Towards Automated Android App Internationalisation: An Exploratory Study

Pei Liu[a], Qingxin Xia[b,*], Kui Liu[c], Juncai Guo[d], Xin Wang[d], Jin Liu[d], John Grundy[a], Li Li[e]

[a]*Monash University, Victoria, Australia*
[b]*North China Institute of Science and Technology, Hebei, China*
[c]*Huawei, China*
[d]*Wuhan University, Wuhan, China*
[e]*Beihang University, Beijing, China*

Abstract

Android has become the most popular mobile platform with over 2.5 billion active users who use many different languages across many different countries. In order for Android apps to be usable by all of them, app developers usually need to add an internationalisation feature that adapts the app to the users' linguistic and cultural requirements. Such a process, including the translation from the default language to up to thousands of languages, is usually achieved via manual efforts and hence is resource-intensive, time-consuming, and error-prone. Automated approaches are hence in demand to help developers mitigate such manual efforts. Since there are millions of apps proposed already for Android users, we are interested in knowing to what extent internationalisation has been supported. Our experimental results show that Android apps, at least the ones released on online markets, have mostly been equipped with internationalisation features, with the number of supported languages varies significantly. By mapping the actual term translations among different languages, we further find that the translations tend to be consistent among different apps, suggesting the possibility to learn from this data to achieve automated app internalization. To explore this idea we implemented a Transformer-based prototype approach *Androi18n*, that learns from developers' practical translations to achieve automated mobile app text translations. Experimental results show that *Androi18n* is effective in achieving our objective, and its high performance is generic across the translations of different languages.

1. Introduction

Android, as the most popular Mobile Operating System [1], very widely used since its first version released in September 2008. As of May 2019, there are 2.5 billion active Android devices worldwide. To date, there are more than 3 million Android apps on the official Google Play store.

Among many reasons making Android a huge success in the mobile market, internationalisation (or i18n in short) is an important one[1]. Internationalisation enables the apps to be used with various languages and regions without engineering changes. This is achieved by preparing the code to load content from multiple files representing supported usage locales. A toggle is then used between different content and settings based on the chosen locale (e.g., FR for

*Corresponding author
Email addresses: Pei.Liu@monash.edu (Pei Liu), xiaqingxin@ncist.edu.cn (Qingxin Xia), brucekuiliu@gmail.com (Kui Liu), guojuncai1992@163.com (Juncai Guo), xinwang0920@whu.edu.cn (Xin Wang), jinliu@whu.edu.cn (Jin Liu), John.Grundy@monash.edu (John Grundy), lilicoding@ieee.org (Li Li)

[1]https://www.martechadvisor.com/articles/proximity-marketing/mobile-app-localization-and-internationalization/

French). This approach allows the app to be smoothly used by users from different countries and regions speaking different languages. While increasing the visibility of the app, it also positively impacts the app's install and usage rate. As recently revealed by Infopulse[2], the largest number of mobile users are located in India, Indonesia, South Africa, Turkey, and China. Ignoring those languages could be the main reason causing an app to be low-downloaded or low-rated. As disclosed by the Common Sense Advisory Survey [2], users will also have an emotional connection to the app that talks in their mother tongue and hence will be more likely to choose apps supporting their native languages.

Internationalisation is the key to successfully spread Android apps in the world. However, it is not clear how internationalisation is currently supported in practice in real-world Android apps. The internationalisation rate of apps in the Android ecosystem and the number of languages supported by real-world Android apps remain unclear for researchers and practitioners. It is also unknown if the provided language translations are complete and reliable for those apps that do have internationalisation feature. To the best of our knowledge, this research direction has been little explored by the research community.

To deepen our systematic knowledge of Android app internationalisation, we first conducted an exploratory study on the internationalisation status quo of real-world Android apps. Our experimental results reveal that existing apps, especially closed-source ones, are most likely to support internationalisation with a range of languages supported. We further looked into the internationalisation provided by Android apps and confirm that similar terms have recurrently appeared in different Android apps, and those apps generally agree with each other when translating the terms to other languages. This empirical evidence suggests that it is possible to achieve automated Android app internationalisation by learning from existing apps' internationalisation contents.

There are, to the best of our knowledge, no existing works conducted to help developers characterize internationalisation for the development of Android apps. Each development team has to rely on professional translators to implement dedicated internationalisation features for their apps, resulting in uncountable efforts spent on repetitive yet boring tasks. Hence, we argue that there is a strong need for inventing an automated Android app internationalisation approach that can liberate developers from completing such labor-intensive tasks. The only work we are aware of is the one proposed by Wang et al. [3], who proposed an RNN-based approach [4] to achieve such a purpose. Unfortunately, RNN-based approaches process embedded tokens one by one sequentially and hence will suffer from long dependency issues, which will subsequently impact the prediction results.

Motivated by these findings, we have prototyped an automated app internationalisation approach based on the famous Transformer model[3] trained on the translation agreements among real-world Android apps. Experimental results show that our approach is effective, being able to outperform the state-of-the-art and achieve reliable text translations when fulfilling the internationalisation feature for Android apps.

To summarise, this research makes the following key contributions:

- We present the first exploratory study to understand the status quo of app internationalisation in the Android community.

[2]https://medium.com/@infopulseglobal_9037/mobile-app-internationalization-ways-and-methods-to-boost-revenue-by-26-4f8985d3c4bd

[3]Unlike RNN, which treats a sentence word by word, Transformer processes sentences as a whole. Transformer further goes beyond RNN by supporting multi-head attention and positional embeddings, which further provide information about the relationship between different words.

- We design and prototype a neural network-based tool called *Androi18n* for achieving automated app internationalisation through learning knowledge from existing Android apps.
- We evaluate *Androi18n* using a large set of real-world Android apps and against five popular speaking languages. The corresponding experimental results show that our approach is effective, being able to outperform the state-of-the-art approaches and generate reliable translations.

Open source. The source code and datasets are all made publicly available in our artifact package [5].

2. Background

Internationalisation has been a common feature implemented in modern software to enter the global market for decades [6]. The idea of internationalisation is to decouple multi-language support from engineering works, so that developers can exclusively focus on function development while dedicated language translators can address the translations between different languages.

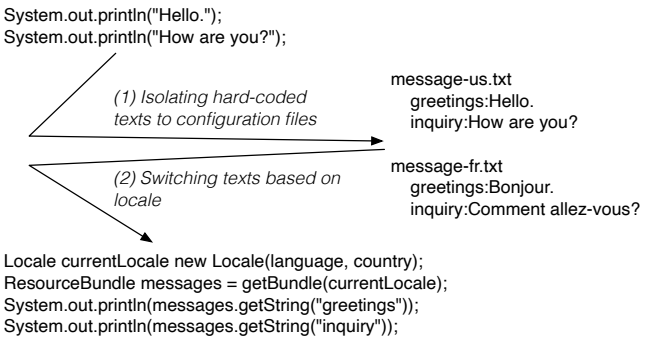

Figure 1: Example of Java internationalisation excerpted from [7].

Figure 1 shows a simple example[4] excerpted from [7] that illustrates how the internationalisation of programs is (manually) achieved. At the beginning, developers need to identify all the hard-coded texts that will be shown to users and should be translated based on users' locale. The identified hard-coded texts are then maintained in a dedicated configuration file. Their corresponding translations are subsequently maintained in other configuration files (i.e., one per language). After that, the programming code can refer to these files for accessing the texts needed to show on the software (e.g., messages.getString("greetings")). Based on users' locale, the corresponding configuration file can be loaded (without largely modifying the programming code), and the displaying languages will be switched to the one best fit for the users.

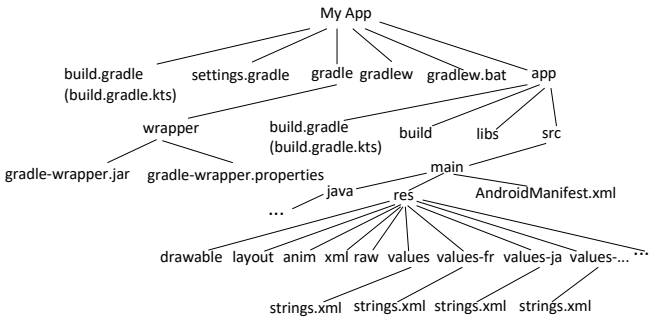

Figure 2: The typical directory structure of an Android application.

```
<!-- Code snippet from strings.xml under directory values -->
<resources>
    <string name="app_name">Launcher Play</string>
    <string name="wallpaper">Wallpaper</string>
    <string name="a_beer">A Beer</string>
    <string name="icon_size">Icon size</string>
    <string name="circle_menu_apps">Circle menu apps</
        string>
    ...

<!-- Code snippet from strings.xml under directory values-es -->
<resources>
    <string name="app_name">Launcher Play</string>
    <string name="wallpaper">Fondo de pantalla</string>
    <string name="a_beer">Una cerveza</string>
    <string name="icon_size">Tamaño de icono</string>
    <string name="circle_menu_apps">Aplicaciones del circulo<
        /string>
    ...
```

Listing 1: Code snippets of strings.xml excerpted from Android project [8].

