Technical Q&A Site Answer Recommendation via Question Boosting*

ZHIPENG GAO, Monash University, Australia
XIN XIA, Monash University, Australia
DAVID LO, Singapore Management University, Singapore
JOHN GRUNDY, Monash University, Australia

Software developers have heavily used online question and answer platforms to seek help to solve their technical problems. However, a major problem with these technical Q&A sites is *"answer hungriness"* i.e., a large number of questions remain unanswered or unresolved, and users have to wait for a long time or painstakingly go through the provided answers with various levels of quality. To alleviate this time-consuming problem, we propose a novel DeepAns neural network-based approach to identify the most relevant answer among a set of answer candidates. Our approach follows a three-stage process: question boosting, label establishment, and answer recommendation. Given a post, we first generate a clarifying question as a way of question boosting. We automatically establish the *positive*, *neutral*$^{+}$, *neutral*$^{-}$ and *negative* training samples via label establishment. When it comes to answer recommendation, we sort answer candidates by the matching scores calculated by our neural network-based model. To evaluate the performance of our proposed model, we conducted a large scale evaluation on four datasets, collected from the real world technical Q&A sites (i.e., Ask Ubuntu, Super User, Stack Overflow Python and Stack Overflow Java). Our experimental results show that our approach significantly outperforms several state-of-the-art baselines in automatic evaluation. We also conducted a user study with 50 solved/unanswered/unresolved questions. The user study results demonstrate that our approach is effective in solving the answer hungry problem by recommending the most relevant answers from historical archives.

CCS Concepts: • **Software and its engineering → Software evolution**; **Maintaining software**;

Additional Key Words and Phrases: CQA, Question Boosting, Question Answering, Sequence-to-sequence, Deep Neural Network, Weakly Supervised Learning

ACM Reference Format:
Zhipeng GAO, Xin Xia, David Lo, and John Grundy. 2019. Technical Q&A Site Answer Recommendation via Question Boosting. *ACM Trans. Softw. Eng. Methodol.* 9, 4, Article 39 (March 2019), 35 pages. https://doi.org/0000001.0000001

*Corresponding Authors: Xin Xia

Authors' addresses: Zhipeng GAO, Monash University, Melbourne, VIC, 3168, Australia, zhipeng.gao@monash.edu; Xin Xia, Monash University, Melbourne, VIC, 3168, Australia, xin.xia@monash.edu; David Lo, Singapore Management University, Singapore, Singapore, davidlo@smu.edu.sg; John Grundy, Monash University, Melbourne, VIC, 3168, Australia, john.grundy@monash.edu.

1049-331X/2019/3-ART39 $15.00
https://doi.org/0000001.0000001

1 INTRODUCTION

The past decade has witnessed significant social and technical value of Question and Answer (Q&A) platforms, such as Yahoo! Answers[1], Quora[2], and StackExchange[3]. These Q&A websites have become one of the most important user-generated-content (UGC) portals. For example, on the Stack Exchange forums, more than 17 million questions have been asked so far, and more than 11 million pages of these forums are visited daily by users. To keep up with the fast-paced software development process, the technical Q&A platforms have been heavily used by software developers as a popular way to seek information and support via the internet.

StackExchange is a network of online question and answer websites, where each website focuses on a specific topic, such as academia, Ubuntu operating system, latex, etc. There are a lot of technical Q&A sites which are heavily used by developers, such as Stack Overflow (with a focus on programming-related questions), Ask Ubuntu (with a focus on Ubuntu operating system), Super User (with a focus on computer software and hardware), and Server Fault (with a focus on servers and networks). These Q&A websites allow users to post questions/answers and search for relevant questions and answers. Moreover, if a post is not clear/informative, users routinely provide useful comments to improve the post. Fig. 1 shows an example of an initial post and its associated question comment in Ask Ubuntu Q&A site. By providing the question comment to the original post, it can assist potential helpers to write high quality answers since the question is more informative.

In spite of their success and active user participation, the phenomenon of being *"answer hungry"* is still one of the biggest issues within these Q&A platforms. This concept means that a very large number of questions posted remain *unanswered and/or unresolved*. According to our empirical study in different technical Q&A sites, Ask Ubuntu[4] and Super User[5], and Stack Overflow[6]. we found that (1) developers often have to wait a long time, spanning from days to even many weeks, before getting the first answer to their questions. Moreover, around 20% of the questions in Ask Ubuntu and Super User do not receive any answer at all and leave the askers unsatisfied; and (2) even with provided answers, about 44% questions in Ask Ubuntu and 39% questions in Super User are still unresolved, i.e., the question asker does not mark any answer as the accepted solution to their post. In such a case, information seekers have to painstakingly go through the provided answers of various quality with no certainty that a valid answer has been provided.

In this work, we aim to address this answer hungry phenomenon by recommending the *most relevant* answer or *the best* answer for an unanswered or unresolved question by searching from historical QA pairs. We refer to this problem as *relevant answer recommendation*. We propose a deep learning based approach we name DeepAns, which consists of three stages: *question boosting*, *label establishment* and *answer recommendation*. Given a post, our first step is to generate useful clarifying questions via a trained sequence-to-sequence model. The clarifying question is then appended to the original post as a way of question boosting, which can help eliminate the isolation between question and answers. Then, in the label establishment phase, for each enriched question, we pair it with its corresponding answers and automatically label the QA pair as *positive*, *neutral*$^+$, *neutral*$^-$ and *negative* samples by leveraging four heuristic rules. In the answer recommendation phase, given a question q and an answer candidate a_i, our goal is calculating the matching degree of the $\langle q, a_i \rangle$ pair. We formulate this problem as a four-category classification problem (i.e., a question and answer pair can be *positive*, *neutral*$^+$, *neutral*$^-$, or *negative* related). We propose a weakly

[1]https://answers.yahoo.com/
[2]https://www.quora.com/
[3]https://stackexchange.com
[4]https://askubuntu.com/
[5]https://superuser.com/
[6]https://stackoverflow.com/

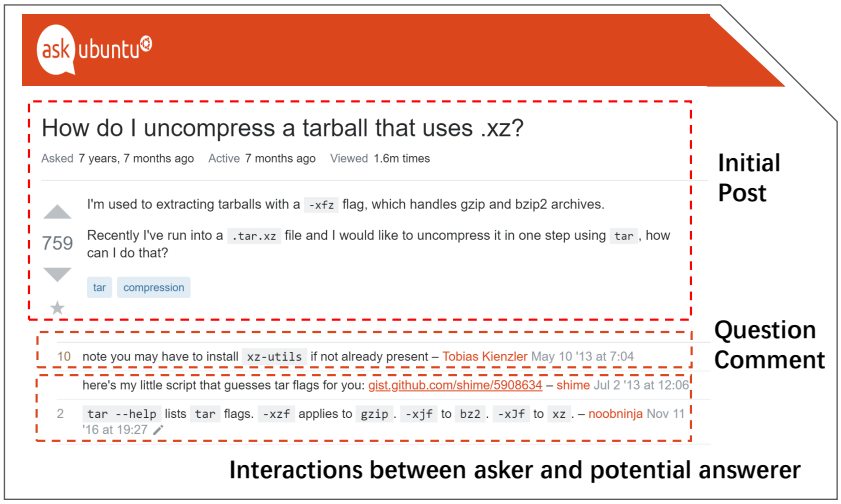

Fig. 1. Example Post on Askubuntu

supervised neural network that can be trained with the aforementioned four kinds of training samples.

The key usage scenarios of DeepAns are as follows: (1) for unresolved questions which do not have an asker-accepted answer, developers can use DeepAns to recommend the best answers; and (2) for unanswered questions, developers can use DeepAns to get the most relevant answers by mining answers to other related questions.

To evaluate the performance of our proposed approach, we conducted comprehensive experiments with four datasets, collected from the technical Q&A sites Ask Ubuntu, Super User and Stack Overflow respectively. The large-scale automatic evaluation results suggest that our model outperforms a collection of state-of-the-art baselines by a large margin. For human evaluation, we asked 5 domain experts for their feedback on our generated clarifying questions and answers. Our user study results further demonstrate the effectiveness and superiority of our approach in solving unanswered/unresolved questions. In summary, this paper makes the following contributions:

- Previous studies neglect the value of interactions between the question asker and the potential helper. We argue that a clarifying question between the question and answers is an important aspect of judging the relevance and usefulness of the QA pair. Therefore, we train a sequence-to-sequence model to generate useful clarifying questions for a given post, which can fill the lexical gap between the questions and answers. To the best of our knowledge, this is the first successful application of generating clarifying questions for technical Q&A sites.
- We present a novel method to constructing *positive*, *neutral*$^+$, *neutral*$^-$, *negative* training samples via four heuristic rules, which can greatly save the time consuming and labor intensive labeling process.
- We develop a weakly supervised neural network model for the answer recommendation task. For any question answer pairs, we fit the QA pair into our model to calculate the matching score between them; the higher matching score is estimated by our model, the better chance the answer will be selected as the best answer. In particular, the Q&A sites can employ our approach as a preliminary step towards marking the potential solution for the

unanswered/unresolved question. This can avoid unnecessary time spent by developers to browse questions without an accepted solution.
- Both our quantitative evaluation and user study show that DeepAns can help developers find relevant tehnical question answers more accurately, compared with state-of-the-art baselines. We have released the source code of DeepAns and the dataset[7] of our study to help other researchers replicate and extend our study.

The rest of the paper is organized as follows. Section 2 presents our empirical study of *answer hungry* problem in technical Q&A sites. Section 3 presents the details of our approach to identifying the most relevant answers. Section 4 presents the experimental set up and evaluation metrics. Section 5 presents the results of our approach on automatic evaluation. Section 6 presents the results of our approach on human evaluation. Section 7 discusses the strength of our approach and the threats to validity in our study. Section 8 presents key related work and techniques of this work. Section 9 concludes the paper with possible future work.

2 MOTIVATION

2.1 Answer Hungry Q&A Site Phenomenon

Despite of – or perhaps even because of – the success of technical Q&A sites, the *answer hungry* problem still widely exists in these online forums. We wanted to find out the degree of the problem for technical Q&A sites. To do this we quantitatively analyzed the prevalence of this *answer hungry* issue in real world technical Q&A sites (i.e., Ask Ubuntu, Super User and Stack Overflow). Since it is too expensive to run the empirical study on all the Stack Overflow dataset, we only focus on Python and Java related programming language questions in Stack Overflow for our experiment, which refer to *SO (Python)* and *SO (Java)* respectively in this study. The following two metrics are used in our experiment: (1) the proportion of questions that remain unanswered and/or unresolved within these technical Q&A sites, and (2) the time interval between the posting of one answer and its corresponding question.

To investigate the proportion of the unanswered and unresolved questions, we first counted the number of questions that have received at least one answer, and refer to these questions as *Answered Questions*. Questions not receiving any answers are referred to as *Unanswered Questions*. For those *Answered Questions*, we further divided them into two groups of *Resolved Questions* and *Unresolved Questions* based on whether any answer within the question thread has been marked or not as the accepted answer by the asker. Then, we empirically studied the average waiting time measured from the time of question creation to answer posting. We also calculated the average time interval for accepting an answer, which is the time difference between the time a question is created and the time an answer post is accepted. Table 1 presents the statistical results of our collected data[8]. From the table, we have the following observations:

(1) A large proportion of questions do not receive any answers in these technical Q&A sites. Consider Ask Ubuntu and Super User as examples – around 22% questions in Ask Ubuntu and 19% questions in Super User do not get any response since the time questions have been created, leaving the askers unsatisfied.
(2) A large amount of questions are still unresolved. For instance, 31.3% questions in SO (Python) and 35.4% questions in SO(Java) remain to be unresolved. This phenomenon is probably caused by the following reasons: (a) no good answer was provided within the current question thread, (b) even provided with good answers, it is common for the less experienced users to forget marking a potential answer as a solution.

[7] https://github.com/beyondacm/DeepAns
[8] For duplicated questions, we only keep the master ones, and remove the others.

Table 1. Answer Hungry Statistics

Ask Ubuntu	# Questions	315,924
	# Unanswered Questions	69,528
	# Resolved Questions	106,301
	# Unresolved Questions	140,095
	Avg Waiting Time	135.75 (days)
	Avg Accepting Time	18.63 (days)
Super User	# Questions	380,940
	# Unanswered Questions	73,584
	# Resolved Questions	160,200
	# Unresolved Questions	147,156
	Avg Waiting Time	173.03 (days)
	Avg Accepting Time	25.69 (days)
SO (Python)	# Questions	1,236,748
	# Unanswered Questions	175,859
	# Resolved Questions	674,360
	# Unresolved Questions	386,529
	Avg Waiting Time	103.97 (days)
	Avg Accepting Time	7.78 (days)
SO (Java)	# Questions	1,581,814
	# Unanswered Questions	213,963
	# Resolved Questions	808,040
	# Unresolved Questions	559,811
	Avg Waiting Time	100.52 (days)
	Avg Accepting Time	8.52 (days)

(3) Developers usually have to wait a long time before getting answers to their questions. It takes on average more than 135 days and 173 days to receive an answer in Ask Ubuntu and Super User sites respectively. The average time to accept an answer is much shorter, which are 18 and 25 days respectively. This further justifies our assumption that users may often forget to mark their accepted answers.
(4) The number of questions posted on Stack Overflow far outnumber the questions posted on Ask Ubuntu and Super User. At the same time, the ratio of the resolved questions in Stack Overflow are also higher than the other two technical Q&A sites. For instance, 54.5% questions in SO (Python) were resolved while the same number in Ask Ubuntu was 33.6%. This reflects that, compared with other technical Q&A sites, Stack Overflow is more popular and more frequently used by the information seekers.

In summary, the *answer hungry* phenomenon widely exists and has been one of the biggest challenges in technical Q&A forums.

2.2 Clarifying Questions in Technical Q&A Sites

Different from general Q&A sites, the comments within technical Q&A sites often include *clarifying questions*. In technical Q&A sites, the experts often ask clarifying questions to comments of a post so that they can understand the problem and help the one posting the question. We define

a "clarifying question" as a question in comments of a post that inquires of missing information for the given post. We wanted to empirically study the proportion and usefulness of clarifying questions in technical Q&A sites.

To investigate the proportion of the clarifying questions, we counted the number of comments on the questions as well as the number of comments containing clarifying questions. We extracted the clarifying questions as follows: We first constructed a Question Comment Set by extracting all the comments on the questions, removing the comments on the answers. Following that, for each comment in the Question Comment Set, we adopted sentence tokenization method from the NLTK toolkit [6] to break comment into multiple sentences. We then used the word tokenization method to separate each sentence into a list of tokens and symbols. If the extracted tokens contain the question mark token "?", we truncated the sentence till its question mark "?" to retrieve the question part of the comment as the clarifying question. If there are multiple clarification questions within the same comment, we kept them as separate clarifying questions. After that, we removed clarifying questions which are more than 20 words. The results are summarized in Table 2.

