RefSum: Refactoring Neural Summarization

Yixin Liu, Zi-Yi Dou, Pengfei Liu [*]
Carnegie Mellon University
`{yixinl2,zdou,pliu3}@cs.cmu.edu`

Abstract

Although some recent works show potential complementarity among different state-of-the-art systems, few works try to investigate this problem in text summarization. Researchers in other areas commonly refer to the techniques of *reranking* or *stacking* to approach this problem. In this work, we highlight several limitations of previous methods, which motivates us to present a new framework *Refactor* that provides a unified view of text summarization and summaries combination. Experimentally, we perform a comprehensive evaluation that involves twenty-two base systems, four datasets, and three different application scenarios. Besides new state-of-the-art results on CNN/DailyMail dataset (46.18 ROUGE-1), we also elaborate on how our proposed method addresses the limitations of the traditional methods and the effectiveness of the *Refactor* model sheds light on insight for performance improvement. Our system can be directly used by other researchers as an off-the-shelf tool to achieve further performance improvements. We open-source all the code and provide a convenient interface to use it: `https://github.com/yixinL7/Refactoring-Summarization`.

1 Introduction

In neural text summarization, system designers commonly have flexible choices in model architectures (Rush et al., 2015; Kedzie et al., 2018), decoding strategies (Paulus et al., 2018) (e.g. beam search) and etc. As a result, even on the same dataset, different selection biases of these choices will lead to diverse system outputs (Kedzie et al., 2018; Hossain et al., 2020).

To combine complementarity of system's output under different setups, researchers have made some preliminary efforts on two-stage learning (Collins and Koo, 2005; Huang, 2008; González-Rubio et al., 2011; Mizumoto and Matsumoto, 2016), consisting of (i) a *base-stage*: first generates different outputs under different setups, and (ii) a *meta-stage*: then aggregates them in diverse ways, exemplified by *stacking* that uses a high-level model to combine *multiple* low-level models (Ting and Witten, 1997), or *reranking* (Collins and Koo, 2005), which aims to rerank different outputs of *one* system. Although these methods each play a role in different scenarios, they suffer from following potential limitations:

(i) *Ad-hoc Methods*: most existing methods are designed for a specific scenario. For example, Li et al. (2015) and Narayan et al. (2018b) resort to reranking techniques to select summary-worthy sentences that are usually generated from one system. By contrast, Hong et al. (2015) focus on summaries generated from different systems and use a non-neural system combination method to make their complementary advantages. Few works explore if the complementarity existing in different scenarios could be utilized in a unified framework.

(ii) *Base-Meta Learning Gap*: parameterized models between two learning stages are relatively independent. For example, Zhou et al. (2017) and Huang et al. (2020) adapt the seq2seq (Sutskever et al., 2014) framework as the meta model for combination, which takes the outputs of multiple base

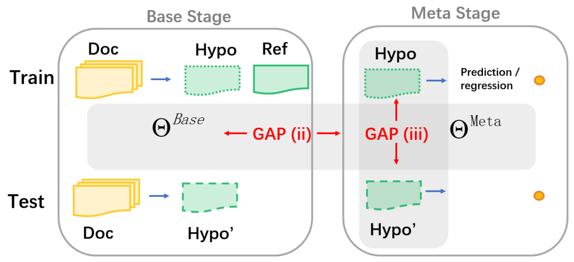

Figure 1: Illustration of two-stage learning. "`Doc, Hypo, Ref`" represent "`input document, generated hypothesis, gold reference`" respectively. "`Hypo'`" represents texts generated during test phase. Θ^{Base} and Θ^{Meta} represent learnable parameters in two stages.

[*]Corresponding author.

systems as a part of the inputs for machine translation. As a result, there is no parameter sharing between the meta model and base systems as shown in Fig. 1, which prevents the meta model from fully utilizing the knowledge encoded in the base systems.

(iii) *Train-Test Distribution Gap*: regarding the meta-learning stage, there is a distribution gap between the training and test distributions. Fig. 1 elucidates this phenomenon: the training distribution of `Hypo` differs from the test distribution of `Hypo'`. Although both two are outputs from the base stage, `Hypo` would be more accurate (closer to gold summaries) since it is the output during the training phase.

In this work, we aim to address these limitations by proposing a general framework, named *Refactor*, which can not only serve as a base system to construct a summary by selecting sentences from the source document but also act as a meta system to select the best system output from multiple candidates. The unification of base and meta systems allows them to share a set of parameters, thereby alleviating the "Base-Meta learning gap". Besides, we propose a *pretrain-then-finetune* paradigm for *Refactor* that mitigates the "Train-Test distribution gap". In practice, our proposed *Refactor* can be applied to different scenarios. For example, as a meta system, it can be used for multiple system combination or single system re-ranking.

Our contributions can be briefly summarized as:

(1) We dissect two major factors that influence the performance of two-stage learning when leveraging the complementarity among different systems: (i) Base-Meta Learning Gap (ii) Train-Test Distribution Gap;

(2) We show these two types of gaps can be alleviated by promoting communication between the two stages in §4 , and therefore present a new paradigm where the base and meta learners are parameterized with shared parameters;

(3) We have made comprehensive experiments (twenty-two top-scoring systems, four datasets). In addition to achieving state-of-the-art results on CNN/DailyMail dataset (§5) by a significant margin, the efficacy of the proposed *Refactor* opens up a thought-provoking direction for performance improvement: instead of pursuing a purely end-to-end system, a promising exploration is to incorporate different types of inductive biases stage-wisely with the same parameterized function. Our experimental results demonstrate that there exists complementarity introduced by decoding algorithms (e.g. beam search) §5.5 or system combination §5.6 among the current state-of-the-art summarization systems, which can be effectively utilized by our model for boosting the system performance.

