Fast Nearest Neighbor Machine Translation

Yuxian Meng♣, Xiaoya Li♣, Xiayu Zheng▲, Fei Wu♠♥
Xiaofei Sun♠♣, Tianwei Zhang♦, Jiwei Li♠♣
♠College of Computer Science and Technology, Zhejiang University, ♣Shannon.AI
▲Peking University, ♦Nanyang Technological University
♥Shanghai Institute for Advanced Study of Zhejiang University, ♥Shanghai AI Laboratory
{yuxian_meng, xiaoya_li, xiaofei_sun, jiwei_li}@shannonai.com
xiayu_zheng@pku.edu.cn, tianwei.zhang@ntu.edu.sg, wufei@zju.edu.cn

Abstract

Though nearest neighbor Machine Translation (kNN-MT) (Khandelwal et al., 2020) has proved to introduce significant performance boosts over standard neural MT systems, it is prohibitively slow since it uses the entire reference corpus as the datastore for the nearest neighbor search. This means each step for each beam in the beam search has to search over the entire reference corpus. kNN-MT is thus two-orders slower than vanilla MT models, making it hard to be applied to real-world applications, especially online services. In this work, we propose Fast kNN-MT to address this issue. Fast kNN-MT constructs a significantly smaller datastore for the nearest neighbor search: for each word in a source sentence, Fast kNN-MT first selects its nearest token-level neighbors, which is limited to tokens that are the same as the query token. Then at each decoding step, in contrast to using the entire corpus as the datastore, the search space is limited to target tokens corresponding to the previously selected reference source tokens. This strategy avoids search through the whole datastore for nearest neighbors and drastically improves decoding efficiency. Without loss of performance, Fast kNN-MT is two-orders faster than kNN-MT, and is only two times slower than the standard NMT model. Fast kNN-MT enables the practical use of kNN-MT systems in real-world MT applications. [1]

1 Introduction

Machine translation (MT) is a fundamental task in natural language processing (Brown et al., 1993; Och and Ney, 2003), and the prevalence of deep neural networks has spurred a diverse array of neural machine translation (NMT) models to improve translation quality (Sutskever et al., 2014; Bahdanau et al., 2014; Vaswani et al., 2017). The recently proposed k nearest neighbor (kNN) MT

model (Khandelwal et al., 2020) has proved to introduce significant performance boosts over standard neural MT systems. The basic idea behind kNN-MT is that at each decoding step, the model is allowed to refer to reference target tokens with similar translation contexts in a large datastore of cached examples. The corresponding reference target tokens provide important insights on the translation token likely to appear next.

One notable limitation of kNN-MT is that it is prohibitively slow: it uses the entire reference corpus as the datastore for the nearest neighbor search. This means each step for each beam in the beam search has to search over the entire reference corpus. kNN-MT is thus two-orders slower than vanilla MT models. The original paper of kNN-MT (Khandelwal et al., 2020) suggests using fewer searching clusters, smaller beams and smaller datastores for generation speedup, but to achieve satisfactory results, carefully tuning on these factors under different tasks and datasets is still required according to analyses in (Khandelwal et al., 2020). The computational overhead introduced by kNN-MT makes it hard to be deployed on real-world online services, which usually require both model performance and runtime efficiency.

In this work, we propose a fast version of kNN-MT – Fast kNN-MT, to tackle the aforementioned issues. Fast kNN-MT constructs a significantly smaller datastore for the nearest neighbor search: for each word in a source sentence, Fast kNN-MT first selects its nearest token-level neighbors, which is limited to tokens of the same token type. Then at each decoding step, in contrast to consulting the entire corpus for nearest neighbor search, the datastore for the currently decoding token is limited within the tokens of reference targets corresponding to the previously selected reference source tokens, as shown in Figure 1. The chain of mappings from the target token to the source token, then to its nearest source reference tokens, and last to cor-

---

responding target reference tokens, can be obtained using FastAlign (Dyer et al., 2013).

Fast kNN-MT provides several important advantages against vanilla kNN-MT in terms of speedup: (1) the datastore in the KNN search is limited to target tokens corresponding to previously selected reference source tokens, instead of the entire corpus. This significantly improves decoding efficiency; (2) for source nearest neighbor retrieval, we propose to restrict the reference sources tokens that are the same as the query token, which further improves nearest-neighbor search efficiency. Without loss of performance, Fast kNN-MT is two-orders faster than kNN-MT, and is only two times slower than standard MT model. Under the settings of bilingual translation and domain adaptation, Fast kNN-MT achieves comparable results to kNN-MT, leading to a SacreBLEU score of 39.3 on WMT'19 De-En, 41.7 on WMT'14 En-Fr, and an average score of 41.4 on the domain adaptation task.