A clarifying question is useful if it helps in getting an answer to a specific question and/or reducing the waiting time. Imagine a scenario that Bob is a software developer who is seeking help in technical Q&A sites, he posts a question on these technical Q&A forums but the question remains unanswered for sometime. Following this, a clarifying question gets asked on the post and then Bob gets an answer. Such a user scenario can help to demonstrate the usefulness of clarifying questions. We estimated the usefulness by calculating the probability of a post getting answered with and without a clarifying question. The data were collected using the following steps:

(1) For a given question post, we removed it if the creator of the post responded to his or her own questions. There are around 10% of the posts being answered by the original question author in these CQA forums. For example, 39,811 and 48,503 questions were removed from the Ask Ubuntu and Super User, while 110,767 and 154,065 questions were removed from the SO (Python) and SO (Java) dataset respectively.
(2) Considering a question may not have enough time to receive answers if it is posted near the creation date of the data dump, we also removed the unanswered post if it is close (within 7 days) to the release date of the data dump. Since the data dump we used was created on September 5, 2019, we removed the unanswered questions posted after August 25, 2019. This results in 521 and 982 unanswered questions were removed from Ask Ubuntu and Super User, while 2,093 and 1,549 unanswered questions were removed from the SO (Python) and SO (Java) dataset respectively.
(3) For a given clarifying question, we removed it from the candidate list if the clarifying question is posted by the same user of the original question. For such a case, 8% of the clarifying questions were deleted from Ask Ubuntu and Super User, and 12% of the clarifying questions were deleted from SO (Python) and SO (Java) respectively.
(4) Considering a clarification question is helpful only if it was posted before the first answer provided on the thread, we checked the creation date of the clarification question as well as the first answer on the thread, and deleted all the clarification questions posted after the first answers. For example, 12,451 and 18,430 clarification questions were deleted from Ask Ubuntu and Super User, while 54,009 and 82,700 clarification questions were deleted from SO (Python) and SO(Java) dataset respectively.

Finally, we calculated the probabilities of a question receiving answers with and without a clarifying question. The probabilities are defined as follows:

Table 2. Clarifying Questions Statistics

Ask Ubuntu	# Question Comments	188,920
	# Clarifying Questions	72,359
	$\Pr(\mathbf{A}\|\mathbf{CQ})$	18.1%
	$\Pr\left(\mathbf{A}\|\overline{\mathbf{CQ}}\right)$	14.8%
Super User	# Question Comments	237,668
	# Clarifying Questions	96,296
	$\Pr(\mathbf{A}\|\mathbf{CQ})$	16.2%
	$\Pr\left(\mathbf{A}\|\overline{\mathbf{CQ}}\right)$	12.3%
SO (Python)	# Questions Comments	766,490
	# Clarifying Questions	329,768
	$\Pr(\mathbf{A}\|\mathbf{CQ})$	8.4%
	$\Pr\left(\mathbf{A}\|\overline{\mathbf{CQ}}\right)$	7.8%
SO (Java)	# Questions Comments	1,032,176
	# Clarifying Questions	467,772
	$\Pr(\mathbf{A}\|\mathbf{CQ})$	8.1%
	$\Pr\left(\mathbf{A}\|\overline{\mathbf{CQ}}\right)$	7.7%

$$P(\mathbf{A}|\mathbf{CQ}) = \frac{count(\mathbf{A}|\mathbf{CQ})}{count(\mathbf{A}|\mathbf{CQ}) + count\left(\overline{\mathbf{A}}|\mathbf{CQ}\right)} \tag{1}$$

$$P\left(\mathbf{A}|\overline{\mathbf{CQ}}\right) = \frac{count\left(\mathbf{A}|\overline{\mathbf{CQ}}\right)}{count\left(\mathbf{A}|\overline{\mathbf{CQ}}\right) + count\left(\overline{\mathbf{A}}|\overline{\mathbf{CQ}}\right)} \tag{2}$$

where $(\mathbf{A}|\mathbf{CQ})$ and $\left(\mathbf{A}|\overline{\mathbf{CQ}}\right)$ stands for answered posts with and without a clarifying question respectively. Similarly, $\left(\overline{\mathbf{A}}|\mathbf{CQ}\right)$ and $\left(\overline{\mathbf{A}}|\overline{\mathbf{CQ}}\right)$ stands for unanswered posts with and without a clarifying question respectively. The results are summarized in Table 2. From the table we have the following observations:

(1) In technical Q&A sites, a large number of comments on questions include clarifying questions. Since our method to extract clarifying questions is rather intuitive, we further sampled 100 clarifying questions from our dataset to do a manual analysis. By manually checking these clarifying questions, we found that 91% of the clarifying questions are positive clarifying questions. The positive clarifying questions often ask more information about the original post, such as "which version of ubuntu are you using?", and/or provide potential solutions to the original post, such as "do you use gnome or kde?". Only 9% of the clarifying questions are negative clarifying questions. The negative clarifying questions are often noisy and/or do not appear to provide any useful information for the original post, such as "did you resolve this?". These results show that a large proportion of clarifying questions are meaningful and informative.

(2) The likelihood of a post getting an answer with a clarifying question is higher than the likelihood of a post getting an answer without a clarifying question. For example in Ask

Ubuntu, without a clarifying question, the probability of a question post to receive answers dropped from 18.1% to 14.8%. This further justifies our assumption that the clarifying questions are helpful in improving the quality of the original post, hence increasing the chance of a question post receiving answers. This is why we employ clarifying questions to boost the original question post in our study.

In summary, clarifying questions appear frequently in technical Q&A sites and can help in improving the original question post and increasing the likelihood of questions to receive answers.

In this paper, we aim to invent a new model to not only help developers identify the best answers from a set of candidate answers when they perform QA search activities online, but also recommend the most relevant answers (given to other questions) when they initially post a question online. Mathematically, let q be the unanswered or unresolved question, let $(a_1, a_2, ..., a_N)$ be a set of answer candidates, our task is defined as finding the most relevant answer $a_i^*(i = 1, 2, 3, ...N)$, such that:

$$a^* = argmax_{a_i} P(\textbf{Accept}|\langle q, a_i\rangle) \tag{3}$$

$P(\textbf{Accept}|\langle q, a_i\rangle)$ corresponds to the probability a_i to be accepted given a QA pair $\langle q, a_i\rangle$.

3 OUR APPROACH

We present our approach named DeepAns, which ranks candidate answers from a relevant answer pool and recommends the most relevant answer to developers. In general, our model follows a three-stage process: *Question Boosting*, *Label Establishment*, and *Answer Recommendation*. Particularly, in the question boosting phase, DeepAns uses an attentional sequence-to-sequence recurrent neural network [46] to generate possible clarifying questions for a given post. These generated questions are appended to the original post as a way of *question boosting*. Then DeepAns automatically constructs *positive*, *neutral*$^+$, *neutral*$^-$ and *negative* training samples for each question and answer pair via four heuristic rules. In the answer recommendation phase, DeepAns trains another convolutional neural network to calculate the matching score between a given question and a candidate answer, the higher a similarity score is estimated, the more probable the answer will be selected as the best answer.

The underlying principle of applying the recurrent networks for the question boosting task is that compared with CNN neural networks, RNN architectures are dedicated sequence models, and this family of architectures has gained tremendous popularity to prominent applications, e.g., machine translation [5, 46]. For the answer recommendation task, we select the convolutional networks. Theoretically, we could also employ the recurrent networks for answer recommendation. However, due to the fact that computing score for each answer in the answer candidate pool is time-consuming, CNN architecture has better performance, lower perplexity, and more importantly, it runs much faster [12, 32] than RNN architecture for text encoding tasks, i.e., we can process all time steps in parallel via convolutional networks in both training and testing processes.

3.1 Question Boosting

The task of question boosting is to automatically generate clarifying questions from the title of an initial post. This can be formulated as a sequence-to-sequence learning problem. Given $\mathbf{Q}$ is a sequence of words within the question title of an initial post, our target is to generate a useful clarifying question $\mathbf{CQ}$, which is relevant, syntactically and semantically correct. To be more specific, the goal is to train a model θ using $\langle q, cq\rangle$ pairs such that the probability $P_\theta(\mathbf{CQ}|\mathbf{Q})$ is maximized over the given training dataset. Mathematically, this query boosting task is defined as finding $\overline{y}$, such that:

$$\overline{y} = argmax_{\mathbf{CQ}} P_\theta(\mathbf{CQ}|\mathbf{Q}) \tag{4}$$

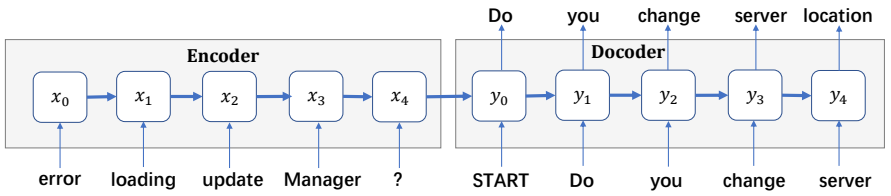

Fig. 2. Question boosting process

$P_\theta(\mathbf{CQ}|\mathbf{Q})$ can be seen as the conditional log-likelihood of the clarification question **CQ** given the input post **Q**. The encoder-decoder architecture has been used in addressing such a problem. We demonstrate an example of the question boosting process in Fig 2. The original post title "error loading update manager ?" is fed into the encoder, and the clarifying question "do you change server location ?" is the decoder target output.

3.1.1 Encoder. The sequence of words within a post title is fed sequentially into the encoder, which generates a sequence of hidden states. Our encoder is a two-layer bidirectional LSTM network,

$$\overrightarrow{\mathbf{f}w_t} = \overrightarrow{\text{LSTM}_2}\left(x_t, \overrightarrow{\mathbf{h}_{t-1}}\right)$$
$$\overleftarrow{\mathbf{b}w_t} = \overleftarrow{\text{LSTM}_2}\left(x_t, \overleftarrow{\mathbf{h}_{t-1}}\right)$$

where x_t is the given input word token at time step t, and $\overrightarrow{\mathbf{h}_t}$ and $\overleftarrow{\mathbf{h}_t}$ are the hidden states at time step t for the forward pass and backward pass respectively. The hidden states (from the forward and backward pass) of the last layer of the encoder are concatenated to form a state s as $\mathbf{s} = [\overrightarrow{\mathbf{f}w_t}; \overleftarrow{\mathbf{b}w_t}]$.

3.1.2 Decoder. Decoder is singe layer LSTM network, initialized with the state s as $\mathbf{s} = [\overrightarrow{\mathbf{f}w_t}; \overleftarrow{\mathbf{b}w_t}]$. Let $qword_t$ be the target word at time stamp t of the clarifying question. During training, at each time step t the decoder takes as input the embedding vector y_{t-1} of the previous word $qword_{t-1}$ and the previous state s_{t-1}, and concatenates them to produce the input of the LSTM network. The output of the LSTM network is regarded as the decoder hidden state s_t, as follows:

$$\mathbf{s}_t = \text{LSTM}_1\left(y_{t-1}, \mathbf{s}_{t-1}\right) \tag{5}$$

The decoder produces one symbol at a time and stops when the *END* symbol is emitted. The only change with the decoder at testing time is that it uses output from the previous word emitted by the decoder in place of $word_{t-1}$(since there is no access to a ground truth then).

3.1.3 Attention Mechanism. To effectively align the target words with the source words, we model the attention [5] distribution over words in the target sequence. We calculate the attention (a_i^t) over the i^{th} input token as :

$$e_i^t = v^t \tanh\left(W_{eh} h_i + W_{sh} s_t + b_{att}\right) \tag{6}$$
$$a_i^t = \text{softmax}\left(e_i^t\right) \tag{7}$$

Here v^t, W_{sh} and b_{att} are model parameters to be learned, and h_i is the concatenation of forward and backward hidden states of source-code encoder. We use this attention a_i^t to generate the context vector c_t^* as the weighted sum of encoder hidden states :

$$\mathbf{c}_t^* = \sum_{i=1,..,|\mathbf{x}|} a_i^t \mathbf{h}_i \tag{8}$$

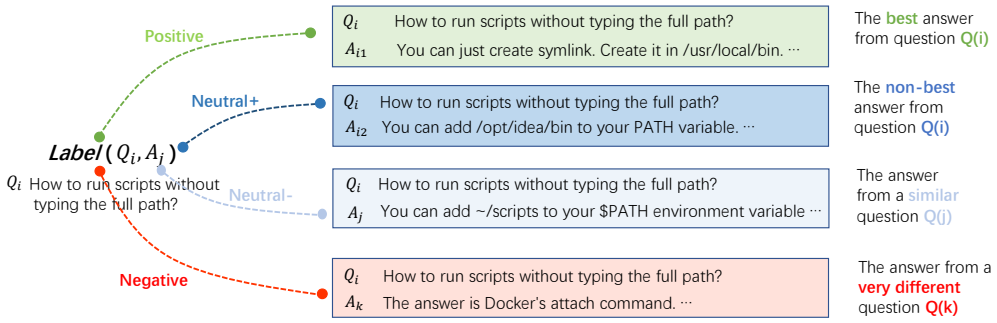

Fig. 3. Label Establishing Process

We further use the c_t^* vector to obtain a probability distribution over the words in the vocabulary as follows,

$$P = \text{softmax}\left(\mathbf{W}_v[s_t, c_t^*] + b_v\right) \tag{9}$$

where W_v and b_v are model parameters. Thus during decoding, the probability of a word is $P(qword)$. During the training process for each word at each timestamp, the loss associated with the generated question is :

$$Loss = -\frac{1}{T}\sum_{t=0}^{T} logP(qword_t) \tag{10}$$

Once the model is trained, we do inference using beam search [19] and append the generated clarifying question to the original post title. The beam search is parameterized by the possible paths number k. The inference process stops when the model generates the END token, which stands for the end of the sentence.

