Mobile Feature-oriented Knowledge Base Generation Using Knowledge Graphs

Quim Motger[1][0000−0002−4896−7515], Xavier Franch[1][0000−0001−9733−8830], and Jordi Marco[2][0000−0002−0078−7929]

[1] Department of Service and Information System Engineering, Universitat Politècnica de Catalunya
{joaquim.motger,xavier.franch}@upc.edu
[2] Department of Computer Science, Universitat Politècnica de Catalunya
jordi.marco@upc.edu

Abstract. Knowledge bases are centralized repositories used for developing knowledge-oriented information systems. They are essential for adaptive, specialized knowledge in dialogue systems, supporting up-to-date domain-specific discussions with users. However, designing large-scale knowledge bases presents multiple challenges in data collection and knowledge exploitation. Knowledge graphs provide various research opportunities by integrating decentralized data and generating advanced knowledge. Our contribution presents a knowledge base in the form of a knowledge graph for extended knowledge generation for mobile apps and features extracted from related natural language documents. Our work encompasses the knowledge graph completion and deductive and inductive knowledge requests. We evaluated the effectiveness and performance of these knowledge strategies, which can be used as on-demand knowledge requests used by third-party software systems.

Keywords: knowledge base · knowledge graph · mobile app

1 Introduction

Knowledge-based chatbots are dialogue systems embedding real-time, centralized access to domain-specific information systems [16]. These knowledge bases (KB) assist in resolving user intent and entity recognition tasks for a particular knowledge domain. With the emergence of disruptive large language models (LLMs) like GPT-4 [18], these KBs offer great potential in terms of accuracy, scalability and performance efficiency for highly adaptive knowledge modelling [29]. While LLMs excel in various tasks they are pre-trained or fine-tuned for, re-training for dynamic knowledge adaptation requires significant time, energy consumption and economic expenses [19].

In this context, one of the main challenges is extending these KBs with advanced knowledge generation techniques. To this end, the use of knowledge graphs (KG) as the underlying infrastructure of a KB is becoming a research trend [8]. The combined use of KGs with machine/deep learning techniques reveals the potential of leveraging graph-structured data towards effective and scalable extended knowledge generation [24, 25]. Nevertheless, constructing and exploiting a KG for a given domain is still a challenging task in terms of KG completion and the design of derived knowledge strategies [8]. Mobile applications and app stores exemplify highly-adaptive knowledge domains [20]. Google Play, the leading global app store [3], releases an average of 75K new apps monthly [4]. Among their top 1000 downloaded apps, 62% are updated at least once a month [1]. Given this context, feature-based knowledge generation, which leverages the knowledge exposed by documented app functionalities, emerges as a relevant research area, benefitting from the abundance of large natural language corpora (e.g., app descriptions, user reviews) in these repositories [10, 13].

In this paper, we present the design and development of a KB in the form of a KG supporting feature-oriented extended knowledge generation in the field of mobile app catalogues. Our main contributions are: (1) a distributable knowledge graph in the field of mobile app repositories to support document-based and feature-oriented data management tasks; (2) a state-of-the-art method for feature-oriented extended knowledge generation in the context of mobile apps; and (3) an effective, scalable approach for designing and developing a semantic web-based KB in the context of LLM-based chatbots for real-time consumption.

2 Background

Based on the notation from Zhao et al. [31], we denote a KG instance as $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ denotes the nodes of the graph and $E = \{e_1, e_2, ..., e_m\}$ denotes the edges in a labelled directed graph [8]. Each e is defined as (v_i, p, v_j), where v_i is the source node, v_j is the target node and p is the relation type from v_i to v_j. A particular type of labelled directed graphs are *Resource Description Framework* [22] or RDF graphs. They are built as semantic web networks based on subject-predicate-object (v_i,p,v_j) triples. The specification of V and E entities is built through a data-modelling vocabulary known as the RDF Schema, for which there are public repositories defining shared schemas for modelling structured data on the Internet. Knowledge representation and extraction in dynamic, large-scale KGs involve deductive and inductive techniques [8]. Deductive techniques entail precise, logic-based transformations $G \rightarrow G'$ based on particular data observations, while inductive techniques involve pattern generalizations within a given G which, although potentially imprecise, can be used to generate novel and complex predictions [8].

