COPY IS ALL YOU NEED

Anonymous authors
Paper under double-blind review

ABSTRACT

The dominant text generation models compose output by selecting words in a fixed vocabulary. In this paper, we formulate text generation as progressively copying text segments (e.g., words or phrases) from an existing text collection. We compute the contextualized representations of meaningful text segments and index them using efficient vector search toolkits. The task of text generation is then decomposed into a series of copy-and-paste operations: at each time step, we seek suitable text spans from existing articles in the text collection rather than selecting from a standalone vocabulary. Experiments on the standard language modeling benchmark (WikiText-103) show that our approach achieves better generation quality by coping from the original training data (0.758 vs. 0.691 MAUVE). We also show that our approach attains additional performance gains by simply scaling up to larger text collections without extra training. Furthermore, our approach allows for effective domain adaptation by simply switching to any domain-specific text collection, again without further training. Finally, we observe that our approach achieves better inference efficiency than standard token-level autoregressive models thanks to the reduction of decoding steps.

1 INTRODUCTION

Most neural language models (LMs) process text generation tasks by making a series of next-token predictions in an autoregressive manner (Radford et al., 2019; Dai et al., 2019; Khandelwal et al., 2020). Specifically, LMs generate the next-token distribution over a fixed vocabulary for any given prefix. Then, the next token is selected by a chosen decoding method, such as greedy search and nucleus sampling (Holtzman et al., 2020). This process continues until some stop condition is reached. For example, a special end-of-generation token is emitted, or the generated text reaches the maximum length limit.

Unlike traditional neural language models, we reformulate text generation as copying text segments from existing text collections. The text segments can be of variable lengths, including single words and multi-word phrases. For clarity, we will use the term "phrase" to refer to any contiguous text segments. Note that a single word can also be seen as a phrase of length 1. We compute a contextualized vector representation for each phrase and pack them into an offline index. At each decoding step, a suitable text segment from the offline index is retrieved and appended to the current prefix. In other words, the next-token predictions in traditional neural language models are replaced by a series of copy-and-paste operations.

Our proposed model, named COPY-GENERATOR, enjoys the following advantages. First, our method allows a sequence of multiple tokens to be generated in one single step. It could reduce the total number of decoding steps, leading to lower generation latency. Second, our method allows for training-free adaptation to new knowledge sources because the text collection can be updated in a plug-and-play fashion. It could be especially useful for application scenarios such as domain adaptation and data expansion/filtering. Third, our method selects phrases in specific contexts rather than standalone tokens in a fixed vocabulary. It potentially allows for more accurate candidate representation and selection.

We conduct extensive experiments to verify the advantages of our proposed COPY-GENERATOR. On the standard language modeling benchmark (WikiText-103), our proposed COPY-GENERATOR substantially outperforms standard baselines on automatic metrics (0.758 vs. 0.691 MAUVE (Pillutla et al., 2021)). Moreover, when we scale up the text collection of COPY-GENERATOR to a larger

one, the En-Wiki dataset, we obtain additional gain (0.806 vs. 0.650 MAUVE) without any further training. Furthermore, when we directly switch the text collection from the WikiText-103 corpus to a domain-specific corpus, Law-MT (Koehn & Knowles, 2017), our proposed COPY-GENERATOR outperforms strong baselines on this domain adaption setting (0.653 vs. 0.603 MAUVE), again without any domain-specific training. Our contributions can be summarized as follows:

- We propose COPY-GENERATOR, a new method that reformulates text generation tasks as a series of copy-and-paste operations from existing text collections.
- We show that COPY-GENERATOR can outperform standard neural language model baselines on existing language modeling benchmarks.
- We demonstrate that COPY-GENERATOR allows for training-free adaptations to larger text collections and domain-specific text collections.

2 BACKGROUND: NEURAL TEXT GENERATION

Neural text generation can be divided into two categories: (1) unconditional text generation; (2) conditional text generation. Unconditional text generation (or language modeling) aims to generate a coherent text continuation given a prefix. In this case, language models perform generation using a density estimation over sequences $p_\theta(x)$. Conditional text generation aims to generate text with some condition c and instead estimates the probability of $p_\theta(x|c)$. Its typical applications include machine translation (Sutskever et al., 2014; Bahdanau et al., 2015) and summarization (See et al., 2017). Throughout this paper, our discussion will be focused on unconditional text generation, however, our approach can be readily adapted to conditional text generation as well.

The canonical approach to language modeling factors the generation in an autoregressive left-to-right manner $p_\theta(x_{0:n}) = \prod_{i=1}^{n} p(x_i|x_{<i})$. In this case, text generation is reduced to the task of repeatedly predicting the next token conditioned on the partial sequence (i.e., prefix) generated so far $p(x_i|x_{<i})$. The model often consists of two parts: (1) a prefix encoder and (2) a set of token embeddings. The prefix encoder is often parameterized by the Transformer architecture (Vaswani et al., 2017), which transforms any prefix into a fixed-sized vector representation $h_i \in \mathbb{R}^d = \text{PrefixEncoder}(x_{<i})$. Then, the probability of the next token being w is calculated as

$$p_\theta(x_i = w|x_{<i}) = \frac{\exp(v_w \cdot h_i)}{\sum_{w \in V} \exp(v_w \cdot h_i)},$$

where v_w is the context-independent token embedding representing the token w, and V is the predefined vocabulary consisting of all possible tokens. Based on the chosen decoding method, such as greedy search and nucleus sampling (Holtzman et al., 2020), the next token is selected according to the probability distribution over the fixed vocabulary V. This process is repeated in an autoregressive manner, until some stop condition is reached, e.g., the maximum length of generation is reached.

3 COPY-GENERATOR

Unlike traditional language models that compute the next token distribution over a fixed vocabulary that is usually composed of words or sub-words (Sennrich et al., 2016; Kudo & Richardson, 2018), our proposed COPY-GENERATOR has a dynamic "vocabulary" that is dependent on the available source text collections. Each item in the "vocabulary" corresponds to a text segment (termed as *phrase* in this paper) in the source text collection. Importantly, all phrases are context-sensitive. That is, the same phrases in different contexts are considered to be different. The overall framework is depicted in Figure 1.

