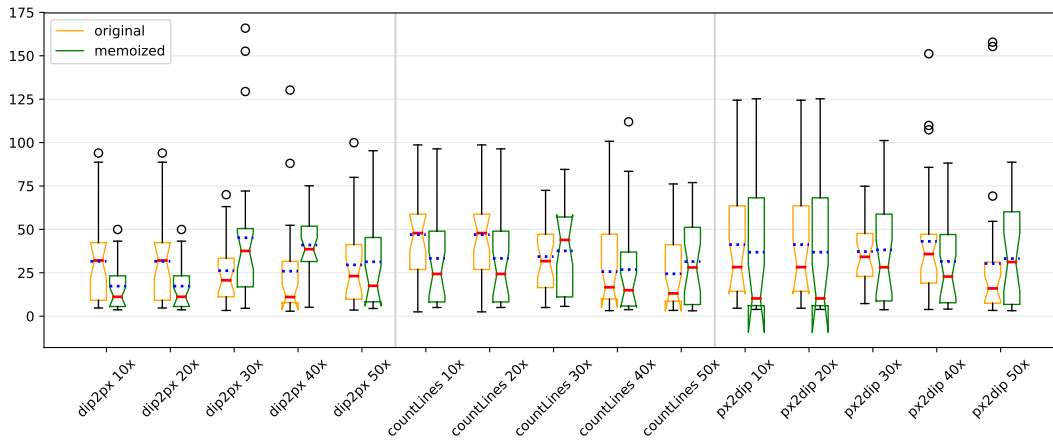


Figure 4: Box plots representing the energy spent by the original and memoized methods, considering 10 to 50 executions of the tests, where no consistent statistical difference among the different runs of tests was found for original and memoized methods.

method	10x	20x	30x	40x	50x
exifText	(87, 90)	(92, 95)			
textViewFilter	(85, 88)	(94, 94)			
planifyText	(77, 83)	(86, 89)			
quoteText	(71, 78)	(88, 92)			
isMobile	(70, 82)	(84, 96)			
readableFileSize	(64, 82)	(82, 89)			
removeLocaleInfoFromFloat	(59, 84)	(66, 71)			
replyText	(44, 64)	(-9, 17)	(41, 76)	(41, 71)	(48, 76)
createIntent	(35, 48)	(-21, 6)	(35, 54)	(21, 43)	(49, 66)
addNegativeSign	(34, 53)	(62, 80)	(61, 75)	(68, 94)	(71, 86)
getUrl	(19, 52)	(32, 66)	(45, 57)	(37, 56)	(49, 62)
getNumeral	(4, 21)	(-21, 2)	(9, 30)	(27, 45)	(60, 75)
addPositiveSign	(-6, 59)	(40, 48)	(51, 65)	(36, 57)	(10, 26)
join	(-11, 11)	(-40, -8)	(-46, -18)	(-30, -22)	(-36, -19)
px2dip	(-55, 0)	(2, 60)	(-27, 53)	(4, 52)	(-82, 5)
threadSubject	(-62, 1)	(-99, -54)	(-92, -75)	(-280, -101)	(-219, -102)
countLines	(-84, -7)	(16, 62)	(-17, 6)	(-15, 21)	(-53, 16)
dip2px	(-161, -60)	(24, 47)	(-71, -35)	(-281, -62)	(-21, 17)

Table 2: 1st (on the left hand-side of the column) and 3rd (on the right) quartiles of the energy losses from the use of the original method to the use of the memoized version.

measured tests. Nevertheless, they already allow us to point out some interesting observations, while discussing their implications.

The first observation is the most obvious one: memoization has indeed a clear impact on energy consumption in Android applications. In fact, for the majority of situations the impact is fairly positive. Considering the 18 tested methods, the energy consumption for 13 of them has consistently decreased when using memoization (as can be seen in Figures 2a, 2b, and 2c). For those methods, we can observe that the average consumption of the 25 measurements is always lower when using memoization, as well as the minimum and maximum values, and the values for the 1st and 3rd Quartile. Figures 2b and 2c show the methods that had the biggest impact of

all. Hence, we already answer to the first research question (RQ1): memoization can, in fact, be used to reduce energy consumption in Android applications.