2 Preliminaries

Existing works commonly design systems in an end-to-end fashion (Sutskever et al., 2014; Sukhbaatar et al., 2015), which, though effective, also proves to be insufficient in some scenarios (Glasmachers, 2017; Webb et al., 2019). Instead of optimizing a system in an end-to-end fashion, one more flexible paradigm, stage-wise learning, is to break down the holistic process into different stages. The basic idea is to incorporate different types of inductive biases stage-wisely and two typical examples are: *Stacking* and *Reranking*.

Stacking Stacking (a.k.a, Stacked Generalization) is a general method of using a high-level model to combine lower-level models to achieve greater predictive accuracy (Ting and Witten, 1997). In NLP research, this method has been widely explored in machine translation (MT) task. Traditionally, it is used to improve the performance of statistical MT systems (González-Rubio et al., 2011; Watanabe and Sumita, 2011; Duh et al., 2011; Mizumoto and Matsumoto, 2016). Some recent work (Zhou et al., 2017; Huang et al., 2020) also extends this method to neural MT where the meta model and base systems are all neural models. There is a handful of works about system combination for summarization (Hong et al., 2015), in which a feature-based meta model is used for combining unsupervised text summarization systems.

Reranking Reranking is a technique to improve performance by reranking the output of an existing system, which has been widely used across different NLP tasks, such as constituency parsing (Collins and Koo, 2005; Huang, 2008), dependency parsing (Zhou et al., 2016; Do and Rehbein, 2020), semantic parsing (Ge and Mooney, 2006; Yin and Neubig, 2019), machine translation (Shen et al., 2004; Mizumoto and Matsumoto, 2016).

Comparing *reranking* and *stacking*, both of them involve two-stage learning and the first stage would provide multiple candidate outputs as the input for the second stage. However, they differ in the way how multiple candidate outputs are generated at the first stage. Specifically, reranking usually decodes k-most qualified results during inference, using one base system. By contrast, stacking generates multiple outputs that are usually from different base systems.

3 Summarization as Two-stage Learning

In what follows, we detail how to formulate summarization as a two-stage learning task.

Base system The system in the base stage aims to generate a summary based on the input text. Specifically, given a document $D = \{s_1, \cdots, s_n\}$ with n sentences, we refer to C as a *candidate summary* of D generated by a summarization system, which can be parameterized in diverse forms:

$$C = \text{BASE}(D, \mathcal{T}, \mathcal{S}, \Theta^{\text{base}}) \tag{1}$$

where $\text{BASE}(, \Theta^{\text{base}})$ represents a base system that can be instantiated either as an extractive model or abstractive model with a specific experimental setup: training method $\mathcal{T}$, decoding strategy $\mathcal{S}$.

Meta system In practice, different choices of parameterized function $\text{BASE}(\cdot)$, training method $\mathcal{T}$ and decoding strategy $\mathcal{S}$ commonly lead to different candidate summaries, $\mathcal{C} = \{C_1, \cdots, C_k\}$, where $\mathcal{C}$ represents a set of different candidate summaries. The goal of the meta system is to utilize complementarities among $\mathcal{C}$ by popular techniques, such as reranking and system combination.

Specifically, given a set of candidate summaries $\mathcal{C}$, a meta system is used to re-construct a new candidate summary C^*

$$C^* = \text{META}(D, \mathcal{C}, \Theta^{\text{meta}}) \tag{2}$$

where Θ^{meta} represents learnable parameters of the meta system.

4 Refactoring Text Summarization

Despite effectiveness of existing meta systems, they, as briefly mentioned in §1, suffer from two major problems: (i) *Base-Meta Learning Gap* and (ii) *Train-Test Distribution Gap*.

4.1 Refactoring

In this paper, we propose the model ***Refactor*** that unifies the goal of the base and meta systems by the view that a summary can be generated by selecting the best combination of document sentences. Therefore, both base and meta systems aim to select an optimal candidate summary, and they only differ in how the candidate summary set is constructed. For example, *Refactor* can be a base system when the candidate summary set $\mathcal{C}$ is formed by directly enumerating different combinations of *document sentences* and would be a meta system when $\mathcal{C}$ represents summaries from different systems. This formulation is advantageous in two points:

(1) No matter where a system selects (from document sentences or multiple system outputs), the chosen criteria that define a good summary are shared. Therefore, the learning process of base and meta systems can be parameterized using a set of parameters, maximizing the information-sharing across two stages and mitigating the *Base-Meta Learning Gap*.

$$C^* = \text{REFACTOR}(D, \mathcal{C}, \Theta^{\text{refactor}}), \tag{3}$$

where $\text{REFACTOR}(\cdot, \Theta^{\text{refactor}})$ is the *Refactor* model, and the candidate summaries $\mathcal{C}$ can be constructed in different ways.

(2) Additionally, learning to select candidate summaries from document sentences enables the system to see more diverse candidates with different distributions. This is effective for solving the *Train-Test Distribution Gap*, where the distribution of the meta system outputs in training samples deviates from the test one.

Specifically, our proposed *Refactor* first learns to select candidate summaries from document sentences (pre-trained *Refactor*) and then learns to select candidate summaries from different system outputs (fine-tuned *Refactor*).

