Effective Architectures for Low Resource Multilingual Named Entity Transliteration

Molly Moran
Brandeis University
mollymoran@brandeis.edu

Constantine Lignos
Brandeis University
lignos@brandeis.edu

Abstract

In this paper, we evaluate LSTM, biLSTM, GRU, and Transformer architectures for the task of name transliteration in a many-to-one multilingual paradigm, transliterating from 590 languages to English. We experiment with different encoder-decoder combinations and evaluate them using accuracy, character error rate, and an F-measure based on longest continuous subsequences. We find that using a Transformer for the encoder and decoder performs best, improving accuracy by over 4 points compared to previous work. We explore whether manipulating the source text by adding macrolanguage flag tokens or pre-romanizing source strings can improve performance and find that neither manipulation has a positive effect. Finally, we analyze performance differences between the LSTM and Transformer encoders when using a Transformer decoder and find that the Transformer encoder is better able to handle insertions and substitutions when transliterating.

1 Introduction

The world's written languages collectively represent hundreds of different writing systems. Transliteration is the process of converting a word's written representation in one language to its equivalent in a target language and is a key component of machine translation and cross-lingual information extraction and retrieval. It is especially relevant for recovering named entities, which often cannot be "translated" in the traditional sense, and linking names across texts in different languages.

In this work, we apply existing architectures used for machine translation to the task of name transliteration, the task of converting the written citation form of a name in one language to another language. We use an existing massively-multilingual resource of aligned parallel names in 591 languages (Wu et al., 2018) and evaluate four neural architectures for transliteration. While the use of neural machine translation architectures for transliteration is not novel, to the best of our knowledge a comparative study of the performance of multiple models and preprocessing strategies in the many-to-one transliteration paradigm has not been previously performed.

The contributions of this paper are as follows: first, we provide a performance comparison of several different neural architectures on the task of name transliteration in a many-to-one paradigm. Second, we evaluate the effectiveness of various methods of manipulating input sequences for name transliteration. Finally, we present a multilingual transliteration model with a 1-Best accuracy of 73%, a 4-point improvement over the previous best model for this task.

2 Related Work

The 2018 Named Entities Workshop included a shared task on Named Entity Transliteration (Chen et al., 2018) that established several useful metrics to evaluate various elements of transliteration quality. We use their mean F-score metric in this work. As part of the shared task, Kundu et al. (2018) compared RNN and CNN architectures on the task of name transliteration for 13 language pairs. They found that their character-level RNN model performed best and achieved state-of-the-art results for one language pair. Notably, the authors focused on only a few higher-resource languages, whereas our study uses a much larger set of languages, including many lower-resourced ones.

Merhav and Ash (2018) released a set of bilingual name dictionaries mined from Wikipedia for English to Russian, Hebrew, Arabic, and Japanese Katakana transliteration, and compare traditional weighted FST models with more modern neural techniques on bilingual transliteration tasks. Their

Proceedings of the 3rd Workshop on Technologies for MT of Low Resource Languages, pages 79–86

models collapse all names written in Latin scripts under the English label. But very different languages may share a script; we aim to create models sensitive to these differences. Benites et al. (2020) recently released a much more comprehensive name corpus covering 3 million names across 180 languages derived from Wikipedia, GeoNames, and the dataset released by Merhav and Ash (2018).

Regarding lower-resourced transliteration, Upadhyay et al. (2018) proposed a bootstrapping method wherein a "weak" generation model is used to guide discovery of possible transliteration candidates for a given word. Le and Sadat (2018) used a combination of G2P, neural networks and word embeddings to improve English-Vietnamese transliteration.

Johnson et al. (2017) introduced a now-popular technique for invoking transfer learning in a many-to-one multilingual translation framework. A single model is trained with a mixed source vocabulary and a single target vocabulary. An artificial "flag" token is appended to each source sequence to help the model identify languages, circumventing the need for training a separate encoder and decoder for each language pair. We use this approach to perform many-languages-to-one transliteration.

Wu and Yarowsky (2018) explored approaches to extremely low-resource name transliteration, using a multiparallel corpus of Biblical names across 591 languages (Wu et al., 2018). The authors also provided a cursory report on the results of an experimental multilingual transliteration model, using the technique proposed by Johnson (Johnson et al., 2017). They found that a character-level RNN trained on concatenated data from all source languages significantly outperformed individual language-pair models. Using the same data, we aim to provide a more comprehensive study of the performance of various neural architectures and the applicability of potential performance-boosting preprocessing techniques for the task of transliteration in a many-to-one framework.

