The VolcTrans System for WMT22 Multilingual Machine Translation Task

Xian Qian¹, Kai Hu¹, Jiaqiang Wang¹, Yifeng Liu²
Xingyuan Pan³, Jun Cao¹, Mingxuan Wang¹
¹ ByteDance AI Lab, ² Tsinghua University, ³ Wuhan University
{qian.xian, hukai.joseph, wangjiaqiang.sonian, caojun.sh, wangmingxuan.89}@bytedance.com
liuyifen20@mails.tsinghua.edu.cn, panxingyuan209@gmail.com

1 Introduction

Multilingual Machine Translation attracts much attention in recent years due to its advantages in sharing cross-lingual knowledge for low-resource languages. It also dramatically reduces training and serving costs. Training a multilingual model is much faster and simpler than training many bilingual ones. Serving multiple low-traffic languages using one model could drastically improve GPU utilization.

The WMT22 shared task on large-scale multilingual machine translation includes 24 African languages (Adelani et al., 2022b). Inspired by previous research works, we train a deep transformer model to translate all languages since large models have been demonstrated effective for multilingual translation (Fan et al., 2021; Kong et al., 2021; Zhang et al., 2020). We participated in the unconstrained track that allows the use of external resources. Our system is a transformer-based multilingual model trained on data from multiple sources including the public training set from the data track, NLLB data provided by Meta AI, self-collected parallel corpora, and pseudo bitext from back-translation. A series of heuristic rules clean both bilingual and monolingual texts. On the official test set, our system achieves 17.3 BLEU, 21.9 spBLEU, and 41.9 chrF2++ on average over all language pairs. The average inference speed is 11.5 sentences per second using a single Nvidia Tesla V100 GPU. Our code and trained models are available at https://github.com/xian8/wmt22

2 Data

2.1 Data Collection

Our training data are mainly from four sources: the official set for constrained track, NLLB data provided by Meta AI, self-collected corpora, and pseudo training set from back translation.

For each source, we collect both parallel sentence pairs and monolingual sentences. A parallel sentence pair is collected if one side is in African language and the other is in English or French. We did not collect African-African sentence pairs as we use English as the pivot language for African-to-African translation. Instead, they are added to the monolingual set. More specifically, we split every sentence pair into two sentences and add them to the monolingual set accordingly. For example, the source side of a fuv-fon sentence pair is added to the fuv set. This greatly enriches the monolingual dataset, especially for the very low-resource

1 https://github.com/bytedance/lightseq
languages.

We merge multiple corpora from the same source into one and use bloom filter \(^2\) (Bloom, 1970) for fast deduplication. To reduce false positive errors which over delete distinct samples, we set the error rate \(10^{-7}\) and capacity of \(4B\) samples which costs \(100G\) host memory.

The official set includes the data from data track participants, OPUS collections, and the NLLB parallel corpora mined from Common Crawl (com) and other sources. All domains in OPUS collections are involved, such as Mozilla-I10n, which could introduce many noises such as programming languages, and needs extra rules to clean.

NLLB data provided by Meta AI has three subsets: primary bitext including a seed set that is carefully annotated for representative languages and a public bitext set downloaded from open sources and mined bitexts that are automatically discovered by LASER3 encoder in a global mining pipeline, back-translated data from a pretrained model. We add the first two subsets in our training set.

Some public bitext data that are no longer available or require authorization such as JW300 (Agić and Vulić, 2019), Lorelei\(^3\) and Chichewa News\(^4\) are not included. We noticed that the NLLB team released another version of mined data recently in hugging-face\(^5\), which is different from the version on the WMT22 website. We merge the new version into the old one and remove duplicates.

We collected additional bitexts in two ways: large-scale mining from general web pages, and manually crawling from specific websites and sources.

