Generation with Dynamic Vocabulary

Anonymous ACL submission

Abstract

Vocabulary is a crucial component of language models. Traditional language models generate text by selecting tokens from a fixed vocabulary. In this paper, we introduce a novel dynamic setting for the vocabulary. Under this setting, vocabulary can include arbitrary text spans on demand. These text spans act as basic bricks, akin to tokens in the fixed vocabulary. Our proposed model can be deployed in a way of plug-and-play. Extensive experimental results demonstrate that our approach yields superior generation quality. For instance, compared to the standard language model, the MAUVE metric increases from 20.47 % to 25.69%. We also demonstrate that dynamic vocabulary can be effectively applied to different domains in a training-free manner, and it also helps to generate reliable citations in question answering tasks (substantially enhancing citation results without compromising answer accuracy). [1]

1 Introduction

Vocabulary, which defines basic bricks (tokens) for composing new sentences, bridging different languages, and alleviating harmful generations, is essential for language models (Stahlberg, 2020; Lample and Conneau, 2019; Liu et al., 2020; Kirk et al., 2022; Weidinger et al., 2021). In modern development, vocabularies are often obtained by training tokenizers with a pre-defined vocabulary size on a pre-defined corpus. Once built, they are kept unchanged in the following model construction and deployment (Sennrich et al., 2015; Radford et al., 2019).

Though it is enough for basic language modeling, this *static* setting makes vocabulary be quietly ignored in advanced generation tasks (Gao et al., 2023; Rozière et al., 2024; Fried et al., 2023; Dagan et al., 2024). For example, it can not be augmented with new phrases for better adapting an unseen domain (Koehn and Knowles, 2017; Jin et al., 2020; Chen et al., 2022) or verbatim reference text spans for better inline evidence generation (Menick et al., 2022; Gao et al., 2023). To bring vocabulary back to the stage, it is natural to ask whether prior constraints posted by tokenization corpus and fixed vocabulary size can be relaxed.

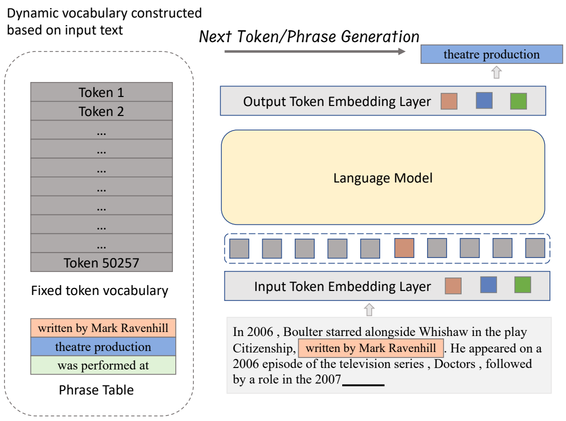

Figure 1: Generation process of our proposed dynamic vocabulary. The model's vocabulary dynamically changes based on the input text, with phrases serving as basic blocks that are directly input and output.

Here, we explore vocabulary in a new *dynamic* setting. Instead of being a fixed token table, dynamic vocabulary is required to be able to include *arbitrary text spans* on demand. This setup brings new challenges to the language model. On the input side, using a single embedding layer is no longer feasible as the full table can not be enumerated. On the output side, the model needs a stronger next-token predictor as the model allows multiple oracles (tokenized to different granularity) for a single string.

In this work, we build a dynamic vocabulary by building a dynamic phrase encoder. Akin to the embedding layer, the encoder maps arbitrary

[1] Our source code is publicly available at https://anonymous.4open.science/r/dynamic_vocabulary-7C1C

text spans (called *phrases*) to the input space of language models. It can be trained with existing language models in the same self-supervised manner, despite that multiple tokens (in the original static vocabulary) can be input or output at a single step. Though the paradigm is almost unchanged, supporting dynamic tokens needs non-trivial modification on data curation. Specifically, we find that, to prevent the learned model from either biased towards full static token outputs or towards full new phrase outputs, it is crucial to make the two properly interleaved in training samples. We also show that the token encoder is hard to learn without informative negative samples. We thus develop two retrieval-based and generation-based methods for accelerating the learning of the dynamic phrase encoder.

The obtained dynamic vocabulary can be deployed in the way of plug-and-play: the underlying architecture (and backbone parameters) of language models are kept, and those new on-demand phrases can be used as ordinary tokens during the generation. To evaluate the dynamic vocabulary, we investigate three exemplar applications, including basic language modeling, domain adaptation, and generating citations for question answering. Results show that the new flexibility of vocabulary both improve basic generation performances (e.g., stronger fluency and diversity scores on WikiText-103 (Merity et al., 2016) with lower latency) and provide a new tool to handle advanced language modeling tasks (e.g., generating more accurate citations with QA scores also increased).

2 The Approach

2.1 Problem Definition

Given a language model LM, denote V as its vocabulary, and $x = x_1, x_2, ..., x_n$ as a tokenized sentence according to V (x_i is a token in V). A *dynamic vocabulary* $V' = V \cup P$ augments V with arbitrary phrases (text spans) P. The same sentence x now can be tokenized to a different sequence $x'_1, x'_2, ..., x'_m$, where $x'_i \in V'$. The usage of dynamic vocabulary V' is identical to the vanilla static vocabulary V: the language model LM can accept any token in V' as input and choose output tokens from V'.

Supporting arbitrary phrase set P and integrating V' with language models are two cruxes to implement dynamic vocabularies. For the first one, it is possible to support new phrases by fine-tuning the language model with V', but it requires updating the model when P changes which can hardly be used in real applications. We will also see that, for the second crux, simply replacing V with V' fails to learn the language model due to the decoding ambiguity introduced by P. We elaborate our solutions in the following sections.

2.2 Dynamic Phrase Encoder

Instead of fine-tuning the language model for every possible P to support arbitrary phrase sets, we build a parametric encoder for those dynamic phrases. Once the encoder is learned, it can be deployed with the model.

Specifically, the dynamic phrase encoder is built with a causal Transformer. To get the representation of a phrase $p \in P$, it first tokenizes $p = w_1, w_2, ..., w_s$ according to the static vocabulary V, and after going through several causal Transformer layers followed by an MLP, the hidden vector of the last token $\mathbf{h}_s$ is the vector representation of p.

The above setting is different from existing work in three ways (Lan et al., 2023; Teehan et al., 2024). First, it is common to use Transformer encoder (full attention) to build the phrase encoder, while we apply Transformer decoders (causal masking). The choice is mainly guided by efficient negative sampling (see Section 2.4 for further details).

Second, the dynamic phrase encoder adopts the same tokenizer of LM (which is used to build the static vocabulary V). Sharing tokenizers means the language model doesn't need to load additional vocabularies and tokenizers during inference. [2]

Third, to further unify the new phrase encoder and the LM, we use a non-contextualized representation of phrases, which makes the new phrases more like the original tokens in V. Contextualized representations can also be used (Joshi et al., 2020; Lan et al., 2023), but it means that, besides the phrases themselves, the contexts of them should also be included in the dynamic vocabulary.

To summarize, the considerations above aim to make the dynamic phrase encoder aligns with the embedding layer as much as possible: both of them map tokens (phrases) into the input space of the language model, one by lookup operations, and another by running the phrase encoder.