3 Experiment Design

3.1 Data

The corpus we use to train our models (Wu et al., 2018) represents 591 languages with varying numbers of alignments to 1129 English names. After removing any blank names, the final dataset comprised 348,991 name pairs. We partitioned the data using the exact train/development/test split (80/10/10) and random seed provided by Wu and Yarowsky (2018), which enables us to directly compare our results with the values they report. Our resulting training set consisted of 279,192 name pairs. The mean number of training pairs per language was 472, the median was 454, and the minimum and maximum were 34 and 938.

3.2 Models

Our models are implemented using OpenNMT version 1.0.2 (Klein et al., 2017). We experimented with several different combinations of encoders and decoders. Our baseline architecture is the model used by Wu and Yarowsky, a GRU encoder paired with the default decoder, using their hyperparameters as our defaults. We configured three additional encoders: an LSTM, a biLSTM, and a Transformer. We paired the GRU, LSTM and biLSTM encoders each with two decoders, a Transformer and OpenNMT's default LSTM decoder with attention. We paired the Transformer encoder with a Transformer decoder; for brevity we will refer to this as our Transformer model without separately mentioning the encoder and decoder. After testing each model on a single random seed with embedding sizes of 200, 300 and 400—using the same size for both source and target—we chose the size with the lowest development set perplexity (in all cases, 200).

The LSTM, biLSTM and GRU encoders are 2-layer models with hidden size 200, trained for 44k steps, the same hyperparameters used by Wu and Yarowsky 2018.[1] The Transformer encoder is a 4-layer model with 8-headed self-attention, sinusoidal positional encoding with clipping distance of 2, and hidden feed-forward size of 1024. We tested several positional encoding clipping distances, choosing the value that produced the lowest development set perplexity. Based on inspecting development set perplexity, we increased the Transformer encoder's training duration to 90k updates, with 10k warmup steps and a learning rate decay interval of 10k steps.

OpenNMT's default decoder is a 2-layer LSTM RNN of size 500 with attention that implements input-feeding (see Luong et al. 2015), wherein attention vectors are fed as input to the next time step. The hidden size of each layer is 200. The Transformer decoder is identical to our Transformer encoder—4 layers with 8-headed self-attention—

[1] In the case of the biLSTM encoder, each direction has 100 hidden units for a total of 200. We experimented with doubling the total number of hidden units so that each direction was size 200, but this increase did not improve performance.

Encoder	Mean F-score	Accuracy	CER
GRU	92.27 ± .07	68.66 ± .25	17.53 ± .23
LSTM	**92.94** ± .13	**71.14** ± .34	**16.10** ± .44
biLSTM	**92.93** ± .09	**71.13** ± .20	**16.05** ± .23

Table 1: Mean and standard deviation of all metrics for each encoder using the default (LSTM) decoder

Encoder	Mean F-score	Accuracy	CER
GRU	92.87 ± .07	70.78 ± .25	16.15 ± .30
LSTM	93.08 ± .09	71.49 ± .32	15.65 ± .36
biLSTM	92.99 ± .19	71.37 ± .28	15.82 ± .35
Trans.	**93.48** ± .03	**73.01** ± .10	**14.73** ± .08

Table 2: Mean and standard deviation of all metrics for each encoder using the Transformer decoder

except it has a hidden size of 2048.

All models were trained using ADADELTA, with dropout of 0.2 and a batch size of 64. Training was performed on a single computer with an AMD 2990WX CPU, two RTX 2080 Ti GPUs, and one Titan RTX GPU. Training on a single RTX 2080 Ti for the GRU (1,818,431 parameters with the default decoder, 3,103,631 with a Transformer decoder) and LSTM (2,180,031 parameters with the default decoder, 3,545,727 with a Transformer decoder) models took approximately 26 minutes. The biLSTM model (2,020,031 parameters with the default decoder, 3,385,727 with a Transformer decoder) took approximately 29 minutes. Transformer training (5,839,623 parameters) took approximately 98 minutes.