3.2 Label Establishment

According to our empirical study results from Section 2, the *answer hungry* phenomenon widely exists in technical Q&A forums, i.e. only a small proportion of questions have an "resolved" answer, while many others remain unanswered and/or unresolved. Due to the reason of professionality of technical questions, only the experts with specific knowledge are qualified to evaluate the matching degree between a question and an answer. Therefore it is very hard to find such annotators and/or the creation of training sets requires a substantial manual effort. To address such a problem, We propose a novel scheme to automatically labeling each QA pair as *positive*, *neutral*$^+$, *neutral*$^-$, and *negative* samples. Fig 3 shows an example of our labeling process. We propose four heuristic rules to label the QA pairs:

- *Positive samples*: for a given question Q_i, we pair it with its marked "best" answer (if it has one) A_{i1}, and label this qa pair as *Positive*.
- *Neutral*$^+$ *samples*: for a given question Q_i, we pair it with its non-best answer (answers within the same question thread, except the one marked as the best answer), and label this qa pair as *Neutral*$^+$.
- *Neutral*$^-$ *samples*: for a given question Q_i, we randomly select one answer A_j from questions similar to it, then label this question-answer pair as *Neutral*$^-$.

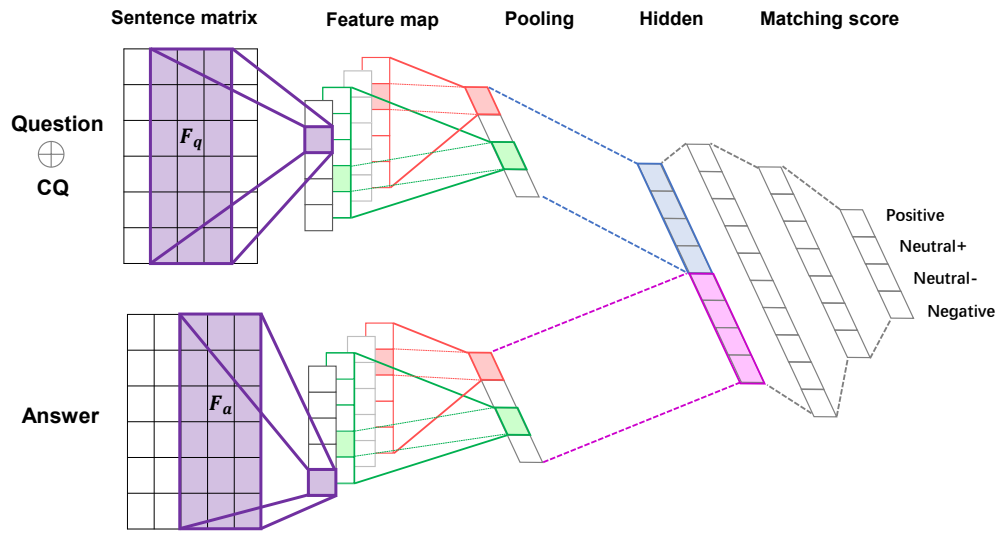

Fig. 4. Overall architecture of the answer recommendation model.

- *Negative samples*: for a given question Q_i, we pair it with a randomly selected answer A_k from non-similar questions and label this QA pair as *Negative*.

Since we are recommending answers from candidate answers of questions relevant to the query question, if the retrieved questions are not relevant to the query question, it is unlikely we can select the best answer from the answer candidates pool. We followed the question retrieval method proposed by Xu et al. [50] to search for similar questions, which has been proven to be more effective for this task of relevant question retrieval. We used the IDF-weighted word embedding to calculate the similarity score between the query and the question title. Thereafter, a set of similar questions can be identified by selecting the top-k ranked questions.

After this label establishing process, we can gather large amounts of labeled examples, which greatly saves the time-consuming and labor-intensive labeling process.

3.3 Answer Recommendation

After collecting large amounts of labeled training data via label establishment, we are able to train the deep learning model based on the four kinds of training samples.

We present a weakly supervised neural network architecture for ranking QA pairs. Fig. 4 demonstrates the workflow of our proposed model. The main building blocks of our architecture are two convolutional neural networks [29, 32]. These two underlying sub-models work in parallel, mapping questions and answers to their distributional vectors respectively, which are then used to calculate the final similarity score between them.

3.3.1 Sentence Matrix. The input to our model are $\langle q \oplus cq, a \rangle$ pairs, where q and a stands for the question and answer of a labelled QA pair, cq stands for the clarifying questions generated by our question boosting model. The questions (including the original questions and clarifying questions) and answers are parallel sentences, where each sentence is treated as a sequence of words: $(w_1, ..., w_s)$, where each word is drawn from a vocabulary $\mathbf{V}$. Words are represented by distributional vectors $\mathbf{w} \in \mathbb{R}^{1\times d}$ via looking up in a pre-trained word embedding matrix $\mathbf{W} \in \mathbb{R}^{d\times|V|}$.

For each input $\langle q \oplus cq, a \rangle$ pair, we build two sentences matrix $\mathbf{S_q}$ and $\mathbf{S_a} \in \mathbb{R}^{d \times |s|}$ for each question and answer respectively, where the ith column represents the word embedding of $\mathbf{w_i}$ at corresponding position i in a sentence.

3.3.2 Convolutional feature maps. To learn to capture and compose features of individual words in a given sentence from low-level word embeddings into higher level semantic concepts, we apply two identical convolutional neural network blocks to the input sentence matrix $\mathbf{S_q}$ and $\mathbf{S_a}$ respectively.

More formally, the convolution operation $*$ between an input sentence matrix $\mathbf{S_{q/a}} \in \mathbb{R}^{d \times |s|}$ and a filter $\mathbf{F} \in \mathbb{R}^{d \times m}$ (called a filter of size m) results in a vector $\mathbf{c} \in \mathbb{R}^{|s|-m+1}$, where each component is computed as follows:

$$\mathbf{c_i} = (\mathbf{S} * \mathbf{F})_i = \sum_{k,j} \left(\mathbf{S_{[:,i-m+1:i]}} \otimes \mathbf{F}\right)_{kj} \tag{11}$$

In the above equation, $\otimes$ is the element-wise multiplication and $\mathbf{S_{[:,i-m+1:i]}}$ is a matrix slice of size m along the columns. Note that the convolution filter is of the same dimensionality d as the input sentence matrix. As shown in Fig. 4, it slides along the column dimension of $\mathbf{S}$ producing a vector $\mathbf{c} \in \mathbb{R}^{|s|-m+1}$. Each component $\mathbf{c_i}$ is the result of computing an element-wise product between a column slice of $\mathbf{S}$ and the filter matrix $\mathbf{F}$, which is then flattened and summed producing a single value. By applying a set of filters (called a filter bank) $\mathbf{F} \in \mathbb{R}^{n \times d \times m}$ to sequentially convolved with the sentence matrix $\mathbf{S}$ will generate a convolutional feature map matrix $\mathbf{C} \in \mathbb{R}^{\mathbf{n} \times (|\mathbf{s}|-\mathbf{m}+\mathbf{1})}$.

3.3.3 Pooling layer. Following that, we pass the output from the convolutional layer to the pooling layer, whose goal is to aggregate the information and reduce the representation. We apply a max pooling operation [11] over the convolutional feature map and take the maximum value $\widehat{\mathbf{c}} = max\{\mathbf{c_i}\}$ as the feature corresponding to a particular filter. The idea is to capture the most important feature - one with the highest value - for each feature map.

3.3.4 Matching score layer. The output of the penultimate convolutional and pooling layers x is passed to a series of fully connected layer followed by a softmax layer. It computes the probability distribution over the four kinds of labels (*positive*, *neutral*$^+$, *neutral*$^-$, *negative*):

$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^T \theta_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^T \theta_k}} \tag{12}$$

where θ_k is a weight vector of the k-th class. $\mathbf{x}$ can be thought of as a final abstract representation of the input QA pair obtained by a series of transformations from the input layer through a series of convolutional and pooling operations.

For the final matching score, we want this score to be high if the input qa pair is *positive* and *neutral*$^+$, and to be low if it is *negative* and *neutral*$^-$. Therefore we define the calculation of the similarity score as follows:

$$Score = \omega_{pos} \times P(pos) + \omega_{neu+} \times P\left(neu^+\right) - \omega_{neu-} \times P\left(neu^-\right) - \omega_{neg} \times P(neg) \tag{13}$$

Algorithm 1: DeepAns Algorithm (Offline Training)

Input: Data dump of technical Q&A sites;
Output: 1.Question Boosting model; 2.Answer recommendation model;

Extract $\langle q, cq \rangle$ pairs from data dump;
Train $\langle q, cq \rangle$ pairs with attentional-based seq2seq model ;
Save the model as Question Boosting model ;
Extract $\langle q, a \rangle$ pairs from data dump;
for $q_i, a_i \in \langle q, a \rangle pairs$ **do**
    if q_i *has accepted-answer* **then**
        if a_i *is accepted-answer* **then**
            Label $\langle q_i, a_i \rangle$ as $Positive$;
        end
        else
            Label $\langle q_i, a_i \rangle$ as $Neutral^+$;
        end
        Select similar answer a_j then Label $\langle q_i, a_j \rangle$ as $Neutral^-$;
        Select random answer a_k then Label $\langle q_i, a_k \rangle$ as $Negative$;
    end
end
for $q_i \in labelled \langle q, a \rangle pairs$ **do**
    Generate cq_i for q_i using Question Boosting model;
    Append cq_i to q_i to make labelled $\langle q_i \oplus cq_i, a_i \rangle$ pair ;
end
Train labelled $\langle q \oplus cq, a \rangle$ pairs with CNN-based classification model ;
Save the model as Answer Recommendation model

Algorithm 2: DeepAns Algorithm (Online Recommendation)

Input: User search query q_{user};
Output: A set of candidate answers with a matching score for each answer ;

Generate cq for q_{user} using Question Boosting model ;
Search top-k similar questions for the given query q_{user} ;
Add top-k questions to similar question set SQ ;
for $q_i \in SQ$ **do**
    for $a_j \in q_i$ **do**
        Add answer to candidate answers set CA ;
    end
end
for $a_i \in CA$ **do**
    Pair a_i with expanded query to make a $\langle q_{user} \oplus cq, a_i \rangle$ pair ;
    Fit $\langle q_{user} \oplus cq, a_i \rangle$ pair to Answer Recommendation model ;
    Compute the final matching score s_i via Equation 13
end
Rerank answers in CA via matching scores

There are four weights as shown in Equation 13. We initially set all the four weights to 1 at the beginning. Then the optimal settings of these weights are carefully tuned on our validation set (detailed in Section 5.3.1). We use the final matching score to measure the relevance between a question and an answer.

3.4 DeepAns Algorithm

We divide our model into two components: offline training and online recommendation. The detailed algorithms of *DeepAns* for offline training and online recommendation are presented in Algorithm 1 and Algorithm 2 respectively. To be more specific, during the offline training, we use the data from technical Q&A sites to train the question boosting model (lines 1-3) and answer recommendation model (lines 4-20). When it comes to the online recommendation, for a given user query, we first collect a pool of answer candidates via finding its similar questions (lines 1-8). After that, we use the trained question boosting model to perform query expansion, then pair it with each of the answer candidates and fit them into the trained answer recommendation model to estimate their matching scores (lines 9-14).

4 AUTOMATIC EVALUATION EXPERIMENT SETUP

In this section, we first describe the data sets used throughout our experiments. We then discuss the baselines we compare to our new *DeepAns* approach and our experimental settings. Lastly, we explain the automatic evaluation process.

4.1 Data Preparation

We collected data from the official dump of StackExchange, a network of online question and answer websites. The StackExchange data dump contains timestamped information about the posts, comments as well as the revision history made to the post. Each post comprises a short question title, a detailed question body, corresponding answers and multiple tags. For each post, users can add clarifying questions to posts for further discussion. After receiving one or more answers, the asker can select one answer that is most suitable for their question as the accepted/best answer. We choose three different technical Q&A sites, i.e., Ask Ubuntu, Super User and Stack Overflow for our experiment. These three technical Q&A sites are commonly used by software developers and each one focuses on a specific area. For instance, Ask Ubuntu and Super User focus on Ubuntu system questions and computer software/hardware questions respectively, and Stack Overflow is the most popular programming related Q&A site which has been heavily used by software developers via the internet. As with our previous empirical study, we only focus on the Python and Java related questions in Stack Overflow for this study, referred to as SO (Python) and SO (Java) respectively in this study.

The experimental dataset creation process is divided into three phases: extracting $\langle q, cq \rangle$ pairs, constructing labelled $\langle q, a \rangle$ pairs, and constructing labelled $\langle q \oplus cq, a \rangle$ pairs, where q stands for the question, cq stands for the clarifying question, and a stands for the answer. Table 3 describes the statistics of our collected datasets.

(1) **Extract $\langle q, cq \rangle$ Pairs**: For each post, we follow the methods described in Section 2.2 to extract the clarifying questions. According to our manual analysis results, we summarize a list of keywords associated with non-clarifying questions, such as “edit”, “related”, “vote”, etc. We preprocess our dataset to remove all instances that involve such keywords. We also summarize a list of key phrases associated with the clarifying questions, such as “do you”, “have you”, “how”, “which”, etc. We retained the pairs that include the above key phrases. After that, we pair the original post with its associated clarifying question as $\langle q, cq \rangle$ pairs.

Table 3. Number of Training/Validation/Testing Samples

Ask Ubuntu	# $\langle q, cq \rangle$ pairs	68,216	# $\langle q, a \rangle$ pairs	289,062
	# *Positive* pairs	79,726	# *Neutral*$^+$ pairs	49,884
	# *Neutral*$^-$ pairs	79,726	# *Negative* pairs	79,726
Super User	# $\langle q, cq \rangle$ pairs	87,081	# $\langle q, a \rangle$ pairs	447,221
	# *Positive* pairs	119,305	# *Neutral*$^+$ pairs	89,306
	# *Neutral*$^-$ pairs	119,305	# *Negative* pairs	119,305
SO (Python)	# $\langle q, cq \rangle$ pairs	311,127	# $\langle q, a \rangle$ pairs	2,372,232
	# *Positive* pairs	610,948	# *Neutral*$^+$ pairs	539,388
	# *Neutral*$^-$ pairs	610,948	# *Negative* pairs	610,948
SO (Java)	# $\langle q, cq \rangle$ pairs	456,077	# $\langle q, a \rangle$ pairs	3,013,859
	# *Positive* pairs	734,977	# *Neutral*$^+$ pairs	808,928
	# *Neutral*$^-$ pairs	734,977	# *Negative* pairs	734,977

We extract a total of 68,216 pairs in Ask Ubuntu, and 87,081 pairs in Super User. The number of $\langle q, cq \rangle$ pairs in Stack Overflow are much larger, we obtain a total of 311,127 pairs for SO (Python) and 456,077 pairs for SO (Java). These collected $\langle q, cq \rangle$ pairs are used to train a sequence-to-sequence model for question boosting.

