App-Aware Response Synthesis for User Reviews

Umar Farooq*, A.B. Siddique*, Fuad Jamour, Zhijia Zhao, Vagelis Hristidis
University of California, Riverside
ufaro001@ucr.edu, msidd005@ucr.edu, fuadj@ucr.edu, zhijia@cs.ucr.edu, vagelis@cs.ucr.edu

2020 IEEE International Conference on Big Data (Big Data) | 978-1-7281-6251-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/BigData50022.2020.9377983

***Abstract*—Hundreds of thousands of mobile app users post their reviews online. Responding to user reviews promptly and satisfactorily improves application ratings, which is key to application popularity and success. The proliferation of such reviews makes it virtually impossible for developers to keep up with responding manually. To address this challenge, recent work has shown the possibility of automatic response generation by training a seq2seq model with a large collection of review-response pairs. However, because the training review-response pairs are aggregated from many different apps, it remains challenging for such models to generate *app-specific* responses, which, on the other hand, are often desirable as apps have different features and concerns. Solving the challenge by simply building an app-specific generative model per app (i.e., training the model with review-response pairs of a single app) may be insufficient because individual apps have limited review-response pairs, and such pairs typically lack the relevant information needed to respond to a new review.**

To enable *app-specific* response generation, this work proposes AARSYNTH: an app-aware response synthesis system. The key idea behind AARSYNTH is to augment the seq2seq model with information specific to a given app. Given a new user review, AARSYNTH first retrieves the top-K most relevant app reviews and the most relevant snippet from the app description. The retrieved information and the new user review are then fed into a fused machine learning model that integrates the seq2seq model with a machine reading comprehension model. The latter helps digest the retrieved reviews and app description. Finally, the fused model generates a response that is customized to the given app. We evaluated AARSYNTH using a large corpus of reviews and responses from Google Play. The results show that AARSYNTH outperforms the state-of-the-art system by 22.2% on BLEU-4 score. Furthermore, our human study shows that AARSYNTH produces a statistically significant improvement in response quality compared to the state-of-the-art system.

***Index Terms*—App reviews, Natural language generation, Machine translation.**

I. INTRODUCTION

The wide adoption of smartphones has created a large and growing market for mobile apps. Recent studies [1] predicted that the number of smartphone users will reach 3.8 billion worldwide by 2021, projecting a market of trillion dollars for mobile apps by 2023. These apps are typically distributed through app stores such as Apple App Store and Google Play. App stores allow users to give their feedback, ask questions, and publicly express their levels of satisfaction with an app through reviews and ratings: positive reviews and ratings are important factors to acquire and retain users. App developers can respond to user feedback to maintain and improve their app reviews and ratings. According to Google Play, responding to user reviews leads to an increase of 0.7 stars for an app on average [2]. While developers recognize the importance of responding to user reviews promptly, the proliferation of reviews makes it virtually impossible to manually provide responses to all the reviews.

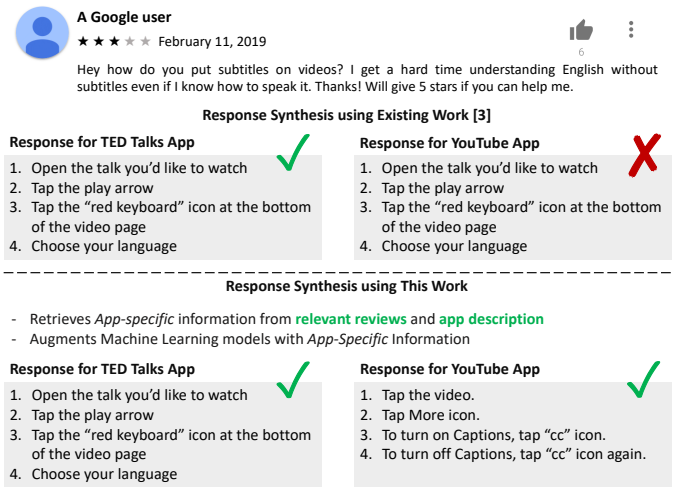

Fig. 1: Example user review and automatic responses: The response generated by RRGen [3] is suitable for TED Talks but not for YouTube. By contrast, our system generates responses *specific* to the considered app with the help of the most relevant reviews and the app description.

Given the importance of responding to user reviews and the impossibility of manually responding to each review, it has become crucial to automatically synthesize responses. Little research has been done to build machine learning models for automatically synthesizing responses to app reviews and questions. RRGen [3] stands out as the first effort in this direction, where the authors proposed using an attention-based sequence-to-sequence (a.k.a. seq2seq) neural model [4]. The authors trained their model using a dataset of review-response pairs for a large collection of apps hosted on Google Play. RRGen generates satisfactory responses to reviews common among many apps (e.g., "*lots of ads*"); however, it fails to synthesize responses specific to an app – see Figure 1 for an example where RRGen generates the same response for two different apps (TED Talks and YouTube). In this example, RRGen was able to generate an appropriate response for TED Talks but not for YouTube because its training data happened to have information specific to TED Talks.

* Equal contribution.

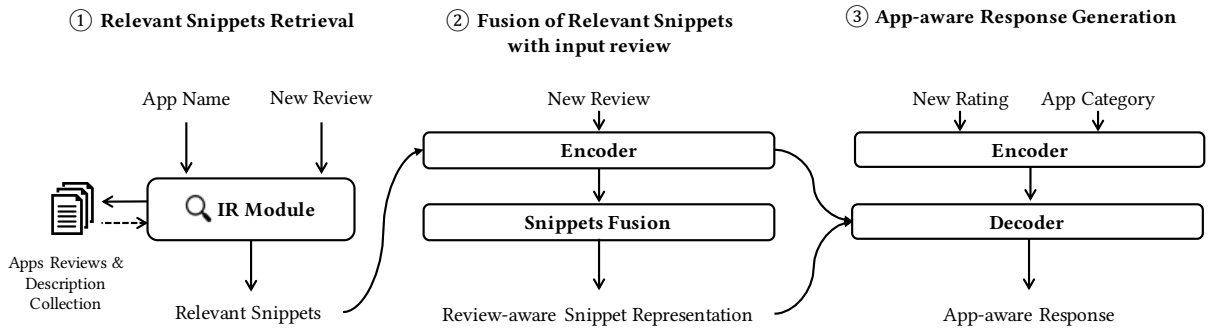

Fig. 2: Overview of AARSYNTH.

There are two key challenges in generating app-specific responses effectively. First, there may be a lack of sufficient review-response pairs for the given app to reliably train a generative model. Second, appropriate responses may not be available in any of the existing training pairs. To address the first challenge, we leverage the review-response pairs from other apps. For the second challenge, we propose to feed the generative model with the most relevant snippets extracted from the app's description and its existing user reviews – important sources of information that have been neglected or under-utilized by prior work. In fact, our analysis of 10 trending apps on Google Play revealed that 50% of the analyzed apps provide FAQ sections and 32.3% have reviews with snippets closely relevant to answer app-specific questions raised in other reviews of the same app.