A particular instance G is a context-specific representation of real world entities and relations. This research focuses on the context of mobile software ecosystems [7], which involves *actors* (e.g., users, developers, platform vendors) and *entities* (e.g., apps, app stores, mobile OS platforms). From the user perspective, these ecosystems provide access to a *catalogue* or a set of *mobile apps*, which users access to build their own application *portfolio* [17]. A potential descriptor for mobile apps is the set of *features* they expose, representing functionalities from the user's perspective, implemented by one or multiple mobile apps in their portfolio [17]. Extended knowledge about app features plays a crucial role in various tasks, such as optimization ranking algorithms [26] and version modelling for app recommendation [15]. Research in this area primarily focuses on using available *documents* through natural language processing (NLP) techniques for tasks like feature extraction and topic modelling [13, 26, 15].

3 Research method

3.1 Project vision

Using the *Goal Question Metric* (GQM) template, we defined the following goal:

Analyse extended feature-based knowledge generation supported by KGs
for the purpose of adaptive KB generation for LLM-based dialogue systems
with respect to the correctness and efficiency of extended knowledge requests
from the point of view of dialogue system developers
in the context of mobile software repositories.

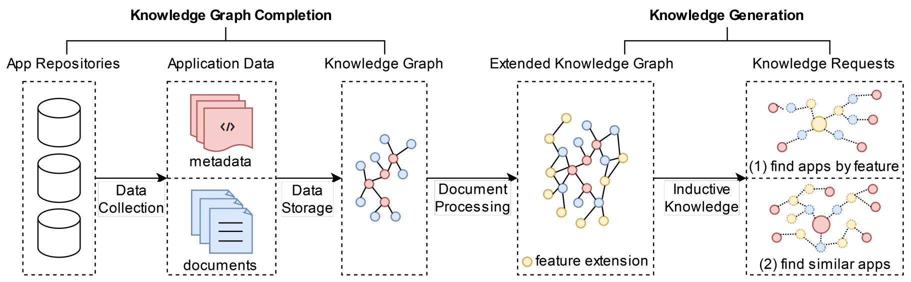


Fig. 1: Knowledge base: project vision

We define two types of feature-based knowledge generation requests:

1. Given a feature f, which mobile apps from the catalogue expose a set of features equivalent or similar to f?
2. Given a mobile app m from the catalogue, which mobile apps are more similar to m based on the features they expose?

Figure 1 summarizes the KB generation process. KG completion involves decentralized data collection from web-based app repositories, followed by modeling and storage in the KG. Knowledge generation extends the KG using deductive techniques based on NLP for feature extraction from available documents. The extended KG exposes knowledge requests for third-party software systems. Complementary materials and resources are available in the replication package[3].

[3] Available at https://doi.org/10.5281/zenodo.8038301

3.2 Research plan

To guide the design of this KB, we define the research question (RQ) as follows:

RQ. How does KB development in mobile app repositories benefit from KGs and extended knowledge requests? Specifically, how does this approach contribute to correctness, efficiency, and adaptability?

To respond to this RQ, we conducted an evaluative sample study [27], structured through the following stages: (1) **sample study definition**, including scope, specifications and limitations of the sample data set; (2) **KG completion**, including data source selection and data collection techniques; (3) **knowledge generation**, including feature-oriented, NLP-based inductive and deductive knowledge strategies; and (4) **evaluation** of the aforementioned strategies.

3.3 Sample study: multicategory Android mobile apps

We limited the research scope to publicly accessible Android mobile apps in a repository, focusing on **trailing and sport activity** related apps due to: (1) feature heterogeneity (e.g., geolocation, biomonitoring, social networking); (2) integrated use with common, daily-use mobile apps (e.g., instant messaging, task management, note-taking); and (3) increased popularity post COVID-19 [14]. This specification was formalized through the selection of a set of *representative apps* from 10 domain-related application categories based on their features. For each category, we used one *representative feature* as a trigger keyword to search for a *representative app* for that feature. We used Google Play as the leader app store worldwide [3] to retrieve the most popular apps for each representative feature. Finally, we manually inspected the top 10 apps retrieved by each search, and based on their suitability with the representative feature, we selected one representative app for each of these categories.