[4]The full example is available as a tutorial [7].

The internationalisation of Android apps is supported in a way similar to traditional programs. The internationalisation mechanism adopted by Android apps is also implemented through configuration files. Figure 2 illustrates the typical file structure of an Android project. The configuration files are located under the res directory. The texts configured in file values/strings.xml will be displayed as the default setting, which should be the language that most intended users are familiar with. The alternative texts for different languages are provided in file strings.xml in dedicated directories (often in the form of values-<qualifier> in the res directory. The <qualifier> is a locale name indicating the language that is provided for [9]. For example, values-es indicates that its strings.xml configuration file is written in Spanish (hence the app will display Spanish if the users' locale is configured as so) as the example in Listing 1. When users run the app, the Android system selects the specific resources according to the devices' locale. If no specific <qualifier> is provided, the default setting will be used.

3. Exploratory Study

3.1. Dataset

We plan to conducted our exploratory study on real-world Android apps, including both open-source and closed-source Android apps.

Open-source Android apps: Apps in this category have their source code made publicly available in the community, e.g., on popular code hosting sites such as Github and Bitbucket or dedicated sharing sites such as F-Droid for distributing open-source Android apps. In this work, we used the AndroZooOpen [10] dataset which collects many open-source Android apps. AndroZooOpen currently contains over 70,000 open-source Android apps collected from the aforementioned resources (i.e., Github, Bitbucket, F-Droid, etc.).

Closed-source Android apps: Apps in this category come as compiled versions that are usually distributed by their developers through app markets such as the official Google Play store. In the current mobile ecosystem, there are over 300 app stores today, including over 60 app markets in China and what's more, this number is still growing[5]. In this work, instead of directly crawling apps from those app markets, we leverage AndroZoo to collect closed-source apps. The team of AndroZoo has pre-crawled over 10 million Android apps from various app markets, including the official Google Play store and Chinese ones such as App China.

3.2. Research Questions

To fulfill our exploratory study aiming at understanding the status quo of Android app internationalisation, we resort to answering the following four research questions, formed mainly from two perspectives: (1) configuration and (2) content. The former perspective concerns the internationalisation mechanism at the file level (e.g., whether internationalisation has been introduced and how many languages are supported, etc.). The latter perspective investigates the actual internationalisation content provided by app developers. This will only apply to such apps that have been supported with an internationalisation mechanism.

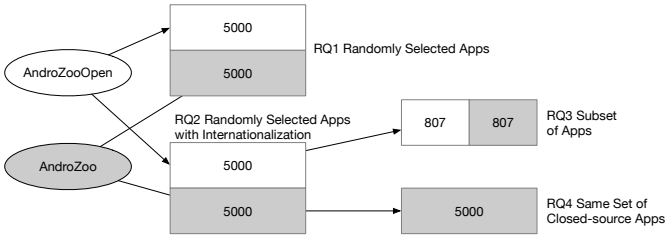

Figure 3: The number of selected apps leveraged in the experiments designed to answer the four research questions.

Particularly, the four research questions are as follows:

RQ1: *To what extent are real-world Android apps internationalized?*

Motivation and Dataset. There are two types of real-world Android apps (e.g., open-source and closed-source

[5]https://www.businessofapps.com/guide/app-stores-list/

ones) frequently leveraged by our fellow researchers to explore security and quality issues of Android apps. As the first research question, we would like to understand if internationalisation has been considered by these two types of apps. Specifically, as illustrated in Figure 3, we randomly select 5,000 open-source apps from AndroZooOpen and 5,000 closed-source apps from AndroZoo to support this research question.

Main Findings. Internationalisation is widely addressed in the closed-source Android apps, while it has not been seriously taken into account by open-source app developers.

RQ2: *How many languages are supported by real-world Android apps?*

Motivation and Dataset. For such apps (both open-source and closed-source) that have been provided with internationalisation, we explore how many languages they support. This research question will be helpful in understanding the most popular languages supported by real-world Android apps. Since not all the apps of the 10,000 apps selected for answering the first RQ have been internationalised, we have to randomly re-select 10,000 apps (5,000 open-source and 5,000 closed-source) to fulfill this research question. This time, we guarantee that both of the 5,000 apps have been internationalized.

Main Findings. Closed-source Android apps generally support more languages than that supported by open-source apps. In terms of the most popular supported languages, the difference between open-source and closed-source apps is relatively small, i.e., English and Spanish the most popular two languages for both open-source and closed-source apps.

RQ3: *Do the supported languages change during the evolution of the Android apps?*

Motivation and Dataset. This research question concerns the evolution of the internationalisation feature. With this research question, we aim to understand if the supported languages will be changed in the lifetime of given Android apps and observe hints in understanding why such changes need to happen. Since this research question concerns the history of Android apps, we have to limit our experimental apps to have (1) explicit releases (tags) for open-source apps and (2) lineage versions for closed-source apps. To this end, we are able to select 807 open-source apps and 807 closed-source apps from the corresponding datasets used in the second research question to prepare the experimental study of this research question.

Main Findings. During the evolution of Android apps, app developers will likely support new popular languages to attract more users.

RQ4: *To what extent is the internationalisation provided by existing apps consistent with each other?*

Motivation and Dataset. This research question goes beyond file-level investigation to further look at the contents of supported languages. Since open-source apps are generally non-commercial ones (as revealed in the findings of RQ1, only a small set of open-source apps are also uploaded to Google Play), they might not have been designed to be used by large-scale app users. Their quality, including the contents put into supporting internationalisation, cannot be guaranteed. Therefore, in this research question, we decide to only look at the internationalisation contents of closed-source Android apps. Specifically, the same set of 5,000 closed-source apps used in RQ2 is used to prepare the experiments for this research question.

Main Findings. With over 95% of consistency rate, the translation tends to be consistent among different Android apps.

3.3. ***RQ1:*** *Internationalisation Rate*

Experimental Setup: To understand the status quo of Android app internationalisation, we investigate to what extent real-world Android apps supported with internationalisation feature. We randomly selected 5,000 real-world open-source Android apps from AndroZooOpen [10].

We then randomly selected 5,000 closed-source Android apps from AndroZoo [11].

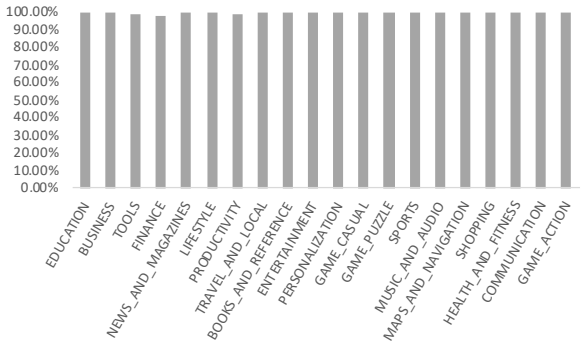

Figure 4: The percentage of internationalisation among the top-20 categories.

For each of the selected apps, we checked if it has been supported with internationalisation feature based on the following two rules: **Rule 1:** The app has adopted the internationalisation mechanism introduced in Section 2 (e.g., the non-empty strings.xml shown in Figure 2 exists in Android apps). **Rule 2:** The app supports at least two languages. For open-source apps, we directly search for the relevant files in their repositories, while for closed-source apps, we leverage the Android Asset Packaging Tool (AAPT) [12] to traverse the *res* directory to locate the relevant files. To determine the type of the supported language, we check the qualifier of the directory containing the strings.xml. The actual supported language is determined based on the name of the directory (e.g., the directory values-fr in Figure 2) where the internationalisation file is located. However, the default language of the app does not provide such information (e.g., directory values), we resort to a popular Python tool (polyglot) [13] to identify the languages of internationalisation. We extract the values of the items in the strings.xml and feed the values into the popular package polyglot [13] to determine the type of the language.

Results: Table 1 illustrates the international rate of Android apps. Overall, **5,485 of the selected 10,000 real-world Android apps are supported with internationalisation feature**, giving an internationalisation rate of 54.85%.

Table 1: International rate of Android apps.

