Automated Software Engineering (2023) 30:13
https://doi.org/10.1007/s10515-023-00377-x

Data-driven prototyping via natural-language-based GUI retrieval

Kristian Kolthoff[1] · **Christian Bartelt**[1] · **Simone Paolo Ponzetto**[2]

Received: 25 March 2022 / Accepted: 28 January 2023 / Published online: 14 March 2023

Abstract

Rapid GUI prototyping has evolved into a widely applied technique in early stages of software development to facilitate the clarification and refinement of requirements. Especially high-fidelity GUI prototyping has shown to enable productive discussions with customers and mitigate potential misunderstandings, however, the benefits of applying high-fidelity GUI prototypes are accompanied by the disadvantage of being expensive and time-consuming in development and requiring experience to create. In this work, we show *RaWi*, a data-driven GUI prototyping approach that effectively retrieves GUIs for reuse from a large-scale semi-automatically created GUI repository for mobile apps on the basis of Natural Language (NL) searches to facilitate GUI prototyping and improve its productivity by leveraging the vast GUI prototyping knowledge embodied in the repository. Retrieved GUIs can directly be reused and adapted in the graphical editor of *RaWi*. Moreover, we present a comprehensive evaluation methodology to enable (i) the systematic evaluation of NL-based GUI ranking methods through a novel high-quality gold standard and conduct an in-depth evaluation of traditional IR and state-of-the-art BERT-based models for GUI ranking, and (ii) the assessment of GUI prototyping productivity accompanied by an extensive user study in a practical GUI prototyping environment.

Keywords Rapid prototyping of graphical user interfaces (GUIs) · Information retrieval for GUIs · GUI prototypes from natural language requirements · Data-driven GUI prototyping · Deriving editable GUI screens

✉ Kristian Kolthoff
kolthoff@es.uni-mannheim.de

Christian Bartelt
bartelt@es.uni-mannheim.de

Simone Paolo Ponzetto
simone@informatik.uni-mannheim.de

[1] Institute for Enterprise Systems, University of Mannheim, Mannheim, Germany

[2] Data and Web Science Group, University of Mannheim, Mannheim, Germany

1 Introduction

GUI prototyping represents an important technique to visualize the analysts' understanding of the NL requirements from customers, enables their verification by the customer as a tangible artifact, provides the foundation for incorporating the customer early into the application development and leads to fruitful discussions, clarification and refinement of requirements (Mukasa and Kaindl 2008; Ravid and Berry 2000; Rudd et al. 1996; Windsor and Storrs 1992; Beaudouin-Lafon and Mackay 2002). Especially high-fidelity GUI prototypes have been proven to be effective, since these prototypes provide the foundation for discussions of higher quality between customers and analysts, and more detailed feedback can be obtained during user tests compared to low-fidelity approaches (Rudd et al. 1996; Coyette et al. 2007; Landay and Myers 1994). However, the benefits of high-fidelity GUI prototypes are accompanied by the disadvantages of increased time and experience required for their development (Rudd et al. 1996). Thus, the applicability for interactive GUI prototyping in the requirements elicitation phase with customers is restricted, since only limited time is available in such a scenario. In general, by reusing GUI prototypes, the required prototyping effort can be reduced (Suleri et al. 2020), however, the search, retrieval and application of a vast amount of such reusable GUIs for prototyping still remains expensive and difficult without adequate support, particularly for novice and inexperienced analysts. Therefore, fast translation of the NL requirements (NLR) from the customer into GUI prototypes with the respective functionality has the potential of positively impacting the quality and efficiency of the overall process. Consequently, in this paper we exploit advances in NLP to investigate NL-based GUI retrieval from a large-scale GUI repository for GUI reuse, and make first steps towards quantifying the potential to tackle this problem and to provide automatic assistance to analysts for interactive GUI prototyping with customers.

To facilitate and simplify GUI prototyping, a plethora of approaches has been proposed before. First, many popular GUI prototyping tools are available and employed in practical prototyping environments such as *Sketch* (Sketch 2019), *Adobe XD* (Adobe XD 2015), *Mockplus* (Mockplus 2014), *Figma* (Figma 2016) and *Balsamiq* (Faranello 2012). These approaches typically allow users to combine basic GUI components and a small amount of GUI templates, however, are time-consuming and demand a wide prototyping experience for effective application. Second, recent GUI retrieval approaches such as *Swire* (Huang et al. 2019), *GUIFetch* (Behrang et al. 2018), *Screen2Vec* (Li et al. 2021) and *VINS* (Bunian et al. 2021) use sketches or GUI screenshots as input for retrieving GUIs to support prototyping, however, initially require a basic GUI prototype and the retrieved GUI images cannot be directly reused or adapted. Overall, these approaches focus on supporting the prototyping of the final GUI design by showing GUI design alternatives, however, cannot be applied for interactive GUI prototyping for requirements elicitation. Third, *Guigle* (Bernal-Cárdenas et al. 2019) represents the first basic search engine for GUIs of mobile apps supporting a query language and stylistic searches, however, proposes only a simple

retrieval approach and found GUI screenshots are not directly reusable. Finally, recent GUI prototyping assistance approaches such as *GUIComp* (Lee et al. 2020) attempt to simplify prototyping by showing GUIs similar to the design created by the user and providing GUI complexity characteristics. Thus, *GUIComp* can only effectively support users after initial GUI design creation, recommended GUIs are not directly reusable and the focus lies on supporting the creation of high-quality GUI designs. Therefore, these approaches cannot be employed to assist analysts during the requirements elicitation phase via interactive GUI prototyping.

NL-based GUI retrieval for GUI prototyping In this work, we introduce *RaWi*, a data-driven rapid GUI prototyping approach to create high-fidelity GUI prototypes that *(i)* exploits a large-scale semi-automatically created GUI repository of mobile applications by employing NL-based ad-hoc GUI retrieval and automatically derives editable GUI screens for reuse and *(ii)* thus effectively leverages the GUI prototyping knowledge embodied in the repository to facilitate prototyping and improve prototyping productivity. Due to enabling fast translation of NL fragments from customers to GUI prototypes, our approach allows for significantly faster GUI prototyping compared to a traditional approach and thus can be applied more effectively by analysts to elicit requirements with customers via interactive GUI prototyping. In particular, we envision to support novice analysts with little or no prior experience in GUI prototyping with our approach by providing easy access to plenty of reusable GUIs and domain knowledge embodied in the GUI repository. Due to the automatic derivation of partly editable GUI screens from a large-scale GUI screenshot repository, our approach is able to provide a vast number of reusable GUI screens, which eliminates the manual effort typically required for providing GUI templates in traditional prototyping approaches. *RaWi* integrates both the GUI retrieval mechanism and editable GUI screen derivation techniques in a web-based graphical prototyping editor. Our approach builds upon recent work from Kolthoff et al. (2020, 2021) and substantially extends it in terms of a comprehensive evaluation methodology, novel experiments, datasets, state-of-the-art retrieval techniques and prototype.

Contributions With our work we make the following research contributions:

- We present a BERT-based Learning-To-Rank (LTR) approach that is trained on GUI relevance data harvested using crowdsourcing techniques, which significantly outperforms all traditional ranking methods.
- We create the first and comprehensive gold standard for NL-based GUI retrieval with crowdsourcing techniques to enable a systematic evaluation of NL-based GUI ranking models and make it publicly available to foster further research.
- We conduct the first in-depth analysis and evaluation of various traditional Information Retrieval (IR), Automatic Query Expansion (AQE) and trained BERT-based Learning-To-Rank models for NL-based GUI ranking using the newly proposed gold standard.
- We propose a comprehensive and general evaluation methodology including multiple metrics for measuring GUI prototyping productivity and conduct an extensive user study to assess the usefulness of the GUI prototyping approach in a practical rapid prototyping environment.

Comparison with our own previous work Our prior work (Kolthoff et al. 2020, 2021) focused mainly on the investigation of traditional IR methods for GUI retrieval, thereby neglecting recent advances in NLP (Young et al. 2018; Devlin et al. 2018), and lacks an extensive evaluation methodology for *(i)* properly and systematically assessing the performance of NL-based GUI ranking systems and for *(ii)* assessing the GUI prototyping productivity. Therefore, we highlight the importance of this paper to fill this gap by investigating state-of-the-art GUI retrieval methods and proposing a comprehensive evaluation methodology. Our interactive prototype, source code and datasets (including the newly created gold standard) are all freely available (RaWi Prototype 2022; RaWi Repository 2022) to foster future work along these research lines.

Focus on the requirements elicitation process GUI prototyping basically consists of two broad activities: *(i)* eliciting the required functionality of the GUIs where the requirements are formulated typically in NL by the customer and *(ii)* creating a final, professional GUI design subsequently. Our approach targets to provide support for the first step: the requirements elicitation. As stated before, previous research showed that by employing high-fidelity GUIs for elicitation, the feedback quality and detail is improved (Rudd et al. 1996; Coyette et al. 2007; Landay and Myers 1994). In an interactive elicitation with customers, the available time for GUI prototyping is typically very limited and requires fast translation of the NL fragments from the customer into GUI prototypes with the respective functionality as shown in Fig. 1. Here our approach provides automatic assistance especially for inexperienced analysts. Accordingly, in our conducted experiments we restricted the available time for GUI prototyping and quantified productivity in terms of selecting the necessary GUI components (cf. Sect. 3.2.3). Similarly, our process of creating a gold standard for GUI retrieval using crowdsourcing techniques is meant to capture the scenario of customers with no particular experience in GUI prototyping providing NL queries, since users of crowdsourcing platforms have much diverse backgrounds.

Structure of the paper The remainder of this article is structured as follows. In Sect. 2 we provide a summary of our GUI retrieval and GUI prototyping approach *RaWi*. Section 3 presents the details on the experimental setting used for evaluation, whose results are presented in Sect. 4. Threats to validity and limitations of this work are presented in Sects. 5 and 6, respectively. In Sect. 7 we provide an overview of related work. Section 8 covers the concluding remarks and presents multiple lines of research for the future.

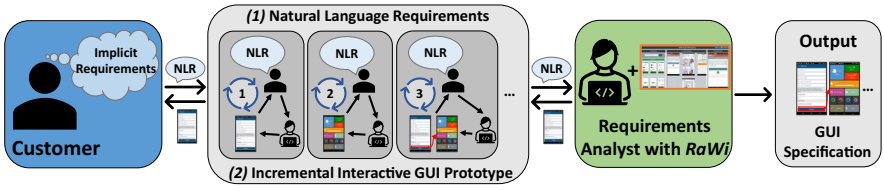


Fig. 1 Requirements elicitation via interactive GUI prototyping assisted by our approach *RaWi* to rapidly translate NL requirements into GUI prototypes

2 Approach: RaWi

The main goal of our approach is to *(i)* enable fast GUI retrieval from a large-scale GUI repository via NL-based search queries to leverage the embodied GUI prototyping knowledge and *(ii)* to allow users to rapidly build prototypes using automatically derived editable GUI screens. Figure 2 shows an overview of the architecture of *RaWi* separated into four components. First, *(A)* we employ the large-scale semi-automatically created GUI repository *Rico* (Deka et al. 2017) for mobile applications as the basis for GUI retrieval. This dataset encompasses a large number of Android applications that are crawled from Google Play. To improve the quality of the GUI repository, we initially cleanse the GUIs based on multiple criteria. Afterwards, we create textual representations of the GUIs by extracting different text segments from the GUI hierarchy data. Second, *(B)* we apply an identical pipeline of text preprocessing techniques to both the textual GUI representations and the NL search queries. The retrieval architecture employed in *RaWi* follows the extended Boolean model (Manning et al. 2008). To this end, we compute an index over the GUI text representations and match the NL query against it to obtain initial matches. Subsequently, we compute a ranking over the GUI matches using popular IR and state-of-the-art BERT-based Learning-To-Rank models. In addition, we experimented with multiple automatic query expansion (AQE) techniques. Third, *(C)* we automatically create partly editable GUI screens by exploiting the GUI screenshots and GUI hierarchy data and extract basic style properties of some GUI components. Finally, *(D)* we provide a web-based implementation of our approach that integrates

