Revisiting Locality Sensitive Hashing for Vocabulary Selection in Fast Neural Machine Translation

Hieu Hoang* **Marcin Junczys-Dowmunt***
Roman Grundkiewicz **Huda Khayrallah**

Microsoft, 1 Microsoft Way, Redmond, WA 98052, USA
{hihoan,marcinjd,rogrundk,hkhayrallah}@microsoft.com

Abstract

Neural machine translation models often contain large target vocabularies. The calculation of logits, softmax and beam search is computationally costly over so many classes. We investigate the use of locality sensitive hashing (LSH) to reduce the number of vocabulary items that must be evaluated and explore the relationship between the hashing algorithm, translation speed and quality. Compared to prior work, our LSH-based solution does not require additional augmentation via word-frequency lists or alignments. We propose a training procedure that produces models, which, when combined with our LSH inference algorithm increase translation speed by up to 87% over the baseline, while maintaining translation quality as measured by BLEU. Apart from just using BLEU, we focus on minimizing search errors compared to the full softmax, a much harsher quality criterion.

1 Introduction

The computation of the output logit, softmax and beam search (the *output layer*) are some of the most compute-intensive tasks in current Neural Machine Translation (NMT) models, often taking the majority of inference time for many models, on many hardware architectures, especially in deployment settings. This is mainly due to the large vocabulary size relative to other dimensions in the model. Methods that reduce the effective vocabulary size can have a major impact on inference speed. Vocabulary selection is one such method.

However, a known downside of vocabulary selection methods is the risk of search errors if the desired output token $\hat{y}_t$ is not a member of the reduced vocabulary $\overline{V}$, forcing the beam search to choose a sub-optimal token. Even when the impact of such search errors on BLEU is minimal, search errors caused by lexical shortlisting degrade human judgements of quality (Domhan et al., 2022).

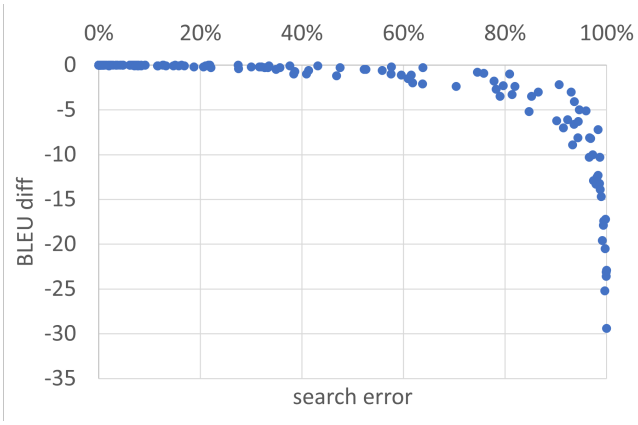


Figure 1: Search errors vs. decrease in BLEU for all experiments in the paper. We vary the two main hyperparameters of our LSH implementation: number of hash functions and the size k of our selected vocabulary subset. 60% of sentences have a search error before we observe a 1 BLEU point degradation.

The degradation in human judgement is less surprising if we inspect Figure 1 which shows that translation quality, as measured by BLEU, is resilient to search errors caused by LSH. Only when over 60% of translations exhibit search errors is there significant BLEU degradation.

We examine vocabulary selection using Locality Sensitive Hashing (LSH), and evaluate specifically in the context of Neural Machine Translation. We introduce an LSH-based vocabulary selection algorithm and compatible models such that:[1]

1. the models have translation quality that is better than or comparable to the baseline model;
2. the LSH-based vocabulary selection algorithm introduces minimal search errors across a number of models and language pairs, including no search errors at all for certain configurations;
3. inference is up to 87% faster than the baseline.

*These authors contributed equally to this work.

[1]We release code in Marian; see Section 7.1 for details.

Proceedings of the Seventh Conference on Machine Translation (WMT), pages 855 - 869

2 Methods of vocabulary selection

The output layer for a target vocabulary V, performs the following computations:

$$p(y_t|y_{1:t-1}, x; \theta) = \text{softmax}(Wh + b)$$
$$\hat{y}_t = \text{argmax}\, p(y_t|y_{1:t-1}, x; \theta), \quad (1)$$

where $W \in \mathbb{R}^{|V| \times d}$ is the weight matrix, $b \in \mathbb{R}^{|V|}$ is the bias vector, $h \in \mathbb{R}^d$, d is the hidden dimension size of the decoder state, and $p(y_t|y_{1:t-1}, x; \theta) \in \mathbb{R}^{|V|}$ is the softmax probabilities. This is computationally expensive due to the target vocabulary size, $|V|$.

Vocabulary selection create a small subset, $\overline{V} \subset V$. This will reduce the size of weight matrix, $\overline{W}$, and bias vector, $\overline{b}$, where $|\overline{V}| \ll |V|$, $\overline{W} \in \mathbb{R}^{|\overline{V}| \times d}$ and $\overline{b} \in \mathbb{R}^{|\overline{V}|}$.

Equation 1 is then replaced with the more efficient Equation 2 which uses $\overline{W}$ and $\overline{b}$ instead.

$$\overline{p}(y_t|y_{1:t-1}, x; \theta) = \text{softmax}(\overline{W}h + \overline{b})$$
$$\overline{\hat{y}_t} = \text{argmax}\, \overline{p}(y_t|y_{1:t-1}, x; \theta), \quad (2)$$

The aim is now to find the subset, $\overline{V}$, such that $\overline{\hat{y}_t} = \hat{y}_t$.

Depending on the method, vocabulary selection (and therefore construction of $\overline{V}$, $\overline{W}$ and $\overline{b}$) can be static or happen dynamically per sentence (or batch), per decoder time step, or even per individual decoder hypothesis.

We restrict our overview of the concept of vocabulary selection to the case where the original softmax layer remains largely unmodified except for sub-selection. Methods that require complex structural reformulations of the softmax layer during training like hierarchical softmax (Morin and Bengio, 2005), adaptive softmax (Grave et al., 2016) or binary code prediction (Oda et al., 2017) are outside the scope of this work.

In-depth overviews of past and current vocabulary selection methods are provided by L'Hostis et al. (2016), Shi and Knight (2017), and more recently Domhan et al. (2022). We only repeat concepts that are either common or required to differentiate our work from previous approaches.

2.1 Word frequency-based methods

For simplicity's sake, when describing word frequency-based methods, we assume that vocabulary identifiers correspond to frequency rank (according to a training corpus or other reference corpus) and hence the top-K first items in a vocabulary list are the top-K most frequent words/segments from the training corpus. The choice of K determines a static subset V_f of V where $|V_f| = K$. Then $\overline{V} = V_f$ and the parameters $\overline{W}, \overline{b}$ of the softmax output layer are sub-selected accordingly.

Word frequency-based vocabulary selection is not a viable method on its own — the quality degradation is simply too large to be acceptable (Shi and Knight, 2017) — but it constitutes an important common back-bone for several of the more accurate methods discussed below as it is an easy way to include common segments like function words, punctuation, etc. in the output vocabulary.

2.2 Word alignment-based methods

Word alignment-based vocabulary selection (Jean et al., 2015a) has been part of the NMT toolbox since the earliest competitive NMT systems. Jean et al. (2015b) first introduce the concept in essentially the form it is widely being used today[2] in their submission to the WMT15 shared task (Bojar et al., 2015). Later work (Mi et al., 2016; L'Hostis et al., 2016; Shi and Knight, 2017) rediscover mostly the same setup or confirm it to be one of the strongest methods amongst a number of other approaches.

