Learning Non-Autoregressive Models from Search for Unsupervised Sentence Summarization

Anonymous ACL submission

Abstract

Text summarization aims to generate a short summary for an input text. In this work, we propose a Non-Autoregressive Unsupervised Summarization (NAUS) approach, which does not require parallel data for training. Our NAUS first performs edit-based search towards a heuristically defined score, and generates a summary as pseudo-groundtruth. Then, we train an encoder-only non-autoregressive Transformer based on the search result. We also propose a dynamic programming approach for length-control decoding, which is important for the summarization task. Experiments on two datasets show that NAUS achieves state-of-the-art performance for unsupervised summarization, yet largely improving inference efficiency. Further, our algorithm is able to perform explicit length-transfer summary generation.[1]

1 Introduction

Text summarization is an important natural language processing (NLP) task, aiming at generating concise summaries for given texts while preserving the key information. It has extensive real-world applications such as headline generation (Nenkova et al., 2011).

State-of-the-art text summarization models are typically trained in a supervised way with large training corpora, comprising pairs of long texts and their summaries (Zhang et al., 2020; Aghajanyan et al., 2020, 2021). However, such parallel data are expensive to obtain, preventing the applications to less popular domains and less spoken languages.

Unsupervised text generation has been attracting increasing interest, because it does not require parallel data for training. One widely used approach is to compress a long text into a short one, and to reconstruct it to the long text by a cycle consistency loss (Miao and Blunsom, 2016; Wang and Lee, 2018; Baziotis et al., 2019). Due to the indifferentiability of the compressed sentence space, such an approach requires reinforcement learning (or its variants), which makes the training difficult (Kreutzer et al., 2021).

Recently, Schumann et al. (2020) propose an edit-based approach for unsupervised summarization. Their model maximizes a scoring function that evaluates the quality (fluency and semantics) of the generated summary, achieving higher performance than cycle-consistency methods. However, the search approach is slow in inference because hundreds of search steps are needed for each data sample. Moreover, their approach can only select words from the input sentence with the word order preserved. Thus, it is restricted and may generate noisy summaries due to the local optimality of search algorithms.

To address the above drawbacks, we propose a Non-Autoregressive approach to Unsupervised Summarization (NAUS). The idea is to perform search as in Schumann et al. (2020) and, inspired by Li et al. (2020), to train a machine learning model to smooth out such noise and to speed up the inference process. Different from Li et al. (2020), we propose to utilize *non-autoregressive* text generators, which generate all tokens in the output in parallel, based on our following observations:

- Non-autoregressive models are several times faster than autoregressive generation, which is important when the system is deployed.

- The input and output of the summarization task have a strong correspondence. Non-autoregressive generation supports encoder-only architectures, which can better utilize such input–output correspondence and even outperform autoregressive models for summarization.

- For non-autoregressive models, we can design a length-control algorithm based on dynamic programming. This can satisfy the output length constraint, which is typical in summarization but can-

[1] Our code is released on real but anonymized repo: https://github.com/ARR-NAUS/NAUS

not be easily achieved with autoregressive models.

We conducted experiments on Gigaword headline generation (Graff et al., 2003) and DUC2004 (Over and Yen, 2004) datasets. Experiments show that our NAUS achieves state-of-the-art performance on unsupervised summarization; especially, it outperforms its teacher (i.e., the search approach), confirming that NAUS can indeed smooth out the search noise. Regarding inference efficiency, our NAUS with truncating is 1000 times more efficient than the search approach; even with dynamic programming for length control, NAUS is still 100 times more efficient than search and several times more efficient than autoregressive models. Our NAUS is also able to perform length-transfer summary generation, i.e., generating summaries of different lengths from training.

2 Approach

In our approach, we first follow Schumann et al. (2020) and obtain a summary by discrete search towards a heuristically defined objective function (§2.1). Then, we propose a non-autoregressive model for the summarization task (§2.2). We present the training strategy and the proposed length-control algorithm in §2.3.

2.1 Search-Based Summarization

Consider a given source text $\mathbf{x} = (\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_n)$. The goal of summarization is to find a shorter text $\mathbf{y} = (\mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_m)$ as the summary.

Our work on unsupervised summarization follows the recent progress of search-based text generation. Schumann et al. (2020) formulate summarization as word-level extraction (with order preserved), and apply edit-based discrete local search to maximize a heuristically designed objective.

Specifically, the objective function considers two aspects: (1) a language fluency score $f_{\mathrm{LM}}(\mathbf{y})$, given by the reciprocal of a language model's perplexity; and (2) a semantic similarity score $f_{\mathrm{SIM}}(\mathbf{y}; \mathbf{x})$, given by the cosine embeddings. The overall objective combines the two aspects as

$$f(\mathbf{y}; \mathbf{x}) = f_{\mathrm{LM}}(\mathbf{y}) \cdot f_{\mathrm{SIM}}(\mathbf{y}; \mathbf{x})^{\gamma} \qquad (1)$$

where γ is a weighting hyperparameter. Interested readers are referred to Schumann et al. (2020) for the details of the scoring function.

Further, the desired summary length can be specified as a hard constraint, achieved by searching only among sentences of the correct length. Suppose the desired summary length is T, the approach selects T random words from the input, and maximizes the scoring function (1) by changing the selection and non-selection of two words.

A greedy hill-climbing algorithm determines whether the change is accepted or not. In other words, a change is accepted if the score improves, or rejected otherwise. Such a process continues until a (possibly local) optimum is found.

A pilot analysis in Schumann et al. (2020) shows that words largely overlap between a source text and its summary. This explains the high performance of such a word extraction approach, being a state-of-the-art unsupervised summarization system and outperforming strong competitors, e.g., cycle consistency (Wang and Lee, 2018; Baziotis et al., 2019).

2.2 Non-Autoregressive Model for Summarization

Despite the high performance, such edit-based search has several drawbacks. First, the search process is slow because hundreds of local search steps are needed to obtain a high-quality summary. Second, their approach only extracts the original words with order preserved. Therefore, the generated summary is restricted and may be noisy.

To this end, we propose a Non-Autoregressive approach to Unsupervised Summarization (NAUS) by learning from the search results. In this way, the machine learning model can smooth out the search noise and is much faster, largely alleviating the drawbacks of search-based summarization. Compared with training an autoregressive model from search (Li et al., 2020), non-autoregressive generation predicts all the words in parallel, further improving inference efficiency by several times.

Moreover, a non-autoregressive model enables us to design an encoder-only architecture, which is more suited to the summarization task due to the strong correspondence between input and output, which cannot be fully utilized by encoder–decoder models, especially autoregressive ones.

Specifically, we propose to use multi-layer Transformer (Vaswani et al., 2017) as the non-autoregressive architecture for summarization. Each Transformer layer is composed of a multi-head attention sublayer and a feed-forward sublayer. Additionally, there is a residual connection in each sublayer, followed by layer normalization.