[2] As a comparison, the phrase encoder in CoG (Lan et al., 2023) is BERT, and one should load both the BERT vocabulary and GPT-2 vocabulary when testing.

2.3 Inference with Dynamic Vocabulary

In testing time, the new dynamic vocabulary can be used as the ordinary vocabulary. We take an auto-regressive language model LM as an example. For a set of new phrases P [3], we run the learned dynamic phrase encoder to get representations of its phrases, denoted by a matrix $\mathbf{P}$. The language model's input and output embedding matrices $\mathbf{W}_{\text{emb,in}}, \mathbf{W}_{\text{emb,out}}$ are expanded with these embeddings,

$$\mathbf{W}'_{\text{emb,in}} = [\mathbf{W}_{\text{emb,in}}, \mathbf{P}],$$
$$\mathbf{W}'_{\text{emb,out}} = [\mathbf{W}_{\text{emb,out}}, \mathbf{P}].$$

At each auto-regressive decoding step, the language model LM outputs a hidden vector $\mathbf{h}_{<i}$ representing current prefix $x'_{<i}$, the probability of next token is

$$\mathbb{P}(x'_i = k | x'_{<i}) = Z^{-1} \exp(\mathbf{h}_{<i} \cdot \mathbf{e}^k_{\text{out}}) \quad (1)$$
$$Z = \sum_{k' \in V} \exp(\mathbf{h}_{<i} \cdot \mathbf{e}^{k'}_{\text{out}}) + \sum_{k' \in P} \exp(\mathbf{h}_{<i} \cdot \mathbf{e}^{k'}_{\text{out}}),$$

where $\mathbf{e}^k_{\text{out}}$ is the k-th column of $\mathbf{W}'_{\text{emb,out}}$. When the i-th token is selected, no matter whether it is a token in V or a phrase in P, its embedding is looked up from $\mathbf{W}'_{\text{emb,in}}$ as the input of the next decoding step. [4]

2.4 Training with Dynamic Vocabulary

Building Samples To train the dynamic phrase encoder, we follow the same self-supervision regime as the training of language models. The key difference here is that, besides tokens in V, we need to organize phrases (text spans) in a training sample for learning the phrase encoder. In particular, 1) the diversity of training time in-domain phrases would influence the generalization of the learned phrase encoder, and 2) the distribution of phrases in samples would influence the how the language model switches between tokens and phrases.

For building phrases, we test the following two methods.

- *"real" phrases*. We can use classical chunking algorithms to recognize phrases in a sentence. The result phrases can recognized as a single grammatical unit or as a common word collocation. Here, we follow Lan et al. (2023) to use an unsupervised chunker forward maximum matching (FMM). Basically, FMM recognizes phrases that frequently appear in a support corpus and as long as possible. The algorithm (and other external chunkers) may need additional time costs to compile samples (e.g., in our experiments, FMM needs ≈ 15 hours to build its phrase table).

- *Ngrams*. Another candidate set of phrases is ngrams, which is much simpler to build than involving external chunkers. Though a ngram may not carry a meaning, it could be a stronger learning target for the phrase encoder: the connections between ngrams and its contexts are more complex than "real" phrases (as they usually follow the simple patterns which are used to extract them). We study two settings, ngrams of words and ngrams of tokens (denoted by N-words and N-ids respectively). Taking N-words as an example, a word tokenizer [5] first recognizes words in a sentence, then randomly sequences of 2-5 consecutive words are grouped into phrases.

Next, given a sentence and a set of candidate phrases, we need to determine the distribution of phrases. One may build samples with full ngrams phrases, but they could be both hard to learn (the learning ignores the prior knowledge of original vocabulary V in the model), and hard to apply (the setting is rare in applications). In our practice, to accelerate learning and prevent unnecessary data bias, it is crucial to make phrases and tokens properly interleaved in training samples. Therefore, we control the interval between two phrases to be at least five tokens.

Negative Phrases After building training samples, we can directly optimize the log-probability defined in Equation 1, which requires the correct next token in $V' = V \cup P$ has the largest logit than other tokens in V and P (negative tokens). However, the number of phrases in the training set would be large, and it is prohibitive to include all of them in the loss function. [6] A common

[3] The phrase set P can change at each decoding step. Here, for simplicity, we assume it is kept unchanged during testing, and we can run the dynamic phrase encoder only once.

[4] When decoding a phrase, another option adopted by (Joshi et al., 2020; Lan et al., 2023) is to unfold tokens in the phrase and input them individually. Despite the inconsistency between input and output vocabulary (our experiments indicate a negative influence on performances), this setting may also slow the decoding speed (or generate shorter texts given a fixed length budget) even if it can predict a phrase.

[5] N-words uses the word tokenizer in the NLTK toolkit, and N-ids uses GPT-2's tokenizer.

[6] It is worth noting that all training time phrases are dropped after learning the encoder. For ngram phrases (N-words and

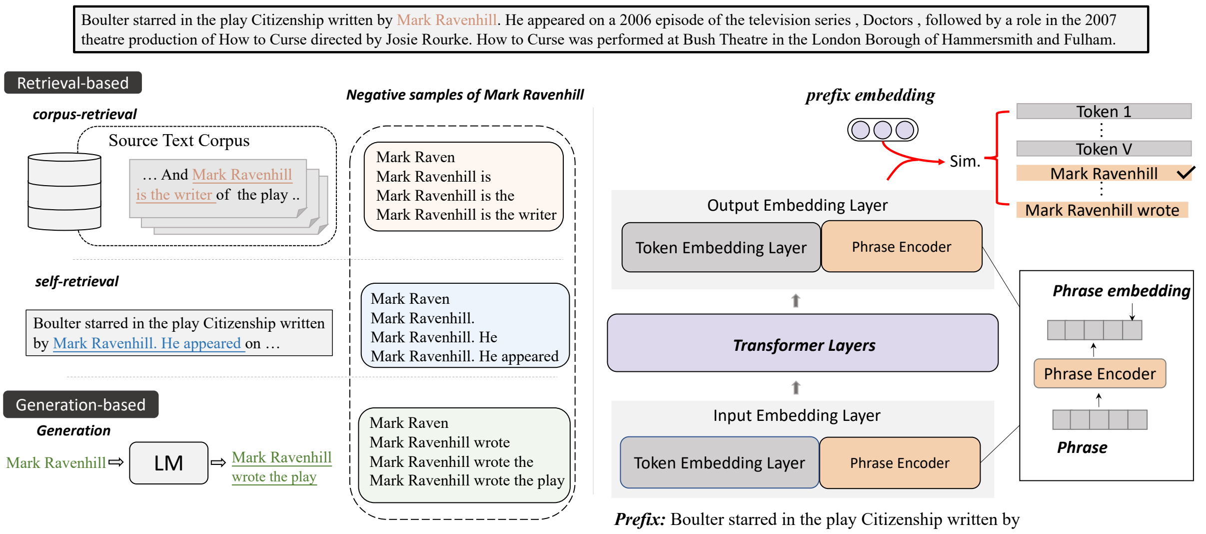

Figure 2: The overall architecture of our proposed dynamic vocabulary. During training, there are four sources of negative phrases: pre-batch, corpus-retrieval, self-retrieval, and generation. Phrases are embedded by the dynamic phrase encoder with an additional linear layer. The hidden layer of the last token serves as the phrase embedding. In the model input layer, phrases are treated as a basic brick without splitting into tokens.

workaround is to include only in-batch and pre-batch phrases in P (Gao et al., 2021). Unfortunately, it doesn't help learning the phrase encoder. Specifically, we find that the model struggles to correctly transit from a phrase token to an ordinary token and vice versa. More concretely, when predicting a phrase $p = w_1, w_2, ..., w_s$, the dynamic phrase encoder has trouble on distinguish p from 1) phrases which are prefixes of that phrase (e.g., $w_1 w_2$ and $w_1 w_2 w_3$) and 2) phrases which have p as their prefix (e.g., pw_{s+1} and $pw_{s+1}w_{s+2}$). Therefore, we also manually add the above phrases to P in each batch (we call them informative negative phrases).

