Context-aware Transliteration of Romanized South Asian Languages

Christo Kirov
Google Research
ckirov@google.com

Cibu Johny
Google Research
cibu@google.com

Anna Katanova
Google Research
akatanova@google.com

Alexander Gutkin
Google Research
agutkin@google.com

Brian Roark
Google Research
roark@google.com

While most transliteration research is focused on single tokens such as named entities—for example, transliteration of "અમદાવાદ" from the Gujarati script to the Latin script "Ahmedabad"[1]*—the informal romanization prevalent in South Asia and elsewhere often requires transliteration of full sentences. The lack of large parallel text collections of full sentence (as opposed to single word) transliterations necessitates incorporation of contextual information into transliteration via non-parallel resources, such as via mono-script text collections. In this article, we present a number of methods for improving transliteration in context for such a use scenario. Some of these methods in fact improve performance without making use of sentential context, allowing for better quantification of the degree to which contextual information in particular is responsible for system improvements. Our final systems, which ultimately rely upon ensembles including large pretrained language models fine-tuned on simulated parallel data, yield substantial improvements over the best previously reported results for full sentence transliteration from Latin*

1 The most populous city in the Indian state of Gujarat.

Action Editor: Kevin Duh. Submission received: 19 April 2023; revised version received: 20 September 2023; accepted for publication: 31 October 2023.

https://doi.org/10.1162/coli_a_00510

to native script on all 12 languages in the Dakshina dataset (Roark et al. 2020), with an overall 3.3% absolute (18.6% relative) mean word-error rate reduction.

1. Introduction

Transliteration has long been a topic of interest in natural language processing (NLP), yet the primary use case has generally been within machine translation or information retrieval, for processing names and technical terms, which are typically transliterated between scripts rather than translated (Knight and Graehl 1998; Moran and Lignos 2020). More recently, some use scenarios have emerged that require transliteration of *full sentences*—for example, languages that are written in two different native scripts (such as Punjabi written in both Gurmukhi, a Brahmic script, and Shahmukhi, a Perso-Arabic script [Murphy 2018]), or that are also written informally in the Latin script, which is known as *romanization* (Wellisch 1978). For example, multilingual speech recognition systems for languages with diverse writing systems can be trained by converting training data transcripts to a common script (typically the Latin script) to improve cross-lingual generalization; then the recognizer output in the common script can be transliterated back to the specific language's native script (e.g., Datta et al. 2020). Similarly, for many languages, mobile keyboard entry can be substantially easier in the Latin script (e.g., via a QWERTY layout) than in their native script, yet the output of text entry is preferred to be in the native script, thus requiring transliteration (as in Hellsten et al. 2017). In these and related scenarios, rather than isolated words or proper names, full sentences are transliterated from one script to another.

Unlike translations, full sentence parallel transliterated text is relatively rare, hence direct application of large-scale full-sentence (i.e., context-aware) sequence-to-sequence modeling is not generally an option. In this article, we present a demonstration of how important contextual information is for this task, as well as exploring several methods for jointly improving the accuracy of full sentence transliteration.

South Asian languages such as Hindi, Tamil, and Urdu are often written informally in the Latin script, despite having official writing systems based on Brahmic or Perso-Arabic scripts (Gella, Bali, and Choudhury 2014; Mhaiskar 2015; Sodhar et al. 2019). Without a standard orthography in the Latin script, romanized text in these languages contains an extensive degree of spelling variation, hence transliteration to their native scripts can be challenging (Irvine, Weese, and Callison-Burch 2012; Riyadh and Kondrak 2019; Choksi 2020). For example, the Tamil word புளி (tamarind) is sometimes romanized as *puli* but also (less frequently) as *pulli*; *puli* is also an attested romanization for புலி (tiger). These examples were taken from the Tamil romanization lexicon in the Dakshina dataset[2] (Roark et al. 2020), a dataset that also contains, among other things, full-sentence parallel romanized/native-script text in 12 South Asian languages. The kind of romanization variation mentioned above is observed in this data—the average number of distinct romanizations per word that occur more than once in each of the 12 languages' development sets[3] are shown in Table 1.

The full-sentence parallel data in the dataset is insufficient to train large-scale sequence-to-sequence models directly, but it does permit full sentence transliteration system development and validation. The dataset additionally includes isolated word transliteration dictionaries, such as the Tamil one mentioned above, which can be used

2 https://github.com/google-research-datasets/dakshina.
3 Please see Section 3.1 for the details on these language codes.

Table 1
Number of distinct romanizations used for words that occur more than once in the Dakshina full sentence romanized development set.

Language:	bn	gu	hi	kn	ml	mr	pa	sd	si	ta	te	ur
Romanizations per word:	1.9	1.9	1.9	1.6	1.8	1.5	2.2	2.3	1.7	1.9	1.7	2.0

to train non-contextual single word transliteration models; as well as native script text samples for training language models. Using that data, Roark et al. (2020) provide baselines for a number of tasks, including full-sentence context-aware transliteration from romanized text to native scripts. Their context-aware methods dramatically outperform non-contextual alternatives, demonstrating context's importance for the task.

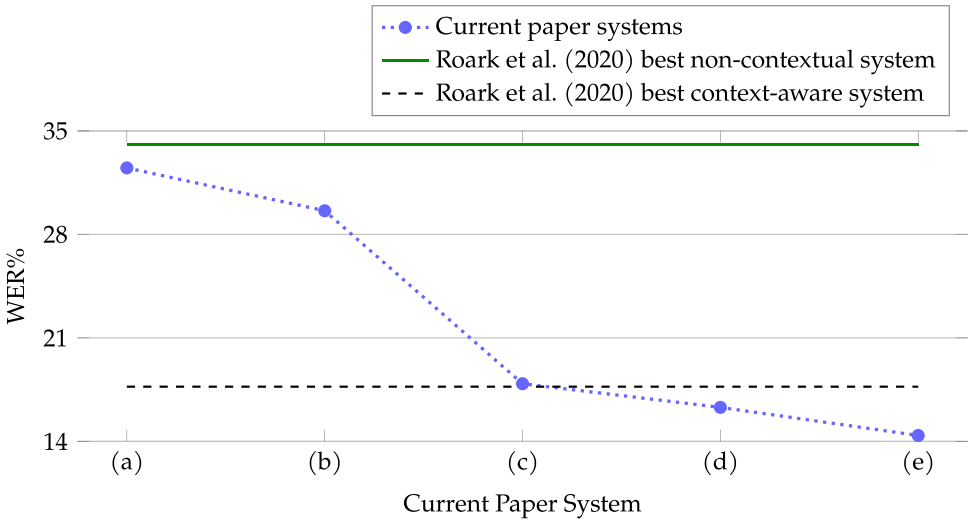
In the interests of clarity, let us explicitly establish some terminology that we have been using. We have been distinguishing between the tasks of full-sentence transliteration and single (isolated) word transliteration, where full-sentence transliteration involves transliterating an entire given sentence versus just a single given word.[4] Throughout the paper, we will use the terms "full-sentence" and "single word" transliteration to differentiate these distinct tasks. We will label models used for full-sentence transliteration as either "context-aware" or "non-contextual." "Context-aware" models take into account the surrounding words in the sentence while transliterating, whereas "non-contextual" models transliterate based only on word-specific characteristics, that is, they treat each word in the sentence independently as a single word transliteration task.

In this article, we examine the question of exactly how important context is for full-sentence transliteration by establishing new non-contextual model baselines that ultimately nearly match the error rates of even the context-aware results reported in Roark et al. (2020). This is achieved through several methods, including model ensembling and non-contextual (i.e., word-internal) language modeling. These improved non-contextual results suggest an efficient distributed algorithm for transliterating a sentence for use in scenarios when efficiency is paramount. In a controlled setting, we demonstrate a roughly three times speedup using non-contextual modeling versus our fastest contextual modeling setup, at the cost of less than 10% relative word-error rate.

Unlike Roark et al. (2020), in this article we also additionally explicitly focus on automatic romanization, that is, transliteration from the native script to the Latin script, primarily as the means of simulating data to fine-tune large language models. In the absence of an orthography for these languages in the Latin script, assessment of the quality of the romanization is tricky, and we present new methods for assessing *k*-best system outputs given a list of attested romanizations. Our best automatic romanization systems are then used to produce simulated parallel training data, which are used to fine-tune models that contribute to our best performing context-aware transliteration systems.

We explore several methods for context-aware modeling in transliteration, including (1) combining single word (non-contextual) transliteration ensembles with native script language models during decoding; and (2) using single word (non-contextual) romanization models to simulate full sentence parallel resources for fine-tuning large

4 Note that we use the term "sentence" to denote any multi-word string, whether or not it grammatically corresponds to a sentence.



System	Description
(a)	The best single method non-contextual transliteration model ensemble.
(b)	The best multiple method non-contextual transliteration model ensemble.
(c)	The best multiple method non-contextual transliteration model ensemble combined with word-level (non-contextual) language models.
(d)	The best multiple method non-contextual transliteration model ensemble combined with sentence-level (context-aware) language models.
(e)	The ensemble of multiple context-aware transliteration systems.

Figure 1
Macro-averaged WER% (lower-is-better) for various full sentence Latin-to-native script transliteration systems, on the development (`dev`) set of the Dakshina dataset. Results include the best performing non-contextual and context-aware system results in Roark et al. (2020), and five system configurations from this current paper, as described above.

pretrained language models. Ultimately, the best results are achieved by ensembling these distinct approaches, yielding an additional 3.5% absolute word-error rate (WER%) reduction versus the above-mentioned non-contextual models. We additionally explore an online version that achieves speedups at the expense of a small accuracy reduction by use of an offline assembled cache, and provide post-hoc analyses regarding, among other things, the contribution of context and ensembling to transliteration accuracy.

To illustrate the main findings of our article, we foreshadow and summarize the extensive results from Section 4 by plotting (in Figure 1) the full sentence Latin-to-native script WER% (macro-averaged across the development sets of the 12 Dakshina languages) achieved[5] for five key system configurations, alongside the best non-contextual and context-aware results presented by Roark et al. (2020). Our best single method non-contextual model—system (a) in the graph—yields slightly better WER% than the best such result presented in Roark et al. (2020). This is at least partially due to the fact that, for each method, we ensemble five model instances trained using a different random initialization, which improves accuracy and lowers variance. Ensembling

5 Please see Section 4 for full experimental details. For plot interpretation it suffices to know that lower WER% is better.

multiple non-contextual transliteration models together—system (b)—yields further improvements. Large further reductions are then achieved by system (c), which combines non-contextual (word-level) language models with ensembles from system (b), resulting in performance commensurate with the best context-aware system in Roark et al. (2020). This represents 80% of the relative error rate reduction achieved between system (b) and system (e).

Using sentence-level instead of word-level language models—system (d), our first context-aware system—yields further gains, as does our final ensemble of multiple context-aware methods, system (e). This last system achieves substantial reductions over previously reported results, and, due to our method of comparing these varied system configurations, we are able to relatively finely allocate the credit for these reductions to a number of different methods.

Note that the data points in Figure 1 are averaged across many languages and trials, and the ensembling methods that provide the gains observed by systems (a), (b), and (e) yield reductions not only in error rate but also in variance between training runs, as will be shown in detail in the experiments presented in Section 4. These experiments are controlled to allow for measurement of multi-run variance (even of ensembles), and one result of this is a clear demonstration that, along with solid error-rate reductions, ensembling provides critical variance reduction.

One key contribution of the article is demonstrating that inclusion of non-contextual language modeling (i.e., word-level wordpiece models capturing only word-form likelihood) dramatically decreases word-error rates compared to using non-contextual transliteration models alone, thus reducing how much of the cumulative final improvement is attributable to context. In contrast, Roark et al. (2020) present a near-halving of WER% due to context (34.1% to 17.7%), which would suggest that context-aware information is absolutely indispensable for effective full sentence transliteration. Our results suggest otherwise. Another contribution of the article worth noting is the demonstration of online cache-driven full sentence transliteration, permitting both non-contextual (hence highly parallelizable) and context-aware inference, with modest accuracy loss for items falling outside of cache coverage. Our fast non-contextual online transliteration system achieved 18.4% macro-averaged WER% over the set of languages, just 0.7% higher than the above mentioned best context-aware system from Roark et al. (2020).

The key contributions of this article additionally include new methods for preparing training data to fine-tune pre-trained models, which control for tricky outliers/annotation errors. We clearly demonstrate the essential role of model ensembling, both to improve overall accuracy and reduce variance, both for non-contextual and context-aware models, indicating that ensembling is no longer just the means to achieve modest additional system improvements, but is indispensable for stable and accurate systems. We also provide further evidence that contextual information can be profitably incorporated into transliteration systems even in the absence of *any* given full sentence parallel data—not as critical as Roark et al. (2020) suggest, but still yielding up to 20% relative error-rate reduction. Along the way, we provide detailed analyses/examples of where context-awareness and ensembling are beneficial to the systems, to supplement the extensive experimental results across the 12 languages in the Dakshina dataset. All processed data and code required to reproduce these results are publicly released.[6]

6 https://github.com/google-research/google-research/tree/master/context_aware_transliteration.

2. Background and Related Work

2.1 Monotonic Sequence-to-sequence Modeling

Transliteration is a monotonic sequence-to-sequence task, that is, it involves taking a sequence as input, producing a sequence as output, and the output sequence can be monotonically aligned to the input sequence. For example, automatic speech recognition (ASR, also known as speech-to-text, STT) involves taking an acoustic waveform as input and producing a text transcription of the speech as output (Yu and Deng 2015). Words in the transcription can be aligned with temporal spans in the input speech in a way that preserves the ordering of the text, which is what makes it monotonic. In contrast, speech-to-speech translation is a sequence-to-sequence task that does not preserve the ordering, since translation typically involves some reordering of information from the input sequence in the output sequence (Jia et al. 2019). Going the other way, text-to-speech is also generally a monotonic sequence-to-sequence task, since the generated speech maintains the order of the input text (Taylor 2009).

Considering NLP tasks that operate on input text and produce output text, various kinds of tagging tasks, such as part-of-speech tagging (Voutilainen 2003), are typically modeled as monotonic sequence-to-sequence tasks. Textual transformations in service of other monotonic tasks—such as text normalization (Zhang et al. 2019), or grapheme-to-phoneme (g2p) conversion (Deri and Knight 2016), both of which can be useful for text-to-speech—also have this characteristic.

Transliteration has much in common with tasks such as text normalization or g2p in that it simply changes the textual representation of the same linguistic information. Text normalization may convert a written representation of a numerical value, for example, $\frac{2}{3}$, into how it would be spoken ("two thirds"), while g2p would convert that Latin script representation of the words into IPA or another representation of the pronunciation. One important characteristic of these tasks—including transliteration—is that they are fundamentally designed to preserve the linguistic content of the input, that is, words in the input should not be deleted nor should words that do not appear in the input be added to the output. Segmentation of tokens may differ—for example, single whitespace-delimited words may correspond to multiple tokens in a different script, such as the single token "الجزيرة" in Arabic, which is transliterated in the Latin script conventionally as "Al Jazeera." Modulo these sorts of orthographic conventions, however, the linguistic content of the output should be the same as the input.

Monotonic sequence-to-sequence tasks have been around for a long time, and have been addressed using methods such as hidden Markov models (Baum and Petrie 1966; Jelinek, Bahl, and Mercer 1975; Rabiner 1989), Markov random field approaches (e.g., conditional random fields [Lafferty, McCallum, and Pereira 2001]), and, of course, through various neural architectures, such as recurrent neural networks, including long short-term memory (LSTMs) (Hochreiter and Schmidhuber 1997), convolutional networks (Chae et al. 2018), or transformers (Vaswani et al. 2017). In this article, we make use of LSTM and transformer models of various sorts (including pretrained large language models, LLMs), as well as an approach that combines a transliteration model with a language model to obtain contextual influence. This latter approach has similarities to so-called noisy channel models, such as hidden Markov models, though without the same graphical model structure. Even so, we follow Roark et al. (2020) in labeling such approaches as "noisy channel" to distinguish them from the end-to-end neural alternatives.