3.3 Metrics

We report three separate evaluation metrics for each model: 1-best accuracy, the percentage of perfectly transliterated names; character error rate (CER), the error rate of characters across names as calculated using Levenshtein distance; and mean F-score or "Fuzziness in Top-1," a metric introduced by the NEWS 2018 Shared Task on Named Entity Transliteration (Chen et al., 2018). Mean F-score computes the character-level F1-score based on the longest common subsequence between a source and target sequence.

3.4 Evaluation Procedure

Each experimental configuration for each model architecture was run 10 times using 10 different random seeds. Checkpoints were saved every 5k updates, and the checkpoint with the lowest development set perplexity was used for evaluation. For a given experimental configuration we report the mean and standard deviation for each evaluation metric taken across all random seeds.

4 Results

Figure 1 displays box plots for mean F-score for each configuration of each architecture. Each box's whiskers give the minimum and maximum scores; each box gives the 25th, 50th, and 75th percentiles. As the plots for the other metrics look extremely similar, we provide plots for accuracy (Figure 4) and CER (Figure 5) at the end of the paper. Table 1 gives results for all architectures using the default decoder, and Table 2 gives results for all architectures using the Transformer decoder.

As many of the results are similar—especially the LSTM and biLSTM—we performed statistical significance tests and bolded the highest score and the scores that were not found to be significantly different at the $p < 0.05$ level from the highest score. We use Dunn's test, applying Bonferroni correction to a Kruskal-Wallis test, which is similar to Mann-Whitney U. In short, this approach tests differences between median scores without making any assumptions that the data come from a normal distribution, and ti adjusts for the fact that we are comparing more than two scores. While we follow the convention of reporting the mean and standard deviation in tables, neither the mean nor standard deviation is used by our non-parametric statistical significance testing procedure.

As Table 1 shows, when combined with the default decoder the LSTM and biLSTM encoders perform almost identically (comparing using each metric, all comparisons $p > 0.05$), and outperform the GRU (all comparisons $p < 0.001$).

As Table 2 shows, using a Transformer decoder generally leads to improved performance compared to the default decoder. Among the encoders evaluated, the Transformer performs best, and the differences are statistically significant when compared to each of the other encoders and across all metrics (all comparisons $p < 0.05$).

As Figure 1 shows, there is no overlap between the minimum and maximum mean F-score values attained by the Transformer (when used as both encoder and decoder) and other architectures. In addition to achieving the highest performance across all metrics, the Transformer's variation across random

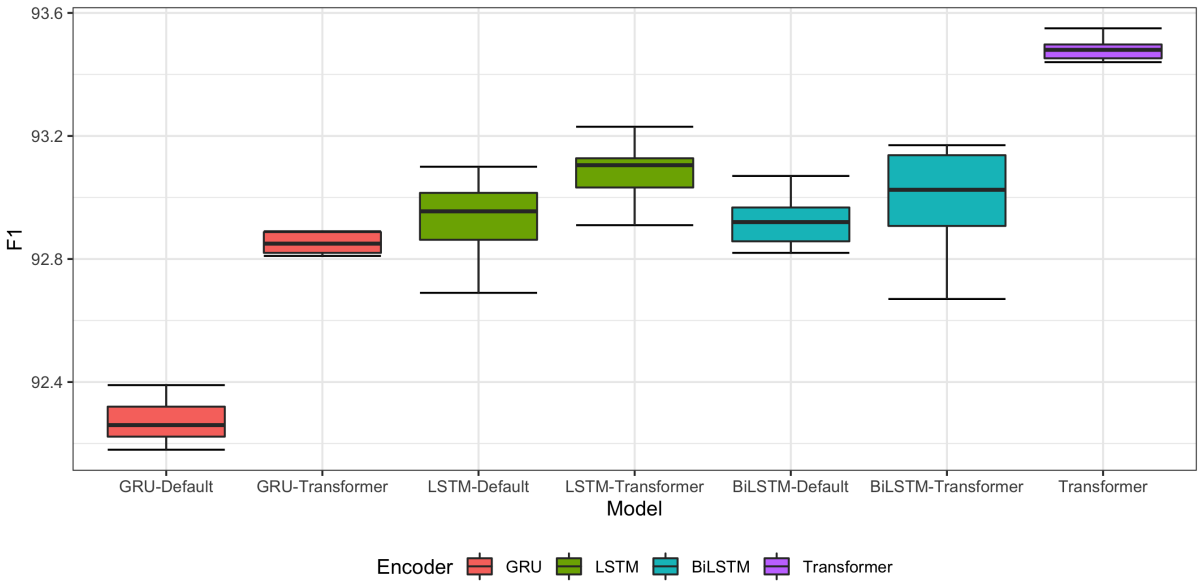

Figure 1: Mean F-score across encoder and decoder architectures

Format	Source	Target
Standard	<aln> e m a n u e l	i m m a n u e l
Macrolang.	<aln> <sqi> e m a n u e l	i m m a n u e l

