Automatic Dialogue Summary Generation for Customer Service

Chunyi Liu, Peng Wang, Jiang Xu, Zang Li, Jieping Ye
AI Labs, Didi Chuxing*
{liuchunyi,wangpengeric,xujiang,lizang,yejieping}@didiglobal.com

ABSTRACT

Dialogue summarization extracts useful information from a dialogue. It helps people quickly capture the highlights of a dialogue without going through long and sometimes twisted utterances. For customer service, it saves human resources currently required to write dialogue summaries. A main challenge of dialogue summarization is to design a mechanism to ensure the logic, integrity, and correctness of the summaries. In this paper, we introduce auxiliary *key point sequences* to solve this problem. A key point sequence describes the logic of the summary. In our training procedure, a key point sequence acts as an auxiliary label. It helps the model learn the logic of the summary. In the prediction procedure, our model predicts the key point sequence first and then uses it to guide the prediction of the summary. Along with the auxiliary key point sequence, we propose a novel *Leader-Writer* network. The Leader net predicts the key point sequence, and the Writer net predicts the summary based on the decoded key point sequence. The Leader net ensures the summary is logical and integral. The Writer net focuses on generating fluent sentences. We test our model on customer service scenarios. The results show that our model outperforms other models not only on BLEU and ROUGE-L score but also on logic and integrity.

KEYWORDS

Dialogue summarization, Hierarchical encoder-decoder, Customer service, Leader-Writer net

ACM Reference Format:
Chunyi Liu, Peng Wang, Jiang Xu, Zang Li, Jieping Ye. 2019. Automatic Dialogue Summary Generation for Customer Service. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3292500.3330683

1 INTRODUCTION

For customer service, a dialogue between a user and an agent contains important information, such as the user's question and the agent's solution. Directly understanding a dialogue by going through long and sometimes twisted utterances is time-consuming. Dialogue summarization is the task of distilling the highlights of a dialogue and rewriting them into a concise version. Summarization is especially useful for the customer service in DiDi.[1] DiDi requires its customer service agents to write summaries about dialogues with users. Table 1 gives an example of the <dialogue, summary> pair (it is translated from Chinese to English manually). The summary helps other agents and relevant systems quickly capture the user's question and boost the efficiency of subsequent steps. On average, writing the summaries takes about 25% of agents' time. DiDi has thousands of agents. Thus, automatic summary generation can save vast human resources. What's more, poorly written summaries lead other agents to make the wrong decisions on users' questions. Automatic summary generation can help guarantee that all the summaries have the same standard.

Table 1: A dialogue-summary pair.

Dialogue
AGENT: Hello, what can I do for you?
USER: What's the standard of electric vehicles for the Express.
AGENT: Do you have a car?
AGENT: Or are you going to buy a car?
USER: I am hesitating which car to buy. One is Jianghuai EV Seven, the other is BYD YUAN.
AGENT: OK, you can fulfill the table in this link (link info) with the type of vehicle you wish to check. We will give you feedback in seven days.
USER: I have not bought yet.
USER: Can you check it now?
AGENT: I am quite sorry for that. A specialist on this issue will check it and call you back.
AGENT: They will give a precise answer for your question.
USER: OK.
AGENT: Thanks for your understanding. What else can I do for you?
USER: Nothing, thanks. Bye.
AGENT: Thank you. Have a nice day.
Summary
The user's question was about the standard of EV car for the Express. He asked the standard to decide which car to buy. I told the user to fill in the type of the cars in our system and we would give feedback in seven days. The user approved the result. The user hung up.
Key point sequence
Question description ⟶ Solution ⟶ User approval ⟶ End

Dialogue summarization is closely related to text summarization. The task of text summarization has two main paradigms: extractive and abstractive. The former method directly chooses and outputs the sentences (or phrases) in the original document [6, 11, 18, 20]. In our setting, input dialogues consist of utterances (direct quotes of people), but our output summaries should not contain such direct quotes. The abstractive approach gives summaries of the dialogues

* Peng Wang is the corresponding author.

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330683

[1] DiDi is the world's leading mobile transportation platform.

Table 2: A partial list of our 51 key points.

Key point	Corresponding sample summary
Question description	The user's question is about the standard of EV car for the Express drivers.
Solution	I told the user to fulfill the type of the cars in our system and we would give feedback in seven days.
End	User hung up.
User approval	The user approved the result.
User disapproval	The user disapproved the result.
Suggestion	I suggested the user go to the downtown for orders.
Need call back	We should call her back in 30 minutes.
Acknowledgement	The user acknowledged the results.
Escalation	Need senior agent's further investigation.
Judgment from systems	The system showed that it is the driver's duty for the detour.
Close the order	I closed the order.
User's appeal	The passenger requests that the driver should be punished.
Keywords	Keywords: detour.
Related order	Related order id: xxxxxxx
Question type	Question type: fee issue.
Time of event	Time of event: 2019-01-01 00:00:00
Type of vehicle	BMW
The driver's status	The driver has passed the checking.

by distilling the information and rewriting it in a concise manner [5, 9, 22, 23, 28, 30]. This paradigm achieves great success due to neural sequence-to-sequence models.

Even with this progress, dialogue summarization still meets great challenges. First, summarization should be logical, i.e., the summaries from the dialogue must be organized in a readable order. For example, for a dialogue in customer service, a summary usually describes the user's question first, then the solution, and then the user's feedback. A dialogue can be logically chaotic, but the human agent writes the summary in a logical and understandable manner. The model should generate the summary with similar logic as the human agents. Second, summarization should be integral, i.e., the summaries cover all the important facts in the dialogue. Missing facts will lead to misjudgment or mistreatment of the user's question. Third, the summary should ensure its correctness for key facts. For example, the sentences *User approved our solution* and *User disapproved our solution* are semantically similar, but have opposite facts. Lastly, for complicated dialogues, the summaries can be long. 34.2% of summaries are longer than 100 Chinese characters. Learning how to generate the long summaries is also a challenge.

To this end, we introduce auxiliary *key point sequences* to help dialogue summarization.