Let $X^{(n)} \in \mathbb{R}^{T \times d}$ be representation at the nth

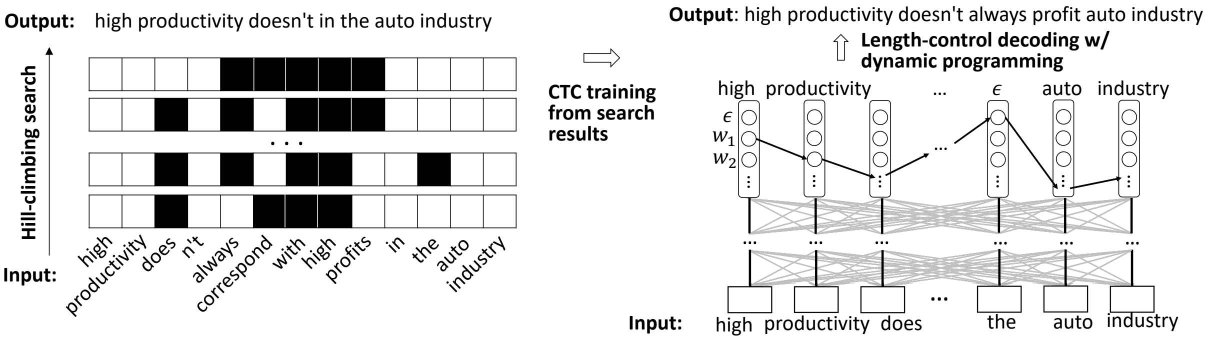
Output: high productivity doesn't in the auto industry
Hill-climbing search
Input:
high
productivity
does
n't
always
correspond
with
high
profits
in
the
auto
industry
CTC training from search results
Output: high productivity doesn't always profit auto industry
Length-control decoding w/ dynamic programming
high productivity ... ϵ auto industry
ϵ
w1
w2
Input: high productivity does ... the auto industry

Figure 1: The overview of our NAUS approach.

layer, where T is the number of words and d is the dimension. Specially, the input layer $X^{(0)}$ is the embeddings of words. Suppose we have h attention heads. The output of the ith head in the nth attention sublayer is $A_i^{(n)} = \operatorname{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_k}}\right)V_i$, where Q_i, K_i, and V_i are matrices calculated by three distinct multi-layer perceptrons (MLPs) from $X^{(n-1)}$; d_k is the attention dimension.

Multiple attention heads are then concatenated:

$$A^{(n)} = \operatorname{Concat}\left(A_1^{(n)}, \ldots, A_h^{(n)}\right)W_O$$

where $W_O \in \mathbb{R}^{d\times d}$ is a weight matrix.

Then, we have a residual connection and layer normalization by

$$\bar{A}^{(n)} = \operatorname{LayerNorm}\left(X^{(n-1)} + A^{(n)}\right) \qquad (2)$$

Further, an MLP sublayer processes $\bar{A}^{(n)}$, followed by residual connection and layer normalization, yielding the nth layer's representation

$$X^{(n)} = \operatorname{LayerNorm}\left(\bar{A}^{(n)} + \operatorname{MLP}(\bar{A}^{(n)})\right) \quad (3)$$

The last layer $X^{(N)}$ is fed to softmax to predict the summary in a non-autoregressive manner, that is, the probability at the tth step is given by $\operatorname{softmax}(W\boldsymbol{x}_t^{(N)})$, where $\boldsymbol{x}_t^{(N)}$ is the tth row of the matrix $X^{(N)}$ and W is the softmax weight.

It is emphasized that, in the vocabulary, we include a special blank token ϵ, which is handled by dynamic programming during both training and inference (§2.3). This enables us to generate a shorter summary than the input with such a multi-layer Transformer.

Our model can be thought of as an encoder-only architecture, differing from a typical encoder–decoder model with cross attention (Vaswani et al., 2017; Baziotis et al., 2019; Zhou and Rush, 2019). Previously, Su et al. (2021) propose a seemingly similar model to us, but put multiple end-of-sequence (EOS) tokens at the end of the generation; thus, they are unable to maintain the correspondence between input and output. Instead, we allow blank tokens scattering over the entire sentence; thus, the residual connections in Eqns (2) and (3) can better utilize such input–output correspondence for summarization.

2.3 Training and Inference

In this section, we first introduce the Connectionist Temporal Classification (CTC) training. Then, we propose a length-control decoding approach for summary generation.

CTC Training. The Connectionist Temporal Classification (CTC, Graves et al., 2006) algorithm allows a special blank token ϵ in the vocabulary, and uses dynamic programming to marginalize out such blank tokens. In addition, non-autoregressive generation suffers from a common problem that words may be repeated in consecutive steps (Gu et al., 2018; Lee et al., 2018); thus, CTC merges repeated words unless separated by ϵ. For example, the sequence of tokens $a\epsilon\epsilon aabb\epsilon$ is reduced to the text aab, denoted by $\Gamma(a\epsilon\epsilon aabb\epsilon) = aab$. The CTC training is by maximum marginal likelihood estimation, treating the predictors as unobserved latent variables.

Concretely, the likelihood is marginalized over all possible fillings of ϵ, i.e., all possible token sequences that are reduced to the groundtruth text:

$$P(\mathbf{y}|\mathbf{x}) = \sum\nolimits_{\mathbf{w}:\Gamma(\mathbf{w})=\mathbf{y}} P(\mathbf{w}|\mathbf{x}) \qquad (4)$$

where $P(\mathbf{w}|\mathbf{x})$ is the probability of generating a sequence of tokens $\mathbf{w}$. Although enumerating every candidate in $\{\mathbf{w} : \Gamma(\mathbf{w}) = \mathbf{y}\}$ is intractable, such marginalization fortunately can be computed by dynamic programming in an efficient way.

Let $\alpha_{s,t} = \sum_{\mathbf{w}_{1:s}:\Gamma(\mathbf{w}_{1:s})=\mathbf{y}_{1:t}} P(\mathbf{w}_{1:s}|\mathbf{x})$ be the marginal probability of generating $\mathbf{y}_{1:t}$ up to the

sth decoding slot. Moreover, $\alpha_{s,0}$ is defined to be the probability that $\mathbf{w}_{1:s}$ is all ϵ, thus not having matched any word in $\mathbf{y}$. The $\alpha_{s,t}$ variable can be further decomposed into two terms $\alpha_{s,t} = \alpha^{\epsilon}_{s,t} + \alpha^{\neg\epsilon}_{s,t}$, where the first term is such probability with $\mathrm{w}_s = \epsilon$, and the second term $\mathrm{w}_s \neq \epsilon$. Apparently, the initialization of α variables is

$$\alpha^{\epsilon}_{1,0} = P(\mathrm{w}_1 = \epsilon|\mathbf{x}) \tag{5}$$

$$\alpha^{\neg\epsilon}_{1,1} = P(\mathrm{w}_1 = \mathrm{y}_1|\mathbf{x}) \tag{6}$$

$$\alpha^{\epsilon}_{1,t} = 0, \forall t \geq 1 \tag{7}$$

$$\alpha^{\neg\epsilon}_{1,t} = 0, \forall t > 1 \text{ or } t = 0 \tag{8}$$

Eqn. (7) is because, at the first prediction slot, the empty token ϵ does not match any target words; Eqn. (8) is because the predicted non-ϵ first token must match exactly the first target word.

The recursion formula for $\alpha^{\epsilon}_{s,t}$ is

$$\alpha^{\epsilon}_{s,t} = \alpha_{s,t-1} P(\mathrm{w}_t = \epsilon|\mathbf{x})$$

since the newly predicted token ϵ with probability $P(\mathrm{w}_t = \epsilon|\mathbf{x})$ does not match any target word, inheriting $\alpha_{s,t-1}$.