Given a source sentence $x_{1:m}$ and a word-alignment dictionary with alignment probabilities between source and target segments $p_a(y|x)$, this method creates $V_a = \bigcup_{t \in 1:m} V_a(x_t)$, where for instance $V_a(x_t) = \{y \in V : p_a(y|x_t) \geq \overline{p}\}$ for a given threshold $\overline{p}$. Other criteria for constructing $V_a(x_t)$ are possible: such as K' most probable aligned target words or combinations of multiple criteria.

Finally, the alignment-based method is typically combined with the frequency-based method as $\overline{V} = V_f \cup V_a$. Word alignment thus extends and refines the target word-frequency method by mapping source sentence context to plausible target language vocabulary candidates (ranked or selected by translation probability). Note, that V_a is constructed dynamically once per source sentence or batch which forces a dynamic construction of $\overline{V}$.

2.3 Earlier LSH-based approaches

Locality Sensitive Hashing (LSH) as a way to accelerate the computation of inner products has been

[2] See submissions to the recent shared tasks on efficient NMT (Hayashi et al., 2019; Heafield et al., 2020, 2021).

investigated as early as 2014 (Vijayanarasimhan et al.) for handling large vocabularies and remains an active area of research for more general neural network training (see e.g. Chen et al., 2020).

Previous work on using LSH in NMT (Shi and Knight, 2017; Shi et al., 2018) takes an approach that is analogous to the alignment-based methods in the sense that a static vocabulary based on word frequency is extended with target vocabulary items that are plausible in the dynamic context of the decoded sentence. However, instead of mapping source segments to target segments via alignment dictionary look-up, the decoder hypothesis state vector h is used to find set $V_l(h)$ of the k target vocabulary items y with the corresponding output layer embedding vector w_y most *similar* to h. As before, the static word-frequency based vocabulary set is merged with the contextual set to form $\overline{V}(h) = V_f \cup V_l(h)$. Note however, that $\overline{V}$ now depends dynamically on each decoder state h.

The specifics of how similarity between the vectors is defined determine the speed and accuracy of the method. The output layer itself can be seen as a similarity function (inner product with softmax normalization) that has perfect accuracy but is least interesting in terms of speed.

Shi and Knight (2017) and Shi et al. (2018) use Winner-Take-All (WTA; Yagnik et al., 2011) hashing with banding to approximate the output layer. However, the type of similarity as expressed via WTA hashing seems to result in fairly low accuracy and therefore needs to be merged with several thousand most frequent vocabulary items to remain competitive in terms of translation quality compared to the full vocabulary.

2.4 Selection as binary classification

L'Hostis et al. (2016) and more recently Domhan et al. (2022) propose to approach the vocabulary selection problem as a per target vocabulary item binary classification problem where each of $|V|$ binary classifiers decides if the corresponding target vocabulary item should be included in the sentence-level (or batch-level) target vocabulary.

L'Hostis et al. (2016) train a suite of $|V|$ binary SVM classifiers which are learned independently from the neural model. The set of words in the source sentence serves as a sparse bag-of-words feature set.

Domhan et al. (2022) train their "neural vocabulary selection" model jointly with the translation model via a multi-objective cost function. They construct $z = \sigma(\text{maxpool}(WH + b))$ where $H \in \mathbb{R}^{d \times m}$ is the hidden encoder context, $W \in \mathbb{R}^{|V| \times d}$, $b \in \mathbb{R}^{|V|}$ and $z \in \mathbb{R}^{|V|}$.

Generally, for both methods, given the binary classifier z_y corresponding to the vocabulary entry y, we have $\overline{V} = \{y \in V : z_y(x_{1:m}) \geq \lambda\}$ where λ is the decision threshold for including y in $\overline{V}$. $\overline{V}$ is constructed dynamically once per sentence and both methods do not need to be merged with the word-frequency-based vocabulary list V_f. The threshold λ seems to be sufficient to control for speed versus accuracy trade-offs.

3 Our LSH-based method

Our work contrasts with prior research on LSH for NMT by Shi and Knight (2017); Shi et al. (2018) in that:

1. We use SimHash hash instead of WTA hash.
2. We do not need to expand the LSH vocabulary subset $\overline{V}$ by merging with a static list of the most frequent words.
3. We do not need to merge $\overline{V}$ across batch and beam entries.
4. We create $\overline{V}$ by finding the top-k smallest Hamming distances, rather than banding hashes and Cuckoo lookups.
5. Our target vocabulary is smaller than most experiments in the above works which experimented with target vocabulary sizes of 66k, 50k, 40k and 25k. We believe larger vocabularies are unnecessary as a result of the use of sub-word units (Sennrich et al., 2016) and their variants. We use sub-word units while the above works do not.
6. We are concerned with search errors introduced by vocabulary selection as well as with translation quality degradation. Quality metrics are often insensitive to errors caused by deviation from an otherwise unfiltered vocabulary.

3.1 SimHash for Softmax approximation

Prior research on the application of LSH for NMT by Shi and Knight (2017); Shi et al. (2018) relies on WTA hashing. We found SimHash (Charikar, 2002) to result in much lower search error.

For a random normal vector $r \in \mathbb{R}^d$ and an input vector v (of the same size as r), SimHash introduces the following hash function H_r:

$$\mathrm{H}_r(v) = \begin{cases} 1 & \text{if } v \cdot r \geq 0 \\ 0 & \text{if } v \cdot r < 0 \end{cases}$$

which maps v to a single bit. The above is generalized to c bits by generating and applying c different random vectors and concatenating the results. This can be simplified via multiplying with a projection matrix $R \in \mathbb{R}^{d \times c}$ and the same dimension-wise mapping to bits of the result.[3] We call this function $\mathrm{H}_R(v) : \mathbb{R}^d \to \{0,1\}^c$ and use it to obtain the LSH representation of v. Further, $\mathrm{D}(\mathrm{H}_R(u), \mathrm{H}_R(v))$ denotes the bit-wise Hamming distance between the hashed binary representations of vectors u, v.

SimHash has been designed in such a way that for two vectors u, v for which the angle $\theta(u, v)$ between these vectors is small, the Hamming distance D over their hashed binary vectors should be small as well.[4] Naturally, the cosine similarity $\cos(\theta(u, v))$ will be high for such cases.

It is this property which allows us to apply a series of transformations and approximations to find a promising candidate for the most probable vocabulary item $\hat{i}$ for a decoder state vector h (and the output layer parameters W and b) using fast Hamming distance computation:

$$\begin{aligned}
\hat{i} &= \operatorname*{argmax}_{i \in V} \operatorname*{softmax}_i(Wh + b) && (3)\\
&= \operatorname*{argmax}_{i \in V} w_i \cdot h + b_i && (4)\\
&\approx \operatorname*{argmax}_{i \in V} w_i \cdot h && (5)\\
&\approx \operatorname*{argmax}_{i \in V} \cos(\theta(w_i, h)) && (6)\\
&\approx \operatorname*{argmax}_{i \in V} \cos\left(\mathrm{D}(\mathrm{H}_R(w_i), \mathrm{H}_R(h))\frac{\pi}{c}\right) && (7)\\
&= \operatorname*{argmin}_{i \in V} \mathrm{D}(\mathrm{H}_R(w_i), \mathrm{H}_R(h)). && (8)
\end{aligned}$$

In every step above which leads with $\approx$, we introduce a new approximation to the previous step, potentially reducing the accuracy of the search for $\hat{i}$. When moving from Equation 4 to Equation 5, we drop the bias term b_i as it cannot be easily incorporated in the search in Hamming space. For models with large values in the bias vector b, this will inadvertently lead to search errors. The easy solution to this problem is to drop the bias term during training as well. More on this in Section 5.1.