	Open-source	Closed-source	Total
#. Apps	5,000	5,000	10,000
#. Apps (inter.)	497	4,988	5,485
Ratio (%)	9.91	99.76	54.85

Among the 5,000 open-source apps, only 497 apps support at least two different languages, giving an internationalisation rate of 9.91%. The low internationalisation rate of open-source Android apps may be caused by the lack of a dedicated team to develop and maintain the internationalisation task. Indeed, it requires experienced multilingual developers to implement the internationalisation feature. We further go one step deeper to check to what extent the selected open-source apps are also released on the official Google Play store. Our experimental result reveals that only 2.74% (137/5,000) of the open-source Android apps are currently available on Google Play.[6] Among the 137 Android apps, only 48 of them support internationalisation, giving an internationalisation rate of 35%, which is much higher than that of apps not available on Google Play, for which the internationalisation rate is only around 9.2% (449/4863).

In contrast, most of the selected closed-source Android apps (i.e., 99.76%=4,988/5,000) do support internationalisation. This big contrast, compared with open-source Android app development teams, indicates that internationalisation is regarded as important by closed-source Android apps. Indeed, almost every app released to app markets has supported internationalisation, which is expected as app developers usually want to obtain as many worldwide users as possible. We further go one step further to check the internationalisation rate across different categories among the selected real-world Android apps.

[6]There might be more apps uploaded to Google Play initially as Google Play is regularly removing apps [14].

Since AndroZoo does not provide the category information about the Google Play Apps. We write scripts to crawl such categories directly from Google Play, for which each published Android app has been assigned to a category. We then calculate the internationalisation rate for apps in each category. Figure 4 illustrates the rate for the top-20 categories. As shown in the figure, as well as revealed in our experimental results, the internationalisation rate crossing different categories is generally stable. This experimental evidence strongly suggests that category has a limited impact on the adoption of the internationalisation feature in Android apps.

Answer to RQ1

Internationalisation is widely addressed in the closed-source Android apps, while it has not been seriously taken into account by open-source app developers.

*3.4. **RQ2:** Diversity of Supported Languages*

Experimental Setup: In our second research question, we further investigate the diversity of languages supported by the apps that have internationalisation. Since there are only 497 Android apps with the internationalisation feature (cf. Section 3.3), which may not be representative enough to fulfill the experiment, we thus re-select 5,000 open-source Android apps (We actually checked more apps and only stopped at the point when 5,000 apps (with internationalisation supported) are located.) that have been internationalized (i.e., with at least two languages supported). We did the same re-selection for closed-source Android apps (i.e., eventually collecting 5,000 closed-source apps with internationalisation). We follow the approach presented in Section 3.3 to collect the set of languages supported by each app.

Results: Table 2 summarises the statistic results, including the most supported language, language pair (i.e., two different languages that are supported simultaneously), and language triple (i.e., three different languages that are supported simultaneously). **English and Spanish are the two most popular languages for both open-source and closed-source Android apps.** This is consistent with the fact that English and Spanish are the most popular (in terms of the number of countries and regions) spoken languages in the world. Among the top-10 popular languages, nine of them are considered by both open-source and closed-source Android apps. It implies that app developers prefer to support the most popular languages first when supporting internationalisation as it enlarges the potential user base of their apps.

When looking at the number of apps each language is supported, **we can observe that the number of languages supported decreases sharply for the open-source app set while that is generally stable in the closed-source app set.** This phenomenon also applies to the top supported language pairs and language triples. This result suggests that closed-source apps are not only more likely to be integrated with internationalisation feature than open-source apps (as confirmed previously) but also tend to include more languages compared to open-source apps when implementing internationalisation. Similar trends could also be observed when concerning language pairs and language triples between open-source and closed-source apps. This finding is further backed up by the distribution of the number of supported languages in open-source and closed-source apps, as illustrated in Figure 5. The difference is also statistically significant, as confirmed by an MWW test.

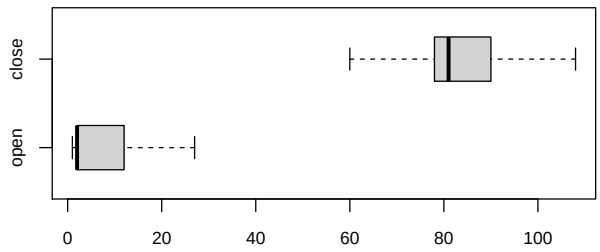

Figure 5: Distribution on the number of languages supported by open-source and close-source Android apps.

Table 2: Top-10 languages, language pairs, and language triples ranked by the number of apps that supported them. To better present the results, we have considered different language dialects as the same language (e.g., both American English (i.e., *en-rUS*) and British English (i.e., *en-rGB*) are regarded as English (i.e., *en*).

# Open-source Apps						# Closed-source Apps					
Language		Language Pair		Language Triple		Language		Language Pair		Language Triple	
en (English)	4,883	en - es	2,088	de - en - fr	1,498	en	4,988	en - es	4,805	de - en - fr	4,756
es (Spanish)	2,112	en - zh	1,937	en - es - fr	1,498	es	4,814	en - fr	4,802	en - es - fr	4,755
zh (Chinese)	1,988	de - en	1,929	de - en - es	1,471	fr	4,808	de - en	4,782	de - en - es	4,744
de (German)	1,941	en - fr	1,875	de - es - fr	1,402	de	4,784	es - fr	4,759	de - es - fr	4,737
fr (French)	1,888	en - ru	1,827	en - fr - ru	1,371	ru	4,750	de - fr	4,758	en - fr - it	4,720
ru (Russian)	1,838	en - pt	1,606	de - en - it	1,369	zh	4,737	en - ru	4,749	en - es - it	4,719
pt (Portuguese)	1,625	en - it	1,543	en - es - ru	1,368	it	4,733	de - es	4,746	en - es - ru	4,719
it (Italian)	1,551	es - fr	1,506	de - en - ru	1,367	pt	4,722	en - zh	4,735	es - fr - it	4,717
ja (Japanese)	1,398	de - fr	1,504	en - es - pt	1,364	ja	4,717	en - it	4,730	en - es - pt	4,716
pl (Polish)	1,319	de - es	1,477	en - fr - it	1,357	ko (Korean)	4,694	fr - it	4,723	en - fr - ru	4,716

Answer to RQ2

When supporting internationalisation, app developers tend to include the most popular speaking languages first, which is true for both open-source and closed-source Android apps. In practice, closed-source apps have generally included more languages than open-source Android apps.

3.5. ***RQ3:*** *Evolution of Internationalisation*

Experimental Setup: In this research question, we are interested in understanding how internationalisation evolves during development and maintenance phases of Android apps.

For the open-source projects, as mentioned early, we have retained 807 projects to support this study. The 807 apps are selected from the initial 5,000 apps leveraged for answering the previous research question. The reason why 807 apps are selected is that we limit the selected apps to contain at least two public releases (via tags on Github). Among the 5,000 open-source apps, only 20.48% (1,024/5,000) of them have been explicitly released by their developers. Unfortunately, 217 of the 1,024 projects contain only one release, which cannot be used to support our evolutionary study. Therefore, we have to further exclude them from consideration. Finally, we retain a total of 807 open-source Android app projects to fulfill this study.

For closed-source Android apps, we cannot extract their evolutionary histories from the apps per se as such information is not included. To overcome this limitation, we followed the idea of Gao et al. [15, 16] to extract the evolutionary histories based on the apps' historically released versions (termed as app lineages). We selected 807 apps (the same number as the open-source apps) from 5,000 closed-source apps and resorted to AndroZoo again to mine their historical releases (i.e., the same app but is released at a different time). Eventually, we mined five versions for each of the selected 807 apps based on the app's last modification time (one version per year[7]). This process leads to in total 4,035 closed-source apps, which are then leveraged to fulfill this study.

To determine the languages supported in our selected Android apps, we first exclude languages represented by the directory values-<qualifier> but with an empty strings.xml file. For such language translations that are indeed not empty, we investigate the proportion of terms

[7]If multiple versions are released in the same year, we will only consider the last one in that year to form the dataset.

translated in their strings.xml files compared with the default setting (i.e., strings.xml file). We further exclude languages with less than 30% terms translated for the selected Android apps. The rationale behind this exclusion is that such translations (with less than 30% terms translated) may not be representative, i.e., haven't been fully completed or come with low qualities.

Results: We present our analysis results both for open-source projects and closed-source apps, respectively.

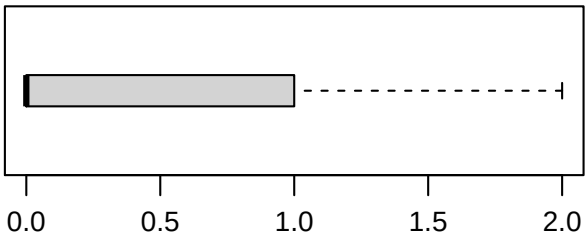

Figure 6: Language update times during the development of open source Android apps. (open-source apps)

- **Open source evolution:** Among all the selected open-source projects, we excluded 357 empty strings.xml. Since there is no strings.xml containing items less than 30% compared with the default one, all the remaining strings.xml (69,349 files) are considered.