Figure 2 shows an overview of AARSYNTH. Given a new user review for an app, our information retrieval (IR) module first retrieves relevant text snippets from the app's description and existing user reviews. Then, the snippets fusion layer uses these snippets to build a review-aware representation (i.e., a representation of the snippets that is associated with the input review) similarly to machine reading comprehension models [5], [6] (MRC). Finally, the review-aware snippets representation and the input review representation along with other app-specific features produced by our encoder are passed to the decoder to produce an app-specific response.

Note that neither an MRC model nor a seq2seq model by themselves are sufficient for the above task. On one hand, an MRC model typically produces a span text – a substring extracted from a textual document, which is hard to comprehend as the response to a review. On the other hand, despite that a seq2seq model can produce free-form text, it can not use the relevant snippets unless they are transformed into a representation that is aware of the input review. Our fusion between the seq2seq and MRC models resolves the limitations of each model in the context of app response generation.

We evaluated AARSYNTH using a large corpus of reviews, responses, and app descriptions collected from Google Play. Our results show that AARSYNTH outperforms the state-of-the-art system RRGen [3] by 22.2% in BLUE-4 score – a widely used metric for text generation. Moreover, our human study using Amazon Mechanical Turk shows that the responses generated by AARSYNTH better address user concerns, are app-specific, and are more fluent than the responses generated by RRGen with a statistically significant difference.

In summary, this paper makes the following contributions:

- It proposes a novel neural architecture that fuses seq2seq and machine reading comprehension models to synthesize free-form responses specific to an app.
- It releases a large dataset[1] that consists of more than 570K review-response pairs and more than 2 million user reviews for 103 popular applications.
- It conducts extensive experimental analysis using our large dataset and compares AARSYNTH against the state-of-the-art systems. The evaluation using automatic metrics and real-user studies confirms the competitiveness of AARSYNTH with a statistically significant improvements.

In the following, we first present a brief review on attentional encoder-decoder models and information retrieval techniques in Section II. Then, we present AARSYNTH in Section III, followed by our experimental setup (Section IV) and evaluation (Section V). Finally, we discuss the related work in Sections VI and conclude this work in Section VII.

II. BACKGROUND

AARSYNTH is built on top of seq2seq neural networks and state-of-the-art information retrieval techniques. This section offers a brief introduction to these topics.

A. Attentional Encoder-Decoder Model

The goal of an encoder-decoder model, like seq2seq [4], is to synthesize a target sequence (e.g., developer response) $\mathcal{Y} = (y_1, y_2, ..., y_m)$ given an input sequence (e.g., user review) $\mathcal{X} = (x_1, x_2, ..., x_n)$, where m and n are target and input sequence lengths. Figure 3 presents a high level overview of the seq2seq model that shows the encoder and the decoder parts.

Encoding. The encoder reads an input sequence of length n, one token at a time until it encounters the end of sequence token (i.e., $< eos >$). It transforms the sequence into hidden states $\mathcal{H} = (\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_n)$ by applying a Recurrent Neural Network (RNN), such as Long Short-Term Memory (LSTM) [7]. Specifically, it transforms input token x_i to hidden state $\mathbf{h}_i = \text{encoder}(\mathbf{h}_{i-1}, \text{emb}(x_i))$, where encoder$(\cdot)$ is a non-linear mapping function, emb(x_i) is the word embedding of input token x_i, and $\mathbf{h}_{i-1}$ is the previous hidden state.

[1] Available at https://github.com/AARSynth/Dataset

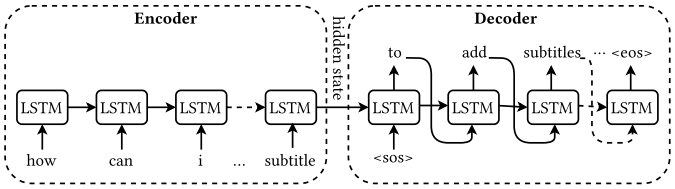

Fig. 3: Overview of the seq2seq neural model.

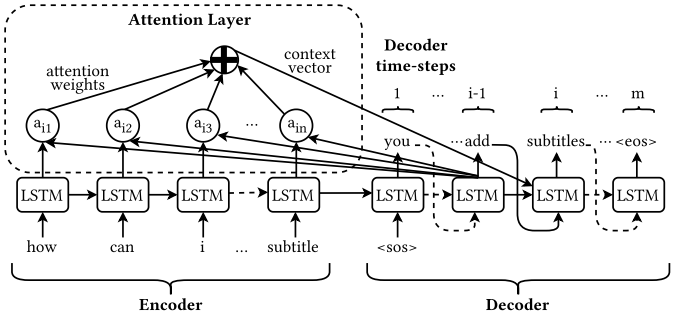

Fig. 4: Attention-based seq2seq neural model.

Decoding. The decoder is initialized with the encoder's last hidden state $\mathbf{h}_n$ and the start of sequence token (i.e., $< sos >$), then it utilizes another RNN to generate the target sequence $\hat{\mathcal{Y}}$. The decoder also generates one token at a time, until the end token (i.e., $< eos >$) is generated. The generation, at time-step i, is conditioned on the previously generated words $\hat{y}_{i-1}$,..., $\hat{y}_1$, and the decoder's current hidden state $\mathbf{h}'_i$, according to the following probability distribution:

$$P(\hat{y}_i|\hat{y}_{i-1}, ..., \hat{y}_1, \mathcal{X}) = \text{softmax}(\text{decoder}(\mathbf{h}'_i, \hat{y}_{i-1})),$$

where decoder$(\cdot)$ is a non-linear mapping function and softmax$(\cdot)$ converts the given vector into a probability distribution. The encoder-decoder model is trained jointly by minimizing the negative log-likelihood loss of the given N training input-target sequence pairs of the form ($\mathcal{X}^i$, $\mathcal{Y}^i$):

$$\mathcal{L}(\theta) = -\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \log p_\theta(\mathcal{Y}^i|\mathcal{X}^i),$$

where θ is a set of trainable parameters estimated using optimization algorithms such as stochastic gradient descent.

