Summarizing Like Human: Edit-Based Text Summarization with Keywords

Anonymous ACL submission

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.

1 Introduction

Non-autoregressive (NAR) generation (Gu et al., 2018; Lee et al., 2018) 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 (Zou et al., 2021; Higuchi et al., 2021) including text summarization (Liu et al., 2022a).

Text summarization (Rush et al., 2015; Zhong et al., 2020) 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 (Qi et al., 2021; Su et al., 2021) 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 (Xiao et al., 2023). Some previous works adopt static (Yang et al., 2021) or dynamic (Su et al., 2021) 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 (Agrawal and Carpuat, 2022) or an empty sequence (Gu et al., 2019), 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.

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 (Li et al., 2020; Dou et al., 2021) 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 (Liu and Lapata, 2019; Qi et al., 2021) that utilize pre-trained language models to boost the performance, we incorporate BERT (Devlin et al., 2019) into the framework by first initializing the encoder and decode with a single BERT model, and then adding lightweight 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 (Nallapati et al., 2016b) and Gigaword (Rush et al., 2015). 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 (Zhang et al., 2020) and even better than AR baselines such as Transformer (Vaswani et al., 2017) and BertSum (Liu and Lapata, 2019).

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

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.

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 (Xu et al., 2020; Zhong et al., 2020) and abstractive summarization (Gehrmann et al., 2018; Liu and Liu, 2021; Liu et al., 2022b).

The text summarization models are usually based on the encoder-decoder framework, which takes various forms including recurrent neural networks (RNN) at first (Rush et al., 2015; Nallapati et al., 2016a) and Transformer (Vaswani et al., 2017) layers recently. With the rise of large-scale pre-trained models, both general (Lewis et al., 2020; Song et al., 2019) and specific (Zhang et al., 2020; Qi et al., 2020) 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 (Gu et al., 2018) 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 (Qi et al., 2021; Su et al., 2021), 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

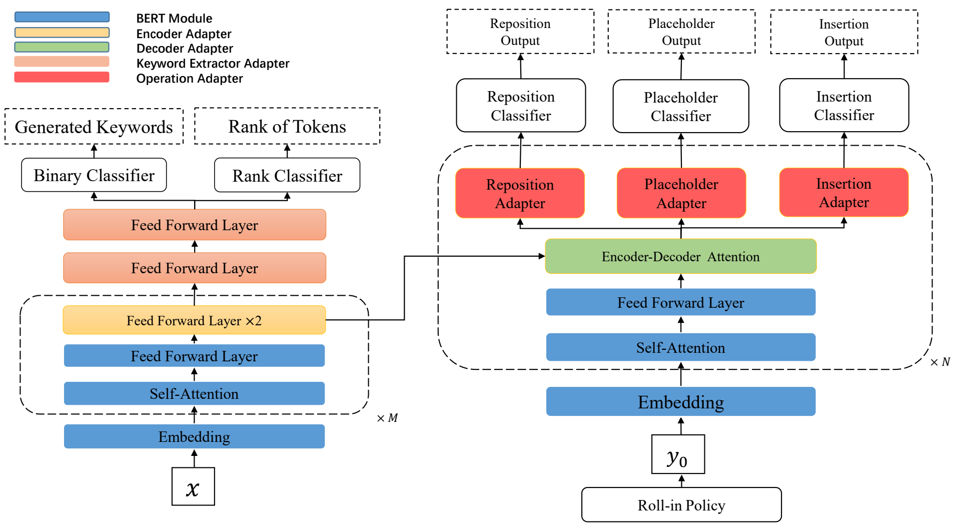

Figure 1: The model architecture of EditKSum.

models.

2.2 Non-autoregressive Generation

Non-autoregressive (NAR) generation is first proposed in the neural machine translation task (Gu et al., 2018; Lee et al., 2018; Ghazvininejad et al., 2019). 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 (Qian et al., 2021; Saharia et al., 2020; Guo et al., 2020a; Wang and Geng, 2022) 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 (Stern et al., 2019), deletion (Gu et al., 2019), and reposition (Xu and Carpuat, 2021).

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 (Omelianchuk et al., 2020), automatic speech recognition (Higuchi et al., 2021), dialogue (Zou et al., 2021), and summarization (Liu et al., 2022a; Qi et al., 2021; Su et al., 2021).

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.

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

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 (Xu and Carpuat, 2021) 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 Figure 1, our model is based on the encoder-decoder framework (Bahdanau et al., 2014; Vaswani et al., 2017), 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.