To make this results more meaningful, we want to know if there is statistical evidence behind these observations, that is, if the energy consumed by the memoized version of these methods is consistently lower than the original one. In other words, we want to validate if the obtained values for our experiments are statistically significant. Thus, we tested the following hypothesis:

$$H_0 : P(A > B) = 0.5$$

$$H_1 : P(A > B) \neq 0.5$$

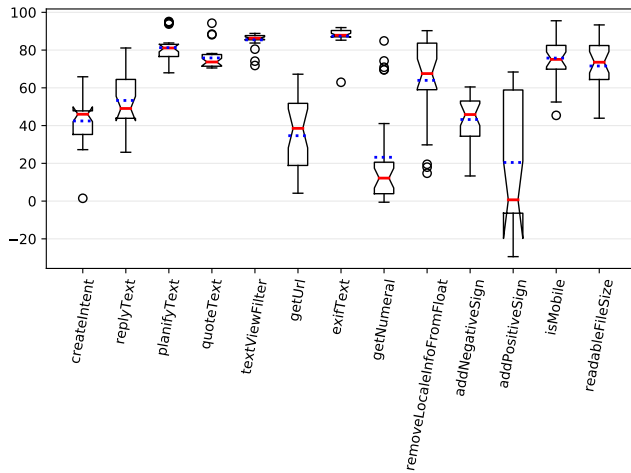


Figure 5: Energy losses from the original method to the memoized one of the methods that have positive results.

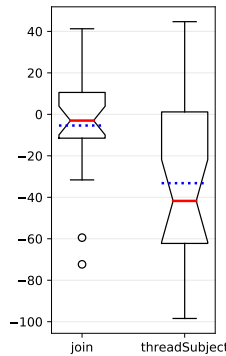


Figure 6: Energy losses from the original method to the memoized one of the methods that have negative results.

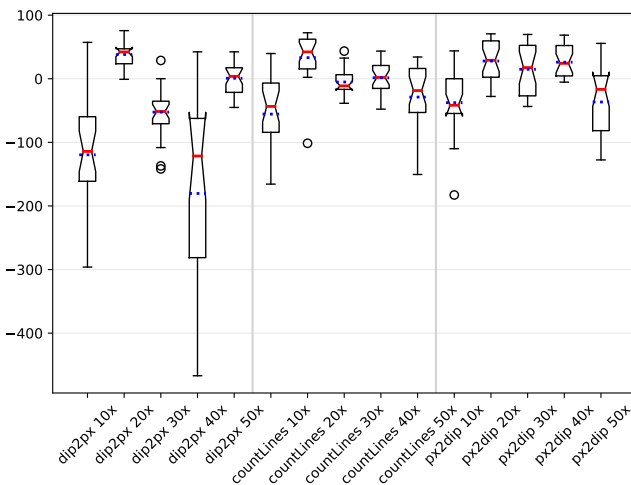


Figure 7: Energy losses from the original method to the memoized one of the methods that have no clear positive or negative direction.

	10x	20x	30x	40x	50x
createIntent	72	<u>56</u>	68	<u>68</u>	72
getUrl	<u>60</u>	72	<u>68</u>	76	76
replyText	68	<u>52</u>	84	92	84
getNumeral	56	<u>56</u>	<u>64</u>	68	84
addNegativeSign	56	88	84	100	100
addPositiveSign	<u>56</u>	72	80	84	80
planifyText	100	100	-	-	-
quoteText	100	100	-	-	-
textViewFilter	100	100	-	-	-
exifText	100	100	-	-	-
removeLocaleInfoFromFloat	80	88	-	-	-
isMobile	100	100	-	-	-
ReadableFileSize	88	100	-	-	-
join	<u>48</u>	<u>32</u>	<u>28</u>	<u>28</u>	<u>24</u>
threadSubject	<u>40</u>	<u>36</u>	<u>20</u>	<u>12</u>	<u>20</u>
countLines	36	<u>68</u>	<u>48</u>	44	<u>44</u>
dip2px	24	<u>68</u>	32	28	<u>56</u>
px2dip	40	<u>56</u>	<u>48</u>	<u>64</u>	44

Table 3: For each method, and for each test setting, we present the percentage (of the 25 runs) that the memoized version consumes less energy than the original one. Underlined values have statistical significance while all the others have; bold values represent more than 50% of the 25 runs spend more energy in the memoized methods than the original ones.

where B and A represent, the act of randomly drawing a value from the set of 25 measurements, with and without using memoization, respectively. Hence, $P(A > B)$ represents, when drawing from both A and B , the probability of getting a value from A (without memoization) larger than the one drawn from B (using memoization). Our null hypothesis is then obtaining a probability of 50%, while obtaining one different than 50% is the alternative hypothesis.

