Efficient Machine Translation Domain Adaptation

Anonymous ACL submission

Abstract

Machine translation models struggle when translating out-of-domain text, which makes domain adaptation a topic of critical importance. However, most domain adaptation methods focus on fine-tuning or training the entire or part of the model on every new domain, which can be costly. On the other hand, semi-parametric models have been shown to successfully perform domain adaptation by retrieving examples from an in-domain datastore (Khandelwal et al., 2021). A drawback of these retrieval-augmented models, however, is that they tend to be substantially slower. In this paper, we explore several approaches to speed up nearest neighbors machine translation. We adapt the methods recently proposed by He et al. (2021) for language modeling, and introduce a simple but effective caching strategy that avoids performing retrieval when similar contexts have been seen before. Translation quality and runtimes for several domains show the effectiveness of the proposed solutions.

1 Introduction

Modern neural machine translation models are mostly parametric (Bahdanau et al., 2015; Vaswani et al., 2017), meaning that, for each input, the output depends only on a fixed number of model parameters, obtained using some training data, hopefully in the same domain. However, when running machine translation systems in the wild, it is often the case that the model is given input sentences or documents from domains that were not part of the training data, which frequently leads to subpar translations. One solution is training or fine-tuning the entire model or just part of it for each domain, but this can be expensive and may lead to catastrophic forgetting (Saunders, 2021).

Recently, an approach that has achieved promising results is augmenting parametric models with a retrieval component, leading to *semi-parametric* models (Gu et al., 2018; Zhang et al., 2018; Bapna and Firat, 2019; Khandelwal et al., 2021; Meng et al., 2021; Zheng et al., 2021; Jiang et al., 2021). These models construct a datastore based on a set of source / target sentences or word-level contexts (translation memories) and retrieve similar examples from this datastore, using this information in the generation process. This allows having only one model that can be used for every domain. However, the model's runtime increases with the size of the domain's datastore and searching for related examples on large datastores can be computationally very expensive: for example, when retrieving 64 neighbors from the datastore, the model may become two orders of magnitude slower (Khandelwal et al., 2021). Due to this, some recent works have proposed methods that aim to make this process more efficient. Meng et al. (2021) proposed constructing a different datastore for each source sentence, by first searching for the neighbors of the source tokens; and He et al. (2021) proposed several techniques – datastore pruning, adaptive retrieval, dimension reduction – for nearest neighbor language modeling.

In this paper, we adapt several methods proposed by He et al. (2021) to machine translation, and we further propose a new approach that increases the model's efficiency: the use of a retrieval distributions cache. By caching the kNN probability distributions, together with the corresponding decoder representations, for the previous steps of the generation of the current translation(s), the model can quickly retrieve the retrieval distribution when the current representation is similar to a cached one, instead of having to search for neighbors in the datastore at every single step.

We perform a thorough analysis of the model's efficiency on a controlled setting, which shows that the combination of our proposed techniques results in a model, the efficient kNN-MT, which is approximately twice as fast as the vanilla kNN-MT. This comes without harming translation performance, which is, on average, more than 8 BLEU points and 5 COMET points better than the base MT model.

In sum, this paper presents the following contributions:[1]

- We adapt the methods proposed by He et al. (2021) for efficient nearest neighbors language modeling to machine translation.
- We propose a caching strategy to store the retrieval probability distributions, improving the translation speed.
- We compare the efficiency and translation quality of the different methods, which show the benefits of the proposed and adapted techniques.

2 Background

When performing machine translation, the model is given a source sentence or document, $\boldsymbol{x} = [x_1, \ldots, x_L]$, on one language, and the goal is to output a translation of the sentence in the desired language, $\boldsymbol{y} = [y_1, \ldots, y_N]$. This is usually done using a parametric sequence-to-sequence model (Bahdanau et al., 2015; Vaswani et al., 2017), in which the encoder receives the source sentence as input and outputs a set of hidden states. Then, at each step t, the decoder attends to these hidden states and outputs a probability distribution $p_{\mathrm{NMT}}(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x})$ over the vocabulary. Finally, these probability distributions are used to predict the output tokens, typically with beam search.

