Summarizing Like Human: Edit-Based Text Summarization with Keywords

Yukang Liang[1,2(✉)], Junliang Guo[1], Yongxin Zhu[1,2], and Linli Xu[1,2]

[1] School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
{liangyukang,guojunll,zyx2016}@mail.ustc.edu.cn, linlixu@ustc.edu.cn
[2] State Key Laboratory of Cognitive Intelligence, Shijiazhuang, China

Abstract. Non-autoregressive generation (NAR) methods, which can generate all the target tokens in parallel, have been widely utilized in various tasks including text summarization. However, existing works have not fully considered the unique characteristics of the summarization task, which may lead to inferior results. Specifically, text summarization aims to generate a concise summary of the original document, resulting in a target sequence that is much shorter than the source. This poses a challenge of length prediction for NAR models. To address this issue, we propose an edit-based keywords-guided model named EditKSum: it utilizes the prominent keywords in the original text as a draft and then introduces editing operations such as repositioning, inserting, and deleting to refine them iteratively to get a summary. This model can implicitly achieve length prediction during the editing process and avoid the bias introduced by the imbalance of different editing operation frequencies during the training process. EditKSum is based on an encoder-decoder framework which is trained in an end-to-end manner and can be easily integrated with pre-trained language models. When both are equipped with pre-trained models, the proposed framework largely outperforms the existing NAR baselines on two benchmark summarization datasets and even achieves comparable performance with strong autoregressive (AR) baselines.

Keywords: Non-autoregressive · Summarization · Keywords-guided

1 Introduction

Non-autoregressive (NAR) generation [12,20] was first proposed in the neural machine translation task. Different from autoregressive (AR) models which generate tokens one by one and from left to right, NAR models can generate all the target tokens in parallel, which brings a significant increase in the generation speed. Benefiting from this, the NAR models have received considerable attention in recent years and have been applied to many other natural language generation tasks [15,52] including text summarization [25].

M. Wand et al. (Eds.): ICANN 2024, LNCS 15022, pp. 333–351, 2024.
https://doi.org/10.1007/978-3-031-72350-6_23

Text summarization [41,51] aims at creating a short summary that conveys the key information from a long document. Existing NAR models simply treat text summarization as a general text generation task [3,44] while ignoring its unique characteristics, which may lead to inferior results. In detail, text summarization is different from other text generation tasks from the following aspects: 1) the target sequence is much shorter than the source sequence, and 2) the source document explicitly or implicitly contains the information that the target needs.

The significant difference in lengths between input and output poses a challenge for length prediction, which is an important step in NAR models [46]. Some previous works adopt static [49] or dynamic [44] length prediction strategies, the performance of which can potentially impact the quality of generated summaries. Another line of research involves edit-based approaches and implicitly achieves length prediction during the process of applying editing operations. However, they generate summaries by editing from either the long original document [1] or an empty sequence [11], both of which suffer from the imbalanced frequency of different operations. For example, given the original document/empty sequence, the model will learn to delete/insert in most cases, which brings biases into model training. We verify the statement quantitatively by analyzing the correlation between the balance of different operations when generating a summary and the final performance as shown in Table 1.

Table 1. Results on the balance ratio and Rouge scores of models with different initial sequences y_0. "Rand Words" indicates randomly selected words that has the same length as keywords. "Rep." and "Ins." indicate the frequency of reposition and insertion, while "Bal." is the balance ratio between them.

y_0	Rep.	Ins.	Bal.	R-1	R-2	R-L
Empty	135	407	0.33	37.61	16.73	34.77
Keywords	48	48	**0.99**	**44.19**	**20.00**	**40.61**
Rand Words	116	94	0.81	32.70	9.17	29.25
Source	1289	10	0.01	24.74	10.96	22.47

To deal with the length prediction challenge in text summarization, we exploit the fact that the source contains the target information explicitly or implicitly in summarization. Previous studies [7,23] demonstrate that prominent keywords in the original text encompass crucial information required for generating summaries. So, we can regard the sequence of the keywords as the initial draft to edit from. Due to the closer alignment between the keyword sequence and the target sequence, this approach can effectively address the issue of imbalanced operation frequencies in previous editing-based methods.

Based on the above analysis, we propose an edit-based text summarization model that edits from prominent keywords, named EditKSum. Specifically, we build the model based on the encoder-decoder framework. The encoder takes the

source document as input, encodes its hidden representations as well as extracts keywords from it, which is achieved by inserting Feed-Forward Layers (FFN) on top of the encoder and introducing the corresponding keyword extraction loss functions. The decoder takes the extracted keywords as input and generates the summary by iteratively applying the editing operations including insertion, deletion, and repositioning. These operations can further modify and refine the extracted keywords, making the generated summary more coherent, fluent, and accurate.

Following previous works [3,26] that utilize pre-trained language models to boost the performance, we incorporate BERT [6] into the framework by first initializing the encoder and decode with a single BERT model, and then adding light-weight and specialized adapters on the encoder and decoder sides to learn the extraction and generation modules accordingly. The framework is jointly trained by optimizing the extraction and generation loss functions in an end-to-end manner, where the parameters of the pre-trained language model are frozen and only the adapters are tuned.

We evaluate our model on two benchmark datasets for text summarization, including CNN/DM [34] and Gigaword [41]. When equipped with pre-trained language models, the proposed model achieves 44.19/20.00/40.61 ROUGE-1/2/L scores on CNN/DM and 40.15/18.05/35.88 ROUGE-1/2/L scores on Gigaword, which outperforms the NAR baseline models with large margins. The proposed model also performs comparably with PEGASUS [50] and even better than AR baselines such as Transformer [45] and BertSum [26].

