

Scaling the Vocabulary of Non-autoregressive Models for Fast Generative Retrieval

Ravisri Valluri*[†]
Microsoft Research
Bengaluru, Karnataka, India
ravisrivk@gmail.com

Akash Kumar Mohankumar
Microsoft
Bengaluru, Karnataka, India
makashkumar@microsoft.com

Kushal Dave
Microsoft
Mountain View, CA, USA
kudave@microsoft.com

Amit Singh
Microsoft
Bengaluru, Karnataka, India
siamit@microsoft.com

Jian Jiao
Microsoft
Bellevue, WA, USA
jian.jiao@microsoft.com

Manik Varma
Microsoft Research
Bengaluru, Karnataka, India
manik@microsoft.com

Gaurav Sinha
Microsoft Research
Bengaluru, Karnataka, India
gauravsinha@microsoft.com


Abstract

Generative Retrieval introduces a new approach to Information Retrieval by reframing it as a constrained generation task, leveraging recent advancements in Autoregressive (AR) language models. However, AR-based Generative Retrieval methods suffer from high inference latency and cost compared to traditional dense retrieval techniques, limiting their practical applicability. This paper investigates fully Non-autoregressive (NAR) language models as a more efficient alternative for generative retrieval. While standard NAR models alleviate latency and cost concerns, they exhibit a significant drop in retrieval performance (compared to AR models) due to their inability to capture dependencies between target tokens. To address this, we question the conventional choice of limiting the target token space to solely words or sub-words. We propose PIXNAR, a novel approach that expands the target vocabulary of NAR models to include multi-word entities and common phrases (up to 5 million tokens), thereby reducing token dependencies. PIXNAR employs inference optimization strategies to maintain low inference latency despite the significantly larger vocabulary. Our results demonstrate that PIXNAR achieves a relative improvement of 31.0% in MRR@10 on MS MARCO and 23.2% in Hits@5 on Natural Questions compared to standard NAR models with similar latency and cost. Furthermore, online A/B experiments on a large commercial search engine show significant increase in clicks and revenue.¹

*Corresponding Author

[†]Work conducted during tenure as Research Fellow at Microsoft Research.

¹Code and other artifacts available at <https://github.com/Ravi-VK/PIXNAR>.

 This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike International 4.0 License.

KDD '25, August 3–7, 2025, Toronto, ON, Canada
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1245-6/25/08
<https://doi.org/10.1145/3690624.3709330>

CCS Concepts

• Information systems → Retrieval models and ranking; • Computing methodologies → Natural language generation.

Keywords

information retrieval, generative retrieval, non-autoregressive generation, natural language processing, fast inference

ACM Reference Format:

Ravisri Valluri, Akash Kumar Mohankumar, Kushal Dave, Amit Singh, Jian Jiao, Manik Varma, and Gaurav Sinha. 2025. Scaling the Vocabulary of Non-autoregressive Models for Fast Generative Retrieval. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1 (KDD '25)*, August 3–7, 2025, Toronto, ON, Canada. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3690624.3709330>

1 Introduction

Generative Retrieval (GR) has emerged as a promising approach within Information Retrieval, particularly for text retrieval tasks [3, 23, 40, 41]. This approach involves creating a set of document identifiers that represent documents from the original corpus. A generative model is then trained to generate document identifiers for an input query. The generated identifiers are subsequently mapped back to the corresponding documents in the corpus. GR methods typically utilize an autoregressive (AR) language model to generate the document identifier as a sequence of words or sub-words tokens from a predefined target vocabulary. By leveraging high-quality document identifiers and capturing complex dependencies between tokens through the autoregressive generation process, GR has achieved substantial improvements in retrieval performance in recent years, closing the gap with dense retrieval [3, 22, 23]. Additionally, GR can be complementary to dense retrieval, and has recently been used in conjunction with typical dense retrieval techniques for improved diversity [30, 31].

Despite these advancements, deploying GR models in low-latency applications, such as sponsored search, remains a significant challenge due to the high inference complexity of AR models [21, 30].

This stems from their sequential token-by-token generation mechanism [15]. To address this challenge, our paper explores the use of non-autoregressive (NAR) language models for GR. These models significantly reduce inference costs by generating all tokens of the document identifier simultaneously. However, this parallel generation limits the model’s ability to capture dependencies among tokens (words, sub-words) in the output identifier, leading to inferior retrieval performance compared to AR-based GR models. To enable NAR-based GR to leverage word and sub-word interactions during generation, we propose expanding the model’s target vocabulary by incorporating multi-word entities and common phrases as tokens. Intuitively, predicting phrases directly at each position in the output sequence allows the NAR model to better understand the intricate relationships between words and sub-words within each phrase, potentially enhancing retrieval performance. This forms the basis of our first research question:

(RQ1)- How does the retrieval accuracy of a NAR-based GR model (with a target vocabulary containing word/sub-word level tokens) change when the target vocabulary is expanded to include phrases from document identifiers as additional tokens?

While a positive answer to the above question will provide an approach to get high quality retrieval from NAR-based GR, it also comes at the cost of increased inference latency. While generating phrases at output instead of solely words or sub-words leads to shorter output sequences, predicting the most likely tokens at each of these output positions becomes computationally demanding leading to much higher overall latency. Consequently, to make NAR-based GR truly viable for latency-sensitive applications such as sponsored search, we need to develop efficient inference methods that can select the top tokens from the enlarged vocabulary more efficiently. This leads us to our second research question:

(RQ2)- How can we reduce the inference latency of a NAR-based GR model with a target vocabulary comprising of millions of tokens without compromising its retrieval accuracy?

In this work, we make progress on both these questions. Our key contributions are outlined below.

1.1 Our Contributions

- (1) We present PIXNAR (Phrase-Indexed eXtreme vocabulary for non-Autoregressive Retrieval), a novel approach to NAR-based GR. By leveraging a vast target vocabulary encompassing phrases within document identifiers, PIXNAR achieves superior retrieval quality compared to conventional NAR-based GR models. Notably, PIXNAR uses a vocabulary size of 5 million tokens, the largest among models seen in literature. Through novel training and inference optimizations, PIXNAR effectively mitigates the computational burden associated with its large vocabulary. This allows for efficient retrieval of relevant documents during the inference process. The architecture of PIXNAR is presented in Figure 1. A comprehensive explanation of each component can be found in Section 4.
- (2) We conducted comprehensive experiments on two widely-used text retrieval benchmarks, MS MARCO [2] and Natural

Questions (NQ) [19]. Our results demonstrate PIXNAR’s significant performance gains: a relative improvement of 24.0% in MRR@10 on MSMARCO and a 23.2% increase in Hits@5 on NQ, compared to standard NAR-based retrieval models while maintaining similar inference latency. These findings underscore PIXNAR’s effectiveness in enhancing retrieval quality for various text retrieval tasks.

- (3) Further, online A/B testing on a large commercial search engine show that PIXNAR improves overall revenue by 1.00% and 1.29% for English and non-english queries, respectively. These findings validate PIXNAR’s practical value in improving user engagement and driving business outcomes.

2 Related Work

Generative retrieval: GR is an emerging paradigm in information retrieval that formulates retrieval as a generation task. A key distinction among different GR methods lies in their approach to represent documents. Some methods directly generate the full text of the document, particularly for short documents like keywords [24, 29, 34]. Others opt for more concise representations, such as numeric IDs [27, 35, 40–42, 44], document titles [5, 6], sub-strings [3], pseudo queries [39], or a combination of these descriptors [22, 23]. GR models generally use either numeric IDs or word-based document identifiers; when they use word-based identifiers, they can leverage pretrained language models. Despite showcasing promising results, existing GR approaches have high inference latency and computational cost due their reliance on AR language models, presenting a significant challenge for their real-world adoption.

Non-autoregressive Models: Recent works have explored NAR models for various generation tasks, such as machine translation [15], text summarization [33], and specific retrieval applications like sponsored search [30, 31]. NAR models aim to accelerate inference by predicting word or sub-word tokens independently and in parallel with a single forward pass. However, NAR models struggle to capture the inherent multimodality in target sequences, where multiple valid outputs exist for a single input, due to their lack of target dependency modeling [15]. This often leads to predictions that mix tokens from multiple valid outputs, resulting in significant performance degradation. To mitigate this, existing approaches focus on accurately predicting a single mode rather than modeling all modes. For instance, some methods use knowledge distillation to simplify the training data [15, 43], while a few others relax the loss function [10, 12, 26, 36]. While these approaches are effective for tasks requiring a *single* correct output, GR necessitates retrieving *all* relevant document identifiers for accurate retrieval and ranking. In this work, we propose an orthogonal approach to improve NAR models for retrieval by directly predicting phrases instead of sub-words. This reduces the number of independent predictions required in NARs, leading to improved retrieval performance.