(2) **Construct labelled $\langle q, a \rangle$ Pairs**: To make the $\langle q, a \rangle$ pairs, we first extract the questions that having explicitly marked accepted answers. Then for each question, we pair it with the accept answer to make the *positive sample*, with non-accepted answer to make the *neutral*$^+$ *sample*, with an answer to a similar question to make the *neutral*$^-$ *sample*, and with an answer to a randomly selected question to make the *negative sample*. We have to clarify that some questions do not have the non-accepted answers; this is the reason why the number of *neutral*$^+$ samples is smaller than the number of other samples, such as Ask Ubuntu, Super User and SO (Python), while some other questions have more than one non-accepted answers, which results in the number of *neutral*$^+$ samples is bigger than those of the rest, such as SO (Java). For the final dataset, we construct 289,062 and 447,221 $\langle q, a \rangle$ labelled pairs for Ask Ubuntu and Super User, and 2,372,232 and 3,013,859 $\langle q, a \rangle$ labelled pairs for SO (Python) and SO (Java) respectively. It is obvious that the number of qa pairs in Stack Overflow far outnumber those of other technical Q&A sites. After the label establishment process, we largely expand the labelled dataset for training. We randomly sample 5,000 questions for validation and 5,000 questions for testing respectively, and kept the rest for training. It is worth mentioning that we first used the validation set for model selection regarding the accuracy of QA pairs classification results, which is a middle result of the answer selection target. After that, we reused the validation set for tuning the four weights as shown in Equation 13. The testing set was used only for testing the final solution to confirm the actual predictive power of our model with optimal parameter settings.

(3) **Construct labelled $\langle q \oplus cq, a \rangle$ Pairs**: For each labelled $\langle q, a \rangle$ pair, we feed the original question to the trained question boosting model to generate a clarifying question. After that, we append the clarifying question to the original question to construct the $\langle q \oplus cq, a \rangle$ pairs. The number of the $\langle q \oplus cq, a \rangle$ pairs is identical with the number of $\langle q, a \rangle$ pairs.

4.2 Implementation Details

We implemented our *DeepAns* system in Python using the PyTorch framework. The main parameters of our deep learning model (tuned using the validation dataset) were as follows:

- Question Boosting: We train an attentional sequence-to-sequence model for this subtask. Previous studies have shown that the deep sequence-to-sequence model can achieve state-of-the-art performance on different tasks [17, 24, 27, 44]. We also used the parameter settings from [17] for training the $\langle q, cq \rangle$ pairs in this study. We use a two-layer bidirectional LSTM for the encoder and a single-layer LSTM for the decoder. We set the number of LSTM hidden states to be 256 in both encoder and decoder. Optimization is performed using stochastic gradient descent (SGD) with a learning rate of 0.01. During decoding, we perform beam search with a beam size of 10.
- Answer Recommendation: Kim et al. [32] have shown that convolutional neural networks trained on top of pre-trained word vectors achieved promising performance for sentence-level classification tasks. Hence in our work, we also followed the experiment settings of their studies. We initialize the word embeddings from our unsupervised corpus and set the dimension of word embedding d to 100. The width m of the three convolution filters is set to 3, 4, 5 and the number of convolution feature maps is set to 100. We use ReLu activation function and a simple max-pooling function. The size of the hidden layer is equal to the size of the join vector obtained after concatenating question and answer vectors from the distributional models.

To train both networks, we used stochastic gradient descent with shuffled mini-batches. The batch size is set to 64. Both network are trained for 50 epochs with early stopping, i.e., we stop the training if no update to the best accuracy on the validation set has been made for the last 5 epochs.

4.3 Baselines

To demonstrate the effectiveness of our proposed DeepAns, we compared it with several comparable systems. We briefly introduced these and how they are used for the task of predicting the best answer among a set of answer candidates. DeepAns is built with the semantic features of words in their dimensions, we used the average word vector of a sentence as features for training all of the baseline models for a fairer comparison. For each baseline method, their parameters were carefully tuned, and the parameters with the best performance were used to report the final comparison results with our *DeepAns* approach on the same datasets:

- **Learning to Rank** The answer prediction problem of our task is similar to the traditional ranking task [42] [2], where the given question and a set of answer candidates are analogous to a query and a set of relevant entities. Hence our task is transformed to find an optimal ranking order of these answer candidates according to their relevance to a given question. We choose the AdaRank[51] and LambdaMART [7] as the baseline learning-to-rank methods for our task. We used the *positive*, *neutral*+ as the target value to define the order of each example. This is reasonable because the label establishment is part of our model, and the heuristic rules for setting up the *neutral*– and *negative* samples are never used before.
- **Traditional Classifiers** Recently Calefato et al. [9] proposed to approach the best answer prediction problem as a binary-classification task, and in their work they assessed 26 best-answer prediction classifiers in Stack Overflow. We choose the two most effective traditional classifiers from their experimental results, xgbTree and RandomForest, for use in our study. As they were doing binary classification, to adapt to our training data, we kept our *positive samples* as positive and consider *neutral*$^+$ samples as negative. Thereafter, we utilize the classification models to generate an answer ranking list by pairwise comparison between the answer candidates.

- **AnswerBot** Xu et al. [50] proposed a framework called AnswerBot to generate an answer summary for a non-factoid technical question. Their user study showed a promising performance for selecting salient answers by their method. We adapted their AnswerBot approach for our task of recommending answers among a set of answer candidates. To be more specific, for a given question, AnswerBot generates a ranked list of candidate answers according to the ranking scores. This ranked list of answers is then used to calculate the precision of answer selection results.

- **IR-DeepAns** To verify the effectiveness of using clarifying questions as a way of question boosting, compared with our sequence to sequence model, we also considered a simple IR-based approach using similar clarifying questions as a query expansion mechanism. For a given question q_i, we first identified the most similar question q_j in $\langle q, cq \rangle$ dataset, and then retrieved the clarifying question cq_j associated with q_j. We applied IDF-weighted word embedding methods to calculate the similarity score between two questions. We feed the q_i and cq_j into our model and name this baseline as IR-DeepAns. This model is close to ours.

4.4 Evaluation Methods

4.4.1 Experiment Setup. To thoroughly evaluate our model, we conducted a large-scale automatic evaluation experiment. We used IDF-weighted word embedding (described in Section 3.2) to calculate the similarity score between two question titles. For each testing qa pair $\langle q_t, a_t \rangle$, we then performed K-NN (K=5) to search for similar questions over the whole data set for the given testing question q_t. We then constructed an answer candidate pool by gathering the top-5 answers associated with these selected questions. Since the top-similar question extracted by K-NN is always the original post itself, we can ensure that the accepted answer a_t paired with the original post q_t is always in the answer candidate pool. In other words, the answer candidate pool for testing question q_t contains 5 answers, one of which is the accepted answer a_t. In summary, for the 5,000 testing questions of each platform, we constructed 5,000 $\times$ 5 QA pairs in total to serve as the final evaluation sets. Following this, for each testing question q_t, we first applied the pre-trained question boosting model to generate a clarifying question cq_t. We then paired the given question with each answer in the candidate pool to construct the $\langle q_t \oplus cq_t, a_t \rangle$ pairs. The $\langle q_t \oplus cq_t, a_t \rangle$ pair was fitted into our model to calculate a matching score, and we then generated a ranking order for each group of candidate answers according to their matching scores to the given question.

4.4.2 Evaluation Metrics. Since the evaluation answer candidate pool includes the accepted answer, one way to evaluate our approach is to look at how often the accepted answer is ranked higher up among members of the answer candidate pool. Thus we adopted the widely-accepted metric, *P@K* and *DCG@K* to measure the ranking performance of our model.

- *P@K* is the precision of the best answer in top-K candidate answers. Given a question, if one of the top-k ranked answers is the best answer, we consider the recommendation to be successful and set $success(best_i \in topK)$ to 1, otherwise, we consider the recommendation to be unsuccessful and set $success(best_i \in topK)$ to 0. The *P@K* metric is defined as follows:

$$P@K = \frac{1}{N} \sum_{i=1}^{N} [success(best_i \in topK)] \tag{14}$$

- *DCG@K* is another popular top-K accuracy metric that measures a recommender system performance based on the graded relevance of the recommended items and their positions in the candidate set. Different from *P@K*, the intuition of *DCG@K* is that highly-ranked items are more important than low-ranked items. According to this metric, a recommender system

gets a higher reward for ranking the correct answer at a higher position. The $success(best_i \in topK)$ is same with the previous definition, while the $rank_{best_i}$ is the ranking position of the best answer i. The $DCG@K$ is defined as follows:

$$DCG@K = \frac{1}{N}\sum_{i=1}^{N}\frac{[success(best_i \in topK)]}{\log_2(1 + rank_{best_i})} \tag{15}$$

5 AUTOMATIC EVALUATION RESULTS

To gain a deeper understanding of the performance of our approach, we conducted an analysis on our large-scale automatic evaluation results. Specifically, we mainly focus on the following research questions:

- *RQ-1:* How effective is our approach under automatic evaluation?
- *RQ-2:* How effective is our use of *Question boosting* and *Label establishing* methods?
- *RQ-3:* How effective is our approach under different parameter settings?

5.1 RQ-1: Automatic Evaluation Results Analysis

The automatic evaluation results of our proposed model and aforementioned baselines over different technical Q&A sites are summarized in Table 4, Table 5, Table 6 and Table 7 respectively. We do not report $P@5$ and $DCG@1$ in our tables, since $DCG@1$ is always equal to $P@1$ and $P@5$ will always be equal to 1. The best performing system for each column is highlighted in boldface. As can be seen, **our model outperforms all the other methods by a large margin** in terms of $P@K$ score and $DCG@K$ score. From the table, we can observe the following points discussed below.

(1) Compared to traditional classifiers, such as xbgTree and RandomForest, one can clearly see that our approach performs much better. For example, it improves over xgbTree on $P@1$ by 42% on Ask Ubuntu dataset, and 39% on Super User dataset.

(2) Compared with the method proposed by [9], which only has two kinds of labels (*positive* and *negative*), our approach constructs four kinds of labeled data (*positive*, $neutral^+$, $neutral^-$, *negative*) automatically via incorporating the *label establishing* process. By introducing the $neutral^+$ and $neutral^-$ training samples, our approach can learn how to separate the best answer from the similar ones, which may explain the obvious advantage of our model in $P@1$.

(3) Our approach also outperforms the AnswerBot by a large margin. We attribute this to the following reasons. Firstly, by adding a clarifying question into our model, we can properly fuse the information between the isolated question sentences and answers, which can reduce the lexical gap between them and better pair the answer with associated questions. Secondly, we use two parallel convolutional neural network block to learn optimal vector representation of QA pairs that preserving important syntactic and semantic features. To compute the matching score, we relate the rich representation features via a weakly supervised way from the available training data.

Table 4. Automatic evaluation (Ask Ubuntu)

Model	P@1	P@2	P@3	P@4	DCG@2	DCG@3	DCG@4	DCG@5
RandomForest	26.6 ± 1.6%	49.2 ± 1.6%	70.8 ± 1.6%	87.1 ± 0.4%	40.9 ± 1.5%	51.7 ± 1.5%	58.8 ± 0.9%	63.7 ± 0.8%
XgbTree	28.8 ± 1.4%	53.6 ± 1.3%	73.0 ± 1.0%	87.9 ± 1.2%	44.5 ± 1.2%	54.2 ± 0.9%	60.7 ± 0.8%	65.3 ± 0.7%
LambdaMART	25.4 ± 1.1%	45.7 ± 1.0%	65.7 ± 1.2%	84.0 ± 1.0%	38.5 ± 1.0%	47.5 ± 1.1%	55.8 ± 0.9%	62.3 ± 0.6%
AdaRank	24.9 ± 1.1%	45.3 ± 1.1%	65.0 ± 1.0%	82.9 ± 0.8%	38.1 ± 1.2%	47.2 ± 1.1%	55.2 ± 1.0%	61.8 ± 0.7%
AnswerBot	27.7 ± 1.6%	52.1 ± 1.5%	73.5 ± 1.0%	89.2 ± 0.7%	43.1 ± 1.5%	53.8 ± 1.1%	60.5 ± 0.8%	64.7 ± 0.8%
DeepAns-IR	37.2 ± 2.0%	59.9 ± 2.1%	77.5 ± 1.7%	92.0 ± 1.0%	50.8 ± 1.5%	59.6 ± 1.3%	65.8 ± 1.1%	68.7 ± 0.8%
DeepAns	**40.9 ± 1.5%**	**61.7 ± 1.9%**	**77.9 ± 0.9%**	**92.0 ± 0.9%**	**54.0 ± 1.7%**	**62.1 ± 1.1%**	**68.2 ± 1.1%**	**71.3 ± 0.9%**

Table 5. Automatic evaluation (Super-User)

Model	P@1	P@2	P@3	P@4	DCG@2	DCG@3	DCG@4	DCG@5
RandomForest	27.4 ± 1.6%	50.2 ± 1.7%	70.5 ± 1.4%	87.3 ± 0.9%	41.7 ± 1.6%	51.9 ± 1.4%	59.2 ± 1.2%	64.1 ± 0.9%
XgbTree	29.2 ± 1.6%	55.9 ± 1.3%	74.2 ± 0.9%	88.5 ± 0.7%	47.6 ± 1.2%	56.7 ± 1.1%	63.0 ± 0.9%	67.4 ± 0.8%
LambdaMART	25.9 ± 1.1%	47.1 ± 1.0%	66.1 ± 1.0%	84.5 ± 1.2%	39.8 ± 1.0%	48.8 ± 1.0%	56.4 ± 0.7%	62.9 ± 0.6%
AdaRank	25.1 ± 1.2%	46.2 ± 1.1%	65.7 ± 1.0%	84.1 ± 0.9%	38.4 ± 1.1%	47.6 ± 1.1%	55.7 ± 1.0%	62.2 ± 0.9%
AnswerBot	29.8 ± 1.4%	53.9 ± 1.2%	74.1 ± 1.3%	89.6 ± 0.8%	45.0 ± 1.2%	55.1 ± 0.9%	61.8 ± 0.7%	65.8 ± 0.6%
DeepAns-IR	38.8 ± 2.1%	63.4 ± 1.8%	80.7 ± 1.2%	92.5 ± 1.2%	54.3 ± 1.8%	63.0 ± 1.3%	68.1 ± 1.2%	70.9 ± 1.0%
DeepAns	**40.7 ± 1.9%**	**65.8 ± 1.1%**	**82.2 ± 1.1%**	**93.9 ± 0.8%**	**56.5 ± 1.2%**	**64.7 ± 1.2%**	**69.8 ± 1.0%**	**72.1 ± 0.8%**

Table 6. Automatic evaluation (SO-Python)