4 Knowledge graph completion

Data collection → To support the KG completion and data source identification, we conducted a literature review focused on grey literature [11], including technical reports and specialized, mobile-oriented publishers. As a result, representatives of three types of mobile app data sources were covered, including: **app store programs** (i.e., Google Play); **sideloading repositories** (i.e., F-Droid[4]); and **app search engines** (i.e., AlternativeTo[5]). For each type, we have developed two data collection operations. The **query** operation uses a set of domain-related keywords as input to search for related apps based on keyword search algorithms from data sources. The resulted set is used as the input for the **scan** operation, which collects all available data items (i.e., *metadata* and *documents*)

[4] https://f-droid.org/es/
[5] https://alternativeto.net/

for each app. We considered 4 document types categorized into 2 categories: (*i*) *Proprietary Documents* (i.e., *summaries*, *descriptions* and *changelogs*), defined by developers and platform vendors; (*ii*) *User Documents* (i.e., *reviews*), defined by users. We used web scraping, an essential tool for web-based knowledge design [12], to collect data from static and dynamic web sources like F-Droid and AlternativeTo, where alternative access methods are unavailable. Whenever possible, we utilized API consumption mechanisms, including non-official REST APIs that wrap the web scraping process, enabling direct HTTP-based access.

Data normalization → We followed the semantic schema definition process for knowledge graph design as described in [8]. We decided to use an RDF semantic schema [2], being a directed edge-labelled graph schema focused on the standardization and interoperability of web-based bodies of knowledges [8], which are some of the main purposes of the resulted knowledge base. For the data schema definition, we used existing schemas from Schema.org[6] as the most popular public schema repository for RDF graphs.

Data and schema integration → For schema integration [28], we combined key matching techniques and manual inspection of app repository schemas, ensuring conformance with Schema.org's RDF data schemas. For data integration, entity recognition relied on key matching using app *name* and *package* properties. Strategies for data inconsistencies prioritized completeness and correctness, including app repository prioritization based on data quality and document merging for increased natural language content availability.

5 Knowledge generation

5.1 KG definitions

We define a KG as $G = (V, E) = (M \cup D \cup F, E_{md} \cup E_{mf})$. Nodes include mobile apps (M), documents (D), and user annotated features (F), where $D = PD \cup UD$, including proprietary documents (PD) and user documents (UD). Edges are represented by E_{md}, linking apps to their related documents, and E_{mf}, linking apps to user annotated features. The extended KG, $G' = (V \cup F', E \cup E_{df} \cup E_{ff})$, includes the set of extracted features F' from documents, the links E_{df} between documents and their respective extracted features, and explicit semantic similarity between features (f_i, f_j) represented by E_{ff}. Deductive techniques process D using NLP to (1) extend the set of features (F') and (2) link mobile apps to their features through transitivity based on the documents these features were extracted from (E_{df}). These transformations are **batch processes**, as they are expected to be executed only once for a given G. Inductive techniques use the extended G' to resolve the knowledge requests depicted in Section 3.1. Hence, these are **on-demand processes** given that, for a given G', they are expected to be executed once for each user request. Figure 2 illustrates a partial representation of G' focused on the Strava app and a subset of its annotated features, extracted features and related apps linked to Strava through direct or indirect connections via each feature f_i.

