2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)

Automating App Review Response Generation

Cuiyun Gao[†] Jichuan Zeng[†] Xin Xia[‡] David Lo[§] Michael R. Lyu[†] Irwin King[†]
[†]Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China
[‡]Faculty of Information Technology, Monash University, Australia
[§]School of Information Systems, Singapore Management University, Singapore
{cygao,jczeng,lyu,king}@cse.cuhk.edu.hk xin.xia@monash.edu davidlo@smu.edu.sg

***Abstract*—Previous studies showed that replying to a user review usually has a positive effect on the rating that is given by the user to the app. For example, Hassan et al. found that responding to a review increases the chances of a user updating their given rating by up to six times compared to not responding. To alleviate the labor burden in replying to the bulk of user reviews, developers usually adopt a template-based strategy where the templates can express appreciation for using the app or mention the company email address for users to follow up. However, reading a large number of user reviews every day is not an easy task for developers. Thus, there is a need for more automation to help developers respond to user reviews.**

Addressing the aforementioned need, in this work we propose a novel approach RRGen that automatically generates review responses by learning knowledge relations between reviews and their responses. RRGen explicitly incorporates review attributes, such as user rating and review length, and learns the relations between reviews and corresponding responses in a supervised way from the available training data. Experiments on 58 apps and 309,246 review-response pairs highlight that RRGen outperforms the baselines by at least 67.4% in terms of BLEU-4 (an accuracy measure that is widely used to evaluate dialogue response generation systems). Qualitative analysis also confirms the effectiveness of RRGen in generating relevant and accurate responses.

***Index Terms*—App reviews, response generation, neural machine translation.**

I. Introduction

Mobile apps are software applications designed to run on smartphones, tablets and other mobile devices. They already serve as an integral part of people's daily life, and continuously gain traction over the last few years. The apps are typically available from app stores, such as Apple's App Store and Google Play. These app stores allow users to express their opinions to apps by writing reviews and giving ratings. User experience determines if users will keep using an app or uninstall it, possibly posting favorable or unfavorable feedbacks. For example, a survey in 2015 [1] reported that 65% users chose to leave a rating or review after a negative experience, and only 15% users would consider downloading an app with a 2-star rating. To compete with the bulk of the apps offering similar functionalities, ensuring good user experience is crucial for app developers.

App reviews act as one direct communication channel between developers and users, delivering users' instant experience after their interactions with apps. Analysis on app reviews can assist developers in discovering in a timely manner important app issues, such as bugs to fix or requested features, for app maintenance and development [2], [3]. Currently, both Apple's App Store and Google Play provide a review response system for developers to manually respond to a review, after which the corresponding user who posted the review will be notified and have the option to update their reviews [4], [5]. In the response, developers can talk about the roadmap about users' proposed feature requests, explain the usage of app functionalities, or just thank users for their shared opinions.

Empirical studies [6]–[9] that analyze the interactions between users and developers demonstrate that responding to user feedback in a timely and accurate manner can (1) enhance app development and (2) improve user experience. Specifically, Nayebi et al. [9] automatically summarized user requests which was proven to shorten the cycle between issue escalation and developers' fix. McIlroy et al. [7] observed that users change their rating 38.7% of the time following a developer response. Hassan et al. [8] found that developers of 34.1% of the apps they analyzed respond to at least one review, and also confirmed the positive effect of the responses on rating change. For example, they discovered that the number of users who increases their ratings after receiving a response are six times more than those who receive no response. App developers can also solve 34% of the reported issues without deploying an update. In spite of the benefits of the review-response mechanism, due to the large and ever-increasing number of reviews received daily, many reviews still did not receive timely response [4], [8]. This highlights the necessity and importance of automatic response generation, which is the focus of our work.

Dialogue generation has been extensively studied in the natural language processing field [10]–[12], for facilitating social conversations, e.g., the Microsoft XiaoIce chatbot [13]. Such work is generally grounded in the basic RNN Encoder-Decoder model (or Neural Machine Translation model, abbreviated as NMT) [14], [15], where the context and corresponding response are regarded as source and target sequences respectively. The RNN Encoder-Decoder model is an end-to-end learning approach for automated translation. It has been applied to a number of software engineering tasks, such as producing a sequence of APIs given a natural language query [16], parsing natural language into machine interpretable sequences (e.g., database queries) [17], generating commit messages according to code changes [18], [19], and inferring variable types based on contextual code snippets [20]. However, the applicability of the NMT model for app review

978-1-7281-2508-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ASE.2019.00025

response generation has not been studied. To fill in this gap, we explore the usability of the NMT model in the app review-response dialogue scenario here, by regarding user reviews and the corresponding replies as the source and target sequences respectively.

Directly applying the NMT model to app dialogue generation may not be appropriate, since the app review-response dialogues and social conversations are different in many ways. First, the purpose of app dialogues is to further understand users' complaints or solve user requests, while social conversations are mainly for entertainment purpose. This implies that app reviews require more accurate and clearer response [4]. Second, users' sentiment expressed in reviews should be precisely identified. Although reviews contain the information of star ratings, the ratings and actual emotions may not be totally consistent [21], [22]. For example, one user may write positive feedback like "*Great*", but only give one-star rating. Third, app reviews are generally short in length and usually with only one round of dialogue. According to Hassen et al. [8], 97.5% of the app dialogues end after one iteration. Such limited context increases the difficulty of generating a concise response.

In this paper, we propose an improved NMT model, named RRGen, for accurate **R**eview **R**esponse **Gen**eration. We extend basic NMT by incorporating review-specific characteristics (e.g., star ratings and review lengths) to capture user's sentiment and complaint topics. To evaluate the effectiveness of our model, we collected 309,246 review-response pairs from 58 popular apps published on Google Play. For a comprehensive comparison, besides the basic NMT model, we also choose the state-of-the-art approach in commit message generation based on code changes [23], named NNGen, as one baseline model. Because NNGen adopts basic information retrieval technique which is commonly used in traditional dialogue generation tasks [24]–[26], and claims better performance than the basic NMT model. Our experimental results show that RRGen significantly outperforms the baseline models by 67.4%∼450% in terms of BLEU-4 score [27] (an accuracy measure that is widely used to evaluate dialogue response generation systems). Human evaluation done through a user study also indicates that RRGen can generate a more relevant and accurate response than NNGen. Besides reporting the promising results, we investigate the reason behind the superior performance of our model and the key constraints on automatic response generation.

The main contributions of our work are as follows:

- To our knowledge, we are the first to consider the problem of automatic review response generation, and propose a deep neural network technique for solving the problem. We propose a novel neural machine translation model, RRGen[1], to learn both topics and sentiments of reviews for a accurate response generation.
- The accuracy of RRGen is empirically evaluated using a corpus of more than 300 thousand real-life review-response pairs. A user study was also conducted to verify RRGen's effectiveness in generating reasonable reviews.

Paper structure. Section II illustrates the background of review-response system, and neural encoder-decoder model. Section III presents our proposed model for user review response generation. Section IV and Section V describe our experimental setup and the quantitative evaluation results. Section VI details the results of a human evaluation of our proposed model. Section VII discusses the advantages, limitation, and threats of our work. Related work and final remarks are discussed in Section VIII and Section IX, respectively.

II. Background

Our work adopts and augments advanced techniques from deep learning and neural machine translations [28]–[30]. In this section, we introduce the user-developer dialogue and discuss the background of these techniques.

A. User-Developer Dialogue

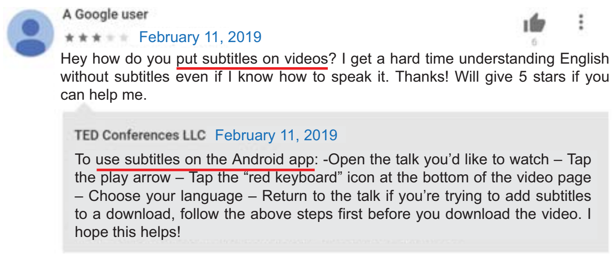
Figure 1 depicts an example of the user-developer dialogue of the TED app in Google Play. A user initiates the dialogue by posting a review, including a star rating, for an app. User reviews convey valuable information to developers, such as major bugs, feature requests, and simple complaints or praise about the experience [31]. As encouraged by the App Store [4], responding to feedback in a timely and consistent manner can improve user experience and an app's ranking. For example, the review in Fig. 1 was complaining about the unclear functionality usage related to adding "*video subtitles*". The TED developer then responded with detailed steps for putting subtitles, and later, the user changed the star rating to five.

Generally, developers could not reply to all app reviews due to their limited time and efforts, and also a large number of reviews. As studied by Hassan et al. [8], developers respond to 2.8% of the collected user reviews, and they tend to reply reviews with low ratings and long contents. The App Store also suggests developers to consider prioritizing reviews with the lowest star ratings or those mentioning technical issues for responding [4]. However, ranking reviews for developers' reply is out of the scope of this work, and the related studies can be found in [7], [8]. We focus on alleviating the manual labor in responding to feedback and aim at automating the process. Moreover, since 97.5% of the app dialogues end after one round [8], in this study, we concentrate on one iteration of user review reply.