Model	P@1	P@2	P@3	P@4	DCG@2	DCG@3	DCG@4	DCG@5
RandomForest	34.0 ± 1.3%	57.2 ± 1.0%	74.8 ± 0.7%	89.7 ± 0.6%	48.6 ± 0.9%	57.4 ± 0.6%	63.9 ± 0.7%	67.8 ± 0.5%
XgbTree	35.4 ± 1.5%	58.4 ± 1.9%	74.2 ± 1.5%	88.7 ± 1.1%	49.9 ± 1.6%	57.8 ± 1.3%	64.1 ± 1.0%	68.4 ± 0.8%
LambdaMART	32.6 ± 1.7%	56.2 ± 2.2%	73.7 ± 1.7%	88.3 ± 0.8%	47.5 ± 1.9%	56.3 ± 1.7%	62.5 ± 1.2%	67.1 ± 1.0%
AdaRank	29.9 ± 1.3%	53.3 ± 1.1%	71.4 ± 0.9%	85.8 ± 0.8%	44.7 ± 1.1%	53.7 ± 0.8%	59.9 ± 0.8%	65.4 ± 0.6%
AnswerBot	31.8 ± 1.8%	52.8 ± 2.0%	71.6 ± 1.9%	88.7 ± 0.9%	44.5 ± 1.7%	54.3 ± 1.6%	62.6 ± 1.2%	68.1 ± 0.9%
DeepAns-IR	42.8 ± 1.3%	62.8 ± 1.9%	78.2 ± 2.0%	90.0 ± 0.9%	55.4 ± 1.6%	63.1 ± 1.6%	68.2 ± 1.0%	72.1 ± 0.8%
DeepAns	**45.7 ± 1.6%**	**65.7 ± 1.6%**	**80.2 ± 1.9%**	**92.1 ± 1.2%**	**58.3 ± 1.4%**	**65.6 ± 1.3%**	**70.7 ± 1.1%**	**73.8 ± 0.8%**

Table 7. Automatic evaluation (SO-Java)

Model	P@1	P@2	P@3	P@4	DCG@2	DCG@3	DCG@4	DCG@5
RandomForest	32.9 ± 1.0%	56.1 ± 1.1%	74.1 ± 1.1%	89.5 ± 0.7%	47.6 ± 0.9%	56.6 ± 0.8%	63.2 ± 0.6%	67.2 ± 0.5%
XgbTree	35.9 ± 1.3%	59.0 ± 1.2%	75.7 ± 0.9%	89.1 ± 1.0%	50.5 ± 1.0%	58.8 ± 0.7%	64.6 ± 0.7%	68.8 ± 0.5%
LambdaMART	31.5 ± 1.2%	54.4 ± 1.2%	72.3 ± 1.7%	87.6 ± 1.3%	46.0 ± 0.8%	54.9 ± 1.1%	61.5 ± 0.7%	66.3 ± 0.5%
AdaRank	29.2 ± 2.1%	52.1 ± 2.2%	69.9 ± 1.8%	86.2 ± 1.5%	43.6 ± 1.8%	52.5 ± 1.7%	59.6 ± 1.5%	64.9 ± 1.1%
AnswerBot	34.7 ± 1.5%	58.0 ± 2.1%	77.8 ± 1.9%	90.2 ± 1.5%	49.4 ± 1.6%	59.3 ± 1.5%	64.7 ± 1.1%	68.4 ± 0.8%
DeepAns-IR	42.3 ± 2.9%	63.7 ± 2.3%	78.3 ± 2.1%	91.8 ± 1.6%	55.7 ± 2.4%	63.1 ± 2.2%	68.9 ± 1.8%	72.1 ± 1.4%
DeepAns	**45.5 ± 1.6%**	**65.9 ± 2.2%**	**79.9 ± 1.6%**	**92.0 ± 0.9%**	**58.4 ± 1.9%**	**65.4 ± 1.5%**	**70.6 ± 1.2%**	**73.7 ± 0.9%**

(4) Compared to our model, the learning-to-rank based approach achieved the worst performance regarding the *P@K* and *DCG@K* scores with different depths. The learning to rank approach ignores the fact that ranking is a prediction task on a list of objects. Because they require a large number of training instances with ranking labels, therefore if the ground truth ordering of input candidates is lacking, they are unable to capture the relative preference between two QA pairs. This may explain the reason why its performance is comparatively suboptimal.

(5) The DeepAns-IR approach has its advantage as compared to other baselines excluding our proposed model. This is because DeepAns-IR employs the same data labeling strategy and the model structure as ours. Moreover, it also incorporates the IR-based approach to expand the query with clarifying questions. This verifies the effectiveness of our model for question and answering tasks in technical Q&A sites. The only difference between DeepAns-IR and our model is that our model generates clarifying questions via deep sequence to sequence learning, while the DeepAns-IR retrieves the clarifying questions from the existing database according to a similarity score, which relies heavily on whether similar questions can be found and how similar the questions are. This results in our model's superior performance as compared to the DeepAns-IR approach.

(6) By comparing the evaluation results of the different technical Q&A sites, i.e., Ask Ubuntu, Super User and Stack Overflow, we can see that our proposed model is stably and substantially better than the other baselines. This suggests that our approach behaves consistently across

Table 8. Ablation Evaluation (Ask Ubuntu)

Measure	Drop CQ	Drop Labeling	DeepAns
P@1	34.2 ± 1.3%	31.3 ± 1.2%	**40.9 ± 1.5%**
P@2	58.9 ± 1.8%	50.5 ± 1.1%	**61.7 ± 1.9%**
P@3	77.3 ± 1.5%	68.9 ± 1.3%	**77.9 ± 0.9%**
P@4	91.4 ± 0.8%	86.0 ± 1.1%	**92.0 ± 0.9%**
DCG@2	49.8 ± 1.5%	43.4 ± 0.9%	**54.0 ± 1.7%**
DCG@3	59.5 ± 1.2%	51.7 ± 0.8%	**62.1 ± 1.1%**
DCG@4	65.8 ± 0.9%	59.1 ± 0.7%	**68.2 ± 1.1%**
DCG@5	68.5 ± 0.7%	64.5 ± 0.5%	**71.3 ± 0.9%**

Table 9. Ablation Evaluation (Super User)

Measure	Drop CQ	Drop Labeling	DeepAns
P@1	35.8 ± 1.4%	29.7 ± 1.4%	**40.7 ± 1.9%**
P@2	60.2 ± 1.0%	53.9 ± 1.9%	**65.8 ± 1.1%**
P@3	79.6 ± 0.9%	72.5 ± 1.5%	**82.2 ± 1.1%**
P@4	92.1 ± 0.5%	89.5 ± 0.9%	**93.9 ± 0.8%**
DCG@2	51.2 ± 1.1%	45.0 ± 1.6%	**56.5 ± 1.2%**
DCG@3	60.9 ± 0.9%	54.0 ± 1.4%	**64.7 ± 1.2%**
DCG@4	66.3 ± 0.7%	61.4 ± 1.1%	**69.8 ± 1.0%**
DCG@5	69.3 ± 0.7%	65.6 ± 0.8%	**72.1 ± 0.8%**

different technical Q&A platforms, regardless of the different topic of the specific technical forums. This supports the likely generalization and robustness of our approach. We also notice that the advantage of our proposed model is much more obvious on SO (Python) and SO (Java) as compared to Ask Ubuntu and Super User. The reason for this phenomenon is likely the large number of training samples from Stack Overflow which benefits the classification performance of our model.

In summary, our model substantially outperforms the baselines under automatic evaluation.

5.2 RQ-2: Ablation Analysis

Ablation analysis is used to verify the effectiveness of the *DeepAns* using *Question boosting* and *Label establishing* methods. More specificly, we compare our approach with several of its incomplete variants:

- **Drop CQ**: removes the clarifying question part generated by *Question boosting* model.
- **Drop Labeling**: removes the training samples generated by *Label establishing* model, to do this, we keep the best QA pairs as *positive samples*, and make other answer pairs as *negative samples*. Our model was trained as a binary classification model.

We performed the ablation analysis experiment on Ask Ubuntu and Super User respectively. The ablation analysis results are presented in the Table 8 and Table 9. We can observe the following points.

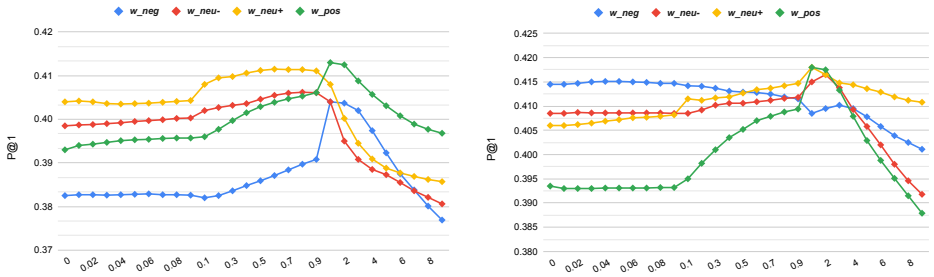

Fig. 5. Sensitivity Analysis on Ask-ubuntu (left) and Super-user (right)

(1) By comparing the results of our approach with each of the variant model, we can see that no matter which method we dropped, it does hurt the performance of our model. This verifies the importance and effectiveness of these three mechanisms.
(2) By comparing the results of **DeepAns** with **Drop CQ**, it is clear that incorporating a clarifying question improves the overall performance. When adding a clarifying question to our model, the $P@k$ score is improved by 19.5% and 13.9% on Ask Ubuntu and Super User dataset respectively. We attribute this to that the useful clarifying question can reduce the lexical gap between answer and questions, which can make the information properly fused between them.
(3) By comparing the results of **DeepAns** with **Drop Labeling**, we can measure the performance improvements achieved due to the incorporation of "Label establishment" process. After removing the training samples constructed by *Label establishment*, there is a significant drop overall in every evaluation measure. This is because by employing our *label establishing* process, the size of the training data is largely expanded, in the meanwhile, by introducing $neutral^+$ and $neutral^-$ samples, our model can learn to better distinguish best answer from similar ones.

In summary, both the *question boosting* module and *label establishing* model are effective and helpful to enhance the performance of our approach.

5.3 RQ-3: Parameters Tuning

In this section, we tune the key parameters of our model for sensitivity analysis and robustness analysis.

5.3.1 Sensitivity Analysis. We have four key parameters (i.e., $\omega_{pos}, \omega_{neu+}, \omega_{neu-}, \omega_{neg}$) in Equation 13. The optimal settings of these weights were carefully tuned on our dataset. We demonstrate the weights tuning on Ask Ubuntu and Super User respectively. In particular, the validation set was leveraged to validate our model and the grid search method was employed to select optimal parameters between 0 and 10 with small but adaptive step sizes. The step sizes were 0.01, 0.1, and 1 for the range of [0, 0.1], [0.1, 1] and [1, 10], respectively. The parameters tuning process was varying one weight while fixing the other three weights. For example, in order to tune the parameter ω_{neg}, we fix the other three parameters and change ω_{neg} from 0 to 10 with different step sizes. After that, we fix ω_{neg} to its optimal settings for tuning other parameters. Fig. 5 illustrates the performance of our model with respect to different weights on Ask Ubuntu and Super User respectively. From the figure, we have the following observations:

(1) Even though the four parameters vary in a relatively wide range, the performance of our proposed model DeepAns changes within small ranges near the optimal settings. This

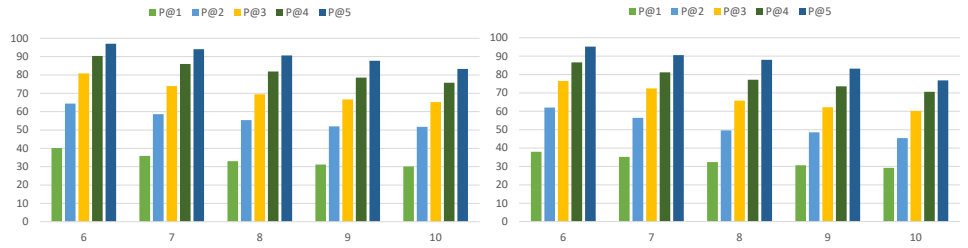

Fig. 6. Robustness Analysis on Ask-ubuntu (left) and Super-user (right)

indicates that our model is non-sensitive to the parameters around their optimal settings, which further supports the generalization ability of our approach.

(2) We notice that most parameters achieve their best performance in the range of [1, 3], we thus recommend to initialize the weights in Equation 13 to be around the above range, which is close to the optimal settings of our model.

5.3.2 Robustness Analysis. In real world Q&A sites, there is no guarantee to find the exactly matched questions from the archive, expecially when k is small. Therefore we have to enlarge k to improve the recall of the similar questions and hence the "matched answers". However, a larger k may introduce more noise into the answer candidate pool with more irrelevant answers. This can then increase the difficulty of our answer recommendation task.

To verify the robustness of our proposed approach, we set different thresholds for the number of returned questions by k-NN method. More specifically, we varied the number of returned similar questions k from 6 to 10 and measured the performance of our approach, we then reported average P@1-5 over each dataset under different parameter settings of k. The results of Ask Ubuntu and Super User are shown in Fig. 6. We can make the following observations:

(1) The trend in overall performance of our model decrease as k increases, which supports our concern that larger k settings introduce more noises and bring bigger challenges for our task. By analyzing the performance of our approach with respect to different k, we notice that our approach achieves good performance when k varies from 5 to 7, while still ensuring the "matched answer" is highly-ranked. We thus recommend setting k within the above range for real-world applications.

(2) The advantage of our proposed model is more obvious on $P@1$ compared with other metrics($P@2-5$). Even when we set k to 10, the performance of our model on $P@1$ is still on a par with the best performance of other baselines, while k is set to 5 in these baselines (See Table 4 and Table 5). This reveals that our model can perform well under a noisy context, which shows the robustness of our model.

In summary, our model is non-sensitive and robust under different parameter settings.

6 USER STUDY SETUP AND RESULTS

Since automatic evaluation results do not always agree with the actual ranking preference of real-world users, we also performed a small, qualitative user study to measure how humans actually perceive the results produced by our approach. Specifically, we mainly focus on the following research questions:

- *RQ-4:* How effective is the question boosting results of our approach under human evaluation?
- *RQ-5:* How effective is the question answering results of our approach under human evaluation?

Table 10. Human Evaluation of Question Boosting Results

Data	Model	$\text{Score}(1)_R$	$\text{Score}(2)_R$	$\text{Score}(3)_R$	Avg_R	$\text{Score}(1)_U$	$\text{Score}(2)_U$	$\text{Score}(3)_U$	Avg_U
Ask Ubuntu	IR-based	21.6%	43.2%	35.2%	2.14	28.8%	34.4%	36.8%	2.08
	Ours	18.4%	32.8%	48.8%	2.30	22.4%	35.8%	42.4%	2.20
SO (Python)	IR-based	19.2%	36.0%	44.8%	2.26	26.4%	32.0%	41.6%	2.15
	Ours	17.6%	32.0%	50.4%	2.33	23.2%	29.6%	47.2%	2.24

For human evaluation, we used the Ask Ubuntu and Stack Overflow (Python) platforms to perform our user study. We invited 5 evaluators to participate in our user study; all of these participates have more than three years of studying/working experience in software development process, have more than one year of experience using technical Q&A sites, and are familiar with the Ubuntu system and Python programming languages. We did not limit the amount of time for evaluators to complete the user study.