To understand if there is an overall significant relevance between the distributions of A and B , we ran the Wilcoxon signed-rank test, with a two-tail p -value considering $\alpha = 0.01$ ². The test was repeated for all the 13 methods where the use of memoization led to improvements in energy consumption. At the end, the test produced significant relevance, with the p -value < 0.01 for 11 of the 13 cases. The exceptions where `getUrl` and `addPositiveSign`, both with parametric distributions, which we will exclude for now and explain later. To calculate a non-parametric effect size, Field [16] suggests using Rosenthal's formula [30, 31] to compute a correlation, and compare the correlation values against Cohen's [11] suggested thresholds:

- 0.1: small significance;
- 0.3: medium significance;
- 0.5: large significance.

From the 11 scenarios, 8 were non-parametric, and the values obtained were: 0.4 (medium) for `createIntent` and `replyText`

²All the values obtained for this test, for all the methods 18 methods, can be found in the appendix Table ??

697 methods, and 0.6 (large) for the remaining methods. For the remain- 755
 698 ing 3 methods, a D'Agostino and Pearson's test [15] revealed we 756
 699 were dealing with normal distributions, and so we should calculate 757
 700 the Cohen's d coefficient to determine the magnitude of the effect 758
 701 size. According to Sawilowsky [33], the reference thresholds and 759
 702 their respective effect size should be 760

- 703 • 0.01: very small;
- 704 • 0.2: small;
- 705 • 0.5: medium;
- 706 • 0.8: large;
- 707 • 1.2: very large;
- 708 • 2: huge significance.

710 We obtained values of 0.4 (small) for the `getNumeral` method, 0.6
 711 (medium) for `addNegativeSign`, and 1.4 (very large) for the method
 712 `removeLocaleInfoFromFloat`. Considering these reference values,
 713 we have statistical supported to say that, for these methods, using
 714 memoization does lead to energy savings.

715 The effect size values calculated so far were only for the scenario
 716 where each method was invoked 10 times in each measured test.
 717 It can still be possible that, for methods with lower significance
 718 values, repeating the experience with an increasing number of
 719 invocations would lead to results more or less supportive. Therefore,
 720 we repeated the experience with 20, 30, 40 and 50 invocations,
 721 calculated the significance values and the effect size (see Table 3).
 722 We observe that all the 13 aforementioned methods continue to
 723 have more than 50% of the 25 tests in favor of memoization and,
 724 although in some specific cases the statistical experiment resulted in
 725 no significance, the percentage of memoization favorable tests kept
 726 increasing with the increasing number of invocations for all of them,
 727 and the same happened with the significance. Such cases were the
 728 previously excluded `getUrl` and `addPositiveSign` method. Hence,
 729 we categorized these 13 methods as *prone to memoization*.

730 Nevertheless, the use of memoization is not always synonym
 731 of saving energy. In some cases, it is not possible to determine
 732 whether memoization is suitable for saving energy or not. In our
 733 experiments, we found 3 methods fitting such criteria. The results
 734 of such experiments are shown in Figure 4, in which the methods
 735 where invoked 10 times on each test. As we can see by examining
 736 the box plots, the values for the original version and the version
 737 using memoization are very similar. For instance, the maximum
 738 energy consumption of the original `join` method is lower than
 739 the memoized version, while both the median and the minimum
 740 values are higher. The memoized `addPositiveSign` method has
 741 higher values for the median and 3rd quartile when compared to
 742 the original version, while the maximum and the 1st quartile values
 743 are slightly lower.

744 By running the same statistical experiment performed for the 13
 745 methods were memoization actually decreased energy consumption,
 746 we observed that the results for `dip2px`, `px2dip`, and `countLines`
 747 methods were not statistically significant. This means that the rea-
 748 son behind the energy consumption being lower is not related to the
 749 use of memoization, neither to the use of the original version of the
 750 method. Similarly, the percentage of tests in favor of memoization
 751 was mostly around 50%, dropping or increasing a few percentage
 752 levels unpredictably when increasing the number of invocations.
 753 Therefore, we categorized these methods as *unpredictable*.

755 For the remaining 2 methods (`threadSubject` and `join`), we
 756 observed that the energy consumption actually increased in the ma-
 757 jority of cases while using memoization. By looking at Figure 3, we
 758 can see that the tendency is for their memoized version to consume
 759 more energy in the test scenario where a method is invoked 10
 760 times. In fact, if we examine Table 3, we see that for such methods
 761 the percentage of times where the memoized version consumes
 762 less energy is always lower than 50%, for all test scenarios (10, 20,
 763 30, 40 and 50 invocations per test). We ran the same statistical test
 764 as before to check if the values were consistently worse for the
 765 memoized version, and we obtained significant relevance for all of
 766 them, except `px2dip`. However, the effect size varied from "small
 767 significance" to "medium significance". Thus, these methods are
 768 then categorized as *unfit for memoization*.

