Faster Nearest Neighbor Machine Translation

Anonymous ACL submission

Abstract

*k*NN based neural machine translation (*k*NN-MT) has achieved state-of-the-art results in a variety of MT tasks. One significant shortcoming of *k*NN-MT lies in its inefficiency in identifying the *k* nearest neighbors of the query representation from the entire datastore, which is prohibitively time-intensive when the datastore size is large.

In this work, we propose **Faster *k*NN-MT** to address this issue. The core idea of Faster *k*NN-MT is to use a hierarchical clustering strategy to approximate the distance between the query and a data point in the datastore, which is decomposed into two parts: the distance between the query and the center of the cluster that the data point belongs to, and the distance between the data point and the cluster center. We propose practical ways to compute these two parts in a significantly faster manner. Through extensive experiments on different MT benchmarks, we show that **Faster *k*NN-MT** is faster than Fast *k*NN-MT (Meng et al., 2021a) and only slightly (1.2 times) slower than its vanilla counterpart, while preserving model performance as *k*NN-MT. Faster *k*NN-MT enables the deployment of *k*NN-MT models on real-world MT services.

1 Introduction

Recent years have witnessed the significant performance boost introduced by neural machine translation models (Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2014; Luong et al., 2015). The recently proposed *k*NN based neural machine translation (*k*NN-MT) (Khandelwal et al., 2020) has achieved state-of-the-art results across a wide variety of machine translation setups and datasets. The core idea behind *k*NN-MT is that at each decoding step, the model is required to incorporate the target tokens with *k* nearest translation contexts in a large constructed datastore. In short, *k*NN-MT refers to target tokens that come after similar translation contexts in the constructed datastore, leading to signficiant performance boost.

One significant shortcoming of *k*NN-MT lies in its inefficiency in identifying the *k* nearest neighbors from the whole target training tokens, which is prohibitively slow when the datastore is large. To tackle this issue, Meng et al. (2021a) proposed Fast *k*NN-MT. Fast *k*NN-MT evades the necessity of iterating over the entire datastore for the *K*NN search by first building smaller datastores for source tokens of a source sentence: for each source token, its datastore is limited to reference tokens of the same token type, rather than the entire corpus. The concatenation of the datastores for all source tokens are concatenated and mapped to corresponding target tokens, forming the final datastore at the decoding step. Fast *k*NN-MT is two-order faster than *k*NN-MT. However, Fast *k*NN-MT needs to retrieve *k* nearest neighbors of each query source token from all tokens of the same token type in the training set. This can be still time-consuming when the current source reference token is a high-frequency word (e.g., "is", "the") and its corresponding token-specific datastore is large. Additionally, the size of the datastore on the target side is propotional to the source length, making the model slow for long source inputs.

In this paper, we propose Faster *k*NN-MT to address the aforementioned issues. The core idea of Faster *k*NN-MT is that we propose a novel hierarchical clustering strategy to approximate the distance between the query and a data point in the datastore, which is decomposed into two parts: (1) the distance between the query and the center of the cluster that the data point belongs to, and (2) the distance between the data point and the cluster centroid. The proposed strategy is both every effective in time and space. For (1), the computational complexity is low since the number of clusters is significantly smaller than the size of the datastore; for (2), distances between a cluster centroid and all

constituent data points of that cluster can be computed in advanced and cached, making (b) also fast. Faster kNN-MT is also effective in space since it requires much smaller datastores than both Fast kNN-MT and vanilla kNN-MT. This makes it feasible to run the inference model with a larger batch-size, which leads to an additional speedup.

Extensive experiments show that Faster kNN-MT is only 1.2 times slower than standard MT model while preserving model performance. Faster kNN-MT makes it feasible to deploy kNN-MT models on real-world MT services.

The rest of this paper is organized as follows: we describe the background of kNN-MT and Fast kNN-MT in Section 2. The proposed Faster kNN-MT is detailed in Section 3. Experimental results are presented in Section 4. We briefly go through the related work in Section 5, followed by a brief conclusion in Section 6.

2 Background

2.1 kNN-MT

General MT. A general MT model translates a given input sentence $x = \{x_1, ..., x_n\}$ to a target sentence $y = \{y_1, ..., y_m\}$, where n and m are the length of the source and target sentences. For each token y_i, $(x, y_{1:i-1})$ is called *translation context*. Let h_* be the hidden representations for tokens, then the probability distribution over vocabulary v for token y_i, given the translation context, is:

$$p_{\text{MT}}(y_i|x, y_{1:i-1}) = \frac{\exp(h_{y_i}^T \cdot h_{i-1})}{\sum_v \exp(h_v^T \cdot h_{i-1})}. \quad (1)$$

Beam search (Bahdanau et al., 2014; Li and Jurafsky, 2016; Vijayakumar et al., 2016) is normally applied for decoding.