Large-scale mining focused on two scenarios, parallel sentences appearing on a single web page such as dictionary web pages that use multiple bilingual sentences to exemplify the usage of a word, and parallel web pages that describe the same content but are written in different languages. We extract these pages from the Common Crawl corpus. Then we utilized Vecalign (Thompson and Koehn, 2019), an accurate and efficient sentence alignment algorithm to mine parallel bilingual sentences. We use LASER (Schwenk and Douze, 2017) encoders released by WMT to obtain multilingual sentence embeddings and facilitate the alignment work. We collected about 3 million sentence pairs namely LAVA corpus and submitted them to the data track. And another 150M pairs for the unconstrained track.

Specific websites and sources have fewer but higher-quality sentence pairs. For example, the bible website\(^6\) labels the order of sentences across languages so we can align them easily without sentence segmentation. Since JW300 is not publicly available, we crawled pages from Jehovah’s Witnesses\(^7\) to recover the dataset.

Monolingual texts have richer sources such as VOA news in Amharic\(^8\) and OSCAR (Abadji et al., 2022), which improve English/French → African translation using back-translation. Monolingual texts from parallel data are also collected as described above. For African → English/French translation, we clean Wikipedia pages in English/French to get monolingual texts. For languages that gain significantly from back-translation such as Wolof, we run another round of back-translation to generate high-quality pseudo data.

2.2 Data Cleaning

We used the following rules to clean parallel datasets, except the NLLB mined bitext.

- Filter out parentheses and texts in between if the numbers of parentheses in two sentences are different.
- Filter out sentence pairs if numbers mismatch or one sentence ends with punctuation : ! ? ... and the other mismatches.
- Filter out sentences shorter than 30 characters, sentences having URLs or emails, or words longer than 100 characters.
- De-duplication: remove sentence pairs sharing the same source or target but having different translations.
- Sentences having programming languages are removed. We manually create a set of keywords to detect programming languages, such as if ( , , == and .getAttribute .
- Language identification using the NLLB language identification model trained by fastText (Joulin et al., 2017)

\(^2\)https://pypi.org/project/bloom-filter
\(^3\)https://catalog.ldc.upenn.edu/LDC2021T02
\(^4\)https://zenodo.org/record/4315018#.YpJWezML0p
\(^5\)https://huggingface.co/datasets/allenai/nllb
\(^6\)https://www.bible.com/languages
\(^7\)https://www.jw.org
\(^8\)https://amharic.voanews.com/
One type of noisy text could survive the rules above, which has repeat patterns and commonly exists in many datasets. Here are some examples:

```
Download Bongeziwe Mabandla mini esadibana ngayo (#001) Mp3 Bongeziwe Mabandla - mini esadibana ngayo (#001).
Coaster Gift, Paper-Cut Coaster Zodiac, Red Coaster Cute, Paper-Cut Zodiac Coaster mm mm mm MPEE(um) MPEP(um) mm mm mm mm mm mm kg kg mm mm mm
```

A natural choice to detect these repeating patterns is the minimum description length (MDL) which finds the optimal compression by encoding frequent substrings with shorter codes.

Specifically, given a sentence $s$, our MDL objective minimizes the length of the codebook plus the bits to encode the sentence:

$$
\text{MDL}(s) = \min_{s = w_1 w_2 \ldots w_n} \left( C \sum |w| \text{ distinct } w - \sum_{i} \log(p(w_i | w_{i-1})) \right)
$$

where $w_1, w_2, \ldots, w_n$ is the word (coding entry) sequence, $C$ is a positive constant, which balances the contribution of the codebook and length of the encoded sequence. $|w|$ is the length of word $w$. In our experiments, we set $C = 2$.

$p(w_i | w_{i-1}) = \frac{\#w_i | w_{i-1}}{\#w_{i-1}}$ is the conditional probability of word bigrams in the sequence.