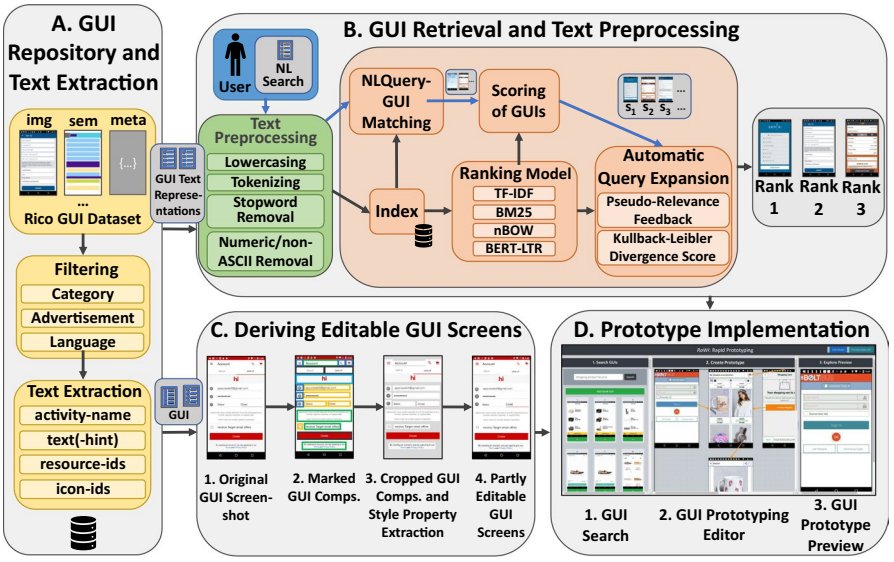


Fig. 2 Overview of *RaWi* with (*A*) the large-scale GUI repository *Rico* and GUI text extraction, (*B*) GUI retrieval and text preprocessing pipeline, (*C*) automatic GUI screen derivation and (*D*) the rapid GUI prototyping editor

all of the previously discussed components. In the following, we describe the individual components of our approach in detail.

2.1 GUI repository and text extraction

To retrieve relevant GUIs for NL-based search queries, we first require a suitable GUI repository. Recent data-driven design research constructed and released several GUI datasets appropriate for our approach by automatically crawling a vast amount of Android applications from Google Play such as *ReDraw* (Moran et al. 2018), *ERICA* (Deka et al. 2016), *Rico* (Deka et al. 2017) and *Enrico* (Leiva et al. 2020). *ReDraw* collects GUIs by automatically exploring 5416 Android applications comprising a total of 14,382 unique GUI screenshots and GUI hierarchy data using GUI testing automation techniques to simulate user input. Screenshots of the GUIs are captured either with third-party frameworks or platform-dependent utilities. In contrast, *ERICA* solely relies on manual human-based GUI exploration and is primarily developed for capturing user interaction traces. *ERICA* comprises around 18,600 unique GUIs harvested from 2400 applications. *Rico*, an extension of the *ERICA* dataset, mines GUI screenshots, GUI hierarchy data, application meta data and interaction traces with both human-based and automatic exploration techniques and constitutes the largest design dataset of the discussed ones with 72,219 GUIs collected from 9772 unique Android applications (from 27 different application categories). *Rico* employs a custom Android service to access detailed information about all GUI components such as text, absolute position on the screen, visibility indicators, resource identifiers and the activity name, among others. For our prototyping approach, we employ *Rico* based on multiple important reasons: *(i)* the large amount of GUIs available for retrieval, *(ii)* the coverage of many diverse application domains and *(iii)* the comprehensive textual information provided particularly valuable for NL-based GUI retrieval.

2.1.1 GUI filtering

Due to the partly automated extraction of GUIs in *Rico*, the GUI repository naturally contains erroneous GUIs inapplicable for our approach. We recognized multiple GUI types to be excluded through manual inspection of the dataset. First, *(1)* GUIs of the entertainment category (e.g. gaming) are discarded since *RaWi* should specifically support rapid prototyping of business and utility applications. Second, *(2)* GUIs displaying advertisement overlays and full screen web views are identified heuristically by particular patterns of component labels and discarded from the repository (based on the presence of a specific number of components and component types e.g., only web view, web view and icons, etc.). By employing this filtering heuristic, a common hierarchy-GUI-screenshot mismatch error type, in which the GUI screenshot appears to be a normal GUI with many low-level GUI components, however, the Rico view hierarchy data represents the entire GUI only as a single full screen web view, is identified and removed. This mismatch occurs often since e.g., developers directly embed their web apps into

Android apps instead of developing a native application. Third, *(3)* we discard non-English GUIs identified through a language detection framework that computes language probabilities by accumulating character-level *n*-gram spelling feature probabilities (Shuyo 2010). In general, other languages could be supported by adapting the retrieval models (e.g. by employing multilingual embeddings or simply translating the corpus). After applying filtering, the GUI repository encompasses 57,764 unique GUIs. Furthermore, research identified several other error types encompassed in the Rico GUI dataset (Leiva et al. 2020). Many of these errors are concerned with component-level mismatches (e.g., vertical offset, missing background image, wrong component type), which are not affecting our retrieval methods. However, the Rico GUI dataset also rarely contains completely mismatched hierarchy-GUI-screenshot pairs, where the GUI screenshot shows completely different functionality compared to the available view hierarchy. These cases can potentially create false positives during the GUI retrieval, however, they cannot be identified easily. Available research provides no evaluation of the extent of their occurrences in Rico and based on our manual inspection, the occurrence of these cases appears to be infrequent. Moreover, the potential negative impact of erroneous GUIs on the retrieval performance is mitigated by the vast amount of available GUIs in the Rico GUI dataset.

2.1.2 GUI text extraction

To support GUI retrieval from NL-based search queries over the GUI repository, we construct a textual representation for each GUI by extracting several text segments through XPath expressions from the GUI hierarchy. First, we extract displayed text and text hints that are marked as being visible on the captured GUI screenshot. Based on initial experiments, we decided to neglect the extraction of text from components marked as non-visible on the captured GUI screenshot since these text sections often tend to mislead the retrieval models. Similarly, we exploit the full activity name of the GUI and the resource identifier of each GUI component since developers often provide semantically descriptive naming especially valuable for GUI retrieval. However, these strings (for example "*com.sample.sens.register.CreateNewAccountActivity*") require additional preprocessing through a tokenization pipeline to make them fully searchable. To this end, we apply punctuation, camel and snake case tokenization and finally use a probabilistic tokenizer based on English Wikipedia unigram frequencies to extract tokens from the remaining parts. To clean the created tokens from non-descriptive and general tokens (for example "*com*", "*main*" and "*activity*"), we employ a custom domain-specific stopword list for filtering, containing the top most frequent words in the corpus. Furthermore, we exploit the semantic labels of icons provided by *Rico* due to their similarly meaningful semantic descriptions. Figure 3 shows an example GUI retrieved with *RaWi* and multiple text segments that are extracted for retrieval, including the activity name, the icon labels and the resource identifiers. By extracting these dimensions of text, our approach supports both the search for entire GUIs and the retrieval of GUIs that only contain parts relevant to the NL search query.

Fig. 3 Multiple GUI text segments (activity name, icon and resource identifiers) that are extracted and preprocessed for NL-based GUI retrieval

2.2 GUI retrieval and text preprocessing

For text preprocessing of both the GUI text representations and the NL input query, we apply a standard preprocessing pipeline. First, we lowercase the text and apply tokenization. Tokens are then excluded by the following multiple filters. We discard stopwords and words comprising numeric or non-ASCII characters. Subsequently, we describe the employed baseline ranking models, the AQE techniques and the semantic BERT-based Learning-To-Rank models.

2.2.1 Baseline ranking models

For enabling retrieval of relevant GUIs with NL-based search queries from our GUI repository, we adopt well-known ranking models that showed their effectiveness in various domains before and are established in general-purpose search engines (Manning et al. 2008). Therefore, we decided to adopt the TF-IDF, BM25 (Robertson et al. 1995) and a TF-IDF-weighted neural Bag-Of-Words (nBOW) method (Galke et al. 2017) using pretrained dense word embeddings for similarity scoring to establish a first baseline for GUI ranking.

2.2.2 Automatic query expansion

For further enhancements of the retrieval performance and tackling the vocabulary mismatch problem (or synonymy) we additionally examined Automatic Query Expansion (AQE) techniques (Manning et al. 2008; Azad and Deepak 2019). Synonyms such as "*choose*" and "*select*" can not be matched by traditional IR methods such as BM25 (although the later described BERT-LTR approach mitigates this problem). AQE methods attempt to extend and refine the initial query automatically with relevant terms in order to improve the ranking performance. In the absence of user feedback, AQE methods employ pseudo-relevance feedback (PRF) where the initially retrieved top-*k* documents (from a base model such as BM25) are assumed to be relevant and thus used as

input to compute the query expansion terms. For our GUI retrieval approach, we therefore focused on the incorporation and evaluation of PRF methods. Based on the notion of PRF, many expansion term scoring techniques have been devised in AQE research before (Azad and Deepak 2019). These scoring techniques generally follow a similar procedure. By comparing the term distributions of term t in the relevant documents D_R and the entire corpus D_C, the importance of a term in the relevant documents D_R can be estimated. The initial NL query can then be expanded with the top-n terms according to their ranking score. For our GUI retrieval system, we compute the Kullback–Leibler Divergence (KLD) score (Carpineto et al. 2001) for each term $t \in D_R$ as

$$Score_{KLD}(t) = p(t \mid D_R) \cdot \log \frac{p(t \mid D_R)}{p(t \mid D_C)}$$

with $p(t \mid D_R)$ and $p(t \mid D_C)$ being the probability of term t occurring in the relevant documents D_R and in the entire document collection D_C, respectively. To compute these probabilities we use the maximum likelihood estimates (MLE) i.e. $p(t \mid D_X) = \frac{f_{t,x}}{\sum_{d \in D_x} |d|}$, where d is a document of D_x with $\mid d \mid$ tokens. Applying and evaluating the KLD score in our AQE experiments is based on the notion that the KLD score showed its effectiveness compared to other expansion term scoring methods before (Carpineto et al. 2001). In addition to using this method for expansion term selection solely, we evaluated a second variant that includes the KLD score as a weight for the expanded terms in the retrieval model in order to control their effect on the GUI document ranking. Since the GUI prototypes are represented through multiple text segments, we adapt the PRF-KLD score to compute expansion candidates for each GUI text segment individually and aggregate the top-n terms of the different text segments to obtain the query expansion terms. If duplicate top-n terms from different text segments appear, we only incorporate them once in the expanded query. Similarly to the previously discussed method, we additionally evaluated a second variant of the text segment-wise PRF-KLD by weighting the expansion terms with their respective scores to control their influence on the ranking. Figure 4 exemplifies the employed AQE technique as *(1)* retrieving the initial top-k

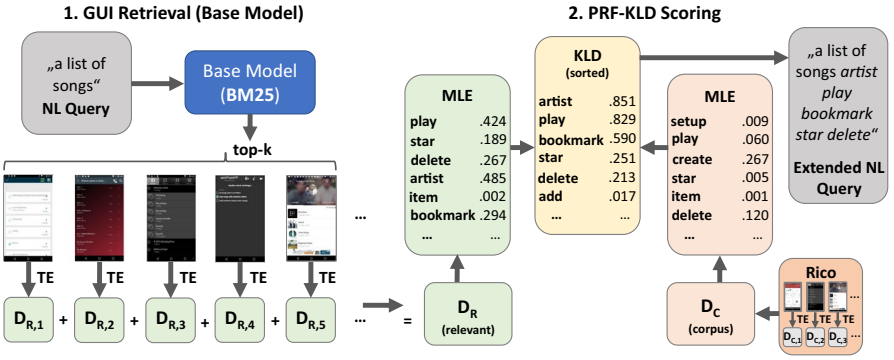


Fig. 4 Example of AQE using the KLD scoring: (*1*) Initial top-k GUI retrieval with a base ranking model and (*2*) calculating KLD scores based on the MLEs

GUIs and extracting the text (TE) to obtain D_R, then *(2)* computing the MLEs for D_R and D_C to obtain the KLD scores.