2 Related Work

Neural Machine Translation
Neural machine translation systems (Vaswani et al., 2017; Gehring et al., 2017; Meng et al., 2019) are often implemented by the sequence-to-sequence framework (Sutskever et al., 2014) and enhanced with the attention mechanism (Bahdanau et al., 2014; Luong et al., 2015) which associates the current decoding token to the most semantically related part in the source side. At decoding time, beam search and its variants are used to find the optimal sequence (Sutskever et al., 2014; Li and Jurafsky, 2016). The development of self-attention (Vaswani et al., 2017) and pretraining (Devlin et al., 2018; Lewis et al., 2019) has greatly motivated a line of works for more expressive MT systems. These works include incorporating pretrained models (Zhu et al., 2020; Guo et al., 2020), designing lightweight model structures (Kasai et al., 2020; Lioutas and Guo, 2020; Tay et al., 2020; Kasai et al., 2021; Peng et al., 2021), handling multiple languages (Aharoni et al., 2019; Arivazhagan et al., 2019; Liu et al., 2020b) and mitigating structural issues in Transformers (Wang et al., 2019; Nguyen and Salazar, 2019; Liu et al., 2020a; Li et al., 2020; Xiong et al., 2020b) for more robust and efficient NMT systems.

Retrieval-Augmented Models
Retrieving and integrating auxiliary sentences has shown effectiveness in improving robustness and expressiveness for NMT systems. (Zhang et al., 2018) up-weighted the output tokens by collecting from the retrieved target sentences n-grams that align with the words in the source sentence, and (Bapna and Firat, 2019) similarly retrieved n-grams but incorporated the information using gated attention (Cao and Xiong, 2018). (Tu et al., 2018) updated and stored the hidden representations of recent translation history in cache for access when new tokens are generated, so that the model can dynamically adapt to different contexts. (Gu et al., 2018) leveraged an off-the-shelf search engine to retrieve a small subset of sentence pairs from the training set and then perform translation given the source sentence along with the retrieved pairs. (Li et al., 2016; Farajian et al., 2017) proposed to retrieve similar sentences from the training set for the purpose of adapting the model to different input sentences. (Bulté and Tezcan, 2019; Jitao et al., 2020) used fuzzy matches to retrieve similar sentence pairs from translation memories and augmented the source sentence with the retrieved pairs. Our work is motivated by kNN-MT (Khandelwal et al., 2020) and target improving the efficiency of kNN retrieval while achieving comparable translation performances.

Apart from machine translation, other NLP tasks have also benefited from retrieval-augmented models, such as language modeling (Khandelwal et al., 2019), question answering (Guu et al., 2020; Lewis et al., 2020b,a; Xiong et al., 2020a) and dialog generation (Weston et al., 2018; Fan et al., 2020; Thulke et al., 2021). Most of these works perform retrieval at the sentence level and treat the extracted sentences as additional input for model generation, whereas fast kNN-MT retrieves the most relevant tokens in the source side and fixes the probability distribution using the aligned target tokens at each decoding step.

3 Background: kNN-MT

Given an input sequence of tokens $\boldsymbol{x} = \{x_1, \cdots, x_n\}$ of length n, an MT model translates it into a target sentence in another language $\boldsymbol{y} = \{y_1, \cdots, y_m\}$ of length m. A common practice to produce each token y_i on the target side is to obtain a probability distribution over the vocabulary $p_{\text{MT}}(y_i|\boldsymbol{x}, \boldsymbol{y}_{1:i-1})$ from the decoder and use beam search for generation. The combination of the complete source sentence and prefix of the target sentence $(\boldsymbol{x}, \boldsymbol{y}_{1:i-1})$ is called *trans-*

lation context. kNN-MT interpolates this probability distribution with a multinomial distribution $p_{\text{kNN}}(y_i|\boldsymbol{x}, \boldsymbol{y}_{1:i-1})$ derived from the k nearest neighbors of the current translation context $(\boldsymbol{x}, \boldsymbol{y}_{1:i-1})$ from a large scale datastore $\mathcal{S}$:

$$
\begin{aligned}
p(y_i|\boldsymbol{x}, \boldsymbol{y}_{1:i-1}) &= \lambda p_{\text{kNN}}(y_i|\boldsymbol{x}, \boldsymbol{y}_{1:i-1}) \\
&+ (1-\lambda)p_{\text{MT}}(y_i|\boldsymbol{x}, \boldsymbol{y}_{1:i-1})
\end{aligned} \tag{1}
$$

More specifically, kNN-MT first constructs the datastore $\mathcal{S}$ using key-value pairs, where the key is the high-dimensional vector of the translation context produced by a trained MT model $f(\boldsymbol{x}, \boldsymbol{y}_{1:i-1})$, and the value is the corresponding gold target token y_i, forming $\mathcal{S} = \{(\boldsymbol{k}, v)\} = \{(f(\boldsymbol{x}, \boldsymbol{y}_{1:i-1}), y_i)\}$. The context-target pairs may come from any parallel corpus. Then, using the dense representation of the current translation context as query $\boldsymbol{q} = f(\boldsymbol{x}^{\text{in}}, \boldsymbol{y}_{1:i-1})$ and L_2 distance as measure, kNN-MT searches through the entire datastore $\mathcal{S}$ to retrieve k nearest translation contexts along with the corresponding target tokens $\mathcal{N} = \{\boldsymbol{k}_j, v_j\}_{j=1}^{k}$. Last, the retrieved set is transformed to a probability distribution by normalizing and aggregating the negative L_2 distances, $-d$, using the softmax operator with temperature T, which can be expressed as follows:

$$
\begin{aligned}
&p_{\text{kNN}}(y_i|\boldsymbol{x}^{\text{in}}, \boldsymbol{y}_{1:i-1}) \\
&= \frac{\sum_{(\boldsymbol{k}_j, v_j) \in \mathcal{N}} \mathbb{1}_{y_i = v_j} \exp(-d(\boldsymbol{q}, \boldsymbol{k}_j)/T)}{Z} \\
&Z = \sum_{(\boldsymbol{k}_j, v_j) \in \mathcal{N}} \exp(-d(\boldsymbol{q}, \boldsymbol{k}_j)/T)
\end{aligned} \tag{2}
$$