A sentence is noisy if the ratio of MDL over sentence length is less than a predefined threshold:

$s$ is noisy if $\frac{\text{MDL}(s)}{|s|} < T$

If a sentence has no repeat patterns at all, then the length of the codebook should be $C|s|$, and $\text{MDL}(s) \geq C|s|$. Thus we choose $T = C$.

For the NLLB mined corpus, we remove pairs with laser score $< 1.06$ or language score $< 0.95$ provided by LASER. Monolingual texts are cleaned using language scores only.

Table 1 and Figure 1 summarize the size of our training data after data cleaning.

### Table 1: Number of sentence pairs from different sources after data cleaning.

<table>
<thead>
<tr>
<th>Source</th>
<th>Sentence Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constrained Track</td>
<td>50.5M</td>
</tr>
<tr>
<td>NLLB</td>
<td>29.1M</td>
</tr>
<tr>
<td>Self Collected</td>
<td>151.6M</td>
</tr>
<tr>
<td>Back Translation</td>
<td>1.41B</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1.64B</strong></td>
</tr>
</tbody>
</table>

We reuse the mRASP2 tokenizer, a unigram model trained on 150 languages using SentencePiece (Pan et al., 2021). To support new African languages, we train another tokenizer for new languages and merge it to the mRASP2 tokenizer. To ensure that the merged tokenizer produces the same segmentation for old languages, new words that can be made by joining two or more old words are removed and the rest new words’ probabilities are scaled down.

We notice that the Yoruba text in FLORES200 has more accented characters than other corpora. According to NLLB team’s report, the way FLORES200 marks the tone of vowels is similar to MAFAND dataset (Adelani et al., 2022a). Thus, we use the MAFAND data to train an accent model to post-process the translated sentences for $X \rightarrow$ Yoruba translation. It takes the Yoruba character sequence with accents removed as the input and outputs the accented characters. The structure of the model is a two-layer bidirectional LSTM having 50 hidden units in each layer. Correspondingly, we train another accent model using non-MAFAND datasets to preprocess source text for Yoruba→$X$ translation.
3 Model

3.1 Model Architecture

Existing research works demonstrate that small models suffer from the underfitting problem for multilingual machine translation. On the other hand, training and serving large models are expensive. Sometimes model parallelism or pipeline parallelism is necessary if it is impossible to run training on a single GPU due to memory constraints. And quantization is required to reduce the latency of inference. Our compromised model is a pre-layer norm transformer with $2.1B$ parameters which can be trained using $A100$ GPUs with 80G memory without parallelism. Details of the model are described in Table 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoder Layer</td>
<td>64</td>
</tr>
<tr>
<td>Decoder Layer</td>
<td>64</td>
</tr>
<tr>
<td>Hidden Size</td>
<td>1024</td>
</tr>
<tr>
<td>FFN dimension</td>
<td>4096</td>
</tr>
<tr>
<td>Max Length</td>
<td>512</td>
</tr>
<tr>
<td>Shared Embedding</td>
<td>Decoder input output</td>
</tr>
<tr>
<td>Positional Embedding</td>
<td>Learned</td>
</tr>
</tbody>
</table>

Table 2: Architecture of our transformer model

3.2 Language Tag

There are two popular language tag strategies for multilingual MT: S-ENC-T-DEC which adds source language token to encoder input and target language token to decoder input (Fan et al., 2021; Liu et al., 2020; Wu et al., 2021), and T-ENC which adds target language token to encoder input (Yang et al., 2021; Wu et al., 2021). Our system uses T-ENC-T-DEC which adds the target language token to both encoder and decoder inputs. We did not use source language information for two reasons. First, most translation engines detect input languages automatically, which may introduce incorrect source language tokens. Second, a source sentence may be written in mixed languages.

4 Training and Optimization

4.1 Platform

Our models are trained on 6 machines each equipped with 8 Nvidia A100 80G GPUs. We use our internal version of ParaGen (Feng et al., 2022), a self-developed text generation framework, to train the model. For back-translation, monolingual data are split and translated in parallel using 50 Nvidia Tesla V100 GPUs.