2.2.3 BERT-based learning-to-rank models

The models presented previously can be applied to our GUI text documents, however, GUI ranking differs from other text-based ranking tasks in multiple aspects. First, GUIs are not standard well-structured text documents, but typically provide only short text fragments (e.g. button or short label text), named components (often with proprietary abbreviations), many irrelevant data items (e.g. product descriptions or names) and no natural ordering of the text. Accordingly, the gap between the NL queries and the GUI text representations is large. Therefore, more sophisticated semantic models are required to further improve the retrieval performance. To this end, we fine-tune a state-of-the-art BERT-based (Devlin et al. 2018) learning-to-rank (LTR) model (Han et al. 2020). BERT is a generally effective pretrained and large-scale language model that outperforms many traditional models in different NLP tasks. We prepare the model by concatenating the *[CLS]* token, followed by the NL query, a separator *[SEP]* token and the GUI document text and a final *[SEP]* token as the input to the BERT-LTR model. Then, the pooled *[CLS]* representation is used as input to a ranking model. We trained three BERT-LTR models including *(1)* a pointwise LTR model (with *sigmoid cross entropy loss*), *(2)* a pairwise LTR model (with *pairwise logistic loss*) and *(3)* a pointwise LTR model (with *softmax loss*) and evaluated their NL-based GUI ranking performance. We employ the BERT-LTR model proposed by (Han et al. 2020) and illustrate its application for NL-based GUI ranking in Fig. 5. The respective GUIs are first transformed to GUI text documents, concatenated with the NL query and fed into the BERT model to obtain the BERT embedding. To compute the ranking score for a GUI, the embedding is fed through a feed-forward network. On this basis, pointwise and pairwise losses can be computed to optimize the feed-forward network and fine-tune the pretrained BERT model. In addition, we included a pretrained Sentence-BERT model (Reimers and Gurevych 2019) in our evaluation, in order to better understand the

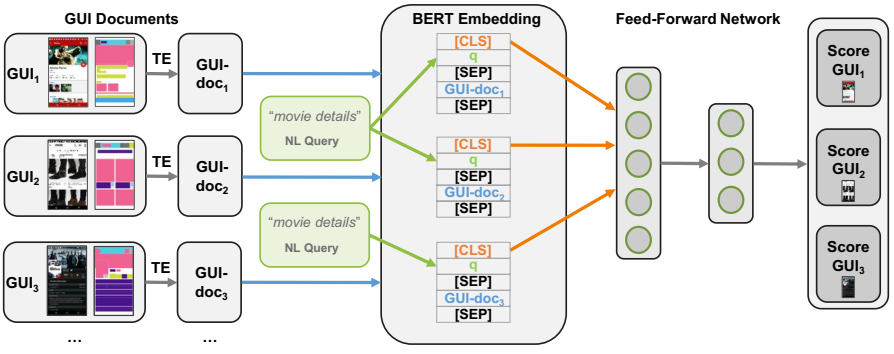


Fig. 5 BERT-LTR model based on the work of Han et al. (2020) illustrated for its application for NL-based GUI ranking using a pretrained BERT model

performance gained by the proposed BERT-LTR models, which are trained specifically for the NL-based GUI ranking task. This model has also been used as a text-only baseline in the *Screen2Vec* approach (Li et al. 2021).

2.3 Deriving editable GUI screens

Since *Rico* provides only GUI screenshots which are not directly reusable for rapid GUI prototyping and editing, we propose a simple yet effective algorithm to transform them into partly editable GUI screens. The automatic derivation of editable GUIs from the GUI repository is integrated into the graphical editor of *RaWi*, which will be described in more detail in the subsequent section. In particular, we can exploit the GUI hierarchy data and semantic labels for GUI components encompassed in *Rico* to extract them accordingly from the accompanied GUI screenshots. Therefore, *(1)* we initially obtain the original GUI screenshot provided by *Rico* and *(2)* crop each contained GUI component from the screenshot based on their absolute position. We applied this procedure for both individual GUI components and additionally identified layout components. Third, *(3)* for three GUI component types including *labels*, *buttons* and *text-input* we extract multiple style properties in our current prototype to provide more detailed editing capabilities. For the remaining components, we currently simply reuse their image crop in the editable GUI screens. For each layout group, we compute the background color as the top-ranked RGB color in their histogram. For *labels*, we identify the font color by first obtaining the background color as before and then computing a normalized color distance to the background color for all colors in the histogram. The first top-ranked color in the histogram that exceeds a predefined distance threshold is selected as the font color being a simple yet effective heuristic. In addition, we estimate the font size by comparing the bounding boxes from the hierarchy data and an instantiated bounding box containing the text. Since *Rico* often provides only rough bounding boxes for the *labels*, we compute refined bounding boxes using Tesseract-OCR (Smith 2007). For *buttons* and *text-inputs* we similarly extract the background and font colors. We enrich the *Rico* dataset with the identified style properties for enabling proper GUI component instantiation in the editor later. Finally, *(4)* the cropped GUI components are placed on their exact GUI screen positions. Subsequently, we describe our prototypical implementation.

2.4 Prototype implementation

The presented GUI retrieval and GUI screen derivation components are integrated in a prototype that we implemented as a web-based rapid prototyping editor using HTML5 and JavaScript. Our current prototype provides a GUI search view, a graphical prototyping editor and an interactive and dynamically updated preview of the built application as shown in Fig. 6. Subsequently, we provide more details about the features of our current implementation. Our current interactive prototype is publicly available at RaWi Prototype (2022).

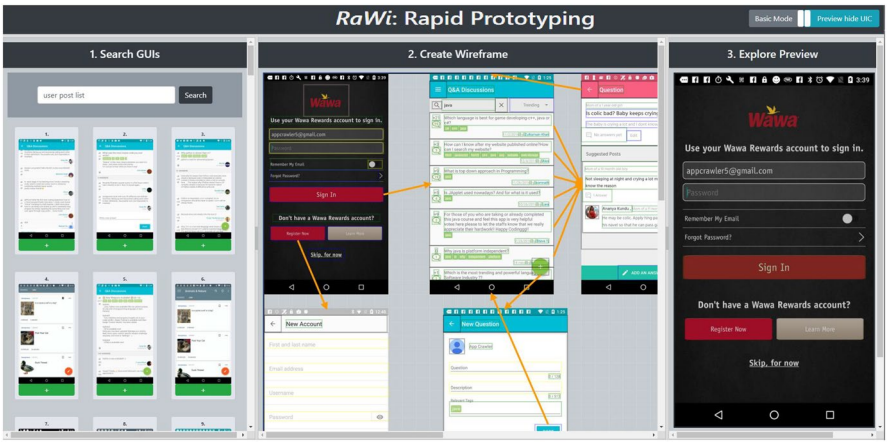


Fig. 6 Web-based GUI prototyping editor of our approach *RaWi* with (*1*) the GUI search view, (*2*) the graphical editor and (*3*) the prototype preview

2.4.1 GUI search

NL-based search queries can be submitted by users through a simple search view with autocompletion capabilities. The Python-based GUI retrieval component is accessible through a REST interface which is implemented in a *Django* application. In our interactive prototype, BM25 is the retrieval preset.

2.4.2 GUI prototyping editor

Users are enabled to add retrieved GUI prototypes to the graphical rapid prototyping editor. Here, the previously created enriched GUI hierarchy data is employed to create the editable GUI screens. Initially, we place all the image crops of layout and individual GUI components at their absolute positions on the GUI screen to preserve the complete original appearance of the GUI. By clicking on *labels*, *buttons* or *text-input* components, an editable version of the GUI component is instantiated and basic properties such as text, color and size can be modified. Furthermore, users can create a number of custom GUI components directly or reuse different components from other retrieved GUI screens. In addition, we currently provide basic editing functionality: GUI components can be removed, copied and bound to other GUI screens. To create an interactive prototype, *RaWi* provides the capabilities to create simple click-event transitions on the components. The graphical editor of our approach *RaWi* is based on the 2D canvas JavaScript library *KonvaJS* (Konva.js 2015).

2.4.3 GUI prototype preview

We provide a dynamically updated preview of the created prototype showing all modifications similar to traditional prototyping approaches. The preview enables

users to directly explore an interactive version of the created application prototype using previously created click-events as the foundation.

3 Experimental evaluation

In this section, we present the design and methodology of our experimental procedure to evaluate *RaWi*. In particular, the main goals of our experimental evaluation are *(i)* to measure the performance of the employed GUI retrieval techniques and conduct a comprehensive comparison between them on a newly created gold standard, *(ii)* to assess the productivity improvements of *RaWi* compared to a traditional rapid prototyping approach in a practical GUI prototyping environment and *(iii)* to measure the perceived usefulness and gain user insights for our approach. To this end, we investigate the following three research questions in our evaluation:

- $\mathbf{RQ}_1$: Which retrieval method performs best for GUI retrieval on the basis of NL search queries?
- $\mathbf{RQ}_2$: Does *RaWi* increase the GUI prototyping productivity compared to a traditional prototyping tool?
- $\mathbf{RQ}_3$: Do users perceive *RaWi* as useful for rapid high-fidelity GUI prototyping?

3.1 $\mathbf{RQ}_1$: GUI retrieval performance

3.1.1 Gold standard

Due to the absence of an evaluation dataset for GUI retrieval from NL queries, we decided to build a new and comprehensive gold standard through crowdsourcing techniques (Carvalho et al. 2011) and publicly release it to foster further research. The construction of the retrieval gold standard is divided into three main phases: *(i)* the collection of NL queries, *(ii)* the collection of relevancy annotations for potentially relevant GUIs with reference to NL queries and *(iii)* the derivation of the final gold standard via data cleansing. To accomplish the large-scale data collection, we used the well-established crowdsourcing platform Amazon Mechanical Turk (AMT) (Paolacci et al. 2010). The GUI retrieval gold standard is available at RaWi Repository (2022) including detailed descriptive statistics of the dataset (such as the distribution of contained app categories, the average number of tokens per query etc.), which will also be discussed briefly at the end of this section. Subsequently, we explain the three phases of the gold standard construction in detail as shown in Fig. 7.