[6] https://schema.org/

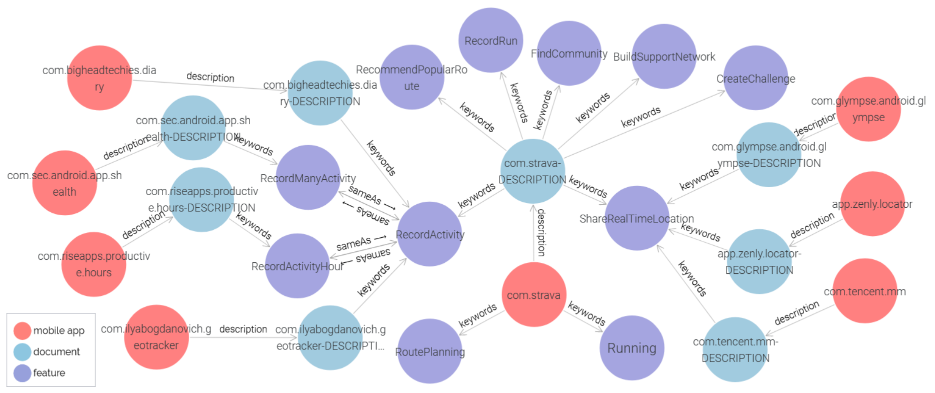

com.bigheadtechies.diary
description
com.bigheadtechies.diary-DESCRIPTION
RecommendPopularRoute
RecordRun
FindCommunity
BuildSupportNetwork
CreateChallenge
com.glympse.android.glympse
com.sec.android.app.shealth-DESCRIPTION
com.sec.android.app.shealth
com.riseapps.productive.hours-DESCRIPTION
com.riseapps.productive.hours
RecordManyActivity
RecordActivity
RecordActivityHour
sameAs
com.strava-DESCRIPTION
com.glympse.android.glympse-DESCRIPTION
ShareRealTimeLocation
app.zenly.locator
app.zenly.locator-DESCRIPTION
com.strava
com.tencent.mm
com.tencent.mm-DESCRIPTION
com.ilyabogdanovich.geotracker
com.ilyabogdanovich.geotracker-DESCRIPTI...
RoutePlanning
Running
keywords
description
● mobile app
● document
● feature

Fig. 2: Partial representation of G' focused on Strava app

5.2 Deductive knowledge: extracting features

ExtractFeatures $\rightarrow$ Each document $d_i \in D$ is processed through a NLP-based feature extraction process in the form of a pre-trained RoBERTa-based transformer pipeline built for this purpose [6]. This pipeline utilizes linguistic annotations and PoS-based expression patterns to identify natural language features in the documents. As a result, for each $d_i \in D$ we extend G through $V \leftarrow V \cup F'(d_1) \cup F'(d_2) ... \cup F'(d_n)$ and $E \leftarrow E \cup E_{df}(d_1) \cup E_{df}(d_2) ... \cup E_{df}(d_n)$, where $F'(d_i) \subset F'$ includes all features extracted from d_i (e.g., *RecordActivity*) and $E_{df}(d_i) \subset E_{df}$ includes all links between those features and d_i.

LinkSimilarFeatures $\rightarrow$ Each feature $f_i \in F \cup F'$ is processed through a semantic similarity pipeline to compute a similarity score $sim(f_i, f_j)$ for all $(f_i, f_j) \in F \cup F'$ using a TF-IDF-based search index built upon the database management system for scalability and applicability in large repositories [21]. For all $sim(f_i, f_j) >= thr$, where thr is a domain-dependent similarity score threshold, we extend G through $E \leftarrow E \cup E_{ff}(f_1) \cup E_{ff}(f_2) ... \cup E_{ff}(f_n)$. $E_{ff}(f_i)$ models the link between all (f_i, f_j) for which $sim(f_i, f_j) >= thr$ (e.g., *RecordActivity* is semantically linked to *RecordManyActivities* and *RecordActivityHours*).

5.3 Inductive knowledge: finding apps

FindAppsByFeature $\rightarrow$ Two feature-to-app similarity methods were designed and compared. The **index-based** method uses a term-to-document similarity index for each PD type (i.e., *summary*, *description*, *changelog*). For a given f_i (e.g., *ShareRealTimeLocation)*, the index returns a ranked list of $m_j \in M$ (e.g., *app.zenly.locator*) sorted by $sim(f_i, m_j)$ in descending order. This approach is parameterized to allow similarity evaluations filtered by a given document type (e.g., *descriptions*), or to compute an aggregated similarity score based on all available $E_{md}(m_j)$ from $d_k \in PD$. The **graph-based** method relies on a Python adaptation [23] of the SimRank* algorithm [30], designed for scalable similarity search, which has proven to be an effective approach for its adoption in large-scale graphs. For a given f_i, the graph-based method computes a similarity score based on its shared paths with respect to $E_{df}(d_i) \cup E_{ff}(f_j)$ for all $d_i \in D$. This method is parameterized to allow similarity based on $d_i \in PD$ (i.e., without reviews), or based on $d_j \in D$ (i.e., with reviews).