B. RNN Encoder-Decoder Model

The RNN Encoder-Decoder [14] model is an effective and standard approach for neural machine translation and sequence-to-sequence (seq2seq) [32] prediction. In general, the RNN encoder-decoder models aim at generating a target sequence $\boldsymbol{y} = (y_1, y_2, ..., y_{T_y})$ given a source sequence $\boldsymbol{x} = (x_1, x_2, ..., x_{T_x})$, where T_x and T_y are sequence lengths of the source and target respectively. Fig. 2 illustrates an overall architecture of the RNN encoder-decoder model.

[1] available at: https://github.com/ReMine-Lab/RRGen


A Google user

★★★☆☆ February 11, 2019

Hey how do you put subtitles on videos? I get a hard time understanding English without subtitles even if I know how to speak it. Thanks! Will give 5 stars if you can help me.

TED Conferences LLC February 11, 2019

To use subtitles on the Android app: -Open the talk you'd like to watch – Tap the play arrow – Tap the "red keyboard" icon at the bottom of the video page – Choose your language – Return to the talk if you're trying to add subtitles to a download, follow the above steps first before you download the video. I hope this helps!

Fig. 1: Example of TED developer's response to one user review. The red underlines highlight some topical words of the dialogue.

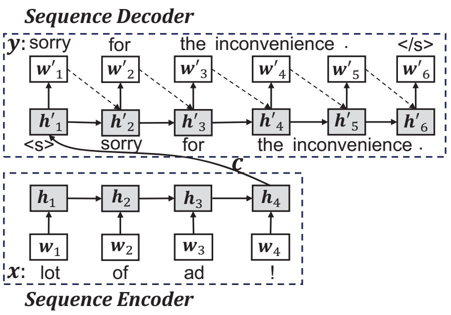


Fig. 2: An overall architecture of RNN encoder-decoder model.

To do so, an encoder first converts the source sequence $\boldsymbol{x}$ into a set of hidden vectors $\{\boldsymbol{h}_1, \boldsymbol{h}_2, ..., \boldsymbol{h}_{T_x}\}$, whose size varies regarding the source sequence length. The context representation $\boldsymbol{c}$ is generated using a Recurrent Neural Network (RNN) [33]. The encoder RNN reads the source sentences from the first token until the last one, where $\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{w}_t)$, and $\boldsymbol{c} = \boldsymbol{h}_{T_x}$. Here, $\boldsymbol{w}_t$ is the word embedding of the source token x_t, where word embeddings [34] are distributed representations of words in a continuous vector space, and trained with a text corpus. The f is a non-linear function that maps a the word embedding $\boldsymbol{w}_t$ into a hidden state $\boldsymbol{h}_t$ by considering the previous hidden state $\boldsymbol{h}_{t-1}$.

Then, the decoder, which is also implemented as an RNN, generates one word y_t at each time stamp t based on the hidden state $\boldsymbol{h}'_t$ as well as the previous predicted word y_{t-1}:

$$Pr(y_t|y_{t-1}, ..., y_1, \boldsymbol{c}) = g(\boldsymbol{h}'_t, y_{t-1}, \boldsymbol{c}), \tag{1}$$

where g is a non-linear mapping function, and the context vector $\boldsymbol{c}$ returned by the encoder is set as an initial hidden state, i.e., $\boldsymbol{h}'_1 = \boldsymbol{c}$. The decoder stops when generating the end-of-sequence word $<\backslash s>$.

The two RNN encoder-decoder models are jointly trained to maximize the conditional log-likelihood:

$$\mathcal{L}(\theta) = \max_{\theta} \frac{1}{N} \sum_{i=1}^{N} \log p_{\theta}(\boldsymbol{y}_i|\boldsymbol{x}_i), \tag{2}$$

where θ is the set of the model parameters (e.g., weights in the neural network) and each $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ is a (source sequence, target sequence) pair from the training set. The $p_\theta(\boldsymbol{y}_i|\boldsymbol{x}_i)$ denotes the likelihood of generating the i-th target sequence $\boldsymbol{y}_i$ given the source sequence $\boldsymbol{x}_i$ according to the model parameters θ. Through optimizing the loss function using optimization algorithms such as gradient descent, the optimum θ values can be estimated.

C. Attention Mechanism

A potential issue with the RNN encoder-decoder model is that a neural network needs to compress all the necessary information of a source sequence into a fixed-length vector. To alleviate this issue, Bahdanau et al. [28] proposed the attention mechanism to focus on relevant parts of the source sequence during decoding. We use the attention mechanism in our work because previous studies [35]–[37] prove that attention-based models can better capture the key information (e.g., topical or emotional tokens) in the source sequence. Fig. 3 shows a graphical illustration of the attentional RNN encoder-decoder model.

During decoding, besides the hidden state $\boldsymbol{h}'_t$ and previous predicted word y_{t-1}, an attention vector $\boldsymbol{a}_t$ is also involved for generating one word y_t at each time stamp t:

$$Pr(y_t|y_{t-1}, ..., y_1, \boldsymbol{c}) = g(\boldsymbol{h}'_t, y_{t-1}, \boldsymbol{c}, \boldsymbol{a}_t). \tag{3}$$

The attention vector $\boldsymbol{a}_t$ depends on the relevance between the hidden state $\boldsymbol{h}'_t$ and the encoded source sequence $(\boldsymbol{h}_1, ..., \boldsymbol{h}_{T_x})$:

$$\boldsymbol{a}_t = \sum_{j=1}^{T_x} \alpha_{tj} \boldsymbol{h}_j, \tag{4}$$

where T_x is the length of the source sequence, and the attention weight α_{tj} measures how helpful the j-th hidden state of the source sequence $\boldsymbol{h}_j$ is in predicting next word y_t with respect to the previous hidden state $\boldsymbol{h}'_{t-1}$. In this way, the decoder decides parts of the source sentence to pay attention to.

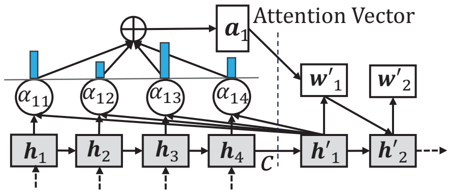
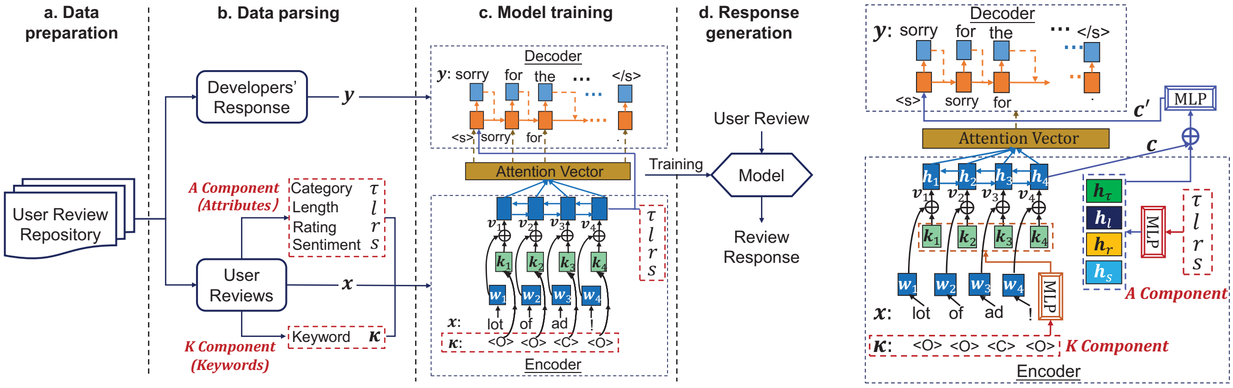


Fig. 3: Graphical illustration of the attentional RNN encoder-decoder model. The dotted line without arrow marks the division between the encoder (left) and decoder (right), and the dotted lines with arrows indicate that we simplify the RNN encoder-decoder [14] steps for clearness.

III. RRGen: App Review Response Generation

In this section, we present the design of RRGen that extends the basic attentional RNN Encoder-Decoder model for app review response generation. We regard user reviews as the source sequence and developers' response as the target sequence. Fig. 2 shows an example of the RNN Encoder-Decoder model for generating a sequence of tokens as a developer's response from a sequence of tokens that constitute a user review "*Lot of ad!*". For accurately capturing the topics



(a) Overall architecture of RRGen

(b) Detailed structure of RRGen

Fig. 4: Structure of the review response generative model.

and sentiment embedded in the input review sequence, we explicitly incorporate both high-level attributes (e.g., app category, review length, user rating, and sentiment) and keywords into the original RNN Encoder-Decoder model. We adopt the keywords provided by Di Sorbo et al. [3] which were manually curated to identify 12 topics (e.g., GUI, contents, pricing, etc.) commonly covered in user reviews. We refer to the high-level attributes and keywords extracted from a review as its A and K components, respectively.

Figure 4 (a) shows the overall architecture of our RRGen model. RRGen mainly consists of four stages: Data preparation, data parsing, model training, and response generation. We first collect app reviews and their responses from Google Play, and conduct preprocessing. The preprocessed data are parsed into a parallel corpus of user reviews and their corresponding responses, during which the two components of reviews are also extracted and processed. Based on the parallel corpus of app reviews and responses, we build and train a generative neural model with the two pieces of extracted information (high-level attributes and keywords) holistically considered. The major challenge during the training process lies in the effective consideration of both components of reviews for effective response generation. In the following, we will introduce the details of the RRGen model and the approach we propose to resolve the challenge.