Formally, our approach assumes a set of source documents $\{D^1, \ldots, D^m\}$ is available. For each document D^i, a phrase $k = D^i_{s:e}$ of length $e - s + 1$ can be extracted, where s and e mark the start and end positions of the phrase in the document, respectively. We denote all the e all the phrases in the source text collection as $\mathcal{P}$. For a given prefix $x_{<i}$, we aim to select the best phrases that can form a coherent text continuation following the prefix. To this end, we compute a contextualized representation for each phrase $p_k \in \mathbb{R}^d = \text{PhraseEncoder}(s, e, D^i)$ using a phrase encoder. Thus, a phrase table $\{(k, p_k)|k \in \mathcal{P}\}$ can be constructed. Similar to traditional language models, at test time,

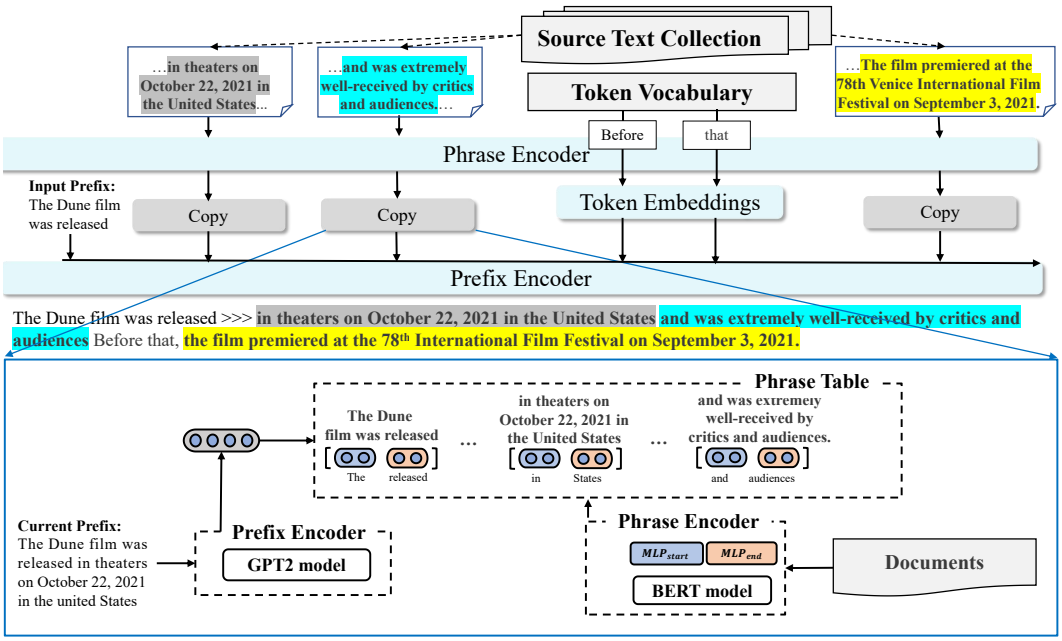

Figure 1: The overview of our proposed COPY-GENERATOR. Given the prefix *The Dune film was released*, COPY-GENERATOR retrieve 3 phrases (in different colors) from the documents and generates 2 tokens (*Before* and *that*) from the fixed vocabulary to form the whole generation.

COPY-GENERATOR also employs a prefix encoder to map the prefix $x_{<i}$ into a vector representation h_i. The fitness of a phrase k to the prefix $x_{<i}$ is then measured by the dot product of their vector representations p_k and h_i:

$$p(k|x_{<i}) \propto \exp(p_k \cdot h_i). \tag{1}$$

At each time step, a suitable phrase is selected and appended to the current prefix accordingly.

Note that the size of the phrase table can be up to billions. To search over this large candidate pool, we pre-compute the phrase representations and use a coarse-to-fine search pipeline. The details are deferred to Section 4.2. Moreover, to support the scenarios where no suitable phrases are available, we also add the context-independent token embeddings $\{(w, v_w)|w \in V\}$ in standard LMs to the phrase table.

Ethical Consideration The text generated by COPY-GENERATOR contains text segments copied from other documents, which may cause copyright disputes in real-world applications. Therefore, there are a few things that need to be considered: (1) The copyright of the source text documents needs to be carefully checked, one should not use documents with strict copyright protection and/or private information; (2) It is recommended to quote the original source explicitly, especially when the retrieved phrases are rather long.

3.1 MODEL ARCHITECTURE

As illustrated in Figure 1, our proposed model consists of three major components: (1) a *prefix encoder* that maps prefixes to fixed-sized representations; (2) a context-dependent *phrase encoder* that computes the vector representations of the phrases in the source text collection; (3) a set of context-independent *token embeddings* similar to the one used in standard neural language models.

Prefix Encoder The prefix encoder is responsible for encoding the prefix $x_{<i}$ into a vector representation for the next-phrase prediction. We treat the prefix as a sequence of tokens (previously predicted phrases are split into tokens as well) and model them using the standard Transformer architecture with causal attention (Vaswani et al., 2017; Radford et al., 2019). Causal attention only allows each position in the input sequence to attend to its preceding positions. Therefore, the prefix

representation can be computed incrementally as the generation progresses, leading to faster inference. Concretely, the prefix encoder transforms a prefix $x_{<i}$ of length i into a matrix $\mathcal{H}_i \in \mathbb{R}^{i \times dL}$, where d is the hidden dimension and L is the number of Transformer layers. The computation can be written as:

$$\mathcal{H}_i = \text{PrefixEncoder}(x_i, \mathcal{H}_{i-1}).$$

We use the hidden state of the last token in the last layer as the prefix representation.

Phrase Encoder Given a set of source documents $\{D^1, ..., D^m\}$, the phrase encoder computes the vector representations of all the phrases in the documents. Inspired by previous work (Lee et al., 2016; Seo et al., 2018; Lee et al., 2021), we construct context-dependent phrase representations as follows. For a document $D = D_1, \ldots, D_m$ of length m, we first apply a deep bidirectional Transformer (Devlin et al., 2019) to obtain contextualized token representations $\mathcal{D} \in \mathbb{R}^{m \times d_t}$, where d_t is the dimension of token representations. Then, we apply two MLPs models, $\text{MLP}_{\texttt{start}}$ and $\text{MLP}_{\texttt{end}}$, to convert $\mathcal{D}$ into start and end token representations $\mathcal{D}_{\texttt{start}}, \mathcal{D}_{\texttt{end}} \in \mathbb{R}^{m \times \frac{d}{2}}$, respectively:

$$\mathcal{D}_{\texttt{start}} = \text{MLP}_{\texttt{start}}(\mathcal{D}), \mathcal{D}_{\texttt{end}} = \text{MLP}_{\texttt{end}}(\mathcal{D}).$$

For each phrase $D_{s:e}$ that starts at s and ends at e in the document, we use the concatenation of the corresponding start and end vectors starts as the phrase representation.

$$\text{PhraseEncoder}(s, e, D) = [D_{\texttt{start}}[s]; D_{\texttt{end}}[e]] \in \mathbb{R}^d$$

The advantages of the above representation method are that (1) we only need to encode the document once to obtain all phrase representations; and (2) we only need to store all the token representations instead of all phrase representations.