- **What is a key point sequence?** A key point is the theme of a contiguous set of one or more summary sentences. Take the summary in Table 1 as an example, the first two sentences are about the key point *Question description*. A key point sequence describes the logic and key facts of the summary. In our setting, there are 51 key points. Table 2 gives a partial list of key points. Note that facts with contrary meaning are treated as different key points, e.g. *User approval* and *User disapproval*, instead of one key point, e.g., *User feedback*. In this way, we expect the model can ensure the correctness of the key facts in the summary.
- **How to generate a key point sequence?** In the preprocessing procedure, we extract a key point sequence from a summary by rules. The rules of labeling the key point sequences are given by the domain experts. For example, if a sentence in a summary starts with *I suggested* or *I recommended*, the sentence will be labeled with the *Suggestion* key point. If a sentence does not match any rules, it will be labeled with the key point of its preceding sentence. If the first sentence does not match any rules, it will be labeled with the *Question description* key point. Eventually, all the sentences of a summary are assigned with corresponding key points. In the training procedure, key point sequences are auxiliary labels. Our model learns how to generate the key point sequences from dialogues. In the prediction procedure, the key point sequences are auxiliary outputs. Our model predicts a key point sequence first, then uses it to guide the prediction of the summary.

With the help of auxiliary key point sequences, we present a novel *Leader-Writer net* for dialogue summarization. The Leader-Writer net mainly contains a hierarchical decoder structure. The decoder has the *Leader net* which decodes the key point sequence to ensure the logic and integrity of a summary and the *Writer net* which decodes the sub-summaries under key points to ensure the grammar and semantics of a summary. The final summary is built by concatenating all the sub-summaries. Moreover, based on the Transformer [33] architecture, we propose a hierarchical Transformer to encode the dialogue. A token-level Transformer uses the token embeddings to generate the context-free representations of utterances; while an utterance-level Transformer uses the outputs of the lower encoder to generate the contextual representations of utterances which acts as memory for the decoder. In the loss layer, we introduce the reinforcement loss by self-critic [27] to directly learn from the ROUGE-L [21] rewards. The contributions of this paper are as follows.

- We propose to use auxiliary *key point sequences* to ensure the logic and integrity of dialogue summaries.
- We propose a novel hierarchical decoder architecture, the *Leader-Writer net*, to generate both key point sequences and the summaries.
- Our model achieves 55.9 BLEU score and 68.9 ROUGE-L score on the DiDi customer service dialogue datasets, which outperforms the state-of-the-art methods.

2 RELATED WORKS

2.1 General Text Summarization

Text summarization has two main paradigms: extractive and abstractive. Extractive methods choose sentences (or phrases) from the original documents [6, 11, 18, 20]. As we have mentioned in Section 1, extractive methods are improper for dialogue summarization in customer service. Abstractive methods [5, 9, 22, 23, 28, 30] distill the highlights of dialogues and rewrite them in a concise manner. We investigate the techniques of the abstractive methods.

The **encoder-decoder framework** and **attention mechanism** are widely used in summarization models [9, 10, 22, 28, 30]. These models use an encoder to get the contextual representations of tokens and a decoder to generate a summary. The attention mechanism [3] is adopted to calculate the weighted average of contextual representations for better decoding.

The **pointing mechanism** [13, 14, 22, 23, 34, 38] allows models to directly copy tokens from the inputs, which can help handle the out of vocabulary (OOV for short) tokens. The Pointer-Generator network [31] uses the pointing mechanism for summarization. The model can switch between pointing (i.e., copying a token) and generation with a gate mechanism. The pointing is based on the attention distribution over the input tokens.

The **reinforcement learning** helps summarization models directly learn from the non-differentiable metrics, e.g. ROUGE and BLEU. It helps the model generate more readable summaries [7, 17, 27, 28]. The traditional teacher forcing training procedure based on cross-entropy loss suffers from the exposure bias problem [2, 28]. Reinforcement learning also helps alleviate the problem by directly optimizing the ROUGE and BLEU score.

Our model is based on the encoder-decoder framework. It introduces the pointing mechanism in the Writer net, and adopts the self-critic method [29] in the reinforcement learning. To our best knowledge, most previous models are only supervised by the summaries. They can learn the semantics and grammar of the summaries, but cannot guarantee the logic and integrity. Different with them, we introduce the key point sequence as the auxiliary label. The Leader net learns the logic and integrity of the summaries and the Writer net learns the semantics and grammar.

2.2 Dialogue Summarization

For dialogue summarization, each utterance is associated with a speaker. Before neural networks were introduced to summarization, some researchers adopted feature engineering [37] and template selection [24] to summarize the dialogues. But these models require people to design features or templates manually. The limited number of features and templates hinder their extensibility. Instead, our model infers the key point sequence from the dialogue, so the potential number of sequences is infinite. Other work introduced the unsupervised learning for summarization [4]. In the scenarios which lack labeled data, unsupervised methods are preferred. In DiDi, we have a huge number of manually written summaries. With this supervision, the generated summaries are closer to the ones written by human agents.

Recently, abstractive dialogue summarization attracts increasing research interests due to the success of the sequence-to-sequence neural networks. A simple way to summarize the dialogue is to treat utterances as sentences and apply the aforementioned neural based methods. However, different from documents, the dialogue summarization should consider the speaker of the utterances. Previous work [12, 25] uses a hierarchical encoder to catch the interactions. Auxiliary information is also important in the dialogue summarization. Chih-Wen and Yun-Nung [12] use auxiliary dialogue act to assist the summarization, where each utterance is assigned a dialogue act to label its effect on the interactions. Our model also has a hierarchical encoder to model dialogues. And we propose to use key point sequences as auxiliary labels to ensure the logic and integrity of summaries.

3 MODEL ARCHITECTURE

3.1 Problem Settings

Given an input dialogue between a customer service agent and a user, the goal is to produce a multi-sentence summary which captures the highlights of the dialogue.

Formally, the original record contains a dialogue G and a summary S. The dialogue is a list of utterances, i.e., $G = \{g_1, g_2, \cdots, g_M\}$, where g_i is the i-th utterance in the dialogue and M is the number of utterances. Each utterance contains a list of word tokens, i.e., $g_i = \{x_{i,role}, x_{i,1}, x_{i,2}, \cdots, x_{i,m_i}\}$, where $x_{i,j}$ is the j-th token in g_i and m_i is the token number. At the start of the sequence, we add a special token $x_{i,role}$ which represents the role of speaker, i.e., $x_{i,role} \in \{user, agent\}$. The summary is also a list of word tokens $S = \{s_1, s_2, \cdots, s_N\}$, where N is the number of tokens.

In the preprocessing procedure, we extract an auxiliary key point sequence P from S with rules as described in Section 1. The rule-based analyzer takes the original summary $S = \{s_1, \cdots, s_N\}$ and outputs a key point sequence $P = \{p_1, \cdots, p_n\}$ (where n is the key points number and $n \leq N$) and a list of sub-summaries $\{S_i | S_i = \{s_{i,1}, \cdots, s_{i,n_i}\}\}$, where $i = 1, 2, \cdots, n$. S_i corresponds to a key point p_i and $s_{i,j}$ is the j-th token in i-th sub-summary. We have $S = concat(S_1, \cdots, S_n)$.