6.1 RQ4: Human Evaluation on Question Boosting Results

To gain a deeper understanding of how the clarifying questions impact the results in our study, we conducted human evaluation studies to measure how humans perceive the question boosting results. To do this, we consider two modalities in our user study: *Relevance* and *Usefulness*. *Relevance* measures how relevant the clarifying question is to the original question title. *Usefulness* measures how useful the clarifying question is for adding missing information for the original post. We randomly sampled 25 $\langle q, a \rangle$ pairs from Ask Ubuntu and SO (Python) respectively. For each question, we provided two clarifying questions. One was generated by our approach, the other was generated by the IR-based approach, i.e., DeepAns-IR. We also provided the accepted answer to the question as a reference. We asked the participants to manually rate the generated clarifying questions on a scale between 1 and 3 (1 = worst, 3 = best) across the above modalities. The volunteers were blinded as to which question title was generated by our approach.

Evaluation Results. We obtained 125 groups of scores from evaluators for Ask Ubuntu and SO (Python) respectively. Each group contains two pairs of scores, which were rated for clarifying questions produced by IR-based approach and ours. Each pair contains a score for the *Relevance* modality and a score for *Usefulness* modality. The score distribution and average score of *Relevance* and *Usefulness* across the two methods are presented in Table 10. From the table, we can observe the following points:

(1) Our approach performs better than the IR-based approach on both modalities. We attribute this to the following reason: the IR-based approach relies heavily on whether similar clarifying questions can be retrieved from the existing $\langle q, cq \rangle$ dataset. Considering the complexity of the questions in technical Q&A sites, there may exist only a few questions that are very similar to the given one, hence it is difficult to retrieve relevant clarifying questions from the training set.

(2) Both the IR-based approach and our approach can produce relevant and useful clarifying questions for the given question. This further verifies the clarifying question is helpful in adding missing information and reducing the gap between questions and answers. We also notice that there are still quite a few questions that received low scores for *Relevance* and *Usefulness* modalities. Even though the clarifying questions generated by our approach are still not perfect, our study is the first step on this topic and we also release our data to inspire follow-up work for utilizing the clarifying questions.

Ex1. Question: (41969 - SO Python) Standard way to open a folder window in linux?	**Ex2. Question: (2093 - Ask Ubuntu)** How to install GUI desktop on a server?	**Ex3. Question: (1263 - Ask Ubuntu)** Unable to update thunderbird
Answer: os.system('xdg-open "%s"' % foldername) xdg-open can be used for files/urls also	**Answer:** Depending on which desktop you wish to install, you install the the meta-package that installs all the necessary packages. You can use apt-get or aptitude to do this. sudo apt-get install ubuntu-desktop nstalls the Unity desktop sudo aptitude install kubuntu-desktop would install the KDE desktop Other desktop meta-packages are xubuntu-desktop, ubuntu-gnome-desktop, lubuntu-desktop, and edubuntu-desktop .	**Answer:** Nope. The only way to allow Thunderbird to update itself is to launch it as root, or to find yet another source that built Thunderbird in such a way that you can install it entirely within your own home folder. A better option, if you want to run cutting-edge apps, is to try to find someone making builds on the in a Launchpad PPA. These can update through the normal Synaptic/Upgrade system, which means you won't end up with corner cases like this. As it happens, the Mozilla team maintains a PPA of newer versions of Firefox and Thunderbird, including even nightlies. If you install Thunderbird 3.1 from there, you should be golden.
Clarifying Question (Ours): have you tried 'xdg-open folder()' ? **Relevance**: score_3(5) / score_2(0) / score_1(0) **Usefulness**: score_3(5) / score_2(0) / score_1(0)	**Clarifying Questions (Ours):** did you try 'sudo apt-get install -f' **Relevance**: score_3(5) / score_2(0) / score_1(0) **Usefulness**: score_3(4) / score_2(1) / score_1(0)	**Clarifying Question (Ours):** how did you install thunderbird ? **Relevance**: score_3(2) / score_2(1) / score_1(2) **Usefulness**: score_3(2) / score_2(2) / score_1(1)
Clarifying Question (IR-based): have you looked as 'os.walk()' ? **Relevance**: score_3(3) / score_2(2) / score_1(0) **Usefulness**: score_3(3) / score_2(1) / score_1(1)	**Clarifying Question (IR-based):** how did you try to install gnome desktop? **Relevance**: score_3(4) / score_2(1) / score_1(0) **Usefulness**: score_3(3) / score_2(1) / score_1(1)	**Clarifying Question (IR-based):** do you have any thunderbird extensions installed? **Relevance**: score_3(2) / score_2(2) / score_1(1) **Usefulness**: score_3(3) / score_2(1) / score_1(1)

Fig. 7. Evaluation Examples of Question Boosting.

Evaluation Examples. A major challenge for question answering tasks is the semantic gap between the questions and answers. This is because the questions from technical Q&A sites are, more often than not, very specific and complex, and oriented towards expert professional answers. To fill the gap between question and answers, we employ a deep encoder-decoder model to generate a clarifying question for a given post as a way of question boosting. Fig. 7 presents three examples of human evaluation on question boosting results (the words that appear in both clarifying questions and answers are highlighted). From these cases, we can see that:

(1) The clarifying questions produced by our approach as well as the IR-based approach generally perform well across both modalities. It is clear that the clarifying question can reduce the lexical gap between the answer and the questions, which can add missing information and make the information better linked between question and answers. For example, in the first and second case, our approach generates "xdg-open" and "sudo apt-get install" for the clarifying questions which also appear in the answers. Thus, the added information can eliminate and/or reduce the isolation between questions and answers. We attribute this to the advantage of our model for learning common patterns automatically from the $\langle q, cq \rangle$ pairs.

(2) Not all the clarifying questions are appreciated by the evaluators; an example is shown in the last row of Fig. 7. For such cases, even though the generated clarifying question is not optimal to the participants, our approach still precisely replicates the salient tokens, i.e., "thunderbird" from the question title, which also increases the likelihood of selecting the right answer from answer candidates.

In summary, the clarifying questions generated by our approach are effective under human evaluation results.

6.2 RQ5: Human Evaluation on Question Answering Results

Since the final goal of our study is recommending relevant answers to developers, we also performed a human evaluation to measure the effectiveness of question answering results with respect to human developers. To be more specific, we measured how developers perceive the answers produced by our approach to solved questions, unresolved questions and unanswered questions. For solved questions, we compared our approach with the ground truth; for unresolved questions, we compared

Table 11. Human Evaluation - Ask Ubuntu

Type	Approach	Score(1)	Score(2)	Score(3)	$Rank_{avg}$
Solved	Ground Truth	7.2%	18.4%	74.4%	2.67
	DeepAns	19.2%	32.8%	48.0%	2.29
Unresolved	xgbTree	15.2%	26.4%	58.4%	2.43
	AnswerBot	12.8%	28.0%	59.2%	2.46
	DeepAns	12.0%	23.2%	64.8%	2.53
Unanswered	SE Engine	51.2%	33.6%	15.2%	1.64
	Google	25.6%	32.8%	41.6%	2.16
	DeepAns	22.4%	30.4%	47.2%	2.25

Table 12. Human Evaluation - SO (Python)

Type	Approach	Score(1)	Score(2)	Score(3)	$Rank_{avg}$
Solved	Ground Truth	5.6%	14.4%	80.0%	2.74
	DeepAns	18.4%	31.2%	50.4%	2.32
Unresolved	xgbTree	12.0%	26.4%	61.6%	2.50
	AnswerBot	9.6%	32.0%	58.4%	2.49
	DeepAns	10.4%	21.6%	68.0%	2.58
Unanswered	SE Engine	54.4%	32.0%	13.6%	1.59
	Google	30.4%	31.2%	38.4%	2.08
	DeepAns	26.4%	28.0%	45.6%	2.19

our approach with xgbTree and Answerbot methods; and for unanswered questions, we compared our approach with Stack Exchange search engine and Google search engine.

6.2.1 User Study on Solved Questions. In order to investigate the agreement of the developers on solved questions, we randomly sampled 25 examples of solved questions from the testing set of Ask Ubuntu and SO (Python) respectively. For each solved question, we provided two answer candidates. One answer was the accepted answer – we refer to it as the ground truth in this study. The other answer was produced by our approach. After that, each evaluator was asked to manually rate on the two answer candidates from 1 to 3, according to the acceptance of the answer. Score 3 means that the evaluator strongly agrees with the acceptance of the answer, and score 0 means that the evaluator strongly disagrees with the acceptance of the answer. It is worth emphasizing that the answer selected by our approach may actually be the same with the ground truth answer, and the participants were blinded as to which answer is the ground truth.

Evaluation Results. We collected 125 groups of scores from participants for Ask Ubuntu and SO (Python) respectively. Each group contains two scores, which were rated for answers of the ground truth and ours. We count the proportion of different scores and calculate the average score for each method. The evaluation results for Ask Ubuntu and SO (Python) are presented in Table 11 and Table 12 respectively. From the table, we can observe the following points:

(1) The evaluators are in agreement with acceptance of the ground truth answers for most cases. For example, around 75% of the ground truth answers in Ask Ubuntu and 80% answers in SO (Python) are appreciated by the volunteers.
(2) The ground truths are better than our approach. This is reasonable because the ground truth answers are usually high-quality answers that have been accepted by the developers. Even though our approach is not as good as the ground truth at the current stage, we observe that a small number of answers produced by our approach are marked with score 1. This indicates that the answers selected by our approach are meaningful and acceptable for the majority of questions.

Ex1. Solved Question (493367 - SO Python): **Python: For each list element apply a function across the list ?**	**Ex2. Solved Question (742371 - SO Python)** **Why does Python skip elements when I modify a list while iterating over it?**	**Ex3. Solved Question (230712 - Ask Ubuntu):** **Can I download Ubuntu 12.04 on a notebook/laptop?**

Answer(Ground Truth) :
score_3(3) / score_2(2) / score_1(0)
You can do this using list comprehensions and min() (Python 3.0 code):
>>> nums = [1,2,3,4,5]
>>> [(x,y) for x in nums for y in nums]
>>> min(_, key=lambda pair: pair[0]/pair[1])
 	
Note that to run this on Python 2.5 you'll need to either make one of the arguments a float, or do from __future__ import division so that 1/5 correctly equals 0.2 instead of 0.
	Answer(Ground Truth) :
	score_3(4) / score_2(1) / score_1(0)
	
	This is a well-documented behaviour in Python, that you aren't supposed to modify the list being iterated through.
	Try this instead:
	for i in x[:]:
	x.remove(i)
	The [:] returns a "slice" of x, which happens to contain all its elements, and is thus effectively a copy of x.
		Answer(Ground Truth) :
		score_3(5) / score_2(0) / score_1(0)
		
		Ubuntu-desktop runs on netbooks and laptops.
		If you are concerned about the performance of your laptop running the desktop version of Ubuntu you should try Lubuntu which is a lightweight distro based on Ubuntu but using the LXDE desktop environment.
		Lubuntu 12.10 is the latest version.

Answer(Ours) :
score_3(4) / score_2(1) / score_1(0)
Using list comprehension:
>>> oldList = [1, 2, 3, 4]
>>> newList = [x for x in oldList for _ in range(2)]
>>> newList
Above list comprehension is similar to following nested for loop.
newList = []
for x in oldList:
for _ in range(2):
newList.append(x)
	Answer(Ours) :
	score_3(3) / score_2(2) / score_1(0)
	
	You edit the list while loop over it. Since you only keep pointers for looping you remeber positions, but they change since elements vanish. Try remebering the elements you want to delete by something like:
	…
	Using List Comprehensions from the start leads to:
	dt_list = [3600, 2700, 1800, 900]
	dt_list = [x for x in dt_list if (3600/x).is_integer()]
		Answer(Ours) :
		score_3(2) / score_2(2) / score_1(1)
		
		Yes, you can download the .deb file from packages.ubuntu.com and then open it like an archive (using file-roller). There are two more archives inside, you'll need to open "data.tar.gz".

Fig. 8. Examples of Solved Questions

Evaluation Examples. Fig. 8 shows three examples of the user study on solved questions. It can be seen that:

(1) In general, our approach can produce acceptable answers. Sometimes, the answers chosen by our approach are actually more accepted by the volunteers than the ground truth answers. For example, in the first sample, three evaluators gave a score of 3 to the ground truth answer, while four evaluators gave a score of 3 to ours. However, our answer does not belong to the current question thread and is selected from answer candidates of other questions (e.g., *Python: duplicating each element in a list*). This further justifies the feasibility of addressing *answer hungry* problem by selecting answers from the historical QA dataset.
(2) Outputs from our model are not always "correct". For example, in the last sample, the information seeker asks a question of "*Can I download Ubuntu 12.04 on a notebook/laptop?*", while the answer provided by our approach is about how to download a file from the packages. This example reveals that considering the complexity of the questions in technical Q&A sites, the gap between the ground truth answers and ours is still large, and hence there is still a large room for our question answering system to be further improved.

6.2.2 User Study on Unresolved Questions. To investigate how developers perceive our approach to solve the unresolved questions, we sampled 25 unresolved questions for Ask Ubuntu and SO (Python) respectively. Each question has multiple answer candidates that have not been selected as *Accept*. By computing the matching score between question and each answer candidate, we can identify a best answer via our approach, xgbTree and Answerbot respectively (note that different approaches may choose the same answer as the best answer). Following that, we ask each evaluator to rank three answer candidates produced by our approach, xgbTree and Answerbot from 1 to 3 (3 is the best) according to the acceptance of the answer. It is worth emphasizing that the answers identified by our approach and others could be the same, and the order of the answers is randomly decided.