2.1 Nearest Neighbor Machine Translation

Khandelwal et al. (2021) introduced a nearest neighbor machine translation model, kNN-MT, which is a semi-parametric model. This means that besides having a parametric component that outputs a probability distribution over the vocabulary, $p_{\mathrm{NMT}}(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x})$, the model also has a nearest neighbor retrieval mechanism, which allows direct access to a datastore of examples.

More specifically, we build a datastore $\mathcal{D}$ which consists of a key-value memory, where each entry key is the decoder's output representation, $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}_{<t})$, and the value is the target token y_t:

$$\mathcal{D} = \{(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}_{<t}), y_t) \ \forall y_t \in \boldsymbol{y} \mid (\boldsymbol{x}, \boldsymbol{y}) \in (\mathcal{X}, \mathcal{Y})\}, \tag{1}$$

where $(\mathcal{X}, \mathcal{Y})$ corresponds to a set of parallel source and target sequences. Then, at inference time, the model searches the datastore to retrieve the set of k nearest neighbors $\mathcal{N}$. Using their distances $d(\cdot)$ to the current decoder's output representation, we can compute the retrieval distribution $p_{k\mathrm{NN}}(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x})$ as:

$$p_{k\mathrm{NN}}(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x}) = \frac{\sum_{(\boldsymbol{k}_j, v_j) \in \mathcal{N}} \mathbb{1}_{y_t = v_j} \exp\left(-d\left(\boldsymbol{k}_j, \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}_{<t})\right)/T\right)}{\sum_{(\boldsymbol{k}_j, v_j) \in \mathcal{N}} \exp\left(-d\left(\boldsymbol{k}_j, \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}_{<t})\right)/T\right)}, \tag{2}$$

where T is the softmax temperature, k_j denotes the key of the j^{th} neighbor and v_j its value. Finally, $p_{\mathrm{NMT}}(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x})$ and $p_{k\mathrm{NN}}(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x})$ are combined to obtain the final distribution, which is used to generate the translation through beam search, by performing interpolation:

$$\begin{aligned} p(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x}) = &(1-\lambda)\, p_{\mathrm{NMT}}(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x}) \\ &+ \lambda\, p_{k\mathrm{NN}}(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x}), \end{aligned} \tag{3}$$

where λ is a hyper-parameter that controls the weights given to the two distributions.

3 Efficient kNN-MT

In this section, we describe the approaches introduced by He et al. (2021) to speed-up the inference time for nearest neighbors language modeling, such as pruning the datastore (§3.1) and reducing the representations dimension (§3.2), which we adapt to machine translation. We further describe a novel method that allows the model to have access to examples without having to search them in the datastore at every step, by maintaining a cache of the past retrieval distributions, for the current translation(s) (§3.3).

3.1 Datastore Pruning

The goal of datastore pruning is to reduce the size of the datastore, so that the model is able to search for the nearest neighbors faster. To do so, we follow He et al. (2021), and use greedy merging. In greedy merging, we aim to merge datastore entries that share the same value (target token) while their keys are close to each other in vector space. To do this, we first need to find the k nearest neighbors of every entry of the datastore, where k is a hyper-parameter. Then, if in the set of neighbors, retrieved for a given entry, there is an entry which has not been merged before and has the same value, we merge the two entries, by simply removing the neighboring one.

[1] We will release all code upon acceptance.

3.2 Dimension Reduction

The decoder's output representations, $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}_{<t})$ are, usually, high-dimensional (1024, in our case). This leads to a high computational cost when computing vector distances, which are needed for retrieving neighbors from the datastore. To alleviate this, we follow He et al. (2021), and use principal component analysis (PCA), an efficient dimension reduction method, to reduce the dimension of the decoder's output representation to a pre-defined dimension, d, and generate a compressed datastore.