For the first type, we can simply enumerate all prefixes of p. For the second type, we develop *retrieval-based* and *generation-based* methods for getting successor tokens of p,

- retrieval-based continuation finds appearances of p in a support corpus and takes p and its successor tokens there as negative phrases (**corpus-retrieval**). [7] One simplification is only considering p's successor tokens in the current sample (**self-retrieval**).

- generation-based continuation, instead of searching corpus, tries to get synthetic negative phrases by employing a language model. [8] The model is prompted with p and the following generations are included in P (**generation**).

Finally, regarding getting embeddings of these informative negative phrases, recall that we adopt an causal Transformer as the phrase encoder and use the hidden state of the final token to represent p, the embeddings of negative phrases could be efficiently obtained by feeding the longest phrase to the encoder.

Loss Functions The first part of the training loss is defined by Equation 1 (with negative samples added to P), which we denote by L_p. We also add a special setting of L_p in the loss (denoted by L_t), in which $P = \emptyset$ (i.e., the vanilla language modeling). It helps to maintain generation ability with the static vocabulary V.

We can further align the above two settings by requiring their next token distributions at each token position are close (measured by KL divergence). Concretely, given a sentence x, recall that (Section 2.1) the oracle of training L_p is $x'_1, x'_2, ..., x'_m$, the oracle of training L_t is $x_1, x_2, ..., x_n$. Assume a function σ which aligns x'_i to a token position in L_t's oracle: if x'_i is a token in V, it is mapped to the same token position, otherwise, x'_i is mapped

N-ids), phrases are built on the fly in the batching process, and there is no global training time P.

[7] Due to the time complexity of matching phrases, we only adopt corpus-retrieval when phrases are obtained by FMM, and keep the efficiency of Ngram phrases.

[8] Here we use GPT-2, stronger models can also be applied.

to its last token's position.

$$L_{kl} = \frac{1}{m}\sum_{i=0}^{m} \mathrm{KL}(\mathbb{P}(x_i'|x_{<i}')||\mathbb{P}(x_{\sigma(x_i')}|x_{<\sigma(x_i')})).$$

The final loss function is $L = L_p + L_t + L_{kl}$.

3 Experiments

3.1 Setups

Configurations For a fair comparison with baselines, we use GPT-2 (Radford et al., 2019) to initialize both the language model and dynamic phrase encoder. To collect phrases for each test sample, k related documents are retrieved by the semantic matching model, DPR (Karpukhin et al., 2020) and the vector search toolkit, FAISS (Johnson et al., 2019). In our paper, the value k is set to 32.

We experiment with several negative sampling and sample-building methods and set N-words with "self-retrieval + generation" as default. Besides, we initialize the language model with two models of different scales, GPT-2 and Tinyllama (Zhang et al., 2024), to verify the effectiveness of our proposed method. We employ full-parameter fine tuning for GPT-2 and LoRA (Hu et al., 2021) for Tinyllama. Please refer to Appendix B for more details.

Baselines We compare the proposed method with the following state-of-the-art models as baselines:

Transformer (Vaswani et al., 2023) is the standard token-level language model. We fine-tune the pre-trained GPT2 in our experiments.

KNN-LMs (Khandelwal et al., 2020) extends a pre-trained neural language model by linearly interpolating it with a k-nearest neighbors(KNN) model.

RETRO (Borgeaud et al., 2022) is a retrieval-enhanced transformer that combines a frozen Bert retriever, a differentiable encoder, and a chunked cross-attention mechanism.

CoG (Lan et al., 2023) decomposes text generation into a series of copy-and-paste operations. It first retrieves semantically relevant documents and then considers all n-grams within them as candidate phrases [9].

MWT (Gee et al., 2023) propose to expand vocabulary with top-k frequent n-grams in support corpus. Rather than expanding vocabulary dynamically, it still focuses on building a static vocabulary.

Metrics We use four automatic evaluation metrics to measure the quality of the generated texts (Lan et al., 2023; Cao et al., 2024),: (i) **MAUVE** (Pillutla et al., 2021) measures the distribution similarity between the reference text and generated text; (ii) **Rep-n** (Welleck et al., 2019) reflects the repetition at different n-gram levels in the generated text; (iii) **Diversity** (Welleck et al., 2019) evaluates the variety of generated content; and (iv) **Perplexity** measure the difficulty in predicting the next word in a sequence. In addition, we also compare the average time cost of different methods to decode a continuation consisting of 128 tokens given a prefix of 32 tokens, referred to as **latency**. The details for these metrics can be found in Appendix C

We investigate three applications: basic language modeling, domain adaptation, and generating citations for question answering.

3.2 Basic Language Modeling

We use GPT-2 and WikiText-103 (Merity et al., 2016) for evaluating open-end language generation. For each test sample, we provide the first 32 tokens as a context prefix, and both the baselines and our model will generate the subsequent 128 tokens (tokens are in GPT-2's original vocabulary).

The results are listed in Table 1. We find that,

- Regarding generation quality, language models with dynamic vocabulary can outperform standard Transformer with 5.22% MAUVE score (better fluency). Meanwhile, our model achieves 47.44% diversity, which is much better than other baselines.

- Regarding generation efficiency, dynamic vocabulary achieves the best latency. The reason is that a single phrase contains several tokens, which translates to fewer decoding steps for a given decoding length budget.

- the perplexity of dynamic vocabulary (our model and CoG) is higher than that of the Transformer. This discrepancy could potentially stem from the fact that during testing, the input prefixes are strictly composed of tokens from a fixed vocabulary, whereas the model is not subjected to such constraints during training, which results in an inconsistency between the training and testing data distributions, potentially leading to the observed difference in perplexity scores.

[9] CoG adopts a two-stage search strategy (document retrieval followed by phrase extraction) while CoG-2 (Cao et al., 2024) generates text directly through phrase retrieval. However, CoG-2 fails to provide any code, thus precluding any comparative analysis.