A. *Component Incorporation*

Here, we elaborate on how we incorporate the two components, including high-level attributes (or A Component) and keywords (or K Component), into RRGen. The detailed structure of RRGen is displayed in Fig. 4 (b).

1) A Component: The A component contains four attributes of one user review: App category, review length, user rating, and sentiment. We choose app category considering that apps of different categories generally contain different functionalities, and major topics delivered by their reviews would be different. Review length is involved because it is an important index of whether the review is informative or not, i.e., longer reviews usually convey richer information [8], [38]. We take user rating into account since it can directly impact the response style of developers, e.g., expressing an apology for negative feedback or thanks for the positive feedback. As user ratings may not be consistent with the sentiment described by the reviews [39], we also regard the predicted actual user sentiment as one attribute.

Review attributes such as app category, review length, and user rating are easy to acquire. For predicting user sentiment, we exploit SentiStrength [40], a lexical sentiment extraction tool specialized in handling short and low-quality texts. We first divide review text into sentences, and then assigns a positive integer value (in the range [+1, +5]) and a negative integer value (within the range [-5, -1]) based on StentiStrength to each sentence because users may express both positive and negative sentiments in the same sentence. A higher absolute sentiment score indicates that the corresponding sentiment is stronger. Following Guzman and Maalej's work [39], when the sentence's negative score multiplied by 1.5 is less than the positive score, we assign the sentence a negative sentiment score; Otherwise, the sentence is assigned a positive sentiment score [39]. The sentiment of an entire review is computed based on the rounded average sentiment scores of all sentences in the review.

We denote the app category, review length, user rating, and sentiment score of the source sequence $\boldsymbol{x}$ as τ, l, r, and s, respectively. To incorporate these attributes into RRGen, we first represent the attribute values into continuous vectors via multilayer perceptions (MLPs), i.e., the conventional fully connected layer [41]. We call the vector representations of the attributes as attribute embeddings. The embedding of app category τ is defined as:

$$\boldsymbol{h}_\tau = \tanh(\boldsymbol{W}^\Gamma \, \mathrm{Emb}(\tau)), \forall \tau = 1, 2, ..., N_\Gamma, \qquad (5)$$

where $\boldsymbol{W}^\Gamma$ is the matrix of trainable parameters in the MLP, and $\boldsymbol{h}_\tau, g = 1, ..., N_\Gamma$ are the embedding vectors of all individual categories. $\mathrm{Emb}(\tau) \in \mathbb{R}^{N_\Gamma}$ is the vector representation of τ, and $\mathrm{Emb}(\cdot)$ indicates one general embedding layer to obtain the latent features of τ. Similarly, we obtain the embedding vectors for user rating r and sentiment score s:

$$\boldsymbol{h}_r = \tanh(\boldsymbol{W}^R \operatorname{Emb}(r)), \forall r = 1, 2, ..., N_R, \tag{6}$$

$$\boldsymbol{h}_s = \tanh(\boldsymbol{W}^S \operatorname{Emb}(s)), \forall s = 1, 2, ..., N_S, \tag{7}$$

where $\boldsymbol{h}_r$ and $\boldsymbol{h}_s$ are embeddings for the attribute values r and s, respectively. For review length l, we convert the continuous variable into its categorical form l' using the *pandas* package[2] before feeding into MLP.

$$\boldsymbol{h}_l = \tanh(\boldsymbol{W}^L \operatorname{Emb}(l'), \forall l' = 1, 2, ..., N_L. \tag{8}$$

We integrate the embedded attribute values at review level by concatenating together with the last hidden state $\boldsymbol{c}$ of the encoder, i.e.,

$$\boldsymbol{c}' = \tanh(\boldsymbol{W}^H [\boldsymbol{c}; \boldsymbol{h}_\tau; \boldsymbol{h}_l; \boldsymbol{h}_r; \boldsymbol{h}_s]), \tag{9}$$

where $[\boldsymbol{a}; \boldsymbol{b}]$ is the concatenation of these two vectors. $\boldsymbol{W}^H$ is the matrix of trainable parameters in the MLP, and H is the number of hidden units. The output vector $\boldsymbol{c}'$ indicates the final hidden state (or context vector) of the encoder. For simplicity, we assume that the dimensions of all attribute embeddings, i.e., $\boldsymbol{h}_\tau$, $\boldsymbol{h}_l$, $\boldsymbol{h}_r$, and $\boldsymbol{h}_s$, are the same.

2) K Component: K component specifically refers to keywords in the input review sequence, since the keywords generally relate to the review topic or sentiment, and are potentially helpful to learn which word to attend to during response generation.

TABLE I: One example of topic-keywords pair in the keyword dictionary provided by Di Sorbo et al. [3].

Topic	Keywords
GUI	screen, trajectory, button, white, background, interface, usability, tap, switch, icon, orientation, picture, show, list, category, cover, scroll, touch, clink, snap, underside, backside, witness, rotation, ui, gui,...

We adopt the keyword dictionary provided by Di Sorbo et al. [3]. Di Sorbo et al. summarize 12 topics[3] commonly covered by user reviews based on manual analysis, and build a keyword dictionary based on WordNet [42] to extract related words for each topic. One topic-keywords pair can be seen in Table I. Di Sorbo et al. utilized the dictionary to predict topics of user reviews and achieved >90% classification accuracy; this indicates the semantic representativeness of these keywords for each topic. This motivates us to use the keywords too in our work.

To explicitly integrate the keyword information into RRGen, we establish a keyword sequence $\boldsymbol{\kappa} = (\kappa_1, \kappa_2, ..., \kappa_{T_x})$ for each input review sequence $\boldsymbol{x}$. Specifically, for the token x_t in $\boldsymbol{x}$, we check the keyword dictionary to determine its subordinate topic, i.e., κ_t. For example, as shown in Fig. 4 (b), the keyword sequence corresponding to the source sequence "*lot of ad !*" is "<O><O><C><O>", where we denote the keyword symbol for the token "*ad*" as "<C>" since "*ad*" is one keyword for topic *contents*. The keyword symbols of non-topical words (e.g., "*of*") are labeled as "<O>". We finally integrate the embedded keyword sequence and the source sequence at token level via MLP:

$$\begin{gathered} \boldsymbol{k}_\iota = \tanh(\boldsymbol{W}^K \operatorname{Emb}(\kappa_\iota)), \forall \iota = 1, 2, ..., N_K \\ \boldsymbol{v}_t = \tanh(\boldsymbol{W}^V [\boldsymbol{k}_t; \boldsymbol{w}_t]) \end{gathered} \tag{10}$$

where $\boldsymbol{k}_\iota, \iota = 1, ..., N_K$ are the embedding vectors of all individual keyword symbols, $\boldsymbol{W}^K$ and $\boldsymbol{W}^V$ are the matrices of trainable parameters in the MLPs, and $\boldsymbol{v}_t$ is the keyword-enhanced embedding for the t-th token x_t in the source sequence. The dimension of $\boldsymbol{k}_\iota$ is similar to the attribute embeddings in the A component, e.g, $\boldsymbol{h}_\tau$.

B. Model Training and Testing

1) Training: We adopt the attention mechanism, described in Section II-C, for review response generation. The RNN has various implementations, we use bidirectional Gated Recurrent Units (GRUs) [14] which is a popular RNN encoder-decoder model and performs well in many tasks [43], [44]. All GRUs have 200 hidden units in each direction. Each attribute in the two components is encoded into an embedding with dimension at 90, i.e., the embedding size of $\boldsymbol{h}_\tau, \boldsymbol{h}_l, \boldsymbol{h}_r, \boldsymbol{h}_s$, and $\boldsymbol{k}_\iota$. Word embeddings are initiated with pre-trained 100-dimensional GloVe vectors [45]. We set the maximum sequence length at 200 and save the model every 200 batches. We discuss the details of parameter tuning in Section V-C. The training goal is cross-entropy minimization based on Equ. (11):

$$\mathcal{L}(\theta) = \max_\theta \frac{1}{N} \sum_{i=1}^{N} \log p_\theta(\boldsymbol{y}_i | \boldsymbol{x}_i, \tau, l, r, s, \boldsymbol{\kappa}_i), \tag{11}$$

where $\tau, l, r, s, \boldsymbol{\kappa}_i$ correspond to the app category, review length, user rating, sentiment score, and keyword sequence of the i-th source sequence $\boldsymbol{x}_i$, respectively. The whole model is trained using the minibatch Adam [46], a stochastic optimization approach and automatically adjusting the learning rate. We set the batch size (i.e., number of review instances per batch) as 32. For training the neural networks, we limit the source and target vocabulary to the top 10,000 words that are most frequently used in user reviews and developers' responses.

For implementation, we use PyTorch [47], an open-source deep learning framework. We train our model in a server with one Nvidia TITAN V GPU with 12GB memory. The training lasts ~80 hours with two epochs.