In Equation 6 we ignore the magnitude of the vectors. This seems to not matter much for the search and we leave investigating the effects or potential mitigation for future work.[5]

Equation 7 sees the introduction of the SimHash LSH as we approximate the angle θ via the Hamming distance. Finally, in Equation 8 we can find the most promising vocabulary candidate by directly minimizing the Hamming distance; note that we flipped from argmax to argmin.

3.2 Integrating LSH with beam search

In Section 2, we categorized methods of vocabulary selection by how and when they construct the set of subselected vocabulary $\overline{V}$.

Before translation begins, the output embedding weights W are hashed once using the SimHash function $\mathrm{H}_R(W)$ to create a set $L \in \{0,1\}^{|V| \times c}$ of LSH keys, one for each target vocabulary entry from V:

$$L = \{l_1, \ldots, l_{|V|}\} = \mathrm{H}_R(W). \qquad (9)$$

Similar to the other LSH-based methods from Section 2.3, we construct $\overline{V}(h)$ dynamically for every decoder state h. During each decoding step the same hash function is applied to the decoder state h to obtain a hashed binary query $q \in \{0,1\}^c$:

$$q = \mathrm{H}_R(h).$$

If the transformations in Equation 3 to Equation 8 were exact, we would only need to find the vocabulary candidate i corresponding to key $l_i \in L$ with the lowest Hamming distance from the query q (in the case of greedy decoding). However, all the approximations lead to search errors and we investigate a set of k best scoring candidates. This set

[3] Following the LSH implementation in FAISS, we use a Gaussian random rotation matrix $R \in \mathbb{R}^{d \times c}$. If $c \geq d$, FAISS constructs a matrix $R \in \mathbb{R}^{c \times c}$ composed of c orthonormal column vectors via QR factorization and then drops rows until we have $R \in \mathbb{R}^{d \times c}$.

[4] See Charikar (2002) for details. In short, the probability that the hash values for two vectors u, v match is given as $Pr(\mathrm{H}_r(u) = \mathrm{H}_r(v)) = 1 - \frac{\theta(u,v)}{\pi}$. When hashing to bit vectors of length c, the Hamming distance between these bit vectors $\mathrm{D}(\mathrm{H}_R(u), \mathrm{H}_R(v))$ approximates $\frac{\theta(u,v)}{\pi}c$.

[5] The magnitudes of decoder states and weight vectors probably do not vary a lot. However, decoder states h would be normalized to norm $\sqrt{d}$ via layer normalization at no additional computational cost if we dropped the affine transformation after layer normalization. The weight vectors of the output layer could be normalized to unit length after each parameter update during training or via weight normalization (Salimans and Kingma, 2016).

Dataset	#sentences de-en	fr-en	es-en
Europarl (Train)	1,920,209	2,007,723	1,965,734
dev2006 (Dev)	2,000	2,000	2,000
nc-dev2007 (Test)	1057	1,057	1,057

Table 1: Data used for training, validation and testing.

is our per decoder state vocabulary subset $\overline{V}(h)$:

$$\overline{V}(h) = \underset{\substack{V' \subset V \\ |V'|=k}}{\operatorname{argmin}} \sum_{i \in V'} \mathrm{D}(q, l_i).$$

In cases where there are more than k elements that would qualify based on their Hamming distance, we retrieve only the first k found. Note, this concludes our construction of $\overline{V}$ and unlike Shi et al. (2018), we do not need to extend $\overline{V}$ with a large word-frequency list.

Next, for every decoder state h, $\overline{W} \in \mathbb{R}^{|\overline{V}| \times d}$ and $\overline{b} \in \mathbb{R}^{|\overline{V}|}$ are subselected from the output layer parameters W and b, respectively, by restricting entries to those corresponding to vocabulary indices in $\overline{V}$. $\overline{W}$ and $\overline{b}$ replace W and b in calculating softmax and the best output token, $\hat{y}_t$, replacing the standard output layer computation in Equation 1 with Equation 2.

The number of hashes per input vector c and the number of target vocabulary to keep k are hyperparameters in our LSH implementation.

4 Experimental Setup

We train a model with 6-layer Transformer (Vaswani et al., 2017) encoder with 6-layer SSRU (Kim et al., 2019) decoder, trained using Marian (Junczys-Dowmunt et al., 2018) and using the same toolkit for inference. This is a strong and realistic model for production MT environments which balances translation quality and efficiency.

We use SentencePiece (Kudo and Richardson, 2018) with 32,000 tokens for all models, shared between both source and target language.

We use the FAISS (Johnson et al., 2019) implementation of SimHash hash described in Section 3.1.

We trained with four translation directions (German-English, English-German, Spanish-English, and French-English) Europarl corpus (Koehn, 2005), validated on the held out development set from the same corpus (*'dev2006'*) and tested on the out-of-domain New Commentary test set (*'nc-dev2007'*). See Table 1 for data set sizes. Results are reported for German-English in Section 5, results for other language pairs are available in the Appendix 7.2.

We use a two stage training procedure. In the first stage, we train a translation model directly on the parallel data. We create a synthetic parallel corpus by translating the source side of the parallel corpus with the initial model. The original source is paired with these translations to form the synthetic corpus for stage two. For the second stage of training, we then consider two cases: (1) where the 2nd stage model topology is identical (i.e. the original model and the new model both have or lack the output bias) and (2) where there is some change in model topology.

For case (1), we use *self-training*: the first-stage model are fine-tuned using the synthetic corpus. For case (2): we use sequence-level *knowledge distillation*: new models are distilled (Hinton et al., 2015) by training from scratch on the synthetic corpus. By abuse of terminology, in both scenarios, we call the first-stage model the *teacher*, and the fine-tuned or distilled model the *student*.

Translation quality was measured using SacreBLEU (Post, 2018). We define *search errors* as the number of lines changed in the translation output when vocabulary selection is applied.

We measure the time taken to do inference on one core of a 12 core Intel Xeon CPU, on a PC with 16GB RAM, running Ubuntu 20.04 within a WSL2 hypervisor.

For short listing, we create a candidate list of target sub-word translations for each source sub-word by using word alignments obtained from FastAlign (Dyer et al., 2013). A shortlist of target subwords is created before the translation of each sentence to constrain the possible output sub-words.

5 Results and analysis

Table 2 shows results on the full teacher-student training procedure, compared to the baseline, teacher, and lexical shortlisting. Our proposed method maintains the same translation quality as greedy search, with a 57% to 80% speedup. By contrast, shortlisting has search errors in 12% to 25% of sentences, with a speedup of between 67% to 74%.

	de-en			fr-en			es-en		
	speed ↑	BLEU ↑	search error ↓	speed ↑	BLEU ↑	search error ↓	speed ↑	BLEU ↑	search error ↓
Teacher	2.84	28.9		2.73	31.0		2.72	40.5	
Student	2.71 (-5%)	**29.9**		2.91 (+7%)	**31.9**		3.01 (+11%)	**42.2**	
Student w/ shortlist (baseline)	**4.76** (+68%)	29.6	25%	4.69 (+72%)	31.8	12%	4.73 (+74%)	42.0	17%
Student w/ LSH (this work)	4.46 (+57%)	**29.9**	**0%**	**4.92** (+80%)	**31.9**	**0%**	**5.08** (+87%)	**42.2**	**0%**

Table 2: Translation speed (sent./sec.), quality (BLEU) and search error for the teacher model, student model with full vocabulary, student model with the shortlist, and student model with LSH vocabulary selection. Hyperparameters chosen for lowest possible search error. Models trained with no bias and with label smoothing, and use a beam size of 1.