Integrating Eq.(2) into Eq.(1) gives the final probability of generating token y_i for time step i. Note that the above kNN search-interpolating process is applied to each decoding step of each beam, and each iteration needs to run on the full datastore $\mathcal{S}$. This gives a total time complexity of $\mathcal{O}(|\mathcal{S}|Bm)$, where B is the beam size and m is the target length. In order for faster nearest neighbor search, kNN-MT leverages FAISS (Johnson et al., 2019), an toolkit for efficient similarity search and clustering of dense vectors.

4   Method: Fast kNN-MT

The time complexity of kNN-MT before search optimization is $\mathcal{O}(|\mathcal{S}|Bm)^2$, which is prohibitively

---

²Time perplexity can slightly be alleviated when approximate nearest neighbor search is used, with computation perplexity being not strictly $\mathcal{O}(|\mathcal{S}|)$.

slow when the size of the datastore $\mathcal{S}$ or the beam size B is large. We propose strategies to address this issue. The same as vanilla kNN-MT, fast kNN-MT system is built upon a separately trained MT encoder-decoder model. To get a better illustration of how Fast kNN-MT works, we give a toy illustration in Figure 1. We use the capitalized characters to denote source tokens and lower-cased letters to denote target tokens. Given the training set, which is:

$$
\begin{aligned}
(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}) &= (\{\text{A,B,C,D}\}, \{\text{b,c,d,a}\}) \\
(\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}) &= (\{\text{B,C,D}\}, \{\text{c,d,e,b}\}) \\
(\boldsymbol{x}^{(3)}, \boldsymbol{y}^{(3)}) &= (\{\text{A,B,D,E}\}, \{\text{a,b,c,d,e}\}) \\
(\boldsymbol{x}^{(4)}, \boldsymbol{y}^{(4)}) &= (\{\text{B,D,E}\}, \{\text{b,d,e}\}) \\
(\boldsymbol{x}^{(5)}, \boldsymbol{y}^{(5)}) &= (\{\text{D,E,F}\}, \{\text{d,e,f}\})
\end{aligned} \tag{3}
$$

in the toy example, an encoder-decoder model is trained. Next, we wish to translate a source string $\{\text{B,C,E}\}$ at test time.

4.1   Datastore Creation On the Source Side

Given a pretrained encoder-decoder model, and the training corpus, we first obtain representations for all source tokens and target tokens of the training set, which are the last layer outputs from the encoder-decoder model. In the toy example, representations for source tokens $\{\text{A,B,C,D}\}$ in the first training example $(\{\text{A,B,C,D}\}, \{\text{b,c,d,a}\})$ are respectively $\boldsymbol{h}_{11}, \boldsymbol{h}_{12}, \boldsymbol{h}_{13}, \boldsymbol{h}_{14}$, and for target tokens $\{\text{b,c,d,a}\}$ are respectively $\boldsymbol{z}_{11}, \boldsymbol{z}_{12}, \boldsymbol{z}_{13}, \boldsymbol{z}_{14}$. Given a test example to translate, which is $\{\text{B,C,E}\}$ in the example, we also obtain the representation for each of its constituent token, denoted by $\boldsymbol{h}_{\text{B}}, \boldsymbol{h}_{\text{C}}, \boldsymbol{h}_{\text{E}}$. Next, we select nearest neighbor tokens for each source token, i.e., $\{\text{B, C, E}\}$. The nearest neighbor tokens are first limited to source tokens of the same token type as the query token. For token B, tokens of the same token type are x_{12}, x_{21}, x_{32}, x_{41}. Similarly, for the token C in the test example, tokens of the same type are $\{x_{13}, x_{22}\}$; for the token E, tokens of the same type are $\{x_{34}, x_{43}, x_{52}\}$. One issue that stands out is that, for common words such as "the", there can be tens of millions of the same type tokens in the training corpus. We thus need to further limit the number of nearest neighbors. Let c denote the hyper-parameter that controls the number of nearest neighbors for each token on the source side, which is set to 2 in the toy example. We rank all candidates based on the distance between the
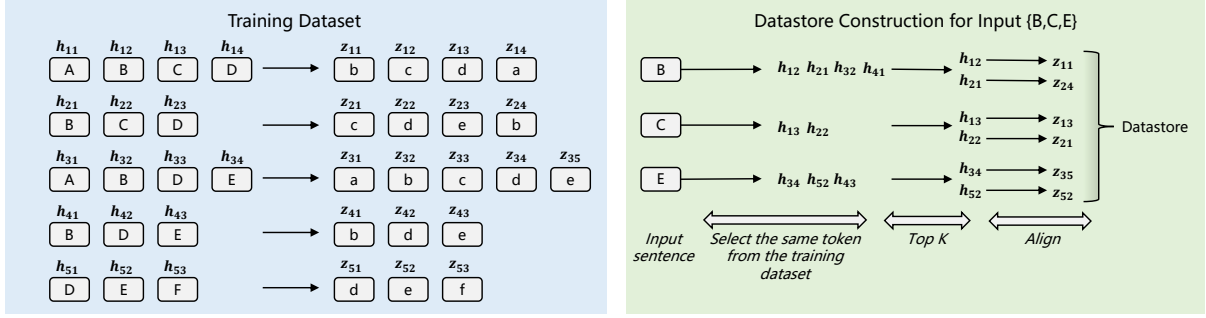
Figure 1: *Caching source and target tokens* (left, blue): Given a trained NMT model f and the training corpus $\mathcal{D}_{\text{train}}$, we obtain representations for all source tokens h and target tokens z in the training set, which are the last layer outputs from f. *Datastore construction* (right, green): Given a test example to translate, which is {B,C,E} in the example, we first navigate each source token to the tokens of the same type in the cache, e.g., x_{12}, x_{21}, x_{32} and x_{41} are identified for token B. Then, the top c nearest neighbors for each source token are preserved according to the distance between the source token representation and candidate token representations, e.g., x_{12}, x_{21} are selected for token B. Last, the selected source tokens are aligned to their target tokens using FastAlign (Dyer et al., 2013). For token B, the aligned target tokens are y_{11}, y_{24}. The collection of all aligned target tokens (along with their representations) constitutes the datastore for the current input {B,C,E}.