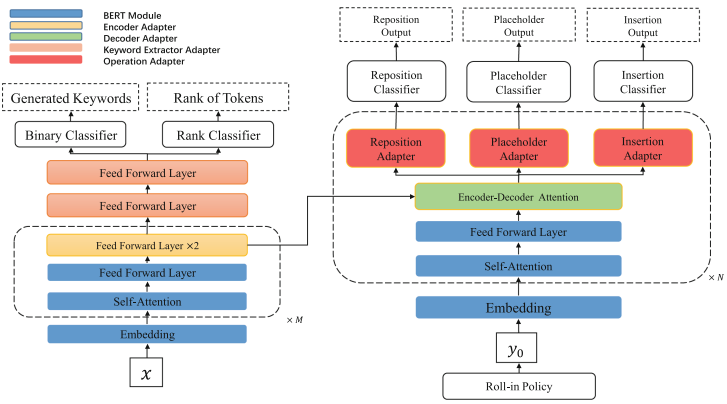

Fig. 1. The model architecture of EditKSum.

2 Related Works

2.1 Text Summarization

Text summarization is a widely studied task in natural language processing, which aims at generating a short summary that contains the key information from a long document. Generally, the existing works of text summarization can be classified into extractive summarization [47,51] and abstractive summarization [9,27,28].

The text summarization models are usually based on the encoder-decoder framework, which takes various forms including recurrent neural networks (RNN) at first [33,41] and Transformer [45] layers recently. With the rise of large-scale pre-trained models, both general [22,31] and specific [39,50] pre-trained models are widely utilized in the text summarization task, achieving impressive results thanks to the powerful representation ability of pre-trained models.

Most of the previous works generate summaries in an AR manner i.e., word-by-word and from left to right. However, AR generation faces the problem of slow inference. Recently, NAR generation methods [12] have been proposed, which can generate all tokens in the target sequence in parallel therefore greatly speeding up the generation process. However, when applied to the text summarization task [3,44], these models still underperform their AR counterparts significantly. In this paper, we propose an edit-based keywords guided model, which greatly improves the performance of NAR summarization models.

2.2 Non-autoregressive Generation

Non-autoregressive (NAR) generation is first proposed in the neural machine translation task [10,12,20]. Unlike AR models that generate tokens sequentially, NAR models typically predict the target sequence length first and then proceed to generate tokens in parallel. NAR models improve the inference speed at the potential price of a decrease in accuracy. Therefore, numerous research efforts [13,17,40,42] have been proposed to narrow the gap between NAR models and AR models in terms of the generation quality. Among these works, edit-based generation models can balance the inference speed and quality, as well as achieving length prediction implicitly by constructing output sequences through a series of atomic operations such as insertion [43], deletion [11], and reposition [48].

Owing to the considerable success of NAR models in machine translation, numerous researchers have recently extended this approach to a wider array of text generation tasks. These include grammatical error correction [37], automatic speech recognition [15], dialogue [52], and summarization [3,25,44].

On the task of text summarization, previous works do not take into account the discrepancy in lengths between the input and output sequences. As a result, difficulties are encountered in the step of length prediction, which in turn affect the summarization quality. In contrast, we generate summaries by editing from the keywords extracted from the source, which alleviates the above issues in principle.

3 Methodology

We introduce the proposed model EditKSum in this section, including an analysis of the editing operations in the existing edit-based methods, and the architecture of EditKSum. We start with the problem definition.

3.1 Problem Definition

Two sub-problems exist in our framework: text summarization and keyword extraction.

Text Summarization. Given a parallel training dataset $\langle \mathcal{X}, \mathcal{Y} \rangle$ which consists of pairs of source documents and reference summaries $\langle X, Y \rangle \in \langle \mathcal{X}, \mathcal{Y} \rangle$. Text summarization aims at generating the summary Y conditioned on the source document X. Model parameters θ are trained to maximize the conditional likelihood of the outputs.

$$\arg\max_{\theta} \sum_{\langle X,Y \rangle \in \langle \mathcal{X}, \mathcal{Y} \rangle} \log p(Y|X;\theta) \tag{1}$$

Keywords Extraction. Given a source sequence $X = (x_1, x_2, ..., x_n)$, the keyword extractor outputs keywords $G = (g_1, g_2, ..., g_l), g_t \in X$ from the source. Since there is no golden label to train the keyword extractor, we use the overlap of the original document and the reference summary as pseudo keyword labels to train the keyword extractor.

3.2 Analysis of Operations

We first conduct a detailed analysis on the relation between the balance of different operations and the model performance. Specifically, we train a model with three operations including insertion, deletion and repositioning following [48] on the CNN/DM dataset, and generate summaries by editing from different initial sequences, including empty, keywords, the full source document, as well as a baseline setting which is the randomly sampled words from the source document with the same number of words as that of keywords.

We evaluate the performance of the generated summary and calculate the frequency of different operations executed in each sample, and define the balance as the frequency ratio between insertion and reposition (we ensure the ratio is in $[0, 1]$ by taking the inverse if it is greater than 1). Since deletion is a special case of reposition, we treat them as one operation. The frequency is calculated as an average among the test set. The results are listed in Table 1.

As can be observed in Table 1, when generating summaries from keywords, the balance ratio is 0.99, indicating that the two operations are well balanced, and the model also achieves the best results.

In other cases, when generating from empty, the full source or the baseline setting, the operations are heavily biased with degraded performance. Specifically,

the Pearson correlation coefficient between the balance ratio and the Rouge-1 score is 0.76, indicating that the performance of edit-based models is highly co-related with the balance of operations. Moreover, the gap between generating from keywords and generating from random words underscores the role that keywords play in the task of text summarization.

3.3 Model Architecture

In this section, we will introduce the architecture of EditKSum. As illustrated in Fig. 1, our model is based on the encoder-decoder framework [2,45], where the encoder is utilized to encode the source document X, followed by a keyword extractor module. The decoder generates the target sequence Y conditioned on the source X and the decoder input y_0 (which will be introduced later) in an iterative edit-based manner.