769 The most interesting observations can, however, be seen in Ta-
 770 ble 2. The data presented there shows the calculated gains (positive
 771 values) or losses (negative values) when using memoization. To
 772 obtain these values, we first sorted, for each method, the energy
 773 consumed by the original and memoized version, in order to com-
 774 pare the lower/higher energy consumption values of one version
 775 with the lower/higher values of the other. Then, we calculated the
 776 gains pairwise and arranged them in a box plot. The first element
 777 of the pair in each table cell is the gain calculated for the 1st quartile
 778 of the box plots, while the second element is the gain considering
 779 the 3rd quartile. With this, we try to show the gains/losses are
 780 independent of the energy consumption measured value. Indeed
 781 in some cases the energy consumption tends to be low (closer to
 782 the 1st quartile), while in other tend to be high (closer to the 3rd
 783 quartile). In any of these cases the gains/losses are approximate.
 784 Each column contains the gains/losses calculated for the obtained
 785 results of running the same experiment, but varying the number of
 786 times each method is invoked from 10x to 50x.

787 As expected, we see that the majority of the values are positive,
 788 that is, memoization is, in the majority of cases, a suitable tech-
 789 nique to save energy. In the first 7 methods, the impact is more
 790 notorious since in all the 25 measurements the memoized version
 791 has a significantly lower energy consumption, and when we re-
 792 peated the experiment for 20 invocations per test, the impact is even
 793 greater. Also, the significance values kept increasing, so we stopped
 794 measuring and categorized them as *strongly prone to memoization*
 795 methods.

796 If we look only to the pairs with both positive gains in the column
 797 for 10 invocations (methods *prone to memoization*), the gains can
 798 go from 3% to 90%. These values tend to increase if we increase
 799 the number of invocations: for these same methods, considering
 800 20 invocations, if we exclude the ones with negative values, since
 801 they were the cases with no statistical significance (see Table 3),
 802 they go from 31% to 96%. For the other scenarios (30, 40 or 50
 803 invocations), the values are always positive, as the significance
 804 keeps being maintained.

805 For the *unpredictable* methods, the gains are also unpredictable:
 806 at the 3rd quartile the value is negative (loss), but for the 1st quartile
 807 is positive (gain). It gets even more unpredictable with the increas-
 808 ing number of invocations: the `px2dip` method, for instance, has a
 809 negative gain (loss) in the 1st quartile for the 10, 30, and 50 invo-
 810 cations scenarios, while the other values are positive; `countLines`
 811 has a similar behavior, but the values are not at all proportional;

dip2px goes from a 161% loss in the 1st quartile for 10 invocations, to 23% gain for 20 invocations, and back to losing for 30 and 40 invocations, with 70% and 281% loss, respectively, and the same behavior is observed for the 3rd quartile. Moreover, by looking at Figure 4, we can see that the interquartile range of the box plot for the memoized version of these methods is sometimes above and other times overlapping the original version one, which corroborates the previous analysis.

Similarly, the values for the *unfit for memoization* methods are also expected. We have losses in every tested scenario, for both the 1st and 3rd quartile, except in the first scenario (10 invocations), where the 3rd quartile has small gains of 10% and 1% for `join` and `threadSubject`, respectively. Even so, in that scenario the statistical test showed there was no significance for both the methods, while the values for both the 1st and 3rd quartile gains stay negative as we increase the number of invocations and tend to decrease proportionally.

This categorization of the methods, as *prone/unfit for memoization* and *unpredictable*, helps us start answering the to **RQ2**. With the data presented and discussed in this section, we can in fact follow these categories to determine when does memoization lead to energy savings. Although these classifications were based on a dynamic approach (i.e., we need to run several experiments first and a thorough analysis later to reach these conclusions), we strongly believe that the observations discussed here can actually be used to perform a more static analysis, by analyzing the resemblances of the methods in each category.

5 THREATS TO VALIDITY

In this section, we address potential issues that may influence the findings we report in this work and we detail how we have minimized them to make the results trustworthy and generalizable.

Measuring the energy consumption of a mobile device is a complex task [9]. This is mostly due to the fact that it is quite difficult to fully isolate the code or application under measured. To address this issue, we executed our application 25 times, thus giving it enough slack to have in average results that actually correspond to the truth and with as low side effects as possible. Moreover, we executed the application in a completely clean Android device, i.e. without any other application installed rather than the default ones, and with all background services disabled. We even put the device in airplane mode, and with the lowest brightness level, to make sure the energy consumed by the display was as low as possible.

To assure the data retrieved were consistent, both the memoized and non-memoized applications were run in the same environment, half of the times starting with the memoized version, and the other half with the non-memoized one. Thus, this gives us confidence the differences we found between the two versions is only due to the usage of memoization or not. Indeed, the absolute energy values themselves are not what matters most. What is truly important is that the measurements are consistently different and in favor of the memoized version.