source token representation (e.g., h_B, h_C, h_E) and candidate token representations, and select the top c. Suppose that in the toy example, x_{12}, x_{21} are selected for token B, x_{13}, x_{23} are selected for token C, x_{34}, x_{52} are selected for token E. The concatenation of selected candidates for all source tokens constitute the datastore on the source side, which is $\mathcal{D}_{\text{source}} = \{x_{12}, x_{21}, x_{13}, x_{23}, x_{34}, x_{52}\}$ in the toy example. The datastore creation for source tokens (e.g., {B, C, E}) can be run in parallel.

4.2 Datastore Creation On the Target Side

For decoding, the model needs to refer to reference target tokens rather than source tokens. We thus need to transform $\mathcal{D}_{\text{source}}$ to a list of target tokens. We use FastAlign (Dyer et al., 2013) toolkit to achieve this goal. FastAlign maps source tokens to target tokens based on the IBM model (Och and Ney, 2003). Source tokens in $\mathcal{D}_{\text{source}}$ that do not have correspondence on the target side are abandoned. Output target tokens from FastAlign form the datastore on the target side, denoted by $\mathcal{D}_{\text{target}}$. In the toy example, $x_{12}, x_{21}, x_{13}, x_{23}, x_{34}, x_{52}$ are respectively mapped to $\mathcal{D}_{\text{target}} = \{y_{11}, y_{24}, y_{12}, y_{21}, y_{35}, y_{52}\}$. The size of $\mathcal{D}_{\text{target}}$ is $c \times n$, where n is the source length.

In practice, we first iterate over all examples in the training set, extracting all the source token representations and all the target token representations. Then, we build a separate token-specific cache $\mathcal{D}_v$ for each v in vocabulary, which consists of (key,

value) pairs where the key is the high-dimensional representation h and the value is a binary tuple containing the corresponding aligned target token along with its representation z. Then we could map each source token of a given input sentence to its corresponding cache $\mathcal{D}_v$, and build the target-side datastore following the steps in Section 4.1 and Section 4.2. The process of caching source and target tokens is present in Algorithm 1.

4.3 Decoding

At the decoding time, the datastore for each decoding step is all limited to $\mathcal{D}_{\text{target}}$, within which kNN search is performed. Since tokens in $\mathcal{D}_{\text{target}}$ are not all related to the current decoding, nearest neighbor search is performed to select the top k candidates from $\mathcal{D}_{\text{target}}$ for each decoding step. For the nearest neighbor search here, we use the current representation h at the decoding time to query target representation z for target tokens in $\mathcal{D}_{\text{target}}$. The selected nearest neighbors and their representations are used to compute the final word generation probability based on Eq.(1) and Eq.(2).

4.4 Quantization

Although the prohibitive computational cost issue of kNN-MT has been addressed, the intensive memory for datastore remains a problem, as we wish to cache all source and target representations of the entire training set. Additionally, frequently accessing Terabytes of data is also extremely time-intensive. To address this issue, we propose to use product quantization (PQ) (Jegou et al., 2010) to

Algorithm 1: Constructing Datastore for a Test Input $\boldsymbol{x}$.