Keyword Extractor. In this paper, we simply utilize two feed-forward layers as the keyword extractor, which can be viewed as an adapter, to keep consistent with the whole framework. Specifically, given a sequence of hidden states $H = (h_1, h_2, ..., h_n)$ which is encoded from the source document $X = (x_1, x_2, ..., x_n)$, the extractor calculates the informativeness of each token by:

$$H_{\text{ext}} = W_2 \cdot \sigma(W_1 \cdot H + b_1) + b_2 \tag{2}$$

where $H \in \mathbb{R}^{n \times d_h}$ and $H_{\text{ext}} \in \mathbb{R}^{n \times 2}$, W and b indicates the parameters of two FFN layers, $\sigma(\cdot)$ indicates the activation function which is ReLU [32] in this paper, and d_h is the dimension of the hidden representation.

Given H_{ext} which contains the informativeness of each token in the document, the extractor is expected to predict whether each token is a keyword or not. We introduce two loss functions to achieve that. The first is a binary classification loss based on cross-entropy:

$$L^1_{\text{ext}} = -\sum_{c=0}^{1} y^c \log \sigma(H^c_{\text{ext}}) \tag{3}$$

where $\sigma(\cdot)$ indicates the softmax function. And y^c is the real label of each class.

Besides predicting keywords directly, an alternative is to rank the positive examples ahead of the negative ones according to the informativeness scores H_{ext} of each token:

$$L^2_{\text{ext}} = \sum_{p_-, p_+ \in T} \max(0, 1 - H^{p_+}_{\text{ext}} + H^{p_-}_{\text{ext}}) \tag{4}$$

where p_+ and p_- denote the keywords and non-keywords respectively.

Then the loss function of the extractor can be written as:

$$L_{\text{ext}} = L^1_{\text{ext}} + L^2_{\text{ext}}. \tag{5}$$

Edit-Based Generation. We introduce the edit-based generation algorithm in this subsection. We regard it as a Markov Decision Process (MDP), which is defined by a quintuple $(\mathcal{Y}; \mathcal{A}; \mathcal{E}; \mathcal{R}; y_0)$, where $\mathcal{Y}$ is a set of discrete sequences $(y_0, y_1, ..., y_n)$. Each sequence $y_i \in \mathcal{Y}$ is a state in the iterative refinement process, and y_0 indicates the initial state. A denotes the set of actions, which includes operations of deletion, insertion, reposition and so on in text generation task. At each decoding iteration, the environment $\mathcal{E}$ receives an input $y_i \in \mathcal{Y}$, chooses an action $a \in \mathcal{A}$, outputs the refined sequence $y_{i+1} = \mathcal{E}(y_i, a)$ and gets a reward r. $\mathcal{R}$ denotes the reward function. Generally, $\mathcal{R}$ measures the distance $\mathcal{D}$ between the generated output and the ground-truth sequence, $R(y) = -D(y, y^*)$. The goal of edit-based generation is to learn a policy π that maps the current sequence y_i to a probability distribution over A, i.e., $\pi : y_i \rightarrow P(A)$.

It is worth mentioning that y_0 is crucial in edit-based generation. As shown in Table 1, the initial state y_0 affects the balance between actions as well as the final performance of the model. In this paper, y_0 is a sequence consisting of keywords rather than an empty sequence or a complete document as in previous summarization works.

Actions. We mainly follow previous works to determine the atomic operations [48], including reposition, deletion and insertion. These operations are suitable for handling keywords. Specifically, the reposition operation is able to change the order of each token. For each token y_i^t in a subsequence y^t, the reposition policy $\pi^{rep}(r|i, y^t)$ predicts an integer r and moves the r-th token into the current index, i.e., y_r^t will be placed in the i-th position after the operation. The deletion is a special case of reposition, i.e., when the policy predicts 0, the i-th token y_i^t will be deleted. As for the insertion operation, it is divided into two steps. Firstly, the placeholder policy $\pi^{plh}(p|i, y^t)$ predicts the number of placeholders p (the `[UNK]` symbol in implementation) to be inserted in each slot between y_i and y_{i+1}. Then, the token prediction policy $\pi^{tok}(j|i, y^t)$ replace the placeholder by predicting the content token y_j^t.

All of the three policies are implemented by the corresponding policy classifiers which are inserted on the top of decoder layers. Specifically, the reposition and deletion policy can be written as:

$$\pi_\theta^{rps}(r|i, y) = \text{softmax}(h_i \cdot [e_0; e_1; ...; e_n]), \tag{6}$$

where e_j denotes the embedding of the j-th token in the current subsequence, e_0 represents the deletion embedding, and $[\cdot;\cdot]$ represent the concatenation function. The placeholder and token prediction policy can be written as:

$$\pi_\theta^{plh}(p|i, y) = \text{softmax}([h_i; h_{i+1}] \cdot W_{\text{plh}}^T), \tag{7}$$

$$\pi_\theta^{tok}(j|i, y) = \text{softmax}(h_i \cdot W_{\text{tok}}^T), \tag{8}$$

where $W_{\text{plh}} \in \mathbb{R}^{(K+1)\times 2d_{\text{model}}}$ and $W_{\text{tok}} \in \mathbb{R}^{|V|\times d_{\text{model}}}$ are the parameters of the two policies to be trained, K is a hyper-parameter that represents the maximum number of tokens that can be inserted in each slot, $|V|$ is the vocabulary size and d_{model} is the hidden size of the model.

Adapters on Encoder and Decoder. Recent text summarization works [26, 50] show that large-scale pre-trained language models such as BERT [6] are able to improve the comprehension and generation capabilities of the summarization model. We incorporate BERT into our framework by first initializing the encoder and decoder with one pre-trained BERT model, and adding light-weight adapters into each pre-trained layer. We only fine-tune the adapters and freeze the pre-trained model while training, in order to reduce the scale of trainable parameters and alleviate the catastrophic forgetting problem [4,16].

Specifically, we mainly follow AB-Net [14] and design different adapter modules on the encoder and decoder sides, i.e., the encoder adapter is based on FFN layers, while the decoder adapter consists of a multi-head cross-attention module as well as two FFN layers. The keyword extractor can also be considered as an adapter on the encoder side.