Attentional Decoding. Attention mechanisms [8] are used in seq2seq models to pay attention to more relevant input tokens while decoding. Figure 4 shows the computation of an attention vector at time-step i. While decoding at time-step i, the decoder$(\cdot)$ is not only conditioned on the decoder's current hidden state $\mathbf{h}'_i$ and the previous generations y_{i-1},..., y_1, but also on the attention context vector $\mathbf{v}_i$, which is computed as:

$$\mathbf{v}_i = \sum_{k=1}^{n} \mathbf{a}_{ik}\mathbf{h}_k,$$

where $\mathbf{a}_{ik}$ is the attention weight for hidden state $\mathbf{h}_k$, capturing how relevant is encoder's k-th hidden state for predicting token

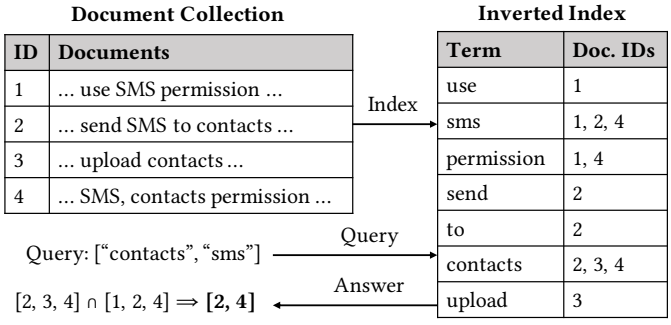

Fig. 5: Overview of IR indexing and searching.

$\hat{y}_i$ while considering the decoder's previous hidden state $\mathbf{h}'_{i-1}$. The attention weight $\mathbf{a}_{ik}$ at time-step i can be computed as:

$$\mathbf{a}_{ik} = \frac{\exp(e_{ik})}{\sum_{j=1}^{n} exp(e_{ij})},$$

where $e_{ik} = \text{align}(\mathbf{h}'_{i-1}, \mathbf{h}_k)$ and align$(\cdot)$ is an alignment model implemented as a Multi-Layer Perception (MLP) unit. The attentional seq2seq neural network is trained jointly. We employ the seq2seq framework as a basic building block in our proposed approach for response synthesis.

B. Relevant Document Retrieval

Given a collection of documents (e.g., app description and reviews) and a query (e.g., new review), an IR system returns a subset of documents relevant to the query. Figure 5 illustrates how an IR system works: the system creates an index for the given document collection, based on which it finds relevant document(s) for a given query.

Document Indexing. IR systems create inverted indexes to facilitate faster document retrieval. An inverted index consists of a set postings lists; a postings list is a list of individual postings, each of which provides information about occurrences of a term (i.e., word) in the document collection including document id and the number of occurrences of the term in each document that contains the term – term frequency (TF). We use Apache Lucene [9], which uses SkipList [10] to implement postings lists for fast retrieval.

Searching and Ranking. IR systems typically use vector space retrieval with Term Frequency-Inverse Document Frequency (TF-IDF) weighting [11]. TF captures the importance of a term in a document, and IDF captures the significance of the term in the whole collection. The BM25 [12] algorithm is widely used by IR systems to improve search engine relevance. BM25 scores a document D for input query Q with terms $q_1, q_2, ..., q_n$ as follows:

$$\text{score}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})},$$

where $f(q_i, D)$ represents the term frequency of q_i in the document D, the document D with length $|D|$, and $avgdl$

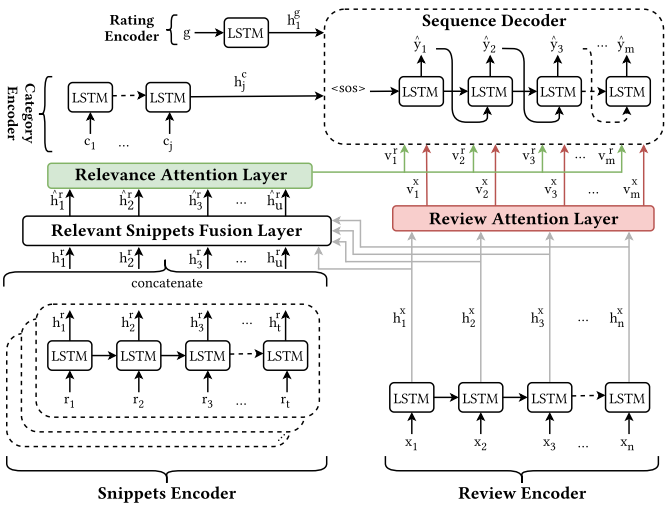

Fig. 6: Architecture of AARSYNTH's neural model.

is the average document length in the whole collection, $k_1 \in [1.2, 2.0]$ and $b = 0.75$. In the context of AARSYNTH, we use IR to retrieve the most relevant text snippets from description and reviews of the given app.

III. App-Aware Response Synthesis

In this section, we present AARSYNTH (see Figure 2 for an overview). In the following, we first introduce the IR module, then provide an overview of the neural model followed by the details of each of its major components.

A. IR Module

AARSYNTH employs IR module as follows. First, it builds an inverted index for all the app descriptions and reviews. Then, it takes a new user review and the app name as inputs, and retrieves top-k relevant documents (i.e., reviews and description of the given app). To get the most relevant text snippets from app description and reviews, we leverage Lucene [9] Highlighter and Sentence Detector [13] to extract the most relevant snippets from the retrieved document(s). We refer to these text snippets as *relevant snippets*.

B. Overview of Neural Model

Figure 6 shows the architecture of the proposed app-aware response synthesis model, which consists of the following components: (*i*) input encoding layers to transform textual inputs to high dimensional contextual representations; (*ii*) relevant snippets fusion layer to produce a review-aware representation of the relevant snippets; (*iii*) attention layers to compute attention weights that capture the significance of tokens in the input review and its relevant snippets; and (*iv*) sequence decoding layer that fuses all the information from the previous layers to ultimately synthesize an app-aware response. We next explain each component in detail.

C. Input Encoding Layers

Review Encoder. To encode an input review $\mathcal{X} = (x_1, x_2, ..., x_n)$, the review encoder first maps each word x_i to a high dimensional vector space (i.e., word embedding) $\text{emb}_i^{\mathcal{X}}$, then an RNN is utilized to produce a new d-dimensional representation $\mathcal{H}^{\mathcal{X}} = (\mathbf{h}_1^{\mathcal{X}}, \mathbf{h}_2^{\mathcal{X}}, ..., \mathbf{h}_n^{\mathcal{X}}) \in \mathbb{R}^{d \times n}$ of all the tokens in the input review, where LSTM is used as an RNN. The token encoding at time-step i is computed using an LSTM as follows:

$$\mathbf{h}_i^{\mathcal{X}} = \text{LSTM}(\mathbf{h}_{i-1}^{\mathcal{X}}, \text{emb}_i^{\mathcal{X}}).$$

This representation is used by the review attention layer to compute the attention weights vector for the sequence decoder and relevant snippets fusion layer to compute the review-aware representation of the relevant snippets.

Snippets Encoder. The relevant snippets are retrieved by our IR module based on the input review and app name, and then passed to the snippets encoder, which produces a new representation $\mathcal{H}^{\mathcal{R}} = (\mathbf{h}_1^r, \mathbf{h}_2^r, ..., \mathbf{h}_u^r) \in \mathbb{R}^{d \times u}$ where u is the total number of tokens in all the retrieved snippets. The representation is the result of concatenating the representations produced by the RNN for each snippet. This representation is used by the relevant snippets fusion layer to compute the importance of the words in the snippets with respect to the given user review $\mathcal{X}$, and is used by the relevance attention layer to generate an attention weights vector of the relevant snippets.