> **Input** : All sentence-pairs in training set:
> $(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), ..., (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)})$, vocabulary $\mathcal{V}$
> NMT encoder f_e, NMT decoder f_d, word alignment for each sentence-pair $(A^{(1)}, ..., A^{(N)})$
> Input for test: $\boldsymbol{x}$
>
> **Output** : Target datastore $\mathcal{D}_{\text{target}}(\boldsymbol{x})$ for $\boldsymbol{x}$
>
> $\mathcal{D}_{\text{target}}(\boldsymbol{x}) \leftarrow \emptyset$ ► initialize the datastore $\mathcal{D}_{\text{source}}(\boldsymbol{x})$ for $\boldsymbol{x}$
>
> **for** $v \leftarrow 1$ **to** $\mathcal{V}$ **do**
> $\mathcal{D}_v \leftarrow \emptyset$ ► initialize the (key, value) datastore for each word in the vocabulary
> **end**
>
> *% Caching Source and Target Tokens*:
> **for** $i \leftarrow 1$ **to** N **do**
> $n_i \leftarrow$ length of $\boldsymbol{x}^{(i)}$
> $m_i \leftarrow$ length of $\boldsymbol{y}^{(i)}$
> $[\boldsymbol{h}_1, ..., \boldsymbol{h}_{n_i}] \leftarrow f_e(\boldsymbol{x}^{(i)})$ ► computing representations for each source word
> $[\boldsymbol{z}_1, ..., \boldsymbol{z}_{m_i}] \leftarrow f_d(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ ► computing representations for each target word
>   **for** $j \leftarrow 0$ **to** n_i **do**
>   **for** $k \leftarrow 0$ **to** m_i **do**
>   **if** $(j, k) \in A^{(i)}$ **then**
>   add $(\boldsymbol{h}_i, (\boldsymbol{z}_k, y_k^{(i)}))$ to $\mathcal{D}_{x_j^{(i)}}$
>   *% key of $\mathcal{D}_{x_j^{(i)}}$ are all hidden representations $\boldsymbol{h}_i$ for tokens of the same token type as $x_j^{(i)}$*
>   **end**
>   **end**
>   **end**
> **end**
>
> *% Generate Datastore for the test input $\boldsymbol{x}$*:
> $n_x \leftarrow$ length of $\boldsymbol{x}$
> **for** $k \leftarrow 1$ **to** N **do**
>   obtain hidden representation for the k-th token $\boldsymbol{h}_k$
> $\{\boldsymbol{z}, y\} \leftarrow$ top c items from $\mathcal{D}_{x_k}$ using $\boldsymbol{h}_k$ as the query
>   add $\{\boldsymbol{z}, y\}$ to $\mathcal{D}_{\text{target}}(x)$
> **end**

compress the high-dimensional representations of each token. Formally, given a vector $\boldsymbol{x} \in \mathbb{R}^D$, we represent it as the concatenation of M sub-vectors: $\boldsymbol{x} = [\boldsymbol{x}^1, \boldsymbol{x}^2, ..., \boldsymbol{x}^M]$, where all subvectors have the same dimension $d = D/M$. The product quantizer q consists of M sub-quantizers $q_1, ..., q_M$, each maps a subvecor $\boldsymbol{x}^m \in \mathbb{R}^d$ to a codeword $\boldsymbol{c}$ in a subspace codebook with n codewords: $\mathcal{C}^m = \{\boldsymbol{c}_1^m, \boldsymbol{c}_2^m, ..., \boldsymbol{c}_n^m\}$. The objective function of PQ is:

$$\min_{q^1, ..., q^M} \sum_{\boldsymbol{x}} \sum_{m=1}^{M} \|\boldsymbol{x}^m - q_m(\boldsymbol{x}^m)\|^2 \qquad (4)$$

Therefore each $\boldsymbol{x}$ is mapped to its nearest codeword in the Cartesian product space $\mathcal{C} = \mathcal{C}^1 \times ... \times \mathcal{C}^M$.

If each subspace codebook $\mathcal{C}^m$ has n codewords, then the Cartetian product space $\mathcal{C}$ could represent n^m D-dimensional codewords with only $n \times m$ d-dimensional vectors, thus significantly eased the memory issue. With the quantization technique, we are able to compress each token representation to 128-bytes. For example, for the WMT19 En-De dataset, the memory size is reduced from 3.5TB to 108GB.

4.5 kNN Retrieval Details

In practice, executing exact nearest neighbor search over millions or even billions of tokens could be time-consuming. Hence we use FAISS (Johnson et al., 2019) for fast approximate nearest neighbor search. All token representations are quantized to 128-bytes. Recall that we build a token-specific datastore $\mathcal{D}_v$ for each v in vocabulary. We do brute force search for tokens whose frequency n_v is lower than 30000. For those tokens whose frequency is larger than 30000, the keys are stored in clusters to speed up search. The number of clusters for token v is set to $\min(4 \times \sqrt{n_v}, n_v/30)$. To learn the cluster centroids, we use at most 5M keys for each token v. During inference, we query the datastore for $k = 512$ neighbors through searching 32 nearest clusters.

4.6 Discussions on Comparisons to Vanilla kNN-MT

The speedup of Fast kNN-MT lies in the following three aspects:
(1) For nearest neighbor retrieval on the source side, we first restrict the reference tokens that are the same as the query token. This strategy significantly narrows down the search space to roughly $|\mathcal{S}|/\text{mid}(F)$ times, where $|\mathcal{S}|$ denotes the number of tokens in the corpus, and $\text{mid}(F)$ denotes the medium word frequency in the corpus.
(2) The nearest neighbor search for all source tokens on the source side can be run in parallel, which is also a key speedup over kNN-MT. For vanilla kNN-MT, kNN search is performed on the target side and has to be auto-regressive: the representation for the current decoding step, which is used for the kNN search over the entire corpus, relies on previously generated tokens. Therefore, the kNN search for the current step has to wait for the finish of kNN searches for all previous generation steps.
(3) On the target side, the datastore in the kNN search is limited to target representations corresponding to selected reference source tokens.