Efficient Softmax: The softmax operation, crucial for generating probability distributions over target vocabularies in language models, presents a significant computational bottleneck, particularly for large vocabularies. Existing approaches address this through techniques such as low-rank approximation of classifier weights [7, 37], clustering of classifier weights or hidden states to pre-select

target tokens [8, 14]. However, these methods remain computationally expensive for NAR models which perform multiple softmax operations within a single forward pass. In contrast, we introduce a novel method that utilizes a dedicated shortlist embedding to efficiently narrow down target tokens for the entire query, thereby significantly reducing latency and maintaining strong retrieval performance.

Large Vocabulary: Recent work has highlighted the benefits of large sub-word vocabularies for encoder models, particularly in multilingual settings [25]. Non-parametric language models, which predict outputs from an open vocabulary of n-grams and phrases using their dense embeddings, have also gained traction for tasks like question answering and text continuation [4, 20, 28]. While our work shares the goal of expanding vocabulary size with non-parametric models, we directly learn classifier weights for an extended target vocabulary within a non-autoregressive framework.

3 Preliminaries

Notation: We let Q to be a set of queries and X to be a finite set of text documents (called the document corpus). Following prior work [3, 22, 23], we use document identifiers (docids), $d \in \mathcal{D}$, to represent documents. Specifically, we leverage pseudo-queries generated by pre-trained language models as docids. We denote the set $\{m, \dots, n\}$ by $[m, n]$, for non-negative integers $m < n$. We use \mathbb{P} (with or without subscripts) to denote probability distributions and the exact distribution is made clear at the time of use. We define $\arg \text{top } k_{x \in X} f(x)$ to be a set of k elements in set X such that no other element in X has a functional value greater than that of any element in this set.

AR models generate a docid d token by token, with each token depending on the previously generated ones. Let s be the length of the output sequence to be generated, AR models select top tokens at each position $t \in [s]$ based on the conditional distribution $\mathbb{P}(\cdot | d^{<t}, q, \theta)$. Here $d^{<t}$ is the sequence of tokens generated so far (i.e. till the $(t-1)^{\text{th}}$ position), q is the input query and θ represents the model parameters. This sequential generation process introduces significant latency, especially when retrieving hundreds of documents requiring large beam sizes or long sequence lengths.

NAR Models generate all tokens of the docid in parallel and therefore lead to faster retrieval than AR models. These models assume conditional independence among target tokens, i.e., $\mathbb{P}(d | q, \theta) = \prod_{t=1}^n \mathbb{P}_t(d^t | q, \theta)$ and so for each position $t \in [s]$, they select the top tokens based on the conditional probability distribution $\mathbb{P}(\cdot | q, \theta)$. This simplification enables efficient inference but comes at a cost. Previous studies in various applications, including machine translation [15, 16], have demonstrated that the assumption of conditional independence rarely holds for real-world data. Consequently, NAR models often struggle to capture crucial dependencies between target tokens, leading to a substantial performance degradation compared to their autoregressive counterparts. In our proposed work described in Section 4, we develop a technique that can overcome this quality degradation by adding phrase level tokens (within docids) and designing novel training/inference mechanisms that can still benefit from the parallel generation mode of NAR.

4 Proposed Work: PIXNAR

The core idea behind PIXNAR is to scale up the target vocabulary of NAR models by including phrases from docids. We explain the methodology for constructing this expanded vocabulary in Section 4.1. To enable efficient inference with a larger vocabulary, PIXNAR constructs a small number of token subsets from the target vocabulary during training. At inference time, PIXNAR selects and combines relevant subsets to create a concise *shortlist* of candidate tokens. For each output position, PIXNAR only re-ranks tokens among this shortlisted subset to predict the top tokens. Finally, these top tokens at different positions are combined using trie constrained beam search to generate the docids. Sections 4.3 and 4.4 provide the complete details of the PIXNAR pipeline, including the novel training and inference mechanisms. Figure 1 illustrates the different components of PIXNAR through a concrete example.

4.1 Vocabulary

Our objective is to develop a target tokenizer and vocabulary with the following key features: (i) Efficient Encoding: The vocabulary should minimize the number of bits required to encode docids, resulting in shorter target sequences, (ii) Token Frequency: Ensure that each token appears with sufficient frequency in the docid set to support effective training of the language model weights, and (iii) Linguistic Structure: Incorporate common phrases while maintaining word boundaries.

While Byte-Pair Encoding (BPE) is a widely-used method for creating vocabularies, it often involves pre-tokenization that splits sentences by spaces and punctuation marks. This approach, used in models like Llama and GPT, typically results in vocabularies limited to words and sub-words, which, as demonstrated in Section 5.4, perform significantly worse than phrase-based vocabularies. Modifying the pre-tokenization heuristic to avoid splitting on spaces can lead to tokens that mix characters from different words. Similarly, unigram tokenization faces issues when spaces are not used as delimiters, producing tokens that span parts of multiple words.

To address these challenges, we employ a tokenization algorithm [11] that considers word boundaries and other crucial heuristics to construct a phrase-based vocabulary, effectively overcoming the limitations of traditional methods. Please refer to Appendix C for additional details. A crucial feature of our vocabulary is its scale. In this study, we construct a vocabulary comprising 5 million tokens, which, to our knowledge, is the largest among the models referenced in the literature. The integration of phrase-level tokens significantly increases the variety of token options, allowing for substantial expansion of the vocabulary.

4.2 Non-autoregressive Generative Retrieval

Overview: Reviewing the entire retrieval process is helpful for understanding the challenges faced at inference and how the proposed solution addresses them.

- (1) **Obtain hidden states.** Pass the query through the transformer layers to obtain hidden states.
- (2) **Compute token probabilities.** Use the language modeling head to calculate token probabilities at each position.
- (3) **Identify top tokens.** Identify the top b tokens and their probabilities at each position.

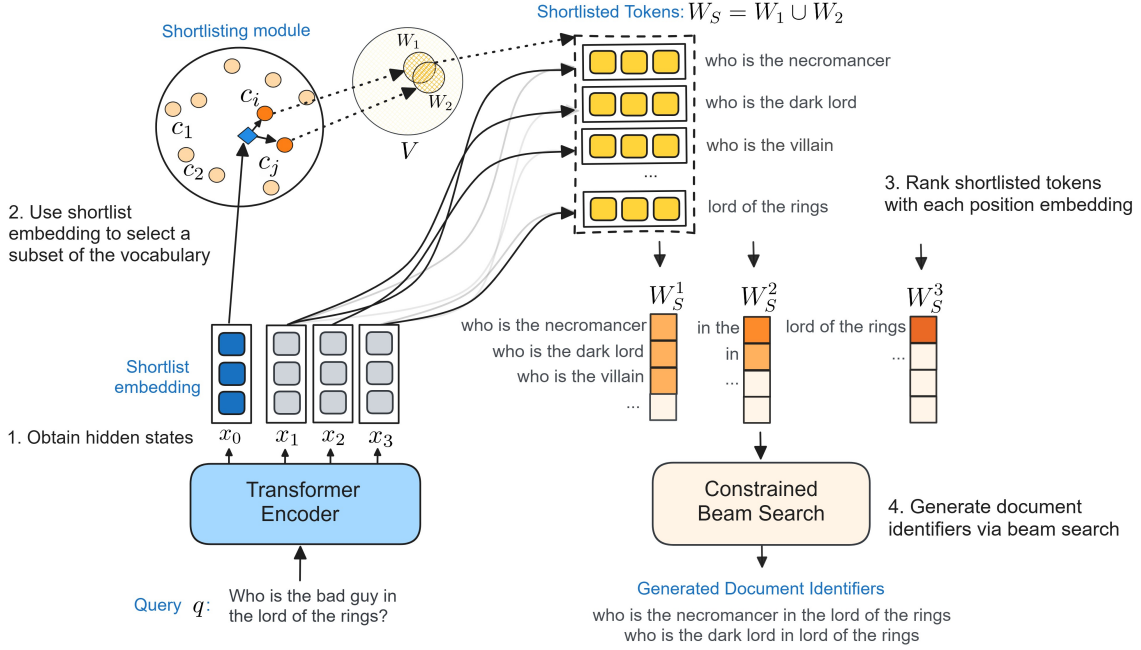


Figure 1: PIXNAR inference pipeline: The query is first encoded by a Transformer to produce shortlist embedding x_0 and token embeddings $\{x_1, \dots, x_s\}$. The shortlist embedding x_0 is used to identify k vocabulary clusters $\{c_i\}_{i=1}^k$. The union of these clusters, W_S , is then re-ranked at each position using the corresponding token embeddings, producing a set of ranked candidate tokens ($W_S^1 \dots W_S^s$) with their probability scores. The docids are predicted from these tokens via constrained beam search.