First, *(i)* the collection of NL queries poses a challenge due to the difficulties of assuring high data quality for free-form NL-based crowdsourcing tasks (Rashtchian et al. 2010). Therefore, we decided to employ AMT for data collection but restricted the tasks to workers that we instructed (hired) specifically for the completion of this

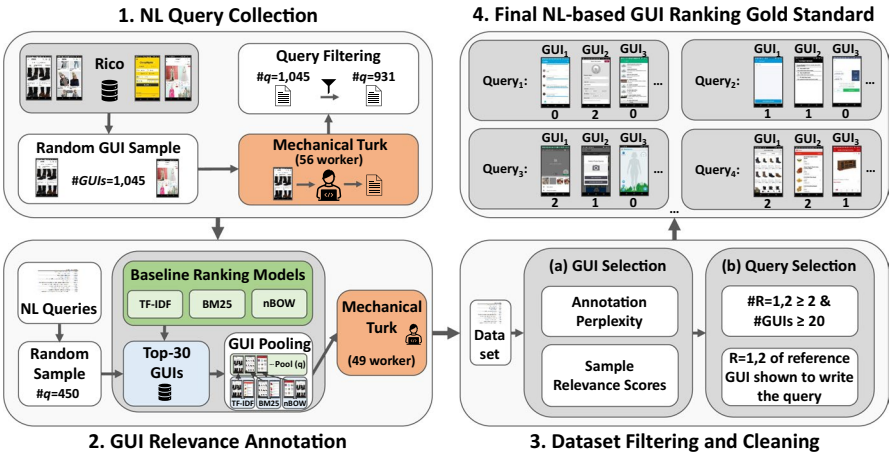


Fig. 7 Overview of the pipeline to create the gold standard for NL-based GUI ranking using crowdsourcing with the three main steps: (*1*) NL query collection, (*2*) GUI relevance annotation and (*3*) dataset filtering and cleaning

task. These annotators represent a good proxy for the population in our scenario, since workers on AMT typically have diverse demographics (Difallah et al. 2018; Ross et al. 2010) (potential threats to validity on worker selection are discussed later). As the foundation for writing queries and to avoid bias, we randomly sampled GUI screenshots from the filtered *Rico* dataset and asked the workers to write queries for them in a Human Intelligence Task (HIT). Overall, for each of 1045 unique GUIs from the sample, we gathered one unique NL query written by one of our overall 56 unique workers (resulting in 1045 GUI-NL-query pairs). For additional quality assurance, we manually reviewed the queries and filtered both six individual queries for GUIs that were erroneous or unusable screenshots from *Rico* (e.g. white screen, foreign language not detected by the language detection framework, among others) and all queries from six workers that made systematic errors due to misunderstandings of the task. By filtering these queries, only apparently erroneous NL queries were discarded, thus no bias is introduced but the quality of the dataset is enhanced. After filtering, we retain 931 GUI-NL-query pairs with queries written by 50 unique workers and apply basic text preprocessing methods (dequoting, stripping and removal of special characters). By employing AMT for query collection, we are enabled to gather a large amount of different NL queries from a variety of annotators, thereby increasing the heterogeneity of the NL queries the ranking models have to handle. Four GUI-NL-query pairs taken from the gold standard are shown in Fig. 8. The examples illustrate the diversity of the gathered NL queries ranging from short and general queries (e.g. Fig. 8a) to long and detailed search queries (e.g. Fig. 8c) across a broad selection of different application domains.

Second, *(ii)* to accomplish the collection of relevance annotations, we initially require a collection of potentially relevant GUI documents to be annotated for each query. Due to the infeasibility of annotating the relevancy of every GUI in *Rico* for each query, we decided to heuristically employ our three baseline retrieval models

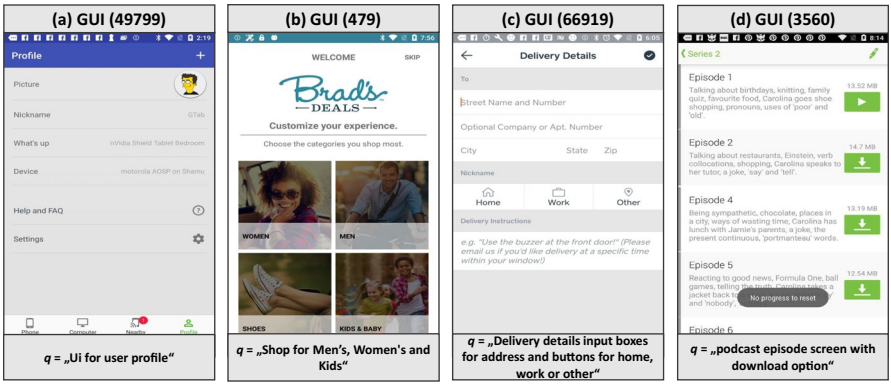


Fig. 8 Four **a**–**d** GUI-NL-query pairs taken from the gold standard with the NL queries gathered via crowdsourcing and the respective *Rico* GUI index

(*TF-IDF*, *BM25* and *nBOW*) to retrieve the potentially relevant top-30 GUIs for each query. To retain a diverse and heterogeneous GUI collection for each query, we apply the pooling technique (Jones and Van Rijsbergen 1976; Teufel 2007) on the GUI retrieval results of the three models by iterating over them in an alternating fashion and adding the top-ranked GUI not yet contained in the pool to the GUI collection. Moreover, we directly added the reference GUI that we previously showed to workers during the NL query harvesting to the pool of potentially relevant GUI documents. The pooling technique comprises the disadvantage of potentially excluding some relevant documents in the collection, however, represents a standard procedure for constructing large IR test collections by overcoming the infeasibility of annotating the large-scale corpus entirely. To keep the data collection feasible while ensuring a sufficient amount of data for a valid retrieval evaluation, we randomly sampled 450 queries from our previously created query collection, computed the pooled top-30 GUIs for each query with the approach described previously and published another HIT on AMT that asked workers for their relevancy annotations. Due to the subjectivity of the relevancy **R**, each GUI was annotated by three different workers on a relevancy scale of $\mathbf{R} = \mathbf{0}$ (*low relevance*), $\mathbf{R} = \mathbf{1}$ (*medium relevance*) and $\mathbf{R} = \mathbf{2}$ (*high relevance*). All 30 GUIs of an NL query were annotated exactly three times, resulting in 90 GUI relevance annotations per NL query by three distinct annotators. As a guidance for the annotators, we provided multiple annotation examples for the relevancy scale and included them in our accompanying material (RaWi Repository 2022). To assure high annotation quality, we restricted the task solely to workers that met multiple qualifications (*#Approved-HITs* $\geq$ 1000, *Approval-Rate* $\geq$ 90%, Residence in English-speaking country (*USA*/*GB*/*AU*)) as previous research found these metrics particularly effective for enhancing the annotation quality (Peer et al. 2014; Akkaya et al. 2010). In addition to these generic AMT platform requirements calculated based on the workers history, we created a custom qualification test to assess the comprehension of the relevancy criteria and the overall task of the annotators on a small set of GUI-NL-query pairs, which simultaneously acted as a training phase. Workers were required to meet a certain performance to be allowed

to work on the annotation task. Furthermore, we informed the workers that submissions were rejected if a submission scored below $\frac{2}{3}$ agreement with the majority voting of a query. Overall, we gathered 40,500 relevancy annotations (for 450 queries with each of the 30 GUIs annotated three times) from 49 unique workers.

Third, *(iii)* to obtain a high-quality gold standard from the previously harvested dataset, we run a pipeline of selection steps described subsequently. Due to the filtering, the final gold standard comprises a reduced set of 100 queries each with their top-20 GUIs. To increase data quality, we removed GUIs with high annotation perplexity. In particular, we discarded GUIs with annotation ties and where annotations diverged to the extremes, for example, two votes for $\mathbf{R = 0}$ and one vote for $\mathbf{R = 2}$ or vice versa since these annotations indicate high uncertainty towards deciding for the ground truth. Annotators may have different opinions on relevancy and are more or less strict in close annotation cases. Therefore, these annotation ties cannot be resolved and the respective GUIs are excluded. Afterwards, we set a suitable predefined ratio of different relevancy scores per query: $14 \times \mathbf{R = 0}$, $3 \times \mathbf{R = 1}$ and $3 \times \mathbf{R = 2}$. This provides a typical relevance distribution with mainly non-relevant GUIs while ensuring a small amount of relevant GUIs. For each query, we randomly sampled GUIs with the respective relevance scores according to the previously defined quantities. If the quantity for a relevance score could not be reached, we randomly sampled from the other scores to obtain 20 GUIs per query. However, we only kept queries that have a minimum of two $\mathbf{R = 1}$, two $\mathbf{R = 2}$ GUIs and 20 GUIs remaining. To obtain the same amount of GUIs per query, we decided to set the cutoff at the top-20. In addition, we discarded queries where the reference GUI that was used to write the query did not receive a relevance score of $\mathbf{R = 1}$ or $\mathbf{R = 2}$ from the human relevance annotators, as a query sanity check. The rationale for discarding these queries is to ensure high-quality in the written NL queries. If the NL queries do not match the shown reference GUIs (based on the relevance annotators judgements), the query is potentially of low quality (e.g., the annotator that wrote the query did not understand the GUI, provided an ambiguous query etc.). From the remaining query collection, we randomly sampled 100 queries to obtain the final gold standard. To illustrate the gold standard, an example NL query with three GUIs and their respective ground truth relevance annotations is shown in Fig. 9. Each of the GUIs comes with a different relevancy with respect to the NL query. For example, GUI_1 is annotated with $\mathbf{R = 2}$ since it fully corresponds to functionality requested in the query. However, GUI_2 only receives a relevancy of $\mathbf{R = 1}$ since the GUI contains the requested photo selection options, but the functionality is not embedded in a user profile context. Finally, GUI_3 is annotated with $\mathbf{R = 0}$ since it is unrelated to the query showing a user profile for water management.

To gain insights into the overall collected dataset and gold standard, we computed several statistics. To assess the Inter-Annotator Agreement (IAA) of the gold standard, we computed *Krippendorff's α* (Krippendorff 2011), a chance-corrected IAA metric supporting ordinal data. As a result, we obtain a score of $\alpha = 0.67$ which indicates a substantial agreement according to score interpretation methods (Landis and Koch 1977; Kraemer et al. 2012). In addition, the collected NL queries consist of 6.17 tokens on average (SD: 3.69) in our dataset, which indicates

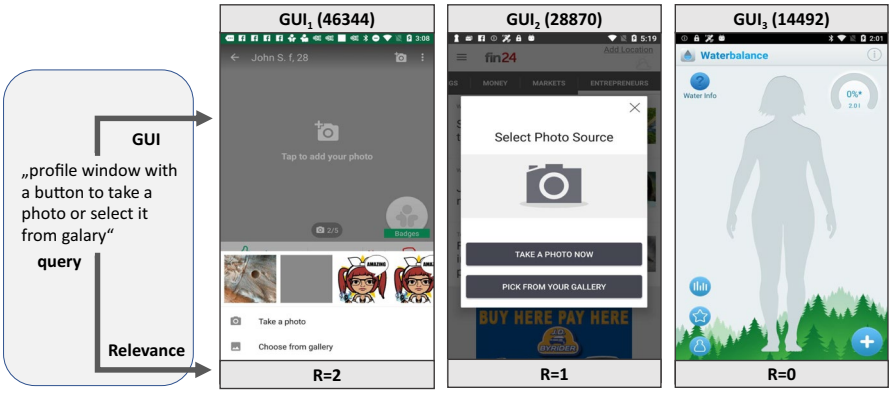


Fig. 9 Example of a NL query with three GUIs annotated with different relevancy levels with reference to the shown query taken from the gold standard

higher information need complexity and specificity for NL-based GUI searches in comparison to typical search queries, where most of the queries consist of only up to three tokens (Phan et al. 2007). Furthermore, based on the applied GUI sampling approach, the distribution of included app categories is similar to the distribution of the original *Rico* dataset, which corresponds to high diversity (26 app categories). Moreover, high diversity is also achieved among the queries indicated by a very small average pairwise Jaccard similarity (2.7%) over all queries in the gold standard.

3.1.2 Ranking model parameters

To evaluate the various ranking models in our experiments, we used TF-IDF, BM25 ($k_1 = 1.5$, $b = 0.75$, $\epsilon = 0.25$ as standard parameters Manning et al. 2008) and nBOW (using 300-dimensional *word2vec* embeddings Mikolov et al. 2013). The four considered AQE methods based on PRF-KLD are evaluated using BM25 as the base ranking model. In particular, we restricted the number of documents employed for expansion term computation to $k = 10$ and set the number of added expansion terms to $n = 10$. We used a similar setup for the text-segment-wise PRF-KLD variants, however, restricted the number of expansion terms per text segment to $n_{ts} = 2$. For brevity, we refer to the text-segment-wise PRF-KLD method as (s) and to the weighted variants as (w) in the evaluation results. For the BERT-LTR models, we used the pretrained large BERT-base version (Devlin et al. 2018) with a vector size of 768 and applied fine-tuning using the relevance annotations of the remaining 350 queries (9,495 instances) not contained in the gold standard. We employed the Tensorflow-Ranking BERT-Extension for training (Han et al. 2020) and train each model for 30,000 iterations (*learning-rate* $= 1 \times 10^{-5}$ and *batch-size* $= 1$). For the Sentence-BERT baseline (Reimers and Gurevych 2019), we employed the full pretrained 768-dimensional model with cosine similarity GUI ranking.