Model	Baseline BLEU ↑	Using LSH BLEU ↑	Using LSH search error ↓
With bias	28.9	0.1	100%
With bias LS	**29.7**	0.0	100%
No bias	29.1	29.1	7%
No bias LS	29.2	29.2	6%

Table 3: Translation quality (BLEU) for baseline teacher models, and when using LSH ($k = 1024$, $c = 2048$).

In order to understand the contributions of different aspects of the method, we perform additional experiments for analysis.

5.1 The effect of output bias

While the softmax of Equation 1 is dependent on the output bias b as well as output weights W, there is no easy way to include the bias b in the hashed representation of L in Equation 9. To see what effect this omission by the LSH hashing function has on translation, we will train and evaluate models with and without the output bias.

The first column in Table 3 compares the translation quality between models with and without output bias, based on BLEU scores. Models with output bias and training with label smoothing (LS) of 0.1 improve translation quality.

Column two and three in Table 3 show the consequences of applying LSH with the output vocabulary size of $k = 1024$ and the number of hashes set to $c = 2048$ to the baseline models. LSH causes overwhelming search errors in models with output biases, leading to catastrophic collapse in BLEU. This is unsurprising as the LSH does not take the bias into account when computing similarity. On the other hand, models without output bias are not hugely affected by LSH. Between 2% to 7% of the translations suffer from search errors but this has a negligible affect on translation quality.

Model	Beam 1	Beam 4
Teacher with bias	28.9	29.9
+ Student no bias	29.1	29.9
+ Student no bias LS	29.6	30.4
Teacher with bias LS	29.7	30.3
+ Student no bias	28.7	29.3
+ Student no bias LS	29.6	30.2
Teacher no bias	29.1	29.8
+ Student no bias	30.1	**30.9**
+ Student no bias LS	**30.2**	30.8
Teacher no bias LS	29.2	29.9
+ Student no bias	30.0	30.4
+ Student no bias LS	29.9	30.5

Table 4: Translation quality of distilled models on held-out test set (BLEU) with different beam widths.

Based on these results, further experiments with LSH only use models without output bias.

5.2 Comparison with lexical shortlisting

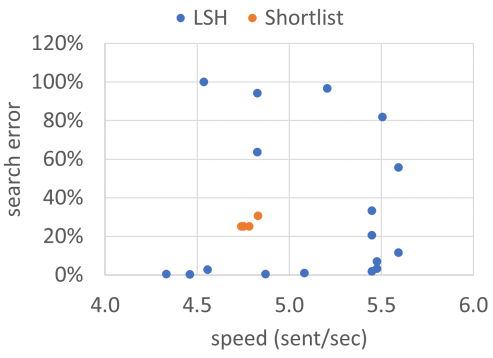


Figure 2: Comparison of translation speed (sent./sec.) vs search error between LSH and lexical shortlisting.

	Beam 1			Beam 4		
	speed ↑	BLEU ↑	search error ↓	speed ↑	BLEU ↑	search error ↓
Teacher model	2.84	29.2		1.52	29.9	
Student model	2.71 (-5%)	29.9		1.47 (-3%)	30.5	
LSH	3.94 (+39%)	29.9	0%	2.01 (+32%)	30.5	13%

Table 5: Translation speed (sent./sec.), quality (BLEU) and search error for student model (trained and distilled with label smoothing, without bias). Compared with teacher without bias or label smoothing. LSH is using parameters 1024-best vocab items, and a hash size of 2048.

Figure 2 shows the translation speed / search error trade-off for LSH and lexical shortlisting for various hyperparameter settings. For shortlisting, We experimented with $|V_f| = 100$ and $|V_a| = 10, 25, 50, 75$ and 100. We have not observed significant changes for larger $|V_a|$. The methods were applied to a model trained, then distilled with no output bias and with layer normalization. Our training procedure and LSH inference algorithm is not only faster than shortlisting but also result in less search errors.

5.3 LSH in teacher-student training

The Hamming distance of the hash vectors in Equation 7 is used in an approximate similarity measure between the decoder state h and each embedding vector corresponding to vocabulary items in V. We would like to increase this similarity for the correct output and decrease it for incorrect output at each time step. Kim and Rush (2016) demonstrated that knowledge distillation create student models with a more peaked distribution, i.e. the probability mass is concentrated around only few vocabulary words. This likely carries over into the space of Hamming distances, separating similar vector pairs from dissimilar ones, a potentially useful phenomenon that the search can take advantage of. See also Section 5.6 for similar considerations on the effects of label smoothing.

Figure 3 shows the trade-off between the LSH top-k versus search errors, for teacher models without and with output bias, and student models without bias, respectively. Similarly, Figure 4 shows the trade-off with the number of hashes c used in LSH. These plots also show that:

1. teacher-student training significantly reduces LSH search errors,
2. teacher models *with* label smoothing have lower search errors,
3. student models *without* label smoothing have lower search errors,
4. all student models converges to minimal, or even zero, search errors with increased top-k.

5.4 Translation speed vs hash size

Predictably, translation speed increases if the LSH parameters decreases. For example, Figure 5 shows the translation speed when the number of hashes are varied. At very low hash counts, the systems often output lengthy sentences with repetitive gibberish, lowering speed. Of course, this has a negative impact on search errors and translation quality.

5.5 Larger beam size

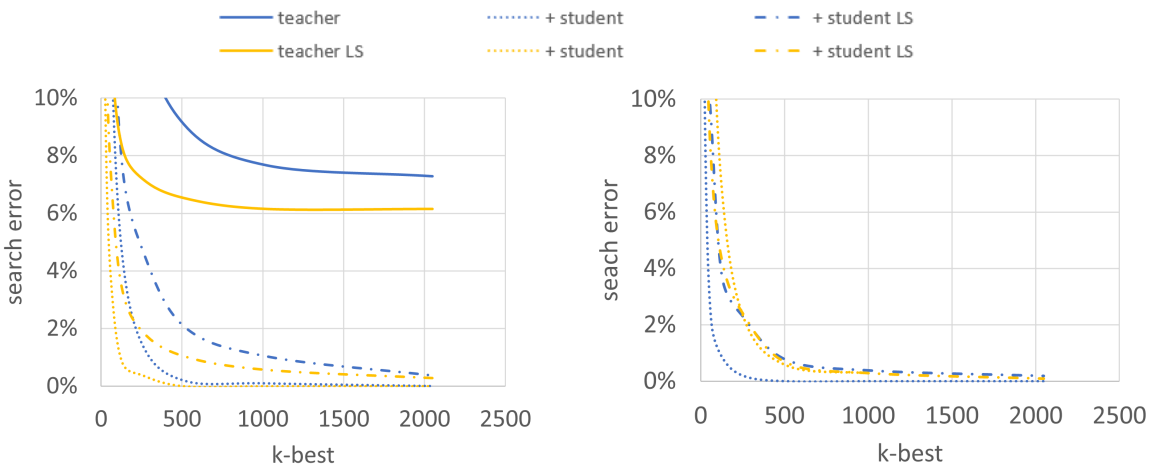
Table 5 shows translation quality when using a larger beam which, as expected, is higher in all cases than using beam width 1.

