CoLo: A Contrastive Learning based Re-ranking Framework for One-Stage Summarization

Chenxin An[1], **Ming Zhong**[2], **Zhiyong Wu**[3] ,**Qin Zhu**[1], **Xuanjing Huang**[1], **Xipeng Qiu**[1*]
[1]School of Computer Science, Fudan University
[2] University of Illinois at Urbana-Champaign
[3] Shanghai AI Lab
{cxan20, qzhu18, xjhuang, xpqiu}@fudan.edu.cn
mingz5@illinois.edu, wuzhiyong@pjlab.org.cn

Abstract

Traditional training paradigms for extractive and abstractive summarization systems always only use token-level or sentence-level training objectives. However, the output summary is always evaluated from summary-level which leads to the inconsistency in training and evaluation. In this paper, we propose a **Co**ntrastive **L**earning based re-ranking framework for **o**ne-stage summarization called CoLo. By modeling a contrastive objective, we show that the summarization model is able to directly generate summaries according to the summary-level score without additional modules and parameters. Extensive experiments demonstrate that CoLo boosts the extractive and abstractive results of one-stage systems on CNN/DailyMail benchmark to 44.58 and 46.33 ROUGE-1 score while preserving the parameter efficiency and inference efficiency. Compared with state-of-the-art multi-stage systems, we save more than 100 GPU training hours and obtaining $3\times \sim 8\times$ speed-up ratio during inference while maintaining comparable results.

1 Introduction

In general, there are two main paradigms to do text summarization: *abstractive* (Rush et al., 2015; Nallapati et al., 2016; Gehrmann et al., 2018) and *extractive* (Cheng and Lapata, 2016; Narayan et al., 2018b; Zhong et al., 2019, 2022) methods.

For extractive summarization, previous studies (Nallapati et al., 2017; Liu and Lapata, 2019) formulate it as a *sentence-level* sequence labeling task. However, there is an inherent gap between the *sentence-level* scoring and the *summary-level* evaluation (Zhong et al., 2020).This means that some high-scoring sentences may share the same meaning, making them not a qualified summary when combined. Similarly, the previous training paradigm for abstractive summarization models can be viewed as a *token-level* scoring process upon the decoder of sequence-to-sequence model. There also exists the issue of exposure bias (Bengio et al., 2015; Paulus et al., 2017) in the teacher-forcing framework leading to the error accumulation during auto-regressive decoding. Therefore, previous frameworks for both extractive and abstractive methods did not perform *summary-level* optimization.

To tackle this problem, state-of-the-art summarization systems (Zhong et al., 2020; Liu and Liu, 2021) are enhanced with an additional module (called re-ranker) and follow a two-stage paradigm. They first train a summarizer to model the conditional distribution $p(Y|X)$ where X is the document and Y is the output summary. Then the re-ranker is trained to re-score candidates sampled from the pre-trained summarizer in the second stage. However, this paradigm trades efficiency for accuracy, the auxiliary re-ranking greatly harms the inference efficiency especially for the highly efficient extractive systems. Experimentally, the decoding speed of two-stage re-ranking models is only ~7.0 samples/s while removing the re-ranker module will greatly boost the decoding speed to **~42.0** samples/s[1]. This makes two-stage summarization systems may be unacceptable in real-world scenarios that require timely feedback.

The limitations of the existing work motivate us to build a one-stage summarization system that can 1) replace previous naive sentence/token-level score with a summary-level score and 2) do not sacrifice the parameter and inference efficiency. In this paper, we propose a **Co**ntrastive **L**earning based re-ranking framework for **o**ne-stage summarization called CoLo for both extractive and abstractive approach. Contrastive learning has been explored in summarization (Sun and Li, 2021; An et al., 2021b) and generation (Lee et al., 2020; An et al., 2022).

[*]Corresponding author.

[1]We run these two models on the test set of CNN/DailyMail using single GeForce GTX TITAN XP GPU for 3 times and report the average speed.

Proceedings of the 29th International Conference on Computational Linguistics, pages 5783–5793
October 12–17, 2022.

CoLo uses a contrastive re-ranking training objective. We first present a novel sampling method that can be equipped to any one-stage summarization systems so that it can re-score candidates without the second stage. The existing two-stage models use **offline sampling** to preprocess samples for training of re-ranker where candidate samples are drawn from a fixed model distribution. This is a huge obstacle to turning *summarize-then-rerank* two-stage framework into an efficient end-to-end model. To solve this issue, we propose an **online sampling** approach. Concretely, instead of sampling from a fixed distribution, we draw positive and negative samples from a dynamic distribution of model outputs during training, which ultimately eliminates the requirement for additional modules in the overall framework. We then introduce a summary-level optimization strategy in addition to the traditional sentence-level (for extractive systems) or token-level loss (for abstractive systems). As a result, as a one-stage model, CoLo achieves comparable performance to two-stage systems, and greatly improves decoding speed to meet the needs of real-world applications.

We summarize our contributions as follows:

- We are the first to propose a one-stage re-ranking framework CoLo for both extractive and abstractive summarization systems.
- Results on the popular CNN/DailyMail benchmark show that both the extractive and abstractive versions of CoLo outperform previous state-of-the-art one-stage systems by a large margin. Compared to the two-stage systems, CoLo achieves comparable performance without additional pre-trained model. More importantly, CoLo do not sacrifice inference speed and thus can be more widely used in real-world scenarios.