4.2 Pre-trained Refactor

Pre-trained *Refactor* takes as input a document $D = \{s_1, \cdots, s_n\}$ as well as a set of candidate summaries $\mathcal{C} = \{C_1, \cdots, C_m\}$, which can be constructed by enumerating possible combinations of source sentences with heuristic pruning. For example, an extractive system could be used to prune unlikely sentences to control the number of candidates. $\text{REFACTOR}(\cdot, \Theta^{\text{refactor}})$ is instantiated as a score function which quantifies the degree to which a candidate summary C_i is matched with the source document D.

$$\begin{aligned} C^* &= \text{REFACTOR}(D, \mathcal{C}, \Theta^{\text{refactor}}) \\ &= \underset{C_i \in \mathcal{C}}{\text{argmax}}(\text{SCORE}(\mathbf{D}, \mathbf{C}_i)) \end{aligned} \tag{4}$$

where $\mathbf{D}$ and $\mathbf{C}_i$ denote document and summary representations respectively, which are calculated by a BERT (Devlin et al., 2019) model. $\textsc{Score}(\cdot)$ is a function that measures the similarity between a document and candidate summary.

Contextualized Similarity Function To instantiate $\textsc{Score}(\cdot)$, we follow the forms as mentioned in Zhang et al. (2019b); Zhao et al. (2019); Gao et al. (2020), which have shown superior performance on measuring semantic similarity between documents and summaries.

Specifically, $\textsc{Score}(\cdot)$ is defined based on the greedy matching algorithm, which matches every word in one text sequence to the most similar word in another text sequence and vise versa. Given the document embedding matrix $\mathbf{D} = \langle \mathbf{d}_1, \cdots, \mathbf{d}_k \rangle$ and the candidate embedding matrix $\mathbf{C} = \langle \mathbf{c}_1, \cdots, \mathbf{c}_l \rangle$ encoded by BERT, $\textsc{Score}(\cdot)$ can be calculated as:

$$\textsc{Score}(\mathbf{D}, \mathbf{C}) = 2\frac{\mathrm{R}(\mathbf{D}, \mathbf{C}) \cdot \mathrm{P}(\mathbf{D}, \mathbf{C})}{\mathrm{R}(\mathbf{D}, \mathbf{C}) + \mathrm{P}(\mathbf{D}, \mathbf{C})} \quad (5)$$

where the weighted recall R, precision P are defined as follows:[1]

$$\mathrm{R}(\mathbf{D}, \mathbf{C}) = \frac{\sum_i w_i \max_j \cos(\mathbf{d}_i, \mathbf{c}_j)}{\sum_i w_i} + 1, \quad (6)$$

$$\mathrm{P}(\mathbf{D}, \mathbf{C}) = \frac{\sum_j \max_i \cos(\mathbf{d}_i, \mathbf{c}_j)}{l} + 1, \quad (7)$$

w_i is the weight of the i-th token in the document. We use weighted recall R based on the assumption that for text summarization, tokens in the source document have different importance and the summary should capture the most important information of the source document. Therefore, we introduce a weighting module built by a two-layer Transformer (Vaswani et al., 2017) assigning weights w_i:

$$w_i = \frac{\exp\left(\mathrm{dot}(\mathbf{d}_i, \hat{\mathbf{d}}_0)/\sqrt{d}\right)}{\sum_j \exp(\mathrm{dot}(\mathbf{d}_j, \hat{\mathbf{d}}_0)/\sqrt{d})}, \quad (8)$$

where $\hat{\mathbf{D}} = \mathrm{Transformer}(\mathbf{D})$ and $\hat{\mathbf{d}}_0 = \hat{\mathbf{D}}[0]$ represents the embedding of the "`[CLS]`" token which encodes the global information. d is the dimension of $\mathbf{d}_i$.

Learning Objective We use a ranking loss to learn the parameter Θ^{refactor}, inspired by the assumption (Zhong et al., 2020) that a good candidate summary should be as close with the source document as possible. Formally,

$$L = \sum_i \sum_{j>i} \max(0, \textsc{score}(\mathbf{D}, \mathbf{C}_j) - \textsc{score}(\mathbf{D}, \mathbf{C}_i) + (j-i) * \lambda_c) \quad (9)$$

where C_i and C_j denote the i-th and j-th sample of the candidate list which is descendingly sorted by the ROUGE (Lin, 2004) scores between the reference summary $\hat{C}$ and candidates. That is, $\mathrm{ROUGE}(C_i, \hat{C}) > \mathrm{ROUGE}(C_j, \hat{C})$ for $i < j$. λ_c is the corresponding margin set to 0.01.

[1]We found that adding 1 to the precision and recall helps to stabilize the training.

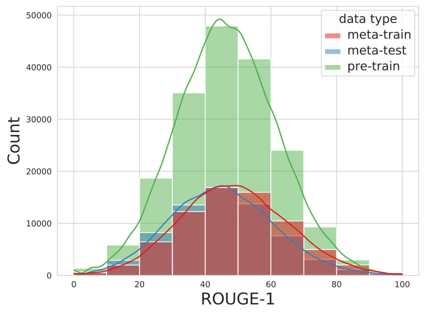

Figure 2: ROUGE-1 distributions of the candidates in pre-training stage training set (*pre-train*), fine-tuning stage training set (*meta-train*) and fine-tuning stage test set (*meta-test*) on `XSum` dataset.

4.3 Fine-tuned Refactor

In order to fit the distributions of the specific types of input, we then fine-tune *Refactor* using the outputs generated by the base systems. Specifically, fine-tuning is also based on Eq. 9 where the candidate summaries C are generated by the base systems under different application scenarios.