Category and Rating Encoders. The category encoder produces a representation $\mathbf{h}_1^c, \mathbf{h}_2^c, ..., \mathbf{h}_b^c$ for the category of the app and the rating encoder encodes the review rating into $\mathbf{h}_1^g \in \mathbb{R}^d$. The final hidden states of these layers are passed to the sequence decoder.

D. Relevant Snippets Fusion Layer

This layer associates and fuses information from the relevant snippets and the words of the input review. First, we compute a similarity matrix $\mathcal{S} \in \mathbb{R}^{u \times n}$ between the encodings of the snippets $\mathcal{H}^{\mathcal{R}}$ and the encodings of the review $\mathcal{H}^{\mathcal{X}}$, where $\mathcal{S}_{bk}$ (value at row b and column k) represents the similarity between the b-th word in the snippets and k-th word in the user review, which is computed using $\mathcal{S}_{bk} = \alpha(\mathcal{H}^{\mathcal{R}}_{:b}, \mathcal{H}^{\mathcal{X}}_{:k}) \in \mathbb{R}$. α is a function trained to capture the similarity between input vectors $\mathcal{H}^{\mathcal{R}}_{:b}$ and $\mathcal{H}^{\mathcal{X}}_{:k}$, where $\mathcal{H}^{\mathcal{R}}_{:b}$ and $\mathcal{H}^{\mathcal{X}}_{:k}$ are b-th and k-th column-vectors of $\mathcal{H}^{\mathcal{R}}$ and $\mathcal{H}^{\mathcal{X}}$, respectively. $\alpha(r, x) = w_{(s)}^{\top}[r \oplus x \oplus r \otimes u]$, where $\oplus$ is vector concatenation, $\otimes$ is element-wise multiplication, and $w_{(s)}$ is a trainable weight vector. Then, from the similarity matrix $\mathcal{S}$, we can get the most important snippet words with respect to the review, i.e., with the closest similarity to the user review. The attention weights for the snippet words are computed using $\mathbf{z} = \text{softmax}(\max_{col}(\mathcal{S})) \in \mathbb{R}^u$, where $\max_{col}$ represents max across columns. Then, the most important words in the snippets with respect to the review $\hat{\mathcal{H}}^{\mathcal{R}} \in \mathbb{R}^{d \times u}$ can be computed by tiling the operation $\hat{\mathbf{h}}^{\mathcal{R}}$ across columns u times,

where $\hat{\mathbf{h}}^{\mathcal{R}} = \sum_b \mathbf{z}_b \mathcal{H}^{\mathcal{R}}_{:b} \in \mathbb{R}^{d}$. The matrix $\hat{\mathcal{H}}^{\mathcal{R}} \in \mathbb{R}^{d \times u}$ represents the *fused* information between user review and relevant snippets, i.e., review-aware representation of the snippets. This representation can be thought of an MRC model representation of the snippets that is fused with seq2seq representations in the sequence decoder while synthesizing the response.

E. Attention Layers

Review Attention Layer. This layer computes the attention weights of each token in the review as follows. It takes in the encoded representation of the review $\mathcal{H}^{\mathcal{X}}$ and the decoder's hidden state at previous time-step $\mathbf{h}'_{i-1}$, and produces an attention context vector $\mathbf{v}^{\mathcal{X}}_i$ for the decoder at time-step i. The context vector $\mathbf{v}^{\mathcal{X}}_i$ captures the importance of each hidden state of the review encoder while generating token $\hat{y}_i$ at time-step i. The details on how to compute the context vector are presented in section II-A.

Relevance Attention Layer. This layer produces a representation that enables the decoder to pay more attention to the important words in the relevant snippets while generating the final response. It takes in the review-aware representation of the relevant snippets $\hat{\mathcal{H}}^{\mathcal{R}}$ and decoder's hidden state at previous time-step $\mathbf{h}'_{i-1}$ and computes attention context vector $\mathbf{v}^{r}_i$ for decoder at time-step i. The attention context vector $\mathbf{v}^{r}_i$ signifies the importance of each hidden state of the review-aware representation produced by relevant snippets fusion layer (Section III-D) that sequence decoder utilizes while synthesizing the token at time-step i.

F. Sequence Decoder

The sequence decoder fuses the representations of the review, review rating, and category with the MRC style review-aware representation of snippets to generate the final response. The sequence decoder has a softmax-based linear layer that follows RNN to map the d-dimensional hidden states to a probability distribution over the whole vocabulary. At time-step i, the decoder computes a conditional probability to generate $\hat{y}_i$ as given below:

$$P(\hat{y}_i|\hat{y}_{i-1}, ..., \hat{y}_1, \mathcal{X}, \mathcal{R}, \mathcal{C}, g) = \text{decoder}(\mathbf{h}'_i, \text{emb}^{\hat{y}}_{i-1}, \mathbf{v}^{\mathcal{X}}_i, \mathbf{v}^{r}_i, \mathbf{h}^{c}_b, \mathbf{h}^{g}_1).$$

Note that, at time-step i, the decoder considers its current hidden state $\mathbf{h}'_i$, the embedding of the token prediction $\text{emb}^{\hat{y}}_{i-1}$ at the previous time-step, review attention vector $\mathbf{v}^{\mathcal{X}}_i$ from the review attention layer, relevant snippets attention vector $\mathbf{v}^{r}_i$ from the relevance attention layer, category encoding $\mathbf{h}^{c}_b$, and rating encoding $\mathbf{h}^{g}_1$ to generate a token $\hat{y}_i$.

G. Training and Inference

Training. All the components of the model are trained jointly over N training examples in an end-to-end fashion to minimize the negative log-likelihood loss as given below:

$$\mathcal{L}(\theta) = -\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \log p_\theta(\mathcal{Y}^i|\mathcal{X}^i, \mathcal{R}^i, \mathcal{C}^i, g^i),$$

where $\mathcal{R}^i$ represents relevant snippets from app description and reviews retrieved by the IR module. App category and review rating are represented by $\mathcal{C}^i$ and g^i respectively, for review-response training example i of form ($\mathcal{X}^i$, $\mathcal{Y}^i$). θ is a set of trainable parameters of the model. The teacher forcing algorithm that always passes ground truth to the decoder at next time-step, has traditionally been used to achieve faster convergence in training, but it causes an incompatibility in the train and test set-ups. Whereas curriculum learning [14] algorithm passes the current prediction of the decoder to the next time-step to minimize the incompatibility of the train and test set-ups, and enables the model to correct itself, but the model may take longer to train and converge. In our training, we use a mix of both algorithms with equal probability in the sequence decoder layer to minimize the incompatibility of the train and test set-ups and achieve fast convergence at the same time.