The recursion formula for $\alpha^{\neg\epsilon}_{s,t}$ is

$$\alpha^{\neg\epsilon}_{s,t} = \begin{cases} \left(\alpha^{\epsilon}_{s-1,t-1} + \alpha^{\neg\epsilon}_{s-1,t}\right) P(\mathrm{w}_s = \mathrm{y}_t|\mathbf{x}), \\ \qquad\qquad\qquad\qquad \text{if } \mathrm{y}_t = \mathrm{y}_{t-1} \\ \alpha_{s-1,t-1} P(\mathrm{w}_s = \mathrm{y}_t|\mathbf{x}), \ \text{otherwise.} \end{cases}$$

Here, w_s is not ϵ, so we must have $\mathrm{w}_s = \mathrm{y}_t$, having the predicted probability $P(\mathrm{w}_s = \mathrm{y}_t|\mathbf{x})$.

If $\mathrm{y}_t = \mathrm{y}_{t-1}$, then we have two sub-cases: first, $\mathbf{w}_{1:s-1}$ is reduced to $\mathbf{y}_{1:t-1}$ with $\mathrm{w}_{s-1} = \epsilon$ separating two repeating words in $\mathbf{y}$, having probability $\alpha^{\epsilon}_{s-1,t-1}$; or second, $\mathbf{w}_{1:s-1}$ is reduced to $\mathbf{y}_{1:t}$ with $\mathrm{w}_{s-1} = \mathrm{y}_t \neq \epsilon$, having probability $\alpha^{\neg\epsilon}_{s-1}$, which implies we are merging w_{s-1} and w_s.

If $\mathrm{y}_t \neq \mathrm{y}_{t-1}$, then we only require w_{s-1} is reduced to y_{t-1}, where w_{s-1} can be either ϵ or non-ϵ. This is given by probability $\alpha_{s-1,t-1} = \alpha^{\epsilon}_{s-1,t-1} + \alpha^{\neg\epsilon}_{s-1,t-1}$.

Finally, $\alpha_{|\mathbf{w}|,|\mathbf{y}|}$ is the marginal probability in Eqn. (4), as it is the probability that the entire generated sequence matches the entire target text.

The CTC maximum likelihood estimation is to maximize the marginal probability, which is equivalent to minimizing the loss $-\alpha_{|\mathbf{w}|,|\mathbf{y}|}$. Since the dynamic programming formulas are differentiable, the entire model can be trained by backpropagation in an end-to-end manner with auto-differentiation tools (such as PyTorch).

Length-Control Inference. Controlling output length is the nature of the summarization task, for example, displaying a short news headline on a mobile device. Moreover, Schumann et al. (2020) show that the main evaluation metric ROUGE (Lin, 2004) is sensitive to the summary length, and longer summaries tend to achieve higher ROUGE scores. Thus, it is crucial to control the summary length for fair comparison.

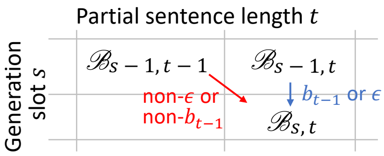

Figure 2: Illustration of our length-control algorithm.

We propose a length-control algorithm by dynamic programming (DP), following the nature of CTC training. However, our DP is an approximate algorithm because of the dependencies introduced by removing consecutive repeated tokens. Thus, we equip our DP with a beam search mechanism.

We define $\mathscr{B}_{s,t}$ to be a set of top-B sequences with s predicted tokens that are reduced to t words. $\mathscr{B}_{s,t}$ is constructed by three scenarios.

First, the blank token ϵ is predicted for the sth generation slot, and thus the summary length t remains the same, shown by the blue arrow in Figure 2. This yields a set of candidates

$$\mathscr{B}^{(1)}_{s,t} = \left\{ \mathbf{b} \oplus \epsilon \,|\, \mathbf{b} \in \mathscr{B}_{s-1,t} \right\} \tag{9}$$

where $\oplus$ refers to string/token concatenation.

Second, a repeated word is predicted for the sth generation slot, i.e., b_{s-1} for a subsequence $\mathbf{b}$ of length $s-1$. In this case, the summary length t also remains the same, also shown in the blue arrow in Figure 2. This gives a candidate set

$$\mathscr{B}^{(2)}_{s,t} = \left\{ \mathbf{b} \oplus \mathrm{b}_{s-1} \,|\, \mathbf{b} \in \mathscr{B}_{s-1,t} \right\} \tag{10}$$

Third, a non-ϵ, non-repeating word w_s is generated, increasing the summary length from $t-1$ to t, shown by the red arrow in Figure 2. This gives

$$\begin{aligned} \mathscr{B}^{(3)}_{s,t} = &\Big\{ \mathbf{b} \oplus \mathrm{w}^* \,|\, \mathbf{b} \in \mathscr{B}_{s-1,t-1}, \\ &\quad \mathrm{w}^* = \operatorname*{argmax}_{\mathrm{w}_s \neq \epsilon, \mathrm{w}_s \neq \mathrm{b}_{s-1}} P(\mathrm{w}_s|\mathbf{x}) \Big\} \end{aligned} \tag{11}$$

Based on the three candidates sets, we select top-B sequences to keep the beam size fixed:

$$\mathscr{B}_{s,t} = \mathrm{top}_B(\mathscr{B}^{(1)}_{s,t} \cup \mathscr{B}^{(2)}_{s,t} \cup \mathscr{B}^{(3)}_{s,t}) \tag{12}$$

where top_B ranks the sequences by their predicted joint probabilities.

Theorem 1. *(1) If repeating tokens are not merged, then the proposed length-control algorithm with beam size $B = 1$ finds the exact optimum $\mathscr{B}_{S,T}$ being the most probable length-T sentence given by S prediction slots. (2) If we merge repeating tokens predicted by CTC-trained models, the above algorithm may not be exact.*

Appendix A presents the proof of the theorem and provides a more detailed analysis, showing that our length-control algorithm, although being approximate inference, can generate a summary of the desired length properly. Compared with truncating an overlength output, our approach is able to generate more fluent and complete sentences. Also, our length-control algorithm is different from conventional beam search, shown in Appendix C.

3 Experiments

3.1 Setup

Datasets. We evaluated our NAUS model on Gigaword headline generation and DUC2004 datasets.

The head generation dataset (Rush et al., 2015) is constructed from the Gigaword news corpus (Graff et al., 2003), where the first sentence of a news article is considered as input text and the news title is considered as the summary. The dataset contains 3.8M/198K/1951 samples for training/validation/test. Based on the curve in Appendix B, we used 3M samples for training NAUS.

It should be emphasized that, when NAUS learns from search, we only use the input of the training corpus: we perform search (Schumann et al., 2020) for each input, and train our NAUS from the search results. Therefore, we do not utilize any labeled parallel data, and our approach is unsupervised.

Moreover, we considered two settings with desired summary lengths of 8 and 10, following Schumann et al. (2020). Our NAUS is trained from respective search results.

The DUC2004 dataset (Over and Yen, 2004) is designed for testing only with 500 samples, where we also take the first sentence of an article as the input text. Our NAUS is transferred from the above headline generation corpus. Based on the length of DUC2004 summaries, we trained NAUS from search results with 13 words, also following Schumann et al. (2020) for fair comparison.

Evaluation Metrics. We evaluated the quality of predicted summaries by ROUGE scores (Lin, 2004), which are the most widely used metrics in previous work (Wang and Lee, 2018; Baziotis et al., 2019; Zhou and Rush, 2019). Specifically, ROUGE-n evaluates n-gram overlap between a predicted summary and its reference summary; ROUGE-L, instead, measures the longest common sequence between the predicted and reference summaries.

Different ROUGE variants are adopted in previous work, depending on the dataset. We followed the standard evaluation scripts and evaluated headline generation by ROUGE F1 (Wang and Lee, 2018; Baziotis et al., 2019; Schumann et al., 2020) and DUC2004 by Truncate ROUGE Recall (Dorr et al., 2003; West et al., 2019).