3.3 Cache

The model does not need to search the datastore at every step of the translation generation in order to do it correctly. Here, we aim to predict when it needs to retrieve neighbors from the datastore, so that, by only searching the datastore in the necessary steps, we can increase the generation speed.

Adaptive retrieval. To do so, first we follow He et al. (2021), and use a simple MLP to predict the value of the interpolation coefficient λ at each step. Then, we define a threshold, α, so that the model only performs retrieval when $\lambda > \alpha$. However, we observed that this leads to results (§A.3) similar to randomly selecting when to search the datastore. We posit this occurs because it is difficult to predict when the model should perform retrieval, for domain adaptation (He et al., 2021), and because in machine translation error propagation occurs more prominently than in language modeling.

Cache. Because it is common to have similar contexts along the generation process, when using beam search, the model can be often retrieving similar neighbors at different steps, which is not efficient. To avoid repeating searches on the datastore for similar context vectors, $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}_{<t})$, we propose keeping a cache of the previous retrieval distributions, of the current translation(s). More specifically, at each step of the generation of $\boldsymbol{y}$, we add the decoder's representation vector along with the retrieval distribution $p_{k\mathrm{NN}}(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x})$, corresponding to all beams, $\mathcal{B}$, to the cache $\mathcal{C}$:

$$\mathcal{C}=\{(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}_{<t}), p_{k\mathrm{NN}}(y_t|\boldsymbol{y}_{<t},\boldsymbol{x}))\forall y_t \in \boldsymbol{y} \,|\, \boldsymbol{y} \in \mathcal{B}\}. \quad (4)$$

Then, at each step of the generation, we compute the Euclidean distance between the current decoder's representation and the keys on the cache. If all distances are bigger than a threshold τ, the model searches the datastore to find the nearest neighbors. Otherwise, the model retrieves, from the cache, the retrieval distribution that corresponds to the closest key.

4 Experiments

Dataset and metrics. We perform experiments on the Medical, Law, IT, and Koran domain data of the multi-domains dataset (Koehn and Knowles, 2017) re-splitted by Aharoni and Goldberg (2020). To build the datastores we use the in-domain training sets which have from 17,982 to 467,309 sentences. The validation and test sets have 2,000 sentences. To evaluate the models we use BLEU (Papineni et al., 2002; Post, 2018) and COMET (Rei et al., 2020).

Settings. We use the WMT'19 German-English news translation task winner (Ng et al., 2019) (with 269 M parameters), available on the Fairseq library (Ott et al., 2019), as the base MT model. As baselines, we consider the base MT model, the vanilla kNN-MT model (Khandelwal et al., 2021), and the Fast kNN-MT model (Meng et al., 2021). For all models, which perform retrieval, we select the hyper-parameters by performing grid search on $k \in \{8, 16, 32, 64\}$ and $\lambda \in \{0.5, 0.6, 0.7, 0.8\}$. For the vanilla kNN-MT model and the efficient kNN-MT we follow Khandelwal et al. (2021) and use the Euclidean distance to perform retrieval and the proposed softmax temperature. For the Fast kNN-MT, we use the cosine distance and the softmax temperature proposed by Meng et al. (2021). For the efficient kNN-MT we select $k = 2$ for datastore pruning, $d = 256$ for PCA, and $\tau = 6$ as the cache threshold. Additional results and hyper-parameters are reported in Apps. A and B.

4.1 Results

The translation scores are reported on Table 1. We can clearly see that both Fast kNN-MT and the efficient kNN-MT (combining the different methods) do not hurt the translation performance substantially, still having, on average, 8 BLEU points and 5 COMET points more than the base MT model.