Table 3: Source preprocessing with ISO-639-3 language (<aln>) and macrolanguage (<sqi>) codes

seeds is smaller than other architectures, both in the range between the minimum and maximum values and the standard deviation.

5 Extensions

5.1 Artificial Tokens

Johnson et al. (2017) used ISO-639-3 language codes as flag tokens. To our knowledge, subsequent implementations of multilingual translation have all used similar codes, including Wu and Yarowsky (2018)'s transliteration model and our own models. As these language codes are arbitrary, atomic flags, they do not encode known relationships between languages. The ISO-639 schema designates 58 languages as *macrolanguages*, which unite groups of closely-related languages that may exist on a dialect continuum.[2] Of the 7,868 individual languages identified by ISO-639-3, 453 (5.8%) have a corresponding macrolanguage, and those that do tend to be lower-resourced languages. The Biblical names corpus compiled by Wu et al. (2018) mirrors these statistics; of the 591 languages, 36 (6.1%) have a corresponding macrolanguage code. These languages comprised 6.8% of the training set (19,040 name pairs) and 6.9% of the test set (2,432 pairs).

We can leverage this information in our models by appending an additional macrolanguage flag token following a language's ISO-639-3 code, where applicable, as shown in Table 3. We expected that this may lead to small improvements in transliteration quality for lower-resourced languages that belong to the same macrolanguage. We tested this by evaluating using our Transformer model.

As Table 4 shows, the addition of macrolanguage tokens actually slightly hurt the models' performance on every metric. The 36 languages in our dataset with a corresponding macrolanguage shared 24 macrolanguages, making it difficult for a model to leverage cross-lingual similarities. We suspect that macrolanguage information ultimately amounted to noise.

We also evaluated on only the subset of languages with a corresponding macrolanguage, to see whether the addition of macrolanguage tokens improved performance for them. However, performance using macrolanguage tokens (Mean F-score 92.04 $\pm$ .08; Accuracy 69.65 $\pm$ .26; CER 18.29 $\pm$ .18) decreased from the baseline (93.35 $\pm$ .08; 72.33 $\pm$ .38; 14.79 $\pm$ .2).

5.2 Romanization

In analyzing per-language performance, we found the expected relationship between the number of

[2] https://iso639-3.sil.org/code_tables/macrolanguage_mappings/read

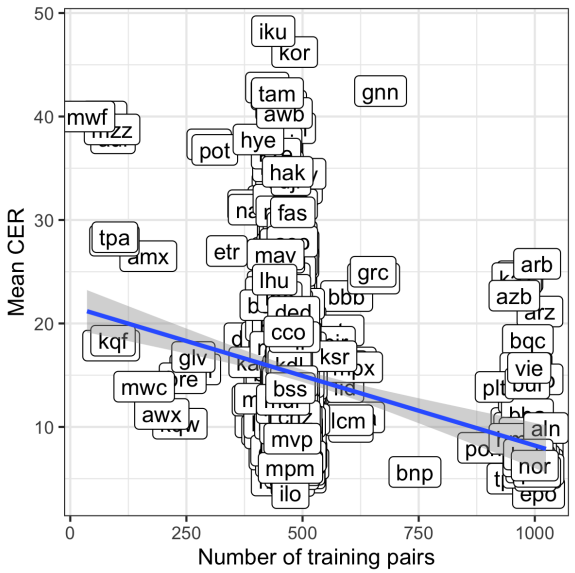

Figure 2: Per-language mean character error rate and number of training pairs, with linear fit line