The energy consumption values were obtained by using Trepn, a tool developed by Qualcomm, a company that produces mobile devices processors and other parts. As Trepn uses embedded hardware sensors, it is believed to produce real results. Indeed it has

been used in several other approaches [19–21]. Moreover, Trepn only measures the energy for the application being monitored, thus producing real results.

Our experimentation occurred only in one smartphone, and thus only on one version of the Android operating system. It is expected to find different absolute values in different smartphones and Android versions, but the differences between the memoized versions and original ones are quite consistent and thus it is not expected to see significant changes in the differences if changing the evaluation settings.

We have also used an application created by us to execute the methods we intended to memoize. We did this because it was not possible to consistently find and run applications under the same settings. Nevertheless, we have selected the applications with no particular criteria, as we simply order them by the number of possible methods to be memoized. Moreover, we have selected all the methods we could find within these applications, not choosing any particular methods or kind of methods. The application we created has only the code of the copied methods, plus the tests we have created, and nothing else.

Finally, another possible issue is the quality of the executed tests. It is usually quite difficult to find some tests for Android applications [12–14]. Thus, we created a test suite for each method we wanted to analyze. Although they are not real uses of the methods, we believe they may represent a reasonable use of memoized methods. Ultimately, only the developer may have some idea about the real use of the method and thus if it deserves to be memoized. In any case, we have shown that in the evaluated circumstances memoization is beneficial to save energy.

6 RELATED WORK

Yang et al. have proposed a technique and tool to determine the functional purity of Java methods to more easily determine which methods can be refactored [35]. In particular, if a method is a pure function, then it can be memoized. In their evaluation they were able to successfully memoize several methods from 3 different Java libraries and to reduce their execution time. Nevertheless, the memory usage has also grown. In our experiments we have had a similar result. The most important difference is that we have shown that memoization has also a positive impact in the energy consumption.

A quite similar approach has also been proposed by Agosta et al. [6]. In their work they have also defined which Java methods can be memoized based on their functional purity. Both definitions are similar. The evaluation was performed using financial functions and quite good results were achieved both for the energy and time. Quite interesting is also the fact that they have defined a theoretical model to predict the effectiveness of the memoization approach in terms of energy consumption. However, their approach has been applied to a particular set of computations in a desktop computer, not a mobile device. In our case, we have used a wide range of Android applications.

Banerjee and Roychoudhury proposed a set of guidelines to improve energy consumption related to the use of energy-intensive resources such as the GPS or the camera [8]. Based on these guidelines they propose a set of refactorings to ensure the guidelines are followed within an application. They were to show that by

applying the refactoring suggested by their platforms applications spent between 3 and 29 percent less energy. Our solution can be combined with this one as they address different issues, possibly achieve even greater gains.

Cruz and Abreu have also proposed a set of refactorings based on performance guidelines to improve the energy consumption of Android applications [14]. They have shown that in general these refactorings improve the energy consumption of the applications, although this does not occur for all applications and refactorings.

7 CONCLUSIONS

In this work we have explored the use of memoization in Android applications, focusing on its impact in the energy consumption. We selected 18 methods from 3 different applications and designed an experiment to evaluate the energy consumption of such methods, as well as their memoized versions. The results from this experiment show that indeed the use of memoization promotes energy saving in most cases. This was the case for 13 methods, but for 2 methods the opposite occurred, and for 3 the memoization was no different from the original version.

Although the results are quite positive with significant gains, we need to consider the fact that we cannot describe the impact in the overall application, as we have isolated the methods from the applications. As in most cases, it is a decision of the developers to use or not memoization. Nevertheless, we have shown that to memoize methods is most likely positive for an Android application. In any case, we intend to study the impact of these gains in the overall consumption of the applications.

Despite the extensive discussion about energy, it is still necessary to understand the impact in terms of execution time and memory usage. By definition, the energy is related to the execution time, as $energy = power \times time$. Indeed, in general both energy and time decrease or increase together. We have not analyzed in detail the execution time, but we have noticed this was also the case for our experiment. In any case, this still needs to be carefully analyzed. Regarding the memory usage, we have also not analyzed the difference between the original and memoized versions of the methods. Nevertheless, the overhead in the memoized method is related to the use of a map to store the computed values, which heavily depends on the number of different values calculated as the map will need to store all of them. The memoization will have less impact if fewer elements need to be stored, and more impact otherwise. On the other hand, the original method memory will no longer be used, which may balance both versions. In any case, once again, most likely only the developer will have such informed guess.