2 Background

2.1 Preliminary about Two-Stage Systems

Two-stage paradigms (Zhong et al., 2020; Liu and Liu, 2021) improve summarization quality by re-ranking and selecting a candidate from a given set of candidates. MatchSum (Zhong et al., 2020) forms a contrastive learning based re-ranking framework where they first generate a set of candidates summaries by a extractive summarization model and then feed them to a re-ranker. The re-ranker is trained to optimize a summary-level score and it can evaluate the candidate summaries holistically. SimCLS (Liu and Liu, 2021) is the abstractive version which replaces the extractive summarizer in Zhong et al. (2020) with a abstractive summarizer.

The training objective for summarization models is to estimate a conditional probability distribution $p(Y|X)$, where X is the document and Y is the output summary. Given a summarization model $\mathcal{M}$ that has already tuned under the conventional framework with loss function $\mathcal{L}_{sum}$ where $\mathcal{L}_{sum}$ could be binary cross entropy loss (BCELoss) or negative log likelihood loss (NLLLoss). The two-stage systems should first use a sampling algorithm e.g. beam search to sample a candidate set $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of size m from the fixed model distribution $C_i \sim p_{\mathcal{M}}(Y|X)$. Candidates in $\mathcal{C}$ are sort by their ROUGE score in descending order. Then the they further train a separate re-ranker,e.g., BERT , with a contrastive-style ranking loss $\mathcal{L}_{rank}$ to select the the best candidate from $\mathcal{C}$ as the final output. The ranking loss used in the best re-ranking system for summarization is the triplet margin loss (Kingma and Ba, 2014). For a candidate pair (C_i, C_j) where $i < j$, if C_i has higher ROUGE score and it will be treated as the positive sample:

$$\mathcal{L}_{i,j} = max\{0, \cos(\mathbf{z}_X, \mathbf{z}_{C_i}) - \cos(\mathbf{z}_X, \mathbf{z}_{C_j}) + \rho\}, \quad (1)$$

where $\mathbf{z}_X$, $\mathbf{z}_{C_i}$, $\mathbf{z}_{C_j}$ are the vector feature representation of X, C_i, C_j output by the re-ranker, and ρ is the margin value. The final ranking loss is obtained by summing up all pairs: $\mathcal{L}_{rank} = \sum_j \sum_{i<j} \mathcal{L}_{i,j}$. The ranking loss ensures that candidates with higher ROUGE score is closer to the document in the embedding space.

2.2 A Comparison between Two-Stage Systems and CoLo

Figure 1 illustrates the difference between the architecture of two-stage systems and CoLo. Although MatchSum and SimCLS significantly outperform all one-stage models, they mainly suffer from three drawbacks which strongly emphasize the necessity of designing an one-stage model:

(1) Training/inference inefficiency. Building the training set of the re-ranker and the second training stage consumes large amounts of GPU and CPU time (see details in Section 5.3). Moreover, the need of re-feeding generation results to another module also requires unaffordable computational

resources.

(2) Coupling between the summarizer and re-ranker. Each improvement to one of these modules requires simultaneous updating or retraining of another module, which limits the use of such systems in the real world. For example, to try a larger candidate set or a different decoding method, we have to prepare the training set again for the second stage. In addition, how to tune the hyperparameters to be optimal in both modules at the same time is another tricky issue. Compared with two-stage systems, our one-stage system has a simple and clean implementation.

(3) Two-stage systems also face difficulties in long document summarization, because the input length of the re-ranker will drastically increase as the length of candidates increasing (see detailed analysis in Appendix A). Correspondingly, CoLo is not easily affected by length variance.

3 Method

3.1 A Naive One-Stage Re-ranking Model

The goal of one-stage re-ranking systems is to enable both training and inference to score candidate summaries holistically without requiring a second stage of computation by a separate model. Ideally, an one-stage summarization model should both function as a summarizer and a re-ranker. A straightforward solution is multi-task learning. The naive training pipeline can be formulated as follows: (i) tuning $\mathcal{M}$ with $\mathcal{L}_{sum}$. (ii) Getting positive and negative samples from $p_{\mathcal{M}}(Y|X)$ via offline sampling for each datapoint X in the training set. (iii) Building the ranking loss with these candidates and further tuning $\mathcal{M}$ with $\mathcal{L}_{rank} + \mathcal{L}_{sum}$. However, in practice, such training method is always suboptimal compared to the state-of-the-art two-stage models. We denote the model after multi-task learning as $\mathcal{M}'$. There is a serious *generalization error* in the naive methods: via multi-task learning, $\mathcal{M}'$ is only able to rank candidates drawn from the original model distribution $p_{\mathcal{M}}(Y|X)$ but not candidates from the new distribution $p_{\mathcal{M}'}(Y|X)$. This error makes the naive approach unable to directly output a good summary in sequence-level generated by itself.