Though the nearest neighbor search in the decoding process is auto-regressive and thus cannot be run in parallel, the running cost is fairly low: recall that the size of $\mathcal{D}_{\text{target}}$ is $c \times n$. Across all settings, the largest value of c is set to 512. The size of $\mathcal{D}_{\text{target}}$ is roughly 15k. Performing nearest neighbor searches among 15k candidates is relatively cheap for NMT, and is actually cheaper than the softmax operation for word prediction, where the vocabulary size is usually around 50k.

The combination of all these aspects leads to Fast kNN-MT two orders of magnitude faster than vanilla kNN-MT.

5 Experiments

5.1 Bilingual Machine Translation

We conduct experiments on two bilingual machine translation datasets: WMT'14 English-French and WMT'19 German-English. To create the datastore, we follow (Ng et al., 2019) to apply language identification filtering, keeping only sentence pairs with correct languages on both sides. We also remove sentences longer than 250 tokens and sentence pairs with a source/target ratio exceeding 1.5. For all datasets, we use the standard Transformer-base model provided by FairSeq (Ott et al., 2019) library.[3] The model has 6 encoder layers and 6 decoder layers. The dimensionality of word representations is 1024, the number of multi-attention heads is 16, and the inner dimensionality of feedforward layers is 8192. Particularly, following (Khandelwal et al., 2020), the model for WMT'19 German-English has also been trained on over 10 billion tokens of extra backtranslation data as well as fine-tuned on newstest test sets from previous years. We report the SacreBLEU scores (Post, 2018) for comparison.[4] Table 1 shows our results on the two NMT datasets. The proposed Fast kNN-MT model is able to achieve slightly better results to the vanilla kNN-MT model on WMT'19 German-English, and competitive results on WMT'14 English-French, with less kNN search cost.

5.2 Domain Adaptation

We also measure the effectiveness of the proposed Fast kNN-MT model on the domain adaptation task. We use the multi-domain datasets which are

Model	De-En	En-Fr
base MT	37.6	41.1
+kNN-MT	$39.1_{(+1.5)}$	$41.8_{(+0.7)}$
+fast kNN-MT	$39.3_{(+1.7)}$	$41.7_{(+0.6)}$

Table 1: SacreBLEU scores on WMT'19 De-En and WMT'14 En-Fr.

originally provided in (Koehn and Knowles, 2017) and further cleaned by (Aharoni and Goldberg, 2020). These datasets include German-English parallel data for train/validation/test sets in five domains: Medical, Law, IT, Koran and Subtitles. We use the trained German-English model introduced in Section 5.1 as our base model, and further build domain-specific datastores to evaluate the performance of Fast kNN-MT on each domain. Table 2 shows that Fast kNN-MT achieves comparable results to vanilla kNN-MT on domains of Medical (54.6 vs. 54.4), IT (45.9 vs. 45.8) and Subtitles (31.9 vs. 31.7), and outperforms vanilla kNN-MT on the domain of Koran (21.2 vs. 19.4). The average score of Fast kNN-MT (41.7) is on par with the result of (Aharoni and Goldberg, 2020) (41.3), which trains domain-specific models and reports in-domain results.

Following (Khandelwal et al., 2020), we also carry out experiments under the out-of-domain and multi-domain settings and report the results on Table 2. "+ WMT19' datastore" shows the results for retrieving neighbors from 770M tokens of WMT'19 data that the model has been trained on, and "+ all-domain datastore" shows the results where the model is trained on the multi-domain datastore from all six settings. The BLEU improvement is much smaller on the out-of-domain setup compared to the in-domain setup, illustrating that the proposed framework relies on in-domain data to retrieve valuable contexts. For the multi-domain setup, the performance for all six domains generally remains the same and only a small drop of the average score is witnessed. This shows that the Fast kNN-MT framework is robust to a massive amount of out-of-domain data and is able to retrieve the context-related information from in-domain data.

5.3 Analysis

Examples To visualize the effectiveness of the proposed Fast kNN-MT model, we randomly choose an example from the test set of the Law domain. Table 3 shows the test sentence, the retrieved

---

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

[4] https://github.com/mjpost/sacrebleu

Model	Medical	Law	IT	Koran	Subtitles	Avg.
(Aharoni and Goldberg, 2020)	54.8	58.8	43.5	21.8	27.4	41.3
base MT	39.9	45.7	38.0	16.3	29.2	33.8
+in-domain datastore:						
kNN-MT	$54.4_{(+14.5)}$	$61.8_{(+16.1)}$	$45.8_{(+7.8)}$	$19.4_{(+3.1)}$	$31.7_{(+2.5)}$	$42.6_{(+8.8)}$
fast kNN-MT	$53.6_{(+13.7)}$	$56.0_{(+10.3)}$	$45.5_{(+7.5)}$	$21.2_{(+4.9)}$	$30.5_{(+1.3)}$	$41.4_{(+7.6)}$
+WMT19' datastore						
kNN-MT	$40.2_{(+0.3)}$	$46.7_{(+1.0)}$	$40.3_{(+2.3)}$	$18.0_{(+1.7)}$	$29.2_{(+0.0)}$	$34.9_{(+1.1)}$
fast kNN-MT	$41.5_{(+1.6)}$	$45.9_{(+0.2)}$	$41.0_{(+3.0)}$	$17.6_{(+1.3)}$	$29.2_{(+0.0)}$	$35.0_{(+1.2)}$
+all-domains datastore						
kNN-MT	$54.5_{(+14.6)}$	$61.1_{(+15.4)}$	$48.6_{(+10.6)}$	$19.2_{(+2.9)}$	$31.7_{(+2.5)}$	$43.0_{(+9.2)}$
fast kNN-MT	$53.2_{(+13.3)}$	$54.6_{(+8.9)}$	$46.4_{(+8.4)}$	$19.5_{(+3.2)}$	$29.7_{(+0.5)}$	$40.7_{(+6.9)}$

