# LinkRadar: Assisting the Analysis of Inter-app Page Links via Transfer Learning

Diandian Gu
Peking University, Beijing, China
gudiandian1998@pku.edu.cn

Ziniu Hu
University of California, Los Angeles, USA
bull@cs.ucla.edu

Shangchen Du
Peking University, Beijing, China
dscinpku@pku.edu.cn

Yun Ma*
Tsinghua University, Beijing, China
yunma@tsinghua.edu.cn

## ABSTRACT

Analyzing links among pages from different mobile apps is an important task of app analysis. Currently, most efforts of analyzing inter-app page links rely on static program analysis, which produces a lot of false positives, requiring significant manual effort to verify the links. To address the issue, in this paper, we propose LinkRadar, a data-driven approach to assisting the analysis of inter-app page links. Our key idea is to use dynamic program analysis to gather a set of actual inter-app page links, based on which we train a model to predict whether there exist links among pages from different apps to help verify the results of static program analysis. The challenge is that inter-app page links are hard to be triggered by dynamic program analysis, making it difficult to collect enough inter-app page links to train the model. Considering the similarity between intra-app page links and inter-app page links, we use transfer learning to deal with the data scarcity problem. Evaluation results show that LinkRadar is able to infer the inter-app page links with high accuracy.

## CCS CONCEPTS

• **Information systems → Data mining**; • **Computing methodologies → Machine learning**;

## KEYWORDS

Inter-app page links; Transfer learning; Dynamic program analysis; Link prediction

**ACM Reference Format:**
Diandian Gu, Ziniu Hu, Shangchen Du, and Yun Ma. 2019. LinkRadar: Assisting the Analysis of Inter-app Page Links via Transfer Learning. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3357384.3358094

*Corresponding author.

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.
*CIKM '19, November 3–7, 2019, Beijing, China*
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3358094

## 1 INTRODUCTION

Pages from different mobile apps are sometimes linked with each other to provide additional services for mobile users [7, 10]. For example, a hotel page from a travel app may contain a phone number of the hotel, and such a page may be linked with the dialing page of the Skype app for users to call the hotel. Analyzing such inter-app page links is an important task of app analysis, since these links may have potential vulnerabilities, such as malicious data access, sensitive data theft, and privilege-escalation attacks.

Currently, most efforts of analyzing inter-app page links rely on static program analysis, which analyzes the source code or bytecode of apps to extract inter-component communication among different apps for link detection. However, it is reported that these approaches suffer from large amount of false positives, meaning that many links reported by the static analysis do not exist in fact [8]. So it requires significant manual effort to verify the analysis results.

To address the issue, in this paper, we propose LinkRadar, a data driven approach to assisting the analysis of inter-app page links. For apps under analysis, we adopt dynamic program analysis to obtain a set of actual inter-app page links, based on which we train a model to infer inter-app page links among other pages. In the training process, we first extract features of app pages to learn their low-dimensional representations. These representations are then integrated to estimate the likelihood of the existence of a link between app pages.

However, it is challenging for dynamic analysis tools to get enough samples of inter-app page links.The reason is that in order to trigger an inter-app page link, dynamic analysis tools have to reach certain positions in an app and perform specific actions on certain UI elements, which cannot be afforded by state-of-the-art dynamic analysis tools. So it is hard learn the patterns of inter-app page links directly. We notice the similarities between intra-app and inter-app page links. Therefore, we pre-train our model with a large amount of intra-app page links and then fine tune this model with small number of inter-app page links. Our learning framework has the following major traits:

- **Data-driven approach to assisting inferring inter-app page link.** There have been lots of studies on app analysis, but to the best of our knowledge, we are the first to assist static analysis tools to infer inter-app page links with data-driven method.
- **Transfer learning.** Inter-app page links are difficult to collect. So it can be difficult to learn from these inter-app links