To accelerate training, LightSeq is integrated. Unlike approaches that proposed alternative model structures to trade quality for speed, LightSeq used a series of GPU optimization techniques tailored to the specific computation flow and memory access patterns of transformer models. It has been demonstrated 50% to 250% faster than Apex on machine translation tasks. (Wang et al., 2021, 2022) Its inference speed is about 11.5 sentences per second using a single Nvidia Tesla V100 GPU, which allows us to translate all monolingual texts within a month.

As the training set’s size exceeds the local disk’s capacity, it is stored on a remote Hadoop file system.

4.2 Hyper-parameter Tuning

We tune the hyperparameters using a hill climbing approach where each iteration searches along one direction with a different value in the hyperparameter space while keeping the others constant in order to converge to the locally optimal solution on the validation set. To search efficiently, we fix a small batch size and tune other parameters, then increase the batch size after the other parameters have been tuned.

The final configuration is listed in Table 3.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Learning Rate</td>
<td>0.001</td>
</tr>
<tr>
<td>Warmup Steps</td>
<td>1000</td>
</tr>
<tr>
<td>Learning Rate Scheduler</td>
<td>Inverse Square Root</td>
</tr>
<tr>
<td>Dropout Rate</td>
<td>0.1</td>
</tr>
<tr>
<td>Sampling Temperature</td>
<td>5</td>
</tr>
<tr>
<td>Label Smoothing</td>
<td>0.1</td>
</tr>
<tr>
<td>Optimizer</td>
<td>AdamW(0.9, 0.98)</td>
</tr>
<tr>
<td>Activation Function</td>
<td>ReLU</td>
</tr>
<tr>
<td>Batch Size</td>
<td>21M tokens</td>
</tr>
</tbody>
</table>

Table 3: Hyperparameters for training.

4.3 Streaming Data Shuffling

Data Shuffling reduces the variance of mini-batches and lowers the risk of local optimum. However, it is challenging to shuffle a Terabyte-scale dataset.
dynamically. Our system uses multi-source streaming data based shuffling, which maintains a small in-memory buffer and a set of file pointers that point to random offsets of the training set. Each time a file pointer is selected randomly and loads the next sample to the buffer. A batch of samples is drawn from the buffer randomly once the buffer is full. This approach takes the advantage of data prefetching for sequential access in the Hadoop file system. The randomness of the sampling is controlled by the number of file pointers and the size of the buffer. In our experiments, we use about 5k file pointers and 300G host memory for the buffer.

To compare with global dynamic shuffling, we run a simulation experiment. We train a model until convergence, then shuffle the full dataset statically, and continue training on the shuffled data. Repeat shuffling until no significant change in loss or performance. For clarity, the original model is named as $M_0$, and the model trained with $i-th$ round of statically shuffled data is named as $M_i$.

Table 4 shows the averaged per token loss of the last 100 training steps and averaged BLEU of English $\leftrightarrow$ African language translations. We observed a slight improvement in the first round, but no significant change in the second round. This experiment suggests that our shuffling method combined with a limited number of static shuffling is a good approximation of global dynamic shuffling.

4.4 Small Dynamic Vocabulary vs Large Static Vocabulary

Existing studies on vocabulary size do not reach a consensus. Large vocabularies often outperform small ones (Gowda and May, 2020), but not always (Liao et al., 2021)

Our vocabulary has 100k words, smaller than most of the other systems. Another difference is that our vocabulary is incrementally built for more than 150 languages, it may miss important words in new languages.

To understand the impact of vocabulary, we train another large unigram model with 200$K$ words on the 26 languages in this shared task. Table 5 shows the performance with different vocabularies. It is obvious that the 100$K$ vocabulary outperforms the 200$K$ vocabulary, about 0.3 improvement in BLEU on average.