***k*NN-MT.** The general idea of kNN-MT is to combine the information from k nearest neighbors from a large-scale datastore S, when calculating the probability of generating y_i. Specifically, kNN-MT first constructs the datastore $\mathcal{S}$ using key-value pairs $(f(x, y_{1:i-1}), y_i)$, where the key is the mapping representation of the translation context h_{i-1} for all time steps of all sentences using function $f(\cdot)$, and the value is the gold target token y_i. The complete datastore is written as $\mathcal{S} = \{(k, v)\} = \{(f(x, y_{1:i-1}), y_i), \forall y_i \in y\}$. Then, for each query $q = f(x, y_{1:i-1})$, kNN-MT searches through the entire datastore $\mathcal{S}$ to retrieve k nearest translation contexts along with the corresponding target tokens $\mathcal{N} = \{k_j, v_j\}_{j=1}^k$. Last, the retrieved set is transformed to a probability distribution by normalizing and aggregating the negative ℓ_2 distances, $-d(\cdot, \cdot)$, using the softmax operator with temperature T. $p_{\text{kNN}}(y_i|x, y_{1:i-1})$ can be expressed as follows:

$$\begin{aligned} &p_{k\text{NN}}(y_i|x, y_{1:i-1}) \\ &= \frac{\sum_{(k_j,v_j)\in\mathcal{N}} \mathbb{1}_{y_i=v_j} \left\{\exp(-d(q, k_j)/T)\right\}}{Z}, \quad (2) \\ &Z = \sum_{(k_j,v_j)\in\mathcal{N}} \exp(-d(q, k_j)/T). \end{aligned}$$

The final probability for the next token in kNN-MT, $p(y_i|x, y_{1:i-1})$, is a linear interpolation of $p_{\text{MT}}(y_i|x, y_{1:i-1})$ and $p_{k\text{NN}}(y_i|x, y_{1:i-1})$ with a tunable hyper-parameter λ:

$$\begin{aligned} p(y_i|x, y_{1:i-1}) =& \lambda p_{k\text{NN}}(y_i|x, y_{1:i-1}) + \\ & (1 - \lambda) p_{\text{MT}}(y_i|x, y_{1:i-1}). \quad (3) \end{aligned}$$

The problem for kNN-MT is at each decoding step, a beam search with size B needs to perform $B \times k$ times nearest neighbor searches on the full datastore S. It is extremely time-intensive when the datastore size S or the beam size is large (Khandelwal et al., 2020).

2.2 Fast kNN-MT

To alleviate time complexity issue in kNN-MT, Meng et al. (2021a) proposed Fast kNN-MT, which constructs a significantly smaller datastore for the nearest neighbors. Fast kNN-MT consists of the following three steps (also illustrated on the right side of blue part in Figure 1).

Building a Smaller Source Side Datastore. For each source token in the test example, Fast kNN-MT limits the kNN search to tokens of the same token type, in contrast to the whole corpus as in vanilla kNN-MT. Specifically, for each source token of the current test sentence, Fast kNN-MT selects the top c nearest neighbors from tokens of the same token type in the the source token corpus, rather than from the whole corpus. The datastore on the source side D_{source} consists of selected nearest neighbors of all constituent tokens within the source sentence.

Transforming Source Datastore to Target Datastore. As kNN-MT collects the k nearest target tokens during inference, D_{source} needs to be transformed to a datastore on the target side. Meng et al.

(2021a) leverages the FastAlign toolkit (Dyer et al., 2013) to link each source token in D_{source} to its correspondence on the target side, forming D_{target}. Each instance in D_{target} is a tuple consisting of the aligned target token mapped from the source token and its high-dimensional representation.

Decoding. At each time step t, the representation h_{t-1} produced by the decoder is used to query the target side representations in D_{target} to search the k nearest target neighbors. Then the kNN-based decoding probability $p_{k\text{NN}}$ is computed according to the selected nearest neighbors. Since D_{target} is significantly smaller than the corpus as a datastore, which is used in kNN-MT, Fast kNN-MT is orders of magnitude faster than vanilla kNN-MT.

3 Our Proposed Method: Faster kNN-MT

We observe two key issues that hinders the running time efficiency in Fast kNN-MT: (1) To construct D_{source}, we need to go through all tokens in the training set of the same token type. It can still be time-consuming when the query token is a high-frequency word (e.g., "is", "the"). (2) The size of D_{target} can be large, as D_{source} combines c nearest neighbors of all input tokens, making it proportional to the size of the source input.

In this work, we propose Faster kNN-MT to tackle these issues. The core idea of our method is to enable a much faster kNN search through a hierarchical strategy. Faster kNN-MT first group tokens of the same type into clusters (in Section 3.1). Then, the distance between the query and a data point in the datastore is estimated by (1) the distance between the query and the centroid of the cluster that a data point belongs to, and (2) the distance between the data point and the cluster centroid (in Section 3.2). We provide an overview of our proposed method in Figure 1 and use an example (in Section 3.3) to demonstrate our method.

3.1 Obtaining D_{target} on the Target Side

Our first step is to construct a datastore on the source side. For each token type, we cluster all tokens in the training set of that token type into g different clusters. Clusters are obtained by using k-means clustering algorithm on the token representations, which are the last layer representations from a pretrained MT model as in vanilla kNN-MT. g is the hyperparameter. At test time, for a given source token, we make an approximating assumption that its nearest neighbors should all come from its nearest cluster. Experimental results show that this approximation works well. In this work, the nearest clusters are identified based on the ℓ_2 distance between the representation of the source token and the cluster centroid. We combine all selected nearest clusters of all constituent tokens of the source input to constitute the cluster-store on the source side, denoted by $D^{\text{cluster}}_{\text{source}}$.