2.2 Transliteration and Romanization Models and Corpora

As mentioned above, transliteration is akin to g2p conversion, in that it preserves linguistic content and is generally monotonic. Common methods for g2p, such as the finite-state joint multigram models (Galescu and Allen 2001; Bisani and Ney 2002; Chen 2003; Bisani and Ney 2008), also known as pair *n*-gram models, are hence also directly applicable for transliteration (Finch and Sumita 2010; Jiampojamarn, Cherry, and Kondrak 2010; Hellsten et al. 2017),[7] as are general neural sequence-to-sequence models (Kunchukuttan et al. 2018; Merhav and Ash 2018; Kundu, Paul, and Pal 2018; Gow-Smith et al. 2022; Wu et al. 2022). Given the many distinct modeling options for the task, system combination or ensembling methods have naturally also been investigated (Nicolai et al. 2015; Najafi et al. 2018).

Transliteration in NLP has generally been focused on named entities or other specialized vocabulary in the context of machine translation or information retrieval (Knight and Graehl 1998; Chen et al. 1998; Virga and Khudanpur 2003; Li, Zhang, and Su 2004), and this remains a continuing predominant focus in transliteration research (e.g., Kunchukuttan et al. 2018; Amrhein and Sennrich 2020; Khakhmovich et al. 2020; Madhani et al. 2022). Methods for transliterating full sentences of informal romanized text have been explored for languages using Perso-Arabic (Maleki and Ahrenberg 2008; Al-Badrashiny et al. 2014; Eskander et al. 2014) and Brahmic (Hellsten et al. 2017) scripts (or both as in Lehal and Saini 2012, 2014; Roark et al. 2020), either as the means of processing existing text written in the Latin script, or within transliterating virtual keyboards (Hellsten et al. 2017; Wolf-Sonkin et al. 2019).

In recent years there has been an increased interest in transliteration as a means of "bridging the script gap" between related languages for constructing multilingual LLMs in NLP (Murikinati, Anastasopoulos, and Neubig 2020; Muller et al. 2021; Dhamecha et al. 2021; Moosa, Akhter, and Habib 2023) and multilingual ASR (Datta et al. 2020; Khare et al. 2021). Such LLMs pretrained on large amounts of general multilingual text data generalize well to many specific NLP scenarios when fine-tuned using smaller amounts of task-specific data (Izacard and Grave 2021; Markewich et al. 2022; Moezzi et al. 2023).

In South Asia, romanization is very common in most languages, and due to the lack of standardized orthography in the Latin script in those languages, as well as a general mismatch between phonemes in the languages and conventional use of the Latin script,[8] there is a high level of spelling variation, complicating accurate transliteration to the native scripts. The previously mentioned Dakshina dataset (Roark et al. 2020) provides text in both the Latin and native scripts of 12 South Asian languages. For each language, in addition to (1) a corpus of mono-script (i.e., only native script) Wikipedia text, which varies in size depending on the amount of raw Wikipedia material in the language, there is (2) a modest-sized romanization lexicon, where around 30k words in the native script are associated with one or more attested romanizations, as well as (3) full sentences from the native script Wikipedia sample that have been manually romanized in context. Of this latter collection, there are 5,000 development sentences and 5,000 test sentences, that is, sufficient for validation but not for training large-scale sequence-to-sequence models. Roark et al. (2020) evaluate finite-state-based pair *n*-gram (i.e., joint multigram), LSTM

7 See Karimi, Scholer, and Turpin (2011) for an overview of non-neural methods for transliteration.

8 For example, the Latin script letter 't' does not distinguish between dental, alveolar, or retroflex voiceless stops. Demirsahin et al. (2022) discuss various issues with Latin script representation in more depth.

and transformer transliteration models in both single token (non-contextual) and full sentence (context-aware) scenarios. It is this latter scenario that we mainly address in this paper, comparing with the results from that paper as baselines, as already seen in Figure 1. En route to these context-aware methods, however, we also investigate several non-contextual methods, including those mentioned above, which inform and/or form part of the later context-aware methods.

One may wonder about the rationale for focusing on full sentence transliteration in the Latin-to-native script direction and not the native-to-Latin script direction. There are a couple of reasons for this. First, and perhaps least satisfying, is that there is no orthography in these languages in the Latin script, hence there are many possible ways to effectively realize the text in the Latin script and it is difficult to decide when one is better than another—in contrast to the native scripts of these languages, which have orthographies, hence a meaningful notion of word-error rate. The second reason is that we do not have access to the kinds of high-quality romanized corpora that would allow either language modeling or parallel data simulation methods of the sort we pursue in this article. We rely upon the various native script Wikipedia text collections that form the basis of the Dakshina dataset, and there is no equivalent resource in the Latin script for these languages. In the absence of such resources, such work will have to wait.

We note that single-word transliteration dictionaries, of the sort that, for example, Kunchukuttan, Puduppully, and Bhattacharyya (2015), Kunchukuttan, Jain, and Kejriwal (2021), and Madhani et al. (2022) manually construct and/or mine from various resources, as well as non-contextual transliteration systems built from such data, form an essential part of the context-aware systems that we present in this article. The methods presented in the above and related papers are thus complementary to what we present here. In other words, while context-aware transliteration of the sort we ultimately pursue here is an important use scenario, our approaches make critical use of more conventional isolated-term transliteration data and modeling. The key questions are how best to make transliteration systems context-aware since full sentence parallel resources are not available at the scale of those for single words; and exactly how important is context-awareness for full sentence transliteration. In this article, we attempt to answer these questions.

3. Methods

3.1 Data

We train and evaluate[9] on data from the Dakshina dataset[10] (Roark et al. 2020), which consists of text corpora and lexicons derived from Wikipedia for 12 South Asian languages: Bengali (`bn`), Gujarati (`gu`), Hindi (`hi`), Kannada (`kn`), Malayalam (`ml`), Marathi (`mr`), Punjabi (`pa`), Sindhi (`sd`), Sinhala (`si`), Tamil (`ta`), Telugu (`te`), and Urdu (`ur`).[11] Four of these languages (`kn`, `ml`, `ta`, and `te`) are in the Dravidian family of languages; the rest are Indo-Aryan. Sindhi and Urdu are natively written in Perso-Arabic scripts, while the rest have native Brahmic scripts.[12]

9 Code, models and processed data are available at `https://github.com/google-research/google-research/tree/master/context_aware_transliteration`.

10 `https://github.com/google-research-datasets/dakshina`.

11 We use ISO 639-1 two-letter language codes as representational shorthand here and below (ISO 2002).

12 Punjabi is natively written in both a Perso-Arabic script (Shahmukhi) and a Brahmic script (Gurmukhi), but this data set only has Gurmukhi Wikipedia data.

For each language, there are three types of data: (1) monolingual text data in the native script of the language; (2) single word romanization dictionaries with one or more attested romanizations for a lexicon of words in the native script; and (3) romanizations of full Wikipedia sentences in the native script. Each language has a varying amount of monolingual text, depending on the size of the Wikipedia resource in that language, ranging from over a million sentences in Hindi and Tamil, to less than 100,000 sentences in Sindhi. Eleven of the twelve languages have 30,000 native script words in their romanization dictionaries, with 25,000 allocated to a training set and 2,500 each in development and test partitions. Sindhi has 20,000 native script words in its dictionary, 15,000 allocated for training and the rest split between development and test sets. For each language, 10,000 romanized Wikipedia sentences are split evenly between development and test partitions, although Sindhi and Urdu had a small number of sentences removed that were not in those languages. See Roark et al. (2020) for further details.[13]

3.2 Evaluation

Similar to Roark et al. (2020), our approach is to evaluate on the full sentence Wikipedia data, but not to train on any portion of that, so as to simulate the typical scenario of not having substantial human validated full sentence parallel training data. Rather, we train our models only on the single word romanization dictionaries and the monolingual (native script) text resources in the dataset. This includes scenarios where full sentence parallel data is simulated from the monolingual text, using native-to-Latin script transliteration models.[14] Hence, in addition to our final context-aware Latin-to-native script evaluations, we also examine non-contextual (single isolated word) transliteration performance in both directions.

3.2.1 Character-error Rate Percentage. For single word transliteration evaluation in the Latin-to-native script direction, where there is generally a single canonical spelling for the languages we are investigating, we evaluate systems with character-error rate percentage (CER%). Let the reference word be taken as a string of Unicode codepoints $R = r_1 \dots r_{|R|}$, and the system output word also a string of Unicode codepoints $S = s_1 \dots s_{|S|}$. Let $E(S, R)$ be the minimum number of edits (substitutions, deletions or insertions) required to change S to R, that is, the Levenshtein distance (Levenshtein 1966). Then, over a full corpus of (S, R) pairs, CER% is defined as

$$\text{CER\%} = 100 * \frac{\sum_{S,R} E(S, R)}{\sum_R |R|} \tag{1}$$

For example, if the reference string R is `abcd` and the system output string S is `aebd` then $E(S, R) = 2$ (deletion of `e` and insertion of `c`) and CER% is $100 * 2/4 = 50.0$.

13 In particular, Roark et al. (2020) detail the construction of single-word romanization dictionaries and full-sentence romanizations in Sections 3.2 and 3.3 of their paper, and the overall corpus statistics are provided in their Table 1.

14 This is similar to the use of *back-translation* for generating synthetic parallel data using the target-side monolingual data and the reverse model in machine translation (Sennrich, Haddow, and Birch 2016; Edunov et al. 2018).

3.2.2 Minimum CER%. When evaluating native-to-Latin transliteration, there are often multiple possible (attested) romanizations for the input native script term. In that scenario, we can define the minimum CER% (minCER%) as the minimum that can be achieved with any of the given references. For a given system output S, let $\{R_1, \ldots, R_k\}$ be the set of k attested reference romanizations for the input word, and let $\hat{R}(S) \in \{R_1, \ldots, R_k\}$ be the reference that yields the minimal CER% for S. Then

$$\text{minCER\%} = 100 * \frac{\sum_{S,\{R_1 \ldots R_k\}} E(S, \hat{R}(S))}{\sum_{S,\{R_1 \ldots R_k\}} \left|\hat{R}(S)\right|} \tag{2}$$

3.2.3 Earth Mover's Distance k-best Evaluation. While the minCER% evaluation provides some basis for comparing the highest probability system outputs, it does not account for either how frequently the various reference romanizations were attested, nor for the quality of k-best output from the systems beyond the 1-best. This is particularly important given our principal use scenario for automatic romanization: simulation of full-sentence parallel transliteration data from native script Wikipedia sentences. To produce realistic romanizations that include the kind of spelling variability that will be encountered, we will sample from likely alternatives—see Section 3.4.2. Hence quality of k-best lists is important to also assess. In this section, we present a new evaluation method, based on earth mover's distance, to address these shortcomings. First we'll motivate the approach via the sampling use case.

For a given input word, let $\{R_1 \ldots R_m\}$ be m distinct attested romanizations in the set of references, and let $c(R_i)$ denote the number of times romanization R_i was attested, that is, its count. Based on these counts, we define the maximum likelihood multinomial distribution $P(R_i)$ over possible romanizations:

$$P(R_i) = \frac{c(R_i)}{\sum_{j=1}^{m} c(R_j)} \tag{3}$$

Let $\{S_1 \ldots S_k\}$ be unique romanizations in a softmax-normalized k-best list produced by the romanization system. This also defines a multinomial distribution over possible romanizations for the input word. If we sample with replacement N times from the m reference romanizations, based on the distribution defined in Equation (3), this gives us N items in a reference sample. If we also sample with replacement N times from the k system romanizations, based on the distribution defined by the softmax scores, then we have N items in a system sample. We can then ask the question: How well does the system sample match the reference sample?

One natural way of assessing the match is via an error rate: the minimum number of edits required to convert the system sample into the reference sample, divided by the size of the reference sample. One method to determine this value is by treating this as a special case of the assignment problem, which involves minimizing the cost of allocating items from one set to items in another set. Each system item should be matched to a reference item with as low a cost as possible, while maintaining a 1-1 mapping between system and reference items. This is often formally presented as an optimization over a bipartite graph, with two disjoint sets of nodes X and Y, and edges $E(x, y)$ between one node in $x \in X$ and one node in $y \in Y$, where each edge has a cost. In our case, nodes in X would be items in the reference sample; nodes in Y items in the system sample; and the cost of an edge the minimum number of edits to convert the system sample item into the reference sample item. Since the size of the reference

sample is constant, minimizing the number of edits would give us the minimum error rate, which in this case is defined at the character level as in CER% defined above in Section 3.2.1.

Rather than sampling many times and calculating this assignment-based CER% for each sample as the means for scoring, we can instead calculate the earth mover's distance, which also takes pairwise distances (in our case character edits) between items in the system and reference *k*-best lists, and directly finds the minimum cost (distance times probability) to convert the system *k*-best list to the reference *k*-best list.[15] In essence, the probability mass (earth) associated with particular system items is allocated (moved) to occupy probability mass associated with particular reference items, accruing cost based on the distance moved. The algorithm that we use to solve the earth mover's distance[16] is based on Pele and Werman (2008, 2009).

Note that, if there is only one reference and one system output, this measure is equivalent to the standard character error-rate.

3.2.4 Word-error Rate Percentage. When moving to full-sentence evaluation, we shift from CER% evaluation to word-error rate percentage (WER%), which is defined similarly but with whitespace-delimited words rather than Unicode codepoints. Let the reference sentence be taken as a string of whitespace delimited words $R = r_1 \dots r_{|R|}$, and the transliteration system output also a string of whitespace delimited words $S = s_1 \dots s_{|S|}$. Let $E(S, R)$ be the minimum number of edits (substitutions, deletions, or insertions) required to change S to R, where these are edits on whole words (rather than the Unicode codepoints used to calculate CER%). Then, over a full corpus of (S, R) pairs, WER% is defined as

$$\text{WER\%} = 100 * \frac{\sum_{S,R} E(S, R)}{\sum_R |R|} \tag{4}$$

We shift from CER% to WER% in this scenario because this is conventional, and it allows for direct comparison with the results in the paper that introduced the Dakshina dataset (Roark et al. 2020). We follow their "whitespace evaluation" of full-sentence transliteration, which requires some data preprocessing. Briefly, this approach to evaluation treats any character that does not appear in the native script portion of the language's romanization dictionary as part of the whitespace for evaluation purposes.[17] Some of the native script text strings from Wikipedia may contain, for example, short Latin script parentheticals or other substrings (digits, etc.) outside of the native script letters of the language, which annotators were instructed to include in their romanized version unchanged. We refer readers to the dataset URL (see footnote 2) for relevant details on corpus creation and to Roark et al. (2020) for further details on this preprocessing.

3.3 Sequence-to-sequence Modeling

We use a variety of sequence-to-sequence models in this paper, both for non-contextual and context-aware transliteration, and we describe our specific methods in this section.

15 Thanks to an anonymous reviewer for pointing this out.

16 `https://pypi.org/project/pyemd/`.

17 Similarly, input strings in the Latin script are lowercased, and any non-alpha characters are treated as whitespace.

The romanization lexicons in the Dakshina dataset pair single words in the native script with romanizations and how often they were attested. For example, the Hindi word अंकगणित (arithmetic) is represented as a three-tuple (अंकगणित, *ankganit*, 3) indicating that annotators romanized this word as "*ankganit*" 3 times. From these word-level alignments, we build several distinct kinds of non-contextual transliteration models that take a single word in either the Latin or native script as input and provide *k*-best transliterations of that word into the other script as output. For each of these modeling methods, training data is prepared from the above lexicon format by repeating each training example the number of times it is attested, e.g., three times for the above example. For this task we explore the following methods:

1. Two standalone neural models: LSTM and transformer (Section 3.3.1),
2. Fine-tuned pretrained neural sequence-to-sequence models (mT5[18] and ByT5, Section 3.3.2),
3. A non-neural finite-state transducer (FST) based method (Section 3.3.3).