2) Testing: We evaluate on the test set when the trained model after one batch shows an improvement on the validation set regarding BLEU score [27]. We take the highest test score and corresponding generated response as the evaluation result. We use the same GPU as we used in training. The testing process took around 25 minutes.

[2] https://pandas.pydata.org/pandas-docs/stable/

[3] The 12 topics are app, GUI, contents, pricing, feature, improvement, updates/versions, resources, security, download, model, and company.

IV. Experimental Setup

A. Data Preparation

1) Data Collection: We select the subject apps for collecting the user-developer dialogues from Google Play based on app popularity. We focus on popular apps since they contain more reviews than unpopular apps [48], which should facilitate enough data for studying user-developer dialogues. We select the top 100 free apps in 2016 according to App Annie [49], an app analytics platform, as these apps were top apps two years prior to the start of our study. The decision was made to ensure the studied apps had enough reviews to collect and also avoid the influence of an app's price on developers' review response behavior [8]. We further remove the apps that are no longer available in Google Play on April 2018 and those with fewer than 100 user reviews, which leaves us with 72 apps that match our selection criteria.

For each selected app, we created a Google Play crawler to collect user-developer dialogues from Google Play, specifically including review title, review text, review post time, user name, rating, developer response time, and the text in the developer response. We run our crawler from April 2016 to April 2018. During that period, we collected 15,963,612 reviews for the 72 apps. We find that 58/72 apps and 318,973 collected reviews have received a response from the app developer. Table II describes the statistics of the 58 subject apps which belong to 15 app categories.

2) Data Preprocessing: Since app reviews are generally submitted via mobile terminals and written using limited keyboards, they contain massive noisy words, such as repetitive words and misspelled words [2]. We first convert all the words in the reviews and their response into lowercase, and adopt the method in [50] for lemmatization. We then replace all digits with "<digit>". We also detect email address and URL with regular expressions, and substitute them into "<email>" and "<url>" respectively. Besides, we build an app list containing all the app names, and a user list with all the user names. For the app names and user names mentioned in the dialogue corpus, we replace them with "<app>" and "<user>" respectively. We finally adopt the rule-based methods based on [50], [51] to rectify repetitive words and misspelled words. After removing empty review texts or review texts with only one single alphabet, we obtained 309,246 review-response pairs. We randomly split the dataset by 8:1:1, as the training, validation, and test sets, i.e., there are 279,792, 14,727, and 14,727 pairs in the training, validation, and test sets, respectively.

TABLE II: Mean and five-number summary of collected data for every studied app.

	Avg.	Min.	1st Qu.	Med.	3rd Qu.	Max.
#reviews per app	203,025	5,582	83,317	179,457	287,286	665,203
#reviews with responses per app	5,406	2	181	1,149	4,290	55,165

B. Similarity Measure - BLEU

BLEU [27] is a standard automatic metric for evaluating dialogue response generation systems. It analyzes the co-occurrences of n-grams in the ground truth $\boldsymbol{y}$ and the generated responses $\hat{\boldsymbol{y}}$, where n can be 1, 2, 3, or 4. BLEU-N, where N is the maximum length of n-grams considered, measures the proportion of co-occurrences of n consecutive tokens between the ground truth $\boldsymbol{y}$ and generated response $\hat{\boldsymbol{y}}$. The most commonly used version of BLEU uses $N = 4$ [19], [23], i.e., BLEU-4. Also, BLEU-4 is usually calculated at the corpus-level, which is demonstrated to be more correlated with human judgments than other evaluation metrics [52]. Thus, we use corpus-level BLEU-4 as our evaluation metric.

C. Baseline Approaches

We compare the performance of our model with a random selection approach, the basic attentional RNN encoder-decoder (NMT) model [28] (as introduced in Section II-C), and a state-of-the-art approach for code commit message generation [23], namely NNGen. In the following, we elaborate on the first and last baselines:

Random Selection: This is a strawman baseline. This baseline randomly picks a response in the training set and uses it as a response to a review in the test set.

NNGen: We choose NNGen as one comparing approach since it is demonstrated to perform better than the basic NMT model [19] in producing code commit message based on code changes. NNGen leverages the nearest neighbor (NN) algorithm to retrieve the most relevant developer response. Based on the training set and the new user review, NNGen first represents them as vectors in the form of "bags of words" [26], and then selects the top five training user reviews which present highest cosine similarities to the new review. After that, the BLEU-4 score between the new review and each of the top five training reviews is computed. NNGen finally regards the response of the training review with the highest BLEU-4 score as the result.

V. Evaluation Using An Automatic Metric

In this section, we conduct quantitative analysis to evaluate the effectiveness of RRGen. In particular, we intend to answer the following research questions.

RQ1: What is the accuracy of RRGen?
RQ2: What is the impact of different component attributes on the performance of RRGen?
RQ3: How accurate is RRGen under different parameter settings?

A. RQ1: What is the accuracy of RRGen?

The comparison results with baseline approaches are shown in Table III. We can see that our RRGen approach outperforms all the three baselines. Specifically, the result that random selection approach achieves the lowest BLEU-4 score (6.55), indicates that learning knowledge from existing review-response pairs can facilitate generating the response for a newly-arrived review. Also, we find that the NMT model performs better than the non-deep-learning-based NNGen model, which shows an increasing rate of 53.48% in terms of BLEU-4 score. This is opposite to the conclusion achieved by Liu et al. [23]. One possible reason is that the tasks between ours and Liu et al.'s [23] are different, i.e., Liu et al. aim at producing texts based on code, while we focus on generating texts for dialogues and modeling code is different from modeling dialogue texts [53], [54]. The higher BLEU-4 score of the proposed RRGen model than that of the NMT model explains that the response generated by the RRGen model is more similar to developers' response than the response generated by the NMT model. We then use Wilcoxon signed-rank test [55] for statistical significance test, and Cliff's Delta (or d) to measure the effect size [56]. The significance test result ($p - value < 0.01$) and large effect size on BLEU-4 scores ($d = 0.74$) of RRGen and NMT confirm the superiority of RRGen over NMT.

TABLE III: Comparison results with baseline approaches. The p_n indicates the n-gram precision when comparing the ground truth and generated responses. Statistical significance results are indicated with $^*(p - value < 0.01)$.

Approach	BLEU-4	p_1	p_2	p_3	p_4
Random	6.55	27.64	6.90	3.55	2.78
NNGen [23]	14.08	34.47	13.85	9.77	8.59
NMT [28]	21.61	40.55	20.75	16.78	15.47
RRGen	**36.17***	**53.24***	**35.83***	**31.73***	**30.04***

B. *RQ2: What is the impact of different component attributes on the performance of RRGen?*

To evaluate the effectiveness of different component attributes in response generation, we perform contrastive experiments in which only a single component attribute is added to the basic NMT model [28]. Table IV shows the results.

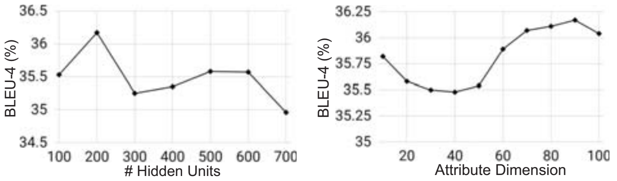
Unsurprisingly, the combination of all component attributes gives the highest improvements, and all the attributes are beneficial on their own. User sentiment gives the lowest improvement (+0.58 in terms of BLEU-4 score) comparing to the NMT model, while the app category yields highest improvement (+9.92 in terms of BLEU-4 score). Also, the result that user rating contributed more on the BLEU-4 score than user sentiment indicates that user ratings would be more helpful in review response generation. Moreover, the gain from different component attributes is not fully cumulative since the information encoded in these component attributes overlaps. For instance, both the user sentiment and user rating attributes encode the user emotion expressed by user reviews. Also, the keywords in the K component highlights the words belonging to the same topics, and such information may be already captured by the word embeddings [34].

TABLE IV: Contrastive experiments with individual component attributes.

Approach		BLEU-4	p_1	p_2	p_3	p_4
NMT [28]		21.61	40.55	20.75	16.78	15.47
A Component	+App Category	31.53	47.49	30.64	26.84	25.30
	+Review Length	24.22	41.96	22.30	18.16	16.76
	+Rating	26.90	46.19	26.06	21.69	20.12
	+Sentiment	22.19	40.42	20.95	16.99	15.69
K Component	+Keyword	24.34	43.41	23.66	19.27	17.74
RRGen		**36.17**	**53.24**	**35.83**	**31.73**	**30.04**

Dimension of Word Embedding	BLEU-4	p1	p2	p3	p4
50	35.80	52.78	35.09	30.67	28.91
100	**36.17**	**53.24**	**35.83**	**31.73**	**30.04**
200	35.61	51.77	33.77	29.63	28.00
300	35.54	51.25	33.18	29.09	27.39

(a) Different dimensions of word embedding.



(b) Different numbers of hidden units.

(c) Different dimensions of component attribute embedding.

Fig. 5: BLEU-4 scores of different parameter settings.

C. *RQ3: How accurate is RRGen under different parameter settings?*

We also quantitatively compare the accuracy of RRGen in different parameter settings. We analyze three parameters, that is, the dimension of word embeddings, the number of hidden units, and also the dimension of component attribute embeddings. We vary the values of these three parameters and evaluate their impact on the BLEU-4 scores.