3.4 Training and Inference

We utilize imitation learning to train the edit-based model, which consists of a roll-in policy π^{in} and a roll-out policy π^{out}. The roll-in policy generates an initial sequence to be edited from, and the roll-out policy provides the oracle demonstration to be learned from. In this paper, the roll-in policy for training reposition/insertion predictor is a stochastic mixture of the outputs of the insertion/reposition predictor and a noised version of the reference y^* with random word dropping and shuffle.

And the oracle roll-out policy is determined by the Levenshtein edit distance [21], which indicates the minimum number of editing operations required to convert from one sequence to the other. We define the oracle policy π^* as the optimal action to transform y_0 to y^*, and train the model policy π by minimizing the KL divergence between the action distributions produced by π and π^* [5]:

$$L_{\text{gen}} = D_{KL}[\pi^*(a|y, y*)||\pi(a|y)] \tag{9}$$

In summary, we train the proposed framework by minimizing the linear combination of the keyword extractor loss L_{ext} and the the summary generator loss L_{gen}:

$$L = L_{\text{ext}} + L_{\text{gen}}. \tag{10}$$

At the inference stage, we first obtain keywords from the keyword extractor conditioned on the source document. Then, we join the keywords together with the order in the source document to get the initial sequence. Then, we apply a sequence of actions $(\boldsymbol{a}^1, \boldsymbol{a}^2, ...)$ in a circulation of insertions and repositions, e.g., as $(\boldsymbol{p}^1, \boldsymbol{t}^1, \boldsymbol{r}^1; \boldsymbol{p}^2, \boldsymbol{t}^2, \boldsymbol{r}^2, ..)$, to refine the initial sequence iteratively. The refinement will be terminated when either two consecutive refinement iterations return the same output or a maximum number of iterations is reached. We provide an illustration of the generation process in Fig. 2.

4 Experiments

4.1 Datasets

We conduct experiments on two mainstream public datasets of the text summarization task, CNN/DM [34] and Gigaword [41].

CNN/DM is a widely used text summarization corpus consisting of pairs of news articles and the corresponding multi-sentence summaries. We use its non-anonymized version which contains 287112 training pairs, 13367 validation pairs and 11490 test pairs. The source documents and the reference summaries consist of an average number of 781 and 56 tokens respectively. We truncate the source documents that exceed the maximum length (which is 512 in our implementation).

Gigaword is a sentence summarization corpus with short documents and summaries, which contains 3.8M/190K/2K training/validation/test samples. The average numbers of tokens in the documents and the summaries are 31.4 and 8.3 respectively.

It is worth mentioning that **X-Sum** [35] is also a widely used text summarization dataset. However, due to its abstractive nature, i.e. the average overlap rate between the source documents and summaries is relatively low, it is not suitable for the edit-based models [29]. In addition, we regard the overlap between a given source document and the corresponding reference summary as the golden keywords. Too few coincidence tokens also induce difficulties to train the key-

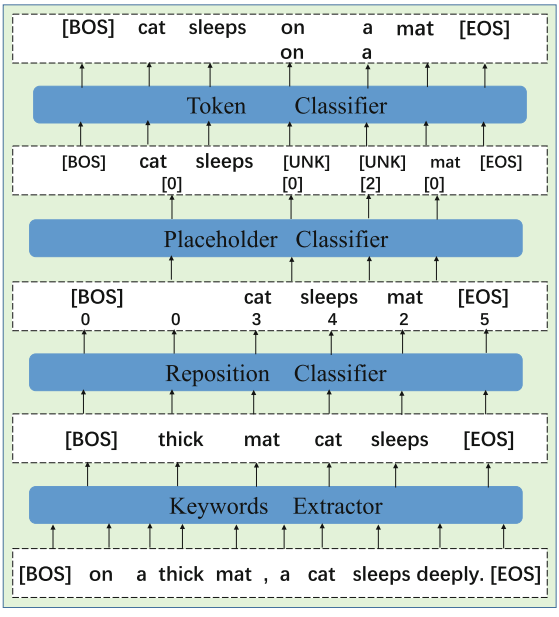

Fig. 2. An illustration of the generation process of the proposed model.

word extractor. Based on the above reasons, we do not conduct experiments on the X-Sum dataset.

4.2 Experimental Setup

Following previous edit-based and non-autoregressive methods [11,12,48], we apply sequence-level knowledge distillation [18] to distill the original training set, in order to alleviate the multi-modality problem of non-autoregressive models. Specifically, we distill CNN/DM by an autoregressive pre-trained model `bart-large-cnn` and use the raw training set on Gigaword as it is much shorter. Then we tokenize the datasets by the `bert-cased` tokenizer, resulting in a dictionary with 30K tokens.

Evaluation Metrics. Following previous text summarization models, we used ROUGE [24] as the automatic evaluation metric which reports the precision/recall/F scores of the 1-gram/2-gram/longest common subsequences that are overlapped between the generated and the reference summary.

Model Configurations. We build our model based on the `bert-base-cased` model ($n_{\text{layers}} = 12, n_{\text{heads}} = 12, d_{\text{hidden}} = 768, d_{\text{FFN}} = 3072$). We set $d_{\text{FFN}} = 2048$ and $d_{\text{hidden}} = 768$ for FFN and the attention based adapters. As for the keyword extractor, we set $d_{\text{FFN}} = 512$. Besides, we apply dropout on the encoders and decoders with a probability of 0.1.

We train our framework on 4 Nvidia 3090 GPUs for 100 epochs and it takes 2~3 days to converge. The batch size is set as 8000 tokens. The Model is optimized with Adam [19] with a beta value of (0.9, 0.98). We set the learning rate to 5e-4 and use 10% of the epochs to warm-up with the initial learning rate as 1e-7. We implement our model and baselines on fairseq [38]. The code and pre-trained models will be released upon acceptance.