4.2 Generation speed

Computational infrastructure. All experiments were performed on a server with 3 RTX 2080 Ti (11 GB), 12 AMD Ryzen 2920X CPUs (24 cores), and 128 Gb of RAM. For the speed measurements, we ran each model on a single GPU while no other process was running on the server, to have a controlled

	BLEU					COMET				
	Medical	Law	IT	Koran	Average	Medical	Law	IT	Koran	Average
Base MT	40.01	45.64	37.91	16.35	34.98	.4702	.5770	.3942	-.0097	.3579
kNN-MT	54.47	61.23	45.96	21.02	45.67	.5760	.6781	.5163	.0480	.4546
Fast kNN-MT	52.90	55.71	44.73	21.29	43.66	.5293	.5944	.5445	-.0455	.4057
cache	53.30	59.12	45.39	20.67	44.62	.5625	.6403	.5085	.0346	.4365
PCA + cache	53.58	58.57	46.29	20.67	44.78	.5457	.6379	.5311	-.0021	.4282
PCA + pruning	53.23	60.38	45.16	20.52	44.82	.5658	.6639	.4981	.0298	.4394
PCA + cache + pruning	51.90	57.82	44.44	20.11	43.57	.5513	.6260	.4909	-.0052	.4158

Table 1: BLEU and COMET scores on the multi-domains test set, for a batch size of 8.

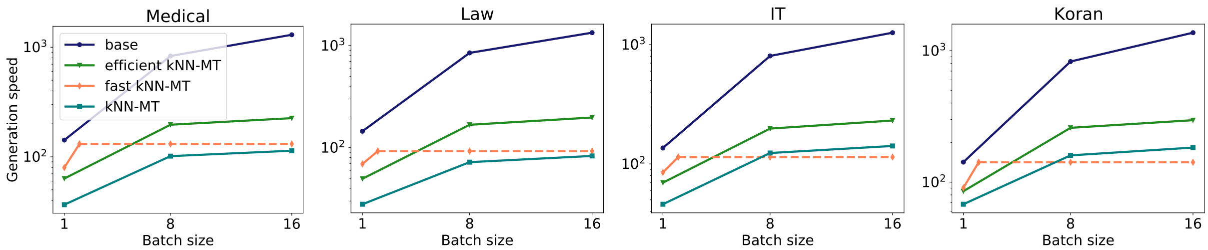

Figure 1: Plots of the generation speed (tokens/s) for the different models on the medical, law, IT, and Koran domains, for different batch sizes (1,8,16). The generation speed (y-axis) is in log scale. When using the Fast kNN-MT model, the maximum batch size that we are able to use is 2, due to out of memory errors.

environment. To search the datastore, we used the FAISS library (Johnson et al., 2019). When using the vanilla kNN-MT and efficient kNN-MT, the nearest neighbor search is performed on the CPUs, since not all datastores fit into memory, while when using the Fast kNN-MT this is done on the GPU.

Analysis. As can be seen on the plots of Figure 1, for a batch size of 1 Fast kNN-MT leads to a generation speed higher than our proposed method and vanilla kNN-MT. However, because of its high memory requirements, we are not able to run Fast kNN-MT for batch sizes larger than 2, on the computational infrastructure stated above. On the contrary, when using the proposed methods (efficient kNN-MT) we are able to run the model with higher batch sizes, achieving superior generation speeds to Fast kNN-MT and vanilla kNN-MT, and reducing the difference to the base MT model.

Ablation. We plot the generation speed for different combinations of the proposed methods, for several batch sizes, on Figure 2. On this plot, we can clearly see that every method contributes to the speed-up achieved by the model that combines all approaches. Moreover, we can observe that the method which leads to the largest speed-up is the use of a cache of retrieval distributions, by saving, on average 57% of the retrieval searches.

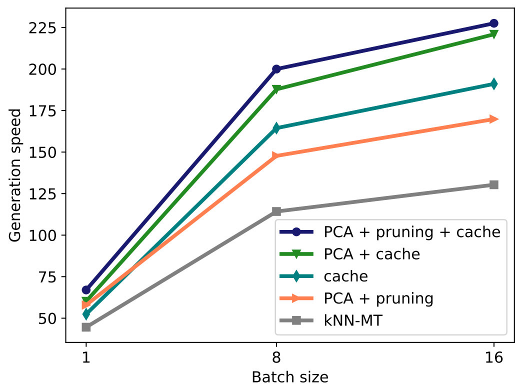

Figure 2: Plot of the generation speed (tokens/s) for combinations of the proposed methods.