Context-Independent Token Embeddings Although COPY-GENERATOR can copy phrases from other documents, we would like to retain the generalization capability to compose output with standalone tokens. This can be especially useful when there is no suitable phrase in the source text collection. Therefore, we also add the traditional context-independent token embeddings $\mathcal{M} \in \mathbb{R}^{|V| \times d}$ to our phrase table. The tokens in the fixed vocabulary V can be seen as phrases of length 1 without any context information.

3.2 MODEL TRAINING

COPY-GENERATOR decomposes the task of text generation into a series of copy-and-paste operations: at each time step, it selects the next phrase either from the source text collection or the fixed token vocabulary. In other words, phrases are used as the basic building blocks for text generation. To train COPY-GENERATOR, each document in the training set is chunked into a sequence of phrases in a similar spirit. Specifically, we propose a greedy segmentation algorithm based on forward maximum matching. Taking a document $D = D_1, \ldots, D_m$ of m tokens as an example, our algorithm segments the document from left to right. The first i tokens will be cut out as a phrase if it can be found as a sub-sequence in other documents and i is the maximum valid value. The above process is repeated until all tokens are cut out. Note that some resultant phrases can be single tokens in the fixed token vocabulary when no proper matching can be found. Detailed explanations of the phrase segmentation algorithm can be found in Appendix A.

Suppose that a document D has been split into n phrases $D = p_1, \ldots, p_n$. If the k-th phrase p_k is copied from another document, the phrase encoder is used to extract its context-dependent phrase representations. On the other hand, we directly retrieve the context-independent token embedding of p_k if it is copied from the fixed token vocabulary. As illustrated by Eq. 1, COPY-GENERATOR relies on a shared vector space of prefix and phrase representations, where the representations of semantically coherent prefixes and phrases should be closer to each other while others should be pushed apart. We define the training loss for next-phrase predictions using the InfoNCE loss with in-batch negatives (Karpukhin et al., 2020):

$$\mathcal{L}_p = -\frac{1}{n} \sum_{k=1}^{n} \log \frac{\exp(q_k \cdot p_k)}{\sum_{p \in P} \exp(q_k \cdot p_p) + \sum_{w \in V} \exp(q_k \cdot v_w)}$$

where P is the set of all phrases in the source documents, V is the set of all tokens in the token vocabulary, and q_k denotes the representation of the prefix preceding the phrase p_k.

Additionally, to retrain the capability of token-level generation, we also train COPY-GENERATOR with the standard token-level autoregressive loss.

$$\mathcal{L}_t = -\frac{1}{m}\sum_{i=1}^{m}\log\frac{\exp(q_i, v_{D_i})}{\sum_{w\in V}\exp(q_i, v_w)}$$

where q_i denotes the representation of the prefix preceding the token D_i. Finally, the training loss is the sum of these two losses:

$$\mathcal{L} = \mathcal{L}_p + \mathcal{L}_t$$

4 EXPERIMENTAL SETUP

4.1 BASELINES

We compare COPY-GENERATOR with the following two baselines:

- **Vanilla Transformer** (Vaswani et al., 2017) has been the *de facto* model for neural language models. Concretely, we fine-tune the pre-trained GPT2 model (Radford et al., 2019) in our experiments.
- k**NN-LM** (Khandelwal et al., 2020) is a retrieval-augmented generation model, which extends a pre-trained neural language model by linearly interpolating its next token distribution with a k-nearest neighbors (kNN) model.

Following conventions (Welleck et al., 2020; Su et al., 2022), we use greedy search and nucleus sampling (Holtzman et al., 2020) for generating the text continuations throughout our experiments.[1] The best parameters of the decoding methods are chosen by grid search on the MAUVE scores.

4.2 IMPLEMENTATION DETAILS

Our source codes are based on the popular HuggingFace package.[2] For a fair comparison, the prefix encoders in vanilla Transformer, kNN-LM, and COPY-GENERATOR use the same model architecture as the pre-trained GPT2 model (Radford et al., 2019). However, in our preliminary experiments, we found that the sub-word vocabulary (Kudo & Richardson, 2018) of the pre-trained GPT2 does not suit work well with the kNN-LM framework. Thus, we train a new GPT2 model (12 layers, 12 heads, and 768 hidden dimensions) from scratch with the word-level vocabulary. For the phrase encoder in COPY-GENERATOR, we fine-tune the pre-trained BERT base model (Devlin et al., 2019) (12 layers, 12 heads, and 768 hidden dimensions). The token embeddings are randomly initialized. More details about training can be found in Appendix C.

To improve the inference efficiency of COPY-GENERATOR, we encode all the documents in the source text collections offline. Note that retrieving from such a super large phrase collection faces severe challenges on the engineering side. This paper uses a coarse-to-fine pipeline to address this challenge. Specifically, we first use a document retriever to retrieve top-k related documents for each given prefix. Then, their corresponding phrase representations are collected for selection. In this paper, a popular semantic matching model, SimCSE (Gao et al., 2021) and vector search toolkit, FAISS (Johnson et al., 2019) are used as the document retriever, which can recall documents that have a similar topic with the prefix. The value k is empirically set to 50.

COPY-GENERATOR can be used with both greedy search and nucleus sampling. For greedy search, COPY-GENERATOR selects the phrase that has the highest fitness scores at each time step. As for nucleus sampling, we first obtain the next-phrase distribution by using the `softmax` function over the fitness scores of all candidate phrases. Then, the next phrase is sampled over this distribution.

[1] We use a variant of nucleus sampling, which additionally restricts the candidate pool to the k most likely next tokens (Fan et al., 2018).

[2] https://huggingface.co/

4.3 EVALUATION METRICS

For each document in the test set, we use the first 50% tokens as the prefix. The baselines and our proposed COPY-GENERATOR generate text continuations of length 128 based on the same prefix. Following previous work (Welleck et al., 2020; Su et al., 2022), we use the following evaluation metrics:

- **MAUVE** (Pillutla et al., 2021), an efficient, interpretable, practical automatic evaluation, is highly coherent with human judgments and widely used to evaluate modern text generation models (Su et al., 2022; Krishna et al., 2022). In this paper, MAUVE leverages the GPT2-large model to generate the scores.
- **Rep-n** (Welleck et al., 2020) measures the sequence-level repetition as the portion of duplicate n-grams in the generated text (Welleck et al., 2020). For a generation text x, Rep-n can be formulated as: $100 \times (1.0 - \frac{|\text{unique n-gram(x)}|}{|\text{total n-gram(x)}|})$. Higher Rep-$n$ denotes the severe degeneration problem in generations.
- **Diversity** scores (Welleck et al., 2020) measure the diversity of the generations, which can be formulated as: $\Pi_{n=2}^{4}(1 - \frac{\text{Rep}-n}{100}))$. A generation that has a higher Diversity score usually is more informative.