Apart from the T5-based methods, all of these methods were also used by Roark et al. (2020) for single word transliteration. To replicate their baseline results as our starting point, we adopt architectures and meta-parameters from that paper for the methods that were used there. mT5 and ByT5 were also used to build models for context-aware full-sentence transliteration. In both cases, pretrained checkpoints were fine-tuned using simulated full sentence parallel training data—see Section 3.4.2 for details on data simulation.

For all neural models, *k*-best extraction is done using beam search (Spohrer et al. 1980; Ney et al. 1987), while for FST-based modeling shortest-path extraction algorithms are used (Mohri 2002).

3.3.1 Standalone Neural Models. For training standalone neural models, we use the Adam optimizer (Kingma and Ba 2014), and for each training run, we extract the best performing checkpoint on a small portion of the training set that has been held aside for this purpose.

LSTM. We use both forward and backward LSTM layers within a single deep bidirectional encoder, which is connected via Luong, Pham, and Manning (2015) attention to a forward decoder LSTM (Bahdanau, Cho, and Bengio 2014). Again, following Roark et al. (2020), the 2 layers of the encoder have 256 hidden units, while the 3 layers of the decoder have 128. The character (single Unicode codepoint) embedding has dimension 512. We refer readers to that paper for further training settings, such as dropout for the various layers, which we followed here for all languages.

Transformer. Following Roark et al. (2020), we train transformers (Vaswani et al. 2017) for single word input with the architecture presented in Chen et al. (2018, Appendix A.2), using meta-parameters and settings identical to Chen et al. (2018) other than:

18 We omit results using mT5 for single word transliteration, since it underperforms relative to ByT5. We use mT5 on the full-sentence task, since its subword tokenization has benefits relative to ByT5's byte tokenization in cases with long-distance dependencies, such as context-aware full sentence processing.

dropout (0.36), model dimension (128), hidden dimension (1,024), attention heads (4), and transformer layers in encoder and decoder (4). Input is tokenized into single Unicode codepoints.

3.3.2 mT5 and ByT5. Raffel et al. (2020) introduced the "Text-to-Text Transfer Transformer" (T5) framework, which proved to be successful in many downstream NLP tasks. The idea of T5 is that the same sequence-to-sequence transformer model, initially pretrained as a large language model, can be fine-tuned to any particular task by decorating input text with additional affixes telling the model what to do (e.g., a translation task might add the input prefix: "*Translate English to German:*"). The original T5 was pretrained on the "Colossal Clean Crawled Corpus" (C4) corpus, which is a data set consisting of hundreds of gigabytes of clean English text scraped from the Web. The core pretraining task was recovering corrupted spans, a form of masked language modeling (for example, "*I took a walk in the* <extra_id0>." could map to "<extra_id0>*park*<extra_id1>"). The model's vocabulary consisted of 32k SentencePiece tokens (Kudo and Richardson 2018), built from C4, as well as 100 additional <extra_id> tokens reserved for representing masked spans of text.

Xue et al. (2021) expanded the original T5 by introducing mT5, which is a multilingual version of T5 pre-trained on mC4—a new Common Crawl-based dataset covering 101 languages. Similar to T5, mT5 proved to be effective in several NLP tasks (Ruder et al. 2021; Nagoudi, Elmadany, and Abdul-Mageed 2022). mT5 is also pretrained to perform masked language modeling, but uses a larger SentencePiece vocabulary of 250k items to accommodate all languages, along with 100 additional span tokens. ByT5 (Xue et al. 2022), otherwise pretrained on the same data and tasks as mT5, was an attempt to both universalize this vocabulary, and drastically reduce its size for computational reasons. In ByT5, all input and output text (including Unicode) is broken down into its component byte sequence. The entire vocabulary size, including extra control symbols and rounded to the nearest multiple of 128, consists of 384 items. Using a byte vocabulary allows the model to generalize to any input encoding, but does have the disadvantage that the model can generate malformed outputs (e.g., hallucinate sequences of bytes that don't correspond to any Unicode codepoint). In practice, this is extremely rare, and any spans of bytes that can't be processed with the chosen encoding are simply removed from the output string. All 12 Dakshina languages are included in mC4, hence supported by both mT5 and ByT5.

In this article, we fine-tune ByT5 for single word non-contextual transliteration, starting with the publicly available pretrained "base" configuration.[19] Each fine-tuning run consisted of a mixture of 24 transliteration tasks, each defined using the SeqIO framework (Roberts et al. 2022) on the T5X codebase,[20] for each Dakshina language in both Latin-to-native and native-to-Latin directions. To distinguish each task, each input string was prefixed with the language being transliterated, and the source and target scripts (e.g., "hi-Deva-Latn-अच्छा" maps to "accha"). The fine-tuning data used for each language was the same as the individual language training data used to train our other LSTM, transformer, and pair *n*-gram models. Fine-tuning on the task mixture proceeded for 50,000 steps, with a batch size of 64, and dropout of 0.1. Checkpoints were evaluated on a held-out portion of the training set every 500 steps using the CER%

19 https://github.com/google-research/t5x/blob/main/docs/models.md#byt5-checkpoints.
20 https://github.com/google-research/t5x.

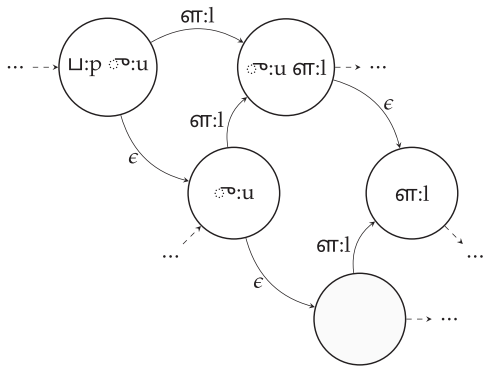


Figure 2
Schematic of pair trigram Tamil romanization model over Unicode codepoints.

metric. For any subsequent inference, for each task we selected the checkpoint with the best performance on the held-out portion of the training set.[21]

For full-sentence context-aware transliteration (using the simulated parallel data described below, organized into the same 24-task mixture used for single-word ByT5 transliteration) we apply both ByT5 and mT5.[22] We experiment with both the "base" and "large" pretrained configurations available. We fine-tune for 400k steps (more than the 50k used for the single-word task to account for the increased complexity and longer sequences involved in full-sentence transliteration). In all cases we use a batch size of 64 with dropout set at 0.1.

3.3.3 Pair n-gram Models. The FST-based method for single-word transliteration relies on *n*-gram models over pair symbols, an approach originally taken for grapheme-to-phoneme conversion (Bisani and Ney 2008) but also used for transliteration (Hellsten et al. 2017). For example, the Tamil example earlier (புளி, romanized as "*puli*") would be modeled as a sequence of paired symbols, one Unicode codepoint (or the empty string ϵ) from the input and one (or ϵ) from the output: ப:p ◌ு:u ள:l ◌ி:i

Given aligned words of the sort provided by the lexicon-derived training data, we use the expectation maximization algorithm to derive single-character alignments between input and output strings (similar to details in Hellsten et al. 2017), which are then used to estimate an *n*-gram model. Following Roark et al. (2020) we train pair 6-gram models with Witten-Bell smoothing (Witten and Bell 1991), using the OpenGrm library (Roark et al. 2012),[23] yielding models in the OpenFst format (Allauzen et al. 2007).[24] In the experiments described in Section 4 we refer to this model as "Pair 6g."

Figure 2 presents the schematic of an FST-representation (Roark et al. 2012) of a pair trigram model, using the புளி/*puli* Tamil example. This model can be converted directly to a finite-state transducer, by splitting pair labels to an input and an output label corresponding to the input and output sides of the pair, respectively. An automaton

21 We did investigate separately fine-tuning each task independently from the others, but did not observe any accuracy gains from this. Since the joint fine-tuning is significantly more efficient, we only report results from the joint fine-tuning.

22 https://github.com/google-research/t5x/blob/main/docs/models.md#mt5-checkpoints.

23 http://www.opengrm.org.

24 http://www.openfst.org.

representing the input string can be composed with this transducer to derive all possible output transliterations with their probabilities according to the model. See Appendix A for more explicit details on using these finite-state models.

3.4 Training Data Preparation

We have two important methods related to training data preparation, one to improve fine-tuning performance of ByT5 for single-word transliteration, and another to simulate full-sentence parallel training data by automatically romanizing native script text.

3.4.1 Romanization Dictionary Pruning for Fine-tuning. Initial experiments using ByT5 for single-word transliteration yielded generally good performance across the 12 languages, but some unusual results for Tamil Latin-to-native script transliteration in particular led us to a method for data preparation for fine-tuning that we describe here. We found that fine-tuning of the pretrained model was quite sensitive to outliers/noise in the training set, so that pruning up to 10% of the least representative data yielded uniformly better performance and dramatically better performance in some languages, such as Tamil. In contrast, the non-pretrained models—LSTM, Transformer, and pair *n*-grams—were more robust to outliers/noise, so that no improvements were achieved by such pruning of the training data. We thus apply this pruning only to the ByT5 fine-tuning data and leave the training data for the other methods unchanged.[25]

For outlier detection in each training set, we assign scores to each training instance by performing 20 random train/test splits of the set, where half of the examples go into a training partition and half into a test partition. For each of these 20 random splits, we train a pair 4-gram model (see Section 3.3.3) from the training partition to transliterate native-script words into the Latin script, which is used to automatically romanize the words in the test partition. We then measure a normalized distance d between each reference romanization in the test half with the model-predicted romanization. This is repeated 20 times, and each item's score is the mean distance for the trials where it fell in the test partition. The 10% of items with the highest mean distance are removed from the fine-tuning set. See Appendix B for more specific details on this pruning method, as well as a comparison with another investigated alternative.

As detailed in Appendix B, this has a large impact on accuracy of Latin-to-native script single-word transliteration in Tamil, reducing mean CER% from 12.7 to 7.9 while also greatly reducing variance (Table B.3). While that was the largest observed reduction among the 12 languages, this method reduced CER% for most of the languages and did not meaningfully change it for the few that were not improved. In contrast, as noted above, the non-pretrained methods (pair *n*-gram, LSTM, and Transformer) did not benefit from this training set pruning, hence were trained on the whole set. Please see Appendix B for more analysis and further details.

3.4.2 Parallel Data Simulation for Full-sentence Transliteration. We have two broad categories of context-aware systems that we investigate: those trained on simulated

25 The higher sensitivity of neural models to noise compared to statistical approaches has been noted in the past in the context of recurrent neural network-based NMT (Khayrallah and Koehn 2018). Our LSTM and transformer transliteration models, however, were robust to the kind of noise present in our parallel data. The anomalous behavior of the fine-tuned ByT5 on Tamil is in line with more recent observations on popular LLMs (Kumar, Makhija, and Gupta 2020; Moradi and Samwald 2021; Lee et al. 2022; Schoch, Mishra, and Ji 2023).

parallel data, and those that incorporate language model information during decoding. Both rely upon native script Wikipedia text in the Dakshina dataset for training. The pages included in the training partition of that collection are disjoint from the pages from which the manually romanized sentences were drawn, hence this provides an independent source of text from which we can learn contextual dependencies. Here we detail the training data simulation methods, and in the next section, language modeling. Because we follow the "whitespace evaluation" approach of Roark et al. (2020), as described in Section 3.2.4, the same tokenization is performed on these corpora for both parallel data simulation and language modeling, to match the testing condition, that is, Unicode codepoints not used in the native script part of the romanization lexicon become whitespace.

We used the following approach to simulate parallel sentences. Using the best system for native-to-Latin single word transliteration (see Section 4.2 for experiments assessing this), we produce weighted k-best romanizations[26] for each word in the corpus. In order to capture the spelling variability in the Latin script, we romanize 10 copies of the corpus: In each pass, at each word, we randomly sample a romanization from the k-best list, according to the multinomial distribution defined by the system. For example, suppose that we had a 3-best list of romanizations for the Hindi word जाँबाज़ (daredevil): "*janbaaz*" with probability 0.7; "*janbaz*" with probability 0.2; and "*janbaj*" with probability 0.1. Every time that particular word is encountered in the Wikipedia corpus, a romanization is sampled from this set according to the multinomial distribution, so that, if 100 instances of the word are found in the corpus, on average 70 of them would be romanized as *janbaaz*, 20 as *janbaz*, and 10 as *janbaj*.

3.5 Language Modeling

One of our context-aware transliteration approaches makes use of monolingual language models as part of the approach, and we detail those methods in this section. See note in the previous section about tokenization of the corpora to remain consistent with "whitespace evaluation."

3.5.1 Sentence-level Context-aware Modeling. Our language models are made open-vocabulary by virtue of using wordpiece tokenization (Schuster and Nakajima 2012),[27] modulo character coverage in the wordpiece model. A wordpiece tokenizer segments words into sub-word units, and is trained via agglomerative clustering, starting with a vocabulary of single Unicode codepoint segments and adding new substrings to the vocabulary by combining existing vocabulary items until a target vocabulary size is reached. Given a vocabulary, segmentation is done by maximizing the unigram likelihood of all the wordpieces in a string (Kudo and Richardson 2018). For this article, we targeted 32k vocabularies for each language. Each wordpiece vocabulary consists of two disjoint sets: word-initial wordpieces and word-internal wordpieces, the former by convention distinguished with an underscore (_) prefix. Conversion from words to wordpieces (and back) is fully deterministic.

26 For this paper, we set $k = 8$.

27 Wordpiece tokenization was also used in Roark et al. (2020), and differs from the SentencePiece tokenization used for the mT5 models described above.

We train FST-based *n*-gram language models with the same OpenGrm *n*-gram library (Roark et al. 2012) as the pair *n*-gram transliterations models. We use *n*-gram models encoded as FSTs to enable straightforward composition with lattices of possible transliterations, followed by global shortest-path extraction to derive *k*-best lists—see Appendix A for details. This provides a complementary alternative to the beam-search methods used for the neural pretrained models, and yields very competitive results (both alone and in ensembles)—see experiments in Section 4.3. For experimentation, we train both context-aware models, that is, over full sentences, and non-contextual models, just over single words. In both conditions we train models with wordpiece-based vocabularies, and for the experiments in this paper, both types of models are 4-gram Kneser-Ney backoff models (Kneser and Ney 1995).

3.5.2 Word-level Non-contextual Modeling. A non-contextual model with a word-based vocabulary is just a unigram model, trained standardly via relative frequency estimation. For non-contextual models with wordpiece vocabularies, however, we build word-level wordpiece 4-gram models. This is achieved by first segmenting the training corpus so that each word occurs on a separate line, then tokenizing to wordpieces. As stated earlier, standard 4-gram Kneser-Ney backoff language models are trained on this segmented/tokenized corpus. However, since word-initial pieces always and only occur word-initially and word-internal pieces cannot, there is a slight difference in backoff structure from standard language models, which we implement as follows.

A schematic of the canonical FST format for *n*-gram language models was shown in Figure 2 for the pair *n*-gram transliteration models, but the same format is used for any FST-based *n*-gram language model regardless of the vocabulary. When modeling at the word-level with a wordpiece vocabulary, there is an additional constraint that changes how the automaton should represent the model, namely, that word-internal pieces (conventionally those tokens missing the prefix ▁) cannot occur word-initially and all others *must* occur word initially. Because all word-initial pieces label transitions leaving the start state of the FST, and no other pieces are allowed, there is no need to back off to the unigram state from the start state, hence that backoff transition is removed. As a result, only word-internal pieces are required at the unigram state, so (1) all other unigrams are removed, (2) the unigram probabilities are renormalized and (3) all backoff weights are recalculated.