- Generate document identifiers.** Perform trie-constrained beam search to generate document identifiers.
- Rank the documents.** Rank the documents based on scores derived from the generated document identifiers.

After adding a special [CLS] token to the query q and passing it through the transformer layers, we obtain a sequence of embeddings known as hidden states: $x_0(q), x_1(q), \dots, x_s(q) \in \mathbb{R}^d$, where s is the sequence length and d is the hidden size. For simplicity, we omit q when only a single query is considered. For each token $u \in V$, the corresponding token vector in the language modeling head is $w_u \in \mathbb{R}^d$. At position $t \in [s]$, the probability of token v is:

$$\mathbb{P}_t(v | q) = \frac{\exp(x_t^\top w_v)}{\sum_{u \in V} \exp(x_t^\top w_u)} = \frac{\exp(x_t^\top w_v)}{Z_t},$$

where Z_t is the normalization factor. Once token probabilities are obtained at each position, the top b tokens are identified:

$$S_t = \arg \operatorname{top} b \mathbb{P}_t(u | q).$$

The top b tokens and their probabilities are then used by the decoding algorithm (refer to Appendix A) to generate document identifiers. The trie-constrained beam search produces a list of potential document identifiers. The score for each document identifier d is the length-normalized sum of the log probabilities of its tokens. The length normalization factor depends on the document identifier length l and hyperparameters A , a , and b . The score is calculated

as follows:

$$\operatorname{score}(d, q) = \left(\frac{a}{b+l}\right)^A \cdot \sum_{t=1}^l \log \mathbb{P}_t(d^t | q).$$

Let \mathcal{D}_p denote the set of generated document identifiers mapping to document p , and let \mathcal{X}_d denote the set of documents that identifier d maps to. The score for document p is then given by:

$$\operatorname{score}(p, q) = \sum_{d \in \mathcal{D}_p} \operatorname{score}(d, q) \cdot \frac{1}{|\mathcal{X}_d|}.$$

Challenges: Steps 2 and 3 are computationally expensive when dealing with large vocabularies. In Step 2, computing the normalization factor Z_t requires the inner product of the hidden state x_t with each token vector in the vocabulary. Step 3 involves searching for the most probable tokens within a vocabulary-sized space at each position. In PIXNAR, the vocabulary size reaches 5 million tokens, which is orders of magnitude larger than that of a typical language model. At this scale, even a single decoding step can be prohibitively slow and costly. To handle this increased vocabulary size, it is essential to significantly reduce inference costs.

4.3 PIXNAR Inference

PIXNAR attempts to speed up inference by making it faster to compute the token probabilities, and by narrowing the search space when identifying high probability tokens at each position. This is achieved by means of two key strategies set during training. (1)

Using self normalization [13] [9], we encourage the normalization factor Z_t to approach 1 for *all* hidden states and (2) using a specially designed *shortlist embedding*, we generate a small vocabulary subset W_S (orders of magnitude smaller than the full vocabulary) to identify the top b tokens at each position.

This leads to the following estimates for the probability distribution $\mathbb{P}_t(\cdot|q)$ and for the set of the top b tokens S_t :

$$\tilde{\mathbb{P}}_t(v|q) = \exp(x_t^T w_v), \quad (1)$$

$$\tilde{S}_t = \arg \operatorname{top} b \mathbb{P}_t(u|q). \quad (2)$$

Notice that we compute a single shortlist W_S for all positions. If we were to directly adopt a method to identify a subset of the vocabulary for computing Softmax [1][8] and the top b tokens at each position, it would be necessary to compute multiple shortlists - one for each position. This unified approach is particularly beneficial for NAR models, allowing for a single shortlist to be used across multiple positions. The shortlist is generated at the query level rather than individually at each token position.

The shortlist W_S is derived from the shortlist embedding x_0 , which, due to the training process, has a higher affinity with tokens relevant to the query. Rather than searching the entire vocabulary for the top b tokens at each position, we can use this shortlist:

$$\arg \operatorname{top} r \langle x_0, w_u \rangle, \quad u \in V$$

for an appropriate value of hyperparameter r . However, this would still have a linear dependence on vocabulary size.

To tackle this issue, we draw inspiration from prior research that leverages query-side similarities [1, 8]. If a shortlist embedding closely resembles a vector c , it's likely that the relevant tokens will also be similar. In the shortlist embedding space, we maintain a set of m learned vectors, denoted as $c_1, c_2, \dots, c_m \in \mathbb{R}^d$. For each vector, the r most relevant tokens are precomputed:

$$W_i = \arg \operatorname{top} r \langle c_i, w_u \rangle, \quad u \in V$$

The top k vectors most similar to x_0 are selected:

$$\text{Top-}k \text{ vectors} := \arg \operatorname{top} k \langle c_i, x_0 \rangle, \quad i \in [m]$$

Assuming the selected vectors are c_1, c_2, \dots, c_k , the final shortlist is the union of their relevant token subsets:

$$W_S = W_1 \cup W_2 \cup \dots \cup W_k.$$

The maximum size of this union is rk , where r is the number of tokens per vector and k is the number of selected vectors. By carefully choosing the hyperparameters, we can ensure that $rk \ll |V|$.

At each position, the tokens in W_S are reranked using the efficient estimates of $\mathbb{P}_t(\cdot|q)$ (see Equation 1). This results in ordered sets W_S^t for each $t \in [s]$, and the top b tokens in W_S^t comprise \tilde{S}_t (see Equation 2). The final document identifiers are generated using permutation decoding, which employs constrained beam search on trie data structures representing document identifiers from \mathcal{D} as token sequences from the target vocabulary V .

4.4 PIXNAR Training

The training objectives of PIXNAR are designed to enable the non-autoregressive generation of document identifiers while minimizing estimation errors in both the probability distribution and the set of top tokens. Specifically, we aim to minimize the errors $|\mathbb{P}_t(v|q) - \tilde{\mathbb{P}}_t(v|q)|$ and $|S_t \setminus \tilde{S}_t|$. We train PIXNAR using a dataset of query-docid pairs $(q_1, d_1), \dots, (q_N, d_N)$. The training process is divided into two stages.

Stage 1: The first stage focuses on training the language model to generate entire sequences non-autoregressively and to produce a shortlist embedding. We minimize a loss function $\ell(\bar{\theta})$ to learn the vector $\bar{\theta}$, which consists of the hidden parameters from the transformer layers and the token parameter vectors w_u , where $u \in V$. Our loss $\ell(\bar{\theta})$ comprises three components:

1. Cross-Entropy Loss: The first term, $\ell_1(\bar{\theta})$, is the standard cross-entropy loss between the Softmax predictions at each $t \in [1]$ and the actual docid sequence in the training data:

$$\ell_1(\bar{\theta}) = - \sum_{i=1}^N \sum_{t=1}^s \log \left[\mathbb{P}_t(d_i^t | q_i) \right].$$

2. Self-Normalization Loss: The second term, $\ell_2(\bar{\theta})$, encourages the normalization factor Z_t to approach 1 for all hidden states. If this condition holds, $|\mathbb{P}_t(v|q) - \tilde{\mathbb{P}}_t(v|q)| = 0$. After minimizing this loss [9][13], we can use the probability estimates $\tilde{\mathbb{P}}_t(v|q) = \exp(x_t^T w_v)$ instead of $\mathbb{P}_t(v|q)$.

$$\ell_2(\bar{\theta}) = \sum_{i=1}^N \sum_{t=1}^s \log^2 \left[\sum_{v \in V} \exp(x_t(q_i)^T w_v) \right] = \sum_{i=1}^N \sum_{t=1}^s \log^2 Z_t(q_i).$$

3. Shortlist Embedding Loss: The third term, $\ell_3(\bar{\theta})$, evaluates how well $x_0(q)$ can predict the tokens in the output docid. This loss term helps ensure the shortlist embedding summarizes the entire query effectively:

$$\ell_3(\bar{\theta}) = - \sum_{i=1}^N \sum_{t=1}^s \log \left[\mathbb{P}_0(d_i^t | q_i) \right].$$

The overall loss function is then:

$$\ell(\bar{\theta}) = \ell_1(\bar{\theta}) + \lambda_2 \ell_2(\bar{\theta}) + \lambda_3 \ell_3(\bar{\theta}),$$

where λ_2 and λ_3 are hyperparameters to be tuned.