3.3.1 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:

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

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 (Nair and Hinton, 2010) 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:

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

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:

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

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

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

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

3.3.2 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 (Xu and Carpuat, 2021), 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:

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

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:

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

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

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.

3.3.3 Adapters on Encoder and Decoder

Recent text summarization works (Liu and Lapata, 2019; Zhang et al., 2020) show that large-scale pre-trained language models such as BERT (Devlin et al., 2019) 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 (Bapna and Firat, 2019; Houlsby et al., 2019).

Specifically, we mainly follow AB-Net (Guo et al., 2020b) 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 (Levenshtein, 1965), 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

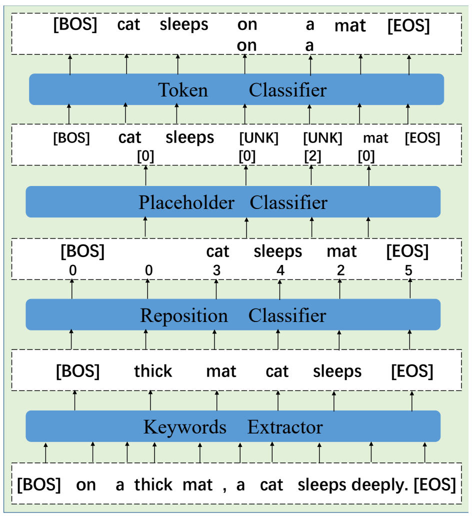
[BOS] cat sleeps on a mat [EOS]
on a
Token Classifier
[BOS] cat sleeps [UNK] [UNK] mat [EOS]
[0] [0] [2] [0]
Placeholder Classifier
[BOS] cat sleeps mat [EOS]
0 0 3 4 2 5
Reposition Classifier
[BOS] thick mat cat sleeps [EOS]
Keywords Extractor
[BOS] on a thick mat , a cat sleeps deeply. [EOS]

Figure 2: An illustration of the generation process of the proposed model.

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^* (Brantley et al., 2019):

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

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 Figure 2.

4 Experiments

4.1 Datasets

We conduct experiments on two mainstream public datasets of the text summarization task, CNN/DM (Nallapati et al., 2016b) and Gigaword (Rush et al., 2015).

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** (Narayan et al., 2018) 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 (Malmi et al., 2022). 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 keyword extractor. Based on the above reasons, we do not conduct experiments on the X-Sum dataset.

The proposed model EditKSum is more suitable for scenarios where there is a high degree of overlap between the original text and the summary. Further discussions about this phenomenon will be provided in Section D of the supplementary materials.

4.2 Experimental Setup

Following previous edit-based and non-autoregressive methods (Gu et al., 2018, 2019; Xu and Carpuat, 2021), we apply sequence-level knowledge distillation (Kim and Rush, 2016) 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 (Lin, 2004) as the automatic evaluation metric which re-

Model	CNNDM ROUGE-1	CNNDM ROUGE-2	CNNDM ROUGE-L	Gigaword ROUGE-1	Gigaword ROUGE-2	Gigaword ROUGE-L
NAUS (Liu et al., 2022a)†	-	-	-	28.55	9.97	25.78
Bert+CRF-NAT (Su et al., 2021)†	-	-	-	35.05	16.48	33.28
BANG (Qi et al., 2021)	35.77	12.87	33.07	-	-	-
LevT (Gu et al., 2019)†	36.35	15.56	33.72	36.14	17.14	34.34
EDITOR (Xu and Carpuat, 2021)	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 (Vaswani et al., 2017)†	39.50	16.06	36.63	37.57	18.90	34.69
BERTSum (Liu and Lapata, 2019)†	42.00	19.44	38.98	-	-	-
PEGASUS (Zhang et al., 2020)†	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**	<u>20.00</u>	<u>40.61</u>	**40.15**	<u>18.05</u>	<u>35.88</u>

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. The top scores are highlighted in **bold**, while the second-best scores are <u>underlined</u>. † signifies that the results for at least one dataset are sourced from the original papers or public leaderboards.

ports the precision/recall/F scores of the 1-gram/2-gram/longest common subsequences that are overlapped between the generated and the reference summary.

Additionally, we also utilized the GPT-4 API(OpenAI, 2023) for subjective evaluation of the fluency and coherence of the generated summaries.

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 (Kingma and Ba, 2015) 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 (Ott et al., 2019). The code and pre-trained models will be released upon acceptance.