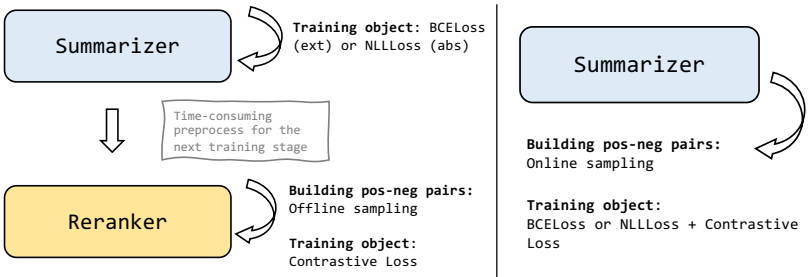
3.2 Our approach: CoLo

The first step of CoLo is also to train the summarization model with $\mathcal{L}_{sum}$ like the naive approach.

In CoLo, we discard using positive-negative samples that from a fixed model distribution, instead, we sample these candidates from a constantly shifting model distribution during multi-task learning. By doing so, we can mitigate the above mentioned generalization error as much as possible because candidates are dynamically changing with the parameters of the model distribution $p_{\mathcal{M}}(Y|X)$ updated by gradient descent. To implement this process, at each training step, we sample the newest candidates along with their feature presentations from the summarization model and calculate the ranking loss. We will give a detailed description about how we performing the online sampling process on mainstream extractive and abstractive summarization models in the following parts.

Online Sampling for Extractive Model The task of extractive summarization is to assign a label $y_i \in \{0, 1\}$ for each sentence $sent_i$ from the source document $X = (sent_1, sent_2, \ldots, sent_n)$ consisting of n sentences. Figure 2 gives an example of our one-stage extractive summarization model. Extractive candidates can be viewed as a subset of sentences from the document. In this figure, we sample $sent_1, sent_2$ to form the first candidate $C_1 = \{sent_1, sent_2\}$, and C_2 is consisting of $\{sent_2, sent_3\}$. After constructing these candidates, the next step is to represent them in the embedding space. In our one-stage model, we employ a heuristic way to obtain the feature presentations of candidates: pooling results of the sentence embedding from the extractive model. Concretely, we denote the sentence embedding for the i-th sentence as $\mathbf{h}_i$. The hidden representation of a candidate is created by pooling the sentence representations belong to it. For example $\mathbf{z}_{C_1}$ is the average pooling result of $\mathbf{h}_1$ and $\mathbf{h}_2$. Suppose C_2 has higher ROUGE score than C_1, then C_2 is treated as a positive sample and C_1 is treated as a negative sample for this pair. Finally, the whole system is trained by the sum of $\mathcal{L}_{rank}$ and $\mathcal{L}_{sum}$.

Sampling informative candidates is essential in re-ranking systems. The first step of the sampling method is to determine $\mathcal{N}$ which represents the number of candidate sentences. $\mathcal{N}$ is set depending on the number of summary sentences of downstream datasets. Take CNN/DailyMail as an example, we set $\mathcal{N}$ to $\{2, 3\}$ because most gold summaries consist of 2$\sim$3 sentences. At each training step, we iterate over $\mathcal{N}$ by combination and form m different candidates $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$. m is



(a) Two-stage models: MatchSum and SimCLS (b) CoLo (this work)

Figure 1: A comparison between two-stage models and CoLo. The two-stage models including two training stages and a time-consuming preprocess while CoLo is trained in an end-to-end fashion. (GPU and CPU hours cost in each stage are shown in Table 6). Two-stage models use offline sampling to build positive-negative pairs while CoLo builds positive-negative pairs with online sampling where we directly get theses pairs from a changing model distribution.

equal to $\sum_i C_n^{num_i}$ where num_i is the i-th element in $\mathcal{N}$ and n is number of sentences of the document. For CNN/DailyMail whose $\mathcal{N}$ is set to $\{2, 3\}$, we can sample $C_n^2 + C_n^3$ different candidates.

However, in practice, we always face the combination explosion problem when the number of sentences n grows larger. The two-stage system (Zhong et al., 2020) pre-trained an extractive model to clip the origin number of sentences to an acceptable size. Notice that our extractive summarizer is also supervised with the BCELoss, so that we can clip the sampling space to n' (a hyperparameter) with the output distribution over the sentences at each training step. Then the total size of the final candidate set decreases to $m' = \sum_i C_{n'}^{num_i}$. For CNN/DailyMail, n' is set to 5, and we can get $C_5^2 + C_5^3 = 20$ different extractive candidates. Details about the setting of $\mathcal{N}$ and n' can be found in Table 1 in Appendix.

Notably, the offline sampling needs to feed each candidate into the pre-trained encoder. In real-life setting, when summarizing some long documents, the number of sentences in the input document and output summary will increase significantly. It will bring a polynomial level increase to the computation and GPU overhead of the two-stage model. But our one-stage system with online sampling is robust to the length variance.

Inference Stage of Extractive Model Since we have modeled a summary-level score during training, it is easy to directly generate summaries according to the summary-level semantic score. Concretely, given a candidate set $\mathcal{C}$ built by the combination strategy, we calculate the cosine similarity between each candidate presentation $\mathbf{z}_{C_i}$ and the document representation $\mathbf{z}_X$:

$$\hat{C} = \max_{C_i \in \mathcal{C}} cos(\mathbf{z}_X, \mathbf{z}_{C_i}). \quad (2)$$

The final output is the candidate with highest cosine similarity score.