However, LSH vocabulary selection causes more search errors for larger beam sizes. Figure 6 compare the search errors for the same model using beam width 1 and 4 by varying the LSH hyperparameters. The same conclusion can be drawn from Table 5 (Beam 4).

A possible cause for this increased search error is in the calculation of the softmax denominator. The denominator for each beam is the sum of logits in the beam. When using vocabulary selection, the denominator is approximated by calculating it only over a k-best subset of the logits. The softmax probability would be distorted if the excluded logits contain significant probability mass.

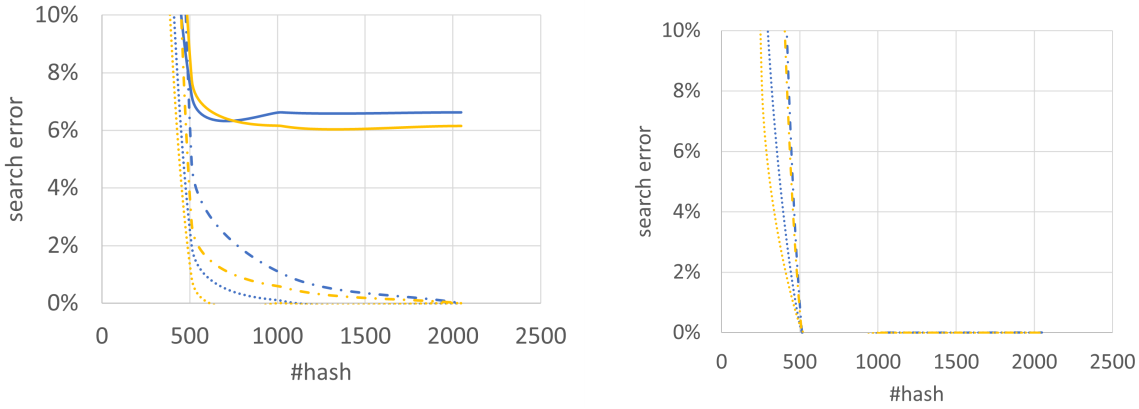
This is not an issue with beam size 1 as an approximation error in the denominator would change the absolute probabilities but won't affect the relative probabilities within the beam.

However, for beam size larger than one, probabilities across different beams are compared. A logit approximation error in this case would distort the



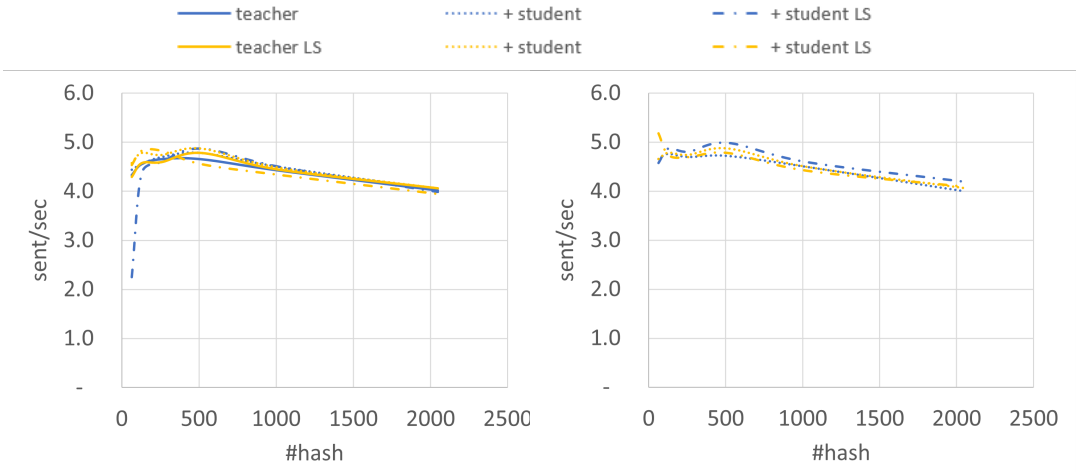
(a) without output bias (b) with output bias

Figure 3: LSH k-best vs. search errors.



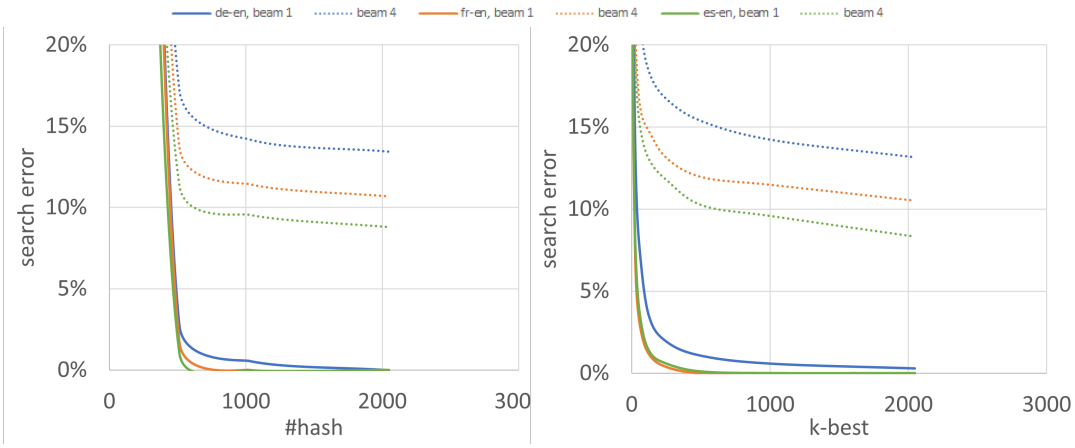
(a) without output bias (b) with output bias

Figure 4: LSH #hashes vs. search errors.



(a) Without bias (b) With bias

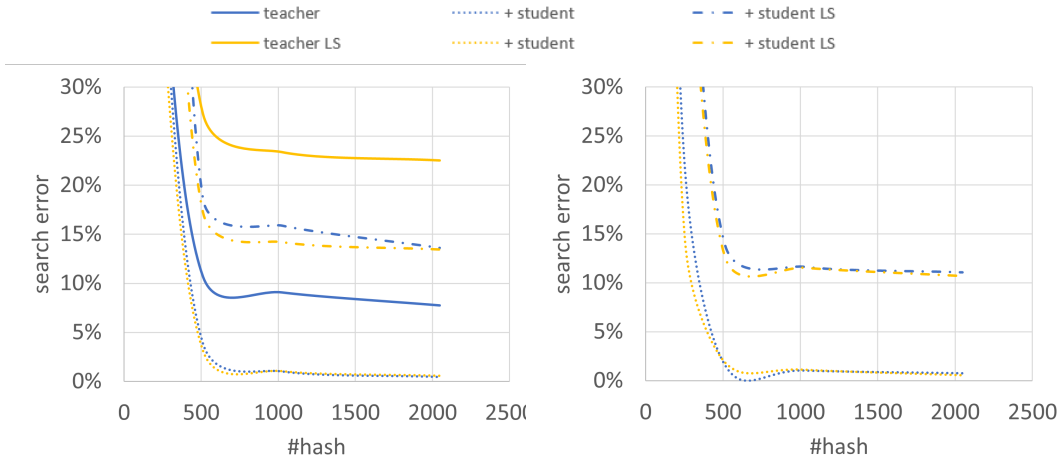
Figure 5: Translation speed (sent./sec.) vs #hashes



(a) Search error vs. #hashes

(b) Search error vs. LSH k-best

Figure 6: Search error for beam size 1 & 4: teacher LS + student LS.