Figure 5 shows the influence of different parameter settings on the test set. We choose the four different dimensions of word embeddings provided by GloVe [45], i.e., 50, 100, 200, and 300, and the result in Fig. 5 (a) indicates that the RRGen model achieves the best BLEU-4 score when the word embedding size equals to 100. For the number of hidden units, we can see that more hidden units may not be helpful for improving accuracy, as shown in Fig. 5 (b). RRGen generates the best result when we define the number of hidden units as 200. Fig. 5 (c) shows that the accuracy of RRGen also changes along with the variations of attribute embedding dimension. The optimum dimension of attribute embedding is around 90.

VI. Human Evaluation

In this section, we conduct a human evaluation to complement the evaluation in Section V that uses BLEU, since BLEU only measures the textual similarity between the generated responses and ground truth while the human study can evaluate users' general satisfaction on the responses.

A. *Survey Procedure*

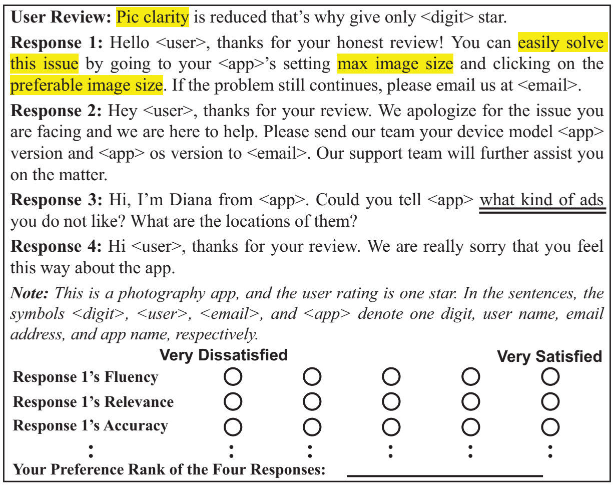
We conduct a human evaluation to evaluate the outputs of RRGen and compare RRGen with NMT and NNGen. We invite 20 participants, including 14 PhD students, two master students, one bachelor, and three senior researchers, all of whom are not co-authors and major in computer science. Among the participants, 15 of them have industrial experience in software development for at least a year, and eight of them have developed one or two mobile apps. Each participant is asked to read 25 user reviews, and assess the responses generated by NNGen, NMT, RRGen, and the app developers.

B. Survey Design

We randomly selected 100 review-response pairs in total, divide them evenly into four groups, and make a questionnaire for each group. We ensure that each review-response pair is evaluated by five different participants. In our questionnaire, each question presents the information of one review-response pair, i.e., its user review, the developer's response, its output from NNGen, and its responses generated by NMT and RRGen. The order of the responses from NNGen, NMT, RRGen, and official developers is randomly decided for each question.

Inspired by [11], [57], all the response types are evaluated considering three aspects - "*grammatical fluency*", "*relevance*", and "*accuracy*". We provided the following instructions at the beginning of each questionnaire to guide participants: The "*grammatical fluency*" (or readability) measures the degree of whether a text is easy to understand; The metric "*relevance*" relates to the extent of topical relevance between the user review and response; And the metric "*accuracy*" estimates the degree of the response accurately answering a user review.

All the three metrics are rated on a 1-5 scale (5 for fully satisfying the rating scheme, 1 for completely not satisfying the rating scheme, and 3 for the borderline cases), since a 5-point scale is widely used in prior software engineering studies [3], [23], [58]. Besides the three metrics, each participant is asked to rank responses generated by the three tools and those from developers based on their preference. The "*preference rank*" score is rated on a 1-4 scale (1 for the most preferred). Fig. 6 shows one question in our survey. Participants do not know which response is generated by which approach or whether it is written by developers, and they are asked to enter to score each response separately.



User Review: Pic clarity is reduced that's why give only <digit> star.

Response 1: Hello <user>, thanks for your honest review! You can easily solve this issue by going to your <app>'s setting max image size and clicking on the preferable image size. If the problem still continues, please email us at <email>.

Response 2: Hey <user>, thanks for your review. We apologize for the issue you are facing and we are here to help. Please send our team your device model <app> version and <app> os version to <email>. Our support team will further assist you on the matter.

Response 3: Hi, I'm Diana from <app>. Could you tell <app> what kind of ads you do not like? What are the locations of them?

Response 4: Hi <user>, thanks for your review. We are really sorry that you feel this way about the app.

***Note:** This is a photography app, and the user rating is one star. In the sentences, the symbols <digit>, <user>, <email>, and <app> denote one digit, user name, email address, and app name, respectively.*

	Very Dissatisfied				Very Satisfied
Response 1's Fluency	○	○	○	○	○
Response 1's Relevance	○	○	○	○	○
Response 1's Accuracy	○	○	○	○	○
:	:	:	:	:	:

Your Preference Rank of the Four Responses: ______________

Fig. 6: A question in our survey. Response 1, 2, 3, and 4 correspond to the developer's response, the outputs of our RRGen model, and the responses produced by NNGen and NMT, respectively. Participants do not know the order of the four types of response during the survey, and are asked to score the three metrics for each response type. The two-dot symbols indicate the simplified grading schemes of Response 2, 3, and 4. The words highlighted in yellow are topical words in the descriptions, and the double-underlined words mean they are topically irrelevant to the user review.

C. Results

We obtained 500 sets of scores from the human evaluation. Each set contains scores for the three metrics regarding the response of NNGen, NMT, RRGen, and official developers respectively, and also a ranking score of the four types of responses. The median time cost for one participant to complete his/her questionnaire is 0.94 hour, with an average value of 2.72 hours. We compute the agreement rate on the the preference ranks given by the participants, and find that 81% of the total 100 review-response pairs received at least three identical preference ranks from the participants. Specifically, 31%, 36%, and 14% were given the same preference ranks by three, four, and five participants respectively. This indicates that the participants achieved reasonable agreement on the performance of the generated responses.

Table V shows the results of human evaluation. Bold indicates top scores. As expected, we can see that the response from official developers is preferred over the three approaches' outputs, which can be observed given the example in Fig. 6. Specifically, the developers' response (Response 1) is more relevant to the user review and provides more accurate solution to the app issue (e.g., reduced picture clarity) complained by the user. In terms of grammatical fluency, however, the RRGen model does quite well, achieving scores that are rather close to those of developers' responses, as shown in Table V. In addition, we see that our RRGen model performs significantly better across all the metrics in comparison to the baseline approaches, which further indicates the effectiveness of RRGen in review response generation.

TABLE V: Human evaluation results for review response generation. Bold indicates top scores. Two-tailed t-test results are shown for our RRGen approach compared to NNGen and NMT (Statistical significance is indicated with $^{*}(p - value < 0.01)$.).

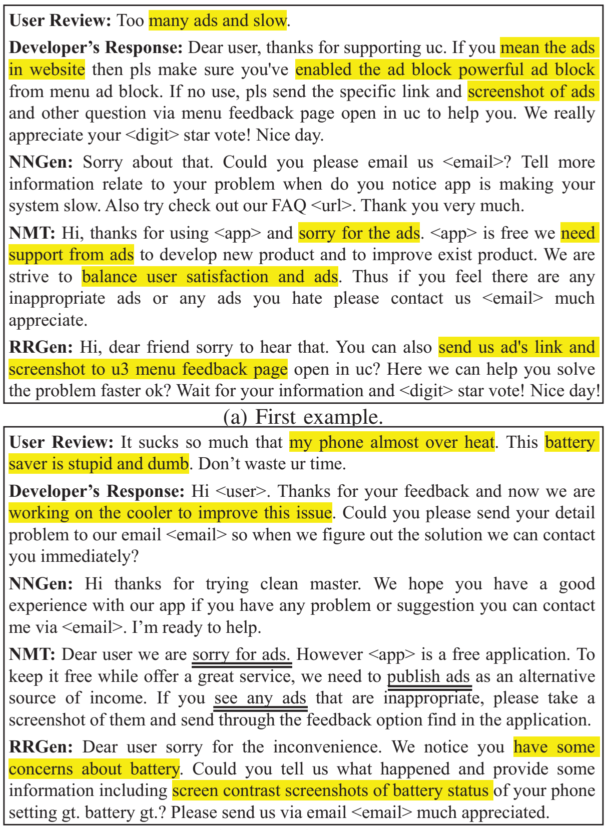
	Grammatical Fluency	Relevance	Accuracy	Preference Rank
NNGen [23]	4.520	3.160	3.104	3.339
NMT [28]	4.609	3.273	3.017	2.680
RRGen	4.626^{*}	3.536^{*}	3.458^{*}	2.244^{*}
Developer	**4.644**	**3.804**	**3.712**	**1.736**

VII. Discussion

A. Why does Our Model Work?

We have identified three advantages of RRGen that may explain its effectiveness in review response generation.

Observation 1: RRGen can better capture salient topics of user reviews. Unlike bag-of-words-based techniques, RRGen learns review and response representations with attentional deep learning. Characteristics of reviews, such as topical words and word orders, are naturally considered in these models [16]. Moreover, keywords that are indicative of review topics are explicitly incorporated into the deep learning model, which would be helpful to better recognize the semantics of review and response. For example, it can learn that the review



User Review: Too many ads and slow.