  Figure 6 illustrates the distribution of the number of languages additionally updated based on the two subsequent releases for every project. Interestingly, **over 42% of the projects have updated at least one language during their developments.** Table 3 further shows the number of languages added or removed during the evolution based on the two consecutive releases among the total 807 multiple releases available projects. From the release perspective in repeated cases, **the most frequently updated languages are Spanish and Russian, with 119 times added (cf. columns 1-3 on row 3) and 20 times removed (cf. columns 4-6 on row 3), respectively**. The number of projects is nearly the same as the number of times given languages are added or removed in their releases. In general, new languages are added along with the releases of the projects which is what we expect. However, a few languages are still added or removed multiple times as the projects releases since the number of projects and languages addition and removal are different between repeated and non-repeated cases.

  We then manually looked into some of the projects and confirmed that this is indeed the case for some of the Android apps. Based on our understanding, the main reason causing those repeated cases is that the developers try to merge other branches with different languages support before the release, such as the project edipo2s/TESLegendsTracker [17]. For example, suppose developers $d1$ and $d2$ have independently contributed to branches $b1$ and $b2$, respectively. Developers $d1$ has added the support of language l when started to contribute to $b1$ while developers $d2$ has not involved any changes related to the app internationalisation. Once $b1$ and $b2$ is merged, all the commits in $b1$ will contain l while commits in $b2$ will not, leading to repeated cases, e.g., language l is added, removed, and then added again, and so on so forth. After excluding the repeated cases (columns 7-12 in Table 3), the number of added and removed cases are reduced. The aforementioned experimental results show that app developers are more likely to add new languages rather than removing existing ones when developing their apps, as suggested by the high number of added languages and the low number of removed languages after excluding repeated cases.

- **Closed source evolution:** Of the total selected 4,035 closed-source apps (807 app lineages), we excluded 8 empty strings.xml and 105,447 strings.xml containing items less than 30% compared with the default one. As a result, 245,378 strings.xml files are

Table 3: The top-10 added and removed languages during the evolution of the open-source Android app projects on the basis of **release**. **With Repeated Cases** include projects that have both added and removed the same languages, while those projects are excluded for **Without Repeated Cases**.

With Repeated Cases						Without Repeated Cases					
Addition	Count	#. Projects	Removal	Count	#. Projects	Addition	Count	#. Projects	Removal	Count	#. Projects
es	119	116	ru	20	19	es	86	86	zh	7	7
fr	107	105	es	20	19	de	75	75	in	7	7
de	103	100	id	19	17	fr	74	74	zh-rTW	6	6
ru	92	89	zh	19	19	ru	64	64	zh-rCN	6	6
zh-rTW	87	85	de	19	19	zh-rCN	53	53	nb	6	6
zh-rCN	77	77	fr	18	18	zh-rTW	52	52	es	6	6
pt-rBR	75	73	it	16	15	ja	49	49	de	5	5
nl	73	72	nl	15	14	nl	49	49	ru	5	5
ja	72	71	uk	13	13	it	47	47	fr	5	5
it	72	71	pt-rBR	12	12	pl	44	44	nl	5	5

Table 4: The top-10 added and removed languages during the evolution of the closed-source Android apps (i.e., app lineages). **With Repeated Cases** include lineages that have both added and removed the same languages, while those lineages are excluded for **Without Repeated Cases**.

With Repeated Cases						Without Repeated Cases					
Addition	Count	#. Lineages	Removal	Count	#. Lineages	Addition	Count	#. Lineages	Removal	Count	#. Lineages
ms	358	350	ms-rMY	239	237	ms	140	140	ms-rMY	65	65
sq	350	346	et-rEE	239	237	sq	135	135	et-rEE	65	65
ur	350	345	mn-rMN	237	235	vi	133	133	hy-rAM	64	64
bn	349	344	lo-rLA	237	235	nb	133	133	ka-rGE	64	64
ml	349	344	km-rKH	237	235	th	133	133	lo-rLA	64	64
gu	346	341	ka-rGE	236	234	ur	132	132	mn-rMN	64	64
kn	345	339	hy-rAM	236	234	da	131	131	km-rKH	64	64
nb	345	334	my-rMN	205	205	fi	130	130	en-rIN	53	53
hy	338	334	ur-rPK	205	205	in	130	130	pt	52	52
be	328	321	ta-rIN	203	203	el	130	130	ml-rIN	68	68

retained for our study.

Figure 7 illustrates the distribution of the number of supported languages with respect to the app versions in the selected lineages. Clearly, as time goes by, developers of closed-source Android apps have appeared to be interested in supporting more languages in their apps. Similar to that of open-source projects, we compare the extracted languages for two subsequent apps (based on their release times) to decide whether new languages are added or existing languages are removed. Table 4 summarises the experimental results obtained based on the evolution of closed-source Android apps. Surprisingly, we also observe repeated cases (i.e., add and remove the same language) during the apps' evolution history. Unfortunately, at this stage, we do not have evidence to explain why such cases happened in real-world Android apps. After excluding the repeated cases, **we could obtained more or less similar observations compared to that obtained during the evolution of the open-source Android app projects.** Nevertheless, the top-10 list of added languages during the evolution of

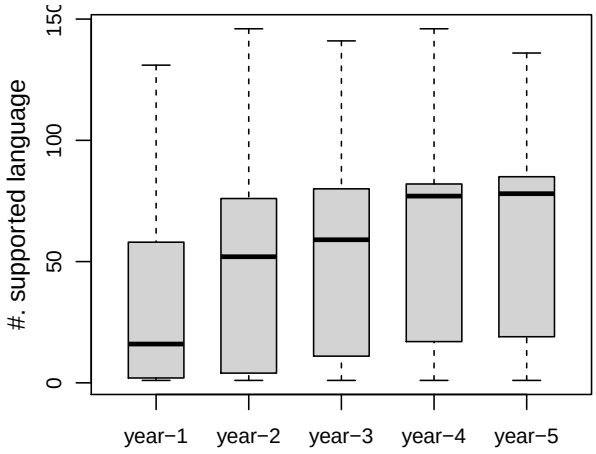

Figure 7: Number of languages supported by five consecutive closed-source Android app releases. As time goes by, the number of supported languages generally increases.

closed-source Android apps is quite different (actually less popular) that that of open-source Android app projects. This result can be explained by the fact that the popular languages (as listed in the open-source projects) have already been included in the closed-source apps during their first release in their lineages.

In summary, internationalisation has been regarded as an important feature by Android app developers, no matter they are developing open-source or closed-source Android apps. Aiming for attracting as many users as possible, developers are interested in supporting more languages (during the evolution of their apps) and tend to prioritize the popular ones.

Answer to RQ3

For both open-source and closed-source Android apps, developers are interested in adding new languages to attract more potential users during the evolution of their apps. When adding new languages, they tend to include and prioritize popular languages over less popular ones.

3.6. ***RQ4:*** *Consistency*

Experimental Setup: One of the objectives of this work is to check if it is possible to learn from historical internationalized apps to achieve automated app internationalisation. For example, if we empirically find that a given term *a* (e.g., *Wallpaper*) in English has always been translated to term *b* (e.g., *Fondo de pantall*) in Spanish in a randomly selected set of apps (c.f. Listing 1), we could conclude that *Fondo de pantall* is the Spanish version of *Wallpaper*.

In this research question, we are hence interested in checking if such consistency has been kept in real-world Android apps. To this end, we use our 5,000 closed-source Android apps that support at least two languages to investigate the term translation consistency among different apps. To determine the translation consistency, we wrote scripts to map translation terms between different languages among the collected Android apps and concluded that the translation is consistent if and only if these exists only one translation between two different languages for the same term, such as the translation from *Wallpaper* in English to *Fondo de pantall* in Spanish is consistent if and only if the English term *Wallpaper* is always translated to *Fondo de pantall* in Spanish among the selected Android Apps.

Results: For the sake of simplicity, we only discuss the top-10 term translations. Figure 8 illustrates the coverage of term translations (i.e., how many terms are translated w.r.t. the total number of terms needed to be translated) of the top-10 pairs. Surprisingly, not all the terms displayed to app users are translated when supporting a new language. In such a case, the default terms will be displayed. Nevertheless, the fact that only a small number of terms are not covered shows that this impact could be neglected, not even mention that some terms look very similar between different languages.