3.1.3 Evaluation metrics

To measure the performance of the GUI ranking models, we computed several well-known IR metrics. First, *(1)* we computed the Precision at rank k ($P@k$) as the fraction of relevant GUI documents $|R_k|$ over the number of all retrieved GUI documents k. Second, *(2)* we computed the Average Precision ($AveP$) as the average of $P@k$ values at all relevant GUI document positions with rel_k being an indicator for the relevance at position k and $|R|$ the number of relevant documents. The $P@k$ and $AveP$ metrics are shown in the following:

$$P@k = \frac{|R_k|}{k} \tag{1}$$

$$AveP = \frac{\sum_{k=1}^{n} P@k \times rel_k}{|R|} \tag{2}$$

Third, *(3)* we computed the Mean Reciprocal Rank (MRR), where $rank_i$ denotes the rank of the highest ranked relevant GUI document of the i-th query. In addition, *(4)* we computed the $HITS@k$ indicating if at least one relevant GUI document is ranked among the top-k ranked documents. To compute these metrics *(1)–(4)* that require binary relevance annotations on the employed relevancy scale, we binarize the relevance scores beforehand, thereby only high relevance scores ($\mathbf{R = 2}$) are considered relevant. In particular, the MRR and $HITS@k$ metrics are computed as follows:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{3}$$

$$HITS@k = \begin{cases} 1 & |R_k| > 0 \\ 0 & |R_k| = 0 \end{cases} \tag{4}$$

In order to further take into account the full annotation scale and the ranking position of GUI documents with high and medium relevance ($\mathbf{R = 2}$ or $\mathbf{R = 1}$), we additionally considered *(5)* the *Dicounted Cumulative Gain* at rank k ($DCG@k$). Relevant GUI documents ranked higher in the retrieval collection are more valuable and the relevance score should accordingly be weighted more than the score of GUI documents at lower ranked positions. Hence, in $DCG@k$ each relevancy score is discounted by a weight (log-based smoothing) that is computed based on the ranking position which decreases with lower ranking positions. The $DCG@k$ is normalized by the $IDCG@k$ (Ideal $DCG@k$) to obtain *(6)* the $NDCG@k$ (normalized $DCG@k$), which is employed in our evaluation. The $IDCG@k$ is computed as the $DCG@k$ of relevancy scores sorted in descending order. To compute the final results, all metrics are averaged over all queries of the GUI ranking gold standard. Subsequently, the $DCG@k$ and $NDCG@k$ metrics are shown, with rel_i denoting the relevance score at the i-th rank:

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)} \tag{5}$$

$$NDCG@k = \frac{DCG@k}{IDCG@k} \tag{6}$$

3.2 RQ_2: productivity of rapid prototyping

3.2.1 User study design

To measure potential productivity improvements of our data-driven GUI prototyping approach, we conducted a controlled experiment with 19 participants as a within-subjects study. In the experiment, we compared our approach to a traditional high-fidelity GUI prototyping approach. With reference to the discussed requirements elicitation scenario, participants in the role of analysts were asked to create two different application prototypes (consisting of two GUIs each) on the basis of typical NL requirements, with each approach separately to mitigate carry-over effects. To avoid bias from both the tasks and the ordering of the approaches, we considered all four combinations of the approaches A/B and the tasks 1/2 (A1B2/A2B1/B1A2/B2A1) and assigned the conditions to participants randomly while ensuring that the occurrence of conditions is evenly distributed. As the approach for comparison, we employed *Mockplus* (Mockplus 2014) since it can be regarded as a representative tool for traditional GUI prototyping approaches, providing a restricted number of manually crafted GUI screens to reuse and enabling users to build GUIs by combining a large number of searchable GUI components in a graphical editor. Overall, we recruited 19 participants with technical backgrounds (BSc: 7/MSc: 8/PhD: 4) mainly having medium to high experience in software development (Mean: 3.52/SD: 1.02) and mainly having little to medium GUI prototyping experience (Mean: 2.42/SD: 0.96) as self-reported by the participants on a five-point Likert scale. Most of the participants (13) reported that they used a prototyping tool before, with the most frequently mentioned ones being general-purpose tools such as *PowerPoint* (9), *Photoshop* (2) and *Adobe XD* (2). Thus, the participants are representative regarding the requirements elicitation scenario, especially targeting to support the inexperienced analysts.

3.2.2 User study tasks

Both GUI prototyping tasks are based on real Android applications within the top-50 apps on Google Play, which are not included in the *Rico* GUI repository to avoid bias. The first task represents a shopping browser consisting of a start GUI asking the user to select from a list of countries (#GUI-comps: 25) and the actual main shopping browser GUI that allows users to search for websites and displays the favorite websites (#GUI-comps: 33), among others. The second task represents

a hotel booking application consisting of a GUI that allows users to enter search details (#GUI-comps: 25) and a GUI that shows a list of search results (#GUI-comps: 29). We omit the detailed task descriptions and NL requirements for brevity, however, provide all study instructions among the other publicly released materials in our accompanying repository (RaWi Repository 2022). We decided to employ these GUIs in our experiments due to multiple reasons. First, *(i)* the GUIs contain a large number of different GUI components, thus providing difficulty to create especially for novice users. Second, *(ii)* both approaches *RaWi* and *Mockplus* cover both the shopping and booking application domain with basic screens allowing for a fair comparison.

3.2.3 Experimental procedure

In the study, we asked the participants to create a GUI prototype with each approach separately. Depending on the assigned condition, the participant started with the first approach and task. Before working on the actual task, we showed a short tutorial video of the approach with a GUI prototyping example and allowed the participants as much time as they needed to test the approach. Afterwards, the actual task was conducted. With regard to the interactive GUI prototyping scenario with customers for requirements elicitation, the available time for prototyping is highly limited. Therefore, we accordingly restricted the available time for creating the GUI prototypes to several minutes, with a maximum of seven minutes. To obtain a fine-grained productivity measurement, we captured screenshots of the current GUI prototype state after each minute starting from three minutes and thus collected five screenshots per GUI. These screenshots formed the basis for our post-experiment analysis. After completing the first task, we continued with the identical procedure for the second approach and task. Overall, we gathered 20 GUI screenshots per participant.

3.2.4 Evaluation metrics

From the gathered screenshots, we computed multiple metrics. However, no prior research proposed metrics for assessing GUI prototyping productivity before. In particular, a closely related problem on GUI implementation effort estimation (Lo et al. 1996) finds especially the number and type of GUI components as valuable factors for predicting the overall GUI implementation effort. In a similar fashion, GUI prototyping in the requirements elicitation phase asks analysts to quickly select and arrange GUI components that properly reflect the requirements specified by the customer. Here, one or more GUI components represent a particular functional requirement and therefore, we consider the number of GUI components as a proxy for GUI prototyping productivity. Thus, we first *(a)* count the number of correct (based on GUI component type e.g. using a password field instead of a plain text field for the functionality to enter passwords etc.) GUI components available at the considered GUI prototype at time t appropriate for the given requirements as an approximation for productivity (denoted by #GUI-comps). GUI components that overlapped the GUI screen or other GUI components were not included in the count. Detailed design decisions (e.g. placing a button bar on top or bottom or similar)

are apparently neglected in this metric, since in the requirements elicitation phase, the focus lies on quickly representing the main functionality and not the final GUI design. Different working styles and other relevant individual factors of participants that influence the outcome are considered by our within-subject design, comparing productivity only between the participants themselves. Second, *(b)* we count the number of GUI components as before, however, data elements (e.g. images or text) are only counted multiple times if they contain realistic and diverse data (e.g. not simple template and copied data) (denoted by #GUI-comps-div). We considered to compute this corrected count since having high content diversity through realistic data is important for more realistic high-fidelity GUI prototypes. Third, *(c)* we count the number of incorrect GUI components available at time t that were not included in the requirements (denoted by #GUI-comps-neg). Finally, *(d)* we compute an adjusted count where including non-required components is punished by taking the count of required GUI components *(a)* minus the count of GUI components not included in the requirements *(c)*. In our evaluation, we want to compare the productivity between the two approaches at each time step t. To evaluate if the productivity is significantly higher using our approach, we compute the Wilcoxon signed-rank test (Woolson 2007) over all pairs of study tasks. Therefore, we compare the productivity of each participant between both first GUIs of the tasks and both second GUIs of the tasks. Moreover, we compute Cliff's δ (Macbeth et al. 2011) to estimate the overall statistical effect size.

3.3 RQ_3: perceived usefulness

In addition to measuring the GUI prototyping productivity, we assessed the perceived usefulness of our GUI prototyping approach by the study participants. Using a five-point Likert scale, we asked participants *(a)* how much our approach supported them during the prototyping process, *(b)* how relevant the retrieved GUIs were on average, *(c)* how much the GUI retrieval results supported the users during the prototyping and *(d)* how much the GUI search results helped in better visualizing how the GUI could be prototyped. Moreover, we computed the System Usability Scale (SUS) (Brooke 1996) which is a reliable and well-known measurement for system usability. To gain further insights, we asked the participants for free-form feedback to potentially discover interesting ideas for improvements of our GUI prototyping approach.

4 Results and discussion

4.1 RQ_1: GUI retrieval performance

Table 1 shows the evaluation results of the different baseline ranking models, AQE techniques and BERT-based LTR models for the $P@k$ and $NDCG@k$ metrics, whereas Table 2 shows the evaluation results for the AP, MRR and $HITS@k$ metrics. Considering the three baseline ranking models, we can observe that BM25

Table 1 Evaluation results overview of the different models on the NL-based GUI ranking gold standard using the Precision@k (p@k) and NDCG@k metrics

	P@k				NDCG@k (N@k)			
	P@3	P@5	P@7	P@10	N@3	N@5	N@10	N@15
TF-IDF	0.223	0.204	0.181	0.175	0.329	0.339	0.395	0.480
BM25	**0.303**	**0.276**	**0.246**	**0.226**	**0.426**	**0.441**	**0.515**	**0.579**
nBOW	0.270	0.234	0.220	0.193	0.395	0.370	0.374	0.398
BM25	0.303	0.276	0.246	0.226	0.426	0.441	0.515	0.579
+PRF	0.313	0.266	0.259	0.236	0.432	0.443	0.520	0.584
+PRF (s)	0.317	**0.280**	**0.260**	0.235	**0.441**	**0.462**	**0.536**	**0.604**
+PRF (w)	**0.320**	0.276	0.244	0.231	0.439	0.417	0.421	0.446
+PRF (sw)	0.317	0.280	0.256	**0.237**	0.429	0.416	0.426	0.456
Sentence-BERT	0.343	0.314	0.294	0.267	0.481	0.511	0.611	0.667
BERT-LTR(1)	0.377	0.350	0.307	0.269	0.530	**0.560**	0.634	0.697
BERT-LTR (2)	**0.400**	0.340	0.304	0.281	**0.543**	0.556	0.636	**0.701**
BERT-LTR (3)	0.363	**0.354**	**0.317**	**0.287**	0.517	0.554	**0.646**	0.694

Bold values indicate best metric score within the result group (baselines, AQE methods and BERT)
Underline values indicate best metric score over all result groups

outperforms the other two models to a large extent across most of the examined metrics. In particular, the *MRR* of the BM25 model indicates that the first relevant GUI appears at rank 1.92 on average over the gold standard. Moreover, the *HITS*@5 indicates that in 69% of the queries the BM25 model ranked at least one relevant GUI among the top-5. Thus, BM25 seems to be an effective baseline model for NL-based GUI retrieval. Comparing the plain TF-IDF and nBOW models, we can observe that especially the *P@k* and *HITS@k* as well as the *NDCG@k* for small *k* are higher for the nBOW model, indicating that pretrained word embeddings can provide additional benefit for GUI retrieval from NL searches.