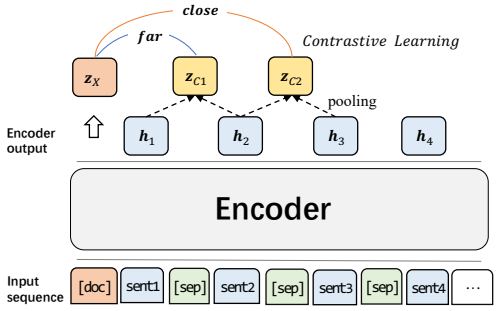


Figure 2: Architecture of our extractive model. Input sequence: The '[doc]' token is used to get vector representation $\mathbf{z}_X$ of the document X, '[sep]' is used as separator for sentences. We omit the classifier and the BCELoss. $\mathbf{h}_i$ is the sentence embedding the i-th sentence in X. $\mathbf{z}_{C_i}$ means the feature representation of the i-th candidate.

Online Sampling for Abstractive Model Our method can also be easily adapted in abstractive summarization. Selecting a generated summary maximum a posteriori (MAP) usually result in poor performance (Stahlberg and Byrne, 2019), thus most state-of-the-art generation model usually use the beam search algorithm at inference stage. The online sampling for the abstractive version is much simpler than the extractive version. We use beam search as sampling algorithm and get the feature representations from the encoder/decoder output. We denote the encoder output of source document

	CNN/DM	Reddit	XSum	SSN	PubMed
n'	5	5	5	8	8
$\mathcal{N}$	2,3	1,2	1,2	6	6,7
$\|\mathcal{C}\|$	20	15	15	28	36

Table 1: candidate size $|\mathcal{C}|$ of each datasets (extractive). $|n'|$ is the clipped candidate size, $\mathcal{N}$ is a set containing all number of possible sentence.

X as H_{enc} and the decoder hidden states of the target summary as H_{dec}. We get the document representation from the encoder output of the 0-th token $\mathbf{z}_X = H^0_{enc}$. The feature representation of the i-th candidate C_i with length = $|C_i|$ is derived from the last step of the decoder output $\mathbf{z}_{C_i} = H^{|C_i|-1}_{dec}$. Hidden states of other steps can not represent the entire sequence because of the sequence mask in transformer decoder. finally we formulate the ranking loss following Eq. 1.

Inference Stage of Abstractive Model The inference stage of our abstractive version is similar to the extractive version. We save the feature representation of the document and each beam during beam search. The final output is determind by the cosine distance between $\mathbf{z}_X$ and $\mathbf{z}_{C_i}$.

4 Experimental Setup

4.1 Datasets

We conduct experiments on five mainstream datasets to evaluate the effectiveness of our approach.

CNN/DailyMail (Hermann et al., 2015) is a classic benchmark which contains articles from the CNN/Daily Mail newspapers. We use the cased version from datasets[2]

XSum (Narayan et al., 2018a) is a one-sentence summary dataset from BBC News. Gold summaries are professionally written by the authors of documents.

Reddit (Kim et al., 2019) is collected from social media platform and we use the TIFU-long version.

PubMed (Cohan et al., 2018) is a long document summarization dataset from scientific domain whose avg summary length is about 4 times longer than CNN/DM.

SSN (An et al., 2021a) consists of papers mainly from math, physics and computer science with the abstract section as gold reference.

4.2 Implementation Details

For the simplity of experimental settings, both extractive model and abstractive mode are based on BART. We use the encoder of BART (170M) as the backbone and a 3-layer MLP as the classifier to implement the extractor. We add two special token '<cls>' to generate the sentence representation and '<sep>' as sentence separator. '<doc>' token is used to generate the document feature representation. candidate size for each dataset can be found in 1 We use adam optimizer (Kingma and Ba, 2014) learning rate schedule follows the setting in transformer (Vaswani et al., 2017). We train our model for 15000 steps with BCELoss and 32000 steps with BCELoss and RankingLoss where each step has a batch size of 36. The margin parameter γ is set to 0.01. The size of generated candidates $|\mathcal{C}|$ is set to 20 for CNN/DM. We report the results. Other settings follow the default setting in Liu and Lapata (2019). Our model is trained on single GeForce RTX 3090 GPU for 8 hours. Both our abstractive model and extractive model are trained on 24G GeForce RTX 3090 GPUs and the inference process is on 12G GeForce GTX TITAN XP GPUs.

For abstractive model, we choose BART initialized with facebook/bart-large-cnn from transformers[3] as the basic summarizer. We further fintune this model by NLLLoss and RankingLoss for 15000 steps where each step with a batch size of 8. Other setting is the same with our extractive version. To encourage diversity, we use the diverse beam search (Vijayakumar et al., 2016) to generate the candidates with beam size set to 16 and diversity penalty set to 1.0. Our model is trained on 8 GeForce RTX 3090 GPUs for about 18 hours.

4.3 Evaluation Metrics

We examine our approach with 4 metrics that measure the distance between generated summaries against the gold reference. **ROUGE** (Lin, 2004) where R-1 and R-2 measure informativeness based on n-gram overlapping and R-L represents fluency. **JS-2 Divergence** (Louis and Nenkova, 2013) measures Jensen-Shannon divergence between the bigram distributions of two input texts.