Developer's Response: Dear user, thanks for supporting uc. If you mean the ads in website then pls make sure you've enabled the ad block powerful ad block from menu ad block. If no use, pls send the specific link and screenshot of ads and other question via menu feedback page open in uc to help you. We really appreciate your <digit> star vote! Nice day.

NNGen: Sorry about that. Could you please email us <email>? Tell more information relate to your problem when do you notice app is making your system slow. Also try check out our FAQ <url>. Thank you very much.

NMT: Hi, thanks for using <app> and sorry for the ads. <app> is free we need support from ads to develop new product and to improve exist product. We are strive to balance user satisfaction and ads. Thus if you feel there are any inappropriate ads or any ads you hate please contact us <email> much appreciate.

RRGen: Hi, dear friend sorry to hear that. You can also send us ad's link and screenshot to u3 menu feedback page open in uc? Here we can help you solve the problem faster ok? Wait for your information and <digit> star vote! Nice day!

(a) First example.

User Review: It sucks so much that my phone almost over heat. This battery saver is stupid and dumb. Don't waste ur time.

Developer's Response: Hi <user>. Thanks for your feedback and now we are working on the cooler to improve this issue. Could you please send your detail problem to our email <email> so when we figure out the solution we can contact you immediately?

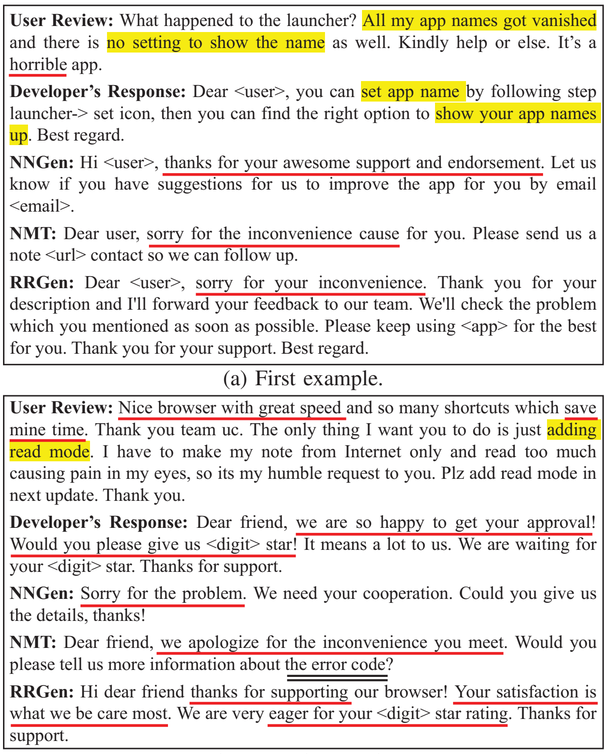
NNGen: Hi thanks for trying clean master. We hope you have a good experience with our app if you have any problem or suggestion you can contact me via <email>. I'm ready to help.

NMT: Dear user we are sorry for ads. However <app> is a free application. To keep it free while offer a great service, we need to publish ads as an alternative source of income. If you see any ads that are inappropriate, please take a screenshot of them and send through the feedback option find in the application.

RRGen: Dear user sorry for the inconvenience. We notice you have some concerns about battery. Could you tell us what happened and provide some information including screen contrast screenshots of battery status of your phone setting gt. battery gt.? Please send us via email <email> much appreciated.

(b) Second example.

Fig. 7: Two sample review-response pairs where RRGen can generate responses with more related topic. The meanings of the highlighted words and double-underlined words are the same as Fig. 6.



User Review: What happened to the launcher? All my app names got vanished and there is no setting to show the name as well. Kindly help or else. It's a horrible app.

Developer's Response: Dear <user>, you can set app name by following step launcher-> set icon, then you can find the right option to show your app names up. Best regard.

NNGen: Hi <user>, thanks for your awesome support and endorsement. Let us know if you have suggestions for us to improve the app for you by email <email>.

NMT: Dear user, sorry for the inconvenience cause for you. Please send us a note <url> contact so we can follow up.

RRGen: Dear <user>, sorry for your inconvenience. Thank you for your description and I'll forward your feedback to our team. We'll check the problem which you mentioned as soon as possible. Please keep using <app> for the best for you. Thank you for your support. Best regard.

(a) First example.

User Review: Nice browser with great speed and so many shortcuts which save mine time. Thank you team uc. The only thing I want you to do is just adding read mode. I have to make my note from Internet only and read too much causing pain in my eyes, so its my humble request to you. Plz add read mode in next update. Thank you.

Developer's Response: Dear friend, we are so happy to get your approval! Would you please give us <digit> star! It means a lot to us. We are waiting for your <digit> star. Thanks for support.

NNGen: Sorry for the problem. We need your cooperation. Could you give us the details, thanks!

NMT: Dear friend, we apologize for the inconvenience you meet. Would you please tell us more information about the error code?

RRGen: Hi dear friend thanks for supporting our browser! Your satisfaction is what we be care most. We are very eager for your <digit> star rating. Thanks for support.

(b) Second example.

Fig. 8: Two sample review-response pairs where RRGen can generate responses with more accurate sentiment. The words underlined in red are indicative of the description sentiments, and the meanings of the highlighted and double-underlined words are the same as Fig. 6.

"*Too many ads and slow*" is talking about the ad issue, and generate response related to the in-app ads, as shown in Fig. 7 (a). In the example in Fig. 7 (b), RRGen can well learn that the review is discussing about the battery issue, while NMT infers the topic wrongly. For the bag-of-words approach, NNGen, it may be easily confused by non-topical words. For the example in Fig. 6, NNGen (i.e., Response 3) focuses more on the words "*give*", "*<digit>*", and "*star*", and selects the closest review "*If it has no advertising, I will give <digit> star*" which has totally different topics comparing to the given review.

Observation 2: RRGen can generate responses with more accurate sentiment. User sentiment can be explicitly (e.g., the "*horrible*" word in Fig. 8 (a)) or implicitly (e.g., the "*slow*" word in Fig. 7 (a)) reflected in user reviews. For the bag-of-words approach, the effect of sentiment words may be weakened by other words since their occurrence frequencies are similar. As can be seen in Fig. 8 (a), NNGen fails to infer the negative sentiment expressed by the review, and considers it as an endorsement message; while RRGen can accurately capture the negative information embedded in the review. Another example can be found in Fig. 8 (b), where both NNGen and NMT do not recognize that the positive sentiment of the given review. Without review attributes such as user ratings involved, NMT also fails to ask the user to increase his/her given rating.

Observation 3: RRGen can effectively capture knowledge relations between reviews and their corresponding responses. RRGen learns the correspondence between reviews and response mainly through the high-dimensional hidden units and attention layer. The topical words in reviews tend to produce hidden states of semantically similar words in the RNN decoder. Fig. 9 visualizes the latent alignment over the user review to help generate the response based on the attention weights α_{tj} from Equ. (4). Each column indicates the weight distribution over the user review for generating each word. From this we can see which words in the user review were considered more important when generating the target word in the response. We can observe the obvious correlations between the word "*save*" (in the review) and "*save*" (in the response), "*hd*" (in the review) and "*max*" (in the response), and "*pixel*" (in the review) and "*image*" (in the response), as shown in Fig. 9. This illustrates that RRGen is able to build implicit relations between the topical words in reviews and corresponding responses, which can help generate relevant and accurate response given a review.

B. Post-Processing Steps

RRGen generates responses with placeholders, e.g., "<email>", "<url>", etc. Moreover, RRGen may not generate perfect responses and developers may want to verify RRGen responses for some more "sensitive" cases. To partially address the above-mentioned limitations, we propose several post-processing steps. First, we build a placeholder-value dictionary for automatically replacing placeholders (e.g., "<url>") with corresponding values (e.g., "https://www.facebook.com/groups/vivavideoapp/") for each app. Second, we design a quality assurance filter to automatically detect the generated responses that *require further check*.

The placeholder-value dictionary for each app is saved during preprocessing, and for simplicity, only the most common value for each placeholder is saved. We define a generated response requiring further check based on its token length l, the overlapped keyword ratio ω with the corresponding review, and also the review rating r. Specifically, we define responses that satisfy the following constraint, i.e., $\omega < 0.05$ or $(l < 38 \, and \, r \leq 2)$ to require further check. The thresholds are determined as follows: 0.05 is determined by following the keyword overlapping threshold in [3], 38 is the first quartile of response token lengths in the whole dataset, and the constraint for review rating is set as such as reviews with lower ratings (e.g., 1, 2) tend to express users' strong dissatisfaction with certain aspects of apps [38], [59].

We evaluate our solution after the above mentioned post-processing strategy using a similar experiment setting used to produce results presented in Section V-A. We find that the BLEU-4 score is 34.63. It is only slightly lower than the BLEU-4 score (36.17) reported in our earlier experiment using ground truths with placeholders rather than actual values.

C. Limitations

Although our proposed RRGen model aims at producing accurate responses to user reviews, not all the reviews require responses, some reviews require carefully crafted replies, and some other reviews can be delegated to an automated bot. We have tried to address this issue partially by adding some preliminary post-processing steps (see Section VII-B).