Furthermore, we also compare the average time cost of different methods for completing the generation on the test set. Since the phrase representations are pre-computed offline, its encoding time cost is not included.

Note that previous work (Khandelwal et al., 2020; Dai et al., 2019) often uses perplexity as the primary evaluation metric to measure the performance of language modeling. However, since our proposed COPY-GENERATOR does not calculate next-token distributions over a fixed vocabulary, the comparison of perplexities is not reliable and thus omitted.

5 EXPERIMENTAL RESULTS

In this paper, we evaluate baselines and our proposed COPY-GENERATOR in three different settings: (1) standard language modeling; (2) domain adaption; (3) enlarged memory.

5.1 LANGUAGE MODELLING ON WIKITEXT-103 DATASET

In this setting, models are trained on the training set of the WikiText-103 dataset and evaluated on its test set. The WikiText-103 dataset (Merity et al., 2017) contains an extensive collection of Wikipedia articles with over 100 million words, which is widely used to evaluate the performance of universal language modeling (Khandelwal et al., 2020; Dai et al., 2019; Su et al., 2022).

Results Table 1 shows the performance comparison between the baselines and our proposed COPY-GENERATOR on the test set of the WikiText-103 corpus. It can be found that our proposed COPY-GENERATOR substantially outperforms the Transformer and kNN-LM baselines on most metrics. Specifically, COPY-GENERATOR improves MAUVE score over the best baseline (Transformer with nucleus sampling) from 0.691 to 0.758 – an improvement of 6.7%. Interestingly, although it is well known that greedy search could raise severe degeneration problems (Welleck et al., 2020), COPY-GENERATOR with greedy search still outperforms the standard Transformer baseline (nucleus sampling), with 6.4% and 8.6% improvements on MAUVE and Diversity scores, respectively. This observation demonstrates that COPY-GENERATOR is more robust and less prone to the degeneration problem, which can be considered as an additional bonus. We can see that the kNN-LM baseline is slightly worse than the Transformer baseline, likely due to that the kNN-LM baseline is trained from scratch while the Transformer baseline is fine-tuned from the pre-trained GPT2.

Inference Speed We also report the generation latency of different models in Table 1. As seen, COPY-GENERATOR achieves over $2\times$ speedup over the standard Transformer baseline. The reason is that most phrases contain multiple tokens. The average length of the retrieved phrases is 3.95

during generation. As a result, our proposed COPY-GENERATOR conducts much fewer decoding steps than standard language models when generating text continuations of the same length. Unlike COPY-GENERATOR that uses a coarse-to-fine search pipeline, kNN-LM conducts large-scale vector search at every decoding step. Thus, its inference latency is much higher than Transformer and COPY-GENERATOR.

Model	Decoding	MAUVE ↑	Diversity ↑	Rep-2↓	Rep-3↓	Rep-4 ↓	Latency↓
Transformer	greedy	0.639	0.051	0.668	0.624	0.591	0.662
	nucleus	0.691	0.826	0.118	0.047	0.024	0.733
kNN-LM	greedy	0.630	0.029	0.727	0.688	0.658	4.27
	nucleus	0.687	**0.954**	0.040	0.055	0.009	4.31
Copy-Generator	greedy	0.755	0.912	0.049	0.025	0.017	0.331
	nucleus	**0.758**	0.942	**0.039**	**0.013**	**0.007**	**0.303**

Table 1: The automatic evaluation on the test set of WikiText-103. MAUVE score ($c = 1$) is calculated by the GPT2-Large model. Latency is measured by the average inference time cost (in seconds) on the test set.

Case Study For a better understanding of the performance of COPY-GENERATOR, we present an example of the text continuations generated by our proposed COPY-GENERATOR in Figure 2. It can be found that COPY-GENERATOR can retrieve phrases that are semantically coherent and fluent for given prefixes. For example, at the second decoding step, COPY-GENERATOR copied the token *"* from the pre-defined vocabulary to close the film name *"The Hounds of Baskerville"*. Besides, at the sixth decoding step, COPY-GENERATOR copied the phrase *Boulter is an* from the given prefix, which is consistent with the content (Boulter) included in the prefix.

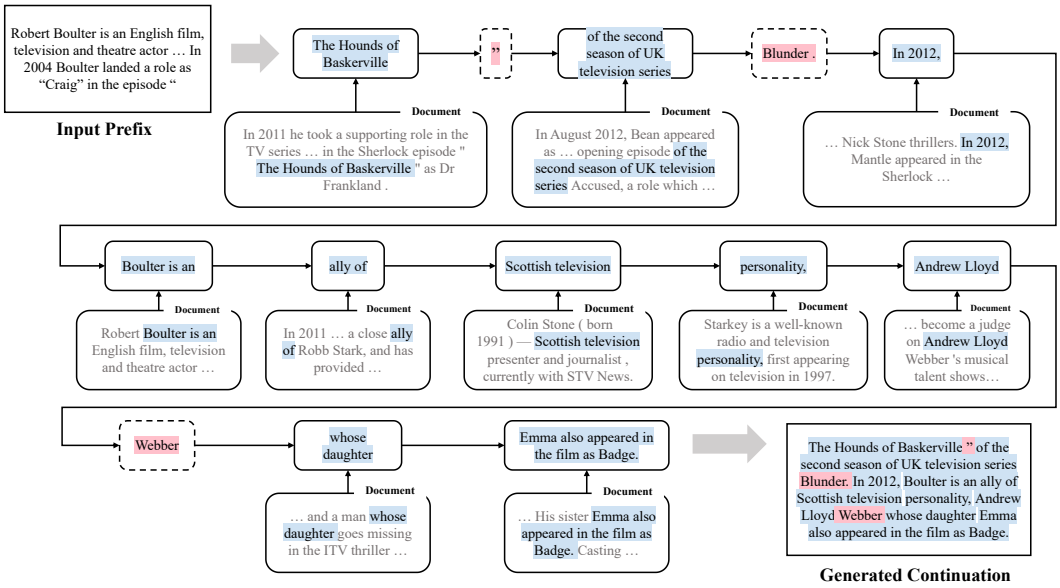

Figure 2: An example generated by COPY-GENERATOR. The dotted squares denote that the content (highlighted in red)is generated from the token vocabulary, and the solid squares denote that the content (highlighted in blue) is copied from other documents.

5.2 DOMAIN ADAPTION ON LAW-MT DATASET

In the domain adaption setting, the models trained on the WikiText-103 dataset are tested on a specific domain. Following previous work (He et al., 2021; Alon et al., 2022), we use the English

part of Law-MT (Koehn & Knowles, 2017), which is an English-German translation dataset for law documents. The datastore in kNN-LM and the phrase table in COPY-GENERATOR are constructed from the training set of Law-MT. We also present the performance of Transformer baselines with or without further fine-tuning on the training set of Law-MT.