Our model $h(\cdot)$ uses the dialogue G to predict the auxiliary key point sequence P as well as the the summary S, i.e., $h(G) \rightarrow (P, S)$. Our neural model contains an embedding layer, a hierarchical encoder, a *Leader-Writer* decoder, and a loss layer. The Transformer [33] is the building block of our model. Figure 1 gives the overview of our model.

3.2 Embedding Layer

The embedding layer maps each word token and key point token into a d_e-dimension vector. Our model learns the embedding vectors from scratch. Dialogues and summaries contain tokens in vocabulary $\mathcal{V}$. They share the same token embeddings. $\mathcal{P}$ indicates the set of key points. We add a top dot to the notations defined in Section 3.1 as their embeddings. For examples, $\dot{x}_{i,j}$ is the token embedding of j-th token of the i-th utterance in the dialogue, and $\dot{s}_{i,j}$ is the j-th token in the i-th sub-summary.

3.3 Hierarchical Encoder

28% of the dialogues have more than 20 utterances. So the token representations as the working memory is too big for the decoder. In our scenarios, both agents and users usually express one thing in an utterance. We propose a hierarchical Transformer encoder for the utterance-level encoding, whose outputs act as the working memory for the decoder. The token-level Transformer takes the token embeddings and outputs the context-free utterance-level representations. Then, the utterance-level Transformer takes previous outputs and generates contextual utterance-level representations. In this section, we first introduce the general Transformer encoder, then describe the hierarchical encoder.

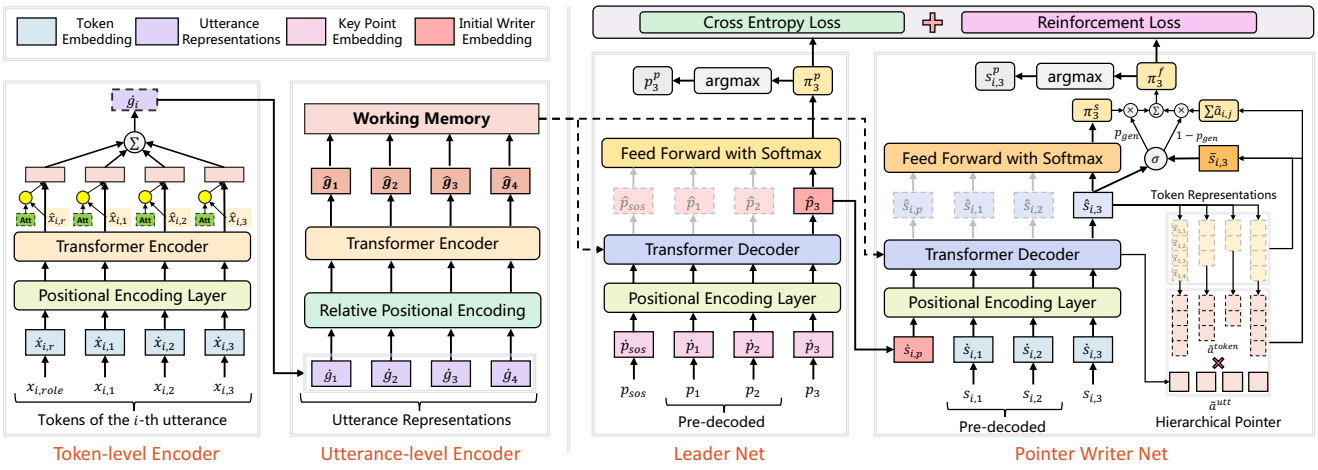


Figure 1: Model architecture. The left part is the hierarchical encoder which contains a token-level encoder and an utterance-level encoder. The encoder generates the working memory for decoding. The right part is the hierarchical decoder which contains the Leader net that generates the key point sequence and the Writer net that generates the sub-summary for a key point. The top part is the loss layer which contains cross entropy loss and reinforcement loss.

3.3.1 General Transformer encoder. The encoder stacks multiple Transformer encoding layers to encode the inputs, formalized as:

$$TransEnc\left(\dot{Z}\right) \longrightarrow \hat{Z} \tag{1}$$

where $\dot{Z} = \{\dot{z}_1, \cdots, \dot{z}_n\}$ is the embedding sequence and $\hat{Z} = \{\hat{z}_1, \cdots, \hat{z}_n\}$ are the contextual representations of $\dot{Z}$.

The Transformer encoding layer consists of a multi-head self-attention sub-layer and a feed forward sub-layer. The general form of attention is:

$$Att(Q, K, V) = softmax(score(Q, K))V \tag{2}$$

In self-attention, the one head attention is

$$\begin{aligned} Self\_att\left(\dot{Z};\Theta\right) &= Att\left(W^Q\dot{Z}, W^K\dot{Z}, W^V\dot{Z}\right) \\ score(Q,K) &= \left(W^Q\dot{Z}\right)\left(W^K\dot{Z}\right)^T/\sqrt{d} \end{aligned} \tag{3}$$

where $\Theta = \left\{W^Q \in \mathbb{R}^{d\times l}, W^K \in \mathbb{R}^{d\times l}, W^V \in \mathbb{R}^{d\times l}\right\}$ is a set of trainable parameters, and l is the feature dimension and d is the hidden size.

The multi-head attention (MHA) sub-layer uses independent parameters for each head, concatenates the attention results, and compresses the results with a linear transformation.

$$\begin{aligned} \dot{Z}^{self} &= MHA^{self}\left(\dot{Z};\{\Theta_1,\cdots,\Theta_H\}\right) \\ &= W^O concat\left(head_1,\cdots,head_H\right) \\ head_i &= Self\_att\left(\dot{Z};\Theta_i\right) \end{aligned} \tag{4}$$

$W^O \in \mathbb{R}^{l\times l}$ is a trainable parameter, H is the number of heads. Note that we have $l = Hd$.

Then, we apply a feed forward sub-layer to the $\dot{Z}^{self}$ as follows:

$$\hat{Z} = W_2 \text{Gelu}\left(W_1\dot{Z}^{self} + b_1\right) + b_2 \tag{5}$$

where $W_1 \in \mathbb{R}^{d'\times l}, b_1 \in \mathbb{R}^{d'}, W_2 \in \mathbb{R}^{l\times d'}$ and $b_2 \in \mathbb{R}^l$ are trainable parameters. Gelu$(\cdot)$ [16] is the active function. With the feed-forward sub-layer, we keep the output dimension the same as the input dimension.