Why does *Pre-train and Fine-tune* matter? We elaborate on the proposed two-step training using a real case. Fig. 2 depicts the distribution of ROUGE-1 scores regarding the candidate summaries in the pre-training stage training set, fine-tuning stage training set and test set on the `XSum` dataset, where we sample the same number of $\{\text{document, candidate summaries}\}$ pairs. We can observe that:

(i) there is a distribution gap between train and test samples in fine-tuning stage. (ii) in *pre-training* stage the pre-trained *Refactor* has seen a large number of candidate summaries with diverse performance (ROUGE value), which improves its generalization ability. In §5 we will show that the Pre-train and Fine-tune paradigm outperforms one-step training where the model is directly trained with data generated from the base systems.

4.4 Application Scenarios

Our *Refactor* can be used as different roles in different scenarios as follows.

4.4.1 Refactor as Base Learner

The pre-trained *Refactor* can not only be fine-tuned for a better selection of candidate summaries, but also be regarded as a base system, providing one system output. This feature of *Refactor* maximizes parameter sharing across the two training stages.

4.4.2 Refactor as Meta Learner

Both pre-trained *Refactor* and fine-tuned *Refactor* can be used as a meta system to select the best candidate when we have multiple system summaries. In this work, we explore the following settings:

(1) **Single System**: It considers re-ranking candidate summaries generated from a single abstractive system using beam search.

(2) **Multi-system Summary-level**: It is tasked to select the best candidate summary from the results of different systems.

(3) **Multi-system Sentence-level**: We also take a step towards the fine-grained fusion of summaries from extractive and abstractive systems. Specifically, here candidate summaries are generated by combining the results of different systems at the sentence level.

5 Experiments

5.1 Datasets

We mainly experiment on four datasets, whose statistics are shown in Tab. 1.

Datasets	# Num			Avg. Len	
	Train	Valid	Test	Doc.	Sum.
CNNDM	287K	13K	11K	768.6	55.7
XSum	203K	11K	11K	429.2	23.3
PubMed	83K	4.6K	5K	468.7	210.3
WikiHow	168K	6K	6K	579.1	62.2

Table 1: Datasets Statistics. Len is the length of tokens.

CNNDM[2] (Hermann et al., 2015) is a widely used dataset containing news articles and the associated highlights which are used as the reference summaries. We follow the work of Nallapati et al. (2016) for data preprocessing.

XSum[3] (Narayan et al., 2018a) contains online articles collected from BBC with highly abstractive one-sentence summaries.

PubMed[4] (Cohan et al., 2018) contains scientific papers collected from PubMed.com.

WikiHow[5] (Koupaee and Wang, 2018) is a large-scale dataset constructed from the articles using online WikiHow knowledge base.

5.2 Base Systems

Below, we mainly use BART, GSum and PEGASUS as the base systems since they have achieved state-of-the-art performance on at least one dataset.

BART (Lewis et al., 2020) is a large pre-trained sequence-to-sequence model that achieves strong performance on the abstractive summarization.

GSum (Dou et al., 2020) enhances the performance of BART using additional guidance information, which achieves the current state-of-the-art performance on the `CNNDM` dataset.

PEGASUS (Zhang et al., 2020) achieves competitive performance on various summarization datasets and is the current state-of-the-art on the `XSum` dataset.

To make a comprehensive evaluation of our proposed model, we additionally collect 19 top-scoring systems as base systems on `CNNDM`.[6] In details, for §5.7 we use the following systems: pointer-generator+coverage (See et al., 2017), REFRESH (Narayan et al., 2018b), fastAbsRL-rank (Chen and Bansal, 2018), CNN-LSTM-BiClassifier (Kedzie et al., 2018), CNN-Transformer-BiClassifier (Zhong et al., 2019), CNN-Transformer-Pointer (Zhong et al., 2019), BERT-Transformer-Pointer (Zhong et al., 2019), Bottom-Up (Gehrmann et al., 2018), NeuSum (Zhou et al., 2018), BanditSum (Dong et al., 2018), twoStageRL (Zhang et al., 2019a), preSummAbs (Liu and Lapata, 2019), preSummAbs-ext (Liu and Lapata, 2019), HeterGraph (Wang et al., 2020), MatchSum (Zhong et al., 2020), Unilm-v1 (Dong et al., 2019), Unilm-v2 (Dong et al., 2019), T5 (Raffel et al., 2020).

[2]https://cs.nyu.edu/~kcho/DMQA/

[3]https://github.com/EdinburghNLP/XSum

[4]https://github.com/acohan/long-summarization

[5]https://github.com/mahnazkoupaee/WikiHow-Dataset

[6]Since `CNNDM` is the most popular dataset, we can collect more existing systems on it.

5.3 Baseline Systems

Neural system combinator: We use BERTScore (Zhang et al., 2019b) as an unsupervised baseline with neural models, which is an automatic evaluation metric computing the similarity of text pairs based on the corresponding BERT-encoded representations. We use it to directly compute the similarity score between the source documents and candidate summaries.

Non-Neural system combinator: We use RankSVM[7] (Joachims, 2002) as a non-neural baseline. We perform cross-validation on the development set for hyper-parameter searching and train the model on the development set. The set of features is listed in Appendix A.

Oracles: We compare our model with sample-wise **Min**, **Max** and **Random** oracles using ROUGE.