Inference. We select the model with the best performance on the validation set for inference. We utilize the beam search algorithm [15] that has been employed in natural language generation tasks like neural machine translation. While generating the response, it picks multiple alternative tokens (i.e., the ones with high probabilities) from the decoder's probability distribution at every time-step. The parameter $\mathcal{B}$ controls the number of alternatives. At subsequent time-steps, $\mathcal{B}$ copies of the decoder are created, each receives a different input from the previous time-step and picks multiple alternative choices. Finally, the output that maximizes the joint probability of the response is selected. While this approach is computationally expensive, it has shown better performance than greedy decoding that picks the word with maximum probability at every time-step.

IV. Experimental Setup

A. Dataset

Data Collection. We crawled 103 popular apps (those with at least 25K star ratings and at least 100 developer replies) across 23 app categories, and collected over 3.4 million reviews and more than 570K review-response pairs. We collected app name, description, number of star ratings, app category, review text, review time, review rating, developer response, and response time. Thanks to our app selection criteria, we were able to collect reviews with a much higher response rate: 14.4% compared to 2.8% in [16].

Preprocessing. User reviews often contain noisy data [17]. To mitigate this, we performed the following preprocessing steps: (*i*) removed non-English reviews and responses using a language detector [18], which ensures a concise and valid vocabulary; (*ii*) performed standard natural language processing (NLP) preprocessing steps such as conversion of letters to lower case, replacement of numbers with "<number>", emails and URLs with "<email>" and "<url>", respectively; (*iii*) replaced greetings and signatures with "<salutation>" and "<signature>", respectively, to preserve user anonymity;

TABLE I: Sample input reviews along with relevant reviews and app description snippets: relevant reviews and app descriptions not only contain contextual keywords, but also often provide partial answers to questions raised in input reviews.

App	User review	Relevant reviews from the same app	App description
Udemy (Education)	... intermittent coverage on my train ride and wanted to watch ... seemingly impossible.	▷ ... download lecture videos for offline ... on my train ride to campus ... ▷ ... problem with video ... It only work after I downloaded it.	... Download courses to learn offline. On the go ...
Uber Eats (Food)	... food was not delivered ... cannot refund the money ...	▷ ... my order was cancelled ... my money was transferred back ... ▷ ... food that is undercooked ... they refunded for the entire meal.	Track your food order ... See the estimated delivery time ...
Uber (Navigation)	... charge cancellation fee ... requires credit or debit card ...	▷ ... do not charge cancellation fees, trust worthy, make refunds easily ...	... request a ride ... pay with credit or cash in select cities ...

and (*iv*) removed reviews and responses with less than four words since such reviews/responses are not likely to contain useful information. After preprocessing, we obtained $425,618$ review-response pairs and $2,077,674$ reviews with no response. We found that 47 out of 103 apps overlap with RRGen [3] dataset apps. Next, we applied the same preprocessing steps to the RRGen dataset, which resulted in 145K review-response pairs (out of 309K pairs in the original RRGen dataset). We finally merged the two datasets and the final dataset contains $570,881$ review-response pairs.

We randomly split the review-response pairs of our dataset into training ($530,872$), validation ($19,511$), and test ($19,480$ $\approx 3.5\%$ of dataset) sets.

Dataset Analysis. Table I shows a few examples of user reviews, relevant reviews, and app description snippets from our dataset. The snippets from the relevant reviews and app description contain keywords and similar context that can help synthesize a response. In fact, the relevant snippets in many cases provide partial answers for the given user review. For example, while using the Udemy app (see Table I), a user was unable to watch a video due to intermittent network coverage on a train ride. In this case, the app description happens to have relevant guidance – "download courses to learn offline". Luckily, other users who faced a similar issue shared the solution "download lecture videos".

B. Evaluation Metrics

We use quantitative automatic metrics as well as subjective human studies to evaluate the performance of AARSYNTH and the competing systems.

Automatic Metrics. BLEU [19] score is a standard automatic metric to evaluate natural language generation solutions such as machine translation [20] and paraphrasing [21]. It has been demonstrated to have a positive correlation with human judgements. BLEU-n ($n \in \{1,2,3,4\}$) score $\in [0,100]$ captures the percentage of the n-grams from the synthesized response $\hat{\mathcal{Y}}$ that also co-occur in the the ground truth $\mathcal{Y}$, where 0 means no matching n-grams and 100 means a perfect match. We utilize BLEU-4, which is considered a standard metric. Moreover, we also use the recall-based automatic metric ROUGE [22], which measures n-grams overlap between the synthesized response $\hat{\mathcal{Y}}$ and the ground truth $\mathcal{Y}$. We use ROUGE-L that identifies the longest co-occurrence of n-grams using the Longest Common Subsequence (LCS) [23], which naturally evaluates sentence structure similarity.

Human Study. We conducted a subjective human study to evaluate the quality of the responses generated by AARSYNTH and other competing methods with respect to the developer responses. We made use of Amazon Mechanical Turk, a crowd sourcing platform for human evaluation, where human evaluators rate the quality of the response on a scale of $1-5$, 1 being the worst and 5 being the best. We asked human evaluators to consider three aspects in their evaluations: (*i*) the response is specific to the app; and (*ii*) whether the response addresses the concern of the user raised in the input review; (*iii*) the language fluency and the grammatical correctness of the response. We randomly selected 150 generated responses by AARSYNTH and the competing systems for the same reviews, and each response was scored by five different human evaluators who are familiar with the respective app.

C. Competing Approaches

We compare AARSYNTH with two IR baselines, an MRC model R-Net [5], and the state-of-the-art response generation system RRGen [3]. We briefly describe each competing method below:

IR-Reviews: This baseline builds an index for the reviews in the training set. For an input review and app name from the test set, it retrieves the most relevant indexed review (i.e., top-1) as a response.

IR-Response: This baseline builds an index for the developer responses in the training set. For an input review and app name from the test set, it retrieves the most relevant indexed developer response (i.e., top-1) as a response.

R-Net [5]: This system was proposed for machine reading comprehension style question answering, and it achieves state-of-the-art results on SQuAD [24] and MS-MARCO [25] datasets. MRC models require annotated answer spans in a passage (app description and reviews in the context of response synthesis) for supervision, which are not available in our dataset, and acquiring such manual annotations is laborious. We annotated spans using BLEU-2 heuristics: the reviews that achieve the maximum BLEU-2 score with the developer response are considered a span. R-Net obtains question matching representation of a passage by passing the question and the passage through a GRU, then self-matching attention mechanism is employed to refine the generated repreentation.

RRGen [3]: RRGen uses attention-based seq2seq for producing a response for an input review. It conditions the response generation on app category, rating, review

TABLE II: Automatic metrics results: p_n represents n-gram precision for the ground truth and synthesized response.