We then construct the datastore on the target side, as the source datastore can not be readily used to search for nearest neighbors of target tokens during decoding. We directly map selected source clusters to their corresponding target clusters, since the target correspondence for each token in each source cluster can be readily obtained using FastAlign (Dyer et al., 2013). The target cluster corresponding to a source cluster is the union of all target tokens corresponding to source tokens in that source cluster. We denote the cluster-store on the target side as $D^{\text{cluster}}_{\text{target}}$. The concatenation of constituent data points in clusters within $D^{\text{cluster}}_{\text{target}}$ constitute the target datastore, denoted by D_{target}. In practice, the mapping between source and target clusters can be obtained in advance and cached.

3.2 Selecting kNN on the Target Side

We now have the target datastore, the next step is to run nearest neighbor search in each decoding step. k nearest neighbors of h_{t-1} from D_{target} is selected by ranking $d(h_{i-1}, z_j)$, the distance between the query representation h_{i-1} and a point z_j in the target datastore. To simplify notations and without loss of generality, below we will only consider a 1 nearest neighbor situation, we note k nearest neighbors can be computed in a similar way.

We obtain the index for the nearest data point by:

$$\text{index for 1 NN} = \operatorname*{argmin}_j d(h_{i-1}, z_j). \quad (4)$$

The key point of Faster kNN-MT is to approximately compute the distance $d(h_{i-1}, z_j)$ by decoupling it into two parts: (1) $d(c_l, h_{i-1})$, which is the distance between the h_{i-1} and the cluster centriod c_l that a given target point z_j belongs to; and (2) $d(c_l, z_j)$, which is the distance between the cluster centriod and the point z_j:

$$d(h_{i-1}, z_j) \approx d(c_l, h_{i-1}) + d(c_l, z_j). \quad (5)$$

In this work, to enable faster computations, we approximate the minimum of the addition of

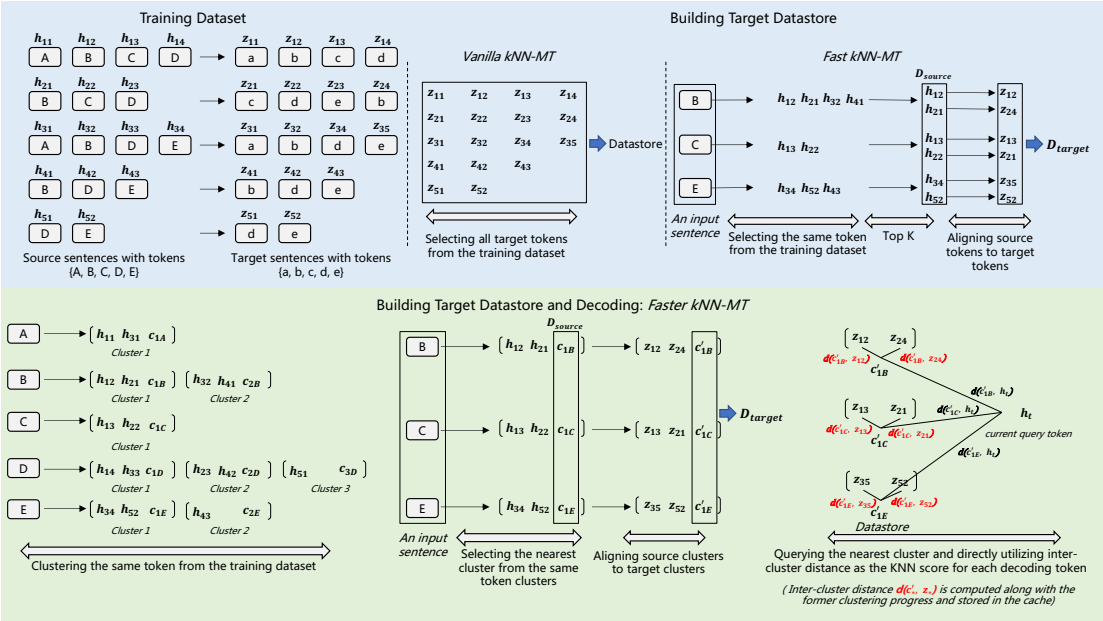

Figure 1: Comparison between vanilla kNN-MT, Fast kNN-MT and our proposed Faster kNN-MT. For our proposed Faster kNN-MT (bottom, green), there are three core steps. (1) *Clustering* (bottom, left): We cluster all occurrences of a particular token type from training set into g different groups. (2) *Datastore construction* (bottom, middle): Given a test example containing three tokens $\{B, C, E\}$, we first choose the nearest cluster for each source token. Then the selected clusters are aligned to their target clusters. The concatenation of all the centroids in the aligned target clusters constitutes the datastore for the current input. (3) *Decoding* (bottom, right): At each decoding step, we query the nearest cluster and directly use inter-cluster distances which is computed along with the former clustering progress and stored in the cache as the kNN score for each decoding token.