5.4 Training Details

For the following experiments in §5.5, §5.6 and §5.7 on CNNDM, we pre-train the *Refactor* model with a candidate set generated by enumerating combinations of sentences in the source documents. To reduce the number of candidates, we prune the sentences assigned with lower scores by an extractive model, BERTSum (Liu and Lapata, 2019), following Zhong et al. (2020). The maximum number of candidates for one data sample is 20. The ***pre-trained Refactor*** is also used a base system in §5.6, whose outputs are used together with other base systems as candidate summaries. For different experiments, we fine-tune *pre-trained Refactor* on the base system's output, and name the model as ***fine-tuned Refactor***. To analyze the effectiveness of the proposed two-stage training, we additionally train the model without the pre-training step, which is named as ***supervised Refactor***.

The pre-trained BERT model we used is from *Transformers* library (Wolf et al., 2020).[8] We use Adam optimizer (Kingma and Ba, 2015) with learning rate scheduling.

$$lr = 0.002 \cdot \min(\text{step\_num}^{-0.5}, \quad \text{step\_num} \cdot \text{warmup\_steps}^{-1.5}), \tag{10}$$

where the warmup_steps is 10000. The model performance on the validation set is used to select the checkpoint. Pre-training takes around 40 hours on 4 GTX-1080-Ti GPUs while fine-tuning takes around 20 hours.

[7]http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html

[8]We use the 'bert-base-uncased' version with 110M parameters.

System	Method	R-1	R-2	R-L
BART	Base	44.26	21.12	41.16
	Min	41.58	19.27	38.69
	Max	47.22	23.28	43.90
	Random	44.40	21.26	41.28
	BERTScore	44.50	21.28	41.37
	RankSVM	44.50	21.39	41.43
	Supervised†	45.05	21.64	41.92
	Pre-trained†	44.78	21.49	41.68
	Fine-tuned†	**45.15**	**21.70**	**42.00**
GSum	Base	45.93	22.30	42.68
	Min	44.37	21.25	41.29
	Max	47.37	23.21	43.99
	Random	45.84	22.22	42.61
	BERTScore	45.84	22.25	42.64
	RankSVM	46.04	22.29	42.78
	Supervised †	46.11	22.32	42.85
	Pre-trained	45.88	22.23	42.67
	Fine-tuned†	**46.18**	**22.36**	**42.91**

Table 2: Single system reranking on CNNDM. **Base** denotes the base system. **Supervised** denotes the *Refactor* directly trained on the base systems' outputs. **Pre-trained** denotes the pre-trained *Refactor*. **Fine-tuned** denotes the fine-tuned model. R-1, R-2 and R-L denote ROUGE-1, ROUGE-2 and ROUGE-L. †: significantly better than the base system ($p < 0.01$).

5.5 Exp-I: Single System Reranking

We use BART and GSum for this experiment, and use beam search to generate the candidate summaries where the beam size is set to 4.

The results are listed in Tab. 2, which shows that (1) *Refactor* can boost the base system's performance by a significant margin, (2) the *fine-tuned Refactor* outperforms *supervised Refactor* directly trained on the base system's outputs, showing the effectiveness of the two-step training. Notably, we observe the fine-tuned *Refactor* can boost BART's performance from 44.26 to 45.15 on ROUGE-1, indicating that the top-1 output selected by beam search is not always the best one, and *Refactor* can effectively utilize the complementarity introduced by considering all the beam search results.

5.6 Exp-II: Multiple Systems Stacking

Summary-level For summary-level combination, we explore two-system combination (BART & pre-trained *Refactor*) and three-system combination (BART, GSum & pre-trained *Refactor*). The results are shown in Tab. 3.

Sentence-level For sentence-level combination, we use BART and pre-trained *Refactor* as the base

Setting	Method	R-1	R-2	R-L
Base	BART	44.26	21.12	41.16
	Refactor	44.13	20.51	40.29
	GSum	45.93	22.30	42.68
Two	Min	40.40	17.64	37.12
	Max	47.99	23.99	44.33
	Random	44.25	20.87	40.78
	BERTScore	43.95	20.45	40.23
	RankSVM	44.66	21.32	41.44
	Supervised †	44.75	21.40	41.47
	Pre-trained†	44.66	21.19	41.15
	Fine-tuned†	**45.04**	**21.61**	**41.72**
Three	Min	39.51	17.01	36.35
	Max	49.94	25.59	46.30
	Random	44.82	21.35	41.44
	BERTScore	44.10	20.64	40.42
	RankSVM	45.72	22.13	42.58
	Supervised	45.80	22.25	42.68
	Pre-trained	45.27	21.74	41.93
	Fine-tuned†	**46.12**	**22.46**	**42.92**

Table 3: Summary level combination on CNNDM. **Two** denotes two-system combination (BART and pre-trained *Refactor*). **Three** denotes three-system combination (BART, pre-trained *Refactor* and GSum). R-1, R-2 and R-L denote ROUGE-1, ROUGE-2 and ROUGE-L. †: significantly better than the best single system ($p < 0.01$).

System	R-1	R-2	R-L
BART	44.26	21.12	41.16
Refactor	44.13	20.51	40.29
Min	31.51	10.83	28.87
Max	50.91	26.07	46.97
Random	41.66	18.77	38.27
BERTScore	43.55	20.14	39.84
RankSVM	43.18	19.91	39.51
Supervised †	**44.96**	**21.50**	**41.43**
Pre-trained†	44.88	21.13	41.16
Fine-tuned†	44.93	21.48	41.42

Table 4: Sentence level combination on CNNDM. R-1, R-2 and R-L denote ROUGE-1, ROUGE-2 and ROUGE-L. †: significantly better than the best single system ($p < 0.01$).

systems. The sentences of each system's output are merged together to form the candidate sentence set, and all combinations of three sentences in the candidate set are generated as candidate summaries. To prune the candidates, we use tri-gram blocking to filter out candidates of which there exists an identical tri-gram in two sentences. The average number of candidates in the test set is 15.8. The results are shown in Tab. 4.