Method	BLEU	ROUGE-L	p_1	p_2	p_3	p_4
IR Reviews	13.67	12.76	21.49	15.56	11.44	8.56
IR Response	19.19	17.99	27.18	20.89	17.14	14.88
R-Net	29.16	39.92	42.89	29.83	23.19	16.72
RRGen	34.55	46.26	50.38	37.54	28.25	22.63
AARSYNTH	**42.22**	**51.89**	**56.99**	**44.50**	**36.56**	**30.83**

length, review sentiment, and a set of dictionary-based keywords. RRGen achieves current state-of-the-art results on BLEU [19] metric for the response generation task. Since RRGen outperformed NNGen [26] and NMT [20] models, we do not consider them as competing approaches in this work. We used RRGen's open-source implementation [27] in our experiments.

D. Implementation Details

We used Apache Lucene [9] and the BM25 [12] scoring algorithm in our IR module, which retrieves 4 snippets from the relevant user reviews for an app and 1 snippet from the app description. We implemented AARSYNTH in PyTorch [28]. The word embedding dimensions were set to 128 and the vocabulary size was set to $10,000$. Based on the word length outlier analysis, we set the maximum length for the review, snippets, app category, rating, and response to 75, 50, 4, 1, and 120, respectively. AARSYNTH uses 128 LSTM units as an RNN that has 2 layers for all the encoders (review, snippets, app rating, and category encoders) and the decoder. We set the batch size to 128 and the dropout rate to 0.2 (i.e., to avoid over-fitting to training set). We trained the neural network for 25 epochs using Adam Optimizer and employ negative log likelihood loss with a learning rate of 0.01.

V. RESULTS

A. Automatic Metrics

Performance comparison. Table II shows the BLEU, ROUGE-L, and n-gram precision scores for all competing systems. As expected, the IR baselines IR Reviews and IR Response produce the worst results (13.67 and 19.19 BLEU scores, respectively), since they can not generate a response and are only capable of retrieving the most relevant reviews or responses from the training set. The MRC model R-Net produces mediocre results, mainly because of the limitation of MRC models that they attempt to predict spans from the related documents (i.e., app description and reviews), which rarely contain perfect responses. However, R-Nets's BLEU score of 29.16 shows that snippets from the app description and reviews can be helpful for synthesizing a high-quality response. AARSYNTH achieves the best results on all automatic metrics; specifically, it outperforms the state-of-the-art system RRGen by 22.20% on BLEU and 12.17% on ROUGE-L. The outstanding performance of AARSYNTH is attributed to the core idea of this paper, i.e., fusing MRC with attention-based seq2seq. The MRC model, coupled with our IR module, discovers relevant app-specific knowledge from relevant reviews and app descriptions, and produces a review-aware representation that is associated with the input review and its important keywords. The seq2seq model learns from the available review-response training pairs. The seq2seq decoder fuses knowledge from both MRC and seq2seq encoders, and thus learns to synthesize responses that are not only fluent and relevant but also app-aware.

TABLE III: Contribution of each component on AARSYNTH's performance. The attention-based seq2seq is the basic component of AARSYNTH; thus, it is present in all configurations.

Component	BLEU	ROUGE-L
Attentional seq2seq	21.36	26.19
+ Rating	24.20	31.80
+ Category	30.97	44.57
+ Rating + Category	31.81	44.96
+ Description	34.75	49.71
+ Rating + Category + Description	36.11	49.98
+ Description + Review	38.31	50.39
AARSYNTH (Rating + Category + Desc. + Review)	**42.22**	**51.89**

Ablation study. Table III shows the effect of each component on the performance of AARSYNTH; that is, the table shows the performance of AARSYNTH when certain components are enabled. When only the attention-based seq2seq is enabled, AARSYNTH achieves a 21.36 and 26.19 BLEU and ROUGE-L scores, respectively. Enabling the use of app features such as review rating and app category results in a modest improvement of $+2.84$ and $+9.61$ on BLEU score, respectively. While other features such as review length and sentiment may, in principle, improve the performance of the system, it has been confirmed in [3] that such features merely cause small improvements. We highlight that enabling the use of our fused architecture (configuration Att. seq2seq + Description + Review) produces the maximum gain on all the metrics: $+16.95$ on BLEU and $+24.2$ on ROUGE-L metrics. Moreover, if no relevant reviews are available for a given app (configuration Att. seq2seq + Rating + Category + Description), AARSYNTH continues to provide better performance compared to the state-of-the-art system RRGen (36.11 vs. 34.55 BLEU scores). Several other configurations are also provided in Table III for completeness.

Effect of hyperparameters. Table IV shows the effect of adjusting the hyperparameters on the overall performance of AARSYNTH. Configuration 5, which has 128 hidden and word embedding dimensions and uses up to top-5 relevant snippets, outperforms the other configurations. Increasing the size of the hidden units, word embedding dimensions, or the number of relevant snippets does not necessarily improve the performance. Additionally, configuration 3 and 5 are similar with one exception: the number of layers. Increasing the number of layers improves the results since more layers can capture better representations. Note that even configuration 1, which produces the worst results among the reported configurations, outperforms the state-of-the-art system RRGen. This confirms that our performance is not due to parameter tuning, but rather due to fusing seq2seq and MRC architectures.

TABLE IV: Effects of different hyperparameters on the performance of AARSYNTH. L is the number of layers, H is the number hidden dimensions, E is the number of embedding dimensions, and S is the number of relevant snippets.

Sr. #	Configuration	BLEU	ROUGE-L
1	L=2, H=256, E = 256, S = 5	35.95	46.16
2	L=2, H=128, E = 256, S = 5	36.80	49.30
3	L=1, H=128, E = 128, S = 5	39.31	49.69
4	L=2, H=128, E = 128, S = 10	41.72	51.58
5	L=2, H=128, E = 128, S = 5	**42.22**	**51.89**

B. Human Study

We conducted a human study to evaluate the quality of the responses generated by each of the competing systems from a human perspective. We limit this study to the best performing systems based on the results of the automatic metric evaluation. Specifically, we compare AARSYNTH, R-Net, and RRGen in this study. We randomly selected 150 input reviews and generated their respective responses using each of the competing systems. For each input review, we randomly selected 5 human evaluators to score the respective responses without knowing which responses were generated by which method or whether the response was produced by a developer. Additionally, we required that the evaluators are familiar with the app whose responses are under investigation; in total, 750 evaluators participated in the evaluations. Evaluators were asked to judge a response based on three criteria: app-specificity, whether it contains information that addresses user concerns, and fluency. We report average scores with confidence intervals for responses generated by developers (our ground truth) and each of the competing systems in Table V. The responses generated by AARSYNTH achieve the best scores on all the aspects when compared to other generative methods. The statistical significance test result ($p-value < 0.01$) [29] also shows that the responses generated by our method achieve statistically significant improvement on all three criteria when compared with R-Net and RRGen. Moreover, responses generated by AARSYNTH achieve very close scores to those manually generated by developers. While AARSYNTH produces app-specific high-quality responses, these are often not as fluent as manually written responses. This is a common limitation in most conditional natural language generation tasks. In future, we plan to improve the fluency of AARSYNTH's responses by incorporating a neural language model. Overall, the results of our human study show that the responses generated by AARSYNTH are not only better than other automatic response generation methods, but also on a par with the developers' manually generated responses.