Admittedly, our post-processing steps are not perfect. First, our preliminary post-processing steps may generate responses with inappropriate values due to the coarsely-defined placeholder-value dictionary. This issue can be improved by creating a context-sensitive dictionary for each app. Also, our simple rule-based detection of responses that require further check can be improved further. For this, we can learn the thresholds of the rule conditions or design new detection criteria. We leave the design, implementation, and evaluation of a full-fledged system that can route reviews to *do not respond*, *require human careful response*, and *can be responded by an automated bot* queues for future work. As our work is the first to automate app review generation, although it is not perfect, it opens up way for future research to continue our study and improve it further.

D. Threats to Validity

One of the threats to validity is about the limited number of studied apps. We studied developer responses for reviews of free apps only. One of the main reasons for removing non-free apps is that the pricing of an app is likely to impact developers' response behavior [8]. Also, we only consider Google Play apps in this work, because Apple's App Store started to support review response from 2017 while the feature has been standard in Google Play since 2013 [60]. Although our study is based on apps from various categories and large numbers of review-response pairs, future work can be extended to multiple app stores and paid apps.

The second threat to validity is about the component attributes incorporated into our proposed model. Although we involve both high-level attributes and keywords, some other characteristics such as review title length and post date, which would be helpful for response generation, are not considered. Besides, the review sentiment predicted by SentiStrength [40] might not be reliable [61], and could influence the generated response. However, accurate sentiment prediction based on reviews is out of the scope of this paper, and the effectiveness of StentiStrength in detecting user sentiment about app features has been demonstrated in [39]. In the future, we will explore the impact of more review characteristics on automatic review response generation.

Another threat to validity is about manual inspection in Section VI. The results of the human evaluation are impacted by the experience of the participants and their intuition of the evaluation metrics. To reduce the errors in the manual analysis, we ensure that each review-response pair was evaluated by five different participants. As our participants are mainly students, they may not be representative of (CRM) professionals who are likely to benefit from our tools in practice [62], [63]. We try to mitigate this threat by inviting the students with at least one year of software development experience. In addition, we randomly disrupt the order of the three types of response for each question, so that the results are not influenced by participants' prior knowledge about the response orders.

VIII. Related Work

A. User Review Mining

Identifying the complaint topics expressed by user reviews is the basis for user review mining [64]–[66]. Iacob et al. [67] manually label 3,278 reviews, and discover the most recurring issues users report through reviews. To alleviate the labor in manual labeling, many studies focus on automating the process. For example, Iacob and Harrison [68] design MARA for retrieving app feature requests based on linguistic rules. Maalej and Nabil [31] adopt probabilistic techniques to classify reviews. Di Sorbo et al. [3] separately categorize user intentions and topics delivered by app reviews. Understanding user sentiment about specific app aspects is another typical direction of review mining. Guzman and Maalej [39] use topic modeling approach and StentiStrength [40] (a lexical sentiment extraction tool) to predict sentiment of app features.

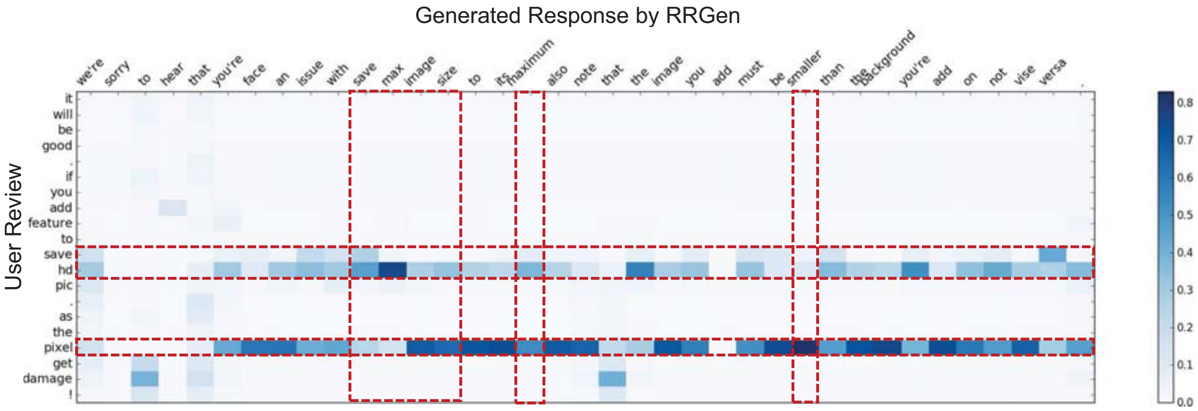


Fig. 9: A heatmap representing the alignment between the user review (left) and generated response by RRGen (top). The columns represent the distribution over the user review after generating each word. Each pixel shows the weight α_{tj} of the annotation of the j-th source word for the t-th target word (see Equ. (4)). A higher attention weight (indicated in darker color) manifests a stronger correlation between the target word and source word. The red dotted rectangles highlight partial topical words in corresponding descriptions.

Gu and Kim [69] propose SUR-Miner to exploit grammatical structures for aspect-opinion identification. More research of mobile review analysis can be found in [70]. Different from these existing review analysis research, we contribute to facilitating the bidirectional dialogue between users and developers instead of analyzing only the feedback from user side.

B. Analysis of User-Developer Dialogues in App Stores

Oh et al. [6] conduct a survey on 100 smartphone users to understand how developers and users interact. They find that most users (69%) tend to take a passive action such as uninstalling apps, and the main reason for such behavior is that these users think that their inquiries (e.g., user reviews) would take long time to be responded or receive no response. McIlroy et al. [7] analyze reviews of 10,000+ free Google Play apps and find that 13.8% of the apps respond to at least one review. They also observe that users would change their ratings 38.7% of the time following a response. Such positive impact of developers' response is also confirmed by Hassan et al. [8]. Although these studies do highlight the importance of responding to user reviews, they do not provide an explicit method to alleviate the burden in the responding process, which is the focus of this work.

C. Short Text Dialogue Analysis

Short text dialogue analysis is one popular topic in the field of natural language processing, in which given a message from human, the computer returns a reasonable response to the message [24], [71]. Short text dialogue can be formalized as a search or a generation problem. The former formalization is based on a knowledge base consisting of a large number of message-response pairs. Information retrieval techniques [26] are generally utilized to select the most suitable response to the current message from the knowledge base. The major bottleneck for search-based approaches is the creation of the knowledge base [72]. Ritter et al. [73] and Vinyals and Le [74] are the first to treat generation of conversational dialog as a data-driven statistical machine translation (SMT) [75] problem. Their results show that the machine translation-based approach works better than one IR approach, vector space model (VSM) [76], in terms of BLEU score [27]. However, generation-based approaches cannot guarantee that the response is a legitimate natural language text. In this work, we propose to integrate app reviews' unique characteristics for accurate response generation.

IX. Conclusion and Future Work

Replying to user reviews can help app developers create a better user experience and improve apps' ratings. Due to the large numbers of reviews received for popular apps each day, automating the review response process is useful for app developers. In this work, we propose a novel approach named RRGen by explicitly incorporating review attributes and occurrences of specific keywords into the basic NMT model. Analysis using automated metric and human evaluation shows that our proposed model outperforms baseline approaches. In future, we will conduct evaluation using a larger dataset and deploy the model with our industry partners.

Acknowledgement

The work described in this paper was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14210717 and No. CUHK 14208815 of the General Research Fund), and Microsoft Research Asia (2018 Microsoft Research Asia Collaborative Research Award).

References

[1] "Survey on user ratings and reviews," https://www.apptentive.com/blog/2015/05/05/app-store-ratings-reviews-guide/.

[2] C. Gao, J. Zeng, M. R. Lyu, and I. King, "Online app review analysis for identifying emerging issues," in *Proceedings of the 40th International Conference on Software Engineering (ICSE)*. ACM, 2018, pp. 48–58.

[3] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my app? summarizing app reviews for recommending software changes," in *Proceedings of the 24th SIGSOFT International Symposium on Foundations of Software Engineering (FSE).* ACM, 2016, pp. 499–510.

[4] "Ratings, reviews, and responses in app store," https://developer.apple.com/app-store/ratings-and-reviews/.

[5] "View and analyze your app's ratings and reviews," https://support.google.com/googleplay/android-developer/answer/138230?hl=en.

[6] J. Oh, D. Kim, U. Lee, J. Lee, and J. Song, "Facilitating developer-user interactions with mobile app review digests," in *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013, Extended Abstracts*, 2013, pp. 1809–1814.

[7] S. McIlroy, W. Shang, N. Ali, and A. E. Hassan, "Is it worth responding to reviews? studying the top free apps in google play," *IEEE Software*, vol. 34, no. 3, pp. 64–71, 2017.

[8] S. Hassan, C. Tantithamthavorn, C. Bezemer, and A. E. Hassan, "Studying the dialogue between users and developers of free apps in the google play store," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1275–1312, 2018.

[9] M. Nayebi, L. Dicke, R. Ittyipe, C. Carlson, and G. Ruhe, "Essmart way to manage user requests," *CoRR*, vol. abs/1808.03796, 2018.

[10] D. Wang, N. Jojic, C. Brockett, and E. Nyberg, "Steering output style and topic in neural response generation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, 2017, pp. 2140–2150.