$d(c_l, h_{i-1})$ and $d(c_l, z_j)$ in Eq.5 by finding the minimum for each term: (1) finding the nearest cluster by $l = \text{argmin}_l\, d(c_l, h_{i-1})$, and (2) finding the nearest neighbor by $j = \text{argmin}_j\, d(c_l, z_j)$. This approximation works well because clusters in $D^{\text{cluster}}_{\text{target}}$ are distinct: recall when we construct $D^{\text{cluster}}_{\text{source}}$, for each source token, we find its nearest cluster from clusters of the same token type, and add the cluster to $D^{\text{cluster}}_{\text{source}}$. As clusters in $D^{\text{cluster}}_{\text{source}}$ are mapped to $D^{\text{cluster}}_{\text{target}}$ in one-to-one correspondence, clusters in $D^{\text{cluster}}_{\text{target}}$ should correspond to different token types, and are thus different.

We observe our two-step procedure for finding the minimum data point above is extremely computationally effective. We only need to go over $O(n)$ clusters for finding the nearest cluster. Ranks of data points based on distances to cluster centroid can be computed in advance and cached, meaning no computations required at the test time.

3.3 An Illustrative Example

In this section, we work through the example in Figure 1 (in green) to better illustrate our procedures. We assume that there are five kinds of source tokens $\{A, B, C, D, E\}$ and five kinds of target tokens $\{a, b, c, d, e\}$ in the training set. We use h_* and z_* for the representations of each token generated by the last layer of the pre-trained MT model in the source side and target side, respectively.

Obtaining D_{target} on the Target Side. We first cluster tokens of the same type in the training set into at most g clusters. In this example, we take g=3 and then generate clusters for tokens based on their hidden representations. In each cluster, besides the specific tokens, we also calculate the corresponding centroid of that cluster, denoted as $\{c_{\text{type}}\}$. For instance, for token B, we generate two clusters $\{h_{12}, h_{21}\}$ and $\{h_{32}, h_{41}\}$, and assign the cluster centroid c_{1B} and c_{2B} to these two clusters.

As we need to build a datastore on the target side for decoding, we now construct the cluster-store on the source side $D^{\text{cluster}}_{\text{source}}$, by querying the nearest cluster according to the distance between the representation of a specific token and the cluster centroid representations for this token. Suppose that the cluster $\{h_{12}, h_{21}, c_{1B}\}$ is the nearest cluster for token B, among two clusters of B. Similarly, we assume the cluster $\{h_{13}, h_{22}, c_{1C}\}$ is the nearest cluster for token C and the cluster $\{h_{34}, h_{52}, c_{1E}\}$ is the

nearest cluster for token E. The concatenation of all the tokens of above three selected clusters constitute the source side $D^{\text{cluster}}_{\text{source}}$ for the given sentence $\{B, C, E\}$.

To construct the cluster-store on the target side $D^{\text{cluster}}_{\text{target}}$, we use FastAlign toolkit for the constituted $D^{\text{cluster}}_{\text{source}} = \{\{h_{12}, h_{21}\}, \{h_{13}, h_{22}\}, \{h_{34}, h_{52}\}\}$ to find the mapped representation in the target side. Suppose that $\{\{z_{12}, z_{24}\}, \{z_{13}, z_{21}\}, \{z_{35}, z_{52}\}\}$ is the mapped set from $D^{\text{cluster}}_{\text{source}}$. We then associate the centroid for each target cluster c'_{1B}, c'_{1C}, and c'_{1E} after averaging all the representations of each target cluster. The target datastore D_{target} contains all centroids in $D^{\text{cluster}}_{\text{target}}$.

Selecting kNN on the Target Side. At each decoding step t, to collect the k nearest neighbors for the current decoding representation h_t, we first utilize h_t to query the nearest target cluster in the target datastore $D_{target} = \{c'_{1B}, c'_{1C}, c'_{1E}\}$ according to the distance $d(c'_{\text{type}}, h_t)$, type $\in \{1B, 1C, 1E\}$. We suppose that the cluster $1B$ is chosen for the current decoding representation h_t. Then we select k nearest neighbors in the inter-cluster representations of the target cluster $cluster^{target}_{1B} = \{z_{12}, z_{24}\}$ according to the inter-cluster distances $\{d(c'_{1B}, z_{12}), d(c'_{1B}, z_{24})\}$. All above distances are computed in advance.

3.4 Comparisons to Fast kNN-MT

We now compare the speed and space complexity of Faster kNN-MT against Fast kNN-MT.

Let g be the number of clusters, c be the number of nearest neighbors for NN search in D_{source}, F be the frequency of the source token, d be the representation dimensionality, and n be the length of the source sentence in the test example.

Time Complexity. For Fast kNN-MT, to construct datastore on the source side, it needs to search k-nearest neighbors from F source tokens on average and construct D_{source} with a size of cn with a time complexity of $O(Fdcn)$. For decoding, the size of D_{target} is the same as D_{source}. For each decoding step, it needs to search the k nearest neighbors from the datastore with size cn, making the time complexity for each decoding step being $O(kdcn)$. We assume that the length of the decoded target is very similar to the source length, i.e., n. The time complexity for decoding is thus $O(kdcn^2)$. Summing all, the time complexity for Fast kNN-MT is $O(Fdcn + kdcn^2)$.