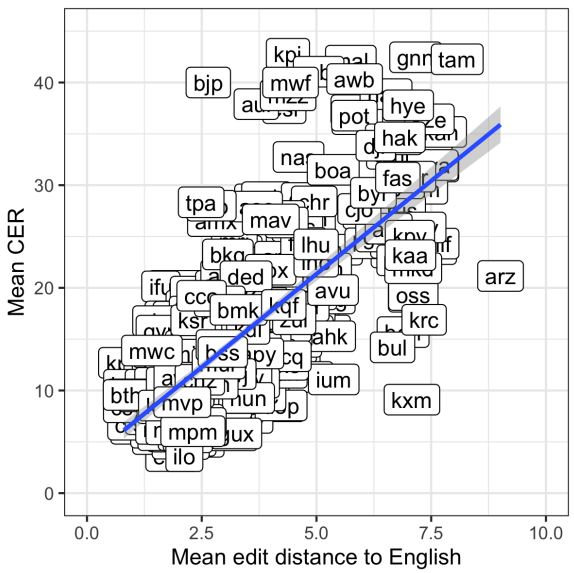

Figure 3: Per-language mean character error rate and mean edit distance from English, with linear fit line

training pairs in the language and the performance in that language, as shown in Figure 2 for character error rate. However, there is a lot of variation across languages, even among those with scripts very different from English, which is suggestive of successful transfer learning across languages. Many languages with fewer examples are doing better than expected, reducing the correlation between per-language training data size and performance. The largest cluster of number of training pairs per language is approximately 500 languages with around 500 name pairs each. Even within that cluster there is tremendous variance in performance around that number of training examples. Iloko (ilo), the best-performing language in that cluster, has a CER of 3.6, while the worst-performing language, Inuktitut (iku), has a CER of 48.2. There is a nearly uniform distribution of languages between those extremes.[3]

As previously identified by Wu and Yarowsky (2018), transliteration is more successful when the edit distance between source and target is low. Figure 3 shows the mean CER, along with the mean edit distance to English, for each language, using a single run of the Transformer model with standard preprocessing. One possible approach to reducing edit distance is to romanize the data before transliteration. Wu and Yarowsky (2018) report that

[3]For this example and in Figures 2 and 3 we exclude three extremely poorly-performing languages (cmn, mya, khm, each with CER > 75) where the poor performance can be traced to errors in the training data which aligned names to phrases.

Condition	Mean F-score	Accuracy	CER
Standard	**93.48** $\pm$ .03	**73.01** $\pm$ .10	**14.73** $\pm$ .08
Macrolang.	93.45 $\pm$ .04	**72.90** $\pm$ .13	**14.75** $\pm$ .11
Romanized	93.40 $\pm$ .03	**72.96** $\pm$ .14	14.86 $\pm$ .07

Table 4: Mean and standard deviation of all metrics for the Transformer encoder and decoder for standard, macrolanguage-marked, and romanized source text

preprocessing data into ASCII using Unidecode yielded no improvement in performance. However, their approach is extremely simple; we evaluate a more sophisticated, hand-tuned romanizer, uroman (Hermjakob et al., 2018). We romanized data before providing it as source text, and trained the Transformer using standard preprocessing. As shown in Table 4, this slightly reduces accuracy and mean F-score, and slightly increases CER. Further analysis revealed that romanization did not actually decrease edit distance to English; it increased it for as many languages as it decreased it, likely because it often lengthened words.

5.3 Summary

As shown in Table 4, neither the use of macrolanguage tokens nor romanization improves performance. As the bolding indicates, for mean F-score the standard condition performs significantly better than macrolanguage ($p = 0.03$) and romanized ($p = 0.0007$); for CER, standard performs better than romanized ($p = 0.008$). The results for the accuracy metric are statistically indistinguishable

(all comparisons $p > 0.05$); all values are bolded as it is essentially an all-way tie.

6 Discussion

Our findings demonstrate that using a Transformer architecture for the encoder and decoder results in reliably strong performance for this task. Attempting to improve the model through simple "tweaks" to the input does not improve performance and may hurt it. However, one may wonder why the Transformer does so much better at this task than an LSTM encoder and a decoder with attention when the transliteration problem is relatively simple compared to sentence translation. There is only limited reordering in transliteration, and there may be fewer long-range dependencies.

While we have manually examined the output of the transliteration system across architectures, it is difficult to identify obvious patterns, and attempting to do so puts us at risk of overgeneralizing our observations that are based on the relatively few writing systems that we can read.