Stage 2: After minimizing $\ell(\bar{\theta})$, we proceed to learn the vectors c_1, \dots, c_m as described earlier. For each training pair (q_i, d_i) , let $e_i \in [m]$ be the index of the vector c_{e_i} that has the highest inner product with the shortlist embedding $x_0(q_i)$:

$$e_i = \arg \max_{j \in [m]} \langle x_0(q_i), c_j \rangle.$$

We then minimize the loss function $\ell'(c_1, \dots, c_m)$, which computes the cross-entropy loss between the Softmax distributions $\mathbb{P}_{c_{e_i}}$ and the docid sequence d_i :

$$\ell'(c_1, \dots, c_m) = - \sum_{i=1}^N \sum_{t=1}^s \log \left[\mathbb{P}_{c_{e_i}}(d_i^t) \right].$$

Intuitively, this objective aims to maximize the likelihood of the tokens present in the docid d_i for the vector c_{e_i} that best aligns with $x_0(q_i)$. This ensures that the set W_{e_i} (defined earlier in 4.3) has a high probability of containing the tokens in d_i .

In the PIXNAR pipeline, we identify the top k vectors that have the highest inner product with $x_0(q_i)$, rather than just the most aligned vector c_{e_i} . This improves the likelihood of the tokens in d_i being present in $W_0(q)$, as it is a union of the token sets corresponding to these k vectors.

To initialize the vectors c_1, \dots, c_m , we compute the shortlist embeddings for a large number of queries and apply spherical k -means clustering. The centroids of the resulting clusters are then used to initialize the vectors for subsequent training.

5 Experiments & Results

In this section, we evaluate our proposed PIXNAR method in three different experimental settings. First, we benchmark PIXNAR against leading GR approaches, including AR and NAR methods. Next, we perform a component-wise ablation study on PIXNAR to examine the impact of each component on retrieval performance and model latency. We also compare our novel inference pipeline (Section 4.3) with inference optimization methods from the literature. Finally, we assess the effectiveness of PIXNAR in a real-world application, focusing on sponsored search.

5.1 Experimental Setup

We evaluate PIXNAR on two types of datasets: (i) public datasets designed for passage retrieval tasks, and (ii) a proprietary dataset used for sponsored search applications. Below, we describe both:

Public Datasets: We use two prominent datasets to evaluate PIXNAR and other GR methods: MS MARCO [2] and Natural Questions (NQ) [19]. The MS MARCO dataset, derived from Bing search queries, provides a large collection of real-world queries and their corresponding passages from relevant web documents. NQ contains real user queries from Google Search that are linked to relevant Wikipedia articles, emphasizing text retrieval for answering intricate information needs. For both these datasets, we follow the preprocessing approach of [23] and utilize pseudo queries generated from passages as docids for PIXNAR.

Proprietary Dataset: We further evaluate PIXNAR in the context of sponsored search, where the objective is to retrieve relevant ads for user queries. We utilize advertiser bid keywords as docids for ads. We perform offline evaluations on SponsoredSearch-1B, a large-scale dataset of query-keyword pairs mined from the logs of a large commercial search engine. This dataset includes approximately 1.7 billion query-keyword pairs, with 70 million unique queries and 56 million unique keywords. The test set consists of 1 million queries, with a retrieval set of 1 billion keywords.

Metrics & Baselines: Following prior work [22, 23], we evaluate all models using MRR@ k and Recall@ k for the MS MARCO dataset, and Hits@ k for NQ. For the SponsoredSearch-1B dataset, we use Precision@ K as the evaluation metric. Additionally, we measure inference latency with a batch size of 1 on a Nvidia T4 GPU. We compare PIXNAR with several AR baselines, including DSI [40], NCI [41], SEAL [3], MINDER [23], and LTRGR [22]. We report retrieval results from the respective papers and obtain inference latency by running the official code. For NAR baselines, we include CLOVERv2 [30] and replicate their method on our datasets due to the absence of reported numbers and official code for these datasets. Complete implementation details are provided in Appendix 7.

5.2 Results

We present the results of PIXNAR and various GR baselines on the MS MARCO dataset in columns 4-7 of Table 1. We observe several key findings from this comparison. First, CLOVERv2 significantly outperforms AR baselines like SEAL, NCI, and DSI, while also offering substantial improvements in inference latency. This highlights CLOVERv2 as a strong NAR baseline. However, CLOVERv2 falls short when compared to more recent AR models, particularly MINDER and LTRGR. For instance, CLOVERv2's recall at 100 is lower than that of MINDER by 11.8 absolute points. Next, our proposed PIXNAR model with a 5M target vocabulary outperforms the strong CLOVERv2 baseline across all metrics, showing approximately 20-30% relative improvements. This strongly supports our hypothesis that increasing the target vocabulary of NAR models can significantly improve retrieval performance. Moreover, PIXNAR exceeds the performance of MINDER in every metric, achieving a 22.5% improvement in MRR at 10, while also achieving substantial speedups in inference latency. Notably, PIXNAR achieves this performance without utilizing multiple types of docids like MINDER (titles, n-grams, pseudo queries) and relies solely on pseudo queries. Additionally, PIXNAR closely rivals LTRGR, lagging by only 1.5 absolute points in MRR@10 (a 5.8% relative difference), despite not using a complex two-stage training with a passage-level loss.

The results on the NQ dataset are presented in the last three columns of Table 1. Here, the baseline CLOVERv2 NAR model significantly trails behind AR models like SEAL, MINDER, and LTRGR. For example, CLOVERv2 exhibits a relative gap of 16.3% with respect to LTRGR on recall at 100. Similar to MS MARCO, PIXNAR substantially outperforms CLOVERv2 on all metrics, yielding around 13-23% gains while maintaining significant latency speedups over AR models. Importantly, PIXNAR reduces the relative gap with LTRGR from 16.3% to 5.1%. These results demonstrate the effectiveness of PIXNAR in leveraging large vocabularies in NAR models to achieve substantially better retrieval performance than standard NAR models while retaining their latency benefits.

5.3 Qualitative Analysis

To gain deeper insights into PIXNAR's superior performance compared to smaller-vocabulary NAR models like CLOVERv2, we present qualitative examples in Table 2. PIXNAR's tokenizer effectively captures multi-word entities like locations (e.g., "des moines iowa") and common phrases (e.g., "average temp", "what's the weather like in") as single tokens. Consequently, the weights in the language modelling head of PIXNAR can learn representations for these multi-word entities and phrases from training data, capturing their semantic meaning. In contrast, standard NAR models like CLOVERv2 tend to break down words representing single concepts into multiple tokens (e.g., "des moines iowa" is fragmented into four tokens: "des", "mo", "ines", "iowa"). This hinders the language modeling head from learning meaningful representations for these concepts. Moreover, representing common phrases like "what's the weather like in" allows PIXNAR to make fewer independent predictions in parallel, reducing the target output sequence length. Specifically, the mean and 99th percentile target sequence length decreases from 10.98 to 4.05 and from 18 to 9 in PIXNAR compared to CLOVERv2. This reduction in target tokens simplifies the model's

Models	GPU Latency	MS MARCO				Natural Questions		
		@5	@20	@100	M@10	@5	@20	@100
DSI [32, 40]	-	-	-	-	17.3	28.3	47.3	65.5
NCI [41]	-	-	-	-	9.1	-	-	-
SEAL-LM [3]	84.3x	-	-	-	-	40.5	60.2	73.1
AR SEAL-LM+FM [3]	84.3x	-	-	-	-	43.9	65.8	81.1
SEAL [3]	84.3x	19.8	35.3	57.2	12.7	61.3	76.2	86.3
MINDER [23]	94.1x	29.5	53.5	78.7	18.6	65.8	78.3	86.7
LTRGR [22]	94.1x	40.2	64.5	85.2	25.5	68.8	80.3	87.1
CLOVERv2 [30]	1.0x	29.2	47.7	66.9	18.3	49.6	63.4	72.9
NAR PIXNAR (Ours)	1.2x	38.7	61.0	80.9	24.0	61.1	74.1	82.7
% improvement	-	32.7%	27.9%	20.9%	31.0%	23.2%	16.9%	13.4%

Table 1: Performance and inference latency on MS MARCO and NQ. We report Recall@5, 20, 100, MRR@10 (MS MARCO) and Hits@5, 20, 100 (NQ), with inference latency relative to CLOVERv2. "-" denotes unreported results.