In addition to summary quality, we also evaluated inference efficiency of different methods, as it is important for the deployment of deep learning models in real-time applications. We report the average inference time in seconds for each data sample, and compare the speedup with Schumann et al. (2020)'s search approach, which achieves (previous) state-of-the-art ROUGE scores. Our experiments were conducted on an i9-9940X CPU and an RTX6000 graphic card. Other implementation details are presented in Appendix B.

3.2 Results and Analyses

Main Results. Table 1 presents the performance of our model and baselines on the Gigaword headline test set. For a fair comparison, we categorize all approaches by average summary lengths of ~8 and ~10 into Groups A and B, respectively.

The Lead baseline extracts the first several words of the input sentence. Despite its simplicity, the Lead approach is a strong summarization baseline adopted in most previous work (Févry and Phang, 2018; Baziotis et al., 2019).

Wang and Lee (2018) utilize cycle consistency (Miao and Blunsom, 2016) for unsupervised summarization; Zhou and Rush (2019) perform beam search towards a step-by-step decomposable score of fluency and contextual matching. Both are unable to explicitly control the summary length: in a fair comparison of length 10 (Group B, Table 1), their performance is worse than the (previous) state-of-the-art approach (Schumann et al., 2020),[2] which performs edit-based local search.

Our NAUS approach follows Schumann et al.

[2] Schumann et al. (2020) present a few variants that use additional datasets for training language models (in an unsupervised way). In our study, we focus on the setting without data augmentation, i.e., the language model is trained on non-parallel the Gigawords corpus.

Group	#	Approach		Len	ROUGE F1				Inf.Time	Speedup
					R-1	R-2	R-L	ΔR		
A (desired length 8)	1	Baseline	Lead (8 words)†	7.9	21.39	7.42	20.03	-11.12	–	–
	2	Search	Schumann et al. (2020)†	7.9	26.32	9.63	24.19	0.18	–	–
	3		Our replication	7.9	26.17	**9.69**	24.10	0	6.846	1x
	4	Learn from search	Su et al. (2021)	7.7	26.88	9.37	24.54	0.83	0.017	403x
	5		NAUS (truncate)	7.8	27.27	9.49	24.96	1.76	**0.005**	**1369x**
	6		NAUS (length control)	7.8	**27.94**	9.24	**25.51**	**2.73**	0.041	167x
B (desired length 10)	7	Baseline	Lead (10 words)†	9.8	23.03	7.95	21.29	-10.2	–	–
	8		Wang and Lee (2018)†	10.8	27.29	10.01	24.59	-0.58	–	–
	9		Zhou and Rush (2019)†	9.3	26.48	10.05	24.41	-1.53	–	–
	10	Search	Schumann et al. (2020)†	9.8	27.52	**10.27**	24.91	0.23	–	–
	11		Our replication	9.8	27.35	10.25	24.87	0	9.217	1x
	12	Learn from search	Su et al. (2021)	9.4	27.86	9.88	25.51	0.78	0.020	461x
	13		NAUS (truncate)	9.8	28.24	10.04	25.40	1.21	**0.005**	**1843x**
	14		NAUS (length control)	9.8	**28.55**	9.97	**25.78**	**1.83**	0.044	210x

Table 1: Results on the Gigaword headline generation test set. **Len:** Average length of predicted summaries. **R-1, R-2, R-L:** ROUGE-1, ROUGE-2, ROUGE-L. **ΔR:** The difference of total ROUGE (sum of R-1, R-2, and R-L) in comparison with the (previous) state-of-the-art search method under replication. **Inf.Time:** Average inference time in seconds for one sample on an i9-9940X CPU and a RTX6000 GPU. **Speedup:** Relative to Schumann et al. (2020). †Results quoted from previous papers; others are given by our experiments.

Model	ROUGE Recall				Time	Speedup
	R-1	R-2	R-L	ΔR		
Lead (75 characters)†	22.50	6.49	19.72	-8.34	–	–
Zajic et al. (2004)†	25.12	6.46	20.12	-5.35	–	–
Baziotis et al. (2019)†	22.13	6.18	19.30	-9.44	–	–
West et al. (2019)†	22.85	5.71	19.87	-8.62	–	–
Schumann et al. (2020)†	26.04	**8.06**	22.90	-0.05	–	–
Our replication	26.14	8.03	22.88	0	12.314	1x
Su et al. (2021)	26.25	7.66	22.83	-0.31	0.022	559x
NAUS (truncate)	26.52	7.88	22.91	0.26	**0.005**	**2463x**
NAUS (length control)	**26.71**	7.68	**23.06**	**0.40**	0.048	257x

Table 2: Results on the DUC2004 dataset. †Quoted from previous papers.

(2020), but trains a non-autoregressive model from search results. We consider two settings for controlling the summary length: truncating longer summaries and decoding with our proposed length-control algorithm. Both of our variants outperform Schumann et al. (2020) by 1.21–2.73 in terms of the total ROUGE score (Rows 5–6 & 13–14, Table 1). As mentioned, Schumann et al. (2020) only extracts original words with order preserved, yielding noisy sentences. Our NAUS, as a student, learns from the search-based teacher model and is able to smooth out its noise. This is a compelling result, as our student model outperforms its teacher.

Regarding inference efficiency, our NAUS method with truncating is more than 1300 times faster than Schumann et al. (2020), because we do not need iterative search. Even with dynamic programming and beam search for length control, NAUS is still over 100 times faster. This shows our NAUS is extremely efficient in inference, which is important for real-time applications.

Although the efficiency of Wang and Lee (2018) and Zhou and Rush (2019) is not available, we still expect our approach to be a few times faster (despite our higher ROUGE scores) because their models are autoregressive. By contrast, our NAUS is non-autoregressive, meaning that it predicts all words simultaneously. We will provide a controlled comparison between autoregressive and non-autoregressive models in Table 3.

Table 2 shows the results on the DUC2004 dataset. The cycle-consistency approach (Baziotis et al., 2019; West et al., 2019) does not perform well on this dataset, outperformed by an early rule-based syntax tree trimming approach (Zajic et al., 2004) and the state-of-the-art edit-based search (Schumann et al., 2020).

The performance of our NAUS model is consistent with Table 1, outperforming all previous methods in terms of the total ROUGE score, and being 100–1000 times faster than the search approach (Schumann et al., 2020).

In general, the proposed NAUS not only achieves state-of-the-art ROUGE scores for unsupervised summarization, but also is more efficient when deployed. Results are consistent on both datasets, demonstrating the generality of our NAUS.

In-Depth Analyses. We conduct in-depth analyses on the proposed NAUS model in Table 3. Due to the limit of time and space, we chose the Gigaword headline generation as our testbed. All the autoregressive (AR) and non-autoregressive (NAR) variants learn from the search output of our replication (Rows 2 & 11), where we achieve very close results to those reported in Schumann et al. (2020).

We first tried vanilla encoder–decoder NAR Transformer (Rows 4 & 13, Gu et al., 2018), where we set the number of decoding slots as the desired summary length and thus length-control is not needed. As seen, a vanilla NAR model does not perform well, and CTC largely outperforms vanilla NAR in both groups (Rows 5–6 & 14–15). Such results are highly consistent with the translation literature (Saharia et al., 2020; Chan et al., 2020; Gu and Kong, 2021; Qian et al., 2021).