For Faster kNN-MT, to construct $D^{\text{cluster}}_{\text{source}}$, we only need to search k-nearest clusters from g source clusters, which leads to a time complexity of $O(gdn)$ for a source of length n. For each token in the source, we only select the nearest neighbor, which leads to the size of $D^{\text{cluster}}_{\text{source}}$ being n. Due to the one-to-one correspondence between source cluster and target cluster, the size of $D^{\text{cluster}}_{\text{target}}$ is also n. At each decoding step, we search the nearest cluster from $D^{\text{cluster}}_{\text{target}}$, leading a time perplexity of $O(dn)$ for each step, and thus $O(dn^2)$ for the whole target. Since all distances and ranks are computed in advance and cached, nearest neighbors in the selected cluster are picked with $O(1)$ time perplexity. The Overall time complexity of Faster kNN-MT is $O(gdn + dn^2)$ which is significantly smaller than $O(Fdcn + kdcn^2)$ of Fast kNN-MT.

Space Complexity. For space complexity, for Fast kNN-MT, the size of D_{source} and D_{target} are both $c \times n$, leading to a space complexity $O(cnd)$, where d denotes the representation dimensionality; while for Faster kNN-MT, the size of $D^{\text{cluster}}_{\text{source}}$ or $D^{\text{cluster}}_{\text{target}}$ is n respectively, leading to a space complexity $O(nd)$. This significant saving in space let us increase the batch size with limited GPU memory, which also leads to a significant speedup.

4 Experiments

4.1 Datasets

We experiment with two types of datasets: traditional bilingual and domain adaptation datasets. Table 1 shows the statistics for these datasets.

Bilingual Datasets. We use WMT'14 English-French[1] and WMT'19 German-English.[2]. We follow protocols in Ng et al. (2019), including applying language identification filtering and only keep sentence pairs with correct language on both source and target side; removing sentences longer than 250 tokens as well as sentence pairs with a source/target length ratio exceeding 1.5; normalizing punctuation and tokenize all data with the Moses tokenizer (Koehn et al., 2007); and utilizing subword segmentation (Sennrich et al., 2016) doing joint byte pair encodings (BPE) with 32K split operations for WMT'19 German-English and 40K split operations for WMT'14 English-French.

[1] http://www.statmt.org/wmt19/translation-task.html

[2] http://www.statmt.org/wmt14/translation-task.html

Domain Adaptation Datasets. We use Medical, IT, Koran and Subtitles domains in the domain-adaptation benchmark (Koehn and Knowles, 2017). For each domain dataset, we split it into train/dev/test sets and clean these sets following protocols in (Aharoni and Goldberg, 2020).

4.2 Implementation Details

Base MT Model. We directly use the Transformer based model provided by the FairSeq (Ott et al., 2019) library as the vanilla MT model.[3] Both the encoder and the decoder have 6 layers. We set the dimensionality of word representations to be 1,024, the number of multi-attention heads to be 6 and the inner dimensionality of feedforward layers to be 8,192.

Quantization. To make sure all the token representations can be loaded into memory, we perform the product quantization (Jegou et al., 2010). For each token representation $x \in \mathbb{R}^D$, we first split it into M subvectors: $[x_1, x_2, ..., x_M]$ with the same dimension $d = D/M$. We then train the product quantizer using the following objective function:

$$\min_{q^1,...,q^M} \sum_{x} \sum_{m=1}^{M} \| x_m - q_m(x_m) \|^2, \qquad (6)$$

where $q_i (1 \leq i \leq M)$ denotes M sub-quantizers used to map a subvector $x_m \in \mathbb{R}^d$ to a codeword in a subcodebook C_m. Lastly, we leverage the M quantizers $q_1, ..., q_M$ to compress the high dimensional vector x to M codewords. We set M to be 128 in this work.

FAISS kNN Search. We use FAISS (Johnson et al., 2019), a toolkit for approximate nearest neighbor search, to speed up the process of KNN search. FAISS firstly samples N data points from the full dataset, which are clustered into M clusters. The remaining data in the dataset are then mapped to these M clusters. For a given query, it first queries the nearest cluster and does brute force search within this cluster. In this paper, we directly adopt the brute force search for tokens with frequency lower than 30,000; For tokens with frequency larger than 30,000, we do the search using FAISS toolkit for tokens.

Other Details We use the ℓ_2 distance to compute the similarity function in k-means clustering and use FAISS (Johnson et al., 2021) to cluster all reference tokens on the source side. The number of clusters for each source token type is set to f/m, where f is the frequency of the type token and m is the hyper-parameter controlling the number of clusters, which is set to 2,048.

[3] https://github.com/pytorch/fairseq/tree/master/examples/translation

4.3 Results on Bilingual Datasets

To tangibly understand the behavior of each module of Faster kNN, we conduct ablation experiments on the two WMT datasets by combining each module of Faster kNN respectively with Fast kNN. We experiment with the following two setups:

- **Fast kNN with Faster kNN's Source datastore**: We replace the source-side datastore of Fast kNN-MT with the datastore $D^{\text{cluster}}_{\text{source}}$ from Faster kNN-MT. This is to test the individual influence of clustering tokens of the same token type when constructing source side datastore, as opposed to using all tokens of the same token type as the datastore in Fast kNN. More specifically, we first construct the cluster-store on the source side $D^{\text{cluster}}_{\text{source}}$ as in Faster kNN-MT. Then, we conduct the token-level mapping to map the source side cluster-store $D^{\text{cluster}}_{\text{source}}$ to target side datastore D_{target}. D_{target} is then integrated to Fast kNN-MT as the target datastore for each decoding step.