[2] https://github.com/huggingface/datasets

[3] https://github.com/huggingface/transformers

BERTScore (Zhang et al., 2019) measures soft overlap between BERT embeddings of two texts instead of using lexical matching methods. **MoverScore** (Zhao et al., 2019) is also based on the neural model but applies a earth mover distance measure to contextualized BERT embeddings.

5 Results

We denote the model without contrastive learning as the baseline system. Since the backbone of our extractive model is BART encoder so that we call the baseline model BARTEXT. The baseline model for abstractive system is BART. Our extractive model is called COLO$_{Ext}$ and its abstractive version is denoted as COLO$_{Abs}$.

5.1 Extractive Results

We compare our models with baseline models which has similar amount of parameters and decoding speed of our models in this section. Our extractive results on CNN/DM are shown in Table 2 We compare our model with previous strong extractive baseline built on pre-trained model (Zhong et al., 2019; Bae et al., 2019; Liu and Lapata, 2019) and strong multi-stage systems (Zhong et al., 2020). From the third section of Table 2, we can see that our model COLO$_{Ext}$ beats the baseline model by 1.49 ROUGE-1 score and achieve the state-of-the-art among all end-to-end systems when input length set to 512 and the results can be further improved while extending the input length to 1024. Even compared with the BERTSUM-large (340M) (Liu and Lapata, 2019) which is built on large PTM, We still have an improvement of 0.42 with only the half number of parameters of theirs. Though RL-based methods hold the motivation of optimizing towards the evaluation metric, but it does not gain much improvement on performance in practice.

To verify whether our model is effective on datasets of various lengths, we also evaluate our model on datasets with short summaries (Reddit and XSum) and long document dataset PubMed and results are shown in Table 3. On reddit and XSum, we achieve the advantage of more than 1.0 point ROUGE-1 than baseline systems and close performance with the upper bound ORACLE. We also gain improvements when tested on the long document summarzation dataset PubMed. Detailed results on long document dataset can be found in Appendix A.

Model	R-1	R-2	R-L
LEAD	40.43	17.62	36.67
ORACLE	52.59	31.23	48.87
Transformer(Vaswani et al., 2017)	40.90	18.02	37.17
BERT-EXT(Bae et al., 2019)	42.29	19.38	38.63
BERT-EXT + RL	42.76	19.87	39.11
BertSum (Liu and Lapata, 2019)	42.57	19.96	39.04
BertSum-large	<u>43.85</u>	20.34	39.90
BARTEXT	42.78	20.24	39.24
BARTEXT ($len = 1024$)	43.65	<u>20.88</u>	<u>40.19</u>
Naive one-stage	43.53	20.54	39.62
COLO$_{Ext}$	44.10	20.97	40.19
COLO$_{Ext}$ + BERTScore	44.27	21.01	40.34
COLO$_{Ext}$ ($len = 1024$)	**44.58**	**21.25**	**40.65**

Table 2: Extractive results on CNN/DM test set. len means the input length of the document, results without the marker using 512 tokens as input. +RL means the addition of reinforcement learning. +BERTScore means we use BERTScore to determine positive-negative samples. COLO clearly outperform all previous one-stage summarization systems. The best results are in bold and the second best ones are underlined.

5.2 Abstractive results

Early work also successfully applies reinforcement learning on abstractive summarization (Paulus et al., 2017; Li et al., 2019). But we do not find related works that successfully combine reinforcement learning with strong pre-trained models. Therefore, most of our baselines are strong pre-trained model finetuned with NLLLoss. Our results is shown in Table 4, due to the huge cost of using large pre-trained model with length set to 1024, we also report results with 512 input tokens and it is able to significantly outperform other baselines which has longer input length (1024). COLO$_{Abs}$ has an improvement of **2.17** R-1 socre on the very strong BART-large baseline without adding additional parameters or modules. Additionally, our method is able to outperform all one-stage baseline systems by a large margin. We also conduct experiments on long document summarzation datasets (see in Table 11 in Appendix).

5.3 Comparison with Multi-stage Systems

Apart from the one-stage systems, we also compare our model with these powerful multi-stage systems: CTRLSum, multi-stage re-ranking models. CTRLSum needs other systems to previously produce a control signal.

Model	Reddit			XSum			PubMed		
	R-1	R-2	R-L	R-1	R-2	R-L	R-1	R-2	R-L
LEAD	12.38	2.17	10.12	14.40	1.46	10.59	37.58	12.22.	33.44
ORACLE	29.10	11.08	23.10	25.62	7.62	18.72	45.12	20.33	40.19
BERTSUM	23.86	5.85	19.11	22.86	4.48	17.16	41.05	14.88	36.57
BARTEXT	23.97	5.68	19.24	22.96	4.70	17.29	41.40	16.18	37.89
COLO$_{Ext}$	**25.06**	**5.90**	**19.52**	**24.51**	**5.04**	**18.21**	**41.93**	**16.51**	**38.28**

Table 3: Results on test sets of reddit, XSum and PubMed. Our model achieve significant improvement on the baseline model BARTEXT. LEAD means we select the first k sentences from the source document as the output summary and ORACLE is the upper bound of extractive methods.