For the translated terms, Table 5 further summarises the experimental results concerning the consistency of the translations. Again, the top-10 language pairs are considered, which are listed in the first column. The second and third columns indicate the number of consistent translations (term *a* in language *X* is always translated to term *b* in language *Y* in different apps) with respect to case

sensitive (i.e., exactly the same) and case insensitive (i.e., the same text, but some characters come with different cases) comparisons. The fourth column shows the percentage of inconsistent translations, i.e., the same term is translated to different ones by different apps. For example, the term "View Posts" in English has been translated to both "Ver publicaciones" and "Ver posts" in Spanish. **The fact that only a small amount of translations is inconsistent among different Android apps shows that the existing translations, with at least 95% of translations agreed by randomly selected apps, are quite reliable.** Thus it seems very possible to learn from those existing (or historical) translations to perform automated text translation so as to achieve accurate and automated internationalisation for Android apps.

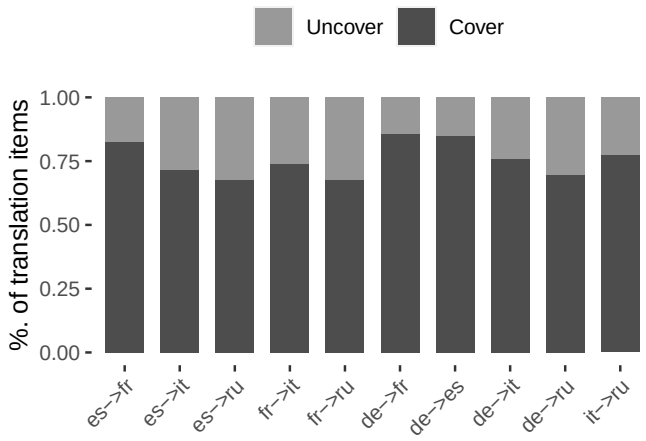

Figure 8: The coverage of translated items of the top-10 language pairs.

Answer to RQ4

When supporting app internationalisation, not all the terms have been translated to the targeted languages. Nevertheless, for such terms that are indeed translated, the translation tends to be consistent among different Android apps (with over 95% of consistency rate).

4. Automated App Internationalisation

Our preliminary study experimentally shows that the term translations provided by existing Android apps are reliable sources for mining practical term translations, which are essential to achieve automated app internationalisation. Motivated by those experimental results, we designed and prototyped an automated approach called *Androi18n* to help app developers more effectively implement app internationalisation.

4.1. Androi18n

Figure 9 illustrates the working process of our *Androi18n* tool, made up of two key modules. The two modules are named (1) Knowledge Base Construction and (2) Transformer-based Term Translation. We now briefly introduce these two modules, respectively.

Module #1: Knowledge Graph Construction: The first module of *Androi18n* constructs a large knowledge graph [18, 19] recording all the term translations in real-world Android apps provided by app developers. This module starts by disassembling real-world Android apps (from their bytecode format) and locating their internationalization-related resources. The terms and their practical translations are then extracted for building our *Androi18n* knowledge graph. This knowledge graph will then be leveraged by *Androi18n* to guide the second module to achieve automated term translations. In our extracted knowledge graph, we model each term in a language as a node and the connection between two nodes containing the same term but with different languages as an edge. For example, *Choose an image* in English and *Sélectionner une image* in French will be regarded as two independent nodes in the knowledge graph. Since these two terms are essentially equivalent (i.e., with the same meaning), the corresponding two nodes will be connected with an edge. To model the agreements among different apps for a given translation, we further assign each edge a weight,

Table 5: Top ten closed source Android apps language translation.

Language Pair	Consistent		Inconsistent	Total
	Case Sensitive	Case Insensitive		
es->fr	205,306 (93.26%)	6,695 (3.04%)	8,146 (3.70%)	220,147
de->fr	200,279 (92.82%)	6,971 (3.23%)	8,519 (3.95%)	215,769
de->es	199,358 (93.18%)	6,671 (3.12%)	7,917 (3.70%)	213,946
fr->it	178,267 (93.76%)	5,548 (2.92%)	6,310 (3.32%)	190,125
es->it	173,828 (93.73%)	5,536 (2.98%)	6,099 (3.29%)	185,463
de->it	172,621 (93.27%)	6,005 (3.24%)	6,446 (3.48%)	185,072
es->ru	162,299 (93.19%)	4,509 (2.59%)	7,357 (4.22%)	174,165
fr->ru	160,851 (93.32%)	4,198 (2.44%)	7,318 (4.25%)	172,367
de->ru	156,717 (93.05%)	4,503 (2.67%)	7,211 (4.28%)	168,431
it->ru	142,639 (93.18%)	3,969 (2.59%)	6,474 (4.23%)	153,082

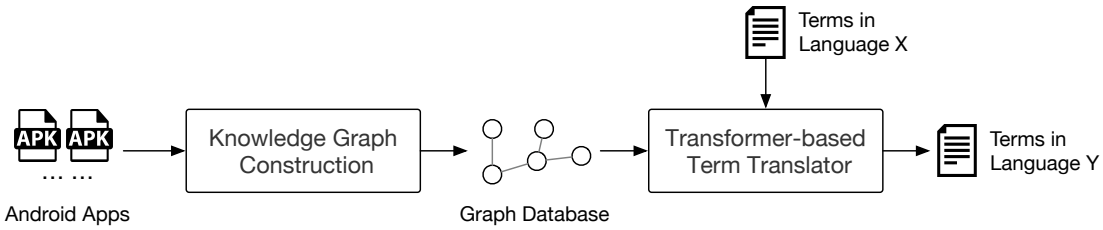

Figure 9: The working process of *Androi18n*.

indicating the number of apps that have shared the same translation.

Module #2: Transformer-based Term Translation: The second module of *Androi18n* takes the state-of-the-art Transformer model [20] to achieve our automated term translation objective. The transformer model is a deep learning method that adopts the self-attention mechanism to deferentially weigh the significance of each part of the input data, i.e., the ability to attend to different positions of the input sequence to compute a representation of that sequence. This model has been widely used (and demonstrated to be useful) in the fields of natural language processing and computer vision. In this work, the Transformer model will be trained based on the practical term translations recorded in the graph database.

Before feeding the text translations into the neural network, the model first embeds the texts into numerical vectors. After that, the model leverages the encoding blocks to extract the input's semantics layer by layer. Each layer here includes two sub-layers: multi-head attention mechanism and fully connected position-wise feed-forward network, which are responsible for mining the relationships between the words in the text and further extracting semantics in text sequences. Different from the encoding blocks, each decoding block includes three sub-layers, with an additional sub-layer called masked multi-head attention included at the beginning of each block. This additional sub-layer is designed to enable output generation in parallel.

We now evaluate the effectiveness and usefulness of *Androi18n* by answering the following two research questions.

- **RQ5:** *How effective is* Androi18n *in automatically translating terms in Android apps?*

- **RQ6:** *How useful is* Androi18n *in helping developers achieve automated Android app internationalisation?*

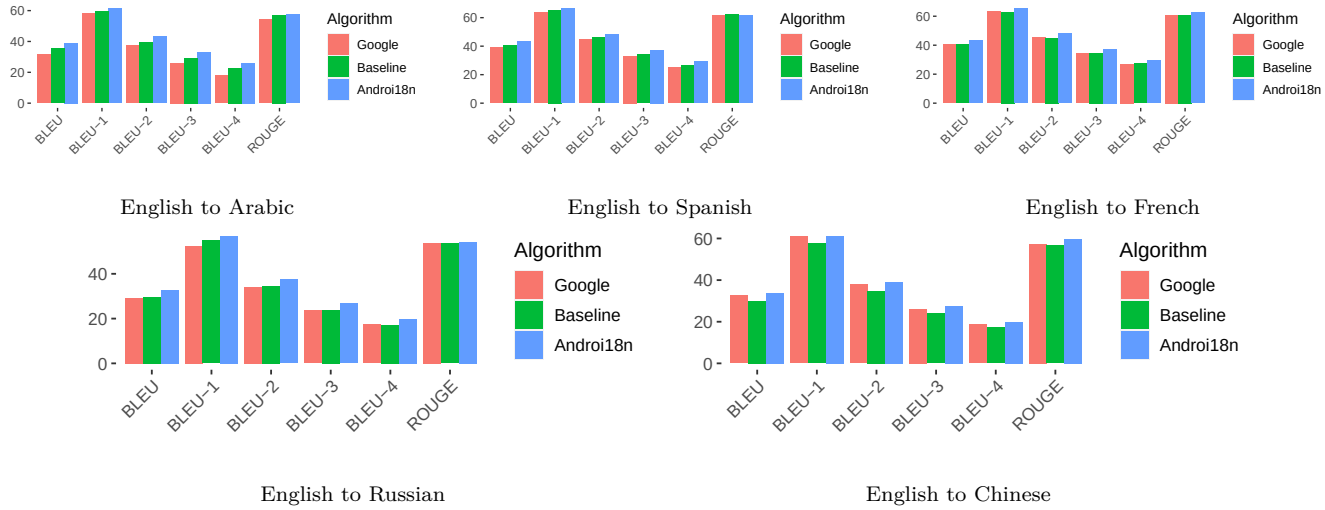

Figure 10: Translation results over Google-based, RNN-based baseline and *Androi18n*.