The proposed encoder-only NAUS model outperforms encoder–decoder ones in both groups in terms of the total ROUGE score, when the summary length is controlled by either truncating or length-control decoding (Rows 8–9 & 17–18). Profoundly, our non-autoregressive NAUS is even better than the autoregressive Transformer (Rows 3 & 12) . We also experimented with previous non-autoregressive work for supervised summarization (Su et al., 2021)[3] in our learning-from-search setting. Although their approach appears to be encoder-only, it adds end-of-sequence (EOS) tokens at the end of the generation, and thus is unable to utilize the input–output correspondence. Their performance is higher than vanilla NAR models, but lower than ours. By contrast, NAUS is able to capture such correspondence with the residual connections, i.e., Eqns. (2) and (3), in its encoder-only architecture.

Generally, the efficiency of encoder-only NAR[4] (without length-control decoding) is ~2 times faster than encoder–decoder NAR and ~20 times faster than the AR Transformer.

Further, our length-control decoding improves the total ROUGE score, compared with truncating, for both encoder–decoder CTC and encoder-only NAUS models (Rows 6, 9, 15, & 18), although its dynamic programming is slower. Nevertheless, our non-autoregressive NAUS with length control is ~200 times faster than search and ~3 times faster than the AR Transformer.

#	Approach		ROUGE Recall				Speedup
			R-1	R-2	R-L	ΔR	
Group A (desired length 8)							
1	Search	Schumann et al.	26.32	9.63	24.19	0.18	–
2		Our replication	26.17	**9.69**	24.10	0	1x
3	AR	Transformer (T)	26.65	9.51	24.67	0.87	58x
4	NAR enc-dec	Vanilla	24.87	8.33	22.74	-4.02	571x
5		CTC (T)	27.30	9.20	24.96	1.5	571x
6		CTC (LC)	27.76	9.13	25.33	2.26	149x
7	NAR enc-only	Su et al. (2021)	26.88	9.37	24.54	0.83	403x
8		Our NAUS (T)	27.27	9.49	24.96	1.76	**1396x**
9		Our NAUS (LC)	**27.94**	9.24	**25.51**	2.73	167x
Group B (desired length 10)							
10	Search	Schumann et al.	27.52	**10.27**	24.91	0.23	–
11		Our replication	27.35	10.25	24.87	0	1x
12	AR	Transformer (T)	27.06	9.63	24.55	-1.23	66x
13	NAR enc-dec	Vanilla	25.77	8.69	23.52	-4.49	709x
14		CTC (T)	28.14	10.07	25.37	1.11	709x
15		CTC (LC)	28.45	9.81	25.63	1.42	192x
16	NAR enc-only	Su et al. (2021)	27.86	9.88	25.51	0.78	461x
17		Our NAUS (T)	28.24	10.04	25.40	1.21	**1843x**
18		Our NAUS (LC)	**28.55**	9.97	**25.78**	**1.83**	210x

Table 3: Model analysis on headline generation. **AR:** Autoregressive models. **NAR enc-dec:** Non-autoregressive encoder–decoder. **NAR enc-only:** Non-autoregressive encoder-only. **T:** Truncating. **LC:** Length control. All AR and NAR models use the Transformer architecture.

	Decoding	Wins	Ties	Loses	p-val
Overall quality	Truncate	18.6%	40.6%	40.6%	0.0004
	Length control	**40.6%**	40.6%	**18.6%**	
Completeness & fluency	Truncate	24.6%	26.6%	48.6%	0.0005
	Length control	**48.6%**	26.6%	**24.6%**	

Table 4: Human evaluation comparing truncating and length control for our NAUS model on 50 samples in the Gigaword headline generation task. The results are statistically significant, where the p-value is given by a one-sided binomial test.

Human Evaluation. We also conducted human evaluation with a focus on truncating and length-control decodings. This is because truncating may generate incomplete sentences, which cannot be adequately evaluated by automatic metrics as their ROUGE scores are close.

Specifically, we invited three human annotators to compare the two decoding algorithms for NAUS on 50 randomly selected samples, in the setting of Group B, Table 1 (Gigaword headline generation with a target length of 10). The annotation was conducted in a pairwise manner in terms of overall quality and fluency/completeness; average results (wins/loses/ties) are shown in Table 5. It should be mentioned that our annotation was strictly blind: the samples of two systems were presented in random order and annotators did not know which system generated a sample.

[3] To the best of our knowledge, the other two non-autoregressive supervised summarization models are Yang et al. (2021) and Qi et al. (2021). Their code and pretrained models are not available, making replication difficult.

[4] The standard minimal encoder–decoder NAR model has 6 layers for the encoder and another 6 layers for the decoder (Vaswani et al., 2017). Our NAUS only has a 6-layer encoder. Our pilot study shows that more layers do not further improve performance in our encoder-only architecture.

As seen, our length-control decoding algorithm largely outperforms the truncating approach in terms of both the overall quality and fluency/completeness. The results are statistically significant (p-values<0.01) in a one-sided binomial test. This verifies that length-control decoding is important for summarization, as truncating yields incomplete sentences, which are reflected by ROUGE scores.

Additional results. We analyze the beam search in length-control decoding in Appendix C and present a case study in Appendix D. We also show length-transfer performance in Appendix E.

4 Related Work

Summarization systems can be generally categorized into two paradigms: extractive and abstractive. Extractive systems extract certain sentences and clauses from input, for example, based on salient features (Zhou and Rush, 2019) or feature construction (He et al., 2012). Abstraction systems generate new utterances as the summary, e.g., by sequence-to-sequence models trained in a supervised way (Liu et al., 2021; Zhang et al., 2020).

Recently, unsupervised abstractive summarization is attracting increasing attention. For example, Yang et al. (2020) propose to use the Lead baseline (first several sentences) as the pseudo-groundtruth. However, such an approach only works with well-structured articles (such as CNN/DailyMail). Wang and Lee (2018) and Baziotis et al. (2019) use cycle consistency for unsupervised summarization. Zhou and Rush (2019) propose a step-by-step decomposable scoring function and perform beam search for generate summarization. Schumann et al. (2020) propose an edit-based local search approach, which allows a more comprehensive scoring function and outperforms cycle consistency and beam search.

Our paper follows Schumann et al. (2020) but trains a machine learning model to improve efficiency and smooth out search noise. Previously, Liu et al. (2020) fine-tune a GPT-2 model based on search results for unsupervised paraphrasing. We extend previous work in a non-trivial way by designing a non-autoregressive generator and further proposing a length-control decoding algorithm.

Non-autoregressive generation is originally proposed for machine translation (Gu et al., 2018). Recently, Jia et al. (2021) apply non-autoregressive models to extractive document-level summarization. Su et al. (2021) stack a non-autoregressive BERT model with a conditional random field (CRF) for abstractive summarization; since the summary is shorter than the input text, their approach puts multiple end-to-sequence (EOS) tokens at the end of the sentence, and thus is unable to utilize the strong input–output correspondence in the summarization task. Yang et al. (2021) apply auxiliary part-of-speech (POS) loss and Qi et al. (2021) explore pretraining strategies for encoder–decoder non-autoregressive summarization; their length is given by POS tag/EOS predictions. All these studies concern supervised summarization, and none can explicitly control the output length. By contrast, our paper focuses on unsupervised summarization. We adopt CTC training in our encoder-only architecture, allowing blank tokens to better align input and output words, which is more appropriate for summarization. We further propose a dynamic programming algorithm to control the summary length.

5 Conclusion

In this work, we propose a non-autoregressive unsupervised summarization model (NAUS), where we further propose a length-control decoding algorithm based on dynamic programming. Experiments show that NAUS not only archives state-of-the-art unsupervised performance on Gigaword headline generation and DUC2004 datasets, but also is much more efficient than search methods and autoregressive models. Appendices present additional analyses and length-transfer experiments.