Model	R-1	R-2	R-L
BertSumAbs(Liu and Lapata, 2019)	41.72	19.39	38.76
Pegasus(Zhang et al., 2020)	44.17	21.47	41.11
BART(Lewis et al., 2020)	44.16	21.28	40.90
BART+R3F(Aghajanyan et al., 2020)	44.38	21.53	41.17
BART ($len = 512$)	43.82	20.96	40.63
ConSum (Sun and Li, 2021)	44.53	21.54	41.57
SeqCo (Xu et al., 2021)	45.02	21.80	41.75
Naive one-stage (ROUGE, $len = 512$)	43.90	20.88	40.69
COLO$_{Abs}$ (ROUGE, $len = 512$)	45.45	21.53	42.35
COLO$_{Abs}$(ROUGE)	**46.33**	**22.15**	**43.08**

Table 4: Abstractive results on CNN/DM test set. len means the maximum input length of the encoder, results without the marker using 1024 tokens as the input. ConSum (Sun and Li, 2021) and SeqCo (Xu et al., 2021) in the second block are also previous contrastive learning based methods without re-ranking.

Model	R-1	R-2	R-L
extractive systems			
COLO$_{Ext}$	44.27	21.01	40.34
BERT+BERT$^{\mathcal{R}}$ (Zhong et al., 2020)	44.22	20.62	40.38
BERT+RoBERTa$^{\mathcal{R}}$ (Zhong et al., 2020)	44.41	20.86	40.55
COLO$_{Ext}$ +RoBERTa$^{\mathcal{R}}$	**44.70**	**21.03**	**40.74**
abstractive systems			
COLO$_{Abs}$	46.33	22.15	43.08
CTRLSum(He et al., 2020)	45.65	**22.35**	42.50
BART+RoBERTa$^{\mathcal{R}}$ (Liu and Liu, 2021)	**46.67**	22.15	**43.54**

Table 5: Comparision with the multi-stage systems. RoBERTa$^{\mathcal{R}}$ means a RoBERTa re-ranker is is added to the summarization model.

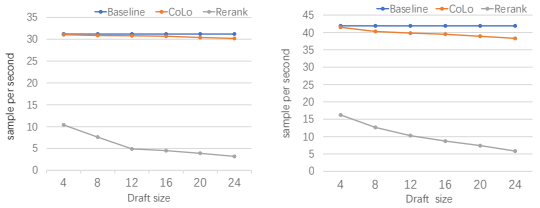
performance The addition of another pre-trained model implicitly introduces more parameters and knowledge, thus it is usually unfair to directly compare one-stage systems with the two-stage systems. But we show that COLO is able to achieve comparable performance with the multi-stage systems. As is shown in the first part of Table 5, compared with the multi-stage models that ensembles another pre-trained encoder as a re-ranker, COLO$_{Ext}$ still performs better than their BERT+BERT$^{\mathcal{R}}$ version without the need to re-feed the generated candidates to another model meanwhile we obtain a ~$\times$5 speed up over the multi-stage systems. We also try concatenating a re-ranker RoBERTa for our model, results shows that COLO$_{Ext}$ can be further improved by combing another pre-trained re-ranker reaching new extractive SOTA on the test set of CNN/DM. For abstractive models, our end-to-end model still legs behind multi-stage systems but we do not need training another model and keep similar inference speed with baseline models.

Inference Efficiency Despite the fact that multi-stage models outperform all end-to-end systems, they frequently suffer from inefficiency. In this part we mainly focus on analysing the efficiency of 3 kinds of systems: 1) baseline, which is trained only with BCELoss or NLLLoss, 2) COLO, our end-to-end constrastive learning framework, 3) Rerank, which means the multi-stage re-ranking systems. it has more 110M parameters than baseline model and COLO. The efficiency experiments for training and inference are respectively conducted on 24G RTX 3090 GPUs and 12G TITAN XP GPUs. For extractve summarization, figures 3(a),3(b) give a detailed comparison of the inference speed between the three models. Y-axis represents the number of samples processed per second. To give a fair comparison, we test the inference efficiency in two settings: i) all models are tested with batch size fixed to 1. ii) all models are tested with the maximum batch size allowed by the GPU. While the candidate size varies from 4$\sim$32, both our model have a $\mathbf{3}\times \sim \mathbf{8}\times$ speed-up ratio over the multi-stage re-ranking model. When the candidate size

Systems	Stage1	Preprocess	Stage2	Total hours
Ext+RoBERTa$^{\mathcal{R}}$	4	5 (+20)	128	137 (+20)
CoLo$_{Ext}$	7	–	–	7 ($\downarrow$ **130**)
Abs+RoBERTa$^{\mathcal{R}}$	80	132 (+18)	128	340 (+18)
CoLo$_{Abs}$	224	–	–	224 ($\downarrow$ **116**)

Table 6: GPU hours spent on training for each process on the training set of CNN/DM(reported results are rounded down after the decimal point. Ext+RoBERTa$^{\mathcal{R}}$/Abs+RoBERTa$^{\mathcal{R}}$ denotes the multi-stage re-ranking systems with an extracitve/abstrastive summarizer. (+18)/(+20) means 18/20 CPU hours are spent on calculate ROUGE score for each candidate with 32 threads.

is set to 20, the baseline model is able to process ~31.2/41.9 (batch = 1/*MAX*) samples per second, the decoding speed of CoLo$_{Ext}$ is ~30.4/38.9 samples/s (batch=1/*MAX*) and the decoding speed of the multi-stage re-ranking model is only ~4.9/7.0 samples/s(batch=1/*MAX*). Our model almost does no harm on inference speed while the candidate size $|\mathcal{C}|$ is less than 16. However, when the candidate size grows larger there is more time spent on generating the representations of the candidates. Figure 4 show the comparison of inference time of the abstractive models. While the bottleneck of abstractive models is the auto-regressive generation process. Our abstractive model generally save ~0.5 GPU hours compared to the re-ranking model.