<table>
<thead>
<tr>
<th>Vocabulary Size</th>
<th>Languages</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>100k words</td>
<td>173</td>
<td>21.97</td>
</tr>
<tr>
<td>200k words</td>
<td>26</td>
<td>21.64</td>
</tr>
</tbody>
</table>

Table 5: Average BLEU of English $\leftrightarrow$ African translations on the FLORES200 devtest set for the models with different vocabularies.

4.5 Pivot vs Direct

As reported in Microsoft’s work, pivot-based translation is more robust, especially for directions between low-resource languages since corpora of $X \leftrightarrow Y$ are commonly sparser than $X \leftrightarrow English$. (Yang et al., 2021) Therefore we use English as the pivot language for African-African translation. For French-African translation, the size of $X \leftrightarrow French$ data is comparable to $X \leftrightarrow English$. Thus, we train a model for both English and French and choose the better one during inference time.

4.6 Model Averaging

As suggested by other works, model averaging is a simple trick that could significantly improve the performance without changing the model structure or slowing the inference speed. The only cost is the external disk spaces to save intermediate checkpoints, which is trivial compared with GPU and memory costs.

We save the checkpoints every 100 updates of gradients and average the last $K$ checkpoints. By enumerating $K$ from 1 to 20, we find that $K = 10$ is large enough to capture most of the gains.

5 Results

5.1 System Tuning

We tune our model on the FLORES200 devtest dataset, starting with a base model trained on the official data for the constrained track. Then we add more datasets and apply the optimization described above to boost performance. Table 6 reports the averaged BLEU over 56 directions includ-
ing 24 African languages from and to English and 4 African languages from and to French.

Table 6: System tuning on FLORES200 devtest set, averaged BLEU over 56 directions is reported. Superscript † means the modification is not included in the final submission.

<table>
<thead>
<tr>
<th>Model Description</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base model</td>
<td>16.92</td>
</tr>
<tr>
<td>+ NLLB and self-collected data</td>
<td>18.89</td>
</tr>
<tr>
<td>+ Data cleaning</td>
<td>19.64</td>
</tr>
<tr>
<td>+ Back-translation data</td>
<td>22.85</td>
</tr>
<tr>
<td>+ X → English → French</td>
<td>22.95</td>
</tr>
<tr>
<td>+ French → English → X †</td>
<td>22.90</td>
</tr>
<tr>
<td>+ Yoruba Accent for X → Yoruba</td>
<td>23.20</td>
</tr>
<tr>
<td>+ Yoruba Accent for Yoruba → X †</td>
<td>23.17</td>
</tr>
<tr>
<td>+ Model Averaging</td>
<td>23.35</td>
</tr>
</tbody>
</table>

Table 7: Results of our primary submission on FLORES200 dev, FLORES200 devtest and official test datasets respectively. Metrics are averaged over 100 language pairs.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>BLEU</th>
<th>spBLEU</th>
<th>chrF++</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLORES dev</td>
<td>17.41</td>
<td>21.70</td>
<td>42.01</td>
</tr>
<tr>
<td>FLORES devtest</td>
<td>17.43</td>
<td>21.71</td>
<td>41.99</td>
</tr>
<tr>
<td>Official test</td>
<td>17.30</td>
<td>21.90</td>
<td>41.87</td>
</tr>
</tbody>
</table>

6 Conclusion

This paper presents our system for the WMT22 shared task on Multilingual Machine Translation for African Languages. We focus on data collection, augmentation, and cleaning. Due to the limited time, we did not try modeling tricks such as reranking and ensemble. Our finding is that the amount of data is crucial for translation quality, especially monolingual data in low-resource languages.

Acknowledgements

We thank Ying Xiong and Yang Wei for building the LightSeq package for this submission. We also thank the GMU-eval team for their effort to make our system work on the evaluation platform.

References


Official evaluation metrics include BLEU, sentence-piece BLEU (spBLEU) score, and chrF++.