Evaluation Results. The human evaluation results of unresolved questions for Ask Ubuntu and SO (Python) are presented in Table 11 and Table 12 respectively. From the table, we can see that:

(1) Our model performs better than xgbTree and Answerbot baselines. This further indicates that the answers selected by our approach are more appreciated by evaluators. The results of human evaluation on unresolved questions are consistent with large-scale automatic

Ex1. Unresolved Question (12109648 - SO Python): **Python: how to adjust x axis in matplotlib ?**	**Ex2. Unresolved Question (12035394 - SO Python)** **Python - replace random items in column**	**Ex3. Solved Question (1169424 - Ask Ubuntu):** **How do I put two Ubuntu OSes on the same hard drive??**
Answer(xgbTree) : **score_3(3) / score_2(2) / score_1(0)** Look at the docs : xlocs, xlabs = plt.xticks() put in xlocs your range, and in xlabs what you want to display. then: plt.xticks(xlocs, xlabs)	**Answer(xgbTree & Ours) :** **score_3(4) / score_2(1) / score_1(0)** I would strongly suggest reading a python primer, the way your problem solved can be done in two steps 1. read items from file - reference 2. use math.random() to change random string- reference by know how to do these points you can easily achieve what you intend to do.	**Answer(xgbTree) :** score_3(3) / score_2(2) / score_1(0) You don't put two OS in the same partition. Each instance of Ubuntu or an Ubuntu family OS requires its own partition. You can run the installer for the second version you wish to install, and choose 'Something Else' when you get to Installation Type, and let it share the same drive; just not the same partition. See this step-by-step workflow illustration for Ubuntu 18.04 LTS.
Answer(Answerbot) : **score_3(3) / score_2(1) / score_1(1)** The size of the plot can be changed by setting the dynamic rc settings of Matplotlib. These are stored in a dictionary named rcParams. The size of the plot figure is stored with the key figure.figsize.	**Answer(Anwerbot) :** **score_3(3) / score_2(1) / score_1(1)** Use this to generate a random string import os random_string = os.urandom(string_length) To loop over a file line by line, do with open('file') as fd: for line in fd: # do stuff No need to close the file handle. use split to well, split on whitespace and place the result in an array (indexing starts at 0) Read more at docs.python.org. Please update your question with some code when you have gotten that far... Good luck	**Answer(Ours & Answerbot) :** **score_3(3) / score_2(2) / score_1(0)** Assuming there is nothing you want to preserve on your hard drive, you can do the following: 1. When you install the first instance of Ubuntu, at some point during installation there will be a window called "Installation type". Choose "Something else". 2. On the next window you can erase and create partitions. Make sure you create at least two partitions large enough for Ubuntu installations. 3. Select the partition on which you want to install the first instance of Ubuntu and press Change. Select Ext4 under "Use as" and "/" under "Mount point". 4. Go ahead, finish the installation. 5. Boot from the installation USB again and repeat step 1. 6. No need to repartition, as that was done in step 2. already. 7. Select the second partition and repeat Step 3. for it. 8. Go ahead, and finish the installation of the second instance. The next time you boot the computer Grub will allow you to boot to whichever instance you want.
Answer(Ours) : **score_3(4) / score_2(1) / score_1(0)** It sounds like you want to changes the limits of the plotting display - for that use xlim (and ylim for the other axis). To change the xticks themselves is provided in the answer by @fp. Show below is an example using without/with xlim: import pylab as plt plt.subplot(2,1,1) plt.hist(X,bins=300) plt.subplot(2,1,2) plt.hist(X,bins=300) plt.xlim(0,100) plt.show()		

Fig. 9. Examples of Unresolved Questions

evaluation results, which reconfirms the effectiveness of our approach for identifying the best answer in unresolved questions.

(2) Compared with the evaluation results of ground truth, the average scores between the answers of unresolved questions and solved questions are close, which supports our previous assumption that users forget to mark the accepted answer is not uncommon in technical Q&A sites.

Evaluation Examples. Fig. 9 shows the examples of the user study on unresolved questions. It can be seen that:

(1) The overall answer quality for the unresolved questions is good. This is because these answers are directly related to the specific problems of the questions, which are more suitable to the needs of information seekers.

(2) Even all the answer candidates of an unresolved question aim at solving the same problem. As can be seen, some answers identified by our approach stand out from the rest and are more appreciated by evaluators, such as samples 1-2. This further verifies the ability of our approach to select the most relevant answer from a set of answer candidates.

6.2.3 User Study on Unanswered Questions. Similar to unresolved questions, We also randomly sampled 25 examples of unanswered questions for Ask Ubuntu and SO(Python) respectively. For each unanswered question, considering that developers usually search for technical help using Google search engine and/or the Q&A site search engine itself, we compare our approach against two baselines built based on the above search engines respectively. We used the question title of the post as the search query. For Google search engine, we add "site:stackoverflow.com" and "site:askubuntu.com" to the end of the search query so that it searches only posts on Stack Overflow and Ask Ubuntu respectively. We use the first ranked question returned by Google search engine as the most relevant question, we extracted the accept answer or the answer with the highest vote if there is no accepted answer of the relevant question. For technical Q&A site search engine, we refer to the first ranked related question recommended by the technical Q&A site search engine as the most relevant question, and extracted the associated accepted answer or the highest-vote answer. After constructing the evaluation set for unanswered questions, for each unanswered question, we asked the evaluators to rank on the 3 answer candidates from 1 to 3 (3 for the best answer), The

Ex1. Unanswered Question (57147927 - SO Python): **Python: How to import package correctly in python?**	**Ex2. Unresolved Question (57152181- SO Python)** **How to read a bit field integer in python?**	**Ex3. Solved Question (1179515 - Ask Ubuntu):** **Ubuntu 18.04 reboots randomly ?**
Answer(Stack Overflow) : **score_3(0) / score_2(2) / score_1(3)** For dictionaries x and y, z becomes a shallowly merged dictionary with values from y replacing those from x. In Python 3.5 or greater: z = {**x, **y} … Resources on Dictionaries.	**Answer(Stack Overflow) :** **score_3(0) / score_2(1) / score_1(4)** For dictionaries x and y, z becomes a shallowly merged dictionary with values from y replacing those from x. In Python 3.5 or greater: z = {**x, **y} … Resources on Dictionaries.	**Answer(Stack Overflow) :** **score_3(2) / score_2(2) / score_1(1)** Your OEM system vendor Asus probably placed the 24GB SSD drive for caching purposes, e.g. Intel Smart Response. This could explain why you can't set it as a boot drive. I suffer from the same issue in my HP laptop with an mSATA SSD slot and HP's response to this is "this is intentional". …… One downside of this approach is that your system will fail to boot even if just one of the drives fails.

Answer(Google) :
score_3(2) / score_2(2) / score_1(1)
 	
"I have a medium size Python application with modules files in various subdirectories." Good. Make absolutely sure that each directory include a __init__.py file, so that it's a package.
"I have created modules that append these subdirectories to sys.path" Bad. Use PYTHONPATH or install the whole structure Lib/site-packages. Don't update sys.path dynamically. It's a bad thing. Hard to manage and maintain.
Bad. Use PYTHONPATH or install the whole structure Lib/site-packages. Don't update sys.path dynamically. It's a bad thing. Hard to manage and maintain. ……
My current project has 100's of modules, a dozen or so packages. Each module imports just what it needs. No magic.
	Answer(Google) :
	score_3(1) / score_2(3) / score_1(1)
	
	The following code will load the requested portions of the binary number into the fields:
	class Register(object):
	  def __init__(self,x):
	  self.fieldwidths = [6,12,6,4,12,8,16]
	  ……
	Which will result in:
	BitField1 = 0b111011
	BitField2 = 0b1110111101110
	……
	The results may not be what you wanted because of the fact that the data you provided is not 64 bits but rather 128 bits, which would mean that the 64 most significant bits of the input data will be ignored by the program.
		Answer(Google) :
		score_3(3) / score_2(2) / score_1(0)
		
		This sounds like a combination of issues.
		
		In the case of an individual system rebooting randomly I would want to replace the power supply in the chassis with one that provided more than adequate amperage for the connected components (as you want it to keep running during periods of peak power draw).
		
		In the case where the entire rack reboot simultaneously I would look at an inadequate UPS as the root cause or possibly an overheat condition due to AC failure in the server location.
		
		An intermittent short in the feed cord to the multi-tap could also result in the multiple reboot result that you describe.

Answer(Ours) :
score_3(3) / score_2(2) / score_1(0)
 	
Since there are already many answers on SO for this*, I will focus on question (2). About what is a better code organization:
…
The (relative) import is done as follows, from inside module_2:
  from ..pkg1 import module1 as m1
Alternatively, you can use absolute imports, which refer to the top package name:
  from top_pkg_name.pkg1 import module1 as m1
In such an organization, when you want to run any module as a script, you have to use the -m flag:
  python -m top_pkg_name.pkg1.module1
	Answer(Ours) :
	score_3(4) / score_2(1) / score_1(0)
	
	In Python 3 you can use something like this:
	  int.from_bytes(byte_string, byteorder='little')
		Answer(Ours) :
		score_3(3) / score_2(2) / score_1(0)
		
		I'm guessing hardware problems.
		
		Your Lubuntu problem "Press S to skip mounting or Press M for Manual recovery." could have been from corrupted filesystems. And random reboots could be an indication of bad RAM too.
		
		I had a laptop with bad RAM that would reboot just like that, after 20-40 minutes, the RAM was physically dirty & I think even had corroded contacts, after a while it wouldn't work at all with that RAM

Fig. 10. Examples of Unanswered Questions

higher grade indicates that the answer is more suitable to the given question. Please note that the participants do not know which answer is generated by which approach.

Evaluation Results. The expert evaluation results of unanswered questions for Ask Ubuntu and SO (Python) are presented in Table 11 and Table 12. We can observe the following points:

(1) Compared with baselines, our model outperforms SE (Stack Exchange search engine) and Google (Google search engine). This suggests that the answers produced by our approach are considered to be more suitable to the given question by the evaluators. We attribute this to the reason that Google search engine identifies the answer via searching from similar questions, thus it is unable to judge the matching degree between the questions and answers. In contrast, our approach estimates the matching score using the context information of the qa pair, which fills the gap between questions and answers. The superior performance of our approach in terms of average score further supports the effectiveness of our approach in identifying the best answer.
(2) For the unanswered questions, a gap for the answer quality between unanswered questions and solved/unresolved questions still exists. We also notice that our approach received more low scores ($score = 1$) with unanswered questions as compared to solved/unresolved questions. This is because in technical Q&A sites, some questions are rather complicated and sophisticated and it is hard to find suitable question-specific answers for these questions.

Evaluation Examples. Fig. 10 shows three examples of the user study on unanswered questions. we can observe the following points:

(1) The search engine of the technical Q&A site achieves worst performance. For example, in sample 1 and sample 2, the SE search engine recommends the same answer to two different questions. This is why the evaluators give comparatively low scores to the answers identified by SE search engine.

(2) Our approach has its advantage as compared to the Google search engine (e.g., sample 1-2). This is because the Google search engine does not consider the contextual information between the questions and answers, but instead only identifies the answers based solely by searching for similar questions. By contrast, our approach takes the question as well as the candidate answers and calculates the matching score between the question and the answers, which results in its superior performance compared to the other baselines.
(3) In technical Q&A sites, some question titles are relatively abstract and uninformative. For example, in sample 3, even the answer selected by our approach is relevant and meaningful, we can not make sure if the answer solves the actual problem or not. For such cases, more detailed information, such as the description in the question body, could be considered when searching for appropriate answers.

In summary, our model is comparatively effective under human evaluation for question answering tasks in technical Q&A sites.

7 DISCUSSION

In this section, we first discuss the strength of our approach as well as the threats to validity of our work, after that we analyze the outlier cases involving in our data creation process.

7.1 Strength of Our Approach

To address the answer hungry problem in technical Q&A sites, we propose a deep learning based approach DeepAns to search relevant answers from historical QA pairs. We summarized the strength of our approach as follows:

7.1.1 Neural Language Model for Question Boosting. One advantage of our approach is training an attentional sequence-to-sequence model for generating clarifying questions as a way of question boosting. Instead of searching similar clarifying questions, our approach builds a neural language model for linking semantics of question and clarifying questions. The neural language model is able to handle the uncertainty in the correspondence between the questions and clarifying questions. Our approach automatically learns common patterns automatically from the $\langle q, cq \rangle$ pairs. The encoder itself is a neural language model which is able to remember the likelihood of different kinds of questions. Following that, the decoder learns the context of the questions fills the gap between the questions and clarifying questions.

7.1.2 Label Establishment for Data Augmentation. Due to the reason of the professional questions in technical Q&A sites, it is thus very hard, if not possible, to find experts and annotators for manual labeling the QA pairs. In this paper, we present a novel labeling scheme to automatically construct *positive*, *neutral*$^+$, *neutral*$^-$, and *negative* training samples. Guided by our four heuristic rules, this label establishment process can collect large amounts of labeled QA pairs, which greatly saves the time-consuming and labor-intensive labeling process.

7.1.3 Deep Neural Network for Answer Recommendation. We present a weakly supervised neural network for the answer recommendation task in technical Q&A sites. Our model architecture is able to incorporate the aforementioned four types of training samples for ranking QA pairs. Our work first uses the deep neural network to solve the problem of best answer selection in technical Q&A sites, which is able to alleviate the answer hungry phenomenon that widely exists in technical Q&A forums.

7.2 Threats to Validity

We have identified the following threats to validity among our study:

Internal Validity Threats to internal validity are concerned with potential errors in our code implementation and study settings. For the automatic evaluation, in order to reduce errors, we have double-checked and fully tested our source code. We have carefully tuned the parameters of the baseline approaches and used them in their highest performing settings for comparison, but there may still exist errors that we did not note. Considering such cases, we have published our source code and dataset to facilitate other researchers to replicate and extend our work.

External Validity The external validity relates to the quality and generalizability of our dataset. Our dataset is constructed from the official StackExchange data dump. We focus on three technical Q&A sites, i.e., Ask Ubuntu, Super User and Stack Overflow for our experiment. These three technical Q&A sites are commonly used by software developers and each one focuses on a specific area. However, there are still many other technical Q&A sites in StackExchange which are not considered in our study (e.g., Server Fault). We believe that our results will generalize to other technical Q&A sites as well, due to the ability of our approach to identify the best answer from a set of answer candidates. We will try to extend our approach to other technical Q&A sites to benefit more users in future studies.

Construct Validity The construct validity concerns the relation between theory and observation. In this study, such threats are mainly due to the suitability of our evaluation measures. For human evaluation, the subjectiveness of the evaluators, the evaluators' degree of carefulness, and the human errors may affect the validity of judgements. We minimized such threats by choosing experienced participants who have at least three years of studying/working experience in the software development process, and are familiar with Ubuntu system and Python programming languages. We also gave the participants enough time to complete the evaluation tasks.