In addition, both tables show the results of the four different AQE methods based on the PRF-KLD score using the previously best performing baseline ranking model BM25 as the base ranker. The results show that the text-segment-wise PRF (s) method obtains the highest scores among most of the considered metrics and clearly outperforms the BM25 base model across all the ranking metrics. Both weighted PRF-KLD variants can outperform the PRF (s) method for some individual metrics and mainly perform better for the binary metrics *P@k* and *HITS@k* compared to the BM25 base ranker. However, both are outperformed on the *NDCG@k* metric that takes into account the entire relevancy scale and additionally requires to rank GUIs with medium relevance high. Overall, the results indicate that AQE techniques in the form of PRF-KLD can improve the performance for NL-based GUI retrieval compared to the baseline BM25 ranking model on average, especially the variant that takes the GUI-specific text segments into account. Additionally, we provide a more detailed per-query analysis of the AQE evaluation results in our accompanying

Table 2 Evaluation results overview of the different models on the NL-based GUI ranking gold standard using Average Precision (AP), MRR and HITS@k

	AveP	MRR	HITS@k (H@k)					
			H@1	H@3	H@5	H@7	H@10	H@15
TF-IDF	0.331	0.451	0.320	0.460	0.580	0.650	0.760	0.910
BM25	**0.413**	**0.520**	**0.370**	**0.600**	**0.690**	**0.760**	**0.860**	0.930
nBOW	0.281	0.490	0.340	0.540	0.630	0.750	0.840	**0.980**
BM25	0.413	0.520	0.370	0.600	0.690	0.760	0.860	0.930
+PRF	0.419	0.505	0.370	0.580	0.680	0.780	0.850	0.930
+PRF (s)	**0.427**	0.532	0.380	**0.610**	**0.700**	**0.780**	**0.880**	**0.960**
+PRF (w)	0.325	0.523	0.380	0.580	0.700	0.720	0.860	0.930
+PRF (sw)	0.333	**0.533**	**0.390**	0.590	0.700	0.770	0.870	0.930
Sentence-BERT	0.454	0.560	0.370	0.680	0.760	0.880	0.960	0.990
BERT-LTR (1)	0.486	0.618	**0.460**	0.710	0.860	0.920	0.980	1.00
BERT-LTR (2)	**0.501**	**0.631**	0.440	**0.750**	**0.910**	**0.960**	0.980	**1.00**
BERT-LTR (3)	0.499	0.626	0.450	0.730	0.860	0.940	**1.00**	1.00

Bold values indicate best metric score within the result group (baselines, AQE methods and BERT)
Underline values indicate best metric score over all result groups

material (RaWi Repository 2022). For example, the *MRR* and *P*@5 metrics were improved or at least as good in 69% and 77% of the queries with an average improvement of .19 and .07 compared to the base model BM25 by applying the PRF (s) method. In particular, for queries that have many relevant GUIs in the GUI repository and the initial retrieval results with the base model BM25 provides relevant and cohesive GUIs, the AQE method can improve the results. For instance, for the query "*a list of songs*" meaningful expansion words such as *artist*, *bookmark*, and *play* can be extracted, which strengthen the query.

Considering the results of the trained BERT-LTR models, we can observe that all of the three models significantly outperform both the baseline ranking and AQE models. In our experiments, the pairwise (BERT-LTR (2)) and pointwise (BERT-LTR (3)) model appear to mainly perform best among the examined models. In addition, the pretrained Sentence-BERT baseline similarly outperforms the standard baseline ranking and AQE models, however, is significantly outperformed by the trained BERT-LTR models across all metrics. Regarding the *NDCG@k* metrics, the BERT-LTR models outperform the Sentence-BERT baseline to a large extent, indicating that the models learned to rank GUIs with both medium ($\mathbf{R = 1}$) and high relevance ($\mathbf{R = 2}$) higher in the overall ranking. Overall, the results indicate that the pretrained and fine-tuned BERT language model helps to better represent the semantics of the query and GUI documents and that, in combination with a LTR model trained specifically on the task of GUI ranking, it can be effectively applied to significantly improve the NL-based GUI ranking performance compared to the examined traditional IR methods and the pretrained Sentence-BERT baseline. In particular, these specifically trained semantic models are better in bridging the gap

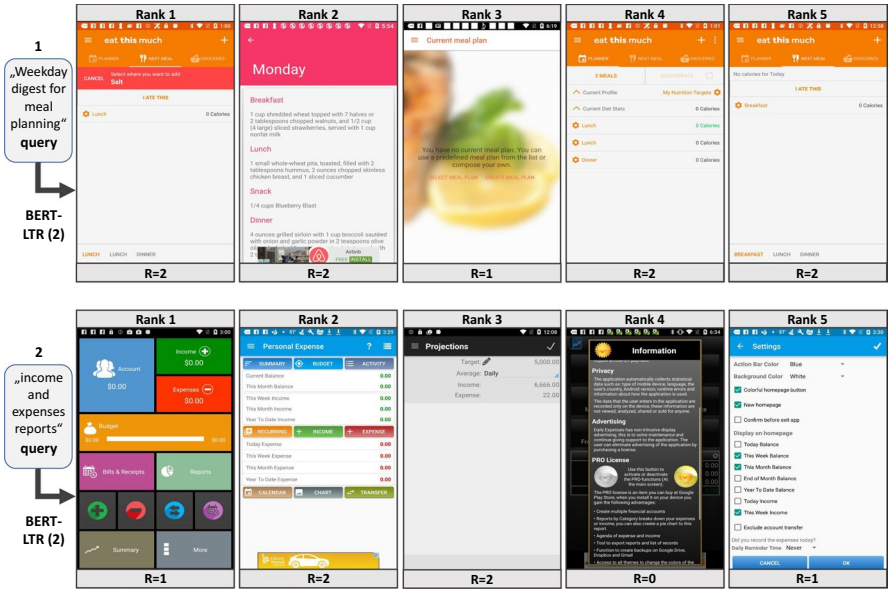


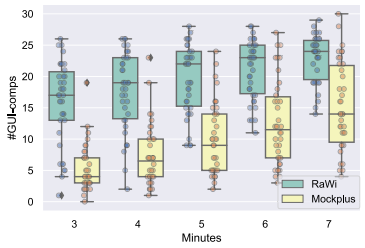
Fig. 10 Example of two queries from the gold standard with the respective top-5 GUI rankings with relevance scores ranked by the BERT-LTR (2) model

between the language used in NL queries and the GUI text representations. Figure 10 shows two example queries taken from the gold standard with their respective top-5 GUIs as ranked by the trained BERT-LTR (2) model. Each GUI is annotated with its ground truth relevance. For the queries we can observe that only a single GUI is annotated with a low relevance (due to an information overlay), whereas all other GUIs appear to have a medium to high relevance for the queries, which shows the ranking strength of the BERT-LTR.

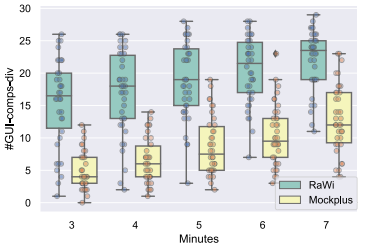
4.2 RQ_2: productivity of rapid prototyping

In Fig. 11 we show the evaluation results of our controlled experiment. Each of the four boxplots *(a)–(d)* shows one of the four #GUI-comp metrics as a proxy for productivity as explained earlier. For the basic #GUI-comp count shown in Fig. 11a, we can observe that our approach outperforms the traditional approach to a large extent at each time step t. Naturally, the counts in both approaches increase with increasing t, however, the differences between the two approaches become smaller with increasing t. Participants are often able to find suitable starting GUI screens via adequate searches in *RaWi* and thus often begin with higher counts followed by smaller adjustments (adding and removal of components). In contrast, in the traditional approach it was often more difficult for participants to find an appropriate GUI at the beginning, however, repetitive component groups could be copied quickly after initial creation which led to a strong increase in

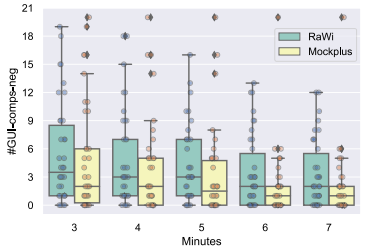
counts over time *t*. When working with *RaWi*, participants were challenged to extract good queries on the basis of the given NL requirements and select relevant starting GUIs. Here, participants were not always able to find optimal starting GUIs, however, productivity improvements typically could still be observed in comparison to the traditional prototyping approach since *RaWi* simplified the task in general. For the #GUI-comp-div counts that take into account the content diversity shown in Fig. 11b, we can observe a similar behaviour as in *(a)*, however, the differences between the two approaches are apparently larger. GUIs retrieved with *RaWi* typically display already diverse and realistic data since they are derived from GUI screenshots of real applications. In contrast, participants often copied repetitive component groups and often were not able to provide diverse data in their GUI prototypes with the traditional approach. For the #GUI-comp-neg counts that measure the components available not requested shown in Fig. 11c, we can observe that in both approaches the counts decrease over time *t* due to the removal of components available in the initial GUI screens that were not required, however, at the same time the GUI prototypes created with *RaWi* tend to contain more unrequested components. In *RaWi*, participants frequently found suitable screens in the beginning, yet often these GUIs included many



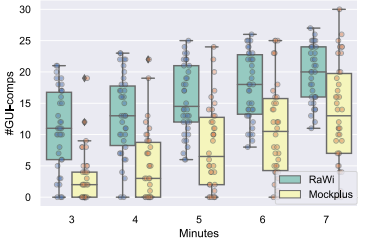
(a) #GUI-comps correct based on the requirements



(b) #GUI-comps-div with content diversity



(c) #GUI-comps-neg that were not requested



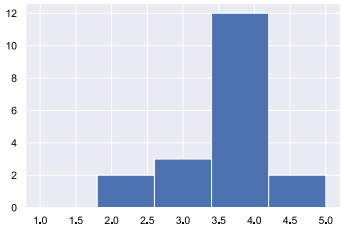
(d) #GUI-comps of (a) adjusted by #GUI-comps-neg (c)

Fig. 11 Evaluation results of our controlled experiment: Each boxplot **a**–**d** shows one of the computed #GUI-comp metrics that the participants achieved with each approach *RaWi* and *Mockplus* at each minute *t* from minutes 3–7

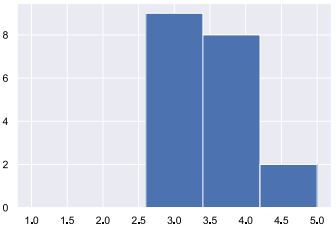
additional components for other specific requirements that were not relevant for the requested prototype. However, we can observe a similar phenomenon in the traditional approach were participants that started with unsuitable GUI screens obtained high counts indicated by the outliers. In particular, one participant used a GUI with many unrequested components and did not manage to remove them up to minute 7. In addition, users of *RaWi* tend to focus on adding more requested functionality not yet contained in the starting GUIs and neglect removing unrequested components initially. This could potentially be due to the fact that the unrequested components in RaWi were possibly often perceived as less wrong (e.g., a star rating for each of multiple search results) compared to unrequested features in the templates in the traditional approach, and therefore potentially neglected initially more. For the adjusted #GUI-comp counts that represents the difference between the original count in *(a)* and *(c)* shown in Fig. 11d, we still can observe that our approach has higher counts at time t, however, the differences become smaller when punishing the prototypes for containing unrequested components. In addition to the differences observed in the boxplots, the Wilcoxon signed-rank test results indicate that the counts as a proxy for productivity using our approach are statistically significant larger at all considered time steps t (p-value < 0.05) and all considered counts *(a)*, *(b)* and *(d)*. Moreover, Cliffs δ indicates a *large* difference between counts of the approaches also in the cases where the difference appears smaller in the boxplots. Overall, the results indicate that our approach can improve the prototyping productivity compared to a traditional GUI prototyping approach and therefore provides a better applicability for interactive GUI prototyping.