FindSimilarApps $\rightarrow$ Given a mobile app $m \in M$ (e.g., *com.strava*), we have applied the same algorithm implementation approach as in *FindAppsByFeature*. The **index-based** method relies on a document-to-document index for each PD type. For a given m_i, the index returns a ranked list of m_j sorted by $sim(m_i, m_j)$ in descending order. The index-based approach is parameterized for a given document type or for aggregated results as in *FindAppsByFeature*. On the other hand, the **graph-based** (adapted from SimRank* algorithm) computes for a given m_i a similarity score based on its shared paths with respect to $E_{md}(m_j) \cup E_{df}(d_k) \cup E_{ff}(f_l)$ for all $m_j \in M$, parameterized by document type.

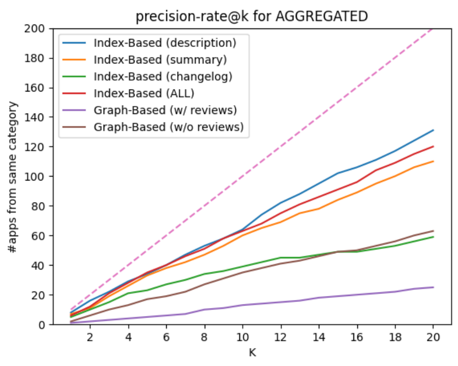
6 Evaluation

We focus the evaluation on the *functional correctness* and *performance efficiency* [9] of the inductive knowledge requests. The evaluation plan is defined as follows:

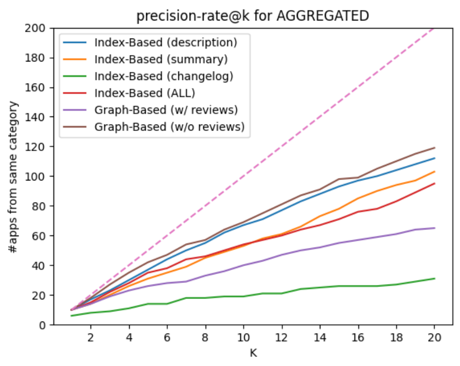
1. Run *ExtractNLFeatures* for all $d_i \in PD$.
2. Create/update the graph database indexes with the extended G′.
3. Run *LinkFeatures* using all thresholds (thr) within [0.1, 0.2, ..., 0.9].
4. For each thr, run *FindAppsByFeature* using the *representative features*.
5. For each thr, run *FindSimilarApps* using the *representative apps*.
6. Compute the precision[7] rate@k ($k <= 20$) for both inductive algorithms.
7. Run *ExtractNLFeatures* for all $d_i \in UD$ and repeat 2$\rightarrow$6.

Functional correctness $\rightarrow$ Figure 3 reports precision rate@k for both inductive algorithms obtained with the optimal value for the similarity threshold score in Step 3 (i.e., $thr = 0.7$). At each k value, the plots show how many recommended apps found by each algorithm matched the category of the input item, which we considered as ground-truth for evaluation (i.e., *representative feature* for *FindAppsByFeature*, and *representative app* for *FindSimilarApps*). Both figures report the aggregated results for all categories. For *FindAppsByFeature*, index-based algorithms show similar precision using description, summary, or aggregated document types, while the graph-based version performs poorly. However, for *FindSimilarApps*, the graph-based version (without reviews) is the best algorithm. Including reviews in the feature extraction algorithm worsens the quality of the prediction. For the best algorithm version of each inductive technique (i.e., index-based with description for *FindAppsByFeature*, graph-based without reviews for *FindSimilarApps*), both algorithms report a precision rate $>= 80\%$ at the top positions of the returned ranked list (i.e., $k <= 3$). At $k = 5$ *FindSimilarApps* still reports a precision rate of 80%, while *FindAppsByFeature* reports 72% of matched suggestions.