3.6 Ensembling

For both single word transliteration and full-sentence transliteration, we make use of ensembling methods, which we present in this section. Additionally, we present methods for constructing and using a cache of ensembled single word transliterations, which avoids the overhead and complexity of ensembling at time of full-sentence transliteration.

3.6.1 Single Word Transliteration Ensembling. Transliterations from two or more systems can be combined, and we take a very simple approach for this. Let $S_1 \ldots S_m$ be m sets of transliteration candidates with associated log probabilities. Assume each set's log probabilities have been softmax-normalized, and let $p_i(t)$ be the probability of transliteration t in set S_i, where $p_i(t) = 0$ if $t \notin S_i$. The new set of transliterations is the union of the transliterations from all systems being ensembled, and the ensembled probability

of each transliteration in that set is the mean probability of that transliteration over all systems being ensembled:

$$S_{(1,\ldots,m)} = \bigcup_{i=1}^{m} S_i \quad \text{and} \quad p_{(1,\ldots,m)}(t) = \frac{1}{m} \sum_{i=1}^{m} p_i(t) \tag{5}$$

We apply this method using k-best output from individual systems.

Note that the multiple systems being ensembled could be using the same modeling method and training data, just differing in their random initialization. We investigate this sort of single method ensembling in the experiments, alongside ensembles of heterogeneous modeling methods.

Also note that we do not tune mixing parameters in the experiments in this paper, so that if m systems are ensembled, they each contribute $\frac{1}{m}$ of the probabilities in the final ensemble. While it is likely that some additional system improvements could be achieved by further tuning of these parameters, for the current article we find ample benefit even without such additional optimization.

3.6.2 Single Word Transliteration Ensemble Caching. Performing inference with multiple models and ensembling the results, as described in the prior section, can require a relatively expensive (i.e., slow) and complex sequence of operations; hence there is a real accuracy/efficiency tradeoff to consider when deciding to use such an approach. One method to address this is to pre-compute a cache of transliterations offline for some vocabulary, which can be accessed via simple lookup at time of full sentence processing, and rely on non-ensembled transliterations for tokens not found in the cache. The key questions in pursuing such an approach are: (1) what is the lexicon to be included in the cache? and (2) what full sentence transliteration methods are amendable to including such a lookup mechanism at time of inference?

Regarding the lexicon to be included in the cache, we would ideally like to pre-compute transliterations into the native script for high frequency romanizations in each language; however we lack a large corpus of romanized text in these languages. Instead, we make use of the simulated full sentence parallel transliterations, construction of which is presented in Section 3.4.2. The romanizations in that collection were automatically produced from the native-script side, hence may or may not match the romanizations that are found in our manually romanized full sentence validation data. However, they were derived from Wikipedia sentences, hence they do provide type frequencies for deciding what words to include in the lookup cache. For each language, we extract the frequency of all romanizations in the simulated parallel data and rank them in descending frequency order. For a given maximum cache size m, we determine the frequency f of the mth most frequent romanization in the corpus; then we include all romanizations with frequency greater than f and a random selection of romanizations with frequency f so that the final cache size includes the requested number of types. In this paper, for Sindhi, there was enough text to include up to 450,000 romanizations in the cache; for all other language, we included up to 1 million romanizations in the cache.

For each romanization in the cache, we include the k-best transliterations of that romanization into the native script with scores derived from the ensemble. Our end-to-end neural full sentence transliteration methods—mT5 and ByT5—have no straightforward way to incorporate these cached single word transliterations. However the "noisy channel" approach, where single word transliteration systems are combined

with language models, do permit easy inclusion of such cached transliterations. For each input Latin script word in a sentence, the cache is queried for pre-compiled transliterations. If they are found, those are the returned candidate transliterations for that word; otherwise, a single model is used to derive the set of candidate transliterations for that word. For this article, we make use of the non-neural pair *n*-gram model as the single model to use for words not present in the cache.

This approach allows transliteration inference to proceed without spending time ensembling multiple models, but at the potential cost of accuracy for those words that fall outside of the cache. If the constructed cache has high coverage, the results will be close to those achieved with full ensembling. We examine performance of these systems in Section 4.4 and show additional coverage information in Appendix E.

3.6.3 Full Sentence Transliteration Ensembling. Ensembling full sentence *k*-best lists differs slightly from single words, and we detail those differences here. Single word ensembling combines probabilities from multiple systems when the words are identical, however, when the system outputs are full sentences, the frequency of exact matches is much lower, so that we want to combine evidence even for partial matches.

The first step in combining partial evidence across output sentences is to align the sentences, so that the same word in the same sentence position in two different system outputs can accrue the benefit from each. There are many heuristic algorithms for such multiple-sequence alignment problems, some of which are more expensive than others (Russell 2014), and here we adopt some simple heuristics that are fast and quite effective. First, one of our full sentence transliteration approaches (the so-called noisy channel) is guaranteed to have the same number of output words as input words.[28] Our neural methods (mT5 and ByT5) do not have such a guarantee, though in practice this is most often the case—since they are fine-tuned on simulated data that has this property. See Section 4.5.3 for a discussion of this. We thus make the simplifying assumption that words in different system outputs are aligned at their absolute position in the sentence, and for simplicity we discard system outputs that differ in length from the input sentence. Since one system is guaranteed to have system outputs of that length, this condition has no impact on coverage.

Let $S_1 \dots S_m$ be the outputs of m systems, which are softmax-normalized k-best lists. Let $S_{ij} = s_{ij1} \dots s_{ijn}$ be item j from k-best system output S_i, which is a string of n words,[29] and let $\mathrm{P}(S_{ij})$ be the probability of S_{ij}.[30] Then we define the ensembled weight $\mathcal{W}_p(w)$ for word w at position $1 \leq p \leq n$ as the sum of the probability over all system outputs with word w at position p, divided by the number of systems:

$$\mathcal{W}_p(w) = \frac{1}{m} \sum_{ij} \delta_{w s_{ijp}} \mathrm{P}(S_{ij}) \tag{6}$$

28 This is due to the method relying on transliteration candidates for each input word, which is both a limitation and a strength of the approach. Real human sentence transliteration is not necessarily one-to-one in this way, though it is likely to be one-to-one most of the time. This constraint can be leveraged during ensembling.

29 Since we required that outputs have the same number of words as the input string, this length is the same for all items.

30 Due to softmax normalization of each list, $\forall i \ \sum_j \mathrm{P}(S_{ij}) = 1$.

where $\delta_{ws_{ijp}}$ is a Kronecker delta that has value 1 if $w = s_{ijp}$ and 0 otherwise. Then for all strings of n words $w_1 \dots w_n$:

$$\mathcal{W}(w_1 \dots w_n) = \prod_{p=1}^{n} \mathcal{W}_p(w_p) \tag{7}$$

Note that some strings not included in the original collection of k-best lists may have non-zero weight, as they accrue evidence from different system outputs. With these definitions, we can extract a new ensembled k-best list, which includes those strings with the highest weight according to Equation (7).

As with the single word ensembles, we do not tune the mixing parameters, so that all systems contribute equally to the final weights.

4. Experiments

In the absence of full sentence parallel training data, our context-aware methods for Latin-to-native script transliteration must rely in some fashion on single word (non-contextual) transliteration models—either to provide candidate transliterations of each Latin script word in the input string, or to simulate possible romanizations from native script full sentences for fine-tuning. We thus begin with experiments investigating single isolated word (non-contextual) informal Latin script to native script transliteration, followed by experiments examining non-contextual native-to-Latin script transliteration. We can then turn to experiments on Latin-to-native script context-aware transliteration.

4.1 Non-contextual Latin-to-native Transliteration

As stated in Section 3.1, for Sindhi the Dakshina dataset contains 15,000 native script words with one or more attested romanizations in the single isolated word training set; for the other eleven languages there are 25,000 native script words in their respective training sets. In this section, we experiment with training Latin-to-native script transliteration models on these training sets and evaluating on the held-aside sets (2,500 words for each language). We reproduce results for three models reported in the Dakshina paper (Roark et al. 2020), with means and standard deviations over 25 training runs rather than the 5 runs from that paper; and add one new modeling approach (ByT5). The 25 different training runs enables us to experiment with producing 5 different ensembles of 5 models each, to demonstrate whether such single method ensembling yields improved accuracy and/or variance reduction while still reporting means and standard deviation over 5 different ensembles. We then examine ensembling with multiple different models, and use the results to choose the non-contextual transliteration approach with the best performance for later experiments with full sentence transliteration.

4.1.1 Single Systems. Table 2 presents means and standard deviations of character error-rate (CER%), comparing Roark et al. (2020) results over 5 training runs (columns labeled "Dakshina") with results from this current article using the same modeling methods (and matched meta-parameters) over 25 training runs. Results with current ByT5 models are also presented. Performance for each language, and the micro and macro averages, are presented in the table. For each of the methods explored in Roark

Table 2
Single word Latin-to-native transliteration character-error rate percentages (CER%) for each language plus macro averages (μ) and micro averages (all). Note that the Roark et al. (2020) macro averages (columns labeled "Dakshina") are calculated from reported values, hence no standard deviation is available.

Lang	Pair 6g		Transformer		LSTM		ByT5
	Dakshina	Current	Dakshina	Current	Dakshina	Current	Current
bn	14.2 (.02)	14.3 (.12)	13.2 (.07)	13.0 (.13)	13.9 (.15)	13.7 (.11)	**12.4** (.67)
gu	12.9 (.04)	13.0 (.06)	11.9 (.15)	11.9 (.16)	12.6 (.06)	12.4 (.15)	**10.0** (.44)
hi	14.7 (.04)	14.8 (.05)	13.4 (.21)	13.4 (.16)	13.9 (.10)	13.8 (.17)	**11.0** (.55)
kn	7.2 (.04)	7.3 (.08)	6.3 (.12)	6.6 (.18)	6.8 (.04)	6.7 (.12)	**5.6** (.46)
ml	10.0 (.07)	10.0 (.03)	9.0 (.04)	8.9 (.14)	9.2 (.03)	9.1 (.10)	**8.5** (.52)
mr	12.4 (.03)	12.6 (.16)	11.6 (.10)	11.6 (.17)	12.5 (.08)	12.4 (.15)	**10.0** (.65)
pa	17.9 (.07)	18.0 (.06)	17.4 (.33)	17.0 (.16)	17.5 (.04)	17.5 (.17)	**15.6** (.50)
sd	20.5 (.06)	20.7 (.15)	22.0 (.32)	20.0 (.22)	20.6 (.11)	20.4 (.21)	**19.6** (.50)
si	9.1 (.01)	9.3 (.02)	9.2 (.10)	**9.0** (.12)	9.3 (.04)	9.2 (.10)	9.2 (.44)
ta	9.3 (.08)	9.4 (.04)	9.4 (.52)	8.2 (.10)	8.4 (.12)	8.6 (.13)	**7.9** (.42)
te	6.9 (.02)	7.0 (.10)	6.2 (.11)	6.2 (.12)	6.8 (.08)	6.7 (.07)	**5.9** (.40)
ur	20.0 (.07)	20.0 (.02)	19.5 (.10)	**19.4** (.24)	19.4 (.08)	19.6 (.12)	20.5 (.61)
μ	12.9	13.0 (.03)	12.4	12.1 (.05)	12.6	12.5 (.05)	**11.4** (.40)
all		12.9 (.03)		12.0 (.05)		12.4 (.05)	**11.3** (.40)

et al. (2020), we implemented and trained our own, and the results are quite similar but not identical. Of course, these are means over many training runs, so some divergence is expected.

Several things can be seen in the results in the table. First, the Roark et al. (2020) results and our reproduced methods are generally very close. Our transformer results are slightly better than what was reported in that paper, but the pattern remains that transformer yields the best results overall, with LSTM slightly worse and the non-neural Pair 6g method trailing the best performance by just under 1% absolute.

The ByT5 results are excellent, providing the lowest CER% in 10 out of 12 languages. As discussed in Section 3.4.1, in order to achieve these results, we had to filter roughly 10% of the data used to fine-tune the models. Without such filtering, the model provided the lowest CER% in just 4 of the 12 languages, and had substantially higher CER% than the others in multiple languages, most notably Tamil (examined in detail in Appendix B), often with extremely high variance. Of the methods presented in the table, ByT5 continues to have the highest variance, something we look to control via ensembling.

4.1.2 Single Method Ensembling. One way to improve the accuracy and reduce the variance associated with some of these methods is to simply train multiple models with random initialization, and ensemble the result. The first two columns of Table 3 present the macro average CER% means and standard deviations over 25 single models (None) and 5 ensembles of 5 models (This x5) for our four modeling methods. The already very low variance Pair 6g model achieves no reduction in mean CER% via these methods, but the other methods do achieve improvements. Interestingly, transformer ensembling yields less improvement through ensembling than for either LSTM or ByT5, so that the LSTM ends up yielding the same CER% as transformer after single method ensembling. ByT5 obtains a substantial reduction in CER% through this method, and the variance between runs is also substantially reduced. Table D.1 in Appendix D presents per-language results, combined with the CER% reduction obtained versus the means of the single (non-ensembled) systems.

Table 3
Single word Latin-to-native transliteration macro average character-error rate percentages (CER%) with various types of ensembling.

Model type	Ensembling method None	This x5	This+ Trans.	This+ LSTM	This+ ByT5	This+all others	All except this model
Pair 6g	13.0 (.03)	13.0 (.02)	11.5 (.01)	11.5 (.02)	10.6 (.02)	10.5 (.02)	10.61 (.03)
Trans.	12.1 (.05)	11.8 (.02)		11.4 (.03)	10.5 (.03)		**10.43** (.03)
LSTM	12.5 (.05)	11.8 (.02)			10.5 (.02)		10.44 (.02)
ByT5	11.4 (.40)	10.9 (.04)					11.13 (.01)

4.1.3 Multi-model Ensembling. In addition to the single method ensembling results, Table 3 presents results of ensembling multiple models. For each modeling method included in the ensemble, we include models from 5 different random initializations, hence if there are k modeling methods being ensembled, there are a total of $5k$ models in the ensemble. Of the six possible 2-method combinations, those with ByT5 are the top 3, and the combination of ByT5 with LSTM provides the lowest error rate of those 2-model combinations. Adding the non-neural pairLM to that 2-model combination (the result in Table 3 for "all except" the transformer model) yields (just barely) the best performing model—even better than ensembling with all of the models. Tables D.2 and D.3 in Appendix D provide per language ensembling results for all of our various combinations. Note that the 10.4% CER is 2% absolute (15% relative) reduction versus the best reported result from the Dakshina paper.

Our principal interest for these models is to contribute to full string context-aware transliteration systems, so we want the methods that provide the best starting point for such systems. For that reason, we choose to conduct our full string experiments (see Section 4.3) using the 3-model ensembled system with ByT5, LSTM, and Pair 6g transliterations. We also make use of single word (non-contextual) transliteration in the other direction (native-to-Latin) for some eventual context-aware systems, and we turn now to examining system behavior and the utility of ensembling in that case.

4.2 Non-contextual Native-to-Latin Transliteration

While Roark et al. (2020) make use of native-to-Latin script transliteration in preparation of simulated parallel training data, they do not explicitly evaluate the performance of their models on this task, hence we must rely on our replicated methods to establish baselines and the best performing system configuration. The same training data can be used for this task as for the Latin-to-native transliteration described in the previous section, by simply swapping the input and output strings. As described in Section 3.2.2, given the many potential attested (reference) romanizations for each input, we calculate the error rate based on the minimum CER% (minCER%) achieved with any of the reference romanizations.

4.2.1 Single Systems. Table 4 presents per language minCER% for our four modeling methods, along with micro- and macro-average performance. Again, the ByT5 results are best in 10 out of 12 languages, though unlike the Latin-to-native script direction, the ByT5 results are generally quite low variance. The Perso-Arabic writing systems (`sd` and `ur`) have notably higher error rates, presumably due to the fact that these writing systems are *abjads* with no explicit vowel markings, yet the romanizations tend

Table 4
Single word native-to-Latin transliteration minimum character-error rate percentages (minCER%) for each language plus macro averages (μ) and micro averages (all).