REFERENCES

- [1] 2018. Android version history. https://en.wikipedia.org/wiki/Android_version_history. (2018). Accessed: 2018-01-03.
- [2] 2018. Chanu app's FDroid page. <https://f-droid.org/en/packages/com.chanapps.four.activity>. (2018). Accessed: 2018-01-14.
- [3] 2018. Number of available applications in the Google Play Store. <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>. (2018). Accessed: 2018-01-03.
- [4] 2018. PixelateFreestyle app's FDroid page. <https://github.com/Pixate/pixate-freestyle-android>. (2018). Accessed: 2018-01-14.
- [5] 2018. Which mobile devices report accurate system power consumption? <https://developer.qualcomm.com/forum/qdn-forums/software/teppn-power-profiler/28349>. (2018). Accessed: 2018-01-14.
- [6] Giovanni Agosta, Marco Bessi, Eugenio Capra, and Chiara Francalanci. 2012. Automatic memoization for energy efficiency in financial applications. *Sustainable Computing: Informatics and Systems* 2, 2 (2012), 105–115. DOI: <https://doi.org/10.1016/j.suscom.2012.02.002> IEEE International Green Computing Conference (IGCC 2011).
- [7] Sushil Bajracharya, Joel Ossher, and Cristina Lopes. 2014. Sourcerer: An Infrastructure for Large-scale Collection and Analysis of Open-source Code. *Sci. Comput. Program.* 79 (Jan. 2014), 241–259. DOI: <https://doi.org/10.1016/j.scico.2012.04.008>
- [8] Abhijeet Banerjee and Abhik Roychoudhury. 2016. Automated Re-factoring of Android Apps to Enhance Energy-efficiency. In *Proceedings of the International Conference on Mobile Software Engineering and Systems (MOBILESoft '16)*. ACM, New York, NY, USA, 139–150. DOI: <https://doi.org/10.1145/2897073.2897086>
- [9] Abhijeet Banerjee and Abhik Roychoudhury. 2017. Future of Mobile Software for Smartphones and Drones: Energy and Performance. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft '17)*. IEEE Press, Piscataway, NJ, USA, 1–12. DOI: <https://doi.org/10.1109/MOBIEMSoft.2017.1>
- [10] John M. Chambers, William S. Cleveland, Beat Kleiner, and Paul A. Tukey. 2017. *Graphical Methods for Data Analysis*. Chapman and Hall/CRC.
- [11] Jacob Cohen. 1988. *Statistical power analysis for the behavioral sciences*. Hillsdale, NJ: Lawrence Erlbaum Associates 2 (1988).
- [12] M. Couto, J. Cunha, J. P. Fernandes, R. Pereira, and J. Saraiva. 2015. GreenDroid: A tool for analysing power consumption in the android ecosystem. In *Scientific Conf. on Informatics, 2015 IEEE 13th International*. 73–78.
- [13] M. Couto, Carção T., J. Cunha, J. P. Fernandes, and J. Saraiva. 2014. Detecting Anomalous Energy Consumption in Android Applications. In *Programming Languages*, Fernando Magno Quintão Pereira (Ed.). LNCS, Vol. 8771. Springer Int. Publishing, 77–91.
- [14] Luis Cruz and Rui Abreu. 2017. Performance-based Guidelines for Energy Efficient Mobile Applications. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft '17)*. IEEE Press, Piscataway, NJ, USA, 46–57. DOI: <https://doi.org/10.1109/MOBIEMSoft.2017.19>
- [15] Ralph B. D'Agostino. 1971. An Omnibus Test of Normality for Moderate and Large Size Samples. *Biometrika* 58, 2 (1971), 341–348. <http://www.jstor.org/stable/2334522>
- [16] Andy Field. 2009. *Discovering statistics using SPSS*. Sage publications.
- [17] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan. 2012. Estimating Android applications' CPU energy usage via bytecode profiling. In *Green and Sustainable Software (GREENS), 2012 First Int. Workshop on*. 1–7.
- [18] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan. 2013. Estimating Mobile Application Energy Consumption using Program Analysis. In *Proc. of 35th Int. Conf. on Software Engineering (ICSE)*.
- [19] Nagaraj Hegde, Edward L. Melanson, and Edward Sazonov. 2016. Development of a real time activity monitoring Android application utilizing SmartStep. In *Proceedings of the 2016 IEEE 38th Annual International Conference of the Engineering in Medicine and Biology Society (EMBC)*. 1886–1889.
- [20] Yan Hu, Jiwei Yan, Dong Yan, Qiong Lu, and Jun Yan. 2017. Lightweight energy consumption analysis and prediction for Android applications. *Science of Computer Programming* (2017). DOI: <https://doi.org/10.1016/j.scico.2017.05.002>
- [21] Reyhaneh Jabbarvand, Alireza Sadeghi, Joshua Garcia, Sam Malek, and Paul Ammann. 2015. EcoDroid: An Approach for Energy-based Ranking of Android Apps. In *Proceedings of the Fourth International Workshop on Green and Sustainable Software (GREENS '15)*. IEEE Press, Piscataway, NJ, USA, 8–14. <http://dl.acm.org/citation.cfm?id=2820158.2820161>
- [22] D. Li and W. G. J. Halfond. 2014. An Investigation into Energy-saving Programming Practices for Android Smartphone App Development. In *Proc. of 3rd Int. Workshop on Green and Sustainable Software (GREENS 2014)*. ACM, 46–53.
- [23] D. Li, S. Hao, W. G. J. Halfond, and R. Govindan. 2013. Calculating Source Line Level Energy Information for Android Applications. In *Proc. of 2013 Int. Symposium on Software Testing and Analysis (ISSTA 2013)*. ACM, 78–89.
- [24] D. Li, Y. Jin, C. Sahin, J. Clause, and W. G. J. Halfond. 2014. Integrated Energy-directed Test Suite Optimization. In *Proc. of 2014 Int. Symposium on Software Testing and Analysis (ISSTA 2014)*. ACM, 339–350.
- [25] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk. 2014. Mining Energy-greedy API Usage Patterns in Android Apps: An Empirical Study. In *Proc. of 11th Working Conf. on Mining Software Repositories (MSR 2014)*. ACM, 2–11.
- [26] K. Liu, G. Pinto, and Y. D. Liu. 2015. Data-Oriented Characterization of Application-Level Energy Optimization. In *Fundamental Approaches to Software Engineering*, Alexander Egyed and Ina Schaefer (Eds.). LNCS, Vol. 9033. Springer Berlin Heidelberg, 316–331.
- [27] R. Pereira, M. Couto, J. Cunha, J. P. Fernandes, and J. Saraiva. 2016. The Influence of the Java Collection Framework on Overall Energy Consumption. In *Proc. of 5th Int. Workshop on Green and Sustainable Software (GREENS '16)*. ACM, 15–21.