(a) teacher without output bias

(b) teacher with output bias

Figure 7: LSH #hashes vs. search errors for teacher models, beam width 4.

comparison between vocabulary items in different beam, leading to search errors.

5.6 Label smoothing

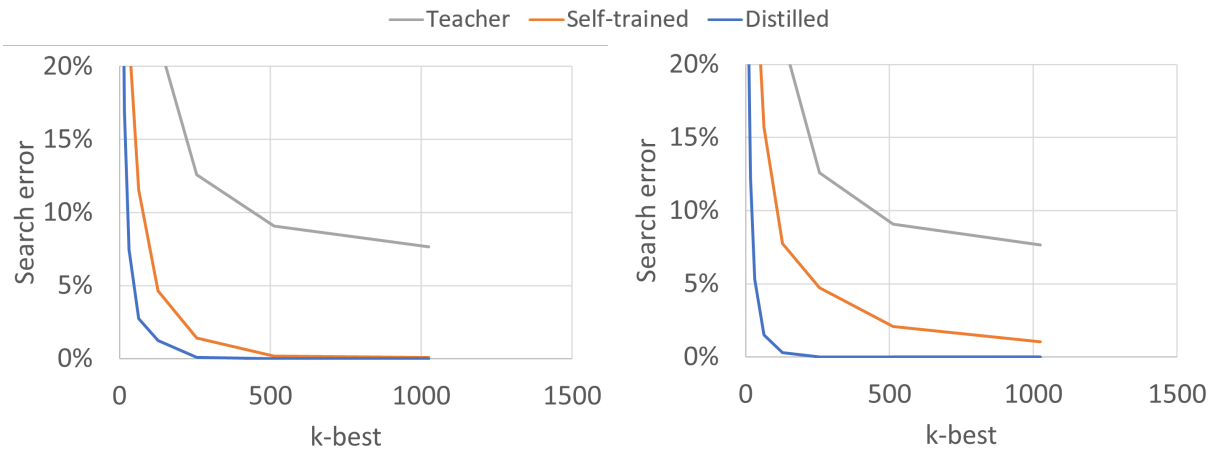
While label smoothing (Szegedy et al., 2016) can improve translation quality—by spreading the probability over many output classes to avoid overfitting— models with label smoothing are detrimentally affected by larger beam widths when LSH is used. Figure 7 shows that both student and teacher models have higher search errors when trained with label smoothing. Since label smoothing distributes a portion of the probability mass over the entire vocabulary, the excluded logits will contain a larger amount of the total probability mass, exacerbating the problem caused by the larger beam size. Since label smoothing may also reduce information transfer in knowledge distillation (Müller et al., 2019), we recommend training students without label smoothing, especially when using larger beams.

5.7 Self-training vs distillation

Thus far, we have fined-tuned ('self-trained') models where the second stage model is architecturally identical to the first, otherwise we distilled a student model from the first stage model.

For first stage models with no bias, Table 6 shows that fine-tuning result in better translation quality than training from scratch with the synthetic data.

However, the fine-tuned models have slightly higher search errors, nevertheless both training strategies result in models which have much lower search errors than the original first stage model, Figure 8.



(a) no label smoothing

(b) label smoothing 0.1

Figure 8: LSH Fine-tuned vs. distilled models. Both are teachers & student with no bias

Model	Beam 1	Beam 4
Teacher no bias	29.1	29.8
+ Self-trained no bias	30.1	**30.9**
+ Self-trained no bias LS	**30.2**	30.8
+ Distilled no bias	29.3	29.9
+ Distilled no bias LS	29.5	30.5
Teacher no bias LS	29.2	29.9
+ Self-trained no bias	30.0	30.4
+ Self-trained no bias LS	29.9	30.5
+ Distilled no bias	29.5	30.3
+ Distilled no bias LS	29.8	30.3

Table 6: Translation quality of fine-tuned vs. distilled models (BLEU).

6 Conclusion

We demonstrate that, with the proper training procedure, using locality sensitive hashing for vocabulary selection can significantly boost translation speed while consistently producing negligible search errors.

We make the following recommendations for use in practice:

For existing models and greedy search, perhaps where we may not know the exact training procedure and model, we can create a model that works with LSH vocabulary selection by distilling the original model to a comparable model without output bias. Using label smoothing in the distillation can improve its translation quality if the original model was not trained with it. There will be minimal search errors in using LSH while achieving significant speed improvement.

To train a new model for use with greedy search, a two stage procedure should also be used where the second stage is fine-tuned on the output of the first. Both stages should train models without output bias. Again, the fine-tuned models can be trained with label smoothing without affecting the effectiveness of LSH.

LSH vocabulary selection introduce search errors for larger beam sizes, especially when label smoothing is used during fine-tuning. Therefore, if using larger beams in inference, it is recommended not to use label smoothing in the distillation or fine-tuning step.

References

Ondřej Bojar, Rajan Chatterjee, Christian Federmann, Barry Haddow, Chris Hokamp, Matthias Huck, Varvara Logacheva, and Pavel Pecina, editors. 2015. *Proceedings of the Tenth Workshop on Statistical Machine Translation*. Association for Computational Linguistics, Lisbon, Portugal.

Moses Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *STOC '02*.

Beidi Chen, Tharun Medini, James Farwell, Sameh Gobriel, Charlie Tai, and Anshumali Shrivastava. 2020. Slide : In defense of smart algorithms over hardware acceleration for large-scale deep learning systems.

Tobias Domhan, Eva Hasler, Ke Tran, Sony Trenous, Bill Byrne, and Felix Hieber. 2022. The devil is in the details: On the pitfalls of vocabulary selection in neural machine translation. In *NAACL*.

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.

Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2016. Efficient softmax approximation for gpus.

Hiroaki Hayashi, Yusuke Oda, Alexandra Birch, Ioannis Konstas, Andrew Finch, Minh-Thang Luong, Graham Neubig, and Katsuhito Sudoh. 2019. Findings of the third workshop on neural generation and translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 1–14, Hong Kong. Association for Computational Linguistics.

Kenneth Heafield, Hiroaki Hayashi, Yusuke Oda, Ioannis Konstas, Andrew Finch, Graham Neubig, Xian Li, and Alexandra Birch. 2020. Findings of the fourth workshop on neural generation and translation. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*, pages 1–9, Online. Association for Computational Linguistics.

Kenneth Heafield, Qianqian Zhu, and Roman Grundkiewicz. 2021. Findings of the WMT 2021 shared task on efficient translation. In *Proceedings of the Sixth Conference on Machine Translation*, pages 639–651, Online. Association for Computational Linguistics.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015a. On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1–10, Beijing, China. Association for Computational Linguistics.

Sébastien Jean, Orhan Firat, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015b. Montreal neural machine translation systems for WMT'15. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 134–140, Lisbon, Portugal. Association for Computational Linguistics.

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

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. 2018. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia. Association for Computational Linguistics.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.

Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan, Alham Fikri Aji, Kenneth Heafield, Roman Grundkiewicz, and Nikolay Bogoychev. 2019. From research to production and back: Ludicrously fast neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 280–288, Hong Kong. Association for Computational Linguistics.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of Machine Translation Summit X: Papers*, pages 79–86, Phuket, Thailand.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pages 66–71. Association for Computational Linguistics.

Gurvan L'Hostis, David Grangier, and Michael Auli. 2016. Vocabulary selection strategies for neural machine translation. *CoRR*, abs/1610.00072.