Lang	Pair 6g	Transformer	LSTM	ByT5
bn	4.2 (.07)	3.1 (.26)	3.2 (.26)	**2.4** (.09)
gu	2.5 (.03)	1.4 (.07)	1.3 (.07)	**1.1** (.10)
hi	4.7 (.03)	3.4 (.15)	3.6 (.10)	**2.9** (.16)
kn	1.4 (.02)	1.3 (.38)	0.9 (.06)	**0.9** (.13)
ml	1.6 (.01)	2.7 (.83)	1.3 (.28)	**1.0** (.07)
mr	2.4 (.05)	1.7 (.09)	1.8 (.09)	**1.5** (.19)
pa	4.3 (.07)	3.2 (.12)	3.5 (.12)	**3.0** (.09)
sd	8.7 (.06)	7.1 (.12)	7.5 (.14)	**6.5** (.28)
si	1.1 (.01)	**0.5** (.03)	0.6 (.04)	0.7 (.06)
ta	3.3 (.05)	3.1 (.26)	2.8 (.12)	**2.5** (.09)
te	2.8 (.02)	2.5 (.27)	**2.2** (.06)	2.4 (.09)
ur	7.7 (.03)	6.1 (.10)	6.6 (.12)	**5.9** (.33)
μ	3.7 (.01)	3.0 (.07)	3.0 (.05)	**2.6** (.08)
all	3.4 (.01)	2.8 (.09)	2.7 (.05)	**2.4** (.07)

Table 5
Single word native-to-Latin transliteration macro average minimum character-error rate percentages (minCER%) with various types of ensembling.

		Ensembling method					
Model type	None	This x5	This+ Trans.	This+ LSTM	This+ ByT5	This+all others	All except this model
Pair 6g	3.7 (.01)	3.7 (.01)	2.7 (.01)	2.6 (.01)	2.4 (.02)	2.2 (.01)	2.2 (.01)
Trans.	3.0 (.07)	2.7 (.04)		2.5 (.01)	2.3 (.02)		2.3 (.01)
LSTM	3.0 (.05)	2.7 (.02)			2.3 (.01)		2.3 (.02)
ByT5	2.6 (.08)	2.5 (.02)					2.4 (.01)

to include vowels, which must be recovered from the limited information available in this non-contextual setting. We also note that the Transformer model has relatively high variance for the four Dravidian languages in particular.

4.2.2 Ensembled Systems. As with the Latin-to-native results in the previous section, for native-to-Latin transliteration, we also derive a benefit from ensembling systems, both in error rate and variance reduction. Table 5 presents minCER% means and standard deviations for variously ensembled systems. Once again, the Pair 6g approach is too low variance to really benefit from single method ensembling. Tables D.4, D.5, and D.6 in Appendix D present the per-language minCER% results that are summarized in Table 5.

4.2.3 Earth Mover's Distance k-best Evaluation. Our primary use scenario for single word native-to-Latin transliteration (automatic romanization) is for parallel training data simulation from monolingual native script text. For that purpose, sampling from likely romanizations is a key way to simulate data that includes the kinds of variations that are observed due to the lack of Latin script orthography in these languages. However, the measure that we just reported—minCER%—only evaluates the highest probability

Table 6
Single word native-to-Latin transliteration macro average earth mover's distance *k*-best character-error rate percentages (EMDCER%) with various types of ensembling.

		Ensembling method					
Model type	None	This x5	This+ Trans.	This+ LSTM	This+ ByT5	This+all others	All except this model
Pair 6g	9.9 (.02)	9.8 (.02)	9.1 (.02)	8.7 (.01)	8.8 (.02)	8.4 (.01)	8.4 (.02)
Trans.	9.8 (.10)	9.4 (.03)		8.7 (.02)	8.7 (.02)		8.4 (.01)
LSTM	9.0 (.07)	8.6 (.01)			8.3 (.01)		8.7 (.02)
ByT5	8.8 (.15)	8.7 (.02)					8.6 (.01)

candidate, not alternative romanizations that we may sample. In Section 3.2.3 we presented the earth mover's distance based *k*-best character-error rate percentage (EMDCER%), which assesses the degree to which the distribution of the output *k*-best list matches the distribution over the reference romanizations. Table 6 presents the macro average results for this measure for various ensembling configurations. Tables D.7, D.8, and D.9 in Appendix D present the per-language EMDCER% results that are summarized in Table 6.

Several interesting things can be observed from these results. First, while the min-CER% reported in Table 5 show virtually identical performance for transformer and LSTM models, here we find that the LSTM *k*-best lists provide lower EMDCER%, both with and without ensembling, than the transformer models. We find that the four lowest EMDCER% systems include both ByT5 and LSTM output, and the best system includes only those two, hence for generating training data for full string context-aware transliteration we chose the two-system ByT5 and LSTM ensemble.

4.3 Context-aware Latin-to-native Transliteration

In the absence of direct parallel training data for full sentence transliteration, we have two broad approaches that can be taken to incorporate context into our transliteration models. First, we can derive language models from monolingual mono-script text—such as the Wikipedia text included in the Dakshina dataset—and combine the language model probabilities with non-contextual transliteration model probabilities. Our language and transliteration model combination is implemented as weighted finite-state transducer (WFST) composition of the language model—encoded as a WFST, as detailed in Section 3.5—with a word lattice encoding possible transliterations for each input word.[31] This is akin to so-called noisy channel approaches broadly used in speech recognition and related tasks (Jelinek 1998), so Roark et al. (2020) used that label as a shorthand for such approaches and we do as well. The second method is to use automatic romanization to simulate parallel training data, which can then be used to train (or fine-tune) standard sequence-to-sequence modeling methods.

Roark et al. (2020) pursue both methods for including context in their systems, and compare them with systems that transliterate each word independently, that is, non-contextually. Table 7 presents their context-aware and non-contextual systems

31 See Appendix A for further details.

Table 7
Baseline full-string Latin-to-native transliteration WER% for the dev section of the Dakshina dataset: (1) Four systems reported by Roark et al. (2020), including two non-contextual (using either Pair 6g or Transformer single-word transliteration models) and two contextual ("noisy channel" – or "NC" – and full sentence Transformer model); and (2) Two non-contextual systems using our current methods: an ensemble of three single word transliteration models: ByT5, LSTM, and Pair 6g; and that ensemble plus a word-level wordpiece language model.

Lang	Roark et al. (2020) systems				Current baselines	
	Non-contextual		Contextual		Non-contextual	
	Pair 6g	Transf.	"NC"	Transf.	Ensemble	+Word LM
bn	35.0 (.11)	32.5 (.71)	**18.6** (.02)	19.7 (.12)	30.9 (.87)	20.6 (.11)
gu	34.4 (.07)	28.1 (1.37)	16.2 (.03)	21.8 (1.36)	27.3 (.23)	**14.8** (.03)
hi	24.6 (.14)	25.0 (1.70)	**11.0** (.01)	15.8 (.24)	23.6 (1.37)	12.2 (.03)
kn	23.4 (.21)	21.0 (.27)	17.1 (.03)	18.3 (.44)	19.3 (.11)	**14.9** (.06)
ml	39.4 (.69)	37.3 (.31)	23.5 (.04)	**21.4** (.27)	35.2 (.35)	21.6 (.09)
mr	29.2 (.03)	28.4 (.62)	13.8 (.03)	13.8 (.07)	25.6 (.15)	**13.0** (.05)
pa	38.2 (.35)	36.1 (1.14)	**16.4** (.02)	19.3 (.04)	34.8 (.52)	18.1 (.07)
sd	55.3 (.13)	63.5 (1.38)	**26.1** (.07)	37.3 (1.20)	50.4 (1.29)	29.4 (.53)
si	37.0 (.03)	35.9 (.96)	**20.3** (.02)	23.0 (.77)	34.0 (.17)	23.4 (.16)
ta	30.7 (.25)	31.9 (.95)	19.3 (.04)	18.9 (.08)	27.2 (.19)	**17.6** (.08)
te	27.6 (.06)	26.4 (.22)	17.0 (.02)	18.9 (.10)	23.3 (.20)	**15.2** (.09)
ur	33.8 (.08)	44.5 (3.25)	**12.5** (.08)	19.3 (.47)	23.4 (.28)	14.5 (.05)
μ	34.1	34.2	**17.7**	20.6	29.6 (.09)	17.9 (.06)
all					30.3 (.13)	18.1 (.07)

word-error rates[32] for all of the Dakshina languages, as well as the macro-average (μ). We also provide the WER% from two of our own non-contextual baseline systems. The first is simply the ensembled Latin-to-native script transliteration model, consisting of ByT5, LSTM, and Pair 6g components.[33] In addition, we show results using that transliteration model combined with a word-level wordpiece language model, which achieves substantial improvements over using just the transliteration model ensemble itself, despite including no sentential context. In fact, the macro-average for that system nearly outperforms even the best context-aware system from Roark et al. (2020)—within 0.2% absolute—which can be attributed to improvements in the non-contextual transliteration models and the inclusion of word-level likelihood, as calculated via the wordpiece LM.

Note that we now have the macro-averaged WER% results from the first three systems in the graph presented in Figure 1 in the Introduction: (a) the best single method non-contextual transliteration model ensemble result (LSTM at 32.5); (b) the best multiple method non-contextual transliteration model ensemble (29.6); and (c) that transliteration model ensemble combined with the word-level (non-contextual) language model (17.9). With these baseline non-contextual results, we can move on to

32 Recall that we are pursuing their "whitespace evaluation" method, hence the WER% results come from that portion of Table 4 in Roark et al. (2020).

33 Table D.10 in Appendix D presents the single method ensembled CER% for each of the three modeling approaches that make up this ensemble, the best of which is the LSTM, which yields a macro-average (μ) CER% of 32.5.

Table 8
Full-context Latin-to-native transliteration WER% for the dev section of the Dakshina dataset: Our version of a "noisy channel" style model, with the single-word transliteration ensemble (ByT5+LSTM+Pair6g) plus a contextual wordpiece LM; mT5 and ByT5 in both base and large configurations; and a full-string ensemble of the "noisy channel" ("NC"), mT5-large, and ByT5-base models.

Lang	"Noisy channel"	mT5		ByT5		"NC" + ByT5-base + mT5-large ensemble
		base	large	base	large	
bn	18.4 (.11)	16.6 (.08)	16.0 (.16)	18.0 (.11)	18.8 (.22)	**14.8** (.08)
gu	14.2 (.04)	14.3 (.11)	13.6 (.18)	15.9 (.15)	16.9 (.31)	**11.8** (.10)
hi	10.4 (.02)	11.5 (.20)	11.1 (.20)	15.4 (.16)	15.0 (.16)	**10.4** (.10)
kn	14.7 (.04)	14.0 (.05)	13.6 (.13)	16.6 (.07)	17.1 (.16)	**12.8** (.05)
ml	20.2 (.05)	21.7 (.15)	20.8 (.27)	22.8 (.13)	24.0 (.13)	**18.5** (.14)
mr	12.2 (.07)	11.7 (.08)	11.1 (.17)	12.1 (.08)	12.9 (.15)	**10.0** (.08)
pa	15.4 (.03)	15.6 (.19)	15.2 (.19)	17.7 (.23)	18.2 (.27)	**14.1** (.11)
sd	26.5 (.34)	28.4 (.60)	27.1 (.42)	29.2 (.20)	29.8 (.44)	**24.2** (.10)
si	21.4 (.14)	30.9 (.06)	30.6 (.10)	18.8 (.11)	19.6 (.09)	**17.1** (.07)
ta	15.9 (.04)	17.3 (.38)	16.6 (.24)	19.8 (.26)	20.9 (.38)	**14.8** (.16)
te	14.4 (.03)	16.3 (.06)	15.6 (.11)	16.5 (.08)	17.1 (.23)	**12.9** (.05)
ur	12.2 (.11)	13.4 (.17)	13.0 (.18)	13.8 (.12)	13.9 (.13)	**11.8** (.18)
μ	16.3 (.05)	17.7 (.11)	17.0 (.10)	18.0 (.05)	18.7 (.13)	**14.4** (.04)
all	16.4 (.06)	17.8 (.11)	17.2 (.10)	18.2 (.06)	18.7 (.13)	**14.5** (.04)

the question of how much performance improvement can be achieved by including contextual information.

Table 8 presents the WER% using six context-aware systems, including: our version of a "noisy channel" approach; two configurations each for ByT5 and mT5; and an ensembling of the the best performing configurations of mT5 and ByT5 with the noisy channel approach ("NC"). A few things are clear from these results. First, while ByT5 models are superior for the shorter-length single-word transliteration, mT5 models, which use sub-word tokenizations rather than bytes, generally perform better than ByT5 on the full sentence task, with lengthier input sequences. The one exception to this is Sinhala, which we later demonstrate in Appendix C to be due to the SentencePiece tokens used by mT5. One danger of using pretrained models for tasks such as this is that even relatively minor design choices of the pretraining influence the applicability of the model to new tasks, and in the current case certain Unicode symbols (zero-width joiner and zero-width non-joiner) were omitted from model vocabulary (see Appendix C for more details). Such model design decisions are beyond our control, and the mT5 model is generally very useful, hence we rely upon multiple system ensembling to ameliorate such problems.

Additionally, we see solid improvements for mT5 with the large configuration over the base configuration, while ByT5 does not benefit from its larger configuration. The "noisy channel" model yields the best performance of any of these stand-alone contextual systems. Ensembling the best mT5 and ByT5 configurations with the noisy channel approach, however, yields the best performance of any system (including the baselines) in every language, with an overall macro-average WER% of 14.4. This result underlines how critical ensembling is, not only for improving the overall accuracy, but for smoothing over outliers that may arise with particular methods, such as Sinhala

with mT5. The nature of pretrained models are such that this sort of ensembling is indispensable for both low error rates and low variance.

The 14.4 macro-averaged WER% is a 3.5% absolute (20% relative) reduction from our best non-contextual baseline, thus making explicit how much system improvement is due to contextual information. Interestingly, four of the five languages that have the least relative error rate reduction due to sentential context are the four Dravidian languages, which are highly inflected, hence contain relatively many within-word dependencies (Kumar et al. 2017; Steever 2019).

4.4 Experiments with Caching

In Section 3.6.2 we presented methods for constructing a cache of single word transliterations, which can be used to avoid expensive ensembling at time of inference—instead relying on an offline ensemble to construct the cache, and using a single model for input words that do not have a cache entry. As stated there, we use a non-neural Pair 6g transliteration model for words outside of the cache, and present trials with (1) no language model; (2) a word-level (non-contextual) language model; and (3) a sentence-level (context-aware) language model. All language models used were the same as in earlier experiments for the same conditions.

The experiments were run on a single machine using only CPUs. The machine was an AMD EPYC 7B12 with 2x12 64-bit 2250Mhz CPU cores and 192G of RAM. The inference engine has some multi-threading, and we allowed up to 8 cores to be used by the process. Only one inference process was run at a time, and we ran each model 5 times, taking the minimum clock time for a model as the time to process the input. As before, we have 5 different models for each language, so we can calculate means and standard deviations of WER% and processing speed.

Table 9 presents WER% for our three caching conditions, and for the offline full "noisy channel" model for comparison. On average, relying on the single Pair 6g transliteration model for tokens that fall outside of the cache increases WER by just

Table 9
Full-context Latin-to-native transliteration WER% for the dev section of the Dakshina dataset for online cache methods, compared to fully offline "noisy channel" result, for each language plus macro averages (μ) and micro averages (all).