Limitation and Future Work. Our paper focuses on unsupervised summarization due to the importance of low-data applications. One limitation is that we have not obtained rigorous empirical results for supervised summarization, where the developed model may also work. This is because previous supervised summarization papers lack explicitly categorization of summary lengths (Yang et al., 2020; Qi et al., 2021), making comparisons unfair and problematic (Schumann et al., 2020). This is also evidenced by Su et al. (2021), where the same model may differ by a few ROUGE points when generating summaries of different lengths. Nevertheless, we have compared with Su et al. (2021) in our setting and show the superiority of the NAUS under fair comparison. We plan to explore supervised summarization in future work after we establish a rigorous experimental setup, which is beyond the scope of this paper.

References

Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. 2021. Muppet: Massive multi-task representations with pre-finetuning. *arXiv preprint arXiv:2101.11038*.

Armen Aghajanyan, Akshat Shrivastava, Anchit Gupta, Naman Goyal, Luke Zettlemoyer, and Sonal Gupta. 2020. Better fine-tuning by reducing representational collapse. In *International Conference on Learning Representations*.

Christos Baziotis, Ion Androutsopoulos, Ioannis Konstas, and Alexandros Potamianos. 2019. Seq3: Differentiable sequence-to-sequence-to-sequence autoencoder for unsupervised abstractive sentence compression. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 673–681.

William Chan, Chitwan Saharia, Geoffrey Hinton, Mohammad Norouzi, and Navdeep Jaitly. 2020. Imputer: Sequence modelling via imputation and dynamic programming. In *Proceedings of the International Conference on Machine Learning*, pages 1403–1413.

Bonnie Dorr, David Zajic, and Richard Schwartz. 2003. Hedge trimmer: A parse-and-trim approach to headline generation. In *Proceedings of the HLT-NAACL 03 Text Summarization Workshop*, pages 1–8.

Thibault Févry and Jason Phang. 2018. Unsupervised sentence compression using denoising autoencoders. In *Proceedings of the Conference on Computational Natural Language Learning*, pages 413–422.

David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2003. English Gigaword. *Linguistic Data Consortium, Philadelphia*.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*, page 369–376.

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

Jiatao Gu and Xiang Kong. 2021. Fully non-autoregressive neural machine translation: tricks of the trade. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 120–133.

Zhanying He, Chun Chen, Jiajun Bu, Can Wang, Lijun Zhang, Deng Cai, and Xiaofei He. 2012. Document summarization based on data reconstruction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 620–626.

Ruipeng Jia, Yanan Cao, Haichao Shi, Fang Fang, Pengfei Yin, and Shi Wang. 2021. Flexible non-autoregressive extractive summarization with threshold: How to extract a non-fixed number of summary sentences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 13134–13142.

Julia Kreutzer, Stefan Riezler, and Carolin Lawrence. 2021. Offline reinforcement learning from human feedback in real-world sequence-to-sequence tasks. In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 37–43.

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

Jingjing Li, Zichao Li, Lili Mou, Xin Jiang, Michael Lyu, and Irwin King. 2020. Unsupervised text generation by learning from search. In *Advances in Neural Information Processing Systems*, pages 10820–10831.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81.

Xianggen Liu, Lili Mou, Fandong Meng, Hao Zhou, Jie Zhou, and Sen Song. 2020. Unsupervised paraphrasing by simulated annealing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 302–312.

Yixin Liu, Zi-Yi Dou, and Pengfei Liu. 2021. RefSum: Refactoring neural summarization. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1437–1448.

Clara Meister, Ryan Cotterell, and Tim Vieira. 2020. If beam search is the answer, what was the question? In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2173–2185.

Yishu Miao and Phil Blunsom. 2016. Language as a latent variable: Discrete generative models for sentence compression. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 319–328.

Ani Nenkova, Sameer Maskey, and Yang Liu. 2011. Automatic summarization. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1–86.

Paul Over and James Yen. 2004. An introduction to DUC-2004: Intrinsic evaluation of generic news text summarization systems. In *Proceedings of the Document Understanding Conference*.

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

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

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

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

Raphael Schumann, Lili Mou, Yao Lu, Olga Vechtomova, and Katja Markert. 2020. Discrete optimization for unsupervised sentence summarization with word-level extraction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 5032–5042.

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

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.

Yaushian Wang and Hung-Yi Lee. 2018. Learning to encode text as human-readable summaries using generative adversarial networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 4187–4195.

Peter West, Ari Holtzman, Jan Buys, and Yejin Choi. 2019. BottleSum: Unsupervised and self-supervised sentence summarization using the information bottleneck principle. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, pages 3752–3761.

Kexin Yang, Wenqiang Lei, Dayiheng Liu, Weizhen Qi, and Jiancheng Lv. 2021. POS-constrained parallel decoding for non-autoregressive generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*, pages 5990–6000.

Ziyi Yang, Chenguang Zhu, Robert Gmyr, Michael Zeng, Xuedong Huang, and Eric Darve. 2020. TED: A pretrained unsupervised summarization model with theme modeling and denoising. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1865–1874.

David Zajic, Bonnie Dorr, and Richard Schwartz. 2004. BBN/UMD at DUC-2004: Topiary. In *Proceedings of the HLT-NAACL Document Understanding Workshop*, pages 112–119.

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

Jiawei Zhou and Alexander Rush. 2019. Simple unsupervised summarization by contextual matching. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 5101–5106.

A Proof of Theorem 1

Theorem 1. *(1) If repeating tokens are not merged, then the proposed length-control algorithm with beam size $B = 1$ finds the exact optimum $\mathscr{B}_{S,T}$ being the most probable length-T sentence given by S prediction slots. (2) If we merge repeating tokens predicted by CTC-trained models, the above algorithm may not be exact.*

Proof. [Part (1)] This part concerns a variant of our decoding algorithm, which only removes the blank token ϵ but does not merge consecutive repeated tokens to a single word, i.e., Eqn. (10) is removed. We denote this by Γ', for example, $\Gamma'(a\epsilon\epsilon aabb\epsilon) = aaabb$, as opposed to $\Gamma(a\epsilon\epsilon aabb\epsilon) = aabb$ in our algorithm. We now show that, based on Γ', our dynamic programming algorithm in §2.3 with beam size $B = 1$ is an exact inference algorithm.

We define $\beta_{s,t} = \max_{\mathbf{b}:|\mathbf{b}|=s,|\Gamma'(\mathbf{b})|=t} P(\mathbf{b}|\mathbf{x})$, where $|\cdot|$ denotes the length of a sequence. In other words, $\beta_{s,t}$ is the maximum probability of s tokens that are reduced to t words.

According to the definition, we have

$$\beta_{1,0} = P\left(\mathrm{w}_1 = \epsilon|\mathbf{x}\right) \tag{13}$$

$$\beta_{1,1} = \max\nolimits_{\mathrm{w}_1 \neq \epsilon} P\left(\mathrm{w}_1|\mathbf{x}\right) \tag{14}$$

$$\beta_{s,t} = 0 \quad \text{for } s > t \tag{15}$$

In (13), $\beta_{1,0}$ refers to the probability of one token that is reduced to zero words, in which case, the first predicted token can only be the blank token ϵ, corresponding to Eqn. (9) with $s = 1$ and $t = 0$. Likewise, $\beta_{1,1}$ is the maximum probability of one token that is reduced to one word. Thus, it is the probability of the most probable non-ϵ token, corresponding to Eqn. (11) with $s = 1$ and $t = 0$. Eqn. (15) asserts that fewer tokens cannot be reduced to more words; it is used for mathematical derivations, but need not to be explicitly implemented in our algorithm in §2.3.