Baselines To make a comprehensive comparison, we consider the following baselines.
NAUS (Liu et al., 2022a) is a specialized unsupervised NAR model designed for text summarization tasks; **BERT+CRF-NAT** (Su et al., 2021) employs BERT as the backbone of a NAR model and proposes an elegant decoding mechanism to help length prediction; **BANG** (Qi et al., 2021) 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** (Gu et al., 2019) is a classical edit-based generation model which applies deletion and insertion operations on empty sequences to generate the targets; **EDITOR** (Xu and Carpuat, 2021) 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** (Vaswani et al., 2017) serves as a widely used AR baseline; **BertSum** (Liu and Lapata, 2019) is an AR summarization model with the encoder initialized with BERT; **PEGASUS** (Zhang et al., 2020) 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.

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

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**

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**.

Metrics	Fluency		Consistency	
Datasets	CNNDM	Gigaword	CNNDM	Gigaword
Editor	6.33	6.27	6.91	7.09
EditKSum	**7.03**	**6.95**	**8.26**	**8.47**

Table 4: The fluency and consistency of EditKSum Evaluated by GPT-4. The highest numbers are in **bold**.

.

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.

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.

4.5 Fluency and Consistency

We used the GPT-4 API(OpenAI, 2023) to assess the fluency and consistency of the summaries generated by EditKSum. We chose 100 samples randomly from both CNNDM and Gigaword datasets respectively, then scored the fluency and consistency of the summaries generated by EditKSum and Editor.

As shown in table 4, on the one hand, due to the iterative refinement process of editing, EditKSum can generate results that are more fluent compared to the NAR baseline. On the other hand, the process of extracting keywords preserves most of the key information in the original text, so the consistency of EditKSum is also significantly better than that of EDITOR.

Specifically, on the CNNDM/Gigaword dataset, EditKSum has a score of 7.03/6.95 out of 10 in fluency and 8.26/8.47 out of 10 in consistency on average, outperforming the Editor's score of 6.33/6.27 and 6.91/7.09.

5 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. The proposed EditKSum is particularly well-suited for scenarios where there is a high degree of overlap between the original text and the summary. 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.

Limitations

As mentioned in the experimental section, our proposed model may yield less impressive results for datasets like XSum, where the source text does not explicitly contain prominent words that might appear in the target sequence. This is because the model needs to extract keywords from the source text and then edit them, which can be challenging in such cases. In our future work, we aim to investigate more robust keyword prediction and editing modules to effectively tackle such issues.

References

Sweta Agrawal and Marine Carpuat. 2022. 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), pages 7550–7563, Dublin, Ireland. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. CoRR.

Ankur Bapna and Orhan Firat. 2019. 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), pages 1538–1548, Hong Kong, China. Association for Computational Linguistics.

Kianté Brantley, Kyunghyun Cho, Hal Daumé Iii, and Sean Welleck. 2019. Non-monotonic sequential text generation. In Proceedings of the 2019 Workshop on Widening NLP, pages 57–59.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Zi-Yi Dou, Pengfei Liu, Hiroaki Hayashi, Zhengbao Jiang, and Graham Neubig. 2021. 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, pages 4830–4842, Online. Association for Computational Linguistics.

Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. 2018. Bottom-up abstractive summarization. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 4098–4109, Brussels, Belgium. Association for Computational Linguistics.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. 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), pages 6112–6121, Hong Kong, China. Association for Computational Linguistics.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In International Conference on Learning Representations.

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019. Levenshtein transformer. In Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc.

Junliang Guo, Linli Xu, and Enhong Chen. 2020a. 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, pages 376–385, Online. Association for Computational Linguistics.

Junliang Guo, Zhirui Zhang, Linli Xu, Hao-Ran Wei, Boxing Chen, and Enhong Chen. 2020b. Incorporating bert into parallel sequence decoding with adapters. In Advances in Neural Information Processing Systems, volume 33, pages 10843–10854. Curran Associates, Inc.

Yosuke Higuchi, Nanxin Chen, Yuya Fujita, Hirofumi Inaguma, Tatsuya Komatsu, Jaesong Lee, Jumon Nozaki, Tianzi Wang, and Shinji Watanabe. 2021. A comparative study on non-autoregressive modelings for speech-to-text generation. In 2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), pages 47–54.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In Proceedings of the 36th International Conference on Machine Learning, pages 2790–2799. PMLR.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 1173–1182, Brussels, Belgium. Association for Computational Linguistics.

V. Levenshtein. 1965. Binary codes capable of correcting deletions, insertions, and reversals. Soviet physics. Doklady.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 7871–7880, Online. Association for Computational Linguistics.