Results The experimental results of the automatic evaluation are shown in Table 2. Surprisingly, it can be observed that COPY-GENERATOR even outperforms the Transformer model further fine-tuned on the Law-MT corpus (Transformer w/ FT). Specifically, COPY-GENERATOR outperforms Transformer w/ FT by 5% MAUVE score, 10% Diversity, and 2.1% Rep-4 improvements, respectively. The results indicate that COPY-GENERATOR allows a single model to be specialized in different domains, by simply switching the source text collection. Although kNN-LM brings in higher Diversity and Rep-n scores, COPY-GENERATOR surpasses it by 6.8% MAUVE score, which shows COPY-GENERATOR has higher generation quality in general.

Model	Decoding	MAUVE ↑	Diversity ↑	Rep-2 ↓	Rep-3 ↓	Rep-4 ↓
Transformer w/o FT	greedy	0.534	0.024	0.740	0.709	0.687
	nucleus	0.603	0.710	0.174	0.090	0.056
Transformer w/ FT	greedy	0.588	0.546	0.247	0.167	0.129
	nucleus	0.602	0.710	0.166	0.092	0.063
kNN-LM	greedy	0.525	0.021	0.755	0.721	0.697
	nucleus	0.585	**0.944**	**0.049**	**0.007**	**0.001**
Copy-Generator	greedy	0.644	0.725	0.143	0.092	0.069
	nucleus	**0.653**	0.810	0.111	0.056	0.035

Table 2: The automatic evaluation on the test set of the Law-MT corpus. kNN-LM model is trained on the WikiText-103 dataset, and the memory is built on the training set of the Law-MT. Transformer w/ FT denotes the Transformer model that is further fine-tuned on the training set of the Law-MT corpus. Transformer w/o FT denotes the Transformer model that is only trained on the training set of the WikiText-103.

5.3 ENLARGED MEMORY ON EN-WIKI DATASET

In the enlarged memory setting, we make use of a large text collection, the En-Wiki corpus. The En-Wiki corpus contains a large-scale collection of Wikipedia articles with over 3 billion words, whose size is much larger than the WikiText-103 dataset. The datastore in kNN-LM and the phrase table in COPY-GENERATOR are constructed from the training set of En-Wiki. Similar to the domain adaption setting, we also present the results of Transformer baselines with or without further fine-tuning on the En-Wiki corpus.

Model	Decoding	MAUVE ↑	Diversity ↑	Rep-2 ↓	Rep-3 ↓	Rep-4 ↓
Transformer w/o FT	greedy	0.220	0.038	0.701	0.661	0.630
	nucleus	0.650	0.793	0.132	0.058	0.031
Transformer w/ FT	greedy	0.324	0.078	0.612	0.568	0.537
	nucleus	0.674	0.760	0.145	0.071	0.044
kNN-LM	greedy	0.412	0.052	0.674	0.620	0.580
	nucleus	0.562	**0.946**	**0.048**	**0.006**	**0.001**
Copy-Generator	greedy	0.799	0.694	0.151	0.108	0.085
	nucleus	**0.806**	0.813	0.104	0.058	0.038

Table 3: The automatic evaluation on the test set of the En-Wiki. Transformer w/o FT denotes the GPT2 model that is only trained on the training set of the WikiText-103 corpus. Transformer w/ FT denotes the Transformer model that is further fine-tuned on the En-Wiki corpus.

Results The experimental results are shown in Table 3. COPY-GENERATOR even surpasses the Transformer baseline further fine-tuned on the En-Wiki corpus by 13.2%, 2% MAUVE and Diversity scores. This is especially remarkable because COPY-GENERATOR does not require any additional training, suggesting we can use learn COPY-GENERATOR with a smaller corpus but leverage additional information in a larger corpus in a plug-and-play fashion. Similar to the domain adaption setting, we also notice that, although kNN-LM baseline improves Diversity and Rep-n scores, it obtains a much lower MAUVE score than COPY-GENERATOR (0.562 vs. 0.806).

Affect of Memory Size In this subsection, we further analyze how the size of the added En-Wiki memory influences the generation quality. Specifically, we create source text collections using different numbers of documents randomly sampled from the En-Wiki corpus. As shown in Figure 3, the generation quality generally increases as the number of the source text documents of COPY-GENERATOR becomes larger. It can be observed that COPY-GENERATOR achieves lower MAUVE scores than the standard Transformer baseline when the number of the used documents is lower than 5K. With the number of added documents increasing, the MAUVE scores of COPY-GENERATOR become higher, and the best MAUVE score (0.806) is attained given all the En-Wiki documents.

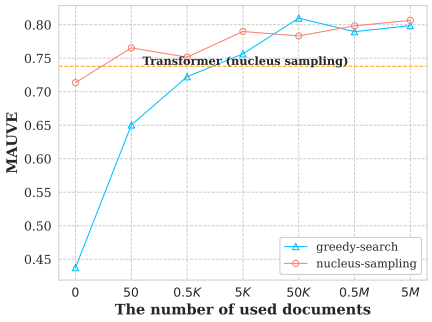

Figure 3: For the Transformer baseline, its nucleus sampling hyper-parameters are chosen by the grid search on the MAUVE score, and its best MAUVE score is 0.74.

6 RELATED WORK

Dense Retrieval The dense retrieval technique (Karpukhin et al., 2020) has been widely used in many downstream NLP tasks, such as open-domain question answering (Karpukhin et al., 2020; Lee et al., 2021) and open-domain dialogue systems (Lan et al., 2021). Different from the traditional sparse retrieval system, such as BM25 and TF-IDF (Robertson & Zaragoza, 2009), dense retrieval learns a shared vector space for queries and documents, where relevant pairs of query and document have smaller distances (i.e., higher similarities) than the irrelevant pairs.

The most closely related work to our study is DensePhrase (Lee et al., 2021). DensePhrase reformulates the question answering task as a phrase retrieval problem, where phrases are directly retrieved and returned as answers to factual questions. Differently, our work aims to generate coherent text continuations through multiple rounds of phrase retrieval. Since the connection between two adjacent phrases should be coherent and fluent in the text generation task, it is much more difficult.

Retrieval-Augmented Text Generation Retrieval-augmented text generation has gained increasing interest recently. Most prior work improves the generation quality (e.g., informativeness) of language models by grounding the generation on a set of retrieved materials (e.g., relevant documents) (Guu et al., 2018; Hashimoto et al., 2018; Weston et al., 2018; Cai et al., 2019a;b; Khandelwal et al., 2020; Wu et al., 2019; Guu et al., 2020; Lewis et al., 2020; Borgeaud et al., 2022). Our work is on this line of research but takes a radical step forward. Unlike prior work that builds the combinations of retrieval and generation, retrieval is generation in COPY-GENERATOR.