Query	PIXNAR	CLOVERv2
average temperatures des moines iowa	<ol style="list-style-type: none"> 1. <u>average temp des moines iowa</u> 2. <u>what's the average temperature in des moines iowa</u> 3. <u>weather in des moines iowa fahrenheit</u> 4. <u>what's the weather like in des moines</u> 	<ol style="list-style-type: none"> 1. <u>average temperature</u> 2. <u>what temperature</u> 3. <u>what is des mo-ines</u> 4. <u>what is des</u>
supernova boost shoes	<ol style="list-style-type: none"> 1. <u>adidas supernova boost running shoes</u> 2. <u>adidas supernova running shoes</u> 3. <u>adidas boost running shoes</u> 4. <u>supernova boost adidas running shoes</u> 	<ol style="list-style-type: none"> 1. <u>adidas ultraboost</u> 2. <u>adidas boost</u> 3. <u>adidas sneakers</u> 4. <u>adidas ultra boost</u>
premier league tickets arsenal	<ol style="list-style-type: none"> 1. <u>arsenal tickets premier league</u> 2. <u>arsenal fc premier league</u> 3. <u>liverpool arsenal premier league</u> 4. <u>adidas ultra boost</u> 	<ol style="list-style-type: none"> 1. <u>arsenal tickets</u> 2. <u>arsenal fc tickets</u> 3. <u>arsenal tickets match</u> 4. <u>premier league football</u>

Table 2: Examples from PIXNAR (5M vocab) and CLOVERv2 (128K vocab) on an MS MARCO dev query and two sponsored search queries. Underlined spans indicate target tokenizer tokens.

prediction task, leading to improved retrieval performance. Interestingly, despite shorter target sequence lengths, PIXNAR tends to predict longer outputs with more words, as each token represents multiple words. This addresses a common issue with NAR models, namely their tendency to generate short outputs [17, 30]. We provide further analysis in Appendix 7.

5.4 Ablations

Our PIXNAR model integrates three primary components: (i) a vocabulary and tokenizer that incorporate phrases in addition to words, (ii) an expanded vocabulary size of 5M tokens, and (iii) an efficient inference pipeline to accelerate NAR inference. To analyze each component's impact, we conducted detailed ablation studies:

Phrase-enhanced Vocabulary: We first investigated the effectiveness of PIXNAR's vocabulary construction strategy (detailed in Section 4.1), focusing on the inclusion of phrases. To isolate this effect, we fixed the vocabulary size to 128K, equivalent to that of

Tokenizer	M@10	R@5	R@20	R@100
DeBERTa	18.3	29.2	47.7	66.9
BPE	18.7	29.8	48.5	67.4
Unigram	19.0	30.5	49.7	68.7
Phrase-based	21.6	34.7	56.0	77.5

Table 3: Retrieval performance of different tokenizers on MS MARCO (vocabulary size of 128K)

DeBERTa-v3, which was used to initialize the encoder. We compared the retrieval performance on the MS MARCO dataset using the original DeBERTa BPE tokenizer, a custom sub-word BPE, a sub-word Unigram, and our phrase-based tokenizer, all trained on the MS MARCO docid set. Table 3 presents the retrieval performance for the different tokenizers. We observed that a custom-tailored BPE tokenizer performs marginally better than the original DeBERTa tokenizer. Further, the Unigram tokenizer outperforms the BPE by approximately 1.9% in MRR@10 and Recall@100, in relative terms.

Vocab Size	MS MARCO				Natural Questions		
	@5	@20	@100	M@10	@5	@20	@100
128K	34.7	56.0	77.5	21.6	56.7	71.6	80.7
500K	34.9	56.9	78.6	21.7	57.8	72.7	81.4
800K	35.2	57.5	79.2	21.9	58.2	73.0	81.2
1M	35.7	58.4	79.6	22.5	58.5	73.0	82.0
5M	38.5	61.0	81.6	24.2	61.2	74.8	83.5

Table 4: Scaling vocabulary improves NAR retrieval: We report the Recall@k and Hits@k for MSMARCO and NQ

Method	MSMARCO		Latency (ms)	
	MRR@10	R@100	Mean	99
Full Softmax	24.2	81.6	47.9	48.3
SVD-Softmax [38]	22.8	78.6	13.7	14.3
HiRE-Softmax [37]	24.0	81.3	12.7	13.2
Centroid Clustering [1]	21.7	78.2	14.2	17.4
Fast Vocab [1]	22.6	79.6	9.5	16.7
PIXNAR (Ours)	24.0	80.9	4.5	5.0

Table 5: Retrieval performance and inference latency (in ms) for PIXNAR and other softmax optimization methods

Most notably, our phrase-based tokenizer substantially outperforms the best baseline (Unigram tokenizer), with a relative improvement of 13.7% in MRR@10 (19.0% to 21.6%) and 12.6% in Recall@100 (68.7% to 77.5%). These results clearly demonstrate the benefits of extending beyond words to include phrases in the target vocabulary.

Vocabulary Scaling: Next, we analyze the impact of increasing the target vocabulary size in NAR models, addressing **RQ1** posed in Section 1. For this study, we utilized the phrase-based tokenizer and varied the vocabulary size from 128K to 5 million tokens. We used the full softmax operation without any approximation to observe the raw effect of scaling. As shown in Table 4, there is a consistent increase in retrieval performance as the vocabulary size increases across both MS MARCO and NQ datasets. Notably, the improvement persists even when the vocabulary size exceeds 1 million tokens. For instance, when increasing the vocabulary size from 1 million to 5 million tokens, Recall@5 on the MS MARCO dataset improves by 7.7% (from 35.7 to 38.5). These findings highlight the clear advantages of scaling up the vocabulary size in NAR models in terms of retrieval performance.

Efficient PIXNAR Inference: Scaling vocabulary size introduces computational challenges due to the expensive softmax operation. Table 5 compares PIXNAR’s inference pipeline (Section 4.3) against established techniques: (i) low-rank approximation methods: SVD-Softmax [38], HiRE-Softmax [37]) and (ii) clustering-based methods: Fast Vocabulary Projection [7] and its variant Centroid Projection. While offering modest speedups, low-rank approximations like HiRE-softmax still result in significantly higher inference latency (3.4x slower than the 128k vocabulary CLOVERv2 baseline) due to their linear complexity with vocabulary size. Clustering-based

Model	Vocabulary	M@10	R@5	R@20	R@100
NQ-MS-FT	NQ	28.2%	29.5%	49.9%	71.3%
MS-T	MSMARCO	24.2%	38.5%	61.0%	81.6%

Table 6: Retrieval performance on MSMARCO with different training and vocabulary configurations. NQ-MS-FT: NQ trained, MSMARCO finetuned; MS-T: MSMARCO trained.

Model	Vocabulary	Acc@5	Acc@20	Acc@100
MS-NQ-FT	MSMARCO	57.4%	71.9%	81.3%
NQ-T	NQ	61.2%	74.8%	83.5%

Table 7: Retrieval performance on NQ with different training and vocabulary configurations. MS-NQ-FT: MSMARCO trained, NQ finetuned; NQ-T: NQ trained.

Component	# Parameters
Embedding Matrix	98M
Transformer Layers	85M
Language Modeling Head	3.84B

Table 8: Model Component Parameters

methods like Fast Vocabulary Projection offer further speedups in mean latency but remain 2.5x slower than CLOVERv2. In contrast, PIXNAR achieves superior performance, delivering a 10.6x speedup over full softmax and a 2.1x speedup over Fast Vocabulary Projection while maintaining comparable retrieval performance to full softmax (within 0.82% in MRR@10 and 0.85% in Recall@100). This translates to a latency only 21% higher than the CLOVERv2 model which has a 39x smaller vocabulary. These results highlight the effectiveness of our tailored softmax approximation, which efficiently predicts shortlist tokens in NAR models.

5.5 Scalability

Increasing vocabulary size poses challenges for both memory and computational efficiency during training and inference. This section outlines how our approach addresses these challenges.

Memory Costs: Larger vocabularies increase memory demands. In our 5M-token model with a hidden size of 768, total parameters reach 3.84B, exceeding those of the transformer layers and embedding matrix. With bf16 quantization, the model fits within 8GB of memory, well within the capacity of 16GB GPUs like T4s and V100s. For larger vocabularies, techniques such as low-rank or sparse matrices may be necessary. The breakdown of parameters by component is given in Table 8.

Computational Costs: Different components in the model have different computational footprints. The main inference bottleneck occurs at the language modeling head, where projecting hidden states onto large vocabularies can drastically slow down retrieval. Our inference algorithm uses a shortlist of up to 100,000 token vectors (only 2% of the vocabulary), reducing computational load and scaling to very large vocabularies.

Training costs scale with vocabulary size, particularly when computing the normalization factor for SoftMax Cross-Entropy.

Language	Δ Revenue	Δ Clicks	Δ QBR	Δ Defect
English	1.00%	0.27%	0.01%	-0.02%
Non-english	1.29%	0.43%	-0.26%	0.01%

Table 9: Results of online A/B tests in sponsored search. Gray color indicates non-significant delta (p-value > 0.01)

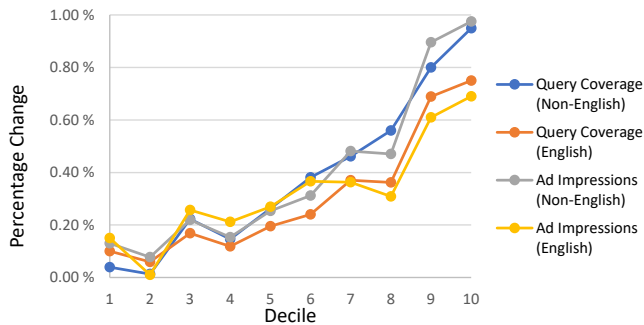


Figure 2: Change in query coverage and Ad impression for different deciles in online A/B tests on sponsored search

Although training with vocabularies up to 5M tokens is feasible, larger vocabularies will demand more efficient techniques such as negative sampling or hardware optimizations.