Haoran Li, Junnan Zhu, Jiajun Zhang, Chengqing Zong, and Xiaodong He. 2020. Keywords-guided abstractive sentence summarization. In Proceedings of the AAAI conference on artificial intelligence, volume 34, pages 8196–8203.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In Text Summarization Branches Out, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Puyuan Liu, Chenyang Huang, and Lili Mou. 2022a. Learning non-autoregressive models from search for unsupervised sentence summarization. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 7916–7929, Dublin, Ireland. Association for Computational Linguistics.

Yang Liu and Mirella Lapata. 2019. Text summarization with pretrained encoders. 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), pages 3730–3740, Hong Kong, China. Association for Computational Linguistics.

Yixin Liu and Pengfei Liu. 2021. 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), pages 1065–1072, Online. Association for Computational Linguistics.

Yixin Liu, Pengfei Liu, Dragomir Radev, and Graham Neubig. 2022b. Brio: Bringing order to abstractive summarization. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2890–2903, Dublin, Ireland. Association for Computational Linguistics.

Eric Malmi, Yue Dong, Jonathan Mallinson, Aleksandr Chuklin, Jakub Adamek, Daniil Mirylenka, Felix Stahlberg, Sebastian Krause, Shankar Kumar, and Aliaksei Severyn. 2022. 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, pages 1–7, Seattle, United States. Association for Computational Linguistics.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel, pages 807–814. Omnipress.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2016a. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. Proceedings of the AAAI Conference on Artificial Intelligence, 31.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gůlçehre, and Bing Xiang. 2016b. Abstractive text summarization using sequence-to-sequence rnns and beyond. In Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, pages 280–290, Berlin, Germany. Association for Computational Linguistics.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. 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, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. 2020. Gector – grammatical error correction: Tag, not rewrite. In Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications, pages 163–170, Seattle, WA, USA → Online. Association for Computational Linguistics.

OpenAI. 2023. Gpt-4 technical report.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. Fairseq: A fast, extensible toolkit for sequence modeling. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations), pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Weizhen Qi, Yeyun Gong, Jian Jiao, Yu Yan, Weizhu Chen, Dayiheng Liu, Kewen Tang, Houqiang Li, Jiusheng Chen, Ruofei Zhang, Ming Zhou, and Nan Duan. 2021. Bang: Bridging autoregressive and non-autoregressive generation with large scale pretraining. In Proceedings of the 38th International Conference on Machine Learning, pages 8630–8639. PMLR.

Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. 2020. Prophetnet: Predicting future n-gram for sequence-to-sequencepre-training. In Findings of the Association for Computational Linguistics: EMNLP 2020, pages 2401–2410, Online. Association for Computational Linguistics.

Lihua Qian, Hao Zhou, Yu Bao, Mingxuan Wang, Lin Qiu, Weinan Zhang, Yong Yu, and Lei Li. 2021. Glancing transformer for non-autoregressive neural machine translation. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 1993–2003, Online. Association for Computational Linguistics.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 379–389, Lisbon, Portugal. Association for Computational Linguistics.

Chitwan Saharia, William Chan, Saurabh Saxena, and Mohammad Norouzi. 2020. Non-autoregressive machine translation with latent alignments. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1098–1108, Online. Association for Computational Linguistics.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. Mass: Masked sequence to sequence pre-training for language generation. In Proceedings of the 36th International Conference on Machine Learning, pages 5926–5936. PMLR.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In Proceedings of the 36th International Conference on Machine Learning, pages 5976–5985. PMLR.

Yixuan Su, Deng Cai, Yan Wang, David Vandyke, Simon Baker, Piji Li, and Nigel Collier. 2021. Non-autoregressive text generation with pre-trained language models. In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, pages 234–243, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc.

Yong Wang and Xinwei Geng. 2022. Improving non-autoregressive neural machine translation via modeling localness. In Proceedings of the 29th International Conference on Computational Linguistics, pages 5217–5226, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Yisheng Xiao, Lijun Wu, Junliang Guo, Juntao Li, Min Zhang, Tao Qin, and Tie-Yan Liu. 2023. A survey on non-autoregressive generation for neural machine translation and beyond. IEEE transactions on pattern analysis and machine intelligence, PP.

Jiacheng Xu, Zhe Gan, Yu Cheng, and Jingjing Liu. 2020. Discourse-aware neural extractive text summarization. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 5021–5031, Online. Association for Computational Linguistics.

Weijia Xu and Marine Carpuat. 2021. Editor: An edit-based transformer with repositioning for neural machine translation with soft lexical constraints. Transactions of the Association for Computational Linguistics, 9:311–328.