Haitao Mi, Zhiguo Wang, and Abe Ittycheriah. 2016. Vocabulary manipulation for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 124–129, Berlin, Germany. Association for Computational Linguistics.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *AISTATS*.

Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. 2019. When does label smoothing help? In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Yusuke Oda, Philip Arthur, Graham Neubig, Koichiro Yoshino, and Satoshi Nakamura. 2017. Neural machine translation via binary code prediction.

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.

Tim Salimans and Diederik P. Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Xing Shi and Kevin Knight. 2017. Speeding up neural machine translation decoding by shrinking run-time vocabulary. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 574–579, Vancouver, Canada. Association for Computational Linguistics.

Xing Shi, Shizhen Xu, and Kevin Knight. 2018. Fast locality sensitive hashing for beam search on gpu.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Sudheendra Vijayanarasimhan, Jonathon Shlens, Rajat Monga, and Jay Yagnik. 2014. Deep networks with large output spaces.

Jay Yagnik, Dennis Strelow, David A. Ross, and Rueisung Lin. 2011. The power of comparative reasoning. In *2011 International Conference on Computer Vision*, pages 2431–2438.

7 Appendix

7.1 Practical Considerations

Here, we discuss some practical considerations for use of the LSH decoding.

To train a Marian model without an output bias, add the following switches.
Marian training command:[6]

```
--output-omit-bias
```

To train without label smoothing:

```
--label-smoothing 0
```

To use the LSH vocabulary selection during inference, execute marian-decoder with the following switches:

```
--output-approx-knn [k] [c]
```

where [k] is the number of k-best vocabulary items and [c] is the number of hashes to use.

7.2 Results for Additional Language Pairs

Model	de-en	fr-en	es-en
With bias	28.9	31.0	40.5
With bias LS	**29.7**	**31.4**	**41.3**
No bias	29.1	31.3	41.0
No bias LS	29.2	31.2	41.1

Table 7: Baseline translation quality (BLEU) w/ & w/o bias and w/ or w/o label smoothing.

Model	de-en	fr-en	es-en
With bias	0.1 100%	0.1 100%	0.0 100%
With bias LS	0.0 100%	0.0 100%	0.0 100%
No bias	29.1 7%	31.2 4%	41.0 2%
No bias LS	29.2 6%	31.1 4%	41.1 2%

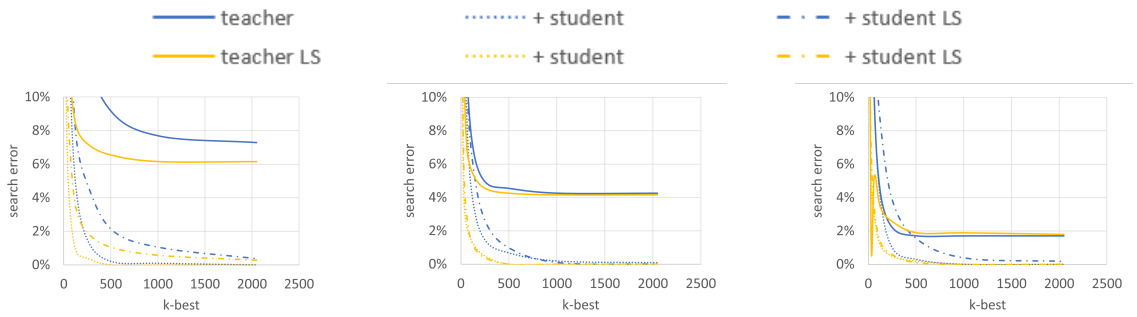
Table 8: Translation quality (BLEU) & search errors (percentages) when using LSH ($k = 1024$, $c = 2048$).

	Beam 1			Beam 4		
Model	de-en	fr-en	es-en	de-en	fr-en	es-en
Teacher with bias	28.9	31.0	40.5	29.9	31.6	41.4
+ Student no bias	29.1	31.3	40.1	29.9	31.8	40.9
+ Student no bias LS	29.6	31.6	41.4	30.4	32.5	42.0
Teacher with bias LS	29.7	31.4	41.3	30.3	32.4	42.2
+ Student no bias	28.7	31.2	40.8	29.3	31.8	41.3
+ Student no bias LS	29.6	31.5	41.0	30.2	32.1	41.9
Teacher no bias	29.1	31.3	41.0	29.8	31.8	41.5
+ Student no bias	30.1	31.4	41.8	**30.9**	32.1	42.2
+ Student no bias LS	**30.2**	32.2	41.8	30.8	32.4	42.5
Teacher no bias LS	29.2	31.2	41.1	29.9	32.3	41.8
+ Student no bias	30.0	**32.5**	41.9	30.4	**32.8**	42.3
+ Student no bias LS	29.9	31.9	**42.2**	30.5	**32.8**	**42.7**

Table 9: Translation quality of distilled models on held-out test set (BLEU) with different beam widths.

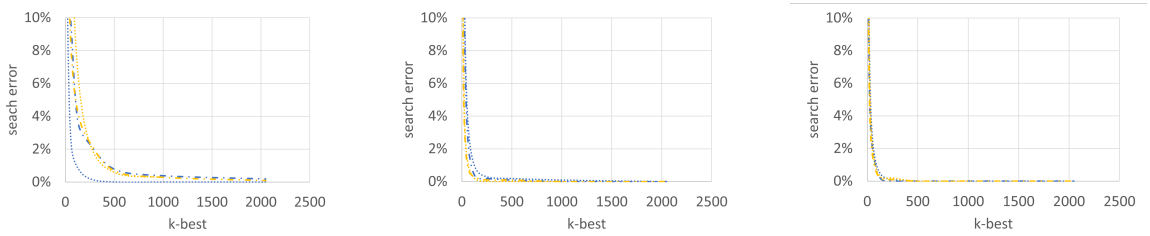
Our results thus far have been on language pairs with English as the target language. We trained and finetuned English to German models, both without output bias and label smoothing. Figure 17 shows that LSH vocabulary selection works just as well when German is the target language.

[6] github.com/marian-nmt/marian



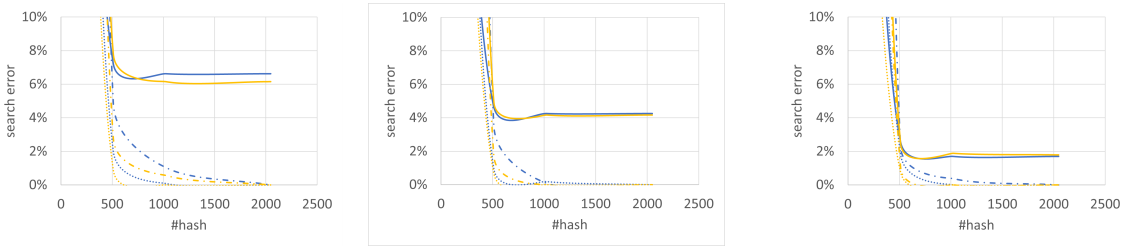
(a) de-en (b) fr-en (c) es-en

Figure 9: LSH k-best vs. search errors for models without output bias.