5.6 Generalization

While constructing a task-specific vocabulary optimized for each dataset can lead to improved performance, building a generalizable vocabulary that works well across multiple datasets is valuable for many practical applications. To assess generalization, we conducted cross-dataset experiments where PIXNAR was first trained on one dataset and then finetuned on the other. Specifically, we trained the model on MS MARCO and finetuned it on NQ, and vice versa. The results of these experiments are presented in Tables 6 and 7.

The MSMARCO-trained model finetuned on NQ achieved nearly the same performance as the NQ-specific model, with only a 2% gap in Accuracy@100. Similarly, the NQ-trained model finetuned on MS MARCO demonstrated strong performance, with a 10% gap compared to the model trained specifically on MS MARCO. These results indicate that the PIXNAR vocabulary exhibits strong generalizability across different datasets, providing robust retrieval performance without the need for dataset-specific vocabularies.

5.7 Application to Sponsored Search

To demonstrate the effectiveness of PIXNAR in real-world scenarios, we conducted extensive experiments in sponsored search, where the task is to retrieve the most relevant advertisements for user queries. In this application, ads are treated as documents, and the keywords bid by advertisers serve as the docids. We first evaluated PIXNAR on the SponsoredSearch-1B dataset, where it significantly outperformed CLOVERv2, increasing P@100 from 23.5% to 29.1% (relative improvement of 23.7%). Further, we deployed PIXNAR on a large-scale commercial search engine and conducted A/B testing against

an ensemble of leading proprietary dense retrieval and generative retrieval algorithms. As shown in Table 9, PIXNAR improved overall revenue by 1.00% for English queries and 1.29% for non-English queries, accompanied by a statistically significant increase in clicks. Additionally, we observed no statistically significant change in the Quick Back Rate (QBR), which denotes the percentage of ad clicks where users quickly returned to the search page. This indicates that the revenue increase resulted from an increase in high-quality clicks on ads that users found relevant. Moreover, we did not observe any statistically significant change in the Ad Defect Rate, as measured by offline relevance models, indicating that the ads displayed were indeed relevant. To further analyze the gains, we grouped queries into frequency-based buckets, referred to as deciles in sponsored search. Decile 1 contains highly frequent queries, while decile 10 consists of a large number of rare queries. For each decile, we measured two metrics: (i) query coverage, the fraction of queries for which any sponsored content was shown, and (ii) ad impressions, the total number of ads displayed for queries. As shown in Figure 2, PIXNAR increased both query coverage and ad impressions across all deciles, with particularly strong gains on tail queries, which are often longer and more ambiguous. These results indicate that PIXNAR was able to serve more relevant sponsored content to users, especially for tail queries that are typically harder to cater.

6 Conclusion & Future Work

In this work, we introduced PIXNAR, a novel NAR-based retrieval approach that incorporates phrase-level tokens within an expanded target vocabulary. Our experiments showed that PIXNAR narrows the performance gap with state-of-the-art AR methods while maintaining the efficiency of NAR models. This speed advantage makes PIXNAR a strong candidate for latency-sensitive applications such as real-time search and recommendation systems. The promising results open exciting avenues for further research. While a 5M vocabulary size is already significant, further expansion could yield even more substantial performance improvements. This will require not only efficient inference optimizations methods but also necessitates efficient training approximations along with data augmentation strategies. Additionally, exploring better techniques for vocabulary construction to further improve PIXNAR’s effectiveness could be another direction for future work.

7 Acknowledgement

Ravisri Valluri would like to thank Siddarth Asokan for feedback and guidance rendered over the course of writing this paper.

References

- [1] Hossam Amer, Young Jin Kim, Mohamed Afify, Hitokazu Matsushita, and Hany Hassan Awadalla. 2022. Fast Vocabulary Projection Method via Clustering for Multilingual Machine Translation on GPU. *ArXiv abs/2208.06874* (2022). <https://api.semanticscholar.org/CorpusID:251564288>
- [2] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2018. MS MARCO: A Human Generated MACHine Reading COMprehension Dataset. *arXiv:1611.09268* [cs.CL]
- [3] Michele Bevilacqua, Giuseppe Ottaviano, Patrick Lewis, Wen tau Yih, Sebastian Riedel, and Fabio Petroni. 2022. Autoregressive Search Engines: Generating Substrings as Document Identifiers. *ArXiv abs/2204.10628* (2022). <https://api.semanticscholar.org/CorpusID:248366293>