Baselines. To make a comprehensive comparison, we consider the following baselines. **NAUS** [25] is a specialized unsupervised NAR model designed for text summarization tasks; **BERT+CRF-NAT** [44] employs BERT as the backbone of a NAR model and proposes an elegant decoding mechanism to help length prediction; **BANG** [3] is a large-scale pre-trained model which simultaneously supports AR, NAR and semi-NAR generation. We finetune the pre-trained model provided for another 20 epochs on the CNNDM and Gigaword datasets; **LevT** [11] is a classical edit-based generation model which applies deletion and insertion operations on empty sequences to generate the targets; **EDITOR** [48] is an edit-based model with reposition, insertion and deletion operations. We compare with it to show the effectiveness of introducing keywords. We also initialize the encoder of EDITOR with BERT to make a fair comparison; **Transformer** [45] serves as a widely used AR baseline; **BertSum** [26] is an AR summarization model with the encoder initialized with BERT; **PEGASUS** [50] is a powerful AR baseline specifically designed as a pre-trained model for summarization. It may not be fair for EditKSum to compare with PEGASUS, but we chose it to demonstrate our strong capabilities of EditKSum.

Table 2. The main results on the CNNDM and the Gigaword datasets. "w/ BERT" or "w/o BERT" indicates whether the model's encoder and decoder are initialized with a single BERT model or not. he top scores are highlighted in **bold**, while the second-best scores are underlined. † signifies that the results for at least one dataset are sourced from the original papers or public leaderboards.

Model	CNNDM			Gigaword		
	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-1	ROUGE-2	ROUGE-L
NAUS [25] †	-	-	-	28.55	9.97	25.78
Bert+CRF-NAT [44]†	-	-	-	35.05	16.48	33.28
BANG [3]	35.77	12.87	33.07	-	-	-
LevT [11] †	36.35	15.56	33.72	36.14	17.14	34.34
EDITOR [48]	37.61	16.74	34.77	36.89	16.30	34.28
EDITOR w/ BERT	38.84	17.63	35.93	38.10	17.50	35.43
Transformer [45]†	39.50	16.06	36.63	37.57	18.90	34.69
BERTSum [26]†	42.00	19.44	38.98	-	-	-
PEGASUS [50]†	44.17	**21.47**	**41.11**	39.12	**19.86**	**36.24**
EditKSum w/o BERT	43.02	18.94	39.39	38.22	16.57	34.35
EditKSum	**44.19**	20.00	40.61	**40.15**	18.05	35.88

4.3 Main Results

The main results on the CNN/DM and Gigaword datasets are summarized in Table 2. The upper group of the table shows the results of NAR (including edit-based) models, while the bottom group shows the AR model results. Overall, the proposed EditKSum model significantly outperforms most of the NAR and AR baselines on the two datasets, while achieving comparable performance in ROUGE-1 with powerful AR pre-trained models PEGASUS. Among the NAR methods, our model already outperforms the baselines with large margins without the help of the pre-trained language model. When equipped with BERT, our model achieves improvements of 5.35/2.37/4.68 Rouge scores over EDITOR w/ BERT, showing the effectiveness of editing with keywords. EditKSum also outperforms BANG, BERT+CRF-NAT, and BERTSum, both of them are boosted by a large-scale pre-trained model.

Table 3. The average inference latency (ms) and the number of iterations of EditKSum and baselines. For edit-based models, one iteration corresponds to completing a cycle of actions, whereas for AR models, the number of iterations is equal to the length of the generated sequence.The top scores are highlighted in **bold**.

Datasets	CNNDM		Gigaword	
	Lat.	Iter.	Lat.	Iter.
EDITOR	90.8	3.4	70.3	2.7
LevT	88.4	3.4	74.5	2.9
Transformer	576.6	52.3	147.1	14.2
EditKSum	**68.6**	**2.4**	**60.3**	**2.3**

4.4 Inference Speed

In this section, we evaluate the inference speed of EditKSum. For a fair comparison, all models have been configured with a beam size of 1 and a batch size of 1. Moreover, the network hyperparameters are consistent across all models.

As shown in Table 3, we measured the inference latency and the number of iterations for different models on the CNN/DM and Gigaword test sets.

Compared with AR or NAR baseline models, thanks to editing from keywords, EditKSum can generate summaries with fewer iterations. As a result, it requires less inference latency to generate a summary.

It is worth noting that for the CNNDM dataset, due to the longer length of the original text, the effect of EditKSum in improving generation efficiency will be more pronounced. Specifically, EditKSum achieves 7.40/0.28 times speedup over Transformer/LevT in the CNNDM dataset.

5 Analysis

5.1 Improvement of Keywords-Guided Summarization Models

For autoregressive models, the keywords are utilized by appending them to the beginning of the summary and taking as the prompt to guide the following generation. However, in this way, the model are not able to edit if the keywords contain errors, while the proposed edit with keywords method can alleviate this problem. To verify the statement, we compare the proposed EditKSum with three strong keywords guided autoregressive methods, i.e., CAS [30], GSum [8]and FROST [36]. As different models are based on different pretrained models, to make a fair comparison, we only consider the impact of keywords by comparing the performance of the model w/ or w/o the help of keywords. The results are visually illustrated in Fig. 3, where the blue and orange columns represent the ROUGE-1 scores on the CNNDM dataset of the model without and with keywords, respectively. Obviously, the proposed model achieves the most significant improvement over its counterparts. Specifically, EditKSum obtains an absolute improvement of 5.35/2.37/4.68 on ROUGE-1/2/L over the initialized baseline, with 13.77% promotion in ROUGE-1, which is far ahead of the other models. The results show that the proposed method benefits more from keywords by utilizing the prominent information contained in it, while also correcting the errors by changing the orders as well as removing inappropriate ones.