Model Validity The model validity relates to model structure that could affect the learning performance of our approach. In this study, for the answer recommendation task, we choose a CNN-based model due to the optimum results achieved by [32]. Recent studies [33, 54] have shown that the RNN-based model can also achieve promising performance on the text classification task, which is similar to ours. For the question boosting task, we use the vanilla sequence-to-sequence model. Recent research has proposed new models, such as the pointer-generator [43], transformer [48] and bert [13]. However, our results do not shed light on the effectiveness of employing other deep learning models with respect to different structures and new advanced features. We will try to use other deep learning models for our tasks in future work and compare them to those we report in this paper.

7.3 Outlier Cases Study

As detailed in Section 3.2, we build our training samples via four heuristic rules, we thus can not ensure that there are no outlier cases distant from our heuristic rules. The outlier cases will produce a series of wrong preference pairs and hinder the learning performance of our model. Fig. 11 shows three outlier examples for label establishment. From the figure we can see that:

(1) From the first example, it can be seen that, the quality of its non-accept answer in terms of informativeness and relevance are better than the accepted ones, not to mention that the link provided within the *Positive* sample has been not available. This shows the outlier case that the non-accept answers may be better than the accepted answers.
(2) From the second example, it can be seen that, for a given question, the answers from its similar questions are more descriptive than its own. This shows the outlier cases that the answers of other questions may be better than its own.
(3) From the last example, it can be seen that, the answers from its similar questions may provide more information cues than its non-accept answers.

Outlier Examples	**Q (Ask Ubuntu)**: How to install SiS 671/771 Video Drivers in ubuntu?	**A**(*Positive*): This is an easy how-to: http://sites.google.com/site/easylinuxtipsproject/sis
		A(*Neutral+*): The display driver for sis would be already installed in Ubuntu. xserver-xorg-video-sis is the display driver for all Sis and XGI video driver. …
	Q (SO Python): How can I create a regular expression in Python?	**A**(*Positive*): Try something like this: r'[a-zA-Z0-9]+_[^_]+_[a-zA-Z0-9]+\.[a-zA-Z0-9]+'.
		A(*Neutral-*): https://www.debuggex.com is also pretty good. It's an online Python (and a couple more languages) debugger, which has a pretty neat visualization of what does and what doesn't match. A pretty good resource if you need to draft a regexp quickly.
	Q (Ask Ubuntu): How to share hotspot through SSH tunnel?	**A**(*Neutral+*): Oh, it works just as that. I didn't notice I have to check the rule specifying the table explicit: …
		A(*Neutral-*): … Here's two options that do work, though: 1. use sshuttle … 2. set up OpenVPN on the remote system and your local system

Fig. 11. Outlier Examples in Label Establishment

Detecting and removing these outlier cases before building the training samples will benefit the learning performance of our proposed DeepAns model, we will focus on this research direction in the future.

8 RELATED WORK

In this section, we describe the related studies on best answer retrieval, query expansion in software engineering, and deep learning in software engineering.

8.1 Best Answer Retrieval

Great effort has been dedicated to addressing the question answering tasks on Q&A sites [1, 8, 9, 28, 39, 41, 47, 53]. Conventional techniques for retrieving answers primarily focus on complementary features of the Q&A sites. For example, Adamic et al. [1] reported the first study on best answer prediction in Yahoo! Answers using user-related features. Following Adamic et al.'s study, Tian et al. [47] trained a classifier on a dataset from Stack Overflow without relying on user-related features. Recently, Calefato et al. [9] modelled the answer prediction task as a binary classification problem, they assessed 26 best answer prediction model in Stack Overflow. Different from these works, we present a novel weakly supervised neural network architecture for ranking answers for a given question. To the best of our knowledge, our work is the first to apply deep neural network to the specific problem of best answer selection in Q&A sites. Our approach can not only identify best answers from a list of candidate answers, but also recommend the most relevant answers for these unanswered posts. Besides, we also compare with Calefato et al.'s [9] approach, and the experimental results have shown that the improvement is substantial.

8.2 Query Expansion in SE

Query expansion has long been investigated as a way to improve the results returned by a search engine [4, 22, 23, 26, 35, 37, 38, 40, 50]. Some software engineering researchers have employed query expansion to improve the performance of tasks such as code search, answer summary, and similar question recommendation. For example, Haiduc et al. [22] proposed an approach that can recommend a good query reformulation strategy by performing machine learning on a set of

historical queries and relevant results. Hill et al. [23] proposed a query expansion tool named Conquer, which introduces a novel natural language based approach to organize and present search results and suggest alternative query words. More recently, Lu et at. [37] presented an approach to expand the original query with synonyms from WordNet, which can help developers to quickly reformulate a better query. Xu et al. [50] proposed a novel framework to reformulate the answer in Stack Overflow to reduce the lexical gap between question and answer sentences. Inspired by these studies we also leverage the idea of query expansion to recommend the relevant answers. Our *DeepAns* tool generates useful clarifying questions as a way of query boosting, which can substantially reduce the lexical gap between the question and answer sentences. In contrast, all of the aforementioned studies ignore the interactions between the asker and the potential helper.

8.3 Deep Learning in SE

Recently, an interesting direction in software engineering is to use deep learning to solve many diverse software engineering tasks [3, 10, 14–16, 18, 20, 21, 24, 25, 30, 31, 34, 36, 45, 49, 52]. For example, White et al. [49] leverage an deep learning approach, DeepRepair, for automatic program repairing. Gu et al. [20] propose a novel deep neural network named DeepCS for code search tasks, where code snippets semantically related to a query can be effectively retrieved. Hu et al. [24] a develop new sequence-to-sequence model named DeepCom to automatically generate code comments for Java methods. Li et al. [34] present CClearner which is a deep-learning based approach for clone detection.

Although the aforementioned studies have utilized deep learning techniques for different kinds of software engineering tasks, to our best knowledge, no one has yet considered the relevant answer recommendation task in technical Q&A sites. We proposed in this paper a novel neural network architecture to address the answer hungry problems in technical Q&A forums.

9 SUMMARY

To alleviate the *answer hungry* problem in technical Q&A sites, we have presented a novel neural network-based tool, DeepAns, to identify the most relevant answer among a set of answer candidates. Our model follows a three-stage process:*question boosting*, *label establishing* and *answer recommendation*. Given a post, we first generate a clarifying question as a way of question boosting, we then automatically generate *positive*, *neutral*$^+$, *neutral*$^-$ and *negative* training samples via label establishing. Finally based on the four kinds of training samples we generated, we trained a weakly-supervised neural network to compute the matching score between the question and candidate answers. Extensive experiments on the real-world technical Q&A sites have comparatively demonstrated the promising performance and the robustness of our approach in solving unanswered/unresolved questions..

10 ACKNOWLEDGEMENTS

This research was partially supported by the Australian Research Council's Discovery Early Career Researcher Award (DECRA) funding scheme (DE200100021), ARC Laureate Fellowship funding scheme (FL190100035), ARC Discovery grant (DP170101932 and DP200100020), and Singapore's Ministry of Education (MOE2019-T2-1-193).

REFERENCES

[1] Lada A Adamic, Jun Zhang, Eytan Bakshy, and Mark S Ackerman. 2008. Knowledge sharing and yahoo answers: everyone knows something. In *Proceedings of the 17th international conference on World Wide Web*. ACM, 665–674.

[2] Arvind Agarwal, Hema Raghavan, Karthik Subbian, Prem Melville, Richard D Lawrence, David C Gondek, and James Fan. 2012. Learning to rank for robust question answering. In *Proceedings of the 21st ACM international conference on*

Information and knowledge management. ACM, 833–842.
[3] Mohammad Alahmadi, Jonathan Hassel, Biswas Parajuli, Sonia Haiduc, and Piyush Kumar. 2018. Accurately predicting the location of code fragments in programming video tutorials using deep learning. In *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, 2–11.
[4] Azilawati Azizan and Zainab Abu Bakar. 2015. Query Reformulation Using Crop Characteristic in Specific Domain Search. In *2015 IEEE European Modelling Symposium (EMS)*. IEEE, 374–379.
[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
[6] Steven Bird and Edward Loper. 2004. NLTK: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*. Association for Computational Linguistics, 31.
[7] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.
[8] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2016. Moving to stack overflow: Best-answer prediction in legacy developer forums. In *Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement*. ACM, 13.
[9] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2019. An empirical assessment of best-answer prediction models in technical Q&A sites. *Empirical Software Engineering* 24, 2 (2019), 854–901.
[10] Qingying Chen and Minghui Zhou. 2018. A neural framework for retrieval and summarization of source code. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 826–831.
[11] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of machine learning research* 12, Aug (2011), 2493–2537.
[12] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *International conference on machine learning*. 933–941.
[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[14] Haoyu Dong, Shijie Liu, Shi Han, Zhouyu Fu, and Dongmei Zhang. 2019. TableSense: Spreadsheet Table Detection with Convolutional Neural Networks. (2019).
[15] Zhipeng Gao, Vinoj Jayasundara, Lingxiao Jiang, Xin Xia, David Lo, and John Grundy. 2019. SmartEmbed: A Tool for Clone and Bug Detection in Smart Contracts through Structural Code Embedding. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 394–397.
[16] Z. Gao, L. Jiang, X. Xia, D. Lo, and J. Grundy. 2020. Checking Smart Contracts with Structural Code Embedding. *IEEE Transactions on Software Engineering* (2020), 1–1. https://doi.org/10.1109/TSE.2020.2971482
[17] Zhipeng Gao, Xin Xia, John Grundy, David Lo, and Yuan-Fang Li. 2020. Generating Question Titles for Stack Overflow from Mined Code Snippets. *arXiv preprint arXiv:2005.10157* (2020).
[18] Zhipeng Gao, Xin Xia, David Lo, John Grundy, and Yuan-Fang Li. 2020. Code2Que: A Tool for Improving Question Titles from Mined Code Snippets in Stack Overflow. *arXiv preprint arXiv:2007.10851* (2020).
[19] Alex Graves. 2012. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711* (2012).
[20] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 933–944.
[21] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 631–642.
[22] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. 2013. Automatic query reformulations for text retrieval in software engineering. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 842–851.
[23] Emily Hill, Manuel Roldan-Vega, Jerry Alan Fails, and Greg Mallet. 2014. NL-based query refinement and contextualized code search results: A user study. In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, 34–43.
[24] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension*. ACM, 200–210.
[25] Qiao Huang, Xin Xia, David Lo, and Gail C Murphy. 2018. Automating intention mining. *IEEE Transactions on Software Engineering* (2018).
[26] Qing Huang, Yangrui Yang, Xue Zhan, Hongyan Wan, and Guoqing Wu. 2018. Query expansion based on statistical learning from code changes. *Software: Practice and Experience* 48, 7 (2018), 1333–1351.
[27] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 2073–2083.

[28] Maximilian Jenders, Ralf Krestel, and Felix Naumann. 2016. Which answer is best?: Predicting accepted answers in mooc forums. In *Proceedings of the 25th International Conference Companion on World Wide Web*. International World Wide Web Conferences Steering Committee, 679–684.

[29] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188* (2014).

[30] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 1039–1049.

[31] Junhwi Kim, Minhyuk Kwon, and Shin Yoo. 2018. Generating test input with deep reinforcement learning. In *2018 IEEE/ACM 11th International Workshop on Search-Based Software Testing (SBST)*. IEEE, 51–58.

[32] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

[33] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*.

[34] Liuqing Li, He Feng, Wenjie Zhuang, Na Meng, and Barbara Ryder. 2017. Cclearner: A deep learning-based clone detection approach. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 249–260.

[35] Zhixing Li, Tao Wang, Yang Zhang, Yun Zhan, and Gang Yin. 2016. Query reformulation by leveraging crowd wisdom for scenario-based software search. In *Proceedings of the 8th Asia-Pacific Symposium on Internetware*. ACM, 36–44.

[36] Peng Liu, Xiangyu Zhang, Marco Pistoia, Yunhui Zheng, Manoel Marques, and Lingfei Zeng. 2017. Automatic text input generation for mobile testing. In *Proceedings of the 39th International Conference on Software Engineering*. IEEE Press, 643–653.

[37] Meili Lu, Xiaobing Sun, Shaowei Wang, David Lo, and Yucong Duan. 2015. Query expansion via wordnet for effective code search. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 545–549.

[38] Liming Nie, He Jiang, Zhilei Ren, Zeyi Sun, and Xiaochen Li. 2016. Query expansion based on crowd knowledge for code search. *IEEE Transactions on Services Computing* 9, 5 (2016), 771–783.

[39] Liqiang Nie, Xiaochi Wei, Dongxiang Zhang, Xiang Wang, Zhipeng Gao, and Yi Yang. 2017. Data-driven answer selection in community QA systems. *IEEE transactions on knowledge and data engineering* 29, 6 (2017), 1186–1198.

[40] Sudha Rao and Hal Daumé III. 2018. Learning to Ask Good Questions: Ranking Clarification Questions using Neural Expected Value of Perfect Information. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, 2737–2746. https://doi.org/10.18653/v1/P18-1255

[41] Tirath Prasad Sahu, Naresh Kumar Nagwani, and Shrish Verma. 2016. Selecting best answer: An empirical analysis on community question answering sites. *IEEE Access* 4 (2016), 4797–4808.

[42] Denis Savenkov. 2015. Ranking Answers and Web Passages for Non-factoid Question Answering: Emory University at TREC LiveQA.. In *TREC*.

[43] Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368* (2017).

[44] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Edinburgh neural machine translation systems for wmt 16. *arXiv preprint arXiv:1606.02891* (2016).

[45] Zeyu Sun, Qihao Zhu, Lili Mou, Yingfei Xiong, Ge Li, and Lu Zhang. 2019. A grammar-based structural cnn decoder for code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7055–7062.

[46] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.

[47] Qiongjie Tian, Peng Zhang, and Baoxin Li. 2013. Towards predicting the best answers in community-based question-answering services. In *Seventh International AAAI Conference on Weblogs and Social Media*.

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[49] Martin White, Michele Tufano, Matias Martinez, Martin Monperrus, and Denys Poshyvanyk. 2019. Sorting and transforming program repair ingredients via deep learning code similarities. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 479–490.

[50] Bowen Xu, Zhenchang Xing, Xin Xia, and David Lo. 2017. AnswerBot: automated generation of answer summary to developersź technical questions. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 706–716.

[51] Jun Xu and Hang Li. 2007. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 391–398.

[52] Xinli Yang, David Lo, Xin Xia, Yun Zhang, and Jianling Sun. 2015. Deep learning for just-in-time defect prediction. In *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 17–26.

[53] Jun Zhang, Mark S Ackerman, and Lada Adamic. 2007. Expertise networks in online communities: structure and algorithms. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 221–230.
[54] Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. 2016. Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. *arXiv preprint arXiv:1611.06639* (2016).

Received Mar 2019; revised ?? 2019; accepted ?? 2019