- [4] Bowen Cao, Deng Cai, Leyang Cui, Xuxin Cheng, Wei Bi, Yuexian Zou, and Shuming Shi. 2024. Retrieval is Accurate Generation. *ArXiv abs/2402.17532* (2024). <https://api.semanticscholar.org/CorpusID:268031947>
- [5] Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. 2020. Autoregressive Entity Retrieval. *ArXiv abs/2010.00904* (2020). <https://api.semanticscholar.org/CorpusID:222125277>
- [6] Jianguo Chen, Ruqing Zhang, J. Guo, Y. Liu, Yixing Fan, and Xueqi Cheng. 2022. CorpusBrain: Pre-train a Generative Retrieval Model for Knowledge-Intensive Language Tasks. *Proceedings of the 31st ACM International Conference on Information & Knowledge Management* (2022). <https://api.semanticscholar.org/CorpusID:251594672>
- [7] Patrick H. Chen, Si Si, Sanjiv Kumar, Yang Li, and Cho-Jui Hsieh. 2018. Learning to Screen for Fast Softmax Inference on Large Vocabulary Neural Networks. *arXiv:1810.12406* [cs.LG]
- [8] Patrick H. Chen, Si Si, Sanjiv Kumar, Yang Li, and Cho-Jui Hsieh. 2018. Learning to Screen for Fast Softmax Inference on Large Vocabulary Neural Networks. *ArXiv abs/1810.12406* (2018). <https://api.semanticscholar.org/CorpusID:53113692>
- [9] Jacob Devlin, Rabi Zbib, Zhongqiang Huang, Thomas Lamar, Richard M. Schwartz, and John Makhoul. 2014. Fast and Robust Neural Network Joint Models for Statistical Machine Translation. In *Annual Meeting of the Association for Computational Linguistics*. <https://api.semanticscholar.org/CorpusID:7417943>
- [10] Cunxiao Du, Zhaopeng Tu, and Jing Jiang. 2021. Order-Agnostic Cross Entropy for Non-Autoregressive Machine Translation. In *International Conference on Machine Learning*. <https://api.semanticscholar.org/CorpusID:235377210>
- [11] A. Forsythe. 2023. Tokenmonster: Ungreedy subword tokenizer and vocabulary trainer for python, go and javascript. (2023). <https://github.com/aldasairforysht/tokenmonster>
- [12] Marjan Ghazvininejad, Vladimir Karpukhin, Luke Zettlemoyer, and Omer Levy. 2020. Aligned Cross Entropy for Non-Autoregressive Machine Translation. In *International Conference on Machine Learning*. <https://api.semanticscholar.org/CorpusID:214795061>
- [13] Jacob Goldberger and Oren Melamud. 2018. Self-Normalization Properties of Language Modeling. In *International Conference on Computational Linguistics*. <https://api.semanticscholar.org/CorpusID:46929672>
- [14] Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2017. Efficient softmax approximation for GPUs. *arXiv:1609.04309* [cs.CL]
- [15] Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2017. Non-Autoregressive Neural Machine Translation. *ArXiv abs/1711.02281* (2017). <https://api.semanticscholar.org/CorpusID:3480671>
- [16] Jiatao Gu and X. Kong. 2020. Fully Non-autoregressive Neural Machine Translation: Tricks of the Trade. In *Findings*. <https://api.semanticscholar.org/CorpusID:229923438>
- [17] Junliang Guo, Xu Tan, Di He, Tao Qin, Linli Xu, and Tie-Yan Liu. 2019. Non-autoregressive neural machine translation with enhanced decoder input (AAAI'19/IAAI'19/EAAI'19). *AAAI Press, Article 457*, 8 pages. <https://doi.org/10.1609/aaai.v33i01.33013723>
- [18] Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing. *ArXiv abs/2111.09543* (2021). <https://api.semanticscholar.org/CorpusID:244346093>
- [19] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics* 7 (2019), 452–466. https://doi.org/10.1162/tacl_a_00276
- [20] Tian Lan, Deng Cai, Yan Wang, Heyan Huang, and Xian-Ling Mao. 2023. Copy is All You Need. *ArXiv abs/2307.06962* (2023). <https://api.semanticscholar.org/CorpusID:259298789>
- [21] Xiaoxi Li, Jiajie Jin, Yujia Zhou, Yuyao Zhang, Peitian Zhang, Yutao Zhu, and Zhicheng Dou. 2024. From Matching to Generation: A Survey on Generative Information Retrieval. <https://api.semanticscholar.org/CorpusID:269303210>
- [22] Yongqing Li, Nan Yang, Liang Wang, Furu Wei, and Wenjie Li. 2023. Learning to Rank in Generative Retrieval. In *AAAI Conference on Artificial Intelligence*. <https://api.semanticscholar.org/CorpusID:259262395>
- [23] Yongqing Li, Nan Yang, Liang Wang, Furu Wei, and Wenjie Li. 2023. Multiview Identifiers Enhanced Generative Retrieval. *ArXiv abs/2305.16675* (2023). <https://api.semanticscholar.org/CorpusID:258947148>
- [24] Yijiang Lian, Zhijie Chen, Jinlong Hu, Kefeng Zhang, Chunwei Yan, Muchenxuan Tong, Wenying Han, Hanju Guan, Ying Li, Ying Cao, Yang Yu, Zhigang Li, Xiaochun Liu, and Yue Wang. 2019. An end-to-end Generative Retrieval Method for Sponsored Search Engine –Decoding Efficiently into a Closed Target Domain. *arXiv:1902.00592* [cs.IR]
- [25] Davis Liang, Hila Gonen, Yuning Mao, Rui Hou, Naman Goyal, Marjan Ghazvininejad, Luke Zettlemoyer, and Madian Khabsa. 2023. XLM-V: Overcoming the Vocabulary Bottleneck in Multilingual Masked Language Models. *ArXiv abs/2301.10472* (2023). <https://api.semanticscholar.org/CorpusID:256231072>
- [26] Jindřich Libovický and Jindřich Helcl. 2018. End-to-End Non-Autoregressive Neural Machine Translation with Connectionist Temporal Classification. In *Conference on Empirical Methods in Natural Language Processing*. <https://api.semanticscholar.org/CorpusID:53083422>
- [27] Sanket Vaibhav Mehta, Jai Gupta, Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Jinfeng Rao, Marc Najork, Emma Strubell, and Donald Metzler. 2022. DSI++: Updating Transformer Memory with New Documents. *ArXiv abs/2212.09744* (2022). <https://api.semanticscholar.org/CorpusID:254854290>
- [28] Sewon Min, Weijia Shi, Mike Lewis, Xilun Chen, Wen tau Yih, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Nonparametric Masked Language Modeling. In *Annual Meeting of the Association for Computational Linguistics*. <https://api.semanticscholar.org/CorpusID:254220735>
- [29] Akash Kumar Mohankumar, Nikit Begwani, and Amit Singh. 2021. Diversity driven Query Rewriting in Search Advertising. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (2021). <https://api.semanticscholar.org/CorpusID:235364004>
- [30] Akash Kumar Mohankumar, Bhargav Dodla, K Gururaj, and Amit Singh. 2022. Unified Generative & Dense Retrieval for Query Rewriting in Sponsored Search. *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management* (2022). <https://api.semanticscholar.org/CorpusID:259075708>
- [31] Akash Kumar Mohankumar, K Gururaj, Gagan Madan, and Amit Singh. 2024. Improving Retrieval in Sponsored Search by Leveraging Query Context Signals. <https://api.semanticscholar.org/CorpusID:271310286>
- [32] Ronak Pradeep, Kai Hui, Jai Gupta, Adam Dániel Lelkes, Honglei Zhuang, Jimmy J. Lin, Donald Metzler, and Vinh Q. Tran. 2023. How Does Generative Retrieval Scale to Millions of Passages?. In *Conference on Empirical Methods in Natural Language Processing*. <https://api.semanticscholar.org/CorpusID:258822999>
- [33] Weizhen Qi, Yeyun Gong, Jian Jiao, Yu Yan, Weizhu Chen, Dayiheng Liu, Kewen Tang, Houqiang Li, Jiusheng Chen, Ruofei Zhang, et al. 2021. Bang: Bridging autoregressive and non-autoregressive generation with large scale pretraining. In *International Conference on Machine Learning*. PMLR, 8630–8639.
- [34] Weizhen Qi, Yeyun Gong, Yu Yan, Jian Jiao, Bo Shao, Ruofei Zhang, Houqiang Li, Nan Duan, and M. Zhou. 2020. ProphetNet-Ads: A Looking Ahead Strategy for Generative Retrieval Models in Sponsored Search Engine. *ArXiv abs/2010.10789* (2020). <https://api.semanticscholar.org/CorpusID:222179129>
- [35] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan H. Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. 2023. Recommender Systems with Generative Retrieval. *arXiv:2305.05065* [cs.IR]
- [36] Chitwan Saharia, William Chan, Saurabh Saxena, and Mohammad Norouzi. 2020. Non-Autoregressive Machine Translation with Latent Alignments. In *Conference on Empirical Methods in Natural Language Processing*. <https://api.semanticscholar.org/CorpusID:215786391>
- [37] Yashas Samaga, Varun Yerram, Chong You, Srinadh Bhojanapalli, Sanjiv Kumar, Prateek Jain, and Praneeth Netrapalli. 2024. HiRE: High Recall Approximate Top-k Estimation for Efficient LLM Inference. *ArXiv abs/2402.09360* (2024). <https://api.semanticscholar.org/CorpusID:267657774>
- [38] Kyuhong Shim, Minjae Lee, Iksoo Choi, Yoonho Boo, and Wonyong Sung. 2017. SVD-Softmax: Fast Softmax Approximation on Large Vocabulary Neural Networks. In *Neural Information Processing Systems*. <https://api.semanticscholar.org/CorpusID:43626912>
- [39] Yubao Tang, Ruqing Zhang, J. Guo, Jianguo Chen, Zuowei Zhu, Shuaiqiang Wang, Dawei Yin, and Xueqi Cheng. 2023. Semantic-Enhanced Differentiable Search Index Inspired by Learning Strategies. *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2023). <https://api.semanticscholar.org/CorpusID:258865792>
- [40] Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. 2022. Transformer Memory as a Differentiable Search Index. *ArXiv abs/2202.06991* (2022). <https://api.semanticscholar.org/CorpusID:246863488>
- [41] Yujing Wang, Ying Hou, Hong Wang, Ziming Miao, Shibin Wu, Hao Sun, Qi Chen, Yuqing Xia, Chengmin Chi, Guoshuai Zhao, Zheng Liu, Xing Xie, Hao Sun, Weiwei Deng, Qi Zhang, and Mao Yang. 2022. A Neural Corpus Indexer for Document Retrieval. *ArXiv abs/2206.02743* (2022). <https://api.semanticscholar.org/CorpusID:249395549>
- [42] Hansi Zeng, Chen Luo, Bowen Jin, Sheikh Muhammad Sarwar, Tianxin Wei, and Hamed Zamani. 2023. Scalable and Effective Generative Information Retrieval. *ArXiv abs/2311.09134* (2023). <https://api.semanticscholar.org/CorpusID:265213270>
- [43] Chunting Zhou, Graham Neubig, and Jiatao Gu. 2019. Understanding Knowledge Distillation in Non-autoregressive Machine Translation. *ArXiv abs/1911.02727* (2019). <https://api.semanticscholar.org/CorpusID:207847275>
- [44] Shengyao Zhuang, Houxing Ren, Linjun Shou, Jian Pei, Ming Gong, G. Zuccon, and Daxin Jiang. 2022. Bridging the Gap Between Indexing and Retrieval for Differentiable Search Index with Query Generation. *ArXiv abs/2206.10128* (2022). <https://api.semanticscholar.org/CorpusID:249890267>

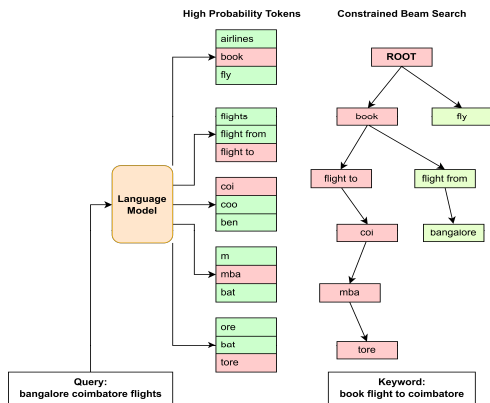


Figure 3: Trie-constrained beam forces the text generated by the model to always lie within the set of valid document identifiers.

A Constrained Beam Search

In generative retrieval, language models are forced to generate only within a closed set of document identifiers. For text generation models, this is typically achieved by incorporating a data structure such as a trie or an FM index into the decoding process.