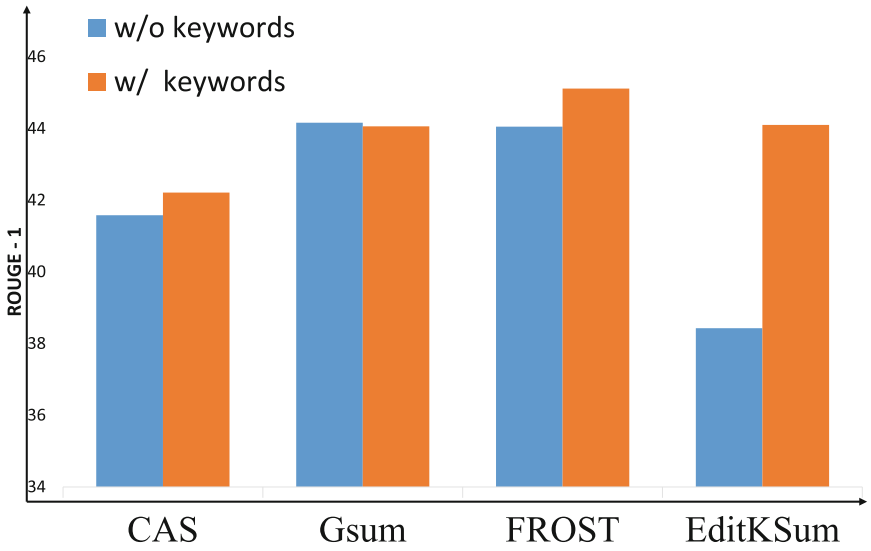

Fig. 3. ROUGE-1 scores of keywords-guided summarization models. Blue/orange bars indicate the results w/ or w/o the help of keywords. (Color figure online)

5.2 Comparison of Different Keyword Extraction Strategies

In EditKSum, the keyword extractor evaluates the informativeness of all the tokens in the source document, based on which the keywords are selected. We adopt two different strategies to select keywords, one is to take the top N tokens with the highest scores as keywords, the other is to set a threshold ϵ and select the tokens with scores exceeding the threshold. We investigate the effect of the keyword selection strategies and hyper-parameter settings. As we can see in Table 4, when we adopt the threshold strategy and ϵ equals 1.0, the best ROUGE-1/L scores is achieved (best ROUGE-2 scores is achieved when ϵ equals 0.8). Although the best result obtained by the topN strategy is similar to the threshold strategy, the selection of N has a great influence on the result. In contrast, the threshold strategy is more robust. So the default setting for our experiments is threshold-1.0.

Table 4. Results of EditKSum on CNN/DM dataset of different keyword extraction strategies. **TopN** and **Threshold** indicates corresponding keyword extraction strategy. The following number represents the hyper-parameter N or ϵ. The highest numbers are in **bold**

Strategy	Hyper-Para.	R-1	R-2	R-L
TopN	20	40.76	18.99	37.50
	40	42.92	19.73	39.32
	60	43.98	19.78	40.17
	80	42.45	18.61	38.29
Threshold	0.4	43.15	19.82	39.54
	0.6	43.77	19.96	40.04
	0.8	44.05	19.93	40.25
	1.0	**44.19**	**20.00**	**40.61**

5.3 The Influence the Abstractiveness of Datasets

To quantify the abstractiveness of different datasets, we measured the overlap rate between source and target tokens among them, a lower overlap rate signifies a higher abstractiveness level.

Table 5. The Influence the Abstractiveness of Datasets

Datasets	CNNDM	Gigaword	XLSum	NewsRoom	XSum
Overlap Rate	91.8%	58.82%	59.85%	80.61%	47.28%
PEGASUS [50]	21.47	19.86	18.28	33.39	24.56
EditKSum	20.00	18.05	7.61	29.02	0.57
Relative Gap	6.85%	9.11%	58.37%	13.09%	97.68%

As shown in Table 5, the overlap rate of CNNDM/Gigaword/XLSum/Newsroom/XSum is 91.78%/58.82%/59.85%/80.61%/47.28% respectively. The relative gap in ROUGE-2 score between EditKSum and PEGASUS was found to have a high Person's correlation coefficient (0.76) with the overlap rate, indicating a strong correlation between model performance and abstractiveness level.

5.4 Case Study

In order to show the generation process and demonstrate the powerful editing ability of our model more clearly, we selected a concrete example and display its specific generation process in Table 6.

As the this example shows, EditKSum extracts the salient tokens in the source document at first, then deletes the inappropriate token **visit** and inserts the correct tokens including **'s**, **visits**, **j**, and **##erus** in the correct positions to make a summary with high quality.

Table 6. Case study on Gigaword dataset. "Source", "Target" and "Hypo" represents the source document, reference summary and generated summary respectively. "Tokenize" and "Extract Keywords" mean tokenizing the source document and extracting keywords from it. Steps means the iterations of sequence. And "Untokenize" means detokenize the final sequence to get generated summary.

Type	Text
Source	jordan 's crown prince hassan ibn talal arrived tuesday for his first visit to jerusalem and was to pay his condolences to the widow of assassinated prime minister yitzhak rabin .
Target	jordan ' s crown prince makes first visit to jerusalem
Hypo	jordan ' s crown prince visits troubled jerusalem city
Tokenize	j ##ord ##an ' s crown prince has ##san ibn ta ##lal arrived t ##ues ##day for his first visit to j ##erus ##ale ##m and was to pay his con ##do ##len ##ces to the widow of assassinated prime minister y ##itz ##hak r ##abi ##n .
Extract Keywords	j ##ord ##an s crown prince visit ##erus ##m
Step0	j ##ord ##an s crown prince visit ##erus ##m
Step1	j ##ord ##an s crown prince ##erus ##m
Step2	j ##ord ##an [UNK] s crown prince [UNK] [UNK] ##erus [UNK] ##m
Step3	j ##ord ##an ' s crown prince visits j ##erus ##ale ##m
Step4	j ##ord ##an ' s crown prince visits j ##erus ##ale ##m
Step5	j ##ord ##an ' s crown prince visits [UNK] j ##erus ##ale ##m
Step6	j ##ord ##an ' s crown prince visits troubled j ##erus ##ale ##m
Step7	j ##ord ##an ' s crown prince visits troubled j ##erus ##ale ##m
Step8	j ##ord ##an ' s crown prince visits troubled j ##erus ##ale ##m [UNK]
Step9	j ##ord ##an ' s crown prince visits troubled j ##erus ##ale ##m city
Untokenize	jordan ' s crown prince visits troubled jerusalem city

6 Conclusion

In this paper, we propose an edit-based model with keywords named EditKSum to slove the length predictiong issue in the text summarization task. We first conduct a thorough analysis validating a strong correlation between the balance of different operations and the generation performance, i.e., more balanced operations imply better results. Considering that the salience words in the source document can provide useful information when generating summaries, we propose to edit from keywords by introducing a keyword extractor to extract prominent words from the source document, which are taken as the initial state for the edit-based decoder. In experiments, on two benchmark text summarization datasets, we show that the proposed EditKSum can achieve significant improvements over other NAR models, while achieving comparable performance to strong NR models with faster decoding speed. For future work, we plan to apply our method to other text generation tasks, such as text simplification, dialogue, and story generation.