Model	MAUVE ↑	Rep-2 ↓	Rep-3 ↓	Rep-4 ↓	Diversity ↑	Latency(s)↓	PPL ↓
Transformer	20.47	41.96	36.82	33.74	24.30	1.10	3.60
RETRO	19.59	43.78	38.58	35.35	22.33	4.43	3.96
KMM-LM*	19.92	43.79	38.76	35.69	22.13	10.36	3.48
CoG	21.61	34.77	30.67	28.35	32.41	1.04	7.89
MWT	24.74	33.78	26.72	22.76	37.48	1.13	5.58
Ours	**25.69**	**27.77**	**20.80**	**17.08**	**47.44**	**0.99**	8.03

Table 1: The automatic evaluation on the test set of WikiText-103. $*$ indicates that we directly utilize the results from the CoG paper for KNN-LM due to limited GPU memory. Additionally, our method retrieves only 32 documents for phrase segments during evaluation, whereas CoG retrieves 1024. MWT Gee et al. (2023) apply MWT to encoder-only model but we implement MWT with GPT-2.

Ours versus ($*$)	Better	No Prefer	Worse
Human Evaluation			
Transformer	0.57	0.22	0.21
MWT	0.55	0.21	0.24
CoG	0.53	0.22	0.25
GPT-4 Evaluation			
Transformer	0.61	0.05	0.34
MWT	0.58	0.02	0.40
CoG	0.58	0.08	0.34

Table 2: Human evaluation and GPT-4 evaluation on WikiText-103. "Better" represents that our proposed model's output is superior; "No prefer" indicates that the performance is comparable; and "worse" denotes that our model's output is inferior.

We also evaluate the generation results under nucleus sampling and attempt real-time adaptability. The details can be found in Appendix A, D separately.

Human and GPT-4 Evaluation To gain further assessment, we also run human evaluation on a random sample of 100 generations. For each test sample prefix, the annotators are given two continuations generated by the baseline and our model respectively in random order. Annotators are asked to choose which one is better (in terms of fluency, coherence, informativeness, and grammar). When annotators make different decisions on the same sample, they will discuss and make the final decision. As shown in Table 2, dynamic vocabulary outperforms the Transformer with better cases of 57 and 21 cases of slight inferiority and the results are consistent with MAVUE, which shows that the model with dynamic vocabulary processes a stronger generation capability.

We also employ GPT-4 (Achiam et al., 2023) for further assessment. Detailed implementations and prompts are in Appendix F. The results are consistent with the above two evaluations.

Sequence Compression Sequence compression reflects the length of text that a model can accommodate within the same window size. Following Dagan et al. (2024), we measure the two compression metrics, normalized sequence length (NSL) and the average number of Bytes per Token. NSL is the token count of a tokenized sequence from the tokenizer T. Given that our model does not incorporate a genuine tokenizer, we take the outputs of each decoding step as the tokenization results. We report scores from tokenizers of GPT and MWT on our model's outputs.

Model	NLS ↓	UTF-8 Bytes ↑
Transformer	127.72	4.28
MWT	114.84	4.77
Ours	**101.38**	**5.54**

Table 3: Compression on WikiText-103. Since CoG, KNN-LM, and RETRO do not modify the model's tokenizer or input vocabulary, the compression results are the same with the Transformer.

As shown in the table 3, our proposed model holds the highest information content per token, averaging 101.38 tokens or phrases per sequence and 5.54 UTF-8 bytes per token, and necessitates fewer tokens or phrases to generate the same text. This is a natural consequence of the fact that the dynamically added phrases contain more tokens.

Scale Up To test our method on recently popular LMs, we deploy the dynamic vocabulary with TinyLlama (Zhang et al., 2024), which is a 1.1B LLaMA-style backbone. The results in Table 4 are consistent with the experimental conclusion above.

3.3 The Influence of Negative Phrases

As discussed, we designed several negative sampling strategies. We study their influence in Table 5. We observe that the choice of the negative phrases

Model	MAUVE ↑	Diversity ↑	Latency(s)↓	PPL ↓
TinyLlama	20.64	32.53	4.92	5.20
Ours	**22.54**	**53.99**	**3.82**	12.88

Table 4: The automatic evaluation on the test set of WikiText-103. In this experiments, we use GPT-2 and TinyLlama to initialize the dynamic phrase encoder and the language model, respectively. We utilize parameter-efficient fine-tuning approach-LoRA on TinyLlama and set r, alpha, dropout as 8, 32, 0.1, separately.

Negative Samples	MAUVE ↑	Diversity ↑	PPL
FMM			
in-batch	21.95	**57.92**	16.48
in-batch + pre-batch	22.28	48.91	9.02
generation	**22.87**	42.19	**6.34**
corpus-retrieval	21.98	41.32	6.40
self-retrieval	21.65	41.67	6.39
self-retrieval + generation	21.25	42.40	6.62
N-words			
in-batch	24.67	**64.15**	17.01
in-batch + pre-batch	23.98	61.80	14.60
generation	24.99	49.03	8.51
self-retrieval	24.83	48.46	8.13
self-retrieval + generation	**25.69**	47.44	**8.03**
N-ids			
in-batch	23.96	**68.44**	21.53
in-batch + pre-batch	23.66	61.16	14.83
generation	23.91	46.40	**8.07**
self-retrieval	23.64	48.38	8.36
self-retrieval + generation	**24.85**	47.08	8.21

Table 5: The automatic evaluation on different negative samples and training samples. During testing, each phrase is constrained to 2-8 tokens. Here, the pre-batch method contains prefixes of gold phrases as well and the number of preceding batches is set to 1.

method significantly impacts the fluency and quality of the generated text.

Specifically, compared with the other negative sampling methods, the in-batch and pre-batch negative sampling methods result in a markedly higher PPL (approximately 10 points and 3 points higher in the FMM setting) [10]. This observation suggests that including negative phrases that are prefixes of gold phrases is crucial for generating fluent text.

Regarding generation-based and retrieval-based negative phrases, there is no significant performance difference. Since generating negative samples requires additional time costs, one may prefer self-retrieval. More analyses can be found in Appendix E.

[10] We have observed that there is a positive correlation between Diversity and PPL, which means that the higher the Diversity, the higher the PPL values tend to be as well. We believe that this phenomenon occurs because the model tends to increase the probability of repeating previous sentences (Xu et al., 2022), leading to a lower PPL and Diversity.

Model	MAUVE ↑	Diversity ↑	Latency(s)↓	PPL
Transformer w/o FT	22.97	72.12	1.03	**3.21**
Transformer w/ FT	23.06	80.21	**1.02**	3.54
RETRO	19.07	72.68	5.72	3.78
KMM-LM*	23.32	19.85	-	-
CoG	19.46	81.93	1.39	6.74
MWT	24.55	77.45	1.10	5.38
Ours	**26.35**	**82.99**	1.09	7.61

Table 6: The automatic evaluation on Law-MT. In this experiment, we retrieve 512 documents for each sample. To guarantee a fair comparison, we also evaluate the performance of the Transformer model both with and without further fine-tuning on LawMT.

3.4 Domain Adaptation

The plug-and-play property of dynamic phrase encoder motivates us to explore the performance on a different domain in a training-free manner. Specifically, we investigate the model trained on the WikiText-103 dataset while tested on LawMT (Koehn and Knowles, 2017) dataset which is an English-German translation dataset in the legal domain. Following (He et al., 2021a; Alon et al., 2022; Lan et al., 2023), we treat the English portion of this dataset as a retrieval corpus. As shown in table 6, only equipped with dynamic vocabulary extracted on the target domain, the model can outperform the transformer fine-tuned on LawMT datasets (3.29% on MAUVE and 2.78% Diversity). Thus, the learned phrase encoder could be an efficient tool for lightweight domain generalization. We also calculate the sequence compression ratio and conduct GPT-4 Evaluation. The details are in Appendix F, G.