We have the following observations: (1) the pre-trained *Refactor* can already outperform the base systems, and (2) fine-tuning can further improve the performance. Meanwhile, we notice there are two exceptions: (i) For sentence-level combination, supervised *Refactor* has similar performance as fine-tuned *Refactor*. We hypothesis that this is because here the number of candidates in the fine-tuning data is relatively large, therefore directly training on the fine-tuning data is sufficient enough. (ii) The pre-trained *Refactor* cannot outperform GSum model in the three-system combination setting in Tab. 3. The reason might be that GSum has much stronger performance than the other two systems, which intuitively makes the expected gain from system combination lower than other settings.

bin	#sys	Max	Min	Rand	Best	Ours
39-40	3	45.28	34.30	39.88	39.98	**40.45**
41-42	8	50.14	32.65	41.44	41.89	**43.20**
42-43	3	47.37	36.79	42.10	42.27	**43.38**
43-44	2	47.60	39.63	43.58	43.97	**44.07**
44-45	3	50.29	38.66	44.58	44.68	**45.29**

Table 5: Multiple system combination. **bin** denotes the bin range. **#sys** denotes the number of systems. **Ours** denotes the pre-trained *Refactor* model. **Best** denotes the candidate system with best performance.

5.7 Exp-III: Generalization on 19 Top-performing Systems

To evaluate the *Refactor*'s generalization ability, we explore another setting where the pre-trained *Refactor* is directly used to select the outputs of multiple systems without fine-tuning.

To this end, we collect 19 top-performing summarization systems on CNNDM dataset. Here, we investigate if our *Refactor* can boost the performance of candidate systems with similar performance. In addition, we also aim to investigate how the range width of different systems' performance affects *Refactor*'s performance. Therefore, we group the candidate systems into equal-width bins based on their average ROUGE-1 scores, and evaluate our *Refactor* on each bin separately.

In Tab. 5 we report the average ROUGE-1 scores of the oracles, *Refactor*, and the best candidate system in each bin whose width is 1. *Refactor* consistently outperforms the best candidate system, showing its generalization ability.

Next, in Fig. 3 we plot the change of *Refactor*'s performance with different bin widths. We define the success rate of *Refactor* with a given bin width to be the number of bins where *Refactor* outperforms the single best base system normalized by the total number of bins. We observe that *Refactor* is more likely to improve the performance of base systems when the *system-level* performance of the

Method	XSum			PubMed			WikiHow		
	R-1	R-2	R-L	R-1	R-2	R-L	R-1	R-2	R-L
Base	47.12	24.46	39.04	43.42	15.32	39.21	41.98	18.09	40.53
Min	42.45	20.50	35.19	39.60	13.57	35.53	40.55	17.40	39.18
Max	51.51	28.04	42.70	45.23	16.72	40.67	43.00	18.44	41.44
Random	46.98	24.08	38.88	42.39	15.12	38.08	41.77	17.92	40.33
BERTScore	47.13	24.04	38.89	43.64	15.40	39.41	41.77	17.93	40.29
RankSVM	46.85	24.31	39.09	43.63	15.34	39.46	42.00	18.08	40.57
Pre-trained	**47.45**	**24.55**	**39.41**	43.58	15.36	39.38	41.97	18.03	40.52
Fine-tuned	47.32	24.31	39.22	**43.72**	**15.41**	**39.51**	**42.12**	**18.13**	**40.66**

Table 6: Single system reranking on other datasets. **Pre-trained** denotes the pre-trained *Refactor* model. **Fine-tuned** denotes the fine-tuned model. R-1, R-2 and R-L denote ROUGE-1, ROUGE-2 and ROUGE-L separately.

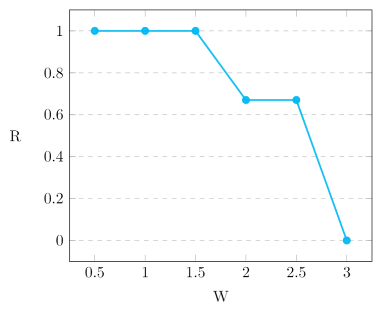

Figure 3: The *Refactor*'s success rates with different bin widths. **W** denotes the bin widths measured by ROUGE-1. **R** denotes the success rate of the *Refactor* outperforming the single best base system.

base systems is similar. Intuitively, if one base system is significantly better than the other systems, it is more difficult for *Refactor* to use other systems to complement the best base system.

5.8 Exp-IV: Effectiveness on More Popular Datasets

Next, we move on to other text summarization datasets to evaluate our proposed method's strength beyond CNNDM dataset. Some of the datasets used here are not as well-studied as CNNDM dataset, so there are less top-performing systems on these datasets. Therefore, here we focus on the experiments of the single system setting.

Setup Regarding the pre-trained *Refactor*, we use an extractive oracle to select document sentences and use the combinations of these sentences as candidates. In addition, since on Xsum the abstractive systems outperform extractive systems by a large margin, we use a pre-trained BART model with Diverse Beam Search (Vijayakumar et al., 2018) to generate 16 candidates per sample for pre-training. Regarding system re-ranking, we use BART as the base system to generate the candidate summaries except on Xsum dataset, where we use PEGASUS since it achieves better performance. Similar to §5.5, we use the outputs of beam search as the candidates. We select the first 4 outputs as the candidates.