C. Sample Responses

We present in Table VI sample responses generated by app developers, R-Net, RRGen, and AARSYNTH. In what follows, we highlight several interesting examples. Consider the UC Browser app which offers a download option. When a user faces problems while using this option, AARSYNTH is able to discover the resume feature in the download option through relevant snippets, and it offers a possible solution for the problem by including "try resume again it starts downloading..." in the response. Similarly, AARSYNTH learns whether "clear the cache of the app" can possibly solve the issue for the Adobe Reader app (see Table VI), and forwards this information by assigning high weights for the important keywords to the sequence decoder to generate app-aware responses. This example highlights how AARSYNTH generates app-specific responses. AARSYNTH takes advantage of the ever increasing relevant snippets from other reviews of the same app, which are likely to have answers to other reviews. This enables AARSYNTH to generate responses that directly address user concerns as opposed to responses that are vaguely related to the input review. Consider the input review "that is not free. they told free but not", where the user is concerned about the app content not being free. AARSYNTH is able to learn that the user concern is about pricing and it communicates a possible remedy by responding with "... you can use free course filter..." (see Table VI).

TABLE V: Human study results: AARSYNTH outperforms all competing systems, and the quality of its responses is close to that of expertly written responses (Developer responses).

Method	App-specific	Addresses Concern	Fluent
R-Net	2.911±0.041	2.881±0.042	2.856±0.043
RRGen	2.915±0.041	2.769±0.044	2.957±0.042
AARSYNTH	**3.408**±0.038	**3.403**±0.038	**3.456**±0.038
Developer	3.436±0.039	3.420±0.039	3.575±0.037

VI. Related Work

Dialogue Response Generation. Response generation is one of the prominent topics in the field of NLP where a system attempts to provide an answer to a given question [30], [31]. IR systems [32]–[34] aim to retrieve the best matching response, and machine translation based approaches [35]–[37] try to generate the response for a given question. Touch your heart [38] proposes to generate tone-aware responses for customer care on social media. RRGen [3] employs features such as category, review sentiments and so on, to generate the responses for user reviews on Google Play. Since RRGen is trained on the review-response pairs of many apps, it can only provide generic responses, and it struggles to answer app-specific reviews. Recently, Google Play also started offering suggested replies [2]. This system also suggests generic replies as others. We utilize snippets from app description and reviews (retrieved through IR module) along with other features to synthesize a fluent and app-aware response that addresses user concerns raised in an input review by fusing seq2seq with MRC models that employ attention mechanisms.

Question Answering. Question answering is an extensively studied research problem, which includes several sub-tasks and datasets. The selection of answers in community question answering [39] and ranking question-answer pairs [40] are related to response synthesis; however, in response synthesis,

TABLE VI: Sample user reviews, developer responses and generated responses using R-Net, RRGen and AARSYNTH.

Review	Method	Response
that is not free. they told free but not	Developer	<salutation> i'm sorry for the inconvenience you are having with the price on udemy. could you please contact us via <email> so we can assist you better.
	R-Net	<salutation> it certainly sounds frustrating and unpleasant experience. we can look into this for you. let us know by sending a quick note to <url>.
	RRGen	<salutation> please send a note us at: <email>.
	AARSYNTH	<salutation> udemy is free, but all online contents are not free. you can use free course filter. please contact us via <email> for further assistance.
not able to share selection? you must be joking. this is the de facto standard reader and behaves in such an autistic way.	Developer	please update your "adobe reader app". issue has been fixed. <signature>
	R-Net	<salutation> we are sorry to hear about your experience with about request you to reach out to us at <url> and we'll will take into this.
	RRGen	<salutation> we have facing difficulties with our app. email a concerns, please contact free to contact with support team at <email>.
	AARSYNTH	<salutation> we are sorry to hear about your experience. please clear the cache of app from the phone settings. if you are still unable to make it work, reach out to us at <email> and we'll will look into this.
fare must be deduce.	Developer	<salutation> we appreciate your feedback and be work towards make this better. if there's anything else we can take a look into for you please let us know to <email> so we can further connect.
	R-Net	we like to hear into this. please email us at <email>.
	RRGen	we appreciate your concern and would like to investigate into your issue. please email us directly afterwards <email> and <number>.
	AARSYNTH	<salutation> we love to hear your feedback. we believe that fare must be higher but comfortable rides than others. Our team is working on this aspect as well. please contact us directly : <email>.
movie downloading stuck at <number> - <number>. worst app.	Developer	<salutation> we regret the inconvenience you have experienced. pls send us more details via menu-feedback page(open in uc browser).you may clear cache, change download path, turn on/off cloud boost, change access point and then <url> a nice day dear!
	R-Net	<salutation> we feel sorry for the inconvenience. if you resume again it starts downloading.
	RRGen	<salutation> we are sorry to hear that. if you will contact to <email> with the version from google app.
	AARSYNTH	<salutation> we regret the inconvenience you have experienced. if you try resume again it starts downloading from <number>. if issue persists, we suggest you to visit <url>.

the goal is to generate coherent and fluent answers rather than answer selection. Similarly, open-domain question answering by reasoning over knowledge bases [41] and large open-domain sources such as Wikipedia [42] can not be employed for the task of review response generation, as knowledge bases are not helpful for synthesizing responses for reviews. Last but not least, closed-domain question answering datasets such as SQuAD [24], SearchQA [43], and MS MARCO [25] assume that the answer to the question is a span from the accompanying set of document(s), whereas developer response is rarely a span from the related reviews or application description.

Analysis of App Reviews. Many studies collected and examined different aspects of app reviews. ChangeAdvisor [44] uses app reviews to extract useful feedback to recommend software maintenance changes. The authors in [45] studied user reviews to improve automated testing. The authors in [17] used app reviews to detect real-time emerging issues in the apps, others used GitHub to study issues [46], [47]. The authors in [48] used probabilistic techniques to classify reviews into bug report, feature requests and so on. Sentiment analysis on app features is performed in [49], [50]. Similarly, MARA [51] uses reviews to predict app feature requests. [16] analyzed 4.5 million reviews and highlighted the importance of developer replies to app reviews. Our research extends the findings of these works by collecting a large dataset of reviews and using the related review snippets to generate app-aware responses.