3.5 Generation with Citations

Given that we can construct the dynamical vocabulary as needed, and that each candidate phrase corresponds to one document, our proposed model could be utilized for citation generation. The task is formalized as follows: given a query q and a few documents D, the model is required to generate an answer with embedded in-line citations of documents in D. We run the experiments on the long-form QA dataset, ASQA (Stelmakh et al., 2022) further processed by Gao et al. (2023), where candidate documents for each query have already been retrieved. We first label each document with a unique ID marker starting from 1 and then extract phrases with the corresponding marker, such as "dynamic vocabulary[1]". Therefore, phrases in the generated answers could reflect the citation process.

Model(shot-1)	Citation_rec	Citation_prec	QA-EM	QA-F1	Rouge-L
TinyLlama	0.62	1.54	6.00	8.78	25.43
ours					
w/ n-grams	**9.76**	**29.30**	8.88	11.83	30.06
w/ parsing	2.94	9.17	**9.87**	**13.06**	**30.16**
w/o phrases	0.20	0.44	8.81	11.81	29.60

Table 7: The automatic evaluation on ASQA. In this experiment, we opt for TinyLlama as the language model to imbue the model with in-context learning capabilities. All baseline models are configured in a one-shot setting, with the number of candidate documents set to 3. Parsing denotes that we use Stanza parser (Qi et al., 2020) to extract phrases from candidate documents, which ensures that the phrases possess a relatively complete and well-defined meaning.

Results We evaluate the generated results from two perspectives: citation quality and QA answer accuracy. The detailed definitions of the metrics can be found in Gao et al. (2023). We provide the model with the k documents and leverage in-context learning to instruct it to cite accordingly.

The results demonstrate a significant boost in the citation capability of our model with citation recall and precision surpassing TinyLlama baseline by 9.14% and 27.76%, respectively. However, phrase collections have a significant impact on the citation results. The phenomenon occurs potentially due to the extensive collection of phrases by the n-grams approach and thus more suitable phrases could align with the generated text.

Furthermore, our model exhibits a superior QA performance with an EM score of 9.87% and an F1 of 13.06%. Due to our model's further training on WikiText-103 and the property that responding to query in ASQA necessitates Wikipedia-based information, our model's QA performance is expected to be excellent with the absence of phrases (i.e., ours w/o phrases).

4 Related Work

Tokenizer Tokenizer is an essential component of language models (Dagan et al., 2024; Mielke et al., 2021), responsible for transforming raw text into a sequence of tokens. Byte-Pair Encoding (BPE) is commonly used to build tokenizer (Radford et al., 2019; Liu et al., 2019; Lewis et al., 2019; He et al., 2021b) and, there exist other tokenization algorithms, such as Unigram (Kudo, 2018) and WordPiece tokenization used in BERT (Devlin et al., 2019). However, these tokenizations are limited to subwords or whole words. Kumar and Thawani (2022) and Gee et al. (2023) generalize the BPE algorithm to multi-words and multi-tokens separately. Whereas these approaches necessitate training the tokenizer and remain static.

CoG (Lan et al., 2023) and CoG-2 (Cao et al., 2024) both employ a "dynamic vocabulary" by expanding vocabulary with phrases extracted from related documents. However, these two methods only employ dynamic vocabulary in the output module and split phrases into tokens in the input. In this paper, we treated phrases as atomic units same as tokens, and dynamically expanded vocabulary both in input and output layers.

Sequence Compression Language models are constrained by the limited length of input sequences they can process. Increasing this length results in a prohibitive computational overhead. A series of techniques have been proposed to compress sentences into one or a few tokens or latent representations (Qin and Van Durme, 2023; Chevalier et al., 2023; Bulatov et al., 2022; Mu et al., 2024). MWT (Gee et al., 2023) enhances compression by retraining the tokenizer, incorporating the most frequent n-grams of a support corpus into the vocabulary. In contrast to the static vocabulary of MWT, our method dynamically adapts the model's vocabulary to the input text, resulting in a more flexible and efficient adaptation.

5 Conclusion

In this paper, we propose a novel approach for dynamically adjusting the model's vocabulary based on input text. It is a plug-and-play approach that can be performed simultaneously with pre-training tasks. We investigated standard language modeling, domain adaptation, and citation generation, and discussed the impact of different training samples and negative phrase construction methods on the quality of generated text. Our experimental results show that our proposed model can rapidly generate high-quality, high-compression text compared to baselines.

6 Limitations

In this paper, we propose a method to dynamically expand the vocabulary based on the input text. While our approach can improve generation speed and increase the effective length of the generated text, our model does not modify the underlying tokenizer. As a result, it cannot reduce the token numbers for known input information like prompts or questions. The dynamic vocabulary is, therefore, limited to the subsequent content generated by the model.

Furthermore, to obtain embedding representations for phrases, a dynamic phrase encoder is necessary. This encoder has a more intricate structure compared to the model's linear embedding layer and requires additional memory allocation during implementation.

Lastly, our method relies on external techniques, such as a retriever, to obtain relevant documents and extract phrases from them during testing. This adds complexity to the preparation process.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Uri Alon, Frank F. Xu, Junxian He, Sudipta Sengupta, Dan Roth, and Graham Neubig. 2022. Neuro-symbolic language modeling with automaton-augmented retrieval. *Preprint*, arXiv:2201.12431.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. 2022. Improving language models by retrieving from trillions of tokens. *Preprint*, arXiv:2112.04426.

Aydar Bulatov, Yuri Kuratov, and Mikhail Burtsev. 2022. Recurrent memory transformer. In *Advances in Neural Information Processing Systems*.

Bowen Cao, Deng Cai, Leyang Cui, Xuxin Cheng, Wei Bi, Yuexian Zou, and Shuming Shi. 2024. Retrieval is accurate generation. *Preprint*, arXiv:2402.17532.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2022. Finqa: A dataset of numerical reasoning over financial data. *Preprint*, arXiv:2109.00122.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. 2023. Adapting language models to compress contexts. *Preprint*, arXiv:2305.14788.

Gautier Dagan, Gabriel Synnaeve, and Baptiste Rozière. 2024. Getting the most out of your tokenizer for pre-training and domain adaptation. *Preprint*, arXiv:2402.01035.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Preprint*, arXiv:1810.04805.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen tau Yih, Luke Zettlemoyer, and Mike Lewis. 2023. Incoder: A generative model for code infilling and synthesis. *Preprint*, arXiv:2204.05999.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. 2023. Enabling large language models to generate text with citations. *Preprint*, arXiv:2305.14627.

Leonidas Gee, Leonardo Rigutini, Marco Ernandes, and Andrea Zugarini. 2023. Multi-word tokenization for sequence compression. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*. Association for Computational Linguistics.

Leonidas Gee, Andrea Zugarini, Leonardo Rigutini, and Paolo Torroni. 2022. Fast vocabulary transfer for language model compression. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 409–416, Abu Dhabi, UAE. Association for Computational Linguistics.