The results in Tab. 6 show that *Refactor* is able to bring stable improvement over the base systems. The average summary length of these datasets varies from 23.3 (XSum) to 210.3 (Pubmed). Therefore, the results here demonstrate the *Refactor* can be applied to datasets with different characteristics. On XSum dataset, the pre-trained *Refactor* outperforms the fine-tuned *Refactor*. This may result from the additional pre-training data we introduced using BART, which is effective enough to train the *Refactor* for reranking PEGASUS output.

5.9 Fine-grained Analysis

We perform a fine-grained evaluation of *Refactor* to understand where improvement mainly comes.

Setup We choose the summary-level system combination setting on CNNDM test set in §5.6 as a case study, where the base systems are: BART and pre-trained *Refactor*, and then we use a fine-tuned *Refactor*[9] to combine them. Specifically, we first (i) define $\delta(C_{\text{BART}}, C_{\text{Pretrain}})$ as the performance (i.e., ROUGE) gap on the candidate summary C. (ii) then partition test samples into different buckets $S_1, \cdots, S_n$ according to the performance gap δ. (iii) calculate *selection accuracy* for each bucket, which represents how accurately the *Refactor* can

[9]As introduced in §4.4, Refactor could be used as either a base system or a system combinator.

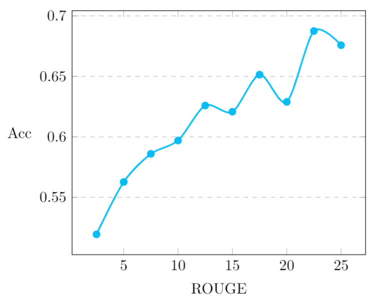

Figure 4: Fine-tuned *Refactor*'s selection accuracy on CNNDM with different difficulties. The X-axis is the difference of ROUGE score of BART and pre-trained *Refactor* outputs.

identify the best one from two candidate summaries.

The results are shown in Fig. 4. We observe that the selection accuracy is increasing as the gap δ becoming larger, indicating that *Refactor* performs better on the candidate summaries with diverse performance. Combining the results we get in §5.7, we conclude that *Refactor* has the largest potential gain when the base systems effectively complement each other – They have similar *system-level* performance but diverse *summary-level* performance. For example, each base system may perform significantly better than others on a subset of data with different characteristics but could not outperform others across the whole dataset.

6 Implications and Future Directions

We present a general framework for utilizing the complementarity of modern text summarization systems by formulating text summarization as a two-stage learning problem. Our proposed model, *Refactor*, can be used either as a base system or a meta system, effectively mitigating the learning gaps introduced in the two-stage learning. Experimental results show that *Refactor* is able to boost the performance of the base systems, and achieves the state-of-the-art performance on CNNDM and XSum datasets. We believe this work opens up a new direction for improving the performance of text summarization systems apart from an iterative process of searching for better model architectures – The gain of performance could be made by fully investigating and utilizing the complementarity of different systems with various architectures, problem formulations, decoding strategies, etc.

Acknowledgements

We thank Professor Graham Neubig and anonymous reviewers for valuable feedback and helpful suggestions. This work was supported in part by a grant under the Northrop Grumman SOTERIA project and the Air Force Research Laboratory under agreement number FA8750-19-2-0200. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

References

Yen-Chun Chen and Mohit Bansal. 2018. Fast abstractive summarization with reinforce-selected sentence rewriting. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 675–686, Melbourne, Australia. Association for Computational Linguistics.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, New Orleans, Louisiana. Association for Computational Linguistics.

Michael Collins and Terry Koo. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*.

Bich-Ngoc Do and Ines Rehbein. 2020. Neural reranking for dependency parsing: An evaluation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4123–4133, Online. Association for Computational Linguistics.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. In *Advances in Neural Information Processing Systems*, pages 13063–13075.

Yue Dong, Yikang Shen, Eric Crawford, Herke van Hoof, and Jackie Chi Kit Cheung. 2018. BanditSum: Extractive summarization as a contextual bandit. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3739–3748, Brussels, Belgium. Association for Computational Linguistics.

Zi-Yi Dou, Pengfei Liu, Hiroaki Hayashi, Zhengbao Jiang, and Graham Neubig. 2020. Gsum: A general framework for guided neural abstractive summarization. *arXiv preprint arXiv:2010.08014*.

Kevin Duh, Katsuhito Sudoh, Xianchao Wu, Hajime Tsukada, and Masaaki Nagata. 2011. Generalized minimum bayes risk system combination. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 1356–1360.

Yang Gao, Wei Zhao, and Steffen Eger. 2020. SUPERT: Towards new frontiers in unsupervised evaluation metrics for multi-document summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1347–1354, Online. Association for Computational Linguistics.

Ruifang Ge and Raymond J. Mooney. 2006. Discriminative reranking for semantic parsing. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 263–270, Sydney, Australia. 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.

Tobias Glasmachers. 2017. Limits of end-to-end learning. *arXiv preprint arXiv:1704.08305*.

Jesús González-Rubio, Alfons Juan, and Francisco Casacuberta. 2011. Minimum Bayes-risk system combination. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1268–1277, Portland, Oregon, USA. Association for Computational Linguistics.