- **Faster kNN without Cached Inter-cluster distance:** At each decoding step, we obtain the top-k nearest neighbors of a target query by directly computing the distance the query with data points in $D^{\text{cluster}}_{\text{target}}$, instead of using cached inter-cluster distance for speed-up purposes. This is to test whether the inter-cluster distance approximation in Equation (5) for selecting top-k nearest neighbors results in a performance loss. Specifically, as in Faster kNN, we use the target side cluster-store $D^{\text{cluster}}_{\text{target}}$ mapped from $D^{\text{cluster}}_{\text{source}}$ as the target side datastore. At each decoding step, instead of selecting the top-k points based on inter-cluster distances as in Faster kNN, we select the top-1 nearest cluster from $D^{\text{cluster}}_{\text{target}}$ and chose top-k nearest target token representations by directly computing the distance between h_t and data points.

Main Results. We report the SacreBLEU scores (Post, 2018) in Table 2. We observe that our proposed Faster kNN-MT model achieves compara-

	Bilingual Translation		*Domain Adaptation*			
	WMT'14 En-Fr	**WMT'19 Ge-En**	**Medical**	**IT**	**Koran**	**Subtitles**
Sentence pairs	35M	32M	0.25M	0.22M	0.02M	0.5
Maximum source sentence length	250	250	469	704	252	65
Average source sentence length	31.8	27.9	13.9	9.0	19.7	7.6
Number of tokens	1.1G	0.9G	3.5M	2.0M	0.3M	3.9M
Number of token types	44K	42K	18K	21K	7K	23K
Maximum token frequency	62M	40M	0.18M	0.11M	0.03M	0.4M
Average token frequency	26K	23.8K	374	182	74	237

Table 1: Dataset statistics for bilingual translation datasets and domain adaptation datasets.

ble BLEU scores to vanilla kNN-MT and Fast kNN-MT on English-French and German-English datasets, but with a significant speedup.

In Figure 2, we show the speed comparison between vanilla MT, Fast kNN-MT and Faster kNN-MT. Results for vanilla kNN-MT are just omitted as it is two orders of magnitude slower than vanilla MT (Khandelwal et al., 2020; Meng et al., 2021a). For Fast kNN-MT we observe that the speed advantage gradually diminishes as the number of nearest neighbors in D_{source} increases. For Faster kNN-MT, since the size of datastore D_{target} used at each decoding step is fixed to the length of source test sentence, it would not suffer speed diminishing when the length of the input source get greater.

Fast kNN-MT with Faster kNN's source side cluster-store $D^{\text{cluster}}_{\text{source}}$. To build datastore D_{source}, for each source token in the test example, Fast kNN-MT selects the top c nearest neighbors from tokens of the same token type in the source token corpus. Note that not all the c nearest neighbors can be clustered in the same one cluster in $D^{\text{cluster}}_{\text{source}}$, and that Faster kNN's only picks one cluster on the source side. The results for that setup is thus lower than Fast kNN-MT. For speed comparison between this setup and Fast kNN-MT shown in figure 2, since the size of D_{target} used at each decoding step for the two setups is approximately equal, the time consumption is almost equal.

Faster kNN-MT without cached inter-cluster distance. The BLEU scores on WMT German-English and WMT English-French datasets is comparable between the proposed setup,Fast kNN-MT and Faster kNN-MT. For speed comparison shown in figure 2, we can see that the speed of the current setup is faster than Fast kNN-MT but still slower than Faster kNN-MT, especially as the number of nearest neighbors queried at each decoding step increases. This result shows that the inter-cluster distance approximation in Eq.5 does improve the speed of Faster kNN-MT at each decoding step, while the performance loss is not significant.

Model	**De-En**	**En-Fr**
Base MT	37.6	41.1
+ kNN-MT	$39.1_{(+1.5)}$	$41.8_{(+0.7)}$
+ Fast kNN-MT	$39.3_{(+1.7)}$	$41.7_{(+0.6)}$
Faster kNN-MT	$39.3_{(+1.7)}$	$41.6_{(+0.5)}$
Ablation Experiments		
Fast kNN-MT + $D^{\text{cluster}}_{\text{source}}$	$39.1_{(+1.5)}$	$41.4_{(+0.3)}$
Faster kNN-MT - cached inter-cluster dist.	$39.5_{(+1.9)}$	$41.6_{(+0.5)}$

Table 2: SacreBLEU scores on WMT'14 En-Fr and WMT'19 Ge-En datasets.

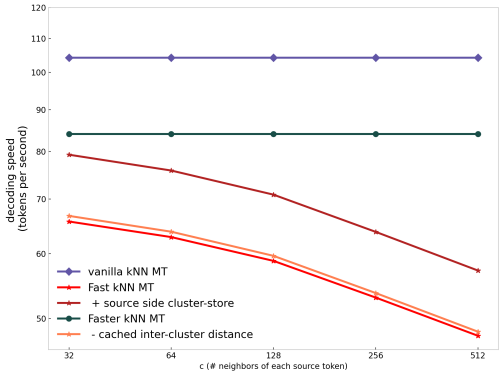

Figure 2: Speed comparison between Base MT, Fast kNN-MT, Faster kNN-MT and two ablation strategies.