7 CONCLUSION

In this paper, we reformulated text generation as progressively copying phrases from the massive text collection. Following this formalization, we proposed a novel neural text generation model, named COPY-GENERATOR, which generates text by retrieving semantically coherent and fluent phrases from other documents. Experimental results proved the advantages of COPY-GENERATOR over the strong baselines on three experimental settings: standard language modeling (WikiText-103), domain adaptation (Law-MT dataset), and enlarged memory (En-Wiki).

REFERENCES

Uri Alon, Frank Xu, Junxian He, Sudipta Sengupta, Dan Roth, and Graham Neubig. Neuro-symbolic language modeling with automaton-augmented retrieval. In *International Conference on Machine Learning*. PMLR, 2022.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning*. PMLR, 2022.

Deng Cai, Yan Wang, Wei Bi, Zhaopeng Tu, Xiaojiang Liu, Wai Lam, and Shuming Shi. Skeleton-to-response: Dialogue generation guided by retrieval memory. 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)*, 2019a.

Deng Cai, Yan Wang, Wei Bi, Zhaopeng Tu, Xiaojiang Liu, and Shuming Shi. Retrieval-guided dialogue response generation via a matching-to-generation framework. 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)*, 2019b.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

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

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

Kelvin Guu, Tatsunori B. Hashimoto, Yonatan Oren, and Percy Liang. Generating sentences by editing prototypes. *Transactions of the Association for Computational Linguistics*, 6, 2018.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*, 2020.

Tatsunori B Hashimoto, Kelvin Guu, Yonatan Oren, and Percy S Liang. A retrieve-and-edit framework for predicting structured outputs. In *Advances in Neural Information Processing Systems*, 2018.

Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. Efficient nearest neighbor language models. *arXiv preprint arXiv:2109.04212*, 2021.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3), 2019.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, 2017.

Kalpesh Krishna, Yapei Chang, John Wieting, and Mohit Iyyer. Rankgen: Improving text generation with large ranking models. *arXiv preprint arXiv:2205.09726*, 2022.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2018.

Tian Lan, Deng Cai, Yan Wang, Yixuan Su, Xian-Ling Mao, and Heyan Huang. Exploring dense retrieval for dialogue response selection. *arXiv preprint arXiv:2110.06612*, 2021.

Jinhyuk Lee, Mujeen Sung, Jaewoo Kang, and Danqi Chen. Learning dense representations of phrases at scale. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2021.

Kenton Lee, Shimi Salant, Tom Kwiatkowski, Ankur Parikh, Dipanjan Das, and Jonathan Berant. Learning recurrent span representations for extractive question answering. *arXiv preprint arXiv:1611.01436*, 2016.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. Mauve: Measuring the gap between neural text and human text using divergence frontiers. *Advances in Neural Information Processing Systems*, 34, 2021.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8), 2019.

S. Robertson and H. Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3, 2009.

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

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016.

Minjoon Seo, Tom Kwiatkowski, Ankur Parikh, Ali Farhadi, and Hannaneh Hajishirzi. Phrase-indexed question answering: A new challenge for scalable document comprehension. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. A contrastive framework for neural text generation. *arXiv preprint arXiv:2202.06417*, 2022.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

Jason Weston, Emily Dinan, and Alexander Miller. Retrieve and refine: Improved sequence generation models for dialogue. In *Proceedings of the 2018 EMNLP Workshop SCAI: The 2nd International Workshop on Search-Oriented Conversational AI*, 2018.

Yu Wu, Furu Wei, Shaohan Huang, Yunli Wang, Zhoujun Li, and Ming Zhou. Response generation by context-aware prototype editing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2019.

A THE PHRASE SEGMENTATION ALGORITHM

COPY-GENERATOR takes phrases as the minimum units that can be put together to form a coherent document. To train COPY-GENERATOR, we design a phrase segmentation algorithm to split each document in the training set into a sequence of phrases. This algorithm makes use of a forward maximum matching strategy to identify phrases. Maximum matching is one of the most popular structural segmentation algorithms. This method favors long phrases and is a greedy algorithm by design. Specifically, we treat each document as a sequence of tokens and scan each document from left to right. At each step, we search for the longest prefix of the unsegmented part that is also a sub-sequence of other documents other than the current document. If the length of that prefix is bigger than 2, we take that prefix as the next phrase. Otherwise, we take the first token as the next phrase and it is labeled as coming from the fixed token vocabulary. In both cases, we process the rest part of the current document recurrently. The algorithm can be very time-consuming because exhaustive searches over millions of documents are compute-intensive. Therefore, we propose an efficient approximation as follows. First, we retrieve the top-k most similar documents for each document using the popular semantic matching model, SimCSE (Gao et al., 2021), and vector search toolkits, FAISS (Johnson et al., 2019). Then, the phrase search only runs on the corresponding top-k documents. The relevant documents usually have similar topics to the current document. The value of k is set as 128 in our experiments. The details of our proposed phrase segmentation algorithm can be found in Algorithm 1: SearchPhrase is a function that searches the cached token sequence (i.e., the current candidate for the next phrase) among the most relevant documents. It returns a label that denotes whether the phrase can be found and its position in the relevant documents.

B HOW THE NUMBER OF RETRIEVED DOCUMENTS AFFECTS THE GENERATION QUALITY

Figure 4 demonstrates how the number of the Top-k documents influences the diversity of the text continuations generated by our proposed COPY-GENERATOR. We could make the following conclusions: (1) with the number of the retrieved documents increasing, the diversity of the text continuations becomes higher, no matter what decoding method is chosen; (2) COPY-GENERATOR with greedy search outperforms the Transformer model with nucleus sampling (0.848 vs. 0.83 Diversity score) when the number of retrieved documents is larger than 20. Noted that the parameters of nucleus sampling for Transformer baseline are carefully selected by the grid search; (3) COPY-GENERATOR with nucleus sampling achieves comparable diversity metric (0.942 vs. 0.94) when the number of the documents is larger than 90.

C MORE TRAINING DETAILS

We train baselines and COPY-GENERATOR for 200,000 steps on 8 Tesla-V100 GPUs. For all the baselines, the learning rate, dropout rate, and gradient clipping is set as 5e-5, 0.1, and 1.0, respectively. To improve the training efficiency, only the last layer of the BERT model in the phrase

Algorithm 1: Phrase Segmentation Algorithm

Data: Document set: $\mathcal{D} = \{d_i, \{d_j\}_{j=1}^{K}\}_{i=1}^{N}$, where d_i denotes the i-th document. K denotes the number of the retrieved documents. N denotes the number of documents in the training set. The pre-defined maximum and minimum phrase lengths are L_{max} and L_{min}.