Kexin Yang, Wenqiang Lei, Dayiheng Liu, Weizhen Qi, and Jiancheng Lv. 2021. Pos-constrained parallel decoding for non-autoregressive generation. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 5990–6000, Online. Association for Computational Linguistics.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. 2020. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In Proceedings of the 37th International Conference on Machine Learning, pages 11328–11339. PMLR.

Ming Zhong, Pengfei Liu, Yiran Chen, Danqing Wang, Xipeng Qiu, and Xuanjing Huang. 2020. Extractive summarization as text matching. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 6197–6208, Online. Association for Computational Linguistics.

Yicheng Zou, Zhihua Liu, Xingwu Hu, and Qi Zhang. 2021. Thinking clearly, talking fast: Concept-guided non-autoregressive generation for open-domain dialogue systems. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 2215–2226, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

A The Structure of Adapters

As mentioned in the main part of paper, we adopt four different kinds of adapters including encoder adaper, decoder adapter, keywords extractor adapter and adapters for three different editing operations. The structure of keywords extractor adapter has been specifically introduced in the main text. Therefore, we will introduce the other kinds of adapters in detail below.

A.1 Encoder Adapter

On the encoder side, we construct the adapter with a normalization layer and two feed-forward layers with a non-linearity ReLU between them:

$$Z = W_1 \cdot LN(H) \tag{11}$$

$$H_{\text{enc}} = H + W_2 \cdot (\sigma(Z)) \tag{12}$$

where W_1 and W_2 are the trained parameters of feed-forward layers. LN denotes layer normalization. H and H_{enc} are the input and output hidden states of adapter respectively. $\sigma(\cdot)$ indicates the the activation function ReLU.

A.2 Decoder Adapter

On the decoder side, the adapter need to process the information from source. So we regard a multi-head cross-attention module as the decoder adapter:

$$H_{dec} = \text{Attn}(H, H^E, H^E) \tag{13}$$

where $\text{Attn}(Q, K, V)$ represents the multi-head attention. H^E and H denotes hidden states of encoder and decoder respectively. H_{dec} is the output of decoder adapter.

A.3 Operation Adapter

Setting	R-1	R-1	R-L
Shared	44.19	20.00	40.61
Separate	43.80	19.69	40.29

Table 5: The results of different settings of operation adapter. **Shared** means placeholder policy and token policy share the same adapter. **Separate** means three policies use different adapter.

Following the decoder adapter are the reposition adapter, placeholder adapter and token adapter. They adopt the same structure as the encoder adapter.

It is worth to mention that the three policies can share the common adapter to achieve the purpose of joint training. Specifically, in this paper, the placeholder policy and token policy share the same adapter because they are both related to the insertion operation. As shown in Table 5, the shared setting achieves better results.

Hyper-parameter of Adapters Whether in encoder adapter or operation adapter, we can flexibly control the amount of parameters by adjusting the bottleneck dimension between the two feed-forward layers. The hidden dimension between two FFN layers are both set as d_{enc}=2048. The hidden dimension of the cross-attention module is set equal to the hidden dimension of BERT model i.e. d_{dec}=768.

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**

Table 6: 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**.

B 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 6, 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.

Datasets	CNNDM	Gigaword	XLSum	NewsRoom	XSum
Overlap Rate	91.8%	58.82%	59.85%	80.61%	47.28%
PEGASUS (Zhang et al., 2020)	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%

Table 7: The influence the abstractiveness of datasets. "Overlap Rate" represents the proportion of overlap between the original text and the summary in different datasets. The following two rows respectively show the performance of the PEGASUS and EditKSum models on these datasets, measured using the ROUGE-2 evaluation metric.

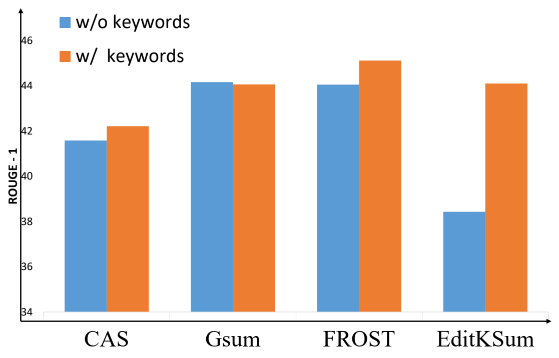

Figure 3: ROUGE-1 scores of keywords-guided summarization models. Blue/orange bars indicate the results w/ or w/o the help of keywords.

C 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.

As shown in Table 7, 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.

D 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, GSum and FROST. 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 Figure 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.

E 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 8.

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.

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

Table 8: 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.