Table 2: SacreBLEU results for domain adaptation.

	Source	Target
original sentence pair	Zwei Fisch@@ betriebe haben ihre Tätigkeit eingestellt .	Two fish establi@@ sh@@ ments have ce@@ ased their activities .
reference of "Zwei"	Zwei Gemein@@ schaft@@ sher@@ steller in Griechenland , die bei der vor@@ ausge@@ gangenen Untersuchung mit@@ gearbeitet hatten , gaben ihre Tätigkeit auf .	Fur@@ ther@@ more , two Community producers in Greece who took part in the previous investigation ce@@ ased their activity .
reference of "Fisch@@"	- fünf auf die Fisch@@ industrie ,	- five to representatives of the fi@@ sh@@ ery industries ,
reference of "betriebe"	(4) Drei Fleisch@@ betriebe mit Übergangs@@ regelung haben erhebliche Anstrengungen unter@@ nommen , neue Betriebs@@ anlagen zu bauen .	(4) Three meat establi@@ sh@@ ments on the list of establi@@ sh@@ ments in transition have made considerable efforts to build new facilities .
reference of "haben", "ihre", "Tätigkeit", "eingestellt" and "."	Einige Betriebe haben ihre Tätigkeit eingestellt .	Cer@@ tain establi@@ sh@@ ments have ce@@ ased their activities .

Table 3: A test sentence pair from the Law domain. We show the original sentence pair for test (the first row), the nearest-neighbor tokens on the source side along with the sentences that retrieved tokens reside in (the second column), and the aligned target tokens extracted from FastAlign, along with sentences in which target tokens reside in (the third column). The retrieved tokens are in red.

nearest neighbor tokens on the source side, and the corresponding target tokens. The first figure in Figure 2 demonstrates the similarity heatmap between the gold target tokens and the selected target neighbors. We can see that the retrieved target nearest tokens are highly correlated with the ground-truth target tokens, exhibiting the ability of Fast kNN-MT to accurately extract nearest reference tokens at each decoding step.

The Effect of the Number of neighbors per token on the source side We queried the datastore for nearest c neighbors for each source token. Intu-

itively, the larger the c is, the more likely the model could recall the nearest neighbors on the target side. The second figure in Figure 2 verifies this point: the model performance increases drastically when c increases from 8 to 64, and then continues increasing as c is up to 512.

The Effect of the Number of neighbors per token on the target side Fast kNN-MT selects top k nearest neighbors at each decoding step for computing the probability p_{kNN} in Eq.(2). The third figure in Figure 2 shows that the model performance first increases and then decreases when we continue
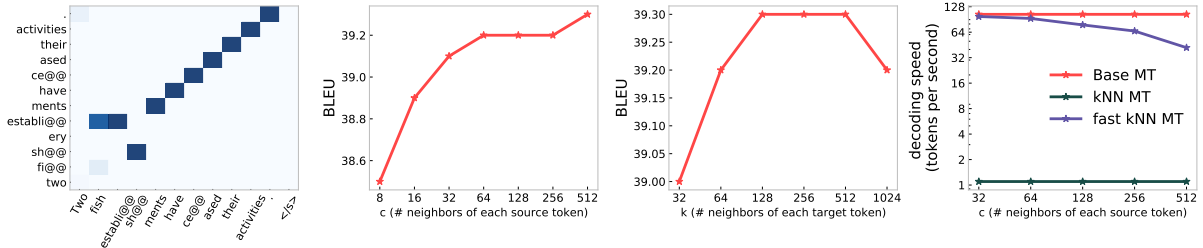
Figure 2: *First figure*: the similarity heatmap between the gold target tokens and the retrieved target neighbors. *Second figure*: BLEU scores with respect to different c, the number of nearest neighbors on the source side for each source token. *Third figure*: BLEU scores with respect to different k, the number of nearest neighbors on the target side for each target token. *Forth figure*: Speed comparison between base MT, kNN-MT and fast kNN-MT.

enlarging the value of k, with c fixed at 512, which is consistent with the observation in (Khandelwal et al., 2020). This is because that using neighbors that are too far away from the ground-truth target token adds noise to the model prediction, and thus hurts the performance.