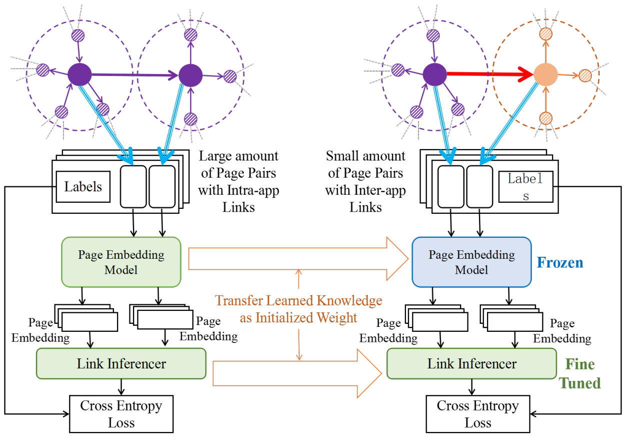

Figure 1: Schematic overview of LinkRadar

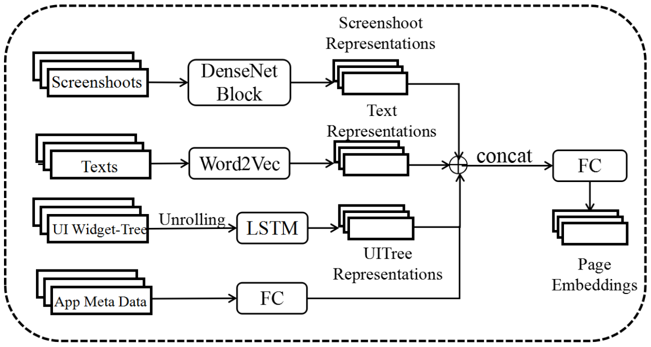

Figure 2: Structure of Feature Extractor

directly. We leverage intra-app link inference task for transfer learning, to deal with low-resource setting.

- **Low-dimensional vector embedding of app pages.** Instead of extracting features from developers' hard-coded logic, we use a contextualized and hybrid page encoder to generate embeddings that can represent app pages.

We evaluate LinkRadar based on the data collected from 1625 popular Android apps. Results show that LinkRadar is able to infer inter-app page links with high accuracy, demonstrating the effectiveness of LinkRadar on assisting the static program analysis of inter-app page links.

## 2 PROPOSED METHOD

In order to verify the inter-app page links reported by static program analysis tools, we propose to use dynamic program analysis tools to execute the apps under analysis, collecting actual inter-app page links, based on which we train a model to infer whether there exist inter-app page links among other pages that are not covered by dynamic analysis tools. Therefore, the core part of our method is how to infer inter-app page links based on the information collected by dynamic program analysis.

In the following part of this section, we formulate the prediction problem, describe the model of LinkRadar, and show how we train our model via transfer learning.

### 2.1 Problem Definition

Through the paper, the set of app pages, which are the user interfaces of apps, is denoted as $\mathcal{P}$. An app page $p_i \in \mathcal{P}$ represents the context of the $i$th app page in the whole data set, including app meta data, image information, text information, etc.

Given two app pages from two apps $p_i, p_j \in \mathcal{P}$, we estimate the probability of the existence of a page link between $p_i$ and $p_j$ as $p(L = 1 \mid p_i, p_j)$, where L is a binary variable indicating whether there is a link between $p_i$ and $p_j$ (L = 1) or not (L = 0). The link probability in LinkRadar is estimated as follows:

$$p(L = 1|p_1, p_2) = \Psi(\phi(p_1), \phi(p_2)) \tag{1}$$

where $\phi$ denotes the app page embedding component and $\Psi$ is an app link inference component that takes the two pages' embedding and generates an app link inference score. The score indicates how likely it is that there is an app link between $p_1$ and $p_2$.

To make a better use of app pages' structural feature, we build an app-page graph with app pages as nodes and page links as edges. For a certain app page, the app pages that have links with that page are the page's *neighbors*. We use pairs of app pages as the input of LinkRadar. To distinguish different types of page pairs, we call two app pages from the same app an *intra-app page pair* and two app pages from different apps an *inter-app page pair*. Similarly, we have *intra-app page link* and *inter-app page link*.

In the training process of LinkRadar, we first collect app link information and page information with a dynamic analysis tool. Then, we train the model of LinkRadar with the data collected. Figure 1 shows the overview of the training pipeline of LinkRadar, which can be divided into two processes: pre-training and fine tuning. The link inference model consists of two parts: the Page Embedding Model and the Link Inferencer.

### 2.2 The Link Inference Model

*2.2.1 Page Embedding Model.* The Page Embedding Model (PEM) encodes an app page to a vector embedding. Feature Extractor, which is a neural network, is the core part of PEM. As is shown in Figure 2, PEM extracts features of app pages to generate an app page's embedding vector.