4.2. ***RQ5:*** *Effectiveness Evaluation*

Experimental Setup: To evaluate *Androi18n*'s automated app term translation capability, we first select 50,000 closed-source Android apps and then construct a knowledge graph with language texts extracted from these 50,000 different Android apps. We then query the translations between English and the other five United Nations (UN) official languages (e.g. Arabic, Chinese, French, Spanish, Russian) [21], and based on the obtained translation texts, we train our Transformer-based model to achieve automated term translation. For each language A to language B translation, the whole dataset would be split into train/valid/test sets with respectively 80%/10%/10% items. In the data pre-processing, the tokens/words that appear in the train set less than three times would be excluded, which is a common step in natural language processing. As for tokenization (or word segmentation) of the texts in train/valid/test sets, we selected different open-source tools for different languages. For example, we performed the widely-used NLPIR toolkit to tokenize Chinese and the well-known NLTK package for English tokenization. In the training process on the train set, we additionally used the valid set to choose the best-trained model. That is, the model with the best performance on the valid set is considered and subsequently be taken as the final trained model for testing.

We compare the performance of *Androi18n* against two baselines.

- **Baseline #1: Google Translate.** Google Translate is often regarded as one of the most performant translation services that have been widely used in the software engineering community. With the fast development of Natural Language Processing (NLP) techniques, the performance of Google Translate has been continuously improved. Many practical software applications (such as Transifex) have directly embedded it to achieve automated text translations. Therefore, in this work, we take Google Translate as the first baseline for comparison.

- **Baseline #2: The RNN-based approach proposed by Wang et al. [3].** Wang et al. [3] leverage an RNN [4] model to achieve domain-specific machine translation for software localization. To the best of our knowledge, this is only one work in the literature that is closely related to ours. They experimentally evaluated their approach based on a set of human-translated bilingual text pairs collected from different Android apps crawled from the official Google Play

Table 6: Metrics of different language translations

Language Translation	Algorithm	Metrics (%)					
		BLEU	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE
en -> ar	Google	31.75	57.94	37.55	26.10	17.88	54.40
	Baseline	35.37	59.24	39.42	29.29	22.89	57.08
	Androi18n	38.93	61.39	43.29	33.16	26.07	57.49
en -> es	Google	39.25	63.64	44.86	33.19	25.04	61.46
	Baseline	40.64	65.31	46.14	34.36	26.34	62.40
	Androi18n	43.16	66.46	48.28	37.22	29.05	61.68
en -> fr	Google	40.34	63.11	45.14	34.50	26.94	60.75
	Baseline	40.43	62.84	45.02	34.69	27.23	60.86
	Androi18n	43.16	65.36	48.12	37.38	29.52	62.44
en -> ru	Google	29.24	52.33	34.21	23.56	17.33	53.69
	Baseline	29.66	54.83	34.46	23.99	17.07	53.56
	Androi18n	32.65	56.84	37.49	26.72	19.97	54.13
en -> zh	Google	32.57	60.94	37.90	26.12	18.66	57.40
	Baseline	30.14	57.56	34.86	23.97	17.17	56.66
	Androi18n	33.75	61.13	39.09	27.39	19.83	59.72

Table 7: Effect Size of different metrics with regard to *Androi18n*

Algorithm	BLEU	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE
Google	0.75	0.62	0.68	0.74	0.81	0.44
Baseline	0.60	0.57	0.63	0.60	0.57	0.28

store. Their experimental results show that their approach is effective and can generate acceptable translation with fewer needs for human revisions. We thus consider it as one of our baselines to compare it w.r.t. the effectiveness of *Androi18n*.

For Google Translate, we directly leverage the Google Translate API[8] to obtain the translation results for the sampled data. For the approach proposed by Wang et al. [3], unfortunately, the authors have not made their tool implementation publicly available, and we cannot directly reuse their approach for comparison. To this end, we re-implemented a text-to-text translator based on the RNN encoder-decoder model (hereinafter referred to as Baseline) and use it for comparison. Following the strategy applied by Wang et al. [3],[9] we also add an attention mechanism and copy mechanism to improve the default RNN model.

Results: Figure 10 and Table 6 summarise our experimental results.For all five experiments, our *Androi18n* approach is able to outperform both the Google Translate and RNN-based baseline approaches, giving both higher BLEU and ROUGE scores. To be more specific, *Androi18n* is 3.7% and 3.08% higher than Google Translate and RNN-based baseline on average, respectively, in terms of BLEU. With regard to ROUGE, *Androi18n* is on average 1.55% and 0.98% higher than Google Translate and RNN-based baseline. Our Mann-Whitney-Wilcoxon (MWW) tests confirm that the performance differences between our approach and the two baseline approaches are all significant, i.e., the p-values are always smaller than 0.005. [10] The effect sizes with regard to our approach

[8]https://cloud.google.com/translate

[9]Wang et al. [3] have adopted three mechanisms to improve the performance of their RNN encoder-decoder neural translation. We have replicated two of them. The remaining one concerning the category information of collected Android apps is ignored as we don't have the category information of the randomly selected apps (AndroZoo does not provide category information at the moment).

[10]Given a significance level $\alpha = 0.005$, if p-value $< \alpha$, there is one

listed in Table 7 are all greater than 0.2 (the small effect size [11]). Especially, they are greater than 0.55 when it comes to the BLEU related metrics, representing that our experimental results are significantly different and our approach is better than others. BLEU (Bilingual Evaluation Understudy, the most important metric used in NLP community) [22] and ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [23] are the standard metrics to evaluate the performance of machine translations, for which the BLEU score measures the precision (how many words generated by machine appears in human summarises) while the ROUGE score measures the recall (how many words in human summaries appeared in the machine-generated summarises). Interestingly, Wang et al. [3] have demonstrated in their evaluation in 2019 that their approach achieves significantly better performance than Google. This is no longer the case, at least based on the results yielded by our reproduced version. This result is expected as Google constantly improves its translation service, and the latest version also relies on the Transfomer neural network. However, even with Google's improvements (including the model itself and the large-scale resources for training [20]), Google Translate still cannot outperform *Androi18n*, which directly leverages the original Transformer model (without improvements) and is only trained on the translations of a limited number of Android apps. This evidence experimentally shows that domain knowledge is vital for implementing machine translation, confirming our previous finding that it is possible to achieve automated app internationalisation by learning from existing apps.

chance in two hundred that the difference between the datasets is due to a coincidence.

[11] https://en.wikipedia.org/wiki/Effect_size

Answer to RQ5

Androi18n is effective in achieving automated app internationalisation, outperforming both Google Translate and the RNN-based state-of-the-art in terms of both BLEU and ROUGE scores.

4.3. ***RQ6:*** *Usefulness Evaluation*

Our approach has been experimentally demonstrated to be effective and be able to outperform two baselines. We now go one step further to evaluate the usefulness of the translations through a user study.

Experimental Setup: To fulfill this purpose, we first recruited six students who are all bilingual in English and Mandarin and then involved three professional Android app developers who are also familiar with English and Mandarin. We randomly selected 278 translation items (from English to Chinese) from our dataset for which the translated results are all different among *Androi18n*, the RNN-based translation, and the Google Translate baseline. The number of translation items was determined by the well-known Sample Size Calculator [24] with a confidence level of 95% and a margin of error of 5%. It gave out a sample size of 278 based on the total sample size 1,000. For each selected translation item, we set up a multiple-choice question that includes the original English sentence as the title and the outputs of the three approaches (in random order) as options. We further add an additional option for each question to indicate that the translations are almost the same among the three approaches. We then put all the generated questions onto a Google Survey Form and independently share it with the six students and the additional three professional developers. The recruited participants are asked to answer all the questions by ticking the best suitable option.

Results. The user study results clearly show in Table 8 that our approach generates more human acceptable translations than Google Translate and the RNN-based baseline, which have only received better votes for 68 and

Table 8: Number of sampled translated items voted better for the corresponding approaches.