The recursion variable $\beta_{s,t}$ is computed by

$$\beta_{s,t} = \max\Big\{\beta_{s-1,t} \cdot P(\mathrm{w}_s = \epsilon|\mathbf{x}), \\ \beta_{s-1,t-1} \cdot \max\nolimits_{\mathrm{w}_s \neq \epsilon} P(\mathrm{w}_s|\mathbf{x})\Big\} \tag{16}$$

In other words, the variable $\beta_{s,t}$ can inherit $\beta_{s-1,t}$ with a predicted blank token ϵ, corresponding to Eqn. (9); or it can inherit $\beta_{s-1,t-1}$ with a predicted non-ϵ token, corresponding to Eqn. (11). Specially, if $t = 0$, then the second term has $\beta_{s-1,-1}$ undefined, and thus is ignored in the $\max$ operation.

Word	$P(\mathrm{w}_1|\mathbf{x})$	$P(\mathrm{w}_2|\mathbf{x})$
I	0.39	0.1
like	0.4	0.9
coding	0.1	0
ϵ	0.11	0

Table 5: An example of predicted probabilities of two generation slots, where we have a vocabulary of three words and a blank token ϵ.

We need the $\max$ operator to take the higher probability in the two cases, since $\beta_{s,t}$ is the maximum probability of s tokens being reduced to t words. This corresponds to Eqn. (12) with beam size $B = 1$.

To sum up, our inductive calculation guarantees that $\beta_{S,T}$ is the exact maximum probability of $\max_{\mathbf{b}:|\mathbf{b}|=S,|\Gamma'(\mathbf{b})|=T} P(\mathbf{b}|\mathbf{x})$ for the desired length T with S generation slots; our algorithm (if not merging repeating tokens) gives the corresponding $\mathscr{B}_{S,T}$ as $\operatorname{argmax} P(\mathbf{b}|\mathbf{x})$ under the same constraints, concluding the proof of Part (1).

[Part (2)] CTC training merges consecutive repeated tokens to a single word, unless separated by the blank token ϵ (Graves et al., 2006). Since our model is trained by CTC, we should adopt this rule in inference as well. We show in this part that our algorithm, with beam size $B = 1$, does not yield the exact optimum with an example in Table 5.

We consider generating a sentence of two words from the two prediction slots, i.e., $S = T = 2$. Apparently, the optimal sequence is "I like" with probability $0.39 \cdot 0.9 = 0.351$. However, the algorithm would predict $\mathscr{B}_{1,1} = \{\text{“like”}\}$ because "like" is the most probably token in the first slot. Then, our algorithm will give $\mathscr{B}_{2,2} = \{\text{“like I”}\}$, because it has to select a non-repeating token based on Γ, yielding a non-optimal solution.

□

It is noted that, if we do not merge repeating tokens as in Γ', our algorithm will give the exact optimum "like like" in the above example. This shows that merging consecutive repeated tokens requires the decoding algorithm to correct early predictions, and thus, our dynamic programming becomes an approximate inference. Nevertheless, our algorithm is able to generate a sequence of the desired length properly; its approximation happens only when the algorithm compares more repetitions with fewer ϵs versus more ϵs with fewer repetitions. Such approximation is further alleviated by beam search in our dynamic programming. Therefore, the proposed length-control algorithm is

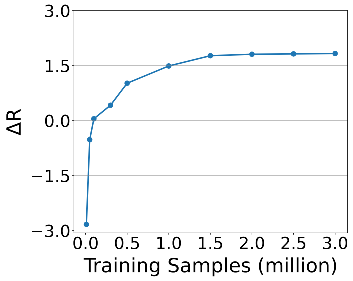

Figure 3: Performance versus the number of training samples in the setting of Group B, Table 1. Notice that NAUS is trained by pseudo-groundtruth given by unsupervised edit-based search (Schumann et al., 2020). Thus, our approach is indeed unsupervised.

better than truncating a longer sentence; especially, our approach generates more fluent and complete sentences.

B Implementation Details

Our NAUS had a Transformer encoder as the basic structure, generally following the settings in Vaswani et al. (2017): 6 encoder layers, each having 8 attention heads. The dimension was 512 for attention and 2048 for feed-forward modules.

Our training used a batch size of 4K tokens, with a maximum of 200K updates. We used Adam with $\beta = (0.9, 0.98)$. In general, the learning rate warmed up to 5e-4 in the first 10K steps, and then decayed to 1e-9 with the inverse square-root schedule, except that we find the maximum learning rate of 1e-4 worked better for headline generation with the summary length of 8. We set the ℓ_2 weight decay to 0.01. Our length-control decoding algorithm had a beam size of 6. More details can be found in our repository (Footnote 1).

Our NAUS training is based on Schumann et al. (2020)'s prediction on the input of the Gigaword headline generation training set. We show performance against the number of training samples in Figure 3. As seen, NAUS outperforms its search teacher even with a small set of 0.1 million samples. The performance saturates as the number of samples increases. Based on this analysis, we used 3 million samples from the 3.8 million Gigaword training set to train our NAUS models.

C Analysis of Beam Search

As mentioned, our length-control decoding algorithm involves beam search within its dynamic programming, because the algorithm does not find the exact optimum when it merges repeating words.

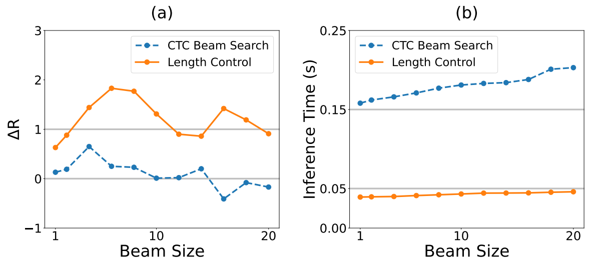

Figure 4: Performance of our NAUS approach when equipped with the length-control decoding and the truncated CTC beam search on the Gigaward headline generation test set. The chosen token at each slot is required to be 10-most probable.

We analyze the effect of the beam size in our length-control algorithm.

In addition, we compare our approach with CTC beam search (Graves et al., 2006).[5] Typically, a CTC-trained non-autoregressive model can be decoded either greedily or by beam search. The greedy decoding finds the most probable token at each step, i.e., $\mathrm{w}_i^* = \operatorname{argmax}_{\mathrm{w}_i} P(\mathrm{w}_i|\mathbf{x})$, and reduces the tokens to a sentence by $\Gamma(\mathrm{w}_1, \cdots, \mathrm{w}_T)$, where T is the number of decoding steps. The CTC beam search algorithm searches for the most likely sentence by marginalizing all token sequences that are reduced to $\mathbf{y}$, i.e., $\operatorname{argmax}_{\mathbf{y}} \sum_{\mathbf{w}:\Gamma(\mathbf{w})=\mathbf{y}} P(\mathbf{w}|\mathbf{x})$.

We show results in Figure 4, where we chose 10-word Gigaword headline generation as the testbed with our NAUS model (Group B, Table 1). Notice that CTC beam search does not control the output length, and for fair comparison, we truncated its generated summaries. This also shows that our novel decoding approach and CTC beam search are distinct algorithms.