4.3 RQ_3: perceived usefulness

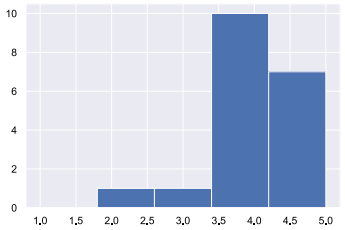
Figure 12 shows the evaluation results of the perceived usefulness of our GUI prototyping approach. The first question *(a)* regarding the support during the prototyping process received a high score (Mean: 3.73/SD: 0.80). The second question *(b)* regarding the perceived relevancy of the search results received a similarly high score (Mean: 3.63/SD: 0.68). The third question *(c)* regarding the support of the results received an even higher score (Mean: 4.21/SD: 0.78). Finally, the fourth question *(d)* regarding the support of the search results for better visualization received the highest score (Mean: 4.47/SD: 0.61). Overall, we achieved a SUS of 81.57 which indicates that our approach has a high usability compared to the average SUS of 68. Participants found our approach very easy to understand and use, even for inexperienced and novice users in GUI prototyping. In general, the participants reported further that they liked the large choice of available GUI screens to create a first GUI prototype quickly. Concerning improvements of the approach, participants mainly reported that the editor functionalities should be extended to include more editing options.



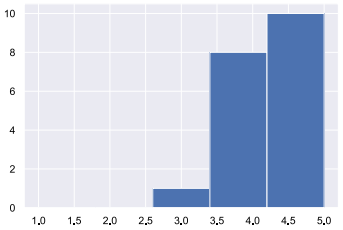
(a) Support during GUI prototyping



(b) Average relevancy of GUIs for query



(c) Support of GUI search results during GUI prototyping



(d) Support of GUI search results for visualizing how the screen could look

Fig. 12 Four histograms showing the results to the four usability questions **a**–**d** regarding *RaWi* for rapid GUI prototyping as perceived by participants

5 Threats to validity

First, we discuss threats to internal validity which refers to potential bias comprised in our experimental evaluation. To reduce bias and subjectivity in the creation of the gold standard, we conducted several steps. Due to the absence of any prior accessible system that could be used to extract search queries for GUIs, we employed 50 workers from AMT to write queries on a random sample of the *Rico* GUIs to reduce bias during the query creation. AMT provides a large amount of workers with diverse demographics, suiting the need for including a large bandwidth of different ways of formulating queries, use of language and vocabulary. However, there could be potential selection bias. For example, workers that are willing to work on the task may have a specific interest in apps and thus may have more knowledge on the topic compared to the average worker on AMT. Similarly, we asked 49 workers to provide relevance judgments and annotated each GUI by three workers to reduce subjectivity. These workers were required to pass our task-specific qualification test, which served the purpose of increasing the relevance annotation quality. Final relevance annotations are based on majority voting and GUIs with high annotation perplexity were discarded due to the high uncertainty of the ground truth relevance for these instances. To obtain a high-quality gold standard, we applied several filters and randomly sampled from both the remaining queries and the GUIs to avoid selection bias. To reduce bias in our controlled experiment, we randomized the tasks

and the ordering of the approaches. In addition, we ensured that the applications employed in the user study tasks are not included in *Rico* and both approaches basically support the selected categories. For each approach we conducted the identical experimental procedure.

Second, we discuss threats to external validity which refers to the generalizability of the results obtained in our experimental evaluation. Similarly, due to the absence of any prior system for GUI search that could be used as a resource for obtaining real-world queries, we employed GUIs taken from *Rico* to write queries for. We already discussed the plenty reasons of employing *Rico* and emphasize that *Rico* is large-scale and covers many diverse domains making it applicable in plenty scenarios. In addition to many domain-specific GUIs, *Rico* provides a plethora of domain-independent GUIs. Using GUIs from *Rico* as the basis for writing queries also assures that models could at least principally retrieve a single relevant GUI for the query. We restricted our controlled experiment to two domains, however, we hypothesize that we could obtain similar results for other domains covered by *Rico*. However, more user evaluation is required that includes more diverse application domains to confirm the obtained results. Moreover, since *Rico* is a semi-automatic approach it could be scaled up to harvest millions of GUIs making our GUI prototyping approach even more valuable for more domains.

6 Limitations

Our retrieval is currently restricted to the *Rico* dataset which, however, is large-scale and covers many diverse domains already. To extend the GUI repository, other smaller GUI datasets could be integrated. Since prototypes are created on the basis of GUIs from different real applications in *RaWi*, the design of the resulting application prototype may not be cohesive. However, the focus of our work lies on rapidly creating GUI prototypes that properly reflect requirements of customers neglecting the final GUI design. Since the implementation of our GUI prototyping approach is an early prototype, the editing functionality is currently restricted, however, we plan to extend our prototype to a fully-fledged graphical GUI editor with more editing options.

7 Related work

7.1 GUI retrieval

Guigle (Bernal-Cárdenas et al. 2019) similarly exploits automatically crawled Android apps from Google Play (Moran et al. 2018) to index multiple parts from the crawled GUI hierarchy data such as the app name, text and type from GUI components, the screen color and employs a basic Boolean query language for relevant GUI retrieval. In contrast, our approach *RaWi* enables users to specify simple NL-based searches to retrieve relevant GUIs and employs a more sophisticated ranking mechanism including BERT-based LTR models. In addition, *RaWi* enables users to directly employ retrieved GUIs to rapidly create GUI prototypes including the derivation of partly editable screens whereas

Guigle stops after GUI image retrieval. However, a direct comparison of the retrieval performance is not possible due to the unavailability of the implementation and the employment of a different, much smaller GUI dataset in *Guigle*. As stated in their paper, *Guigle* is built on top of the open-source retrieval framework *Lucence* (Lucene 2011). The *Lucene* framework employs TF-IDF and BM25 retrieval methods, which are included in our evaluation experiments as well-known standard IR baselines. Therefore, although implementation details such as preprocessing and retrieval function parameters are not specified in *Guigle*, the presented IR baselines can be considered a good proxy to the retrieval methods employed in their approach. To support further research and enable a direct comparison between approaches in the first place, we publicly provide the first and high-quality gold standard for NL-based GUI retrieval on a state-of-the-art GUI repository in this work. Prior research already introduced NL-based GUI retrieval (Kolthoff et al. 2020, 2021), however, we *(i)* considerably extend the retrieval techniques using BERT-based LTR models, *(ii)* conduct a novel and more comprehensive retrieval performance evaluation on the basis of a newly created and contributed high-quality gold standard and *(iii)* provide an extensive user study for evaluating the GUI prototyping productivity improvements in practical GUI prototyping settings. *Gallery D.C.* (Chen et al. 2019) crawls a large number of real-world applications and automatically extracts GUI components such as various types of buttons from GUI screenshots and provides a multi-modal search supporting multiple dimensions such as width, height, main color and text for them. In addition, there is a plethora of GUI retrieval approaches that employ visual inputs such as screenshots, hand-drawn sketches or basic wireframes to retrieve similar GUIs in contrast to our approach that focuses on NL input. For example, Chen et al. (2020) trains an image autoencoder to find relevant GUIs employing a basic wireframe of the GUI as their input. Similarly, *VINS* (Bunian et al. 2021) expects visual input either as a basic wireframe prototype or fully designed GUI image to retrieve similar GUIs using a multi-modal embedding network. *Swire* (Huang et al. 2019) also employs a visual embedding approach to find relevant GUI images on the basis of hand-drawn sketches provided by the user. *GUIFetch* (Behrang et al. 2018) requires an entire Android application sketch as input to retrieve similar apps from GitHub. *d.tour* (Ritchie et al. 2011) uses stylistic keywords and stylistic similarity to find similar website designs. Using a web design as input and structure matching, *FaceOff* retrieves web GUI components fitting the design. Another approach proposes a neural translator for transforming GUI images to GUI skeletons (Chen et al. 2018). *Screen2Vec* (Li et al. 2021) proposes a technique for learning multi-modal (textual content, visual design and layout patterns) GUI embeddings, which are useful for many GUI-related downstream tasks (e.g., retrieving similar GUIs using a GUI as the query). Due to the difference of the problem, these GUI embeddings cannot be directly reused in our approach for comparison. The NL query embeddings (e.g., based on BERT) and *Screen2Vec* embeddings represent two separate embedding spaces that require alignment in order to enable NL-based GUI retrieval with *Screen2Vec* GUI embeddings. However, we included the same text-only Sentence-BERT baseline in our experiments, also used in their approach. The *Screen2Words* (Wang et al. 2021) approach proposes an Encoder–Decoder architecture for translating GUIs into short text descriptions. The GUI encoder could potentially be employed to obtain GUI embeddings, however, these embeddings similarly cannot be directly employed for retrieval based on NL queries as described earlier. Considering general-purpose search engines (e.g., *Google*),

a direct comparison with our approach is not meaningful since these engines cannot be evaluated directly on our gold standard. Especially due to the huge differences in the employed datasets (specialized Rico GUI dataset vs. general Google Image dataset), it would be unclear whether differences between the retrieval performances are due to differences in the algorithms or due to the differences in the datasets. For example, since Google uses a general image dataset, many image results would be obtained that are not related to GUIs, compared to the Rico dataset specializing in Android GUI screenshots. To make the comparison between these approaches meaningful, the algorithm would need to be evaluated directly on the new gold standard.

7.2 Program code retrieval

Apart from GUI search approaches, retrieving code through NL queries has been studied extensively in research before (McMillan et al. 2011; Cambronero et al. 2019; Gu et al. 2018; Lv et al. 2015; Zhang et al. 2016). These approaches range from the application of traditional IR algorithms to specifically developed semantic Deep Learning architectures to find relevant program code for a given NL query. Recently, the *CodeSearchNet* challenge (Husain et al. 2019) released a dataset of NL queries and expert relevance annotations of respective functions from various programming languages to provide a source for systematic evaluation of code search approaches. In contrast, we created the first GUI retrieval gold standard in this work, to foster further research on retrieval methods specifically for NL-based GUI search.

7.3 GUI prototyping

In addition, many traditional prototyping approaches exist and are employed by many designers, developers and analysts in their everyday work such as *Balsamiq* (Faranello 2012), *Sketch* (Sketch 2019), *Figma* (Figma 2016) and *Mockplus* (Mockplus 2014), among others. These approaches enable users typically to create prototypes in a graphical editor on the basis of a small number of hand-crafted templates and basic GUI components supporting low-fidelity or high-fidelity prototyping. Other approaches such as UISKEI (Segura et al. 2012) and SketchiXML (Coyette et al. 2006) attempt to automatically identify GUI components from hand-drawn sketches and generate reusable GUI representations. Recent work on GUI prototyping assistance such as *GUIComp* (Lee et al. 2020) support novice users during the prototyping process via the recommendation of similar GUIs, provide various complexity metrics and visual attention maps based on the design currently created by the user. In contrast to these approaches, *RaWi* enables users to quickly retrieve matching GUIs based on simple NL queries from a large-scale GUI repository and automatically provides partly editable GUI screens to reuse for requirements elicitation via interactive GUI prototyping.

8 Conclusions and future work

In this work, we presented *RaWi*, a data-driven GUI prototyping approach based on exploiting a large-scale GUI repository via effective NL-based GUI retrieval methods and automatically deriving partly editable GUI screens for interactive GUI prototyping in the requirements elicitation phase. Our evaluation indicates that traditional retrieval models and especially state-of-the-art BERT-based semantic ranking models, can be adopted for effective GUI retrieval. In addition, our approach is able to improve the prototyping productivity particularly for novice analysts in comparison to a traditional GUI prototyping approach.

In future work, we plan to further investigate BERT-based LTR semantic models for NL-based GUI ranking. We also plan to examine the applicability of multilingual language models to support GUI retrieval for various languages. This could be extended by automatically translating the editable GUI screens into target languages. In addition, we plan to further extend our evaluation to more users and examine additional domains.

Author contributions Conceptualization, K.K., C.B. and S.P.; Methodology, K.K., C.B. and S.P.; Implementation, K.K.; Validation, K.K.; Data Curation, K.K.; Writing-Original Draft Preparation, K.K.; Writing-Review and Editing, K.K., C.B. and S.P.; Visualization, K.K.; Supervision, C.B. and S.P.; Funding Acquisition, C.B. All authors have read and agreed to the published version of the manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL. The authors did not receive support from any organization for the submitted work.