(a) CNN/DM (batch=1) (b) CNN/DM (batch =*MAX*)

Figure 3: Inference speed on CNN/DM (extractive). we use the candidate size $|\mathcal{C}|$ as the X-axis. The Y-axis represents the number of samples processed per second. batch=*MAX* means we use the maximum batch size allowed by GPU memory.

Training Efficiency Table 6 gives an overview of the training time of our system and the multi-stage models on the training set of CNN/DM. The general pipeline for the multi-stage models is: i) training a generator (Stage1), ii) Preprocess, ii) training a re-ranker (Stage2). The preprocess includes generating the training/dev/test set for training re-ranker and sorting candidates by ROUGE. For extractive system we save **130** GPU hours compared to the multi-stage systems whose bottleneck is training the re-ranking model. For abstractive model, apart from the 128 GPU hours spent on training the ranker, using beam search to generate the training set for re-ranker model is also very time consuming, generally we obtain **116** GPU hours and 18 CPU hours saved.

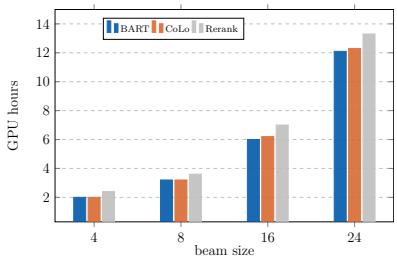


Figure 4: Test inference time with beam size for abstractive model. We use the maximum batch size allowed by GPU memory.

Metric Used	R-1	R-2	R-L	JS-2	BS	MS
Baseline	42.78	20.24	39.23	54.24	43.52	58.27
ROUGE-1,2	44.10	20.97	40.19	54.07	44.26	58.63
ROUGE-L	44.09	20.93	**40.34**	54.06	44.32	58.60
JS-2	43.85	**21.13**	39.98	**53.92**	44.19	58.60
BERTScore	**44.27**	21.01	**40.34**	54.08	**44.85**	58.71
MoverScore	44.21	20.81	40.25	54.33	44.47	**58.78**

Table 7: Extractive results of using different evaluation metrics as the discriminator on CNN/DM test set.

5.4 Ablation for Different Discriminators

In addition to ROUGE, we also select other metrics as the discriminator (shown in Table 7). ROUGE and JS-2 is based on lexical matching while BERTScore and MoverScore are based on the contextualized embedding from BERT. Our model generally obtains the best results on the metric used in training. Because these metrics are not actually separated, using one of these metrics as the discriminator can also gain significant improvements on other metrics. Overall, the neural evaluation metric BERTScore and MoverScore bring more improvements compared with metrics that based on the lexical matching. But incorporating neural model based metrics in training will obviously increase the training time.

5.5 Visualization Experiment

We conduct a visualization experiment on our extractive model to get a close look on the distribution of candidates in semantic space. We randomly sample 100 documents with more than 10 sentences from the test set of CNN/DM. We first select the top 10 sentences based on the predicted score from the classifier. We set the possible number of sentences to $\{2, 3\}$ resulting a candidate size of $C_{10}^2 + C_{10}^3 = 165$ for each sample. We visualize the learned embedding of these candidates and the anchor in a two-dimensional space by applying the t-SNE algorithm. As shown in Figure 5, there is an obvious cluster of the top 50 candidates (colored in purple) and the candidates with higher score are closer to the anchor while the distribution of uninformative candidates (gray,cyan points) is relatively random.

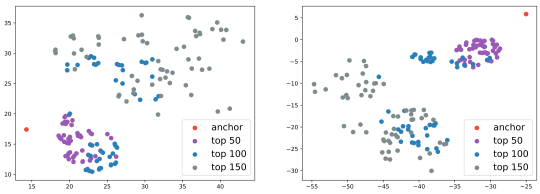


Figure 5: T-SNE Visualization of two examples from CNN/DM test set. We divide the candidates into 3 groups based on ROUGE score: candidates ranking 1~50, candidates ranking 51~100, candidate ranking 101~150. The red point denotes the anchor and the purple/cyan/gray points respectively denote the top 50/100/150 candidates.

5.6 Human Evaluation

We also conduct a human evaluation on our models to get more accurate results . We randomly select 30 articles from the test set of CNN/DM, and each articles have 5 candidate summaries 4 from automatic systems and 1 is the gold reference. We recruit 2 PhD students majoring in computer science and ask them to rank the candidate summries based on the fluency, informativeness. If two of these systems generate the same summary for the source document, this sample will be filtered out. As we can see from Table 8, the COLO$_{Ext}$ with the discriminator as BERTScore achieve the best result among all automatic systems. However, using BERTScore will bring much training time. We also suggest taking JS-2 divergence as the discriminator which also does a good job in human evaluation.