The beam search decoding algorithm is commonly used to select multiple likely sequences of tokens. At each step, it maintains a set of prefixes and searches through the vocabulary to find the most likely next tokens for each prefix. The expanded prefixes are then pruned down to the beam size. The computational cost of beam search depends on the vocabulary size and the number of decoding steps. Given PIXNAR’s massive vocabulary, the standard beam search would be prohibitively expensive. To address this, we modify the algorithm to accept a list of preselected tokens at each position, removing the dependency on the vocabulary size. If this is efficiently computed, beam search becomes more feasible.

In our work, we use a trie constructed from all unique document identifiers to guide the language model’s generation. When considering whether a token can extend a prefix, the trie ensures that a document identifier can be reached from the expanded prefix. Figure 3 provides an illustration. Trie-constrained beam search accepts the top- k tokens and their probabilities at each position to generate a set of document identifiers.

Further, non-autoregressive models don’t need to generate tokens from left to right, as token probabilities at each position are computed simultaneously. This allows us to modify the beam search order to generate document identifiers, provided that there also exists a trie constructed in that order to ensure all generations fall within the set of document identifiers. We perform beam search in multiple orders when decoding non-autoregressive models. This technique is referred to as ‘permutation decoding’ in this paper.

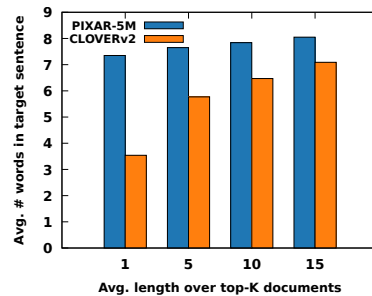


Figure 4: PIXNAR-5M generates longer sentences on average compared to CLOVERv2 (NQ dataset)

B Hyperparameter Study

This section details our hyperparameter study, focusing on the impact of key parameters on the performance and efficiency of our proposed PIXNAR model. We evaluated the influence of the shortlisting module’s hyperparameters (k , m , r) and the beam search parameters (beam size and length normalization factor) on retrieval accuracy and inference latency. These experiments were conducted on the MS MARCO dataset using an A100 GPU.

B.1 Shortlisting Module Hyperparameters

We investigated the impact of the shortlisting module’s hyperparameters: k (number of subsets), m (number of tokens per subset), and r (vocabulary subset size). Table 10 presents the results of this analysis. As expected, increasing r or k generally improves retrieval accuracy, but also leads to a slight increase in inference time. Notably, increasing m improves retrieval performance while slightly reducing inference time. This is likely due to the selection of more relevant subsets during the shortlisting process. The latency remained relatively stable across these parameters as the number of parameters used during inference is a small fraction of the LM head.

B.2 Beam Search Hyperparameters

We further analyzed the effect of beam size and the length normalization factor (A) on retrieval performance. Table 11 shows the results of this analysis. We observed that increasing the beam size yields diminishing returns in terms of retrieval accuracy. The length normalization factor, A , requires careful tuning to balance the model’s bias towards generating shorter or longer document identifiers. We found that a value of 2.5 for A provided the best trade-off between accuracy and the length of the generated identifiers.

C Vocabulary Construction

We adopt a two-stage approach: candidate selection followed by vocabulary construction, as proposed in TokenMonster [11]. Initially, we generate a set of potential token candidates, including words, sub-words and phrases, by considering all possible character substrings up to a specified maximum length. We then filter these substrings based on criteria such as adherence to word boundaries and consistency in character types (letters, numbers, punctuation,

Parameter	Value	MRR@10	R@100
m	256	22.2%	79.0%
	1024	23.8%	80.8%
	4096	24.0%	80.9%
r	1000	18.1%	71.4%
	10000	23.7%	80.5%
	20000	24.0%	80.9%
k	1	23.3%	79.8%
	3	23.9%	80.6%
	5	24.0%	80.9%

Table 10: Ablation study on parameters m, r, and k

Parameter	Value	MRR@10	R@100
Beam Size	50	4.1%	24.4%
	100	23.5%	78.9%
	200	23.9%	79.9%
	300	24.0%	80.9%
Length Norm	0	1.8%	16.8%
	2.5	24.0%	80.9%
	5	22.2%	79.6%

Table 11: Ablation study on Beam Size and Length Norm

Metric	CLOVERv2	128K	500K	1M	5M
Mean	10.98	5.56	4.78	4.46	4.05
99th	18	12	11	10	9

Table 12: Mean and 99th percentile sequence lengths for different vocabulary sizes. The number of tokens needed to tokenize a document identifier nearly halves.

etc.). Only tokens that exceed a minimum occurrence threshold are retained as potential candidates. In the second stage, we iteratively refine the candidate set to construct an optimal vocabulary of a specified size. We generate multiple test vocabularies from the pool of candidate tokens. Each test vocabulary is then used to tokenize the dataset, and a scoring system evaluates the efficiency of each token based on the total number of characters it compresses in the docid set. Tokens that perform poorly are removed, and the process is repeated until the desired vocabulary size is reached. Since we use the approach outlined in [11], we refer the reader to [11] for further implementation details.

D Implementation Details

1. Model Details: We initialize PIXNAR and the baseline CLOVERv2 model with the pretrained DeBERTa encoder [18]. We use the "microsoft/deberta-v3-base" checkpoint available on HuggingFace. For CLOVERv2, we use the provided 128K DeBERTa vocabulary for both the input and target. The language modeling head for PIXNAR must necessarily be initialized from scratch.

2. Document Identifiers: We employ the pseudo queries used in MINDER as our document identifiers. The total number of unique pseudo queries is around 80 million for the Natural Questions Wikipedia passages, and about 170 million for the MS MARCO

passages. In addition to using pseudo queries as our document identifiers, we also augment our training dataset by adding these pseudo queries as questions that map to other pseudo queries asked of the same passage. For each passage, we sample up to 20 pseudo queries and add them to the training dataset.

3. Training Details: All models were trained with a learning rate of 5×10^{-5} , 1000 warmup steps, and an effective batch size of 6400. Hyperparameters λ_3 (self-normalization loss scaling factor) and λ_2 (shortlist loss scaling factor) were set to 1.0 and 0.25. We used the AdamW optimizer with a linear decay learning rate scheduler with warmup. Models were trained for 5 epochs on the MSMARCO dataset and 10 epochs on the Natural Questions dataset. We used HuggingFace Trainer with a DeepSpeed wrapper for training.

4. Compute: We trained models using a 5M target vocabulary on 8 Nvidia H100 GPUs and models of all other vocabulary sizes on 16 AMD Mi200 GPUs. Inference experiments were all carried out on an NVIDIA Tesla T4 GPU. Training time was approximately 3 days on 8xH100s for the 5M vocabulary.

5. Selected Hyperparameters: We set the shortlisting module hyperparameters m, r, k to 4096, 20000 and 5 respectively. The length normalization hyperparameters A, a, b are set to 2.5, 6, 5.

6. Vocabulary Construction: For PIXNAR, we construct a target vocabulary of 5 million tokens using the method described in Section 4.1. We construct separate vocabularies for MS MARCO and NQ datasets, on the full set of document identifiers for each dataset. TokenMonster binaries were used to construct the vocabulary. Vocabularies were built using phrases, words, and sub-words that met frequency and length criteria, pruned iteratively based on token scoring. More details are in TokenMonster. The "min-occur" parameter was set to 20 for constructing the PIXNAR vocabulary, ensuring that candidate phrases occur at least 20 times in the document identifier corpus. While constructing the vocabulary, we use "strict" mode, in order to prevent minor variations of a phrase from receiving multiple tokens in the vocabulary.

7. Sequence Lengths and Target Sentence Lengths: NAR models often generate document identifiers that are sometimes too brief to convey significant semantics. By contrast, PIXNAR generates longer and more relevant target sentences, by generating phrases directly instead of subwords and words. Figure 4 presents the aggregated results that shows that PIXNAR (5M vocabulary) halves sequence lengths. The phrase-based tokens in PIXNAR have another benefit: they enable the generation of longer and relevant target sentences using fewer tokens, thereby enhancing generation quality. Table ?? illustrates how the sequence lengths of the target tokens decrease as vocabulary sizes increase. Notably, the sequence lengths for the 128K vocabulary generated by PIXNAR's vocabulary construction algorithm results in fewer token sequence lengths compared to CLOVERv2 which uses DeBERTa tokenization.

8. Trie Constrained Beam Search: We use a trie structure to guide beam search for efficient keyword generation. The trie is built on unique document identifiers (docids), with pruning at each step to maintain the most probable paths and ensure valid docids. We use an optimized Marisa trie for beam search. Setting a log threshold of -12.0 led to more efficient search. Details on permutation decoding are available in the repository.