Lang	Online using transliteration cache			Full "noisy "channel"
	no LM	Word LM	Sent LM	
bn	31.8 (.47)	21.0 (.08)	18.7 (.15)	18.4 (.11)
gu	29.5 (.27)	15.2 (.06)	14.6 (.07)	14.2 (.04)
hi	22.9 (.90)	12.4 (.03)	10.7 (.04)	10.4 (.02)
kn	20.7 (.15)	15.5 (.08)	15.1 (.05)	14.7 (.04)
ml	36.7 (.12)	22.1 (.07)	20.7 (.03)	20.2 (.05)
mr	26.3 (.17)	13.5 (.01)	12.7 (.06)	12.2 (.07)
pa	35.5 (.42)	18.3 (.12)	15.5 (.06)	15.4 (.03)
sd	51.4 (.96)	30.7 (.40)	27.8 (.48)	26.5 (.34)
si	34.5 (.33)	23.4 (.17)	21.5 (.07)	21.4 (.14)
ta	28.2 (.13)	18.7 (.08)	17.0 (.05)	15.9 (.04)
te	24.2 (.11)	15.6 (.13)	14.8 (.02)	14.4 (.03)
ur	24.2 (.34)	14.6 (.06)	12.2 (.06)	12.2 (.11)
μ	30.5 (.10)	18.4 (.05)	16.8 (.04)	16.3 (.05)
all	31.1 (.14)	18.6 (.06)	16.8 (.05)	16.4 (.06)

Table 10
Statistics per language for online cache methods, including coverage and characters-per-second (x1,000) under different conditions.

Lang	Cache token coverage (%)	Characters-per-second (x1,000)		
		no LM	Word LM	Sent LM
bn	91.7	1.68 (.05)	1.61 (.05)	0.55 (.04)
gu	83.6	1.51 (.04)	1.44 (.05)	0.50 (.04)
hi	96.5	2.10 (.01)	2.05 (.01)	0.68 (.004)
kn	86.8	1.86 (.02)	1.49 (.02)	0.59 (.03)
ml	76.3	0.49 (.02)	0.47 (.003)	0.15 (.001)
mr	91.6	1.43 (.01)	1.36 (.02)	0.50 (.004)
pa	93.1	2.90 (.02)	2.03 (.02)	0.79 (.006)
sd	79.6	3.67 (.05)	3.26 (.04)	0.88 (.05)
si	92.2	3.14 (.02)	2.19 (.02)	1.04 (.01)
ta	79.9	0.55 (.03)	0.51 (.03)	0.17 (.01)
te	86.5	1.50 (.01)	1.23 (.12)	0.47 (.02)
ur	95.6	2.67 (.02)	2.56 (.02)	0.62 (.004)

about 0.5% absolute compared to the full noisy channel approach, which itself is roughly 2% higher than ensembling with the neural contextual models. The non-contextual word-based language model further increases WER% by approximately 1.5% WER. Table 10 presents total cache token coverage for each language, as well as the characters-per-second that are processed under each condition. Languages vary in their throughput—from relatively slow Dravidian languages (ml, ta) at about 500 characters per second with no LM, to much speedier Indo-Aryan languages using the Perso-Arabic script (sd, ur) at between 2,500 and 3,500 characters per second. Using the non-contextual language model slows things down just a bit relative to using no language model at all, but remain roughly 3 times faster than using the fully contextual language model. Cache token coverage is quite good, with only the highly inflected Dravidian languages (and the lower-resource Sindhi) falling below 80%, while half of the languages are well above 90%.

Appendix E contains two graphs in Figure E.1 showing type and token coverage of the cache for each language as the size of the cache increases.

4.5 Analysis

In this section, we provide both quantitative and qualitative analyses to help understand what kinds of errors are being fixed as our models improve, and under what circumstances context and/or ensembling improves performance. We additionally examine the impact of mC4 as a corpus on system performance, and close with some still pending issues.

4.5.1 Improvements Due to Frequent or Infrequent Types. Here we present a brief analysis that highlights certain language differences in how the error rate reductions were achieved for each language between the best non-contextual transliteration model ensembles—system (b) in Figure 1 in the Introduction—and the final best performing context-aware multi-system ensemble, namely, system (e) in Figure 1. Since the latter system is ensembled using a method that requires the number of output words to match the number of input words (which the non-contextual systems also do as a matter

of course), we can straightforwardly compare the output of systems (b) and (e) for a particular language at each input word position. Let b_i be the output of system (b) at word position i in the text collection; e_i the output of system (e) at that word position; and r_i the reference output transliteration at that position. If $b_i \neq e_i$, we can call this an output word difference. If $b_i \neq e_i$ and $e_i = r_i$, then this counts as a "win", that is, the switch from b_i to e_i resulted in a reduction in the number of errors. If $b_i \neq e_i$ and $b_i = r_i$, this is a "loss," that is, an increase in the number of errors. If neither of the different system outputs matches the reference, the difference is neutral. We can count the total wins and losses accrued across the whole development set when output word u in system (b) is replaced with word v in system (e), and use that to characterize the net wins (i.e., wins minus losses) associated with that unique output word difference pair. Formally, for a development set consisting of N words, we define the net wins of a pair of words $u \neq v$ as follows:

$$\begin{aligned} W(u,v) = &\mid \{1 \leq i \leq N : b_i = u \wedge e_i = v \wedge r_i = v\} \mid \\ &- \mid \{1 \leq i \leq N : b_i = u \wedge e_i = v \wedge r_i = u\} \mid \end{aligned} \tag{8}$$

Let UV be the set of word pairs (u, v) such that system (b) output word u in some position where system (e) output the (different) word v, namely,

$$UV = \{(u,v) : u \neq v \wedge \exists i \text{ where } b_i = u \wedge e_i = v\} \ . \tag{9}$$

Figure 3 plots the mean (across 5 trials) net wins per $(u, v) \in UV$ for each language. The four languages that have the lowest score for this measure are the highly inflected Dravidian languages, indicating that the word-error rate reductions in those languages are the result of relatively many distinct word differences between the systems. None of those languages has a score greater than 1.25 per unique difference pair, while the lowest scoring Indo-Aryan language (Marathi) has a score of 1.8. This is consistent with the highly inflected nature of those languages, which would lead to a higher type/token ratio and relatively fewer high frequency types. Indeed, if we compare the 3 highest net win output word difference pairs in Malayalam (`ml`, a Dravidian language) and Marathi (`mr`, an Indo-Aryan language), we can see this explicitly. Both languages have very similar net win totals in the development set, but the top three pairs in Marathi (अहे → आहे, अणी → आणि, वा → व) have mean net wins of 665.2, 502.0, and 394.4, respectively, in the development set, while the top three pairs in Malayalam (നിന്നു → നിന്ന്, ഇതു → ഇത്, എന്നി → എന്നീ) have net wins of 66.8, 61.0, and 53.8, respectively. We also note that two of the three languages with the highest net wins per unique output word difference in Figure 3 use a Perso-Arabic script, which, as *abjads*, will tend to have a lower type/token ratio, hence typically relatively more high frequency words in the sample.

One other interesting thing can be noted from the example pairs presented in the previous paragraph. Two of the Malayalam examples (നിന്നു → നിന്ന്, ഇതു → ഇത്) and one of the Marathi examples (वा → व) are conversions from a less frequent (but still valid) spelling of a word to a more frequent spelling of the same word.[34] Counting from the native script Wikipedia corpus from the Dakshina dataset, നിന്ന് is nearly 5 times more frequent than നിന്നു; ഇത്, is over 7 times more frequent than ഇതു; and व is over

34 Note that, while നിന്നു and നിന്ന് mean and are read as the same things in many contexts, നിന്നു also has the meaning of "stood"—as in past tense of "sit"—which the other word never has.

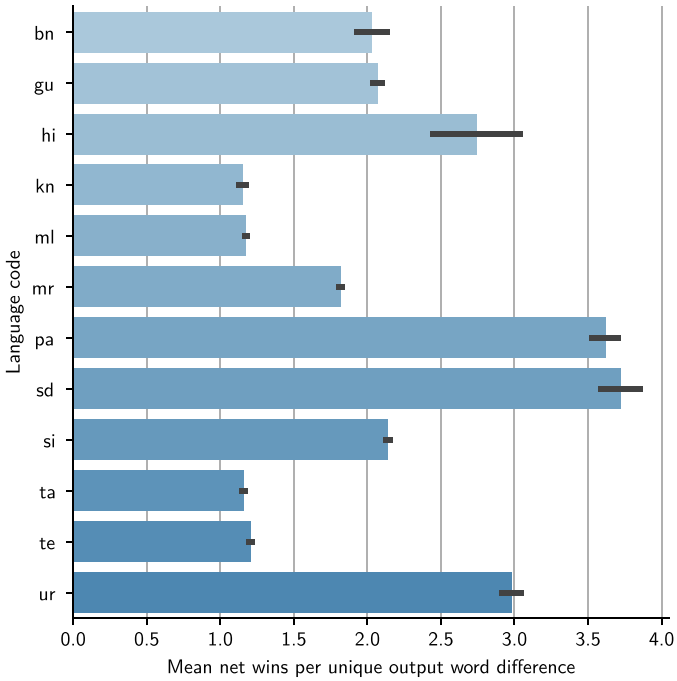


Figure 3
Comparison of the best multiple method non-contextual transliteration model ensemble with the best performing context-aware ensembled system, in terms of net wins per unique output word differences. The means and standard deviations over five runs are shown.

20 times more frequent than वा . Thus at least some of the error rate reductions for these languages involve selecting more canonical spellings for words that have more than one valid spelling.

4.5.2 Improvements Due to Context Awareness. In this section, we will present examples of how context awareness improves transliteration in some languages, by comparing the output of the noisy channel system using non-contextual (word-level) language models versus those using context-aware (sentence-level) language models. Hindi and Punjabi were two of the languages with the largest relative decrease in WER% when including context-aware language models, so we will look at some common ways in which the context is helping for those languages.

The largest contributor to the improvement in Hindi is improved handling of the confusable Hindi words की and कि, both of which are romanized in our dev set between 95% and 99% of the time as "ki." The former (की) is a postposition meaning "of", e.g., "संजय की पत्नी" ("wife of Sanjay" or "Sanjay's wife"). The latter (कि) is a conjunction meaning "that", for example, "विक्रम ने कहा कि" ("Vikram said that"). Without context, "ki" is always transliterated as the more frequent की, but context helps disambiguate, so that with context-aware models this is correctly transliterated as कि 416 times and incorrectly only 19 times, a large overall win. Similar contextual disambiguation of common confusables account for much of the gain, hence the relatively high wins per difference in Figure 3. Similarly, in Punjabi, the related postpositions ਵਿੱਚ and ਵਿਚ are both romanized as "vich" around 99% of the time, and context-aware modeling yields a 5-1 win-loss ratio when disambiguating between them.

Another class of context-aware wins in Hindi come from the same English-origin word being spelled conventionally slightly differently within certain collocations such as proper names. For example, the proper name "अमेरिकन एक्सप्रेस" (American Express) is conventionally written with the "Express" part of the name ("एक्सप्रेस") spelled differently than in other common uses of the term, which typically spell it एक्स्प्रेस (with an extra *virama* symbol), such as rail-related terms "मुंबई एक्स्प्रेस" ("Mumbai Express") or "मेल एक्स्प्रेस" ("Mail Express"). Whether such conventions are as strictly adhered to as they seem to be in Wikipedia is hard to say, but in any case the context-aware models are able to capture the correct spelling for such common collocations.

4.5.3 Improvements Due to Ensembling. In this section, we will present examples of how ensembling improves transliteration in some languages, by comparing the output of the noisy channel system using context-aware (sentence-level) language models and the full ensemble with mT5 and ByT5 contributions. Sinhala and Bengali were two of the languages with the largest relative decrease in WER% when ensembling these context-aware models, so we will look at some examples where ensembling is helping for those languages.

We have already noted one way in which Sinhala is helped by ensembling, since the mT5 model is using SentencePiece tokenization that does not include zero-width joiner, which is commonly used in that writing system (see Appendix C for details). Due to this catastrophic mismatch, the mT5 system is much worse for this language than the others, despite otherwise often being the best of all single context-aware systems. Even so, we find that an ensemble that includes this system provides strong reductions in WER% from the best single system for this language, suggesting that despite its systematic errors, useful signal is found in its output.

One way in which ensembling appears to be helpful is when the number of output tokens differs from the number of input tokens, which can occur relatively frequently depending on the language and system, as shown in Table 11. Other than the Sinhala case in mT5, which we've already discussed, we see very frequent issues in Sindhi in both ByT5 and mT5 systems, and Urdu and Bengali have quite high values for all

Table 11
Per-language percentage of sentences in the full string dev set for which T5-based systems produced a different number of output tokens than the number of input tokens.

Language	ByT5 systems		mT5 systems	
	Base	Large	Base	Large
bn	21.2	21.2	19.4	19.4
gu	12.7	12.7	7.6	7.6
hi	11.3	11.3	6.5	6.5
kn	19.1	19.1	15.6	15.7
ml	7.1	7.0	5.4	5.5
mr	4.7	4.7	4.0	4.3
pa	9.4	9.4	5.6	5.6
sd	44.1	44.1	44.1	44.0
si	18.5	18.5	62.3	62.5
ta	6.5	6.5	3.1	3.1
te	8.2	8.2	11.9	12.0
ur	26.7	26.7	26.4	26.5

of the systems. Other languages have a relatively small percentage of such outputs. While people do produce output that has a different number of tokens than the input, this is a relatively rare occurrence, so that, on balance, having such a constraint is beneficial versus not having it, particularly in the face of such frequent divergence by the pretrained models.

While the *n*-gram modeling is useful for disambiguating items in collocations or frequent constructions, as we saw in the previous section, the neural models provide much more powerful language modeling constraints on the system which can be helpful. For example, in Sinhala, the word "වැනි" (meaning "such as" or "like") can be variously romanized as "weni" or (somewhat less commonly) "veni", much like "වෙනි" (which is used in formation of ordinal numbers, e.g., "2nd"). Predicting when ordinal number formation is likely versus a relatively specialized connective phrase requires more than collocations, hence the usefulness of the neural models. For this example, 72 corrections were made with no regressions. Hence the "noisy channel" model is helpful in controlling for issues related to writing system mismatch (i.e., lack of zero-width joiner and zero-width non-joiner in mT5), while the neural models can provide more subtle disambiguation than is achieved by the *n*-gram based system.

The reasons for improvements to Bengali are a bit harder to tease apart, but one common class of errors that are repaired in ensembling are related to short romanizations ending in the letter -y, such as "dey," "pray," "ney," "jay," and "ray." These collectively account for over 100 wins on the dev set in the ensembled system in Bengali relative to the noisy channel, with no regressions. These short romanizations are difficult due to a couple of factors. First they transliterate very straightforwardly to some foreign-origin words (e.g., the English word "pray" which is typically written প্রে in the Bengali script) or proper names (e.g., "রিও দে লা প্লাতা" for "Rio de la Plata"). Hence, these can be likely competitor transliterations to the more commonly intended words. These likely intended words, however, have the additional complication of having word-final schwa deletion that is not explicitly marked in the orthography (Choudhury, Basu, and Sarkar 2004; Johny and Jansche 2018). For example, the Bengali word প্রায় ("almost"), which is also typically romanized as "pray," does not explicitly mark the final vowel deletion with *virama*. Without final schwa deletion, one would expect the word to be romanized as "praya." The system needs to learn when schwa deletion is likely, in order to make the match between the romanization (which typically omits the vowel) and its target spelling. The neural models unsurprisingly do a better job of capturing these kinds of patterns than the noisy channel model, hence the ensemble benefits.

4.5.4 Impact of mC4. In this section, we investigate the use of mC4—the pretraining data for both mT5 and ByT5 models—for this task in two ways. First, we examine whether we can leverage the additional training data in each language to build better language models and thus improve the noisy channel approach. Second, we ask whether there might have been any influence of Wikipedia data having been included in mC4 on the fine-tuned mT5 and ByT5 results.