VII. CONCLUSION

We have presented AARSYNTH, a system for automatically synthesizing app-specific responses to mobile app reviews. AARSYNTH is motivated by the impossibility of manually responding to the large, growing number of reviews. AARSYNTH enables developers to enjoy the benefits of providing timely and relevant responses including improved ratings and wider adoption of their apps. Our experimental evaluation of AARSYNTH using a large corpus of reviews and responses from Google Play showed that it significantly outperforms the existing approaches: an improvement of 22.2% in BLEU-4 score over the state-of-the-art system. Moreover, human evaluators' ratings of the responses generated by AARSYNTH and other competing methods suggest that, with a statistically significant difference, AARSYNTH's responses are app-specific, better address the concerns raised in an input review, and are more linguistically fluent. The main novelty of AARSYNTH is augmenting seq2seq models with app-specific information, which is made possible due to utilizing information retrieval techniques and our fusion of seq2seq and machine reading comprehension neural architectures.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation under grants IIS-1838222 and IIS-1901379.

REFERENCES

[1] S. OĎea, "Number of smartphone users worldwide from 2016 to 2021," https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/, accessed: 2020-04-28.

[2] K. Glick, "Making it easier to respond to and improve user reviews," https://android-developers.googleblog.com/2019/05/whats-new-in-play.html, 2020, accessed: 2020-04-28.

[3] C. Gao, J. Zeng, X. Xia, D. Lo, M. R. Lyu, and I. King, "Automating app review response generation," in *34th IEEE/ACM International Conference on Automated Software Engineering*, 2019, pp. 163–175.

[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[5] W. Wang, N. Yang, F. Wei, B. Chang, and M. Zhou, "R-net: Machine reading comprehension with self-matching networks," *Microsoft Research Asia, Beijing, China, Tech. Rep*, vol. 5, 2017.

[6] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional attention flow for machine comprehension," *arXiv preprint:1611.01603*.

[7] S. Hochreiter and J. Schmidhuber, "Lstm can solve hard long time lag problems," in *Advances in neural information processing systems*, 1997, pp. 473–479.

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

[9] A. Lucene, "Apache lucene," https://lucene.apache.org/.

[10] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Communications of the ACM*, vol. 33, no. 6, pp. 668–676, 1990.

[11] R. Baeza-Yates and B. Ribeiro-Neto, "Modern information retrieval addison-wesley longman," *Reading MA*, 1999.

[12] S. E. Robertson and K. S. Jones, "Relevance weighting of search terms," *Journal of the American Society for Information science*, vol. 27, no. 3, pp. 129–146, 1976.

[13] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[14] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.

[15] S. Wiseman and A. M. Rush, "Sequence-to-sequence learning as beam-search optimization," *arXiv preprint:1606.02960*, 2016.

[16] S. Hassan, C. Tantithamthavorn, C.-P. 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.

[17] 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 *IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*, 2019, pp. 279–288.

[18] M. Danilák, "Port of googleś language-detection library to python," https://github.com/Mimino666/langdetect, 2020, accessed: 2020-04-28.

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

[20] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint:1409.0473*, 2014.

[21] A. Siddique, S. Oymak, and V. Hristidis, "Unsupervised paraphrasing via deep reinforcement learning," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1800–1809.

[22] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81.

[23] C.-Y. Lin and F. J. Och, "Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics," in *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, Barcelona, Spain, Jul. 2004, pp. 605–612.

[24] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint:1606.05250*, 2016.

[25] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng, "Ms marco: a human-generated machine reading comprehension dataset," 2016.

[26] 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*, 2018, pp. 373–384.

[27] C. Gao, "Repository for the rrgen," https://github.com/armor-ai/RRGen, 2020, accessed: 2020-04-28.

[28] N. Ketkar, "Introduction to pytorch," in *Deep learning with python*. Springer, 2017, pp. 195–208.

[29] R. Woolson, "Wilcoxon signed-rank test," *Wiley encyclopedia of clinical trials*, pp. 1–3, 2007.

[30] Z. Ji, Z. Lu, and H. Li, "An information retrieval approach to short text conversation," *arXiv preprint:1408.6988*, 2014.

[31] 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," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 267–281, 2019.

[32] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge university press, 2008.

[33] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[34] G. Chen, E. Tosch, R. Artstein, A. Leuski, and D. Traum, "Evaluating conversational characters created through question generation," in *Twenty-Fourth International FLAIRS Conference*, 2011.

[35] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens *et al.*, "Moses: Open source toolkit for statistical machine translation," in *Proceedings of the 45th annual meeting of the association for computational linguistics companion*, 2007, pp. 177–180.

[36] A. Ritter, C. Cherry, and W. B. Dolan, "Data-driven response generation in social media," in *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, 2011, pp. 583–593.

[37] O. Vinyals and Q. Le, "A neural conversational model," *arXiv preprint:1506.05869*, 2015.

[38] T. Hu, A. Xu, Z. Liu, Q. You, Y. Guo, V. Sinha, J. Luo, and R. Akkiraju, "Touch your heart: A tone-aware chatbot for customer care on social media," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18. ACM, 2018.

[39] W. Wu, X. Sun, and H. Wang, "Question condensing networks for answer selection in community question answering," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 1746–1755.

[40] S. Yoon, J. Shin, and K. Jung, "Learning to rank question-answer pairs using hierarchical recurrent encoder with latent topic clustering," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018, pp. 1575–1584.

[41] W. Yin, M. Yu, B. Xiang, B. Zhou, and H. Schütze, "Simple question answering by attentive convolutional neural network," *arXiv preprint:1606.03391*, 2016.

[42] Y. Tay, A. T. Luu, S. C. Hui, and J. Su, "Densely connected attention propagation for reading comprehension," in *Advances in Neural Information Processing Systems*, 2018, pp. 4906–4917.

[43] M. Dunn, L. Sagun, M. Higgins, V. U. Guney, V. Cirik, and K. Cho, "Searchqa: A new q&a dataset augmented with context from a search engine," *arXiv preprint:1704.05179*, 2017.

[44] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. Gall, F. Ferrucci, and A. De Lucia, "Recommending and localizing change requests for mobile apps based on user reviews," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. IEEE Press, 2017, p. 106–117.

[45] 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*. IEEE, 2018, pp. 72–83.

[46] U. Farooq and Z. Zhao, "Runtimedroid: Restarting-free runtime change handling for android apps," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, 2018.

[47] U. Farooq, Z. Zhao, M. Sridharan, and I. Neamtiu, "Livedroid: Identifying and preserving mobile app state in volatile runtime environments," in *Proceedings of the ACM on Programming Languages (PACMPL), Volume 4, Article 160, Issue OOPSLA, 2020*. ACM, 2020.

[48] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? on automatically classifying app reviews," in *IEEE 23rd international requirements engineering conference*. IEEE, 2015, pp. 116–125.

[49] T. Mike, B. Kevan, P. Georgios, and C. Di, "Sentiment in short strength detection informal text," *JASIST*, vol. 61, no. 12, pp. 2544–2558, 2010.

[50] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *IEEE 22nd international requirements engineering conference*. IEEE, 2014, pp. 153–162.

[51] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *2013 10th working conference on mining software repositories*. IEEE, 2013, pp. 41–44.