Moreover, we adopt the residual connection [15] and layer norm [1] after each sub-layer, i.e., $x \leftarrow Layer\_norm\left(x + \text{sublayer}(x)\right)$, where sublayer$(\cdot)$ is the MHA or feed forward layer described before.

Finally, multiple Transformer encoding layers stack together and form the Transformer encoder $TransEnc(\cdot)$.

3.3.2 Hierarchical Transformer encoder. The hierarchical Transformer encoder adopts $TransEnc(\cdot)$ as the building block. We first map the embedding from dimension d_e into l dimension with a linear transformation if $d_e \neq l$. Then, we encode each utterance by a token-level Transformer encoder. For the token embedding in i-th utterance $\dot{X}_i = \{\dot{x}_{i,role}, \dot{x}_{i,1}, \cdots, \dot{x}_{i,n_i}\}$, we obtain the encoded token representations $\hat{X}_i = \left\{\hat{x}_{i,role}, \hat{x}_{i,1}, \cdots, \hat{x}_{i,n_i}\right\}$ as

$$\hat{X}_i \longleftarrow TransEnc^{token}\left(\dot{X}_i\right) \tag{6}$$

We apply attentive summarization [35] over $\hat{X}_i$ to get a single vector $\dot{g}_i$ as the utterance-level representation:

$$\dot{g}_i = softmax\left(score\left(\hat{X}_i\right)\right)\hat{X}_i \tag{7}$$

where for each element $\hat{x}_{i,j}$ in $\hat{X}_i$, the score is

$$score\left(\hat{x}_{i,j}\right) = v^T \tanh\left(W^S\hat{x}_{i,j} + b^S\right) \tag{8}$$

where $W^S \in \mathbb{R}^{l\times 1}$ and $b^S \in \mathbb{R}$ are trainable variables. Here $\dot{G} = \{\dot{g}_1, \cdots, \dot{g}_M\}$ are context-free representations of the utterances, as each $\dot{g}_i$ lacks the information from other utterances.

Then, we use another Transformer encoder to get the contextual utterance representations $\hat{G} = \{\hat{g}_1, \cdots, \hat{g}_M\}$.

$$\hat{G} \longleftarrow TransEnc^{utt}\left(\dot{G}\right) \tag{9}$$

where $\hat{G}$ acts as the working memory for the decoder.

3.4 Hierarchical Decoder

We propose a novel *Leader-Writer net* for decoding summaries. The Leader net decodes the key point sequence with the utterance representations $\hat{G}$, while the Writer net decodes the summary based on both the decoded key point sequence and the utterance representations. Both nets are Transformer decoders, as shown in the right part of Figure 1. In this section, we first introduce the general Transformer decoder, then describe the the Leader net and Writer net.

3.4.1 General Transformer decoder. For one decoding step t, the decoder works as follows:

$$Decoder\left(\dot{Z}_t, M\right) \longrightarrow \pi_t, w_t, \hat{z}_t \tag{10}$$

where $\dot{Z}_t = \{\dot{z}_1, \cdots, \dot{z}_t\}$ is the embeddings of the previous decoded sequence, $M = \{m_1, \cdots, m_r\}$ is the memory set, π_t is the distribution of the decoded symbols, w_t is the decoded symbol, and $\hat{z}_t$ is the decoded representation. The decoder stacks multiple Transformer decoding layers and a final output layer. One Transformer decoding layer works as:

$$TransDec\left(\dot{Z}_t, M\right) \longrightarrow \hat{z}_t \tag{11}$$

There are three sub-layers: a multi-head self-attention sub-layer, a multi-head cross-attention sub-layer and a feed forward sub-layer. The multi-head self-attention sub-layer in the $TransDec(\cdot)$ is similar to Eq. (4), but it has a mask to ensure that the prediction for step t depends only on the known outputs at steps less than t. We denote the output as $\dot{Z}_t^{self}$. The feed forward sub-layer is exactly the same as Eq. (5).

The main difference is the multi-head cross-attention layer, where the attention is between $\dot{Z}_t^{self}$ and memory M. For each head, based on Eq. (2), the cross-attention $Cross\_att(\cdot)$ is:

$$\begin{aligned} Cross\_att\left(\dot{Z}_t^{self}, M; \Theta\right) &= Att\left(W^Q \dot{Z}_t^{self}, W^K M, W^V M\right) \\ &= softmax\left(\left(W^Q Z_t^{self}\right)\left(W^K M\right)^T / \sqrt{d}\right) W^V M \end{aligned} \tag{12}$$

where the $\Theta = \left\{W^Q \in \mathbb{R}^{d\times l}, W^K \in \mathbb{R}^{d\times l}, W^V \in \mathbb{R}^{d\times l}\right\}$ is a set of trainable parameters. Note that the softmax function generates the attention distribution $\tilde{a}_h$ of head h to record the weight distribution of working memory. There is also a residual connection [15] and a layer norm [1] after each sub-layer.

$\hat{z}_t$ is the outputs of the last layer $TransDec(\cdot)$. The decoder uses $\hat{z}_t$ to produce the output distribution π_t and symbol w_t with a feed forward layer and softmax function,

$$\begin{aligned} \pi_t &= softmax\left(W^W \hat{z}_t + b^W\right) \\ w_t &= \arg\max \pi_t \end{aligned} \tag{13}$$

where $W^W \in \mathbb{R}^{v\times l}$ and $b^W \in \mathbb{R}^v$ are trainable parameters and v is the vocabulary size.

3.4.2 Leader net. In our decoding procedure, we use the Leader net to decode key point sequence first. For one decoding step, the leader net works as:

$$p_t^p, \pi_t^p, \hat{p}_t \longleftarrow Decoder^{KP}\left(\{\dot{p}_{sos}, \dot{p}_1, \cdots, \dot{p}_{t-1}\}; \hat{G}\right) \tag{14}$$

where $\hat{G}$ are the utterance contextual representations given by the encoder in Eq. (9), $\dot{p}_{sos}$ is the embedding of the start symbol *SOS*. There is a linear layer to map the key points embedding into l dimension if $l \neq d_e$. In the training procedure, $\dot{p}_i$ is the embedding of the reference key point; while in the prediction procedure, $\dot{p}_i$ is the embedding of the predicted key point. We stop the decoding procedure until decoding the *EOS* symbol or reaching the maximum number of decoding steps. The decoded representation $\hat{p}_t$ of the Leader net will be fed into the Writer net for decoding the corresponding sub-summary.