We follow the approach outlined in Section 3.5.1 to train 4-gram Kneser-Ney word-piece models from all the mC4 text available in each of our 12 languages, after doing the same "whitespace normalization" on the text. New wordpiece models are created from these corpora prior to *n*-gram model estimation. Given the size of the Hindi and Bengali sub-corpora in mC4, to control for model size, we pruned 4-grams from these models that only occurred once in the corpus; for other languages the language models are left unpruned.

Table 12
WER% of context-aware "noisy channel" model when using language model trained from either the Dakshina Wikipedia training data (repeated from Table 8) or mC4.

Language	Dakshina Wikipedia LM	mC4 LM
bn	18.4 (.10)	23.3 (.13)
gu	14.2 (.04)	15.1 (.02)
hi	10.4 (.02)	11.3 (.02)
kn	14.7 (.04)	28.6 (.05)
ml	20.2 (.06)	33.1 (.07)
mr	12.2 (.07)	18.2 (.04)
pa	15.4 (.03)	13.5 (.05)
sd	26.5 (.35)	24.1 (.40)
si	21.4 (.14)	23.3 (.18)
ta	15.9 (.05)	25.9 (.05)
te	14.4 (.03)	26.9 (.06)
ur	12.2 (.11)	11.9 (.10)
μ	16.3 (.05)	21.3 (.06)
all	16.4 (.06)	20.0 (.07)

Table 12 presents side-by-side WER% comparison between our existing "noisy channel" system, using a language model trained on the training portion of the Dakshina Wikipedia text, and one using a language model trained on mC4. From this table we can see that the mC4 trained language models are overall much less effective than the Wikipedia trained ones. There were modest improvements for two of the languages with the lowest amount of Wikipedia training data in Dakshina—Punjabi (pa) and Sindhi (sd)—as well as Urdu (ur), but otherwise the results were relatively poor. In particular, the Dravidian languages (kn, ml, ta, te) achieved quite poor performance using this data. One might hypothesize that strong domain and or register (e.g., formal vs. informal) mismatch between Wikipedia and what is otherwise included in mC4 is to blame for these generally poor results.

Given that mC4 is harvested from online text, which includes Wikipedia, one question that arises is whether the models that are pre-trained on this text (mT5, ByT5) achieve any benefit from potentially having seen the output strings (i.e., in the native script) of our dev set during pre-training. To examine this, we made use of the URLs provided by both mC4 and Dakshina to determine which strings in the development set come from documents that might have been included in mC4.

The Wikipedia URLs come in several forms shown in Table 13. For a given language, such as bn, Wikipedia URLs are canonically of the form 1 shown in the table, however

Table 13
Examples of various Wikipedia URLs.

Type	URL example
1.	https://bn.wikipedia.org/wiki/TITLE
2.	https://bn.m.wikipedia.org/wiki/
3.	https://web.archive.org/web/INDEX/http:/ml.wikipedia.org/w/index.php?title=TITLE
4.	https://hi.wikipedia.org/wiki/index.html?curid=INDEX

Table 14
Percentage of dev set sentences from documents possibly included in mC4, for each language.

Language:	bn	gu	hi	kn	ml	mr	pa	sd	si	ta	te	ur
Percentage:	84.7	79.5	71.8	81.2	60.3	65.2	56.8	45.0	65.9	59.5	67.9	63.6

Table 15
Dev set macro-averaged WER% for strings from documents included in mC4 versus strings from documents not included in mC4, as well as relative and absolute WER% differences between these results for each system.

System		All sentences	Documents in mC4	Documents not in mC4
"Noisy channel"		16.3 (.05)	16.4 (.05)	16.0 (.05)
mT5	Base	17.7 (.11)	17.6 (.10)	17.7 (.11)
	Large	17.0 (.10)	17.0 (.11)	17.0 (.10)
ByT5	Base	18.0 (.05)	18.2 (.06)	17.5 (.07)
	Large	18.7 (.13)	18.9 (.13)	18.1 (.13)
Full Ensemble		14.4 (.04)	14.4 (.04)	14.3 (.03)

there can be variants depending on how the page was accessed, e.g., type 2, and we normalize these to ensure we find titles in either format. Additionally, there are aggregators (type 3) that store or point to Wikipedia pages, where the numerical `INDEX` and original Wikipedia `TITLE` are stored in ways idiosyncratic to the particular aggregator. We exhaustively reviewed the aggregators found in the mC4 URLs to ensure that we found all sources pointing to particular documents. Finally, Wikipedia pages can be accessed by an index (shown as type 4 in Table 13), and we also collected these indices.

The Dakshina dataset provides all of the URLs, page IDs, and revision IDs for the data included in the dataset, and we collected the information for the documents from which sentences in the development set were extracted.[35] We count any sentence as possibly being included in mC4 if either the page title or page ID is found from the set extracted from the mC4 URLs.[36] Not every sentence from these documents end up in the mC4 text collection, but we wanted to cast a wide net and exclude sentences that may have been included. Table 14 presents the percentage of dev set sentences that were from documents that matched the mC4 list, hence were possibly included in that corpus.

To examine the impact of inclusion in mC4 on the results, in Table 15 we compare the macro-averaged WER% across languages for dev set sentences possibly included in mC4 versus those from documents not in the corpus. We include the "noisy channel" model, which is trained only on the Dakshina Wikipedia set, hence has not been exposed to these documents, as the means to assess how different the error rates for the two sets of sentences are with no exposure. We use macro-averaged WER% due to the variability in number of sentences in each set per language, which would have reduced

35 Recall that the native-script Wikipedia training data that our language models are trained from are disjoint from the documents that were used to extract validation data, so we only need to track the documents used to create the validation sets.

36 See `https://github.com/google-research/google-research/tree/master/context_aware_transliteration` for resources and scripts used to perform this matching.

comparability if we had used a micro-average. The set of sentences from documents not in mC4 seems very slightly harder, as evidenced by the noisy channel and ByT5 models, while mT5 and the ensemble show essentially the same performance on both sets of sentences.

These results—coupled with earlier results showing that (1) the noisy channel model provides better performance than any of the pretrained models, and (2) training a language model on mC4 degrades noisy channel performance—suggest that, if any system gains are attributable to having seen validation set output sentences as part of pretraining, such gains are extremely modest.

4.5.5 Remaining Issues. We conclude this analysis section by mentioning some remaining issues that both continue to cause errors and for which clear generalizations can be made. One continuing source of errors are acronyms, which in these languages are derived from Latin script acronyms and are produced in the native script as sounded-out letter sequences. For example, the Indian political party commonly referred to as BJP ("Bharatiya Janata Party") is written in Hindi as भाजप, which sounds out the Latin script letters ("beejaypee"). While such common acronyms are easy to memorize, the task of deciding when something is an acronym and should be produced as a letter sequence is difficult.[37] For example, in Malayalam, the sentence

സംസ്ഥാനങ്ങൾ സംസ്ഥാന ജിഎസ്ടി നിയമവും എസ്ജിഎസ്ടി പാസാക്കിക്കഴിഞ്ഞിരുന്നു

contains two related acronyms ജിഎസ്ടി (GST) and എസ്ജിഎസ്ടി (SGST).[38] The final ensembled model gets the first acronym correct, but for the second one it produces സിസ്റ്റ് ("cyst"), that is, a word with somewhat similar spelling. This is a complex phenomenon that exists in all of the languages included in the Dakshina dataset.

Acronyms in isolation remain an issue, but the problem can be particularly acute when they are inflected in the language, namely, when the acronym is the root within the fully inflected form. For instance, the Malayalam phrase "ഐഎഎസിലേക്ക് തിരഞ്ഞെടുക്കപ്പെട്ടു" ("selected to IAS")[39] from the dev set is romanized as "IASlekku thiranjedukkappettu", and our system incorrectly transliterates the first word as "ഐഎസ്എൽഎക്ക് ". Most of our systems apply their transliteration on lowercase input, hence losing some of that potentially useful information. For Dravidian languages such as Malayalam, inflection of English origin words in general (and acronyms based on Latin script phrases) is not uncommon and remain an issue for these systems.

An additional issue for English origin words is that in some cases there may be multiple acceptable ways to spell the word, and some fraction of the errors arise from such a mismatch. For example, the Malayalam sentence

ഇന്റർനാഷനൽ കമ്മിറ്റി ഫോർ ഡൈയിങ് ആൻഡ് ക്ലീനിങ്

is actually just English ("International Committee for Dyeing and Cleaning"), and the word for "dyeing" can be acceptably spelled either ഡയിംഗ് or ഡൈയിങ്. Mapping to a canonical version (reading normalization) would help to remove these kinds of spurious errors.

37 For example, see Schiffman (2008) for discussion of the prevalence and borrowing peculiarities of English acronyms in modern Tamil.

38 "State Goods and Service Tax."

39 IAS is the Indian Administrative Service.

Table 16
Full-string Latin-to-native transliteration WER% for the test section of the Dakshina dataset: (1) A non-contextual system: an ensemble of three single word transliteration models: ByT5, LSTM and Pair 6g, plus a word-level wordpiece language model; and (2) a contextual system: a full-string ensemble of the "noisy channel," mT5-large, and ByT5-base models.

Language	Non-contextual ByT5, LSTM, Pair 6g Translit Ensemble + Word-level LM	Contextual "Noisy channel" + ByT5-Base + mT5-Large Ensemble
bn	20.5 (.09)	**15.0** (.06)
gu	14.5 (.10)	**11.5** (.16)
hi	12.4 (.02)	**10.6** (.09)
kn	15.0 (.03)	**12.8** (.07)
ml	21.2 (.03)	**17.9** (.09)
mr	12.6 (.05)	**9.7** (.06)
pa	18.3 (.11)	**14.2** (.09)
sd	29.1 (.67)	**24.1** (.09)
si	23.9 (.12)	**17.6** (.08)
ta	17.3 (.12)	**14.7** (.20)
te	15.3 (.13)	**13.1** (.02)
ur	14.8 (.04)	**12.1** (.17)
μ	17.9 (.07)	**14.4** (.04)
all	18.1 (.08)	**14.6** (.04)

4.6 Test Partition Results

As one final experiment, we report transliteration results on the test partition of the Dakshina dataset for each language. Roark et al. (2020) only reported on the development set, which is what we compared to above. However, since we examined many systems and how they performed on the development set, it is possible that we overtuned to that particular data. As far as we know, no results have been reported in the literature on this partition yet, so we present our best performing non-contextual system (ensemble of transliteration models combined with a word-level wordpiece language model) and our best performing context-aware system (ensemble of three context-aware systems), to verify that similar behavior is observed. Table 16 presents these two systems on all of the languages, as well as macro-averaged (μ) and micro-averaged (all) WER% results.

We find that both the context-aware and non-contextual systems follow similar patterns on the test set as on the development set, with WER% differing from the development set results by at most ±0.6% absolute for any language, and differences of micro-average and macro-average less than 0.1% absolute. In addition to providing context-aware and non-contextual baselines for this test partition, we replicate the result discovered on the development set, that the use of context provides an average approximately 20% relative reduction in error rate.

5. Conclusion

We have presented an extensive experimental exploration of context-aware full sentence transliteration in the typical scenario of lacking full sentence parallel data. We find that both "noisy-channel" approaches, relying on non-contextual transliteration models and language models, as well as pretrained sequence-to-sequence models fine-tuned on

simulated parallel training data, contribute to the lowest error rate (and lower variance) for this task. We also found that pruning a small fraction of outliers from the training set improved fine-tuning of pre-trained models, sometimes dramatically so, as with Tamil, and this contributed to the excellent noisy channel performance, which was the best single context-aware system. We establish, through careful control, that a non-contextual (word-level) language model provides very large error rate reductions, and is responsible for approximately 80% of the relative error rate reduction between using just non-contextual transliteration models and the best fully context-aware system. This is of particular importance in use scenarios where low latency is required, as it provides a relatively low error rate approach while processing each word independently, which is straightforwardly parallelizable, as we demonstrated in the cache-based results.

We use ensembling for all stages of processing, which results in lower error rates and variance across all languages in the set—not just better on average, but better across the board. This is demonstrated to be of particular importance with pretrained models. Finally, we demonstrate the key importance of context, the use of which ultimately provides substantial error rate reductions across the board, though not as substantial as the results from Roark et al. (2020) would suggest in their results.

There are many directions for future work. We would like to extend the work to larger and more diverse collections of languages and scripts, though that requires at least some full sentence parallel data for system development and validation. It could be possible to mine full-sentence romanized and native data—not necessarily parallel but still useful for (for example) building language models—from multilingual corpora such as CC-100 (Conneau et al. 2020) and mC4 (Xue et al. 2021). Data quality in these corpora can be very low, as documented recently (Kreutzer et al. 2022; Madhani, Khapra, and Kunchukuttan 2023; Doddapaneni et al. 2023), so validating and curating the data would be an interesting research project on its own. Romanized data coverage in particular is also limited. Of the 12 Dakshina languages, mC4 only contains romanized Hindi—see Nielsen, Kirov, and Roark (2023) for evidence that this is actually a mixture of Hindi and Urdu—and CC-100 contains only five (Bengali, Hindi, Tamil, Telugu, and Urdu).

We hope that some of the lessons drawn in this article, about highly inflected languages or *abjad* writing systems, will be valuable as new languages are investigated. We suspect that some additional system improvements may be achieved by optimizing certain meta-parameters that we opted to leave as defaults, such as mixing weights during ensembling. Issues around script and text normalization are also interesting, since both Brahmic and Perso-Arabic scripts come with encoding complexities beyond those encountered in most alphabetic scripts (Johny et al. 2021; Gutkin et al. 2022a,b). Additionally, as we have seen, some words have multiple valid spellings in the native scripts of these languages, which would be useful to account for in evaluation. Finally, we have seen that pretrained large language models are an important component of our highest quality systems, and these models are advancing quickly in size and quality, so we intend to continue working with newer and more powerful such models. For this article, they did not achieve stand-alone performance levels sufficient to suggest abandoning the alternatives altogether, however, there are many ways to try to exploit the power of such models now and in the future, and this will continue to be a focus.

Appendix A. Details of Finite-state Methods

A pair n-gram language model is an n-gram model over pair symbols, i.e., composite symbols of the form $x{:}\,y$ where x and y are individual symbols (Galescu and Allen

2001; Bisani and Ney 2002; Chen 2003). Figure 2 on page 488 presents the schematic of an FST-representation (Roark et al. 2012) of a pair trigram model, using the புலி/*puli* Tamil example from earlier in the paper. Let's assume that the words get aligned at the individual Unicode codepoint level as:

ப:p ு:u ல:l ி:i

These colon-delimited *pair* symbols then become tokens for training language models, which end up with an automaton structure such as that shown in Figure 2. States in the automaton represent conditioning histories (shown with text representing the history in the figure for ease of interpretation). Transitions leaving the states are labeled with the next token, and their destination states encode the updated history. Backoff arcs (labeled here with ϵ) go from higher order states to their lower-order backoff state, where the history loses the most distant token. The unigram state (with empty history) terminates the backoff path. Each transition is also weighted (weights omitted in the figure for clarity), so that the correct *n*-gram probabilities are accrued.

This *n*-gram model automaton can be straightforwardly converted to a finite-state transducer by splitting the pair symbols into an input symbol and an output symbol and leaving ϵ labels (and all weights) unchanged. Note that either script can be on the input or output side, i.e., the model can map in either direction. Assuming that the Latin script is on the input side, the pair symbol "ப:p" would end up with "p" on the input side and "ப" on the output side of the transducer transitions. Let T represent the resulting transducer and P the original *n*-gram model automaton.