[11] J. Li and X. Sun, "A syntactically constrained bidirectional-asynchronous approach for emotional conversation generation," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, 2018, pp. 678–683.

[12] A. Sordoni, M. Galley, M. Auli, C. Brockett, Y. Ji, M. Mitchell, J. Nie, J. Gao, and B. Dolan, "A neural network approach to context-sensitive generation of conversational responses," in *NAACL HLT 2015, The Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pp. 196–205.

[13] L. Zhou, J. Gao, D. Li, and H. Shum, "The design and implementation of xiaoice, an empathetic social chatbot," *CoRR*, vol. abs/1812.08989, 2018.

[14] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2014, pp. 1724–1734.

[15] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[16] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep API learning," in *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, 2016, pp. 631–642.

[17] L. Dong and M. Lapata, "Language to logical form with neural attention," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[18] S. Jiang and C. McMillan, "Towards automatic generation of short summaries of commits," in *Proceedings of the 25th International Conference on Program Comprehension, ICPC 2017, Buenos Aires, Argentina, May 22-23, 2017*, 2017, pp. 320–323.

[19] S. Jiang, A. Armaly, and C. McMillan, "Automatically generating commit messages from diffs using neural machine translation," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, 2017, pp. 135–146.

[20] V. J. Hellendoorn, C. Bird, E. T. Barr, and M. Allamanis, "Deep learning type inference," in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, 2018, pp. 152–162.

[21] M. R. Islam, "Numeric rating of apps on google play store by sentiment analysis on user reviews," in *2014 International Conference on Electrical Engineering and Information & Communication Technology*. IEEE, 2014, pp. 1–4.

[22] K. Sharma and K. Lin, "Review spam detector with rating consistency check," in *ACM Southeast Regional Conference 2013, ACM SE'13, Savannah, GA, USA, April 4-6, 2013*, 2013, pp. 34:1–34:6.

[23] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: how far are we?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, 2018, pp. 373–384.

[24] Z. Ji, Z. Lu, and H. Li, "An information retrieval approach to short text conversation," *CoRR*, vol. abs/1408.6988, 2014.

[25] Y. Song, C. Li, J. Nie, M. Zhang, D. Zhao, and R. Yan, "An ensemble of retrieval-based and generation-based human-computer conversation systems," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, 2018, pp. 4382–4388.

[26] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval.* Cambridge University Press, 2008.

[27] K. Papineni, S. Roukos, T. Ward, and W. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA.*, 2002, pp. 311–318.

[28] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014.

[29] R. Collobert and S. Bengio, "Links between perceptrons, mlps and svms," in *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, 2004.

[30] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, 2015, pp. 1412–1421.

[31] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? on automatically classifying app reviews," in *23rd IEEE International Requirements Engineering Conference, RE, Ottawa, ON, Canada, August 24-28, 2015*, 2015, pp. 116–125.

[32] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 2014, pp. 3104–3112.

[33] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, "Recurrent neural network based language model," in *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, 2010, pp. 1045–1048.

[34] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, 2013, pp. 3111–3119.

[35] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, 2015, pp. 379–389.

[36] Z. Lin, M. Feng, C. N. dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured self-attentive sentence embedding," *CoRR*, vol. abs/1703.03130, 2017.

[37] J. Zeng, J. Li, Y. Song, C. Gao, M. R. Lyu, and I. King, "Topic memory networks for short text classification," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, 2018, pp. 3120–3131.

[38] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang, "Ar-miner: mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th International Conference on Software Engineering (ICSE).* ACM, 2014, pp. 767–778.

[39] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *Proceedings of the 22nd International Conference on Requirements Engineering (RE).* IEEE, 2014, pp. 153–162.

[40] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas, "Sentiment in short strength detection informal text," *JASIST*, vol. 61, no. 12, pp. 2544–2558, 2010.

[41] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, 1989, pp. 762–767.

[42] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[43] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014.

[44] Z. Wu and S. King, "Investigating gated recurrent networks for speech synthesis," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, 2016, pp. 5140–5144.

[45] "Glove: Global vectors for word representation," https://nlp.stanford.edu/projects/glove/.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[47] "Pytorch," https://pytorch.org/.

[48] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: MSR for app stores," in *9th IEEE Working Conference of Mining Software Repositories, MSR, June 2-3, 2012, Zurich, Switzerland*, 2012, pp. 108–111.

[49] "App annie," https://www.appannie.com/.

[50] Y. Man, C. Gao, M. R. Lyu, and J. Jiang, "Experience report: Understanding cross-platform app issues from user reviews," in *27th IEEE International Symposium on Software Reliability Engineering, ISSRE 2016, Ottawa, ON, Canada, October 23-27, 2016*, 2016, pp. 138–149.

[51] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen, "Mining user opinions in mobile app reviews: A keyword-based approach (T)," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, 2015, pp. 749–759.

[52] C. Liu, R. Lowe, I. Serban, M. Noseworthy, L. Charlin, and J. Pineau, "How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, 2016, pp. 2122–2132.

[53] P. Yin and G. Neubig, "A syntactic neural model for general-purpose code generation," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, 2017, pp. 440–450.

[54] S. Liu, H. Chen, Z. Ren, Y. Feng, Q. Liu, and D. Yin, "Knowledge diffusion for neural dialogue generation," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, 2018, pp. 1489–1498.

[55] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[56] S. E. Ahmed, "Effect sizes for research: A broad application approach," *Technometrics*, vol. 48, no. 4, p. 573, 2006.

[57] X. Du and C. Cardie, "Harvesting paragraph-level question-answer pairs from wikipedia," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, 2018, pp. 1907–1917.

[58] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016*, 2016, pp. 165–176.

[59] C. Gao, H. Xu, J. Hu, and Y. Zhou, "Ar-tracker: Track the dynamics of mobile apps via user review mining," in *2015 IEEE Symposium on Service-Oriented System Engineering, SOSE 2015, San Francisco Bay, CA, USA, March 30 - April 3, 2015*, 2015, pp. 284–290.

[61] N. Novielli, D. Girardi, and F. Lanubile, "A benchmark study on sentiment analysis for software engineering research," in *Proceedings*

[60] "Developers can finally respond to app store reviews," https://techcrunch.com/2017/03/28/developers-can-finally-respond-to-app-store-reviews-heres-how-it-works/. *of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, 2018, pp. 364–375.

[62] I. Salman, A. T. Misirli, and N. J. Juzgado, "Are students representatives of professionals in software engineering experiments?" in *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, 2015, pp. 666–676.

[63] R. Feldt, T. Zimmermann, G. R. Bergersen, D. Falessi, A. Jedlitschka, N. Juristo, J. Münch, M. Oivo, P. Runeson, M. J. Shepperd, D. I. K. Sjøberg, and B. Turhan, "Four commentaries on the use of students and professionals in empirical software engineering experiments," *Empirical Software Engineering*, vol. 23, no. 6, pp. 3801–3820, 2018.

[64] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. Gall, F. Ferrucci, and A. D. Lucia, "Recommending and localizing change requests for mobile apps based on user reviews," in *IEEE/ACM 39th International Conference on Software Engineering (ICSE'17)*, 2017, pp. 106–117.

[65] G. Grano, A. Ciurumelea, S. Panichella, F. Palomba, and H. C. Gall, "Exploring the integration of user feedback in automated testing of android applications," in *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER'18)*, 2018, pp. 72–83.

[66] C. Gao, W. Zheng, Y. Deng, D. Lo, J. Zeng, M. R. Lyu, and I. King, "Emerging app issue identification from user feedback: experience on wechat," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP), Montreal, QC, Canada, May 25-31*, 2019, pp. 279–288.

[67] C. Iacob, V. Veerappa, and R. Harrison, "What are you complaining about?: a study of online reviews of mobile applications," in *BCS-HCI '13 Proceedings of the 27th International BCS Human Computer Interaction Conference, Brunel University, London, UK, 9-13 September 2013*, 2013, p. 29.

[68] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*, 2013, pp. 41–44.

[69] X. Gu and S. Kim, ""what parts of your apps are loved by users?" (T)," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, 2015, pp. 760–770.

[70] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE Trans. Software Eng.*, vol. 43, no. 9, pp. 817–847, 2017.

[71] J. Zeng, J. Li, Y. He, C. Gao, M. R. Lyu, and I. King, "What you say and how you say it: Joint modeling of topics and discourse in microblog conversations," *TACL*, vol. 7, pp. 267–281, 2019.

[72] G. Chen, E. Tosch, R. Artstein, A. Leuski, and D. R. Traum, "Evaluating conversational characters created through question generation," in *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference, May 18-20, 2011, Palm Beach, Florida, USA*, 2011.

[73] A. Ritter, C. Cherry, and W. B. Dolan, "Data-driven response generation in social media," in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2011, pp. 583–593.

[74] O. Vinyals and Q. V. Le, "A neural conversational model," *CoRR*, vol. abs/1506.05869, 2015. [Online]. Available: http://arxiv.org/abs/1506.05869

[75] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst, "Moses: Open source toolkit for statistical machine translation," in *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*, 2007.

[76] G. Salton, A. Wong, and C. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, 1975.