3.4.3 Writer net. The Writer net generates a set of sub-summaries based on the key point sequence. All the sub-summaries are concatenated as the final summary.

To be clear, we omit the indexes for the sub-summaries. For one decoding step, the Writer net works as:

$$s_t^p, \pi_t^s, \hat{s}_t \longleftarrow Decoder^{SS}\left(\{\dot{s}_p, \dot{s}_1, \cdots, \dot{s}_{t-1}\}; \hat{G}\right) \tag{15}$$

where $\dot{s}_p$ is initial embedding of the writer, which is critical for generating the key point related summary. A natural idea is to use the embedding $\dot{p}$ of the key point as $\dot{s}_p$, but this method fails when the key point sequence contains duplicate key points. For example, the key point sequence [$\cdots$, *Solution*, *User disapproval*, *Solution*, *User approval*, $\cdots$] has two *Solution* key points. Only with the symbol *solutions*, the Writer net confuses about which solution in the dialogue to summarize, and generates the same sub-summaries for the two key points. Thus, we adopt the decoded representation $\hat{p}$ as $\dot{s}_p$. $\hat{p}$ encodes the information of both utterances and previous key points, so the same key points in different positions have different representations, leading to different summaries.

3.4.4 Pointer Writer net. The pointer mechanism [31, 34] achieves great success in summarization tasks. In the decoding procedure, it chooses to directly copy a token from inputs or generate a token from a vocabulary, where the copying of the token is decided by the attention distribution over the inputs. As the encoder is hierarchical, the vanilla pointer mechanism cannot fit into the Writer net directly. We propose the hierarchical pointer mechanism. The principle is that the final attention probability is the product of the utterance-level and the token-level attention probabilities. To simplify, we omit the subscript t in this sub-section.

We denote the softmax attention distribution in Eq. (12) as $\tilde{a}_h$. We use the average of $\tilde{a}_h$ among heads from the multi-head cross-attention in the last layer of $Decoder^{SS}$ as the utterance-level attention distribution, i.e., $\tilde{a}^{utt} = \frac{1}{H}\sum_{h=1}^{H} \tilde{a}_h$.

With the $\hat{s}$ from $Decoder^{SS}$ and the token representations $\hat{X}_i$ in Eq. (6), we obtain the token-level attention distributions of the i-th utterance a_i^{token} as follows:

$$\begin{aligned} a_i^{token} &= softmax\left(score\left(\hat{s}, \hat{X}_i\right)\right) \\ score(\hat{s}, \hat{X}_i) &= \left(W_{token}^Q \hat{s}\right)\left(W_{token}^K \hat{X}_i\right)^T / \sqrt{l} \end{aligned} \tag{16}$$

where $W_{token}^Q \in \mathbb{R}^{l\times l}$ and $W_{token}^K \in \mathbb{R}^{l\times l}$ are trainable parameters. Finally, we obtain the attended probability on the dialogue token from the hierarchical structure as $\tilde{a}_{i,j} = \tilde{a}_i^{utt} \tilde{a}_{i,j}^{token}$.

Following the vanilla Pointer-Generator net [31], we use a gate to switch between pointing a token or generating one. The generation probability is

$$p_{gen} = \sigma\left(W_{gen} concat\left(\hat{s}, \bar{s}\right) + b_{gen}\right) \tag{17}$$

where $W_{gen} \in \mathbb{R}^{1\times 2l}$ and $b_{gen} \in \mathbb{R}$ are trainable parameters, and $\bar{s} = \sum_{i,j} \tilde{a}_{i,j}\hat{x}_{i,j}$ is the token-attended representation of $\hat{s}$.

Thus, the probability $\pi^f(w)$ of a specific token w is the weighed average of the generation and pointing probabilities as follows:

$$\pi^f(w) = p_{gen}\pi^p(w) + \left(1 - p_{gen}\right) \sum_{i,j:\, x_{i,j}=w} \tilde{a}_{i,j} \tag{18}$$

With the pointer mechanism, the Writer net copies tokens from the input dialogue to generate more accurate and readable sub-summaries.

For all the Transformers in the encoder and decoder, the hidden size is 256, the number of heads is 4, and the number of stacked layers is 3.

3.5 Learning Objective

In the loss layer, we use both the maximum-likelihood cross-entropy loss and the reinforcement loss [27]. Previous work mainly uses the cross-entropy loss. As we adopt ROUGE-L as the metric for summarization, there is a gap between the cross-entropy loss and the metric. So also we consider the ROUGE-L as the reinforcement reward and directly optimize it.

3.5.1 Cross entropy loss. The most widely-used method to train a decoder for sequence generation is minimizing a cross-entropy loss at each decoding step. This method is called the teacher forcing algorithm [36]. For the key point sequence, the reference is $\{p_1, p_2, \cdots, p_n\}$ for the given dialogue. Its cross entropy loss is:

$$L^p_{ce} = -\frac{1}{n}\sum_{t=1}^{n} \log \pi^p_t\left(p_t\right) \tag{19}$$

where $\pi^p_t\left(p_t\right)$ is the predicted probability of key point p_t. Similarly, for the sub-summary decoding, the reference sequence for each sub-summary for key point p_i is $\{s_{i,1}, \cdots, s_{i,m_i}\}$ and the cross entropy loss is:

$$L^s_{ce} = -\frac{1}{\sum_{i=1}^{n} n_i}\sum_{i=1}^{n}\sum_{j=1}^{n_i} \log \pi^f_{i,j}\left(s_{i,j}\right) \tag{20}$$

3.5.2 Reinforcement loss. Minimizing the cross entropy does not guarantee an increase on a discrete evaluation metric, e.g. ROUGE-L, because the model has the ground truth sequence for predicting the next symbol during training, but lack them during predicting. So the errors accumulate in prediction. This phenomenon is called exposure bias [28]. ROUGE-L captures the overlap between our generated summary and a reference sequences. However, we cannot optimize ROUGE-L with gradient descent because it is non-differentiable.

From the point view of reinforcement learning, we treat ROUGE-L as the reward and generated summary as the action sequence. And we adopt the self-critical policy gradient [27, 29] to directly optimize ROUGE-L. For the key point sequence decoding, we generate the sequences $P^r = \left\{p^r_1, \cdots, p^r_n\right\}$ by sampling from π^p_t at each decoding time step, and the sequences $P^p = \left\{p^p_1, \cdots, p^p_n\right\}$ from Eq. (14).