5 Conclusion

In this paper we propose the efficient kNN-MT, in which we combine several methods to improve the kNN-MT generation speed. First, we adapted to machine translation methods that improve retrieval efficiency in language modeling (He et al., 2021). Then we proposed a new method which consists on keeping in cache the previous retrieval distributions so that the model does not need to search for neighbors in the datastore at every step. Through experiments on domain adaptation, we show that the combination of the proposed methods leads to a considerable speed-up (up to 2x) without harming the translation performance substantially.

References

Roee Aharoni and Yoav Goldberg. 2020. Unsupervised domain clusters in pretrained language models. In *Proc. ACL*.

Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. ICLR*.

Ankur Bapna and Orhan Firat. 2019. Non-Parametric Adaptation for Neural Machine Translation. In *Proc. NAACL*.

Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor OK Li. 2018. Search engine guided neural machine translation. In *Proc. AAAI*.

Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. 2021. Efficient Nearest Neighbor Language Models. In *Proc. EMNLP*.

Qingnan Jiang, Mingxuan Wang, Jun Cao, Shanbo Cheng, Shujian Huang, and Lei Li. 2021. Learning Kernel-Smoothed Machine Translation with Retrieved Examples. In *Proc. EMNLP*.

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

Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2021. Nearest neighbor machine translation. In *Proc. ICLR*.

Philipp Koehn and Rebecca Knowles. 2017. Six Challenges for Neural Machine Translation. In *Proceedings of the First Workshop on Neural Machine Translation*.

Yuxian Meng, Xiaoya Li, Xiayu Zheng, Fei Wu, Xiaofei Sun, Tianwei Zhang, and Jiwei Li. 2021. Fast Nearest Neighbor Machine Translation.

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook FAIR's WMT19 News Translation Task Submission. In *Proc. of the Fourth Conference on Machine Translation*.

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. In *Proc. NAACL (Demonstrations)*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proc. ACL*.

Matt Post. 2018. A Call for Clarity in Reporting BLEU Scores. In *Proc. Third Conference on Machine Translation*.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. COMET: A Neural Framework for MT Evaluation. In *Proc. EMNLP*.

Danielle Saunders. 2021. Domain Adaptation and Multi-Domain Adaptation for Neural Machine Translation: A Survey.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. NeurIPS*.

Jingyi Zhang, Masao Utiyama, Eiichiro Sumita, Graham Neubig, and Satoshi Nakamura. 2018. Guiding Neural Machine Translation with Retrieved Translation Pieces. In *Proc. NAACL*.

Xin Zheng, Zhirui Zhang, Junliang Guo, Shujian Huang, Boxing Chen, Weihua Luo, and Jiajun Chen. 2021. Adaptive Nearest Neighbor Machine Translation.

A Additional results

In this section we report the BLEU scores as well as additional statistics for the different methods, when varying their hyper-parameters.

A.1 Datastore pruning

We report on Table 2 the BLEU scores for datastore pruning, when varying the number of neighbors used for greedy merging, k. The resulting datastore sizes are presented on Table 3.

	Medical	Law	IT	Koran	Average
kNN-MT	54.47	61.23	45.96	21.02	45.67
$k = 1$	53.60	60.23	45.03	20.81	44.92
$k = 2$	52.95	59.40	44.76	20.12	44.31
$k = 5$	51.63	57.55	44.07	19.29	43.14

Table 2: BLEU scores on the multi-domains test set when performing datastore pruning with several values of k, for a batch size of 8.

	Medical	Law	IT	Koran
kNN-MT	6,903,141	19,062,738	3,613,334	524,374
$k = 1$	4,780,514	13,130,326	2,641,709	400,385
$k = 2$	4,039,432	11,103,775	2,303,808	353,007
$k = 5$	3,084,106	8,486,551	1,852,191	290,192

Table 3: Sizes of the in-domain datastores when performing datastore pruning with several values of k, for a batch size of 8.