Metric Used	1st	2nd	3rd	4th	5th	Avg R.
Baseline	0%	8.3%	8.3%	23.3%	60%	4.33
JS2	6.7%	25%	33.3%	21.7%	13.3%	3.10
R1+R2	5%	20%	28.3%	30.3%	16.7%	3.35
BERTScore	10%	35%	20%	25%	10%	2.90
Gold label	78.3%	11.7%	10%	0%	0%	1.32

Table 8: Results of human evaluation results. Baseline means the BARTEXT model, Gold-label means the means the human written summary. Avg R. denotes the average ranking of the system.

6 Limitations and Future Work

Compared with the most well-known contrastive learning framework simCLR (Chen et al., 2020) which propose to construct positive and negative pairs from training samples in the same batch, Drawing negative-positive pairs from the summarization model requires more training time. Ideally, providing more positive and negative samples will benefit the performance of COLO . However, decoding with very large beam size in training mode will cost more GPU memory and training time. Future work can search for an efficient way to construct these positive-negative pairs to perform reranking during training.

7 Conclusion

We introduce COLO, a contrastive learning based summarization framework for one-stage summarization where positive-negative pairs are generated directly from the summaizer with online sampling. COLO can be both easily applied on extractive and abstractive methods. Results show that we greatly exceed previous stage-of-the art one-stage systems with no additional parameters and obivious decline of the inference efficiency.

Acknowledgement

We would like to thank Yixin Liu and the anonymous reviewers for their valuable advice. This work was supported by the National Key Research and Development Program of China (No.2020AAA0106702) and National Natural Science Foundation of China (No.62022027).

References

Armen Aghajanyan, Akshat Shrivastava, Anchit Gupta, Naman Goyal, Luke Zettlemoyer, and Sonal Gupta.

2020. Better fine-tuning by reducing representational collapse. *arXiv preprint arXiv:2008.03156*.

Chenxin An, Jiangtao Feng, Kai Lv, Lingpeng Kong, Xipeng Qiu, and Xuanjing Huang. 2022. Cont: Contrastive neural text generation. *arXiv preprint arXiv:2205.14690*.

Chenxin An, Ming Zhong, Yiran Chen, Danqing Wang, Xipeng Qiu, and Xuanjing Huang. 2021a. Enhancing scientific papers summarization with citation graph. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12498–12506.

Chenxin An, Ming Zhong, Zhichao Geng, Jianqiang Yang, and Xipeng Qiu. 2021b. Retrievalsum: A retrieval enhanced framework for abstractive summarization. *arXiv preprint arXiv:2109.07943*.

Sanghwan Bae, Taeuk Kim, Jihoon Kim, and Sanggoo Lee. 2019. Summary level training of sentence rewriting for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 10–20.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. *arXiv preprint arXiv:1506.03099*.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR.

Jianpeng Cheng and Mirella Lapata. 2016. Neural summarization by extracting sentences and words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 484–494.

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)*, volume 2, pages 615–621.

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.

Junxian He, Wojciech Kryściński, Bryan McCann, Nazneen Rajani, and Caiming Xiong. 2020. Ctrlsum: Towards generic controllable text summarization. *arXiv preprint arXiv:2012.04281*.

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 1684–1692.

Byeongchang Kim, Hyunwoo Kim, and Gunhee Kim. 2019. Abstractive summarization of reddit posts with multi-level memory networks. 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 2519–2531.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Seanie Lee, Dong Bok Lee, and Sung Ju Hwang. 2020. Contrastive learning with adversarial perturbations for conditional text generation. *arXiv preprint arXiv:2012.07280*.

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.

Siyao Li, Deren Lei, Pengda Qin, and William Yang Wang. 2019. Deep reinforcement learning with distributional semantic rewards for abstractive summarization. *arXiv preprint arXiv:1909.00141*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

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 3721–3731.

Yixin Liu and Pengfei Liu. 2021. Simcls: A simple framework for contrastive learning of abstractive summarization. *arXiv preprint arXiv:2106.01890*.

Annie Louis and Ani Nenkova. 2013. Automatically assessing machine summary content without a gold standard. *Computational Linguistics*, 39(2):267–300.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2017. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

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.

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

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.

Felix Stahlberg and Bill Byrne. 2019. On nmt search errors and model errors: Cat got your tongue? 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 3356–3362.

Shichao Sun and Wenjie Li. 2021. Alleviating exposure bias via contrastive learning for abstractive text summarization. *arXiv preprint arXiv:2108.11846*.

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*, pages 5998–6008.

Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. 2016. Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv preprint arXiv:1610.02424*.

Shusheng Xu, Xingxing Zhang, Yi Wu, and Furu Wei. 2021. Sequence level contrastive learning for text summarization. *arXiv preprint arXiv:2109.03481*.

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

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

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.

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, ACL 2020, Online, July 5-10, 2020*, pages 6197–6208. Association for Computational Linguistics.

Ming Zhong, Pengfei Liu, Danqing Wang, Xipeng Qiu, and Xuan-Jing 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.

Ming Zhong, Yang Liu, Suyu Ge, Yuning Mao, Yizhu Jiao, Xingxing Zhang, Yichong Xu, Chenguang Zhu, Michael Zeng, and Jiawei Han. 2022. Unsupervised summarization with customized granularities. *arXiv preprint arXiv:2201.12502*.