Data availability The code and datasets generated and analyzed within this paper are available in the RaWi repository at https://doi.org/10.5281/zenodo.7053440 (accessed on 6 September 2022).

Declarations

Conflict of interest The authors declare no conflict of interest.

Informed consent Not applicable.

References

Adobe XD: Adobe xd. https://www.adobe.com/de/products/xd.html (2015). Accessed 21 March 2021

Akkaya, C., Conrad, A., Wiebe, J., et al.: Amazon mechanical turk for subjectivity word sense disambiguation. In: Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk, pp. 195–203 (2010)
Azad, H.K., Deepak, A.: Query expansion techniques for information retrieval: a survey. Inf. Process. Manag. **56**(5), 1698–1735 (2019)
Beaudouin-Lafon, M., Mackay, W.: Prototyping development and tools. Handbook of Human-Computer Interaction, pp. 1006–1031 (2002)
Behrang, F., Reiss, S.P., Orso, A.: Guifetch: supporting app design and development through GUI search. In: Proceedings of the 5th International Conference on Mobile Software Engineering and Systems, pp. 236–246 (2018)
Bernal-Cárdenas, C., Moran, K., Tufano, M., et al.: Guigle: a GUI search engine for android apps. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 71–74. IEEE (2019)
Brooke, J.: SUS: a quick and dirty usability. Usability Eval. Ind. **189** (1996)
Bunian, S., Li, K., Jemmali, C., et al.: Vins: Visual search for mobile user interface design. arXiv preprint arXiv:2102.05216 (2021)
Cambronero, J., Li, H., Kim, S., et al.: When deep learning met code search. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 964–974 (2019)
Carpineto, C., De Mori, R., Romano, G., et al.: An information-theoretic approach to automatic query expansion. ACM Trans. Inf. Syst. (TOIS) **19**(1), 1–27 (2001)
Carvalho, V.R., Lease, M., Yilmaz, E.: Crowdsourcing for search evaluation. In: ACM Sigir Forum, pp. 17–22. ACM, New York (2011)
Chen, C., Su, T., Meng, G., et al.: From UI design image to GUI skeleton: a neural machine translator to bootstrap mobile GUI implementation. In: Proceedings of the 40th International Conference on Software Engineering, pp. 665–676. ACM (2018)
Chen, C., Feng, S., Xing, Z., et al.: Gallery DC: design search and knowledge discovery through auto-created GUI component gallery. Proc. ACM Hum. Comput. Interact. **3**(CSCW), 1–22 (2019)
Chen, J., Chen, C., Xing, Z., et al.: Wireframe-based UI design search through image autoencoder. ACM Trans. Softw. Eng. Methodol. (TOSEM) **29**(3), 1–31 (2020)
Coyette, A., Vanderdonckt, J., Limbourg, Q.: Sketchixml: a design tool for informal user interface rapid prototyping. In: International Workshop on Rapid Integration of Software Engineering Techniques, pp. 160–176. Springer, Berlin (2006)
Coyette, A., Kieffer, S., Vanderdonckt, J.: Multi-fidelity prototyping of user interfaces. In: IFIP Conference on Human–Computer Interaction, pp. 150–164. Springer, Berlin (2007)
Deka, B., Huang, Z., Kumar, R.: Erica: interaction mining mobile apps. In: Proceedings of the 29th Annual Symposium on User Interface Software and Technology, pp. 767–776 (2016)
Deka, B., Huang, Z., Franzen, C., et al.: Rico: a mobile app dataset for building data-driven design applications. In: Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, pp. 845–854 (2017)
Devlin, J., Chang, M.W., Lee, K., et al.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
Difallah, D., Filatova, E., Ipeirotis, P.: Demographics and dynamics of mechanical turk workers. In: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, pp. 135–143 (2018)
Faranello, S.: Balsamiq Wireframes Quickstart Guide. Packt Publishing Ltd (2012)
Figma: Figma. https://www.figma.com/ (2016). Accessed 21 March 2021
Galke, L., Saleh, A., Scherp, A.: Word embeddings for practical information retrieval. INFORMATIK 2017 (2017)
Gu, X., Zhang, H., Kim, S.: Deep code search. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), pp. 933–944. IEEE (2018)
Han, S., Wang, X., Bendersky, M., et al.: Learning-to-rank with BERT in TF-ranking. arXiv preprint arXiv:2004.08476 (2020)
Huang, F., Canny, J.F., Nichols, J.: Swire: sketch-based user interface retrieval. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, pp. 1–10 (2019)
Husain, H., Wu, H.H., Gazit, T., et al.: Codesearchnet challenge: evaluating the state of semantic code search. arXiv preprint arXiv:1909.09436 (2019)
Jones, K.S., Van Rijsbergen, C.J.: Information retrieval test collections. J. Doc. (1976)

Kolthoff, K., Bartelt, C., Ponzetto, S.P.: Gui2wire: rapid wireframing with a mined and large-scale GUI repository using natural language requirements. In: 35th IEEE/ACM International Conference on Automated Software Engineering (ASE'20). ACM (2020)
Kolthoff, K., Bartelt, C., Ponzetto, S.P.: Automated retrieval of graphical user interface prototypes from natural language requirements. In: International Conference on Applications of Natural Language to Information Systems, pp. 376–384. Springer, Berlin (2021)
Konva.js: Konva.js—html5 2d canvas js library for desktop and mobile applications. https://konvajs.org/ (2015). Accessed 23 March 2020
Kraemer, H.C., Kupfer, D.J., Clarke, D.E., et al.: DSM-5: how reliable is reliable enough? Am. J. Psychiatry **169**(1), 13–15 (2012)
Krippendorff, K.: Computing Krippendorff's alpha-reliability (2011)
Landay, J.A., Myers, B.A.: Interactive sketching for the early stages of user interface design. Tech. rep., Carnegie-Mellon Univ Pittsburgh PA Dept of Computer Science (1994)
Landis, J.R., Koch, G.G.: The measurement of observer agreement for categorical data. Biometrics 159–174 (1977)
Lee, C., Kim, S., Han, D., et al.: uicomp: A GUI design assistant with real-time, multi-faceted feedback. arXiv preprint arXiv:2001.05684 (2020)
Leiva, L.A., Hota, A., Oulasvirta, A.: Enrico: a dataset for topic modeling of mobile UI designs. In: 22nd International Conference on Human–Computer Interaction with Mobile Devices and Services (MobileHCI'20 Extended Abstracts) (2020)
Li, T.J.J., Popowski, L., Mitchell, T.M., et al.: Screen2vec: semantic embedding of GUI screens and GUI components. arXiv preprint arXiv:2101.11103 (2021)
Lo, R., Webby, R., Jeffery, R.: Sizing and estimating the coding and unit testing effort for GUI systems. In: Proceedings of the 3rd International Software Metrics Symposium, pp. 166–173. IEEE (1996)
Lucene: Apache lucene. https://lucene.apache.org/ (2011). Accessed 31 Aug 2022
Lv, F., Zhang, H., Lou, Jg., et al.: Codehow: effective code search based on API understanding and extended Boolean model (e). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 260–270. IEEE (2015)
Macbeth, G., Razumiejczyk, E., Ledesma, R.D.: Cliff's delta calculator: a non-parametric effect size program for two groups of observations. Universitas Psychologica **10**(2), 545–555 (2011)
Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
McMillan, C., Grechanik, M., Poshyvanyk, D., et al.: Exemplar: a source code search engine for finding highly relevant applications. IEEE Trans. Softw. Eng. **38**(5), 1069–1087 (2011)
Mikolov, T., Sutskever, I., Chen, K., et al.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, pp. 3111–3119 (2013)
Mockplus: Mockplus. https://www.mockplus.com/ (2014). Accessed 21 March 2021
Moran, K., Bernal-Cárdenas, C., Curcio, M., et al.: Machine learning-based prototyping of graphical user interfaces for mobile apps. arXiv preprint arXiv:1802.02312 (2018)
Mukasa, K.S., Kaindl, H.: An integration of requirements and user interface specifications. In: 2008 16th IEEE International Requirements Engineering Conference, pp. 327–328. IEEE (2008)
Paolacci, G., Chandler, J., Ipeirotis, P.G.: Running experiments on amazon mechanical turk. Judgm. Decis. Mak. **5**(5), 411–419 (2010)
Peer, E., Vosgerau, J., Acquisti, A.: Reputation as a sufficient condition for data quality on amazon mechanical turk. Behav. Res. Methods **46**(4), 1023–1031 (2014)
Phan, N., Bailey, P., Wilkinson, R.: Understanding the relationship of information need specificity to search query length. In: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 709–710 (2007)
Rashtchian, C., Young, P., Hodosh, M., et al.: Collecting image annotations using amazon's mechanical turk. In: Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk, pp 139–147 (2010)
Ravid, A., Berry, D.M.: A method for extracting and stating software requirements that a user interface prototype contains. Requir. Eng. **5**(4), 225–241 (2000)
RaWi Prototype: Rawi—rapid prototyping website. http://rawi-prototyping.com/ (2022). Accessed 12 April 2021
RaWi Repository: Rawi—rapid prototyping github repository. https://github.com/RaWi-Protoyping/RaWi (2022). Accessed 12 April 2021

Reimers, N., Gurevych, I.: Sentence-BERT: sentence embeddings using siamese BERT-networks. arXiv preprint arXiv:1908.10084 (2019)
Ritchie, D., Kejriwal, A.A., Klemmer, S.R.: d. tour: Style-based exploration of design example galleries. In: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, pp. 165–174 (2011)
Robertson, S.E., Walker, S., Jones, S., et al.: Okapi at trec-3. Nist Special Publication Sp **109**, 109 (1995)
Ross, J., Irani, L., Silberman, M.S., et al.: Who are the crowdworkers? shifting demographics in mechanical turk. In: CHI'10 Extended Abstracts on Human Factors in Computing Systems, pp. 2863–2872 (2010)
Rudd, J., Stern, K., Isensee, S.: Low vs. high-fidelity prototyping debate. Interactions **3**(1), 76–85 (1996)
Segura, V.C., Barbosa, S.D., Simões, F.P.: Uiskei: a sketch-based prototyping tool for defining and evaluating user interface behavior. In: Proceedings of the International Working Conference on Advanced Visual Interfaces. pp. 18–25. ACM (2012)
Shuyo, N.: Language detection library for java. Retrieved Jul 7:2016 (2010)
Sketch: Sketch. https://www.sketch.com/ (2019). Accessed 21 March 2021
Smith, R.: An overview of the tesseract OCR engine. In: Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), pp. 629–633. IEEE (2007)
Suleri, S., Hajimiri, Y., Jarke, M.: Impact of using UI design patterns on the workload of rapid prototyping of smartphone applications: an experimental study. In: 22nd International Conference on Human–Computer Interaction with Mobile Devices and Services, pp. 1–5 (2020)
Teufel, S.: An overview of evaluation methods in TREC ad hoc information retrieval and TREC question answering. Evaluation of Text and Speech Systems, pp 163–186 (2007)
Wang, B., Li, G., Zhou, X., et al.: Screen2words: Automatic mobile UI summarization with multimodal learning. In: The 34th Annual ACM Symposium on User Interface Software and Technology, pp. 498–510 (2021)
Windsor, P., Storrs, G.: Prototyping user interfaces. In: IEE Colloquium on Software Prototyping and Evolutionary Development, pp. 4–1. IET (1992)
Woolson, R.: Wilcoxon Signed-Rank Test. Wiley Encyclopedia of Clinical Trials, pp 1–3 (2007)
Young, T., Hazarika, D., Poria, S., et al.: Recent trends in deep learning based natural language processing. IEEE Comput. Intell. Mag. **13**(3), 55–75 (2018)
Zhang, H., Jain, A., Khandelwal, G., et al.: Bing developer assistant: improving developer productivity by recommending sample code. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 956–961 (2016)