929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986

987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044

- [28] G. Pinto and F. Castor. 2014. Characterizing the Energy Efficiency of Java's Thread-Safe Collections in a Multi-Core Environment. In *Proc. of SPLASH'2014 workshop on Software Engineering for Parallel Systems (SEPS)*, SEPS, Vol. 14.
- [29] G. Pinto, F. Castor, and Y. D. Liu. 2014. Mining Questions About Software Energy Consumption. In *Proc. of 11th Working Conf. on Mining Software Repositories (MSR 2014)*. ACM, 22–31.
- [30] Robert Rosenthal. 1991. *Meta-analytic procedures for social research*. Vol. 6. Sage.
- [31] Robert Rosenthal, H Cooper, and LV Hedges. 1994. Parametric measures of effect size. *The handbook of research synthesis* (1994), 231–244.
- [32] C. Sahin, L. Pollock, and J. Clause. 2014. How Do Code Refactorings Affect Energy Usage?. In *Proc. of 8th ACM/IEEE Int. Symposium on Empirical Software Engineering and Measurement (ESEM '14)*. ACM, 36:1–36:10.
- [33] Shlomo Sawilowsky. 2009. New Effect Size Rules of Thumb. 8 (11 2009), 597–599.
- [34] Jiachen Yang, Keisuke Hotta, Yoshiki Higo, and Shinji Kusumoto. 2014. Revealing Purity and Side Effects on Functions for Reusing Java Libraries. In *Proceedings of the 14th International Conference on Software Reuse for Dynamic Systems in the Cloud and Beyond (ICSR 2015)*, Ina Schaefer and Ioannis Stamelos (Eds.). Springer International Publishing, Cham, 314–329. DOI : https://doi.org/10.1007/978-3-319-14130-5_22
- [35] J. Yang, K. Hotta, Y. Higo, and S. Kusumoto. 2015. Towards purity-guided refactoring in Java, In 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME) (2015)*, 521–525. DOI : <https://doi.org/10.1109/ICSM.2015.7332506>

A APPENDIX

A.1 Statistical Significance Values

This section presents the complete set of statistical significance values of the differences between the energy consumption of the original and memoized methods discussed in Section 4.