4.4 Results on Domain Adaptation Datasets

For domain adaptation, we evaluate the base MT model and construct datastore within four German-English domain parallel datasets: Medical, IT, Koran and Subtitles, which are originally provided in (Koehn and Knowles, 2017). Results are shown in Table 3. We observe that our proposed Faster kNN-MT model achieves comparable BLEU scores to Fast kNN-MT and vanilla kNN-MT on the four datasets of domain adaption task, and similar to the performance on the two WMT datasets the time and space consumption of Faster kNN-MT are both

Model	Medical	IT	Koran	Subtitles	Average
Aharoni and Goldberg (2020)	54.8	43.5	21.8	27.4	47.2
Base MT	39.9	38.0	16.3	29.2	30.9
+ *k*NN-MT	54.4$_{(+14.5)}$	45.8$_{(+7.8)}$	19.4$_{(+3.1)}$	31.7$_{(+2.5)}$	37.8$_{(+6.9)}$
+ Fast *k*NN-MT	53.6$_{(+13.7)}$	45.5$_{(+7.5)}$	21.2$_{(+4.9)}$	30.5$_{(+1.3)}$	37.7$_{(+6.8)}$
+ Faster *k*NN-MT	52.7$_{(+12.8)}$	44.9$_{(+6.9)}$	20.4$_{(+4.1)}$	30.2$_{(+1.0)}$	37.1$_{(+6.2)}$

Table 3: SacreBLEU scores on four domain datasets: Medical, IT, Koran and Subtitles.

much smaller than Fast *k*NN-MT and vanilla *k*NN-MT.

5 Related Work

Neural Machine Translation. Recent advances on neural machine translation are build upon encoder-decoder architecture (Sutskever et al., 2014; Cho et al., 2014). The encoder infers a continuous representation of the source sentence, while the decoder is a neural language model conditioned on the encoder output. The parameters of both models are learned jointly to maximize the likelihood of the target sentences given the corresponding source sentences from a parallel corpus. More robust and expressive neural MT systems have also been developed (Guo et al., 2020; Zhu et al., 2020; Kasai et al., 2021a,b; Lioutas and Guo, 2020; Peng et al., 2021; Tay et al., 2021; Li et al., 2020; Liu et al., 2020; Nguyen and Salazar, 2019; Wang et al., 2019; Xiong et al., 2020b) based on attention mechanism (Bahdanau et al., 2014; Luong et al., 2015).

Retrieval Augmented Model. Retrieval augmented models additionally use the input to retrieve a set of relevant information, compared to standard neural models that directly pass the input to the generator. Prior works have shown the effectiveness of retrieval augmented models in improving the performance of a variety of natural language processing tasks, including language modeling (Khandelwal et al., 2019; Meng et al., 2021b), question answering (Guu et al., 2020; Lewis et al., 2020a,b; Xiong et al., 2020a), text classification (Lin et al., 2021), and dialog generation (Fan et al., 2020; Thulke et al., 2021; Weston et al., 2018).

For neural MT systems, Zhang et al. (2018) retrieves target *n*-grams to up-weight the reference probabilities. Bapna and Firat (2019) attend over neighbors similar to *n*-grams in the source using gated attention (Cao and Xiong, 2018). Tu et al. (2017) made a difference saving the former translation histories with the help of cache-based models (Grave et al., 2016), and the model thus can deal with a changing translation contexts.

There are also approaches improving the translation results by directly retrieving the example sentence in the training set. At the beginning of the machine translation, a lot of techniques focus on translating sentences by analogy (Nagao, 1981). These techniques identify the similar examples based on edit distance (Doi et al., 2005) and trigram contexts (Van Den Bosch et al., 2007). For recently, Gu et al. (2018) collected sentence pairs according to the given source sentence from the small subset of sentence pairs from the training set leveraging an off-the-shelf search engine. Since these techniques focus on sentence-level machining, they will be hard to handle facing large and changing contexts. To take more advantage of neural context representations, (Khandelwal et al., 2020) proposed *k*NN-MT that it simply collects all the target representations in the training set, and constructs a much larger datastore than the above approaches. However, the approaches described above mainly focus on either efficiency or performance. To benefit from retrieval augmented model without loss of efficiency, Meng et al. (2021a) proposed the Fast *k*NN-MT. This work offers a further speed-up than Fast *k*NN-MT.

6 Conclusion

In this paper, we propose Faster *k*NN-MT, a method to further speed up the previous Fast *k*NN-MT model. Our method improves the speed to only 1.2 times slower than base MT, compared to Fast *k*NN-MT which is 2 times slower. The core idea of Faster *k*NN-MT is to constrain the search space when constructing the datastore on both source side and target side. We leverages *k*-means clustering for only querying the centroid of each cluster instead of all examples from the datastore. Experiments demonstrate that this strategy is more efficient than Fast *k*NN-MT with minimal performance degradation.

References

Roee Aharoni and Yoav Goldberg. 2020. Unsupervised domain clusters in pretrained language models. *arXiv preprint arXiv:2004.02105*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Ankur Bapna and Orhan Firat. 2019. Non-parametric adaptation for neural machine translation. *arXiv preprint arXiv:1903.00058*.