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

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021b. Deberta: Decoding-enhanced bert with disentangled attention. *Preprint*, arXiv:2006.03654.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. 2020. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Preprint*, arXiv:2009.13081.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. *Preprint*, arXiv:1911.00172.

Hannah Kirk, Abeba Birhane, Bertie Vidgen, and Leon Derczynski. 2022. Handling and presenting harmful text in NLP research. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 497–510, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. *Preprint*, arXiv:1706.03872.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. *Preprint*, arXiv:1804.10959.

Dipesh Kumar and Avijit Thawani. 2022. Bpe beyond word boundary: How not to use multi word expressions in neural machine translation. In *Proceedings of the Third Workshop on Insights from Negative Results in NLP*, pages 172–179.

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *Preprint*, arXiv:1901.07291.

Tian Lan, Deng Cai, Yan Wang, Heyan Huang, and Xian-Ling Mao. 2023. Copy is all you need. *Preprint*, arXiv:2307.06962.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *Preprint*, arXiv:1910.13461.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation. *Preprint*, arXiv:2001.08210.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *Preprint*, arXiv:1907.11692.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. *Preprint*, arXiv:1711.05101.

Jacob Menick, Maja Trebacz, Vladimir Mikulik, John Aslanides, Francis Song, Martin Chadwick, Mia Glaese, Susannah Young, Lucy Campbell-Gillingham, Geoffrey Irving, and Nat McAleese. 2022. Teaching language models to support answers with verified quotes. *Preprint*, arXiv:2203.11147.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. 2021. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp. *Preprint*, arXiv:2112.10508.

Maxim Milakov and Natalia Gimelshein. 2018. Online normalizer calculation for softmax. *Preprint*, arXiv:1805.02867.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. 2024. Learning to compress prompts with gist tokens. *Preprint*, arXiv:2304.08467.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. 2021. Mauve: Measuring the gap between neural text and human text using divergence frontiers. *Preprint*, arXiv:2102.01454.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Guanghui Qin and Benjamin Van Durme. 2023. Nugget: neural agglomerative embeddings of text. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Felix Stahlberg. 2020. Neural machine translation: A review. *Journal of Artificial Intelligence Research*, 69:343–418.

Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. 2022. ASQA: Factoid questions meet long-form answers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8273–8288, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Ryan Teehan, Brenden Lake, and Mengye Ren. 2024. College: Concept embedding generation for large language models. *arXiv preprint arXiv:2403.15362*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need. *Preprint*, arXiv:1706.03762.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. 2023. Large language models are not fair evaluators. *Preprint*, arXiv:2305.17926.

Laura Weidinger, John Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, Zac Kenton, Sasha Brown, Will Hawkins, Tom Stepleton, Courtney Biles, Abeba Birhane, Julia Haas, Laura Rimell, Lisa Anne Hendricks, William Isaac, Sean Legassick, Geoffrey Irving, and Iason Gabriel. 2021. Ethical and social risks of harm from language models. *Preprint*, arXiv:2112.04359.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. 2019. Neural text generation with unlikelihood training. *Preprint*, arXiv:1908.04319.

Jin Xu, Xiaojiang Liu, Jianhao Yan, Deng Cai, Huayang Li, and Jian Li. 2022. Learning to break the loop: Analyzing and mitigating repetitions for neural text generation. *Preprint*, arXiv:2206.02369.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *Preprint*, arXiv:2401.02385.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

A Full Results

We show the full results of our experiments in Tables 8, 9, 10, 11.

B More Implementation Details

The training of our proposed model was carried out on two NVIDIA RTX 3090 GPUs, each with 24GB of memory, over a total of 400,000 training steps. During the training process, we implemented a gradient accumulation step of 2, with a batch size of 4. We also used a linear learning rate schedule with a warmup, alongside the AdamW optimizer (Loshchilov and Hutter, 2019), maintaining the default beta values. The initial learning rate was set at 5e-5. Additionally, we applied gradient clipping with a clipping value of 1.0 to ensure training stability. When conducting nucleus sampling, we set the p to 0.95.

For each test sample, we retrieve top-k documents that have similar topics with the sample prefix and extract candidate phrases to construct the dynamic. In our experiments, the value of k is set to 32 by default and the candidate phrase is restrained to the length of 2-8 tokens.

We initialize the language model with two models of different scales, GPT-2 and Tinyllama (Zhang et al., 2024), to verify the effectiveness of our proposed method. We employ full-parameter fine-tuning for GPT-2 and LoRA fine-tuning (Hu et al., 2021) for Tinyllama. When fine-tuning TinyLlama with LoRA, we set r as 8 and alpha as 32.

The experiments of MWT in paper (Gee et al., 2023) were conducted on encoder-only models such as BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019). In our implementation, we modified the foundation model to GPT2 (Radford et al., 2019), a decoder-only model, and added the top 10000 most frequent 2-grams to the original GPT2 Tokenizer. The embeddings for newly added words were initialized using Fast Vocabulary Transfer(FVT) (Gee et al., 2022). MWT was trained for a total of 150000 steps on the WikiText103 dataset.

C More Details of Automatic Evaluation

In this section, we provide a detailed introduction to the automatic evaluation metrics.

- **MAUVE** Pillutla et al. (2021) measures how closely the token distribution in the generated text matches that in human-written text across the entire test set. We follow prior work and leverage the GPT2-large model to generate the scores. In our implementation, the scaling factor is set as 2.0.

- **Rep-n** Welleck et al. (2019) measures the repetition at different n-gram levels in the generated text. It is defined as $100 \times (1.0 - \frac{|uniquen-gram(x)|}{|totaln-gram(x)|})$. Higher Rep-n represents the severe degeneration problem in generations.

- **Diversity** Welleck et al. (2019) evaluates the variety of generated content, which is formulated as $\prod_{n=2}^{4}(1 - \frac{Rep-n}{100})$. More informative generations get higher Diversity scores.

- **Perplexity** is a measure of the uncertainty or difficulty in predicting the next word in a sequence. A lower perplexity score indicates that the model is more certain about its predictions.

D Real-time Adaptability

We have attempted to verify the efficiency when the proposed model adapts its vocabulary in real-time scenarios where new phrases continuously emerge. We give a simulated experiment with dynamic vocabulary updates in real time. Specifically, we first use a document retriever to retrieve top-k-related documents for each given prefix. Then, the candidate phrases P are collected from these documents for selection. Unlike the full off-line computation (the setting in section 3.2), we gradually expand the vocabulary during the model's generation. Specifically, we added 5% of the phrases from P to the vocabulary for every 10 tokens generated.

Obviously, the computational and memory costs are linear to the size of on-demand vocabularies, which we believe is reasonable since 1) the encoding of phrases could be computed in the way of parallel and off-line; 2) the prediction over the new phrase table could also be paralleled using the tilling trick (Milakov and Gimelshein, 2018); 3) in practice, the size of dynamic vocabulary could be controlled by dynamically off-loading unused phrases. As shown in table 12, the increase in latency can be successfully controlled.

E The Influence of Negative Phrases

We experimented with several negative sampling methods, including in-batch, pre-batch, generation-based, corpus-retrieval (only in FMM), and self-retrieval.