A.2 Dimension reduction

We report on Table 4 the BLEU scores for dimension reduction, when varying the output dimension d.

	Medical	Law	IT	Koran	Average
kNN-MT	54.47	61.23	45.96	21.02	45.67
$d = 512$	55.06	62.04	46.98	21.24	46.33
$d = 256$	54.52	61.84	46.68	21.57	46.15
$d = 128$	53.94	61.17	45.46	21.35	45.48

Table 4: BLEU scores on the multi-domains test set when performing PCA with different dimension, d, values, for a batch size of 8.

A.3 Adaptive retrieval

We report on Table 5 the BLEU scores for adaptive retrieval, when varying the threshold α. The percentage of times the model performs retrieval is stated on Table 6.

	Medical	Law	IT	Koran	Average
kNN-MT	54.47	61.23	45.96	21.02	45.67
$\alpha = 0.25$	45.52	49.91	37.97	16.36	37.44
$\alpha = 0.5$	52.84	59.36	38.58	18.08	42.22
$\alpha = 0.75$	53.90	60.87	43.05	19.91	44.43

Table 5: BLEU scores on the multi-domains test set when performing adaptive retrieval for different values of the threshold α, for a batch size of 8.

	Medical	Law	IT	Koran
kNN-MT	100%	100%	100%	100%
$\alpha = 0.25$	78%	73%	38%	4%
$\alpha = 0.5$	96%	96%	60%	61%
$\alpha = 0.75$	98%	99%	92%	91%

Table 6: Percentage of times the model searches for neighbors on the datastore when performing adaptive retrieval for different values of the threshold α, for a batch size of 8.

A.4 Cache

We report on Table 7 the BLEU scores for a model using a cache of the retrieval distributions, when varying the threshold τ. The percentage of times the model performs retrieval is stated on Table 8.

	Medical	Law	IT	Koran	Average
kNN-MT	54.47	61.23	45.96	21.02	45.67
$\tau = 2$	54.47	61.23	45.93	20.98	45.65
$\tau = 4$	54.17	61.10	46.07	21.00	45.58
$\tau = 6$	53.30	59.12	45.39	20.67	44.62
$\tau = 8$	30.06	23.01	25.53	16.08	23.67

Table 7: BLEU scores on the multi-domains test set when using a retrieval distributions' cache for different values of the threshold τ, for a batch size of 8.

	Medical	Law	IT	Koran
kNN-MT	100%	100%	100%	100%
$\tau = 2$	59%	51%	67%	64%
$\tau = 4$	50%	42%	57%	53%
$\tau = 6$	43%	35%	49%	45%
$\tau = 8$	26%	16%	29%	31%

Table 8: Percentage of times the model searches for neighbors on the datastore when using a retrieval distributions' cache for different values of the threshold τ, for a batch size of 8.

B Hyper-parameters

On Table 9 we report the values for the hyper-parameters: number of neighbors to be retrieved

	Medical			Law			IT			Koran		
	k	λ	T	k	λ	T	k	λ	T	k	λ	T
kNN-MT	8	0.7	10	8	0.8	10	8	0.7	10	8	0.6	100
Fast kNN-MT	16	0.7	.015	32	0.6	.015	8	0.6	.02	16	0.6	.05
cache	8	0.7	10	8	0.8	10	8	0.7	10	8	0.6	100
PCA + cache	8	0.8	10	8	0.8	10	8	0.7	10	8	0.7	100
PCA + pruning	8	0.7	10	8	0.8	10	8	0.7	10	8	0.7	100
PCA + cache + pruning	8	0.7	10	8	0.8	10	8	0.7	10	8	0.7	100

Table 9: Values of the hyper-parameters: number of neighbors to be retrieved k, interpolation coefficient λ, and retrieval softmax temperature T.

$k \in \{8, 16, 32, 64\}$, the interpolation coefficient $\lambda \in \{0.5, 0.6, 0.7, 0.8\}$, and retrieval softmax temperature T. For decoding we use beam search with a beam size of 5.