As seen in Figure 4a, the beam search does play a role in our length-control algorithm. When the beam enlarges from 1 to 6, the performance (orange solid line) increases by 1.2 points in ΔR, the difference of total ROUGE in comparison with Schumann et al. (2020) under our replication (Row 10, Table 1). However, further increasing the beam size does not yield additional performance gain. This is consistent with previous literature in autoregressive generation (Meister et al., 2020), which also suggests a beam size of 5–7 is the best in their applications. In terms of the efficiency (Figure 4b), a larger beam size monotonically increases the inference time. However, the overhead of beam

[5] Our implementation of CTC beam search is based on https://github.com/parlance/ctcdecode

Input: the united nations condemned saturday an attack on russian embassy employees in baghdad that claimed the life of one russian and resulted in the kidnapping of four others
Reference: un condemns murder of russians in iraq with annan comment
Schumann et al. (2020): attack on russian embassy in baghdad claimed one in four
NAUS (truncate): an attack on russian embassy employees in baghdad claimed in kidnapping of four others
NAUS (length control): united nations condemned attack on russian embassy employees in baghdad

Table 6: Example summaries for Gigaword headline generation. The gray words are truncated for fair comparison.

search is relatively small in our dynamic programming, and thus we chose a beam size of 6 in our experiments.

Our length-control algorithm significantly outperforms CTC beam search (dashed blue lines) in terms of both ΔR and efficiency. Especially, CTC beam search is three times slower, and degrades more significantly than our length-control decoding when the beam size increases.

D Case Study

We show in Table 6 example summaries generated by our NAUS with truncating and length-control decoding, as well as the previous state-of-the-art method (Schumann et al., 2020). We observe that NAUS without length control generates slightly longer summaries, and if truncated, the output may be incomplete; by contrast, our length-control algorithm can generate a fluent and complete sentence of the desired length by dynamic programming. Compared with Schumann et al. (2020), our NAUS (length control) generates a more informative summary that includes the main clause (*united nations condemned*), which also appears in the reference summary.

E Length-Transfer Summary Generation

In the main paper, we present results where our NAUS is trained on search outputs (Schumann et al., 2020), which have the same length as the inference target. This follows the common assumption in machine learning that training and test samples are independently identically distributed.

In this appendix, we show the performance of length-transfer summary generation, where the prediction has a different length from that of training. We denote such a model by NAUS$_{i \to j}$, referring to training with i words and testing for j words.

As seen in Groups A & B in Table 7, NAUS with length transfer is slightly worse than NAUS trained on the correct length, which is understandable. Nevertheless, length-transfer decoding still outperforms the search teacher and other baselines.

Moreover, we consider the third setting in Schumann et al. (2020), where the target length is 50% of the input. Since it takes time to obtain pseudo-groundtruths given by the edit-based search, we would directly transfer already trained NAUS models to this setting by our length-control decoding. Results are shown in Group C, Table 7. We observe NASU$_{10 \to 50\%}$ is better than NASU$_{8 \to 50\%}$, which makes much sense because the latter has a larger gap during transfer. Remarkably, both NASU$_{8 \to 50\%}$ and NASU$_{10 \to 50\%}$ outperform Schumann et al. (2020) and other baselines, achieving new state-of-the-art unsupervised performance on this setting as well.

We further compare with Su et al. (2021), who use a length penalty to encourage short summaries. However, their length control works in the statistical sense but may fail for individual samples. Moreover, such a soft length penalty cannot generate longer summaries than trained. Even in the setting of 10 → 8, their generates summaries are slightly longer than required, while the performance degrades much faster than NAUS.

These results show that our novel length-control decoding algorithm is not only effective when generating summaries of similar length to the training targets, but also generalizes well to different desired summary lengths without re-training. In general, our NAUS is an effective and efficient unsupervised summarization system with the ability of explicit length control.

Group	#	Approach		Len	ROUGE F1				Inf.Time	Speedup
					R-1	R-2	R-L	ΔR		
Group A (desired length 8)	1	Baseline	Lead (8 words)†	7.9	21.39	7.42	20.03	-11.12	–	–
	2	Search	Schumann et al. (2020)†	7.9	26.32	9.63	24.19	0.18	–	–
	3		Our replication	7.9	26.17	**9.69**	24.10	0	6.846	1x
	4	Learn from search	Su et al. (2021)$_{8\to8}$	7.7	26.88	9.37	24.54	0.83	0.017	403x
	5		Su et al. (2021)$_{10\to8}$	8.4	25.71	8.94	23.65	-1.84	0.018	380x
	6		NAUS (truncate)	7.8	27.27	9.49	24.96	1.76	**0.005**	**1369x**
	7		NAUS$_{8\to8}$	7.8	**27.94**	9.24	**25.50**	**2.73**	0.041	167x
	8		NAUS$_{10\to8}$	7.9	27.12	9.08	24.86	1.10		
Group B (desired length 10)	9	Baseline	Lead (10 words)†	9.8	23.03	7.95	21.29	-10.2	–	–
	10		Wang and Lee (2018)†	10.8	27.29	10.01	24.59	-0.58	–	–
	11		Zhou and Rush (2019)†	9.3	26.48	10.05	24.41	-1.53	–	–
	12	Search	Schumann et al. (2020)†	9.8	27.52	**10.27**	24.91	0.23	–	–
	13		Our replication	9.8	27.35	10.25	24.87	0	9.217	1x
	14	Learn from search	Su et al. (2021)$_{8\to10}$	–	–	–	–	–	–	–
	15		Su et al. (2021)$_{10\to10}$	9.4	27.86	9.88	25.51	0.78	0.020	461x
	16		NAUS (truncate)	9.8	28.24	10.04	25.40	1.21	**0.005**	**1843x**
	17		NAUS$_{8\to10}$	9.9	28.32	9.58	25.46	0.89	0.044	210x
	18		NAUS$_{10\to10}$	9.8	**28.55**	9.97	**25.78**	**1.83**		
Group C (desired length 50% of the input)	19	Baseline	Lead (50% words)†	14.6	24.97	8.65	22.43	-4.58	–	–
	20		Févry and Phang (2018)†	14.8	23.16	5.93	20.11	-11.43	–	–
	21		Baziotis et al. (2019)†	15.1	24.70	7.97	22.41	-5.55	–	–
	22	Search	Schumann et al. (2020)†	14.9	27.05	9.75	23.89	0.06	–	–
	23		Our replication	14.9	27.03	9.81	23.79	0	17.462	1x
	24	Learn from search	Su et al. (2021)$_{8\to50\%}$	–	–	–	–	–	–	–
	25		Su et al. (2021)$_{10\to50\%}$	–	–	–	–	–	–	–
	26		NAUS$_{8\to50\%}$	14.9	28.39	9.78	24.94	2.48	**0.052**	**336x**
	27		NAUS$_{10\to50\%}$	14.9	**28.53**	**9.88**	**25.10**	**2.88**		

Table 7: Analysis of length-transfer summary generation. A subscript $i \to j$ (or $j\%$) refers to a model trained with i words and tested for j (or $j\%$) words. **Len:** Average length of predicted summaries. **R-1, R-2, R-L:** ROUGE-1, ROUGE-2, ROUGE-L. Δ**R:** The difference of total ROUGE (sum of R-1, R-2, and R-L) in comparison with the (previous) state-of-the-art model (Schumann et al., 2020) under replication. **Inf.Time:** Average inference time in seconds for one sample on an i9-9940X CPU and a RTX6000 GPU. **Speedup:** Relative to Schumann et al. (2020). †Results quoted from previous papers; others are given by our experiments. Su et al. (2021)'s approach has a soft length penalty to encourage short output, but cannot generate longer summaries than trained.