1045		1103
1046		1104
1047		1105
1048		1106
1049		1107
1050		1108
1051		1109
1052		1110
1053		1111
1054		1112
1055		1113
1056		1114
1057		1115
1058		1116
1059		1117
1060		1118
1061		1119
1062		1120
1063		1121
1064		1122
1065		1123
1066		1124
1067		1125
1068		1126
1069		1127
1070		1128
1071		1129
1072		1130
1073		1131
1074		1132
1075		1133
1076		1134
1077		1135
1078		1136
1079		1137
1080		1138
1081		1139
1082		1140
1083		1141
1084		1142
1085		1143
1086		1144
1087		1145
1088		1146
1089		1147
1090		1148
1091		1149
1092		1150
1093		1151
1094		1152
1095		1153
1096		1154
1097		1155
1098		1156
1099		1157
1100		1158
1101		1159
1102		1160

	method	x10	x20	x30	x40	x50	
1161	createIntent	0,008041299	0,58122813		0,097970132	0,004530068	1219
1162		0,374813377	0,078006845	0,704286941	0,234020535	0,401449861	1220
1163		signif.	NOT signif.	Medium signif.	NOT signif.	signif.	1221
1164	replyText	0,008041299	0,903626797	0,000239913		0,000493159	1222
1165		0,374813377	0,017123454	0,519411432	1,214290831	0,492774948	1223
1166		signif.	NOT signif.	signif.	Very Large signif.	signif.	1224
1167	getNumeral	0,008041299	0,882352227	0,231167213			1225
1168		0,408655198	0,020928666	0,169331932	0,630904041	1,609746553	1226
1169		Small signif.	NOT signif.	NOT signif.	Medium signif.	Very Large signif.	1227
1170	removeLocaleInfoFromFloat	0,008041299					1228
1171		1,393672067	1,915076039				1229
1172		Very Large signif.	Very Large signif.				1230
1173	addNegativeSign	0,008041299			1,23E-05	1,23E-05	1231
1174		0,633443528	1,929771523	1,637670151	0,618346942	0,618346942	1232
1175		Medium signif.	Very Large signif.	Very Large signif.	signif.	signif.	1233
1176	getUrl	0,174210276	0,003821977	0,011876382	0,001078559	0,002469714	1234
1177		0,192163204	0,409060285	0,355787318	0,462333252	0,428086345	1235
1178		NOT signif.	signif.	NOT signif.	signif.	signif.	1236
1179	addPositiveSign	0,58122813			0,000890538	0,128450538	1237
1180		0,078006845	0,73071236	1,149329788	0,469943676	0,214994475	1238
1181		NOT signif.	Medium signif.	Large signif.	signif.	NOT signif.	1239
1182	exifText	1,23E-05					1240
1183		0,618346942	4,969633412				1241
1184		signif.	Huge signif.				1242
1185	textViewFilter	1,23E-05	1,23E-05				1243
1186		0,618346942	0,618346942				1244
1187		signif.	signif.				1245
1188	planifyText	1,23E-05	1,23E-05				1246
1189		0,618346942	0,618346942				1247
1190		signif.	signif.				1248
1191	readableFileSize	4,07E-05					1249
1192		0,580294823	2,881652532				1250
1193		signif.	Huge signif.				1251
1194	join	0,59980234		0,032427612			1252
1195		0,074201633	0,536376123	0,30251435	0,948059876	0,931683899	1253
1196		NOT signif.	Medium signif.	NOT signif.	Large signif.	Large signif.	1254
1197	threadSubject	0,008041299	0,026430974		0,000363626	0,000295767	1255
1198		0,406992268	0,313929986	0,86293659	0,504190584	0,511801008	1256
1199		Small signif.	NOT signif.	Large signif.	signif.	signif.	1257
1200	dip2px	0,008041299	0,026430974				1258
1201		0,753436407	0,313929986	0,555634732	0,612999871	0,067050489	1259
1202		Medium signif.	NOT signif.	Medium signif.	Medium signif.	Very Small signif.	1260
1203	countLines	0,008041299	0,087527127	0,544909588		0,367385491	1261
1204		0,218636178	0,241630959	0,085617269	0,044835853	0,1274746	1262
1205		Small signif.	NOT signif.	NOT signif.	Very Small signif.	NOT signif.	1263
1206	px2dip	0,008041299	0,58122813	0,903626797			1264
1207		0,035701915	0,078006845	0,017123454	0,360779422	0,083062057	1265
1208		Very Small signif.	NOT signif.	NOT signif.	Small signif.	Very Small signif.	1266
1209							1267
1210							1268
1211							1269
1212							1270
1213							1271
1214							1272
1215							1273
1216							1274
1217							1275
1218							1276

Table 4: Statistical significance values of the differences between the energy consumption of the original and memoized methods.