[7] We focus on precision to maximize recommendation quality at the top positions



(a) *FindAppsByFeature* (aggregated)



(b) *FindSimilarApps* (aggregated)

Fig. 3: Precision rate@k (k=20) for inductive knowledge algorithms

Performance efficiency → Table 1 reports the average execution time for both deductive (i.e., *ExtractNLFeatures*, *LinkFeatures*) and inductive (i.e., *FindAppsByFeature*, *FindSimilarApps*) algorithms. Batch processes (i.e., deductive algorithms) for the whole sample data set are particularly time-consuming, especially when processing user reviews and their extracted features (up to 44 hours including *ExtractNLFeatures* and *LinkFeatures*). However, updates on the KG instance (i.e., adding a new app) only require a few seconds for *ExtractNLFeatures* and less than 1 second for *LinkFeatures* on average. On the other hand, on-demand processes (i.e., inductive algorithms) take on average less than 1 second, with the exception of *FindSimilarApps* when processing all reviews.

T(s)		**w/o reviews**		**w/ reviews**	
		/item	**total**	**/item**	**total**
ExtractNLFeatures		2.48 s	1570 s (˜0.5 h)	105.9 s	67,690 s (˜19 h)
LinkFeatures		0.29 s	3362 s (˜1 h)	0.68 s	88,753 s (˜25 h)
FindAppsByFeature	Index-Based	0,11 s	1.06 s	-	-
	Graph-Based	0.39 s	3.91 s	0.69 s	6.86 s
FindSimilarApps	Index-Based	0.09 s	0.87 s	-	-
	Graph-Based	0.41 s	4.08 s	1.21 s	12.24 s

Table 1: Average execution time for inductive and deductive algorithms

7 Discussion

Evaluation results show that the designed knowledge requests are effective for on-demand knowledge retrieval, with high prediction precision at the top positions. In the context of integrating the designed KB to support LLM-based dialogue systems, prediction quality for top k values (i.e., $k >= 5$) is especially relevant for reporting simplified snapshots from each knowledge request response [5]. The graph-based similarity measure performs well for comparing nodes with shared semantics (i.e., app-app in *FindSimilarApps*). while the term-to-document similarity measure is better for comparing different node types (i.e., app-feature in *FindAppsByFeature*). Finally, the use of reviews did not improve the quality of the results, while it also added a great overhead in batch process execution.

Regarding validity threats, we primarily focus on construct and internal validity. Construct validity concerns the KG construction, where potential biases may arise from the selection of representative features and app repositories. Nevertheless, we argue that the criteria for selecting these representatives and sources (i.e., popularity, availability of data, quality of data) minimizes the risk of biased coverage and low completeness. Concerning knowledge generation, the extended knowledge process relies on the quality of the feature extraction process, whose design and development was recently published as a NLP-based feature extraction tool [6]. Finally, we considered as ground-truth for inductive techniques the application categories based on the original app query data collection method, relying on the accuracy of the app repositories search algorithms, for which we argue that the criteria for selecting those repositories (i.e., popularity and relevance) reduces the impact of this potential threat.

8 Conclusions

This research aims at providing a holistic perspective on the end-to-end process of building a KB of mobile apps in the form of a KG. The technical infrastructure (i.e., software components), the resulting resources (i.e., data artefacts) and the findings from this research are also intended to lay the groundwork for the integration of additional knowledge requests, either beyond the object of analysis (i.e., NL documents), the descriptor to which similarity is based on (i.e., features), or the knowledge requests formalization (i.e., feature vs. app and app vs. app similarity). Moreover, evaluation results prove that on-demand inductive knowledge can be effectively and efficiently integrated for real-time consumption use cases. We envisage focusing future research efforts on (1) formalization of the concept of feature, and (2) conducting a user-study evaluation.

Acknowledgments
With the support from the Secretariat for Universities and Research of the Ministry of Business and Knowledge of the Government of Catalonia and the European Social Fund. This paper has been funded by the Spanish Ministerio de Ciencia e Innovación under project / funding scheme PID2020-117191RB-I00 / AEI/10.13039/501100011033.