Speed comparison When the beam size is fixed, the time complexity of Fast kNN-MT is mainly controlled by the number of retrieved neighbors c for each source token.[5] The last figure in Figure 2 shows the speed comparison between base MT, kNN-MT and fast kNN-MT when we vary the value of c. Fast kNN-MT decoded nearly as fast as the vanilla MT model when c is small. When c reaches 512, kNN-MT is about two times slower than the vanilla MT model. By contrast, vanilla kNN-MT is two order of magnitude slower than base MT and Fast kNN-MT regarding the decoding speed. This is because Fast kNN-MT substantially restricts the search space during decoding, whereas vanilla kNN-MT has to execute kNN search over the entire datastore at each decoding step.

Similarity function We have tried different similarity functions when retrieving c nearest neighbors on source side and computing the kNN distribution. These functions include cosine similarity, inner product and L_2 distance, the SacreBLEU scores for which are respectively 39.2, 39.1 and 38.8 on WMT'19 German-English, showing that cosine similarity is a better measure for representation distance than L_2 distance and inner product.

Effect of quantization Due to the memory issue, we applied quantization to compress the high-

Model	Medical	Law	IT	Koran	Subtitles	Avg.
fast kNN-MT	53.6	61.0	45.5	21.2	31.9	41.7
+full-precision	53.8	61.1	45.8	21.3	30.7	41.5

Table 4: SacreBLEU scores on domain adaptation when using full precision.

dimensional representation of each token in the training set. We investigate how quantization would affect model performances. As shown in Table 4, quantization has minor side effects in terms of BLEU scores, and when we use full precision instead of quantization, the average BLEU score only increases 0.1, which suggests that computing similarity using compressed vectors is a viable trade-off between memory usage and model performance.

6 Conclusion

In this work, we propose a fast version of kNN-MT – Fast kNN-MT – to address the runtime complexity issue of the vanilla kNN-MT. During decoding, Fast kNN-MT constructs a significantly smaller datastore for the nearest neighbor search: for each word in a source sentence, Fast kNN-MT selects its nearest tokens from a large-scale cache. The selected tokens are the same as the query token. Then at each decoding step, in contrast to using the entire datastore, the search space is limited to target tokens corresponding to the previously selected reference source tokens. Experiments demonstrate that this strategy drastically improves decoding efficiency while maintaining model performances compared to vanilla kNN-MT under different settings including bilingual machine translation and domain adaptation. Comprehensive ablation studies are performed to understand the behavior of each component in Fast kNN-MT. In future work, we plan to further improve the efficiency of Fast kNN-MT

---

[5] k plays a minor role to the overall time complexity because each search on the target side is performed within a total amount of cn tokens, which is negligible compared to the time cost spent on the source side.

by applying clustering techniques to build the data-store.

Acknowledgement

References

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

Roee Aharoni, Melvin Johnson, and Orhan Firat. 2019. Massively multilingual neural machine translation. *arXiv preprint arXiv:1903.00089*.

Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George Foster, Colin Cherry, et al. 2019. Massively multilingual neural machine translation in the wild: Findings and challenges. *arXiv preprint arXiv:1907.05019*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate.

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

Peter F Brown, Stephen A Della Pietra, Vincent J Della Pietra, and Robert L Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Bram Bulté and Arda Tezcan. 2019. Neural fuzzy repair: Integrating fuzzy matches into neural machine translation. In *57th Annual Meeting of the Association-for-Computational-Linguistics (ACL)*, pages 1800–1809.

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

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.

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

M Amin Farajian, Marco Turchi, Matteo Negri, and Marcello Federico. 2017. Multi-domain neural machine translation through unsupervised adaptation. In *Proceedings of the Second Conference on Machine Translation*, pages 127–137.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252. PMLR.

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.

XU Jitao, Josep M Crego, and Jean Senellart. 2020. Boosting neural machine translation with similar translations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1580–1590.

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

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A Smith. 2020. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation. *arXiv preprint arXiv:2006.10369*.

Jungo Kasai, Hao Peng, Yizhe Zhang, Dani Yogatama, Gabriel Ilharco, Nikolaos Pappas, Yi Mao, Weizhu Chen, and Noah A Smith. 2021. 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 and Rebecca Knowles. 2017. Six challenges for neural machine translation. *arXiv preprint arXiv:1706.03872*.

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

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

Patrick Lewis, Ethan Perez, Aleksandara 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*.

Xiaoqing Li, Jiajun Zhang, and Chengqing Zong. 2016. One sentence one model for neural machine translation. *arXiv preprint arXiv:1609.06490*.

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*.

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. 2020a. Understanding the difficulty of training transformers. *arXiv preprint arXiv:2004.08249*.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020b. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Yuxian Meng, Xiangyuan Ren, Zijun Sun, Xiaoya Li, Arianna Yuan, Fei Wu, and Jiwei Li. 2019. Large-scale pretraining for neural machine translation with tens of billions of sentence pairs. *arXiv preprint arXiv:1909.11861*.

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook fair's wmt19 news translation task submission. *arXiv preprint arXiv:1907.06616*.

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

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational linguistics*, 29(1):19–51.

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. *arXiv preprint arXiv:1904.01038*.

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, Brussels, Belgium. Association for Computational Linguistics.

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. 2020. Synthesizer: Rethinking self-attention in transformer models. *arXiv preprint arXiv:2005.00743*.

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. 2018. Learning to remember translation history with a continuous cache. *Transactions of the Association for Computational Linguistics*, 6:407–420.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio,

H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

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 Tieyan Liu. 2020b. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR.

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*.