Participant	Google	RNN-based	*Androi18n*	Same
6 Students	68 (24.46%)	21 (7.55%)	113 (40.65%)	57 (20.50%)
3 Developers	51 (18.35%)	20 (7.19%)	113 (40.65%)	75 (26.98%)
9 Participants	63 (22.66%)	23 (8.27%)	117 (42.09%)	63 (22.66%)

21 questions among six recruited students, respectively. For around one-fifth (57/278) of the cases, the users believe that the translated results are more or less the same among the three approaches. As for the three professional developers, their votes are generally consistent with the ones given by the students even though they categorise slightly more translated items into the same. In total, 42.09% (117/278) items performed better, i.e., our approach generally yields better translation results with the help of Transformer than with Google Translate or the RNN-based baselines. This result suggests that it is possible to achieve automated app internationalisation through machine translations, especially when trained over domain-specific knowledge. For example, the English text "Editing this mails will cost much data. Want to continue?" is translated into Chinese as "编辑此邮件将花费大量数据。想继续吗？", "编辑此邮件将花费较多数据。是否要继续？","编辑此邮件将耗费较多流量，是否继续？" by Google Translate, RNN-based baseline, and *Androi18n* , respectively. The translation clearly shows that our *Androi18n* outperforms other tools as it takes context and domain knowledge into consideration. The keyword "data" in the text is literally translated by Google Translate and RNN-based baseline into "数据" rather than the more meaningful term "流量" by *Androi18n*. This simple example demonstrates that context and convention would achieve better translation results if considered. Another English text "You received a security code" is also translated into "您收到了安全代码", "您收到了一个验证码", and "您已收到安全码" by Google Translate, RNN-based baseline, and *Androi18n*, respectively. The key phrase "security code" was translated by the RNN-based baseline into "验证码", which is not as accurate as the other two translations. Compared to Google Translation, *Androi18n*'s translation explicitly expresses the tense of the English text, leading to higher satisfaction for users who are familiar with Chinese.

Answer to RQ6

Androi18n is useful in achieving more human acceptable translations than both the Google Translate and the RNN-based baseline.

5. Discussion

5.1. Threats to Validity

The primary threat to external validity of our study concerns the choice of experimental Android projects. In this work, we selected different Android projects without considering their intended languages support for specific countries from AndroZoo and AndroZooOpen. Moreover, some of the projects in AndroZooOpen are not even published on the commercialized App Store. To mitigate this, we randomly selected projects from these two datasets, considered the open-source projects with public release information available, and selected a reasonably large number of apps to make the projects in this study more representative.

We assumed that all the text translations collected from real-world Android apps are validated by humans (hence are with high-quality). However, there is no such guarantee in practice. Nevertheless, as confirmed by a small empirical study conducted by Wang et al. [3], among 50 randomly selected apps, none of them have solely relied on machine translations to achieve mobile app text translation. This empirical evidence suggests that this threat will not significantly impact our empirical findings and experimental results.

Furthermore, except for looking at the self-recognized app category, we have not considered the actual scope of the selected apps (i.e., what are the apps primarily used

for?). We simply consider an app containing the internationalisation feature as long as it supports two or more languages. For example, some Android apps may be developed only to provide specific services for targeted regions or countries by supporting only the languages spoken by their residents, which may make the feature of internationalisation unnecessary. To mitigate this, we sample 100 apps[12] from the 5,000 selected Google Play apps used in RQ1 to manually check the impact of ignoring the scope of Android apps. We manually check the description of the sampled apps on the Google Play Store to determine if such a restriction exists for any app. Our exploration finds that none of our sampled apps has such restrictions in their description. Even though some of the apps are developed for specific countries, they still have various types of languages supported. For example, the app [25] moBILET provides a service to buy tickets for public transportation or pay for parking in Poland or Germany. Even though the app was developed for these two countries, it still provides more than 90 different languages for users to select.

The major threat to the construct validity of our study lies in possible error in the implementation of our experimental scripts and tools, such as the default language detection with the Python package polyglot, and the text string extraction with the special placeholder tag "<xliff:g>". To mitigate this threat, we have carefully reviewed the toolchain and manually validate partial experimental results against selected datasets.

Under different contexts, the same text expression may need to be translated into different forms. Such contexts, unfortunately, have not been considered in this work. The corresponding translation results hence may be less accurate under certain conditions. Nevertheless, our manual validation has not yet found such cases, indicating that this threat may not be significant. We have hence decided to mitigate this threat in our future work.

We conclude the effectiveness of our tool under the experimental translations from English to other five languages, including Arabic, Chinese, French, Spanish and Russian. These languages are the six official languages of the United Nations. However, the tool could have different performances in other language translations, which could threaten the usefulness and extensibility of the tool. We do believe better experimental settings should be designed to evaluate the genericity of our automated translation approach. We consider this as our potential future work.

5.2. Implications

The empirical findings and experimental results of this study have raised a number of opportunities for the research and practice communities.

Better release note description: In the study, we manually inspected the description of the sampled apps on the Google Play Store and found that almost all of the app descriptions do not clearly specify which languages the app supports, which is not user-friendly for app users. Therefore, we argue that app developers should describe the languages supported in the description intentionally. App users could comment for new language support if they are not familiar with any of the languages provided initially, which would boost the spread of the app.

Better internationalisation validation: Our study reveals that a few languages are repeatedly added and deleted during the evolution of the Android apps, even though some of the languages may be deleted intentionally by the developers. The inconsistent language support would leave a bad expression to app users and, thus, hinder the app's spread. To have a consistent and better user experience, a strict internationalisation validation procedure before release is necessary. Even for the intentionally deleted ones, a detailed specification in the release note is indispensable. Besides, we intend to investigate the developers to reveal the rationale behind the language addition

[12]The number is first calculated through a well-known sample size calculator [24] with a confidence level of 95% and a margin of error of 10% and then rounded to the nearest hundred.

and removal in the future work.

Better domain-specific training: In this work, we have experimentally demonstrated that the existing text translations conducted by app developers are an effective and useful basis for achieving automated app internationalisation. Nevertheless, we believe that the performance of our *Androi18n* approach can be further improved if better domain-specific information could be leveraged to help in training the model. For example, Wang et al. [3] have empirically shown that the category information of Android apps is useful for improving the performance of neural network-based text translation. Indeed, apps belonging to the same category are more likely to share the same patterns of translations, including the translations of conventional terms that are only available in certain categories. In this work, our closed-source apps are collected from AndroZoo, which does not include apps' category information. We have hence not explored the possibility of including category information to implement *Androi18n*.

Human-in-the-loop translation validation: In this work, we have constructed the translation graph database based on learning from term translations in existing Android apps. The translation accuracy (based on the number of similar translation occurrences in the published Android apps) has been demonstrated to be promising. However, the translation results obtained via the Transformer-based translator (if not directly matched over the database) are not always the case. There is a need to involve human efforts to confirm or revise the automatically translated results. The confirmed translations could then be written back into the database to avoid further human involvement. However, app developers may not always be familiar with the targeted language. The translated expressions given by the automated approaches may not be as accurate as what is expected by the developers but cannot be spotted. Therefore, we argue that crowd-sourced human-in-the-loop validation is required to achieve highly accurate automated app internationalisation.

Going beyond automated text translation: To achieve automated app internationalisation, in addition to automated text translation, there is another essential step needed. That is to set up the internationalisation environment and automatically identify all the hard-coded constant strings (scattered in the code) that need to be translated to other languages. Such constant strings need to be externalized to resource files, allowing translators to translate the app into other languages without actually modifying the app source code. There are no such automated approaches specifically proposed for mobile apps. Nonetheless, our community does have contributed various approaches for other types of applications, such as the *TranStrL* tool for Java GUI applications.

Translation of non-text content: Some of the apps' non-text content may also need to be translated when preparing for internationalisation. For example, iconic forms may only include textual content in a language, which unfortunately will not be translated by the current form of internationalisation. Indeed, they may need more sophisticated image-based analysis techniques to achieve the translation.

6. Related work

There are some IDEs, such as [26], indeed provide some mechanisms to highlight hard-coded string literals in source code. However, they just extract these strings into properties files for further internationalisation or just ignore them if internationalisation is not necessary for them. To the best of our knowledge, we are the first to extensively study the internationalisation of Android applications. However, there are several other works [3, 27–37] that have studied the internationalisation of other different systems. In this section, we summarise and discuss the most relevant ones.

Wang et al. [3] first proposed an RNN [4] model based on the Android apps downloaded from Google Play Store to

achieve a better domain-specific machine translation compared with the official Google Translate for software localization. They extracted closed-source apps and only focused on the translation process. Different from what they do, we not only worked on the translation process per se with a different more advanced translation model (Transformer) producing a better translation result but also first reveal the status quo and the evolution of internationalisation both for open-source and closed-source Android projects.