Result: Segmented document set by phrase granularity: $\mathcal{D}' = \{\{(p_{i,x}, (d_j, \mathrm{pos_j}))\}_{x=1}^{||d_i||_p}\}_{i=1}^{N}$, where $p_{i,x}$ denotes the x-th phrase in d_i that also appears in another document d_j in position j. $||d_i||_p$ denotes the number of the collected phrases in d_i.

Preprocess: split each document into token-level pieces by using the off-the-shelf tokenizer. The preprocessed document set can be formulated as $\mathcal{D} = \{\{t_{i,x}\}_{x=1}^{||d_i||_t}, \{d_j\}_{j=1}^{K}\}_{i=1}^{N}$, where $t_{i,x}$ is the x-th token of d_i, which consists of $||d_i||_t$ tokens. Prepare the empty list $\mathcal{D}' = \{\}$, empty phrase cache cache$_p$={}, and cached search success label label$_{last}$.

```
for i ← 1 to N do
    cursor=0
    PhraseCollection={}
    while cursor≤ ||d_i||_t do
        if L_min ≤ len(cache_p) ≤ L_max then
            label_now, rest=SearchPhrase(cache_p)
        else
            if len(cache_p) > L_max then
                cache_p={}
            end
        end
        if label_last is True and label_now is False then
            cursor -= 1
            PhraseCollection.append(cache_p, rest)
            cache_p={}
        else
            if label_last is False and label_now is False then
                PhraseCollection.append( cache_p, None)
                cache_p={}
            end
        end
        cursor += 1
        label_now=label_last
    end
    D'.append(PhraseCollection)
end
```

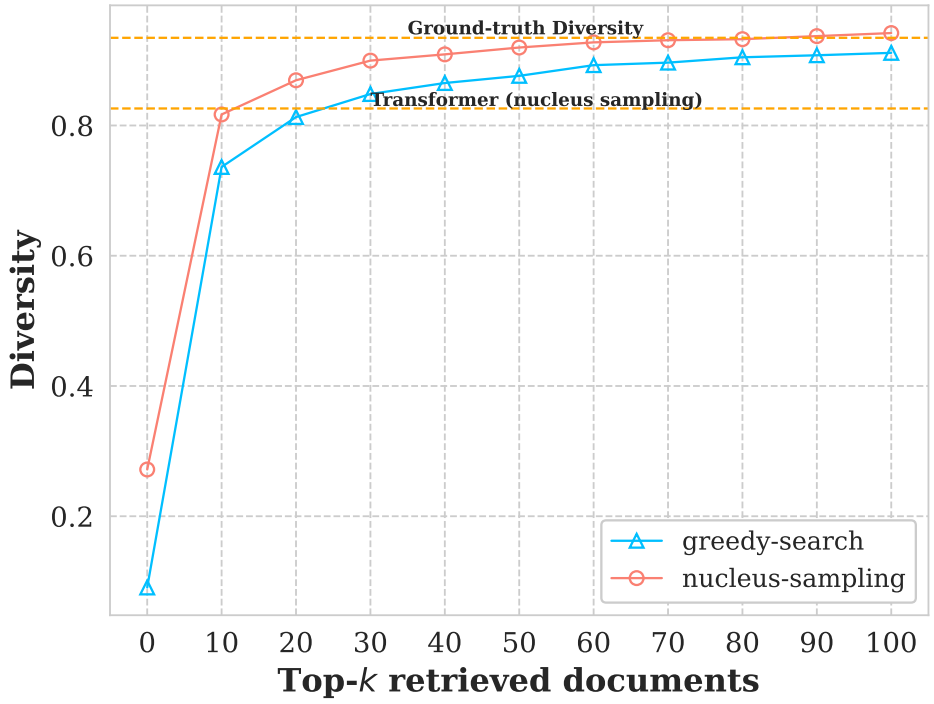

Figure 4: How the number of the retrieved documents influences the diversity of the text continuations generated by our proposed COPY-GENERATOR.

encoder is fine-tuned, and other parameters are frozen. Due to the memory limitation, we only optimize 256 prefix-phrase pairs for one batch. For the BERT model in phrase encoder, the maximum sequence length is set as 256. For the GPT2 model in prefix encoder, the maximum sequence length is set as 512.

D MORE CASES

In this section, we present some generated examples of COPY-GENERATOR given some specific prefixes. As shown in Figure 5 and 6, it can be observed that the generated continuations are fluent and coherent with the prefix. For example, in Figure 5, COPY-GENERATOR lists some different types of matches in the game, such as *elimination type matches*, *tag team matches* and *dark matches* from the documents that have the same topic (*WWE video game*) with the prefix.

Additionally, we also show some bad cases generated by COPY-GENERATOR. As shown in Figure 7, COPY-GENERATOR retrieves a phrase *Beavers)* that contains a right bracket. Since there is no left bracket in the prefix, this inappropriate phrase leads to not fluent and confusing generated continuation. Moreover, as shown in Figure 8, in the fourth step, the retrieved phrase *who become an important Norwegian poet* is incoherent for its prefix. In this case, the right bracket should be directly used to finish the text segments *(The Bridge of the Vaaben)*.

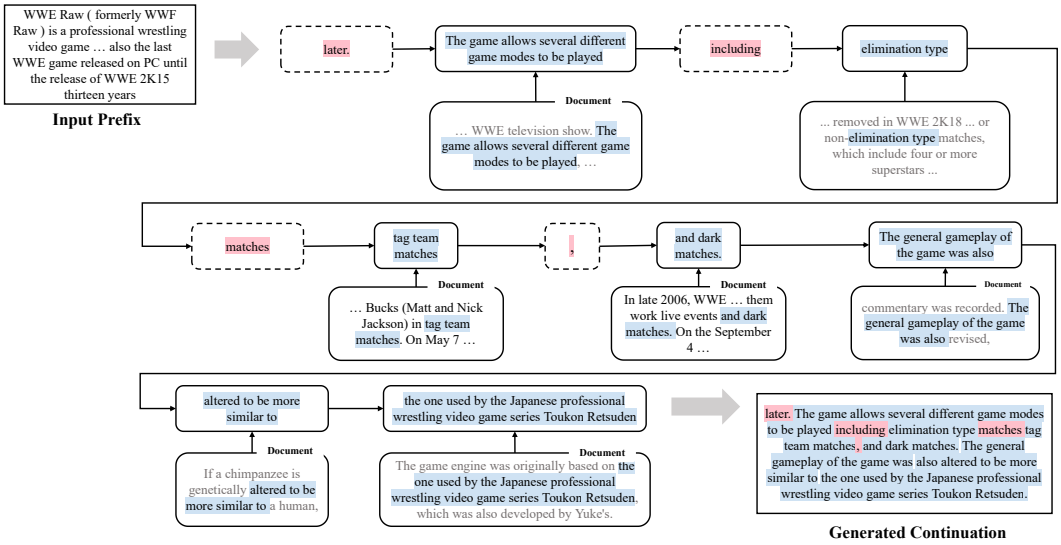

Figure 5: An example generated by COPY-GENERATOR. The dotted squares denote that the content (highlighted in red)is generated from the token vocabulary, and the solid squares denote that the content (highlighted in blue) is copied from other documents.