Model	Decoding	MAUVE ↑	Rep-2 ↓	Rep-3 ↓	Rep-4	Diversity ↑	Latency(s)↓	PPL
Transformer	greedy	20.47	41.96	36.82	33.74	24.30	1.10	3.60
	nucleus	25.05	5.40	1.44	0.51	92.76	1.15	31.01
RETRO	greedy	19.59	43.78	38.58	35.35	22.33	4.43	3.96
	nucleus	20.77	5.83	1.91	0.83	91.61	5.43	39.74
KMM-LM*	greedy	19.92	43.79	38.76	35.69	22.13	10.36	3.48
	nucleus	22.50	3.33	0.69	0.21	95.8	10.42	78.01
CoG	greedy	21.61	34.77	30.67	28.35	32.41	1.04	7.89
	nucleus	25.96	5.43	1.53	0.67	92.50	1.06	36.66
GPT+MWT	greedy	24.74	33.78	26.72	22.76	37.48	1.13	5.58
	nucleus	25.66	4.18	0.90	0.29	94.68	1.17	55.02
Ours	greedy	25.69	27.77	20.80	17.08	47.44	0.99	8.03
	nucleus	24.34	4.59	1.03	0.28	94.16	1.00	51.38

Table 8: The automatic evaluation on the test set of WikiText-103. $*$ denotes that the results are obtained from CoG (Lan et al., 2023) paper. For each sample, the first 32 tokens are provided and models are tasked with generating the subsequent 128 tokens. We can observe that our proposed model achieves the best scores in most metrics.

Negative Samples	Decoding	MAUVE ↑	Rep-2 ↓	Rep-3 ↓	Rep-4	Diversity ↑	Latency(s)↓	PPL
FMM								
in-batch	greedy	21.95	23.42	15.29	10.71	57.92	0.94	16.48
	nucleus	23.17	4.17	0.92	0.29	94.67	0.84	78.20
pre-batch	greedy	22.28	26.90	20.07	16.29	48.91	0.95	9.02
	nucleus	20.59	4.62	1.07	0.35	94.03	0.88	56.28
generation	greedy	22.87	31.17	23.82	19.55	42.19	1.20	6.34
	nucleus	20.33	4.35	1.01	0.31	94.39	1.06	49.51
corpus-retrieval	greedy	21.98	31.47	24.39	20.26	41.32	1.12	6.40
	nucleus	20.52	4.36	1.00	0.32	94.38	1.08	51.60
self-retrieval	greedy	21.65	31.33	24.15	20.00	41.67	1.15	6.39
	nucleus	20.63	4.37	1.00	0.35	94.34	1.04	49.93
self-retrieval + generation	greedy	21.25	30.89	23.73	19.57	42.40	1.16	6.62
	nucleus	20.34	4.24	0.96	0.29	94.57	1.04	52.27
N-words								
in-batch	greedy	24.67	20.80	12.22	7.72	64.15	0.88	17.01
	nucleus	24.24	4.76	1.16	0.40	93.76	0.81	68.25
pre-batch	greedy	23.98	19.58	13.63	11.02	61.80	1.16	14.60
	nucleus	23.60	5.71	1.82	0.92	91.73	1.11	47.17
generation	greedy	24.99	26.72	19.95	16.41	49.03	0.94	8.51
	nucleus	24.85	4.64	1.07	0.31	94.04	0.94	50.65
self-retrieval	greedy	24.83	27.21	20.23	16.54	48.46	0.96	8.13
	nucleus	24.51	4.57	1.05	0.33	94.12	0.94	51.85
self-retrieval + generation	greedy	25.69	27.77	20.80	17.08	47.44	0.99	8.03
	nucleus	24.34	4.59	1.03	0.28	94.16	1.00	51.38
N-ids								
in-batch	greedy	23.96	18.63	10.30	6.22	68.44	0.81	21.53
	nucleus	23.17	4.77	1.18	0.43	93.71	0.70	81.06
pre-batch	greedy	23.66	19.81	13.96	11.36	61.16	1.12	14.83
	nucleus	22.84	5.17	1.52	0.67	92.77	0.92	54.52
generation	greedy	23.91	28.12	21.45	17.82	46.40	0.99	8.07
	nucleus	24.50	4.41	0.97	0.29	94.38	0.96	53.98
self-retrieval	greedy	23.64	27.29	20.33	16.49	48.38	1.02	8.36
	nucleus	23.85	4.43	0.94	0.27	94.41	0.88	55.76
self-retrieval + generation	greedy	24.85	27.85	21.04	17.36	47.08	1.01	8.21
	nucleus	23.91	4.41	0.96	0.28	94.40	0.98	53.03

Table 9: The automatic evaluation on different negative samples with greedy and nucleus sampling (top-p: 0.95) decoding algorithms on the WikiText103 dataset. The constructions of training samples and negative phrases have a significant influence on the generated text.

Model	Decoding	MAUVE ↑	Rep-2 ↓	Rep-3 ↓	Rep-4	Diversity ↑	Latency(s)↓	PPL
Transformer w/o FT	greedy	22.97	13.36	9.69	7.84	72.12	1.03	3.21
	nucleus	24.15	4.05	1.62	0.80	93.64	1.05	31.48
Transformer w/ FT	greedy	23.06	9.74	6.45	5.00	80.21	1.02	3.54
	nucleus	25.12	4.36	1.73	0.87	93.17	1.08	14.94
RETRO	greedy	19.07	13.19	9.34	7.66	72.68	5.72	3.78
	nucleus	21.26	3.30	1.18	0.55	95.03	5.54	57.40
KMM-LM*	greedy	23.32	-	-	-	19.85	-	-
	nucleus	24.75	-	-	-	94.60	-	-
CoG	greedy	19.46	9.29	5.68	4.24	81.93	1.39	6.74
	nucleus	24.45	4.57	1.58	0.72	93.25	0.89	32.01
GPT+MWT	greedy	24.55	11.59	7.34	5.46	77.45	1.10	5.38
	nucleus	22.68	3.15	1.01	0.39	95.49	1.16	68.55
Ours	greedy	26.35	9.26	5.21	3.52	82.99	1.09	7.61
	nucleus	24.80	3.63	1.17	0.48	94.78	0.93	60.70

Table 10: The automatic evaluation on LawMT. We directly retrieve 512 documents for each sample in this experiment. Our proposed model even outperforms the Transformer further fine-tuned on the LawMT corpus.