To further analyze performance, for every item in the test set we computed the Levenshtein distance—expressed as insertions, deletions, and substitutions—between the source name and target name. We then analyzed the relationship between these edit distance metrics and 1-best accuracy by predicting whether the system correctly transliterated each item of the test set using logistic regression. We analyze two systems: the LSTM and Transformer encoders, each paired with the Transformer decoder.[4]

We examined the contribution of each type of edit (insertions, etc.) on the performance of each system. We fit a logistic regression model to the 689,960 predictions across all items and random seeds for the models we are comparing. For each item, we used the number of source-target edit distance operations to predict whether the model made any errors. We employed interactions between the encoder type and each predictor to explicitly test for differences between the LSTM and Transformer encoders. In other words, our model tested whether the two encoders differed in how each edit operation required affected their ability to correctly transliterate.

[4]We exclude pairs from the languages with codes cmn, mya, khm, and nab from this analysis after review of the training data revealed that edit distance between source and target names was artificially high due to data problems.

We found that there were significant differences in the interaction terms of the model for insertions and substitutions, but not for deletions. For the Transformer encoder, with each additional insertion in the edit distance, it was 1.1% less likely to produce an error than the LSTM was ($p = 0.0006$), for substitutions, 1.8% ($p < 0.0001$). This helps further characterize the performance differences between these models. While they are equally capable of handling deletions, the Transformer encoder can better handle insertions and substitutions, and the advantage is larger for substitutions.

7 Conclusion

We conclude that using a Transformer architecture for both the encoder and decoder leads to the best performance on the many-languages-to-English transliteration task that we evaluate. However, using macrolanguage codes and pre-romanizing the input do not improve performance.

Our best multilingual transliteration model achieves a 1-best accuracy of 73%, a 4-point improvement over the baseline provided by Wu and Yarowsky (2018). When using a Transformer encoder, our model demonstrates an improved ability to handle substitution edits in source-target pairs compared to an LSTM encoder.

While an off-the-shelf MT system provides a strong starting system for the task of many-to-one transliteration, future improvements for lower-resourced settings will likely require a greater level of sophistication, possibly using monolingual pretraining to better model the source language given few training examples. Additionally, while using the Transformer for the encoder and decoder gives the strongest results, it may be possible to further simplify the model and achieve similar results.

References

Fernando Benites, Gilbert François Duivesteijn, Pius von Däniken, and Mark Cieliebak. 2020. Large name transliteration resource. In *Proceedings of the Thirteenth International Conference on Language Resources and Evaluation (LREC 2020)*.

Nancy Chen, Xiangyu Duan, Min Zhang, Rafael E. Banchs, and Haizhou Li. 2018. News 2018 whitepaper. In *Proceedings of the Seventh Named Entities Workshop*, pages 47–54, Melbourne, Australia. Association for Computational Linguistics.

Ulf Hermjakob, Jonathan May, and Kevin Knight. 2018. Out-of-the-box universal Romanization tool