- **Screen snapshot**: The screen snapshot displays all visible fragments of the page in one image. It transmits a lot of information to app users in negligible amount of time. The Feature Extractor uses one block of DenseNet to get useful information from these images.
- **Text information**: Users can easily understand the page's content and function by reading the text on screen. To utilize text information, we segment the text and use a pre-trained Word2Vec [12] model to create word representations of each word in an app page. Then, we calculate the average word embedding to represent the overall meaning of that page.
- **Screen layout**: In Android apps, each activity manages a screen of the User Interface and contains a set of UI elements called widgets. The screen layout is a tree that contains all the widgets of an app page. App developers can either use

existing widgets or design their own widgets by extending View, which is the basic building block for UI components. Therefore, the name of a widget's superclass, which we regard as the widget's "tag", contains information about the basic function of this widget. Apart from tags, the position and size of widgets can also describe these elements. We traverse all widgets on the screen layout of an app page via in-order tree traversal to generate a widget sequence. We use one-hot embedding to represent tags and adapt one layer of LSTM to learn the representation of the widget sequence.

- **App Meta Data**: App meta data such as app category contains summation information of the app's function. In PEM, we use one-hot embeddings to represent categories of apps.

Feature Extractor takes the above features as inputs and outputs a page embedding. Notice that normally the feature information of a node is highly related to its neighbors in a graph. Motivated by GraphSAGE [5, 6], we combine an app page's embeddings given by Feature Extractor and its neighbors' embedding to aggregate a final representation of the app page. We use $\mathcal{N}$: v$\longrightarrow 2^{v}$ to represent the neighborhood function. In LinkRadar, where neighborhood information is only additional to the feature of a node, a simple aggregator like *Mean Aggregator* is effective enough for PEM.

Given an app page $p_i \in \mathcal{P}$, $h_{p_i}$ represents the embedding given by Feature Extractor. *Mean Aggregator* takes the element-wise mean of the vectors in $\{h_{p_j}, \forall p_j \in \mathcal{N}(p_i)\}$. The final embedding of $p_i$ can be generated by the following:

$$\phi(p_i) = CONCAT(h_{p_i}, MEAN(h_{p_j}, p_j \in \mathcal{N}(p_i))) \tag{2}$$

The adoption of inducting features from neighborhood improves the performance of LinkRadar greatly (see Section 3.3).

*2.2.2 Link Inferencer.* Motivated by neural tensor network (NTN) [14], we build a module called Link Inferencer to explore the relationship between representation pairs given by PEM. Comparing to traditional methods, NTN provides a more powerful way to model relational information.

Given two app pages $p_i, p_j \in \mathcal{P}$, Link Inferencer computes a score of how likely it is that two pages are in a certain relationship by the following NTN-based function:

$$\Psi(\phi(p_i), \phi(p_j)) = u_R^T f(\phi(p_i)^T W_R^{[1:k]} \phi(p_j) + V_R \begin{bmatrix} \phi(p_i) \\ \phi(p_j) \end{bmatrix} + b_R) \tag{3}$$

where f = tanh is a standard non-linearity applied element-wise and $W_R^{[1:k]} \in \mathbb{R}^{d\times d\times k}$ is a tensor. The bilinear tensor product $\phi(p_i)^T W_R^{[1:k]} \phi(p_j)$ results in a vector $d \in R^k$, where each entry is computed by one slice m = 1,...,k of the tensor: $\phi(p_i)^T W_R^{[m]} \phi(p_j)$. The other parameters for relation $R$ are the standard form of a neural network: $V_R \in \mathbb{R}^{k\times 2d}$ and $U \in \mathbb{R}^k$, $b_R \in \mathbb{R}^k$ [14].

The app-page graph is usually sparse, which means there are far more page pairs without links than page pairs with links in the graph. To better learn app link patterns from our data set, we apply the method of negative sampling, by which we sample a few page pairs without links randomly as a part of the training set.

## 2.3 Transfer learning

With few inter-app page links and a large amount of intra-app page links collected, it is difficult to learn the pattern of inter-app page links directly. However, both inter-app links and intra-app links connect app pages that are related in content and different in functions. For example, an app page that provides information of a hotel can be linked to an app page that contains comments of this hotel by an intra-app app link. The page can also be linked to a "map page" from a map app. Both target pages are about the certain hotel and have a different function from the source page. Considering this similarity, we transfer the shared knowledge between these two types of page links to alleviate the data sparsity problem.