In our setting, we use ROUGE-L as the reward function $R(\cdot)$. The reinforcement loss for a key point sequence is:

$$L^p_{rl} = \frac{1}{n}\left(R\left(P^p\right) - R\left(P^r\right)\right)\sum_{t=1}^{n} \log \pi^p_t\left(p^r_t\right) \tag{21}$$

where $\left(R\left(P^p\right) - R\left(P^r\right)\right)$ is the rewards gap between the actions from greedy policy and actions sampled from policy function. If $R\left(P^p\right) > R\left(P^r\right)$, the reinforcement loss guides the model to lower the probability of $\pi^p_t\left(p^r_t\right)$ to speed up convergence. If $R\left(P^p\right) < R\left(P^r\right)$, the reinforcement loss guides the model to raise the probability $\pi^p_t(p^r_t)$ to encourage exploration.

Similarly, we define the reinforcement loss for a sub-summary as:

$$L^s_{rl} = \frac{1}{\sum_{i=1}^{n} n_i}\sum_{i=1}^{n}\left(R\left(S^p_i\right) - R\left(S^r_i\right)\right)\sum_{j=1}^{n_i} \log \pi^f_{i,j}\left(s^r_{i,j}\right) \tag{22}$$

where S^p_i is the summary generated by argmax, S^r_i is the summary sequence generated by sampling, and $s^r_{i,t}$ is the sampled token in the decoding step t in the i-th sub-summary.

Finally, we combine Eqs. (19)~(22) to obtain the final loss:

$$L = \alpha_1 L^p_{ce} + \alpha_2 L^s_{ce} + \alpha_3 L^p_{rl} + (1 - \alpha_1 - \alpha_2 - \alpha_3) L^s_{rl} \tag{23}$$

where α_1, α_2 and α_3 are hyper-parameters to control the loss contribution. In our experiment, we set $\{\alpha_1 = 0.25, \alpha_2 = 0.25, \alpha_3 = 0.25\}$.

3.6 Implementation Details.

For the Transformer, we use positional encoding to the token-level encoder to inject position information of the tokens. For the utterance-level encoder, as the lengths of dialogues vary a lot, the relative position rather than the absolute position is a more reasonable index for positional encoding. For example, the first quarter of a dialogue is usually on the user's problem, and the last quarter is usually on the user's feedback.

To this end, we propose a trainable relative positional encoding for the utterance. Assuming a dialogue has M utterances, the relative position of i-th utterance is $\left\lfloor \frac{iK}{M} \right\rfloor$, where K is maximum relative position number. We set $K = 30$ in our experiments.

4 EXPERIMENTS

4.1 Dataset

We collect our dialogue-summary dataset from the logs in the DiDi customer service center. The dialogues are between users and customer service agents, and the summaries are written by agents. The statistics of the data are given in Table 3.

The length distributions of the dialogues, summaries, and the key point sequences are shown in Figure 2. It shows that 28% of the dialogues have more than 20 utterances, 34% of the summaries have more than 100 tokens[2], 20% of the summaries have more than 5 key points. So dialogue summarization for our dataset is difficult.

We preprocess the data by the following steps. 1) We normalize the sample by replacing the phone number, plate number, trip information, and time with special symbols such as PHONE, PLATE_NUM,

[2]To eliminate the influence of Chinese word segmentation, the Chinese part is counted in characters and the English part is counted in words.

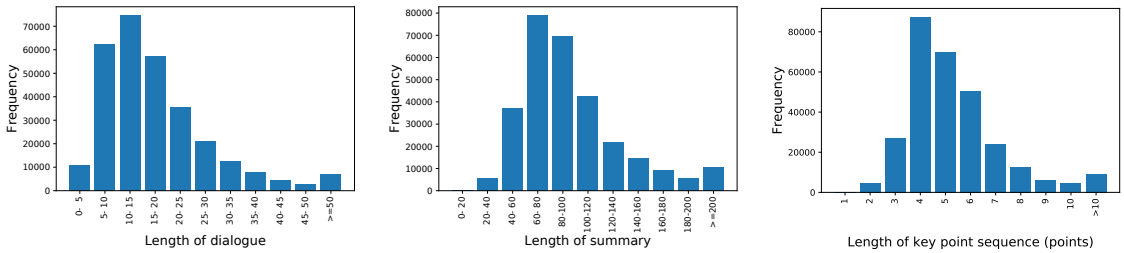


Figure 2: The left figure demonstrates the dialogue length distribution, where x-axis is the dialogue length with respect to the number of utterances; the middle figure demonstrates the summary length distribution, where x-axis is the summary length with respect to the number of Chinese characters; the right figure demonstrates the key point sequence length distribution.

Table 3: Statistics of the datasets

# of <dialogue, summary> pairs in training set	296,263
# of <dialogue, summary> pairs in developing set	2,963
# of <dialogue, summary> pairs in testing set	29,654
# of word vocabulary for dialogues and summaries	23,950
# of key point set	51

TRIP, TIME, etc. 2) For adjacent utterances, if their speakers are the same and their total token length is short (below 15), we concatenate them into one utterance. 3) We analyze the dialogues with rules to get the key point sequences and the corresponding sub-summaries as described in Section 1. 4) For the training data, to avoid long dialogues and utterances, we truncate utterances into no more than 75 tokens[3] and dialogues into no more than 40 utterances.[4] We truncate the sub-summary under each key point into no more than 50 tokens.[5]

4.2 Evaluation Metrics

We evaluate our models with the ROUGE [21] and BLEU [26]. ROUGE-1, ROUGE-2, and ROUGE-L respectively measure the unigram recall, bigram recall, and longest common sequence overlap between the references and prediction summarizes, while BLEU measures the n-gram precision. We consider 4-grams for BLEU at most. To eliminate the influence of Chinese tokenization, we compute these metrics based on the Chinese characters.

To measure the logic of the generated summaries, we also evaluate the ROUGE-1, ROUGE-2, ROUGE-L and BLEU on the key point sequences. For other models which do not generate the key point sequences, we parse predicted summaries with the same rules as described in Section 1 to obtain the key point sequences.

4.3 Training Details

We adopt Adam [19] optimizer with settings $\{\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, lr = 10^{-4}\}$. We add L_2 normalization with $weight = 7 \times 10^{-7}$ and dropout [32] with keeping rate 0.9 to avoid over-fitting. We set the batch size to 16 and use the ROUGE-L on the validation set for early stopping. The max decode lengths of the key point sequence and sub-summary are set to 15 and 50 respectively.

[3] About 5.0% of all the utterances will be truncated
[4] About 4.5% of all the dialogues will be truncated.
[5] About 2.3% of all the sub-summaries will be truncated.

4.4 Comparison Methods

We implement several abstractive summarization models.