Acknowledgments. This research was supported by the National Key Research and Development Program of China (Grant No. 2022YFB3103100), the National Natural Science Foundation of China (Grant No. 62276245).

References

1. Agrawal, S., Carpuat, M.: An imitation learning curriculum for text editing with non-autoregressive models. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Dublin, Ireland, pp. 7550–7563. Association for Computational Linguistics (2022). https://doi.org/10.18653/v1/2022.acl-long.520
2. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014)
3. Qi, W., et al.: BANG: bridging autoregressive and non-autoregressive generation with large scale pretraining. In: International Conference on Machine Learning, PMLR, pp. 8630–8639 (2021)
4. Bapna, A., Firat, O.: Simple, scalable adaptation for neural machine translation. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Hong Kong, China, pp. 1538–1548. Association for Computational Linguistics (2019). https://doi.org/10.18653/v1/D19-1165
5. Brantley, K., et al.: Non-monotonic sequential text generation. In International Conference on Machine Learning, pp. 6716–6726. PMLR (2019)
6. Devlin, J., et al.: BERT: pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT 2019: Minneapolis, Minnesota, June 2019, pp. 4171–4186. Association for Computational Linguistics (2019). https://doi.org/10.18653/v1/N19-1423
7. Dou, Z.Y., et al.: GSum: a general framework for guided neural abstractive summarization. In: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, June 2021, pp. 4830–4842. Association for Computational Linguistics (2021). https://doi.org/10.18653/v1/2021.naacl-main.384
8. Dou, Z.Y., et al.: GSum: a general framework for guided neural abstractive summarization. In: Toutanova, K., et al. (eds.) Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 4830–4842. Association for Computational Linguistics (2021). https://doi.org/10.18653/v1/2021.naacl-main.384
9. Gehrmann, S., Deng, Y., Rush, A.M.: Bottom-up abstractive summarization. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, Oct. 2018, pp. 4098–4109. Association for Computational Linguistics (2018). https://doi.org/10.18653/v1/D18-1443
10. Ghazvininejad, M., et al.: Mask-predict: parallel decoding of conditional masked language models. In: EMNLP-IJCNLP, Hong Kong, China, Nov. 2019, pp. 6112–6121. Association for Computational Linguistics (2019). https://doi.org/10.18653/v1/D19-1633
11. Gu, J., Wang, C., Zhao, J.: Levenshtein transformer. In: NIPS '23: Proceedings of the 37th International Conference on Neural Information Processing Systems, vol. 32. Curran Associates, Inc. (2019)
12. Gu, J., et al.: Non-autoregressive neural machine translation. arXiv preprint arXiv:1711.02281 (2018)

13. Guo, J., Xu, L., Chen, E.: Jointly masked sequence-to-sequence model for non-autoregressive neural machine translation. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, July 2020, pp. 376–385. Association for Computational Linguistics (2020). https://doi.org/10.18653/v1/2020.acl-main.36
14. Guo, J., et al.: Incorporating BERT into parallel sequence decoding with adapters. In: Advances in Neural Information Processing Systems, vol. 33, pp. 10843–10854. Curran Associates, Inc. (2020)
15. Higuchi, Y., et al.: A comparative study on non-autoregressive modelings for speech-to-text generation. In: IEEE ASRU, Dec. 2021, pp. 47–54. IEEE (2021). https://doi.org/10.1109/ASRU51503.2021.9688157
16. Houlsby, N., et al.: Parameter-efficient transfer learning for NLP". In: ICML, May 2019, pp. 2790–2799. PMLR (2019)
17. Wang, Y., Geng, X.: Improving non-autoregressive neural machine translation via modeling localness". In: Proceedings of the 29th International Conference on Computational Linguistics, Gyeongju, Republic of Korea, Oct. 2022, pp. 5217–5226. International Committee on Computational Linguistics (2022)
18. Yoon Kim and Alexander M. Rush. "Sequence-Level Knowledge Distillation". In: Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 1317–1327. DOI: https://doi.org/10.18653/v1/D16-1139. (Visited on 06/23/2023)
19. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2015)
20. Lee, J., Mansimov, E., Cho, K.: Deterministic non-autoregressive neural sequence modeling by iterative refinement. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, Oct. 2018, pp. 1173–1182. Association for Computational Linguistics (2018). https://doi.org/10.18653/v1/D18-1149
21. Levenshtein, V.: Binary codes capable of correcting deletions, insertions, and reversals'. Soviet Phys. Doklady **10**, 707–710 (1965)
22. Lewis, M., et al.: BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In: Annual Meeting of the Association for Computational Linguistics, July 2020, pp. 7871–7880. Association for Computational Linguistics (2020). https://doi.org/10.18653/v1/2020.acl-main.703
23. Li, H., et al.: Keywords-guided abstractive sentence summarization. Artif. Intell. **34**(5), 8196–8203 (2020)
24. Lin, C.Y.: ROUGE: a package for automatic evaluation of summaries. In: Text Summarization Branches Out, pp. 74–81. Association for Computational Linguistics, Barcelona, Spain (2004)
25. Liu, P., Huang, C., Mou, L.: Learning non-autoregressive models from search for unsupervised sentence summarization. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Dublin, Ireland, pp. 7916–7929. Association for Computational Linguistics (2022). https://doi.org/10.18653/v1/2022.acl-long.545
26. Liu, Y., Lapata, M.: Text summarization with pretrained encoders. In: EMNLP-IJCNLP, Hong Kong, China, pp. 3730–3740. Association for Computational Linguistics (2019). https://doi.org/10.18653/v1/D19-1387
27. Liu, Y., Liu, P.: SimCLS: a simple framework for contrastive learning of abstractive summarization. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural

Language Processing (Volume 2: Short Papers), pp. 1065–1072. Association for Computational Linguistics (2021). https://doi.org/10.18653/v1/2021.acl-short.135
28. Liu, Y., et al.: BRIO: bringing order to abstractive summarization. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Dublin, Ireland, pp. 2890–2903. Association for Computational Linguistics (2022). https://doi.org/10.18653/v1/2022.acl-long.207
29. Malmi, E., et al.: Text generation with text-editing models. In: Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Tutorial Abstracts, Seattle, United States, July 2022, pp. 1–7. Association for Computational Linguistics (2022). https://doi.org/10.18653/v1/2022.naacl-tutorials.1
30. Mao, Y., et al.: Constrained abstractive summarization: preserving factual consistency with constrained generation. arXiv preprint arXiv:2010.12723 (2020)
31. Song, K.: MASS: masked sequence to sequence pre-training for language generation. In: PMLR, pp. 5926–5936 (2019)
32. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp. 807–814 (2010). https://icml.cc/Conferences/2010/papers/432.pdf
33. Nallapati, R., Zhai, F., Zhou, B.: SummaRuNNer: a recurrent neural network based sequence model for extractive summarization of documents. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31 (2016). https://doi.org/10.1609/aaai.v31i1.10958
34. Nallapati, R., et al.: Abstractive text summarization using sequence-to-sequence RNNs and beyond. In: Proceedings of the AAAI Conference on artificial intelligence, Berlin, Germany, pp. 280–290. Association for Computational Linguistics (2017). https://doi.org/10.18653/v1/K16-1028
35. Narayan, S., Cohen, S.B., Lapata, M.: Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, pp. 1797–1807. ACL (2018). https://doi.org/10.18653/v1/D18-1206
36. Narayan, S., et al.: Planning with learned entity prompts for abstractive summarization. In: Roark, B., Nenkova, A. (eds.) Transactions of the Association for Computational Linguistics, vol. 9, pp. 1475–1492 (2021). https://doi.org/10.1162/tacl_a_00438
37. Omelianchuk, K., et al.: GECToR – grammatical error correction: tag, not rewrite. In: Workshop on Innovative Use of NLP for Building Educational Applications, Seattle, WA, USA. Association for Computational Linguistics, pp. 163–170 (2020). https://doi.org/10.18653/v1/2020.bea-1.16
38. Ott, M., et al.: fairseq: a fast, extensible toolkit for sequence modeling. In: North American Chapter of the Association for Computational Linguistics, Minneapolis, Minnesota, pp. 48–53. Association for Computational Linguistics (2019). https://doi.org/10.18653/v1/N19-4009
39. Qi, W., et al.: ProphetNet: predicting future N-gram for sequence-to-sequence pre-training. In: North American Chapter of the Association for Computational Linguistics, pp. 2401–2410 (2020). https://doi.org/10.18653/v1/2020.findings-emnlp.217
40. Qian, L., et al.: Glancing transformer for non-autoregressive neural machine translation. In: Conference of the Association for Computational Linguistics, pp. 1993–2003 (2021). https://doi.org/10.18653/v1/2021.acl-long.155

41. Rush, A.M., Chopra, S., Weston, J.: A neural attention model for abstractive sentence summarization. In: Association for Computational Linguistics, Lisbon, Portugal, pp. 379–389 (2015). https://doi.org/10.18653/v1/D15-1044
42. Saharia, C., et al.: Non-autoregressive machine translation with latent alignments. In: Association for Computational Linguistics, pp. 1098–1108 (2020). https://doi.org/10.18653/v1/2020.emnlp-main.83
43. Stern, M., et al.: Insertion transformer: flexible sequence generation via insertion operations. In: PMLR, pp. 5976–5985 (2019)
44. Su, Y., et al.: Non-autoregressive text generation with pre-trained language models. In: Association for Computational Linguistics, pp. 234–243 (2021). https://doi.org/10.18653/v1/2021.eacl-main.18
45. Vaswani, A., et al.: Attention is all you need. In: NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems, vol. 30. Curran Associates, Inc. (2017)
46. Xiao, Y., et al.: A survey on non-autoregressive generation for neural machine translation and beyond. In: IEEE Transactions on Pattern Analysis and Machine Intelligence (2023). https://doi.org/10.1109/TPAMI.2023.3277122
47. Xu, J., et al.: Discourse-aware neural extractive text summarization. In: Association for Computational Linguistics, pp. 5021–5031 (2020). https://doi.org/10.18653/v1/2020.acl-main.451
48. Xu, W., Carpuat, M.: EDITOR: an edit-based transformer with repositioning for neural machine translation with soft lexical constraints. In: Transactions of the Association for Computational Linguistics, vol. 9, pp. 311–328 (2021). https://doi.org/10.1162/tacl_a_00368
49. Kexin Yang et al. "POS-Constrained Parallel Decoding for Non-Autoregressive Generation". In: Online: Association for Computational Linguistics, Aug. 2021, pp. 5990–6000. DOIhttps://doi.org/10.18653/v1/2021.acl-long.467. (Visited on 05/13/2023)
50. Zhang, J., et al.: PEGASUS: pre-training with extracted gap-sentences for abstractive summarization. In: PMLR, pp. 11328–11339 (2020)
51. Zhong, M., et al.: Extractive summarization as text matching. In: Association for Computational Linguistics, pp. 6197–6208 (2020). https://doi.org/10.18653/v1/2020.acl-main.552
52. Zou, Y., et al.: Thinking clearly, talking fast: concept-guided non-autoregressive generation for open-domain dialogue systems. In: Conference of the Association for Computational Linguistics, Punta Cana, Dominican Republic. Association for Computational Linguistics, pp. 2215–2226 (2021). https://doi.org/10.18653/v1/2021.emnlp-main.169