As is shown in Figure 1, we first train LinkRadar with large amounts of page pairs with intra-app links. Then, we transfer what we learned as initialized weights to learn on the inter-app data. PEM is always fixed in the process of transfer learning. The parameters of the Link Inferencer are set trainable to adapt the patterns typical of inter-app page links.

## 3 EXPERIMENT RESULT AND ANALYSIS

In this section, we evaluate the performance of LinkRadar in comparison with baseline models, and analyze the results.

### 3.1 Data and Baselines

Our data set was collected with the help of a dynamic analysis tool called Paladin [11]. Paladin systematically explores apps by sending streams of user events to Android system while extracting app linkage information and app page information at the same time. Regarding app pages as "nodes" and user events that trigger page change as "edges", the app-page graph that describes the linkage information of apps is formed. Paladin also collects information such as screen snapshots, text, widget details and so on. We collected 44,308 intra-app page links and 4,279 inter-app page links from 1,625 popular apps on two app marketplaces, Google Play and Wandoujia. We take 70% app pages and their linkage relationship as training set, 10% app pages and their linkage relationship as validation, and 20% as test set.

We compare our model with three baseline methods: *Random* which generates inference scores randomly, *LinkRadar-noF* which is trained without fine tuning and *LinkRadar-noN* which is trained without neighborhood information.

### 3.2 Evaluation Method and Evaluation Metrics

We evaluate the effectiveness of LinkRadar with four evaluation metrics: Precision, Recall, Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR).

Precision and Recall are calculated on the test data set directly. They mainly evaluate how well the LinkRadar is able to estimate the relationship between two app pages. Considering that Paladin cannot collect all the inter-app page link, we choose MAP and MRR to measure how well LinkRadar can infer missing app page links.

To evaluate whether LinkRadar can be used to assist static analysis tools, we calculate the MAP and MRR based on the results of PRIMO [13], which is a state-of-the-art static program analysis tool. First, we randomly chose 50 app pages as "source pages" and for each "source", we got a set of "target pages" from the result given by PRIMO. Then, We manually validated every inter-app link to see which inter-app links really exist and evaluated LinkRadar on it. LinkRadar can select a few pages that are most probably linked

**Table 1: Performance Comparison with Baselines**

| | Precision | Recall | MAP | MRR |
|---|---|---|---|---|
| LinkRadar | 0.71 | 1.00 | 0.257 | 0.412 |
| Random | 0.07 | 0.44 | 0.089 | 0.103 |
| LinkRadar-NoF | 0.33 | 0.88 | 0.206 | 0.253 |
| LinkRadar-NoN | 0.32 | 0.95 | 0.227 | 0.280 |

with the "source pages" from a large number of "targets" given by PRIMO, meaning that LinkRadar can tell app page linkage relationship more precisely than static analysis tools. MAP and MRR can be calculated as follows:

$$MAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q} \tag{4}$$

$$MRR = \frac{1}{Q} \sum_{i=1}^{Q} \frac{1}{rank_i} \tag{5}$$

where Q is the number of source app pages, rank$_i$ is the rank position of the first real target page that LinkRadar predicts to be linked with the i-th source app page, AveP is the function that calculates the average precision of the prediction results of one certain source page.

## 3.3 Results and Discussion

We evaluate the performance of LinkRadar and study the impact of neighborhood aggregation.

**Overall Performance Comparison**: Table 1 lists the performance of our proposed method as well as its variations and baselines. It can be seen that LinkRadar outperforms all the baselines by a large margin in terms of all evaluation metrics. This means LinkRadar can not only infer whether an inter-app link should exist accurately, but also tend to give page pairs with inter-app links a high score. We see a 115% relative improvement in terms of precision and a 62.8% relative improvement in terms of MRR over LinkRadar-noF. This result suggests that fine tuning the model with inter-app data enables LinkRadar to capture inter-app page link patterns in a more generalized way.

**Impact of Neighborhood Aggregation**: To evaluate the impact of neighborhood aggregation, we compare the performance of LinkRadar with LinkRadar-noN, which is trained after removing neighborhood features. We see that in both evaluation metrics, the performance drops, suggesting that neighborhood contains important information of the app pages, and neighborhood aggregation plays an important role in the final performance of LinkRadar.