- **Seq2seq** [8] is the basic encoder-decoder model. The utterances in a dialogue are concatenated for the token-level encoding, and the last hidden state of encoder is used for decoding the summary.
- **Seq2seq + Att** [23] adds the attention mechanism to the basic Seq2seq model. The attention mechanism collects information from all the encoder states to solve the long-term dependency problem in the RNN network.
- **Transformer** [33] is an attention-based encoder-decoder framework whose encoder and decoder have been introduced in Sections 3.3 and 3.4.
- **Pointer-Generator** [31] has the pointer mechanism. Its decoder decides whether to generate a token from the vocabulary or copy a token from the input dialogue.
- **Hierarchical Transformer** is a variant of our model. It uses hierarchical Transformer encoder in Section 3.3 to encode the dialogue and a single Transformer decoder to generate the summaries.
- **Leader+Writer** is the base version of our Leader-Writer net. It uses the Writer net in Section 3.4.3 to generate sub-summaries and it only considers the cross entropy loss in the loss layer.
- **Leader+Pointer Writer** is a variant of our model. It uses the Pointer-Writer net in Section 3.4.4 to generate sub-summaries and only considers the cross entropy loss.
- **Leader+Pointer Writer+RL** is the proposed model. It uses the Pointer-Writer net as the decoder and considers both the cross entropy loss and the reinforcement loss.

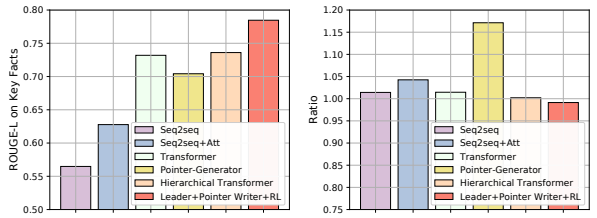
4.5 Results

Table 4 shows the ROUGE and BLEU metrics on the summaries and key point sequences. It shows that our model performs best in both summary generation and key point generation on BLEU and ROUGE-L. Our model is only behind the Pointer-Generator on ROUGE-1 and ROUGE-2. We will analyze the results in details in the following section.

4.5.1 The effects of auxiliary key point sequence. We demonstrate the effects of auxiliary key point sequence by comparing the performances of our Leader-Writer net based models with the others.

Table 4: Model comparison

Models	Parameter size	Summary				Key point sequence			
		BLEU	ROUGE-1	ROUGE-2	ROUGE-L	BLEU	ROUGE-1	ROUGE-2	ROUGE-L
Seq2seq	16M	37.7	49.4	17.2	54.7	25.7	86.7	61.4	66.7
Seq2seq+Att	17M	44.2	58.1	32.0	59.8	35.7	91.5	73.3	72.2
Transformer	14M	48.1	57.3	36.1	63.4	42.7	92.6	77.7	76.1
Pointer-Generator	17M	51.9	**83.1**	**61.6**	64.2	40.4	97.0	78.2	75.4
Hierarchical Transformer	16M	52.3	58.6	38.4	66.9	48.4	93.7	80.1	79.2
Leader+Writer	17M	52.9	72.7	49.5	67.1	55.0	97.0	**86.4**	81.7
Leader+Pointer Writer	17M	55.3	78.7	54.3	68.8	56.9	97.0	83.2	83.0
Leader+Pointer Writer+RL	17M	**55.9**	79.3	54.9	**68.9**	**57.0**	**97.2**	83.0	**83.2**



(a) ROUGE-L scores on key facts (b) ratio= $\text{avg}\left(\frac{\text{generated summary length}}{\text{reference summary length}}\right)$

Figure 3: The left figure shows that our model captures the key facts better than the others. The right figure shows that Pointer-Generator tends to generate longer summaries than the reference summary.

The Leader-Writer net adopts the hierarchical Transformer decoder and its training is supervised by both summaries and key point sequences. The other models are only supervised by summaries. Our models achieves much better performance on the key point prediction. The auxiliary labels help our models predict the key point sequence well. So they can ensure the logic and integrity of the summaries.

We also assess the correctness of the key facts. Some key facts in the summary are important but very short. For example, the key facts *User approval* and *User disapproval* show the users' attitudes towards the service. Missing or misjudging the key facts leads to mistreatment of user's question. To measure the models' performance on fact correctness, we mainly calculate the ROUGE-L score on a small set of key points, including *{User approval, User disapproval, Close the order, · · · .}*. They are the key facts which subsequent agents concern most. The result is shown in Figure 3(a). Clearly, our model achieves the best performance. The non-hierarchical decoder tends to miss or misjudge the key facts, because they are trained only with summaries' supervision. In summary, the auxiliary key point sequence can help the model generate correct summaries.

4.5.2 The effects of the pointer mechanism. The pointer mechanism allows the model to directly copy tokens from the input sequence. Two facts demonstrate that it is extremely useful for dialogue summarization. First, the Pointer-Generator outperforms the other models in ROUGE-1 and ROUGE-2. Second, Leader+Pointer Writer and Leader+Writer are only different in the pointer mechanism, and Leader+Pointer Writer is significantly better than Leader+Writer in all metrics. There are two reasons. 1) When agents write summaries, they tend to use the same words that appear in the dialogue. Some agents directly copy parts of the dialogue into the summary. The pointer mechanism simulates the copying process of the agents. 2) ROUGE-1 and ROUGE-2 measure the n-gram recall of the references. From Figure 3(b), we observe that Pointer-Generator tends to generate longer summaries (with several duplicate sentences) than others, so it has higher ROUGE-1 and ROUGE-2 scores.

4.5.3 The effects of reinforcement loss. The effects of reinforcement loss can be demonstrated by comparing the Leader + Pointer Writer and the Leader+Pointer Writer+RL. They are different in the loss function. Leader+Pointer Writer+RL performs slightly better than Leader + Pointer Writer. From Eq. (21, 22), we can see that at the beginning stage of training, when the sampled actions are better than the greedy actions, the reinforcement loss encourages exploration of the parameter space, otherwise the reinforcement loss speeds up the convergence. However, a model with only the reinforcement loss converges extreme slowly.

4.5.4 The effects of hierarchical encoder. The effects of hierarchical encoder can be demonstrated by comparing the Hierarchical Transformer and Transformer. Hierarchical Transformer outperforms Transformer in summary generation (+4.2 BLEU, +3.5 ROUGE-L). There are two reasons: 1) Decoding the summary from flat dialogue token representations (about 400 in length) is harder than the dialogue utterance representations (about 20 in length); 2) A portion of the dialogues is useless for generating the summary, e.g., greetings and complaints. Determining an utterance's usefulness is much easier than determining every token's usefulness.