Qian Cao and Deyi Xiong. 2018. Encoding gated translation memory into neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3042–3047, Brussels, Belgium. Association for Computational Linguistics.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Takao Doi, Hirofumi Yamamoto, and Eiichiro Sumita. 2005. Example-based machine translation using efficient sentence retrieval based on edit-distance. *ACM Transactions on Asian Language Information Processing (TALIP)*, 4(4):377–399.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A simple, fast, and effective reparameterization of IBM model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, Atlanta, Georgia. Association for Computational Linguistics.

Angela Fan, Claire Gardent, Chloe Braud, and Antoine Bordes. 2020. Augmenting transformers with knn-based composite memory for dialogue. *arXiv preprint arXiv:2004.12744*.

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2016. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*.

Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor OK Li. 2018. Search engine guided neural machine translation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Junliang Guo, Zhirui Zhang, Linli Xu, Hao-Ran Wei, Boxing Chen, and Enhong Chen. 2020. Incorporating bert into parallel sequence decoding with adapters. *arXiv preprint arXiv:2010.06138*.

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

Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. 2021a. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation.

Jungo Kasai, Hao Peng, Yizhe Zhang, Dani Yogatama, Gabriel Ilharco, Nikolaos Pappas, Yi Mao, Weizhu Chen, and Noah A Smith. 2021b. Finetuning pretrained transformers into rnns. *arXiv preprint arXiv:2103.13076*.

Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Nearest neighbor machine translation. *arXiv preprint arXiv:2010.00710*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2019. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, pages 177–180.

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation.

Mike Lewis, Marjan Ghazvininejad, Gargi Ghosh, Armen Aghajanyan, Sida Wang, and Luke Zettlemoyer. 2020a. Pre-training via paraphrasing. *arXiv preprint arXiv:2006.15020*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020b. Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2005.11401*.

Jiwei Li and Dan Jurafsky. 2016. Mutual information and diverse decoding improve neural machine translation. *arXiv preprint arXiv:1601.00372*.

Xiaoya Li, Yuxian Meng, Mingxin Zhou, Qinghong Han, Fei Wu, and Jiwei Li. 2020. Sac: Accelerating and structuring self-attention via sparse adaptive connection. *arXiv preprint arXiv:2003.09833*.

Yuxiao Lin, Yuxian Meng, Xiaofei Sun, Qinghong Han, Kun Kuang, Jiwei Li, and Fei Wu. 2021. Bertgcn: Transductive text classification by combining gcn and bert. *arXiv preprint arXiv:2105.05727*.

Vasileios Lioutas and Yuhong Guo. 2020. Time-aware large kernel convolutions. In *International Conference on Machine Learning*, pages 6172–6183. PMLR.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. 2020. Understanding the difficulty of training transformers. *arXiv preprint arXiv:2004.08249*.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation.

Yuxian Meng, Xiaoya Li, Xiayu Zheng, Fei Wu, Xiaofei Sun, Tianwei Zhang, and Jiwei Li. 2021a. Fast nearest neighbor machine translation.

Yuxian Meng, Shi Zong, Xiaoya Li, Xiaofei Sun, Tianwei Zhang, Fei Wu, and Jiwei Li. 2021b. Gnn-lm: Language modeling based on global contexts via gnn. *arXiv preprint arXiv:2110.08743*.

Makoto Nagao. 1981. A framework of a mechanical translation between japanese and english by analogy principle.

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook fair's wmt19 news translation task submission.

Toan Q Nguyen and Julian Salazar. 2019. Transformers without tears: Improving the normalization of self-attention. *arXiv preprint arXiv:1910.05895*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. 2021. Random feature attention. *arXiv preprint arXiv:2103.02143*.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2021. Synthesizer: Rethinking self-attention for transformer models. In *International Conference on Machine Learning*, pages 10183–10192. PMLR.

David Thulke, Nico Daheim, Christian Dugast, and Hermann Ney. 2021. Efficient retrieval augmented generation from unstructured knowledge for task-oriented dialog. *arXiv preprint arXiv:2102.04643*.

Zhaopeng Tu, Yang Liu, Shuming Shi, and Tong Zhang. 2017. Learning to remember translation history with a continuous cache.

Antal Van Den Bosch, Nicolas Stroppa, and Andy Way. 2007. A memory-based classification approach to marker-based ebmt.

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

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F Wong, and Lidia S Chao. 2019. Learning deep transformer models for machine translation. *arXiv preprint arXiv:1906.01787*.

Jason Weston, Emily Dinan, and Alexander H Miller. 2018. Retrieve and refine: Improved sequence generation models for dialogue. *arXiv preprint arXiv:1808.04776*.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2020a. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv preprint arXiv:2007.00808*.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. 2020b. On layer normalization in the transformer architecture.

Jingyi Zhang, Masao Utiyama, Eiichro Sumita, Graham Neubig, and Satoshi Nakamura. 2018. Guiding neural machine translation with retrieved translation pieces. *arXiv preprint arXiv:1804.02559*.

Jinhua Zhu, Yingce Xia, Lijun Wu, Di He, Tao Qin, Wengang Zhou, Houqiang Li, and Tie-Yan Liu. 2020. Incorporating bert into neural machine translation. *arXiv preprint arXiv:2002.06823*.