Negative Samples	Decoding	MAUVE ↑	Rep-2 ↓	Rep-3 ↓	Rep-4	Diversity ↑	Latency(s)↓	PPL
FMM								
pre-batch	greedy	23.65	9.39	5.00	3.03	83.48	0.90	13.86
	nucleus	22.73	4.82	1.87	0.85	92.60	0.84	68.31
pre-batch	greedy	25.00	8.71	4.76	3.16	84.20	0.98	8.26
	nucleus	23.19	3.71	1.19	0.50	94.66	0.83	60.34
generation	greedy	22.87	11.00	6.76	4.85	78.96	1.26	6.17
	nucleus	22.50	3.50	1.13	0.48	94.95	1.07	65.26
Retrieval-samples	greedy	23.00	10.45	6.36	4.53	80.06	1.21	6.11
	nucleus	23.24	3.43	1.01	0.46	95.07	1.02	68.26
self-retrieval	greedy	23.41	10.98	6.80	4.92	78.89	1.20	6.11
	nucleus	23.22	3.48	1.05	0.43	95.10	0.98	67.14
self-retrieval + generation	greedy	24.15	10.50	6.31	4.49	80.08	1.22	6.24
	nucleus	22.55	3.40	1.16	0.53	94.98	1.04	69.40
N-words								
in-batch	greedy	24.27	10.07	5.31	3.16	82.47	0.86	15.28
	nucleus	25.48	5.36	2.12	1.00	91.71	0.80	61.90
pre-batch	greedy	26.15	6.53	3.11	1.92	88.82	0.61	14.40
	nucleus	25.15	4.07	1.41	0.61	94.00	0.53	45.79
generation	greedy	26.35	9.26	5.21	3.52	82.99	1.09	7.61
	nucleus	24.66	3.53	1.16	0.48	94.89	0.92	62.58
self-retrieval	greedy	23.65	8.92	4.88	3.29	83.87	1.04	8.05
	nucleus	24.71	3.54	1.09	0.42	95.00	0.81	62.51
self-retrieval + generation	greedy	26.35	9.26	5.21	3.52	82.99	1.09	7.61
	nucleus	24.80	3.63	1.17	0.48	94.78	0.93	60.70
N-ids								
in-batch	greedy	25.77	9.12	4.44	2.47	84.70	0.81	17.49
	nucleus	26.04	5.19	2.06	0.95	91.98	0.70	66.18
pre-batch	greedy	25.08	6.70	3.14	1.87	88.68	0.62	14.49
	nucleus	23.93	4.25	1.46	0.65	93.74	0.43	47.94
generation	greedy	22.55	9.24	5.21	3.55	82.98	1.04	8.03
	nucleus	23.14	3.59	1.14	0.49	94.85	0.85	61.89
self-retrieval	greedy	24.63	9.46	5.43	3.71	82.44	1.05	7.86
	nucleus	24.19	3.58	1.11	0.44	94.94	0.78	63.87
self-retrieval + generation	greedy	23.18	9.31	5.25	3.59	82.85	1.07	7.57
	nucleus	24.63	3.57	1.10	0.46	94.93	0.87	60.32

Table 11: The automatic evaluation on different negative samples with greedy decoding and nucleus sampling(top-p: 0.95) on the LawMT dataset.

Settings	MAUVE ↑	Diversity ↑	Latency(s)↓	PPL ↓
Ours(70)	25.27	46.11	1.03	7.78
Ours(70) + real-time	24.42	47.05	1.31	7.99
Ours(100)	25.69	47.44	0.99	8.04

Table 12: The results of real-time adaptability. (x) represents that we construct dynamic vocabulary with $x\%$ of P and real-time denotes the real-time scenarios.

The results indicate that strong negative phrases are crucial for the model's generation quality (by comparing vanilla in-batch and pre-batch negative phrases with the proposed generation-based and retrieval-based negative phrases).

Regarding generation-based and retrieval-based negative phrases, there is no significant performance difference. However, generation-based and corpus-retrieval methods require additional time costs compared to self-retrieval, as the generation-based approach necessitates the continuation of phrase generation, and corpus-retrieval requires retrieving from the related corpus. Self-retrieval may be optimal in this perspective.

Above all, the results indicate that strong negative phrases are crucial for the model's generation quality and the ranking of the aforementioned methods may be: *self-retrieval* > *generation-based* ≈ *corpus-retrieval* > *pre-batch* ≈ *in-batch.*

Furthermore, among all negative phrases, PPL of the FMM setting is consistently lower than that of the N-words and N-ids methods. This phenomenon occurs because phrases obtained with FMM possess a relatively clear meaning.

Interestingly, the average MAUVE values for the N-words and N-ids are approximately 1% higher than that of FMM. The result indicates that using different phrases in training time has a substantial influence on the text quality.

F GPT-4 Evaluation

Although human evaluation is considered the gold standard for assessing human preferences, it is slow and costly. Zheng et al. (2023) have demonstrated that strong LLMs, such as GPT-4, can match most human preferences well , achieving over 80% agreement, which is the same level of agreement between humans. Therefore, LLM-as-a-judge is an interpretable approach to approximating human preferences. We random sample 100 cases and evaluate the results of the Baselines and our model. GPT-4 is asked to evaluate the generated texts by considering fluency, coherence, informativeness, and grammar. Owing to GPT4's sensitivity to the order of the two candidate sentences (Wang et al., 2023), we adhere to the approach employed in Wang et al. (2023) and determine the final result by calculating the average of the outcomes from interchanging the order of the candidate sentences.

Figure 3 shows the detailed prompt used for GPT-4. Despite the template emphasizing that the order should not affect the results (red text), large language models still exhibit a significant positional bias. Therefore, for each triplet (prefix, <generation_1>, <generation_2>), we include another corresponding triplet (prefix, <generation_2>, <generation_1>). This is done to mitigate the impact of the order of the two generations on GPT-4 evaluation.

Here are the full results of our evaluation using GPT-4 shown in Table 13. It can be seen that our model is capable of producing generations that are comparable or even superior to the baselines.

Comparison (VS)	Better	No Prefer	Worse
WikiText103			
Transformer	0.61	0.05	0.34
MWT	0.58	0.02	0.40
CoG	0.58	0.08	0.34
LawMT			
Transformer	0.46	0.02	0.52
MWT	0.67	0.07	0.26
CoG	0.50	0.05	0.45

Table 13: GPT-4 evaluation on WikiText-103. Due to the sensitivity of GPT-4 to the order of two candidates, we got the final result by calculating the average scores by changing the order of the two candidates.

G Sequence Compression On LawMT

Analogous to the section 3.2, we calculate the compression ratio of LawMT. The conclusion aligns with those from section 3.2, indicating that our model could yield the highest information density per token. And for an equal number of tokens, our model encompasses a longer effective text length.

You are a helpful and precise assistant for checking the quality of the text.
[Prefix]
{prefix}
[The Start of Assistant 1's Generation]
{Generation_1}
[The End of Assistant 1's Generation]
[The Start of Assistant 2's Generation]
{Generation_2}
[The End of Assistant 2's Generation]
[System]

We would like to request your feedback on the performance of two AI assistants in response to the user prefix displayed above.Please rate the fluency, coherence, informativeness, and grammar. Each assistant receives an overall score on a scale of 1 to 10, where a higher score indicates better overall performance.
Please first provide a comprehensive explanation of your evaluation, avoiding any potential bias and ensuring that the order in which the responses were presented does not affect your judgment. Then, output two lines indicating the scores for Assistant 1 and 2, respectively.

Output with the following format:
Evaluation evidence: <your evluation explanation here>
Score of the Assistant 1: <score>
Score of the Assistant 2: <score>

Figure 3: The GPT-4 evaluation template with three slot {prefix}, {Generation_1} and {Generation_2}.

Model	NLS	UTF-8 Bytes
WikiText103		
Transformer	127.72	4.28
MWT	114.84	4.77
Ours	101.38	5.54
LawMT		
Transformer	128.79	5.22
MWT	124.94	5.39
Ours	105.38	6.53

Table 14: Compression on WikiText-103 and LawMT. Our model compresses text in a larger margin than MWT in the specific domain.