4.5.5 Model performance on different reference key point sequence lengths. From Figures 4(a)-4(b), we observe that the performance of all the models decreases with the reference key point sequence length. For dialogues whose reference key point sequences are short, our model is slightly better. While for dialogues whose reference key point sequence are long, our model significantly outperforms others. It confirms that our model has great advantage in generating summaries with complicated logic and many key points.

4.5.6 Model performance on different dialogue lengths. From Figures 4(c)-4(d), we observe that the performance of all the models decreases with the dialogue length. Our model is consistently better

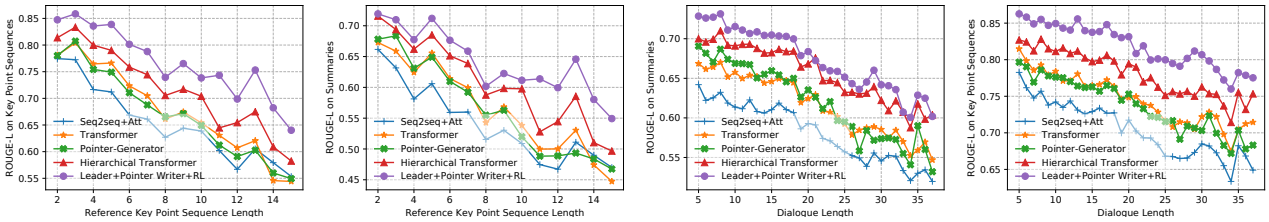


Figure 4: Model performances with respect to key point sequence length and dialogue length.

than the comparison models in different dialogue lengths. With respect to the ROUGE-L on key point sequence, our model decreases more slowly than others. It implies that the dialogue length has less impact on predicting the key point sequence.

5 CONCLUSIONS

This paper studies the important problem of dialogue summarization. To our best knowledge, it is one of the first work targeting on the logic and integrity of the summaries. We introduce the auxiliary key point sequences to describe the logic of the summaries. We also propose a *Leader-Writer* network which jointly decodes the key point sequences and use them to guide the generation of summaries. In experiments, we demonstrate that our model outperforms the state-of-the-art methods not only on BLEU and ROUGE-L, but also on logic and integrity. In future, the Leader-Writer will be deployed as a HTTP service for the online customer service in DiDi after passing the quality check. Moreover, an agent may refer to the user profile when writing a summary, and we plan to consider both the dialogue and the user profile in our model for better summarization.

REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[2] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2016. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086* (2016).

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[4] Siddhartha Banerjee, Prasenjit Mitra, and Kazunari Sugiyama. 2015. Multi-document abstractive summarization using ilp based multi-sentence compression.. In *Proc. IJCAI*. 1208–1214.

[5] Michele Banko, Vibhu O Mittal, and Michael J Witbrock. 2000. Headline generation based on statistical translation. In *Proc. ACL*. 318–325.

[6] Taylor Berg-Kirkpatrick, Dan Gillick, and Dan Klein. 2011. Jointly learning to extract and compress. In *Proc. ACL*. 481–490.

[7] Yen-Chun Chen and Mohit Bansal. 2018. Fast abstractive summarization with reinforce-selected sentence rewriting. *arXiv preprint arXiv:1805.11080* (2018).

[8] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Proc. EMNLP* (2014).

[9] Sumit Chopra, Michael Auli, and Alexander M Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In *Proc. NAACL*. 93–98.

[10] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In *Proc. NAACL-HLT*, Vol. 2. 615–621.

[11] Katja Filippova, Enrique Alfonseca, Carlos A Colmenares, Lukasz Kaiser, and Oriol Vinyals. 2015. Sentence compression by deletion with lstms. In *Proc. EMNLP*. 360–368.

[12] Chih-Wen Goo and Yun-Nung Chen. 2018. Abstractive dialogue summarization with sentence-gated modeling optimized by dialogue acts. *arXiv preprint arXiv:1809.05715* (2018).

[13] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proc. ACL*. 1631–1640.

[14] Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. In *Proc. ACL*. 140–149.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. CVPR*. 770–778.

[16] Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *arXiv preprint 1606.08415* (2016).

[17] Minghao Hu, Yuxing Peng, Zhen Huang, Xipeng Qiu, Furu Wei, and Ming Zhou. 2017. Reinforced mnemonic reader for machine reading comprehension. *arXiv preprint arXiv:1705.02798* (2017).

[18] Hongyan Jing and Kathleen R McKeown. 2000. Cut and paste based text summarization. In *Proc. NAACL*. 178–185.

[19] Diederik P Kingma and Jimmy Ba. 2014. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[20] Kevin Knight and Daniel Marcu. 2000. Statistics-based summarization-step one: sentence compression. *AAAI/IAAI* 2000 (2000), 703–710.

[21] Chin-Yew Lin. 2004. Rouge: a package for automatic evaluation of summaries. *Text Summarization Branches Out* (2004).

[22] Yishu Miao and Phil Blunsom. 2016. Language as a latent variable: discrete generative models for sentence compression. (2016), 319–328.

[23] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. (2016), 280–290.

[24] Tatsuro Oya, Yashar Mehdad, Giuseppe Carenini, and Raymond Ng. 2014. A template-based abstractive meeting summarization: leveraging summary and source text relationships. In *Proc. INLG*. 45–53.

[25] Haojie Pan, Junpei Zhou, Zhou Zhao, Yan Liu, Deng Cai, and Min Yang. 2018. Dial2desc: end-to-end dialogue description generation. *arXiv preprint arXiv:1811.00185* (2018).

[26] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proc. ACL*. 311–318.

[27] Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304* (2017).

[28] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732* (2015).

[29] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. In *Proc. CVPR*. 3.

[30] Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. (2015), 379–389.

[31] Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: summarization with pointer-generator networks. In *Proc. ACL*. 1073–1083.

[32] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[33] 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 *Proc. NeurIPS*. 5998–6008.

[34] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Proc. NeurIPS*. 2692–2700.

[35] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. 2017. Gated self-matching networks for reading comprehension and question answering. In *Proc. ACL*. 189–198.

[36] Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* 1, 2 (1989), 270–280.

[37] Shasha Xie, Yang Liu, and Hui Lin. 2008. Evaluating the effectiveness of features and sampling in extractive meeting summarization. In *Proc. SLT*. 157–160.

[38] Wenyuan Zeng, Wenjie Luo, Sanja Fidler, and Raquel Urtasun. 2016. Efficient summarization with read-again and copy mechanism. *arXiv preprint arXiv:1611.03382* (2016).