## 4 RELATED WORK

App link analysis is important to mobile applications analysis. Inter-Component Communication (ICC) analysis, which is required to understand how the components of Android applications interact, has been performed in past work. Dynamic analysis [1, 3] has attempted to enforce security policies related to ICC. Apposcopy [4] uses static analysis as the basis of a signaturebased malware detection system.

Recently, more and more inter-app analysis have been performed. ComDroid [2] analyzed inter-app communication in Android apps and discovered inter-app communication vulnerabilities. To better execute inter-app analysis, Li et. al. [9] proposed ApkCombiner, which combines different apps into a single apk on which existing tools can indirectly perform inter-app analysis.

## 5 CONCLUSION

In this paper, we presented LinkRadar, which infers inter-app page links with representation learning method and transfer learning method. LinkRadar learns a representation for app pages, and then incorporating app page information and neighborhood information into the model. We trained the model on more than 30,000 app pages collected from 1,625 popular Android apps. Results showed that LinkRadar is able to capture the patterns of inter-app page links, outperforming baselines significantly in terms of all evaluation metrics.

Given the various applications of app link analysis, LinkRadar can not only be used to help static analysis tools for Android apps, in the future, it can be applied to detect malicious data access, sensitive data theft, and privilege-escalation attacks as well.

## ACKNOWLEDGMENTS

This work was supported by the China Postdoctoral Science Foundation, and an Open Project Fund of National Engineering Lab of Big Data System Software of China.

## REFERENCES

[1] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Thomas Fischer, Ahmad-Reza Sadeghi, and Bhargava Shastry. 2012. Towards Taming Privilege-Escalation Attacks on Android. (01 2012).

[2] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. 2011. Analyzing inter-application communication in Android. In *International Conference on Mobile Systems, Applications, and Services*. 239–252.

[3] Michael Dietz, Shashi Shekhar, Yuliy Pisetsky, Anhei Shu, and S. Wallach Dan. 2011. QUIRE: Lightweight Provenance for Smart Phone Operating Systems. *Dissertations & Theses - Gradworks* (2011), 23–23.

[4] Yu Feng, Saswat Anand, Isil Dillig, and Alex Aiken. 2014. Apposcopy: Semantic-based detection of android malware through static analysis. In *Acm Sigsoft International Symposium on Foundations of Software Engineering*.

[5] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. (2017).

[6] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. (2017).

[7] Ziniu Hu, Yun Ma, Qiaozhu Mei, and Jian Tang. 2017. Roaming across the Castle Tunnels: an Empirical Study of Inter-App Navigation Behaviors of Android Users. *CoRR* abs/1706.08274 (2017). arXiv:1706.08274 http://arxiv.org/abs/1706.08274

[8] B. Johnson, Yoonki Song, E. Murphy-Hill, and R. Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *International Conference on Software Engineering*.

[9] Li Li, Alexandre Bartel, TegawendÃĪ F. BissyandÃĪ, Jacques Klein, and Yves Le Traon. 2015. *ApkCombiner: Combining Multiple Android Apps to Support Inter-App Analysis*.

[10] Yun Ma, Ziniu Hu, Yunxin Liu, Tao Xie, and Xuanzhe Liu. 2018. Aladdin: Automating Release of Deep-Link APIs on Android. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*. 1469–1478. https://doi.org/10.1145/3178876.3186059

[11] Yun Ma, Yangyang Huang, Ziniu Hu, Xusheng Xiao, and Xuanzhe Liu. 2019. Paladin: Automated Generation of Reproducible Test Cases for Android Apps. 99–104. https://doi.org/10.1145/3301293.3302363

[12] Tomas Mikolov, Kai Chen, G.s Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *Proceedings of Workshop at ICLR* 2013 (01 2013).

[13] Damien Octeau, Somesh Jha, Matthew Dering, Patrick McDaniel, Alexandre Bartel, Li Li, Jacques Klein, and Yves Le Traon. 2016. Combining Static Analysis with Probabilistic Models to Enable Market-Scale Android Inter-Component Analysis. In *Proceedings of the 43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*.

[14] R. Socher, D. Chen, C. D. Manning, and A. Y. Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *International Conference on Neural Information Processing Systems*.