Max Grusky, Mor Naaman, and Yoav Artzi. 2018. Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 708–719, New Orleans, Louisiana. Association for Computational Linguistics.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in neural information processing systems*, pages 1693–1701.

Kai Hong, Mitchell Marcus, and Ani Nenkova. 2015. System combination for multi-document summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 107–117, Lisbon, Portugal. Association for Computational Linguistics.

Nabil Hossain, Marjan Ghazvininejad, and Luke Zettlemoyer. 2020. Simple and effective retrieve-edit-rerank text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2532–2538, Online. Association for Computational Linguistics.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL-08: HLT*, pages 586–594, Columbus, Ohio. Association for Computational Linguistics.

Xuancheng Huang, Jiacheng Zhang, Zhixing Tan, Derek F. Wong, Huanbo Luan, Jingfang Xu, Maosong Sun, and Yang Liu. 2020. Modeling voting for system combination in machine translation. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 3694–3701. ijcai.org.

Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142.

Chris Kedzie, Kathleen McKeown, and Hal Daumé III. 2018. Content selection in deep learning models of summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1818–1828, Brussels, Belgium. 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*.

Mahnaz Koupaee and William Yang Wang. 2018. Wikihow: A large scale text summarization dataset. *CoRR*, abs/1810.09305.

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.

Chen Li, Yang Liu, and Lin Zhao. 2015. Improving update summarization via supervised ILP and sentence reranking. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for*

Computational Linguistics: Human Language Technologies, pages 1317–1322, Denver, Colorado. Association for Computational Linguistics.

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.

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.

Tomoya Mizumoto and Yuji Matsumoto. 2016. Discriminative reranking for grammatical error correction with statistical machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1133–1138, San Diego, California. Association for Computational Linguistics.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. 2016. 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. 2018a. 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.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018b. Ranking sentences for extractive summarization with reinforcement learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1747–1759, New Orleans, Louisiana. Association for Computational Linguistics.

Romain Paulus, Caiming Xiong, and Richard Socher. 2018. A deep reinforced model for abstractive summarization. In *International Conference on Learning Representations*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

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.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Libin Shen, Anoop Sarkar, and Franz Josef Och. 2004. Discriminative reranking for machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 177–184.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

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

Kai Ming Ting and Ian H. Witten. 1997. Stacked generalizations: When does it work? In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pages 866–873. Morgan Kaufmann.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Ashwin K Vijayakumar, Michael Cogswell, Ramprasaath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. 2018. Diverse beam search for improved description of complex scenes. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Danqing Wang, Pengfei Liu, Yining Zheng, Xipeng Qiu, and Xuanjing Huang. 2020. Heterogeneous graph neural networks for extractive document summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6209–6219, Online. Association for Computational Linguistics.

Taro Watanabe and Eiichiro Sumita. 2011. Machine translation system combination by confusion forest. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1249–1257, Portland, Oregon, USA. Association for Computational Linguistics.

Andrew M Webb, Charles Reynolds, Wenlin Chen, Henry Reeve, Dan-Andrei Iliescu, Mikel Lujan, and Gavin Brown. 2019. To ensemble or not ensemble: When does end-to-end training fail. In *Computer Vision and Pattern Recognition (CVPR)*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Pengcheng Yin and Graham Neubig. 2019. Reranking for neural semantic parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4553–4559, Florence, Italy. Association for Computational Linguistics.

Haoyu Zhang, Jingjing Cai, Jianjun Xu, and Ji Wang. 2019a. Pretraining-based natural language generation for text summarization. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 789–797, Hong Kong, China. 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*, volume 119 of *Proceedings of Machine Learning Research*, pages 11328–11339. PMLR.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019b. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*.

Wei Zhao, Maxime Peyrard, Fei Liu, Yang Gao, Christian M. Meyer, and Steffen Eger. 2019. MoverScore: Text generation evaluating with contextualized embeddings and earth mover distance. 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 563–578, Hong Kong, China. Association for Computational Linguistics.

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.

Ming Zhong, Pengfei Liu, Danqing Wang, Xipeng Qiu, and Xuanjing Huang. 2019. Searching for effective neural extractive summarization: What works and what's next. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1049–1058, Florence, Italy. Association for Computational Linguistics.

Deyu Zhou, Linsen Guo, and Yulan He. 2018. Neural storyline extraction model for storyline generation from news articles. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1727–1736, New Orleans, Louisiana. Association for Computational Linguistics.

Hao Zhou, Yue Zhang, Shujian Huang, Junsheng Zhou, Xin-Yu Dai, and Jiajun Chen. 2016. A search-based dynamic reranking model for dependency parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1393–1402, Berlin, Germany. Association for Computational Linguistics.

Long Zhou, Wenpeng Hu, Jiajun Zhang, and Chengqing Zong. 2017. Neural system combination for machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 378–384, Vancouver, Canada. Association for Computational Linguistics.

A Features for RankSVM

We use 18 features as defined below for RankSVM:

1. document length.
2. candidate summary length.
3. rouge-1, rouge-2, rouge-L between source documents and candidates summaries.
4. copy length: the length of summary's fragments appeared in the source document.
5. fragment coverage, fragment density, compression ratio as defined in Grusky et al. (2018).
6. novelty: the ratio of novel k-grams ($k \in \{1, 2, 3, 4\}$) in the candidate summaries.
7. repetition: the ratio of repeated k-grams ($k \in \{1, 2, 3, 4\}$) in the candidate summaries.
8. sentence fusion ratio: the ratio of sentences in the candidate summaries that combine the content of two source document sentences.