Hau et al. [29] explored some issues of software translation and localization in web based ERP [38] and presented an open source WebERP using in Portugal with local language for the artefact. ERP is always referred to Enterprise Resource Planning and usually treated as a category of business management software. What they realize is that various aspects of accounting that are different from the default setting in the WebERP are all needed to modify according to Portugal accounting.

Wang et al. [30] proposed an approach to automatically locate need-to-translate constant strings and also an Eclipse plug-in tool that locate need-to-translate constant strings in Java source code. The authors first select a set of API methods related to Graphical User Interface (GUI) and then locate need-to-translate strings from these API calls based on string-taint analysis. This approach is evaluated on four different real-world open source applications and the experimental results show the approach is promising. In addition, they also studied internationalisation of web applications [39] as their previous work cannot differentiate constant strings represented at the browser side (need-to-translate) from others not need. To tackle the challenge, they proposed a novel approach for PHP application. The approach first tries to locate constant strings that may be propagate to the browser side for displaying. Secondly, they add location information and further propagate the location flag to all the other terminals and non-terminals to identify user-visible constant strings.

Xia et al. [31] presented their study in software internationa-lization and localization. They studied a large-scale commercial system, PAM of State Street Corporation, which is written in C/C++ and contains more than 5 million lines of source code, and also proposed supporting tools IRanker and I18nLocator. IRanker is used to extract convertible and suspicious patterns based on the selected representative set of code in the most important source files while I18nLocator is leveraged to locate and convert source code.

Rey et al. [32] did an extensive study on the connection between localization and web accessibility, especially the possibility of transferring accessibility throughout the procedure of localization, that is to adapt a web product to another language (web localization). They analysed how does the current localization and internationalisation data exchange standards impact the transferring accessibility qualities and also explored the techniques proposed by the W3C to help web developers to tackle the challenges of transferability. Their preliminary findings have demonstrated that some accessibility features, especially those relevant with textual content and inter-semiotic purposes, can be captured, marked, transferred and annotated through existing mechanisms.

7. Conclusion

In this work, we have first conducted an extensive study on the internationalisation and localization of the popular mobile Android applications. Through this study, we experimentally find that most closed-source Android apps have been provided with internationalisation and each of the apps often supports multiple languages. Furthermore, the apps also tend to agree with each other when translating the same terms from one language to another, demonstrating the possibility of learning from them to achieve automated text translations. This evidence motivates us to go one step deeper to actually implement such a translator that leverages the translations of existing Android apps

to train a Transformer-based neural network model called *Androi18n* to achieve this objective. Ideally, *Androi18n* can support automated translations for as many language pairs as possible, i.e., based on the language supported by real-world Android apps. Experimental results show that *Androi18n* is both effective and useful in achieving automated text translations for Android apps.

Acknowledgements

Grundy is supported by ARC Laureate Fellowship FL190100035.

References

[1] *Smartphone Market Share*, 2020. [Online]. Available: https://www.idc.com/promo/smartphone-market-share/os

[2] *Mobile App Internationalization*, 2019. [Online]. Available: https://medium.com/@infopulseglobal_9037/mobile-app-internationalization-ways-and-methods-to-boost-revenue-by-26-4f8985d3c4bd

[3] X. Wang, C. Chen, and Z. Xing, "Domain-specific machine translation with recurrent neural network for software localization," *Empirical Software Engineering*, vol. 24, no. 6, pp. 3514–3545, 2019.

[4] *Recurrent Neural Network*, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Recurrent_neural_network

[5] "Android internationalization exploratory study," 2022, https://zenodo.org/record/6609238.

[6] T. V. Luong, J. S. Lok, D. J. Taylor, and K. Driscoll, *Internationalization: Developing software for global markets*. John Wiley & Sons, Inc., 1995.

[7] *Tutorial internationalization*, 2021. [Online]. Available: https://docs.oracle.com/javase/tutorial/i18n/intro/quick.html

[8] *Example project*, 2022. [Online]. Available: https://github.com/ccruz17/Launcher-Play/tree/master/app/src/main/res

[9] *Android Alternative Resources*, 2021. [Online]. Available: https://developer.android.com/guide/topics/resources/providing-resources?authuser=1#AlternativeResources

[10] P. Liu, L. Li, Y. Zhao, X. Sun, and J. Grundy, "Androzooopen: Collecting large-scale open source android apps for the research community," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 548–552.

[11] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016, pp. 468–471.

[12] *Android Asset Packaging Tool*, 2021. [Online]. Available: https://developer.android.com/studio/command-line/aapt2

[13] *graph-database*, 2021. [Online]. Available: https://pypi.org/project/polyglot/

[14] *Android Apps remove on Google Play Store*, 2022. [Online]. Available: https://www.bgr.in/news/google-to-remove-nearly-900k-abandoned-apps-from-play-store-report-1270875/

[15] J. Gao, P. Kong, L. Li, T. F. Bissyandé, and J. Klein, "Negative results on mining crypto-api usage rules in android apps," in *The 16th International Conference on Mining Software Repositories (MSR 2019)*, 2019.

[16] J. Gao, L. Li, P. Kong, T. F. Bissyandé, and J. Klein, "Understanding the evolution of android app vulnerabilities," *IEEE Transactions on Reliability (TRel)*, 2019.

[17] *Sample Andriod project*, 2022. [Online]. Available: https://github.com/edipo2s/TESLegendsTracker

[18] A. Singhal, *Knowledge Graph*, 2021. [Online]. Available: https://blog.google/products/search/introducing-knowledge-graph-things-not/

[19] L. Ehrlinger and W. Wöß, "Towards a definition of knowledge graphs." *SEMANTiCS (Posters, Demos, SuCCESS)*, vol. 48, no. 1-4, p. 2, 2016.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[21] *Union Nation Languages*, 2021. [Online]. Available: https://www.un.org/en/our-work/official-languages

[22] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[23] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[24] *Sample Size Calculator*, 2022. [Online]. Available: https://www.surveysystem.com/sscalc.htm

[25] *Sample Android App: moBILET*, 2022. [Online]. Available: https://play.google.com/store/apps/details?id=pl.mobilet.app

[26] *Hard-coded string literals*, 2021. [Online]. Available: https://www.jetbrains.com/help/idea/hard-coded-string-literals.html

[27] A. Alameer, S. Mahajan, and W. G. Halfond, "Detecting and localizing internationalization presentation failures in web applications," in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2016, pp. 202–212.

[28] P.-L. P. Rau and S.-F. M. Liang, "Internationalization and localization: evaluating and testing a website for asian users," *Ergonomics*, vol. 46, no. 1-3, pp. 255–270, 2003.

[29] E. Hau and M. Aparício, "Software internationalization and localization in web based erp," in *Proceedings of the 26th annual ACM international conference on Design of communication*, 2008, pp. 175–180.

[30] X. Wang, L. Zhang, T. Xie, H. Mei, and J. Sun, "Transtrl: An automatic need-to-translate string locator for software internationalization," in *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009, pp. 555–558.

[31] X. Xia, D. Lo, F. Zhu, X. Wang, and B. Zhou, "Software internationalization and localization: An industrial experience," in *2013 18th International Conference on Engineering of Complex Computer Systems*. IEEE, 2013, pp. 222–231.

[32] J. T. del Rey and L. M. Vázquez, "Transferring web accessibility through localization and internationalization standards," *The Journal of Internationalization and Localization*, vol. 6, no. 1, pp. 1–24, 2019.

[33] D. P. Rich, "Method and system for improved software localization," Jul. 26 2011, uS Patent 7,987,087.

[34] A. Burukhin, M. A. Gadre, A. M. Aldahleh, T. Farrell, and J. L. Larrinaga-Pardo, "Dynamically providing a localized user interface language resource," Apr. 9 2009, uS Patent App. 11/869,083.

[35] C. Fitzpatrick, J. P. Whelan, R. P. Doyle, J. G. Lane, B. Mchugh, T. Farrell, P. Barnes, A. M. Mcquaid, and D. Mowatt, "Dynamic screentip language translation," Dec. 17 2013, uS Patent 8,612,893.

[36] C. Escobar-Velásquez, A. Donoso-Diaz, and M. Linares-Vásquez, "Itdroid: A tool for automated detection of i18n issues on android apps," in *2021 IEEE/ACM 8th International Conference on Mobile Software Engineering and Systems (MobileSoft)*. IEEE, 2021, pp. 52–55.

[37] L. A. Reina and G. Robles, "Mining for localization in android," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 2012, pp. 136–139.

[38] *Enterprise Resource Planning*, 2021. [Online]. Available: https://www.oracle.com/au/erp/what-is-erp/

[39] X. Wang, L. Zhang, T. Xie, H. Mei, and J. Sun, "Locating need-to-translate constant strings in web applications," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, 2010, pp. 87–96.