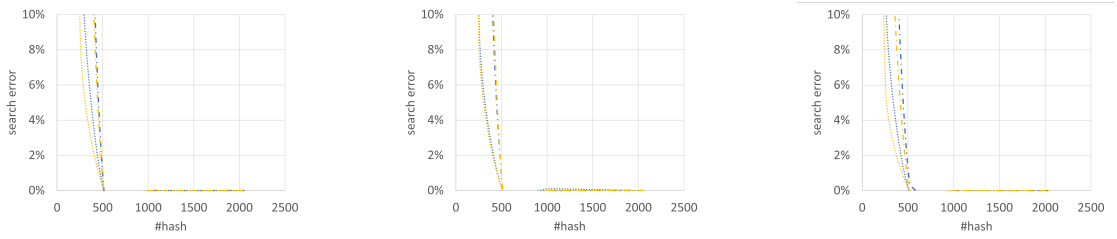
(a) de-en (b) fr-en (c) es-en

Figure 10: LSH k-best vs. search errors for models with output bias.



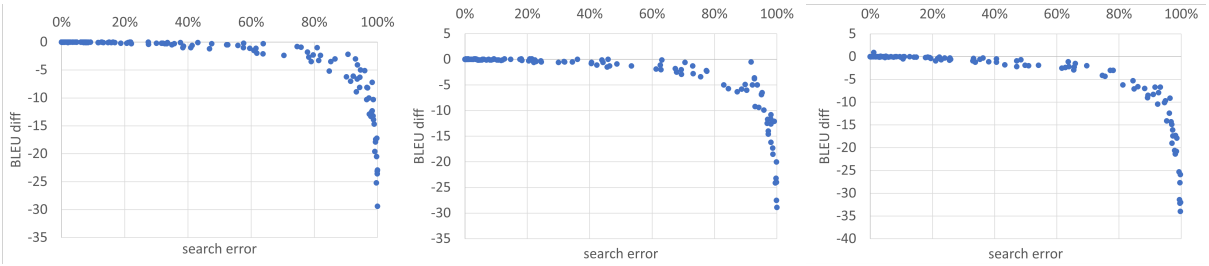
(a) de-en (b) fr-en (c) es-en

Figure 11: LSH #hashes vs. search errors for models without output bias.



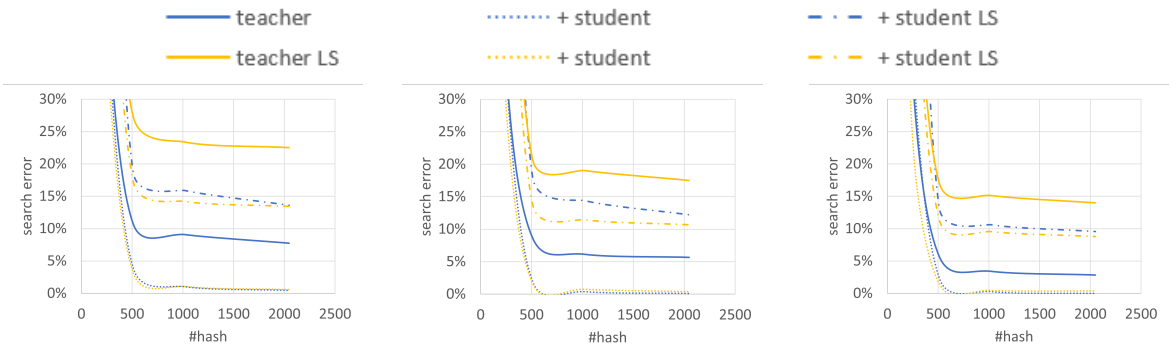
(a) de-en (b) fr-en (c) es-en

Figure 12: LSH #hashes vs. search errors for models with output bias.



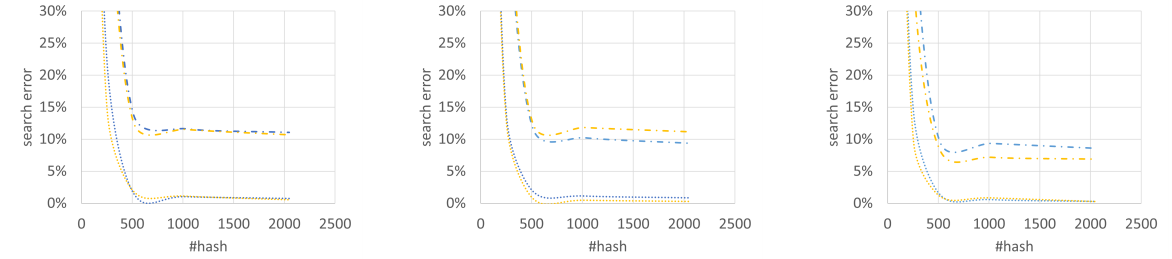
(a) de-en (b) fr-en (c) es-en

Figure 13: Search errors vs. decrease in BLEU.



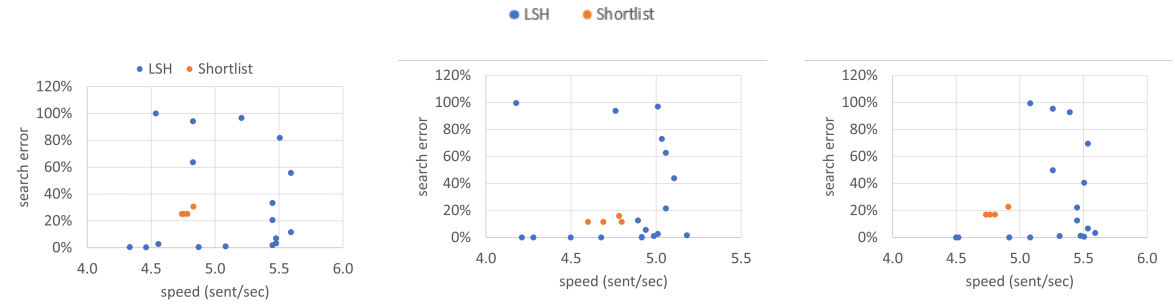
(a) de-en (b) fr-en (c) es-en

Figure 14: LSH #hashes vs. search errors for teacher models without output bias using beam width of 4.



(a) de-en (b) fr-en (c) es-en

Figure 15: LSH #hashes vs. search errors for teacher models with output bias using beam width of 4.



(a) de-en (b) fr-en (c) es-en

Figure 16: Comparison of translation speed (sent./sec.) vs search error between LSH and lexical shortlisting.

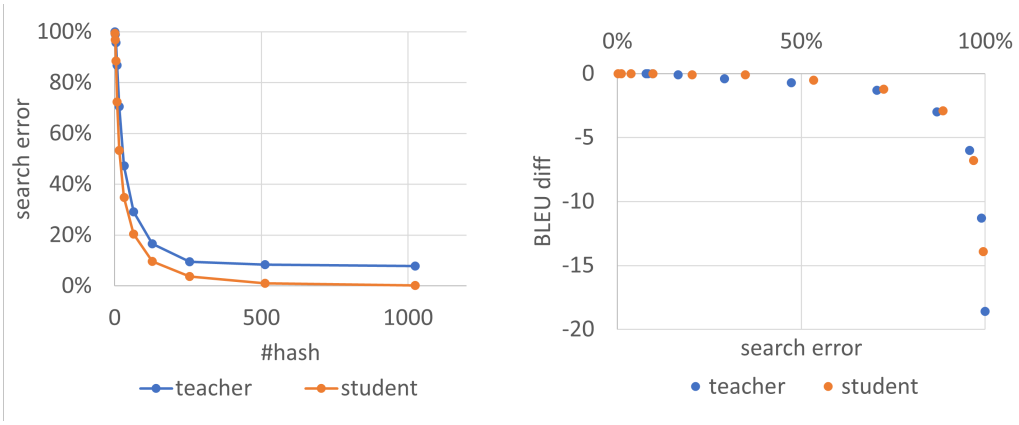


Figure 17: English-German results