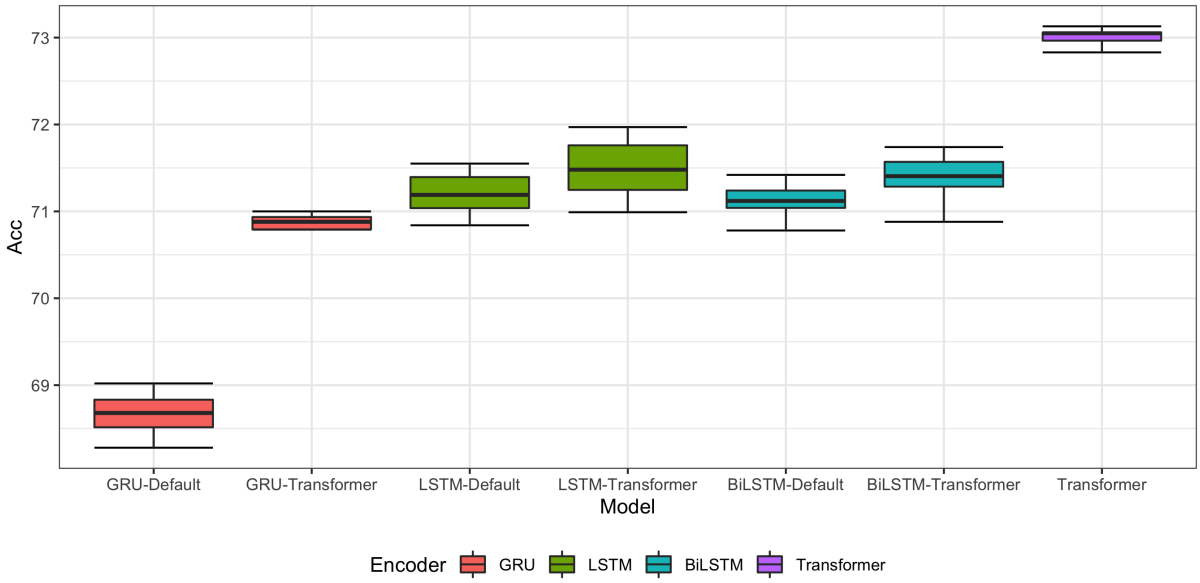

Figure 4: Accuracy across encoder and decoder architectures

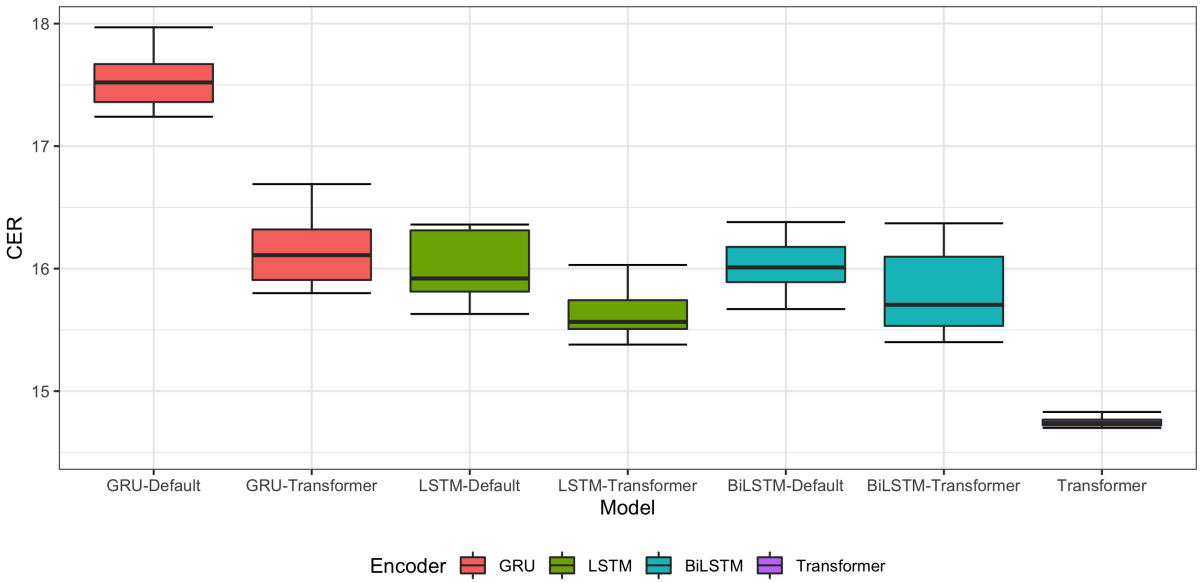

Figure 5: CER (lower is better) across encoder and decoder architectures

uroman. In *Proceedings of ACL 2018, System Demonstrations*, pages 13–18, Melbourne, Australia. Association for Computational Linguistics.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada. Association for Computational Linguistics.

Soumyadeep Kundu, Sayantan Paul, and Santanu Pal. 2018. A deep learning based approach to transliteration. In *Proceedings of the Seventh Named Entities Workshop*, pages 79–83, Melbourne, Australia. Association for Computational Linguistics.

Ngoc Tan Le and Fatiha Sadat. 2018. Low-resource machine transliteration using recurrent neural networks of Asian languages. In *Proceedings of the Seventh Named Entities Workshop*, pages 95–100, Melbourne, Australia. Association for Computational Linguistics.

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

Yuval Merhav and Stephen Ash. 2018. Design challenges in named entity transliteration. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 630–640, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Shyam Upadhyay, Jordan Kodner, and Dan Roth. 2018. Bootstrapping transliteration with constrained discovery for low-resource languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 501–511, Brussels, Belgium. Association for Computational Linguistics.

Winston Wu, Nidhi Vyas, and David Yarowsky. 2018. Creating a translation matrix of the Bible's names across 591 languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Winston Wu and David Yarowsky. 2018. A comparative study of extremely low-resource transliteration of the world's languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).