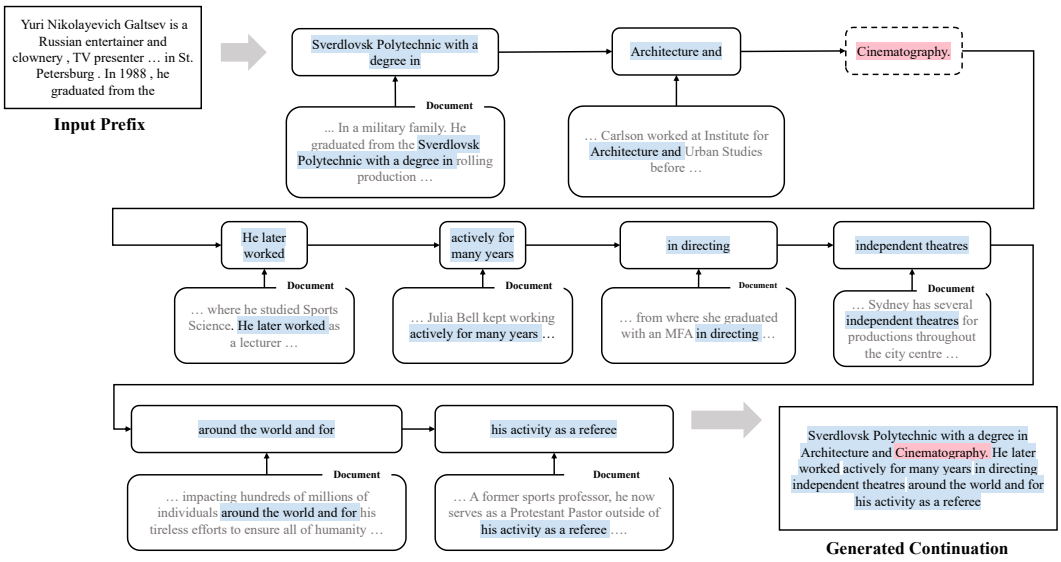

Figure 6: An example generated by COPY-GENERATOR. The dotted squares denote that the content (highlighted in red)is generated from the token vocabulary, and the solid squares denote that the content (highlighted in blue) is copied from other documents.

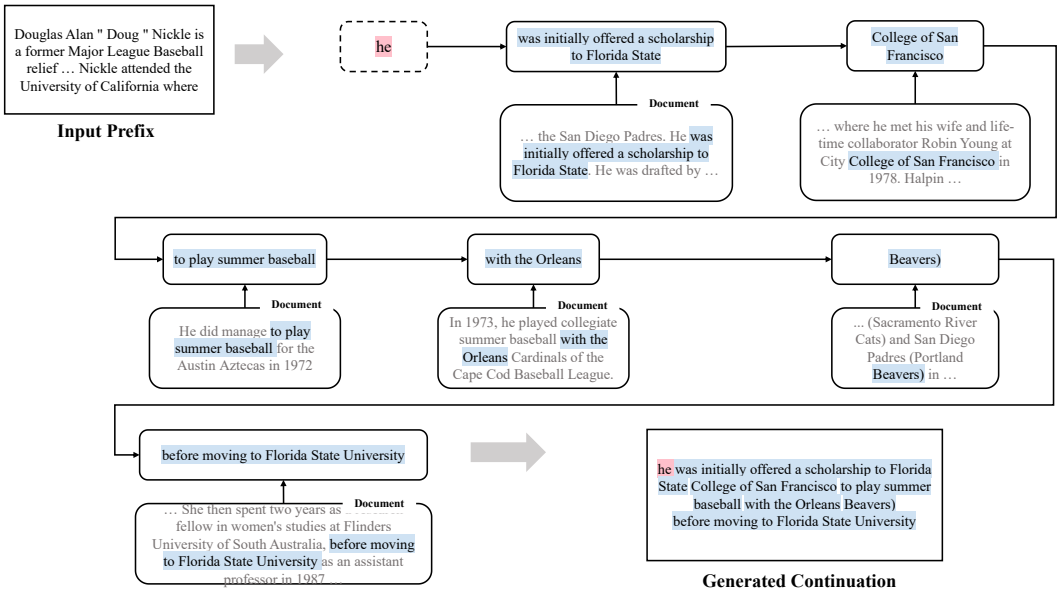

Figure 7: An example generated by COPY-GENERATOR. The dotted squares denote that the content (highlighted in red)is generated from the token vocabulary, and the solid squares denote that the content (highlighted in blue) is copied from other documents.

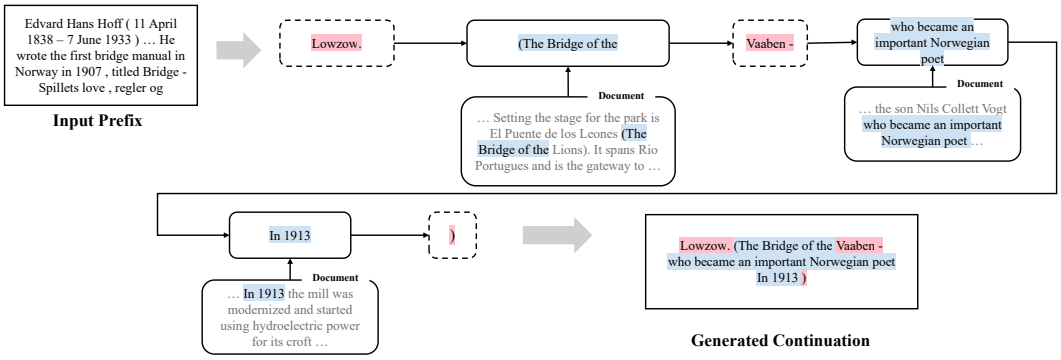

Figure 8: An example generated by COPY-GENERATOR. The dotted squares denote that the content (highlighted in red)is generated from the token vocabulary, and the solid squares denote that the content (highlighted in blue) is copied from other documents.