References

1. 42matters AG: Google Play Store App Update Frequency Statistics (2023), https://42matters.com/google-play-aso-with-app-update-frequency-statistics
2. Brickley, D., Guha, R.V.: (2014), https://www.w3.org/TR/2014/REC-rdf-schema-20140225/
3. Ceci, L.: Biggest app stores in the world 2022 (Aug 2022), https://www.statista.com/statistics/276623/, Accessed 22 November 2022

4. Ceci, L.: Number of monthly android app releases worldwide 2023 (Mar 2023), https://www.statista.com/statistics/1020956/android-app-releases-worldwide/
5. Chen, N., et al.: Mobile app tagging. In: Proceedings of the 9th WSDM (2016)
6. Gallego, A., et al.: TransFeatEx: a NLP pipeline for feature extraction. In: REFSQ 2023, CEUR Workshop Proceedings (2023)
7. Grua, E.M., et al.: Self-adaptation in mobile apps: a systematic literature study. In: 2019 IEEE/ACM 14th SEAMS (2019)
8. Hogan, A., et al.: Knowledge graphs. ACM Comput. Surv. (2021)
9. ISO/IEC: System and software quality models ISO/IEC 25010 (2011)
10. Johann, T., et al.: SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews. In: 25th International RE Conference (2017)
11. Kamei, F., et al.: Grey literature in software engineering: A critical review. Information and Software Technology (2021)
12. Khvatova, T., Dushina, S.: Scientific online communication: The strategic landscape of researchgate users. IJTHI (2021)
13. Kumari, S., Memon, Z.A.: Extracting feature requests from online reviews of travel industry. Acta Scientiarum - Technology **44** (2022)
14. Kwon, J.Y., et al.: Analysis of strategies to increase user retention of fitness mobile apps during and after the covid-19 pandemic. IJERPH (2022)
15. Lin, J., et al.: New and improved: Modeling versions to improve app recommendation. In: Proceedings of the 37th International ACM SIGIR (2014)
16. Motger, Q., Franch, X., Marco, J.: Software-Based Dialogue Systems: Survey, Taxonomy and Challenges. ACM Comput. Surv. (2022)
17. Motger, Q., et al.: Integrating adaptive mechanisms into mobile applications exploiting user feedback. In: Research Challenges in Information Science (2021)
18. OpenAI: Gpt-4 technical report (2023)
19. Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.M., Rothchild, D., So, D., Texier, M., Dean, J.: Carbon emissions and large neural network training (2021)
20. Petrik, D., Schönhofen, F., Herzwurm, G.: Understanding the design of app stores in the iiot. In: IEEE/ACM IWSiB. pp. 43–50 (2022)
21. Raatikainen, M., et al.: Improved management of issue dependencies in issue trackers of large collaborative projects. IEEE TSE (2022)
22. RDF Working Group: Resource Description Framework (RDF), https://www.w3.org/RDF/, Accessed 22 November 2022
23. Reyhani, M., et al.: Effectiveness and efficiency of embedding methods in task of similarity computation of nodes in graphs. Applied Sciences (2021)
24. Rožanec, J.M., et al.: XAI-KG: Knowledge Graph to Support XAI and Decision-Making in Manufacturing. Lecture Notes in Business Information Processing (2021)
25. Schlichtkrull, M., et al.: Modeling relational data with graph convolutional networks. Lecture Notes in Computer Science (2018)
26. Shen, S., et al.: Towards release strategy optimization for apps in google play. In: Proceedings of the 9th Asia-Pacific Symposium on Internetware (2017)
27. Stol, K.J., Fitzgerald, B.: The abc of software engineering research. ACM Trans. Softw. Eng. Methodol. (2018)
28. Wang, L.: Heterogeneous Data and Big Data Analytics. Automatic Control and Information Sciences **3**(1), 8–15 (2017)
29. Xu, P., et al.: MEGATRON-CNTRL: Controllable story generation with external knowledge using large-scale language models. In: EMNLP 2020 (2020)
30. Yu, W., et al.: Simrank*: Effective and scalable pairwise similarity search based on graph topology. The VLDB Journal (2019)
31. Zhao, Y., et al.: A supervised learning community detection method based on attachment graph model. Lecture Notes in Computer Science (2022)