To find transliterations of an input string, we first encode the input string as a linear automaton, which represents a single path labeled by individual Unicode codepoints of the string. Thus, for our example string *puli* we get the automaton S shown in Figure A.1 below. Then composing S with T ($S \circ T$) would provide all paths through T with the input string labeling the input side of the path. If we project to output labels (i.e., throw away the input labels), remove epsilon transitions and determinize (all general transducer operations in OpenFst), we are left with a weighted lattice of possible transliterations for the input string. Note that the weight for each distinct transliteration is the minimum cost (maximum probability) score for that output, which is used as an approximation of the probability for the candidate, rather than summing over all alignments between the input and output string.

There is an alternative method for composing with the pair *n*-gram model, which allows for backoff transitions to be encoded as failure (ϕ) transitions rather than ϵ-transitions (Allauzen, Mohri, and Roark 2003). An ϵ-transition can always be traversed without consuming input, but a ϕ-transition can only be traversed if the input symbol does not label another transition leaving that state. The primary benefit in this case would be that for a given input, fewer paths would be traversed through the model, leading to less intermediate memory usage. Further, the ϕ-transition backoffs provide an exact encoding of the *n*-gram model, rather than the approximation of allowing backoff even when the input symbol is present. See Allauzen, Mohri, and Roark (2003) for discussion.

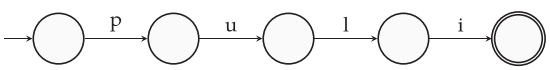


Figure A.1
String automaton over Latin script symbols corresponding to *puli*.

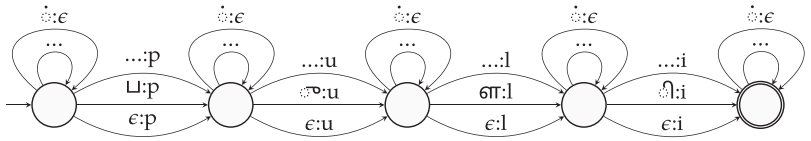


Figure A.2
Schematic of input string pair lattice.

To intersect directly with the pair automaton P using ϕ-transitions to represent backoff, we must encode the input string as a lattice of possible pair symbols. Figure A.2 shows this for our example string, where insertion loops at each state allow output symbols (illustrated here with the *virama* symbol) with ϵ on the input side, and at each position all possible pair symbols with that character on the input side (including ϵ, which results in a deletion) must be included. This can then be intersected with the P automaton, and "projected" to output labels, etc., as with the transducer approach. For this paper, we follow this second approach to decoding with the pair n-gram model.

For context-aware transliteration of full sentences, one of the approaches that we follow is to combine non-contextual transliteration models and language models. The non-contextual transliteration models are used to extract k-best possible transliterations for each word in the input string, along with probabilities. These are encoded in an acyclic weighted finite-state automaton, i.e., a word lattice, where the weights are stored as negative log probabilities. As mentioned in Section 3.5, our language models are also encoded as weighted finite-state automata, and we can combine the two models simply via finite-state composition. This is followed by finite-state shortest path extraction (Mohri 2002), which efficiently returns the lowest cost (highest probability) paths.

One slight complication is the use of word pieces in the language models, however words in the lattice produced by the transliteration models deterministically map to a string of word-piece tokens. An unweighted deterministic finite-state transducer can be constructed to map from words to word pieces (and back), and this transducer can be used as part of the composition mentioned above. Specifically, let T be the word lattice produced by the transliteration models; L be the language model; and W be the transducer mapping from words to word pieces. The highest probability paths from the combination of the transliteration and language models would then be obtained by: ShortestPath($T \circ W \circ L$).[40]

Appendix B. Fine-tuning Dataset Pruning

During initial attempts to fine-tune the pre-trained ByT5 model for single word Latin-to-native script transliteration, we noted that the model yielded relatively poor results for Tamil relative to other modeling methods. For other languages in the set the resulting ByT5 model was generally quite good—often yielding lower CER% than the other modeling methods—but for Tamil the models achieved a relatively high mean CER% and very high variance. An examination of model training patterns led us to suspect that the model was having difficulty dealing with outliers in the fine-tuning data, e.g., stray translations, complex correspondences, or outright errors in the data.

40 Projecting onto the input labels would give the words for each path.

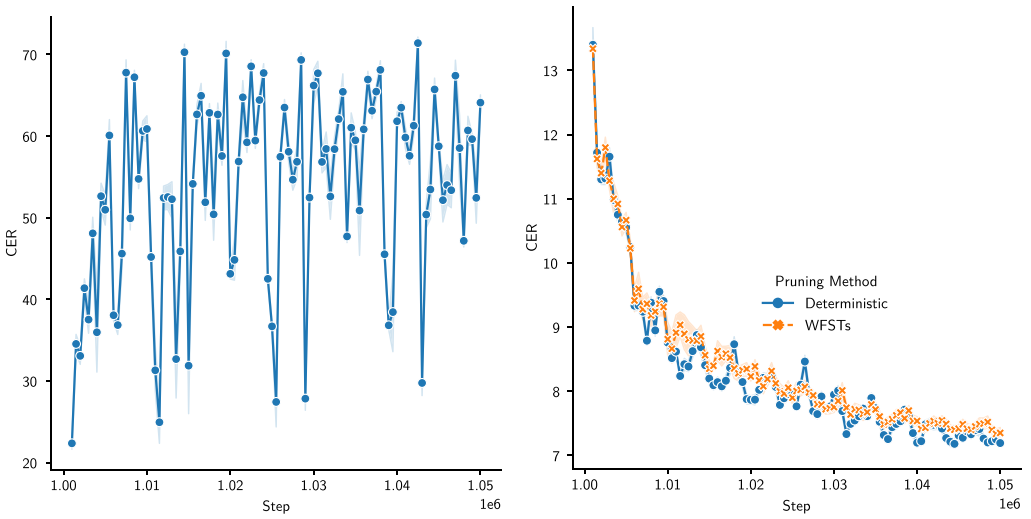


Figure B.1
Tamil Latin-to-native single word transliteration CER% as fine-tuning progresses, evaluated on small held-out portion of the training set, in two conditions: (a) all training data included; (b) removal of 10% of the least representative examples from the fine-tuning data, using two different methods. Each point in the graph is the mean CER% of 25 models with 95% confidence intervals indicated with shading.

Before presenting the details of our examination, we will foreshadow the results. Figure B.1 presents a comparison of held-aside CER% as training progresses, with the original training data on the left, and results using two different methods for identifying the least representative 10% of the training data and excluding those items from the fine-tuning data on the right. Note that the *y*-axis scales are different, so that every point in the right plot represents a lower CER% than any point in the left plot. Something in that 10% of examples caused the fine-tuning to fail to adequately learn the task.

Prior to determining the cause of this failure of fine-tuning, seeing the plot on the left led us to inspect the model predictions at some particularly poor operating point. One point in training that resulted in many regressions was the transition between steps 11,500 and 12,000 of fine-tuning, and Table B.1 presents a subset of salient prediction errors introduced at this point. Input romanizations are shown with targets and predictions in the native script, the latter presented with colloquial romanizations underneath, along with the corresponding *simplified* glosses.[41] Interestingly, although all the predictions in Table B.1 are grossly wrong as transliterations, they are still valid Tamil words. This led us to hypothesize that the model was learning to produce some other kind of correspondence for these items than the transliteration that we were attempting to train it to perform. Note that the first five predictions completely mismatch the targets, while the last two at least share some matching word-final akṣara syllables. For example, for the input sequence "paathukaapputan", the target sequence "பாதுகாப்புடன்" and the corresponding prediction "நடன்" share the suffix "டன்" ("tan"). Only two of the

41 These translations are from Google Translate and were judged by the native speaker as "mostly fine", given the highly agglutinative nature of Tamil (Steever 1987; Lehmann 1993; Andronov 2004).

Table B.1
Examples of Tamil ByT5 model prediction errors in Latin-to-native direction on the held-out portion of the training set during the fine-tuning. The inputs are prefixed with "`ta-Latn-Taml-`" string (not shown). Targets and predictions are shown with the colloquial romanizations underneath along with the simplified translations.

Input	Target	Gloss	Prediction	Gloss
`baathukaappuk`	பாதுகாப்புக் *pathukappuk*	"security"	நடைப் *nataip*	"walking"
`cakoothariyaana`	சகோதரியான *chakothariyana*	"sister"	திருந்த *thiruntha*	"changed"
`ilamaiyin`	இளமையின் *ilamaiyin*	"of youth"	தெரிய *theriya*	"to know"
`kalavaiyaana`	கலவையான *kalavaiyana*	"mixed"	தெரிய *theriya*	"to know"
`maarakkoodiya`	மாறக்கூடிய *marakkutiya*	"variable"	தொழில் *thozhil*	"profession"
`paathukaapputan`	பாதுகாப்புடன் *pathukapputan*	"with security"	நடன் *natan*	"dancer"
`vaiththiruntthanar`	வைத்திருந்தனர் *vaiththirunthanar*	"kept"	திருந்தனர் *thirunthanar*	"stunned"

predictions in Table B.1 ("தெரிய" and "தொழில்") are found in the Dakshina training data as standalone lexical items; the rest are either not found or are present as constituent morphemes of other words.

These observations led us to speculate that some of the training instances were providing confusing guidance to the model, so we began to look for items that did not appear to be transliterations in the training set. To do this, we used a deterministic colloquial romanization scheme[42] to derive what a romanization might look like for each native-script term in the training set, and then compared those romanizations with the ones provided in training. To avoid as many spurious mismatches as possible, it was important to use a romanization scheme that was colloquial versus formal methods such as the ISO 15919 standard (ISO 2001), which does not represent common usage of the sort included in the training set, but is rather focused on exact encoding of the graphemes used and reversibility.

Given the colloquial romanization, for each training instance, we calculated a distance d between the romanization provided and the one that resulted from the deterministic romanization. For this work, we use a sum of normalized Damerau–Levenshtein (Damerau 1964) and Jaro-Winkler (Winkler 1990) distances between individual romanized strings.[43] Table B.2 presents Tamil items that fall in the top 10% of the training set in terms of this distance, including the very highest scoring entries as well as the lowest scoring that still fall within the top 10%. For each entry the score d is shown alongside the informal romanization x, the corresponding Tamil target y, and the output x' of the rule-based romanizer T applied to y. The entities with the highest values of d mostly correspond to annotation noise in Dakshina as evidenced by the third entry, where for the input "`tikket`", the target "ஹால்" ("`hal`") is likely an artifact of mislabeling the

42 We use Aksharamukha rule-based colloquial Brahmic script romanizer available at `https://github.com/virtualvinodh/aksharamukha-python`.
43 We use the implementation of both algorithms from `https://github.com/life4/textdistance/`.

Table B.2
Examples from the 10% of the training set entries with the highest d; the left side shows the highest-scoring entries from this set and the right side shows the lowest-scoring entries from that top 10%.

Highest-scoring entries				Lowest-scoring entries			
d	Input (x)	Target (y)	$x' = T(y)$	d	Input (x)	Target (y)	$x' = T(y)$
2.0	poruttu	அதன்	athan	...	...	...	...
2.0	patcattil	இருக்கும்	irukkum	0.9	brazilin	பிரேசிலின்	pirechilin
2.0	tikket	ஹால்	hal	0.8	nylon	நைலான்	nailan
1.6	vinaayagar	சித்தி	chiththi	0.8	apace	அப்பாச்சி	appachchi
1.5	theriyum	எனக்குத்	enakkuth	0.8	qaitholiq	கத்தோலிக்க	kaththolikka
...	...	...	...	0.8	calcium	கால்சியம்	kalchiyam

original Tamil constituent corresponding to "tikket hal". Lower values of d, such as found on the right side of the table, often correspond to loanwords, English or other foreign-origin words, personal or place names, etc. While these are likely not errors, they do represent quite challenging items in the training which may also be complicating fine-tuning.

Investigations of the single word training data in other languages found similar outliers, hence we sought to find a pruning method that could be applied to all of the languages in the data. Reliance on a third-party package for the secondary romanization needed to score each instance is problematic in particular for the languages using a Perso-Arabic script, Sindhi and Urdu. While some romanization conventions, such as the Hunterian system, are generally followed for Urdu (United Nations 2007), the variance in the ways different annotators romanize Perso-Arabic in Dakshina is significantly higher than for the Brahmic-script languages. This is partially explained by the nature of the writing systems in question, since they do not mark vowels, leaving some degree of freedom to the annotators. This would lead to even more spurious mismatches with such a deterministic system. Sindhi, however, presents a more serious problem, in that it is not supported in the Aksharamukha system and Sindhi romanization is not covered by any formal recommendations (Motlani 2016). It is also worth noting that the Sindhi writing system is significantly more complex than Urdu (Section 3.3 in Doctor et al. 2022; Ahmadi and Anastasopoulos 2023). As an alternative that can cover all languages, we next outline an approach wherein we build our own romanization systems to provide a secondary romanization to compare with the training instances.

Let $\mathcal{D} = \{(x_i, y_i)\}$ be the training set string pairs, where x_i are the romanized inputs and y_i are the corresponding native-script outputs. To train each model we perform a random shuffle and split $\mathcal{D}$ into the disjoint training $\mathcal{D}_k^{\text{Train}}$ and test $\mathcal{D}_k^{\text{Test}}$ subsets.[44] We then define a ranking function as

$$r(x_i, y_i) = \frac{\sum_{k=1}^{K} \sum_{(x_i,y_i)\in\mathcal{D}_k^{\text{Test}}} d\left(x_i, f_k(y_i)\right)}{\sum_{k=1}^{K} \sum_{(x_i,y_i)\in\mathcal{D}_k^{\text{Test}}} 1} \tag{B.1}$$

44 Please note that unlike the classical K-fold cross validation, our K test sets may overlap making the algorithm somewhat similar to leave-P-out cross-validation (Celisse 2008).

Table B.3
Tamil Latin-to-native script CER% for four types of models when trained on full training sets versus trained on pruned data sets.

Model	All data	Pruned data
Pair 6g	9.4 (.04)	9.3 (.03)
LSTM	8.6 (.14)	8.5 (.11)
Transformer	8.2 (.10)	8.2 (.08)
ByT5	12.7 (3.85)	**7.9** (.42)

where the set $\mathcal{D}_k^{\text{Train}}$ is used for training f_k, and the normalization factor in denominator is a count of all instances of (x_i, y_i) in K test partitions. More specifically, we set K to 20 and use 50% random split into training and test partitions using a stratified (w.r.t. native-script types) sampling strategy that ensures that

$$\forall (x_j, y_j) \in \mathcal{D}_k^{\text{Train}} \text{ and } \forall (x_l, y_l) \in \mathcal{D}_k^{\text{Test}} : y_j \neq y_l \tag{B.2}$$

for all $k \in [1, K]$, meaning that there is no overlap of native-script strings y between the training and test partitions of each $\mathcal{D}_k$. We use the same string distance metric d as defined above for comparing individual romanized strings.

For the romanization mappings we train 20 Pair-LM n-gram FSTs for each language. The model details are provided in Section 3.3.3 and Appendix A. The parameters of the models are different from those reported in Section 3.3.3 because each model is estimated using 50% of the original data. In particular, we train 4-gram instead of 6-gram transducers for all languages and retain the unigram state in the resulting models. Single shortest path decoding is used at inference time.

The plot in Figure B.1(b) shows fine-tuning of the Tamil system after pruning the highest scoring 10% from the fine-tuning data, using either the deterministic romanization scoring or the FST-based method. Both yield very similar results.

For final evaluation, we train four types of models using the pruned data—in addition to ByT5 we also train Pair 6-gram FSTs, LSTMs, and Transformers. The results for Tamil are summarized in Table B.3, where we contrast them with the models trained using the full training set. We observe that the significant gains only apply to ByT5 fine-tuned model, while for other models this filtering has essentially no effect, i.e., they are much more robust to outliers. For that reason, all other models were trained using the full training sets.

Appendix C. mT5 SentencePiece Coverage Issues

Table 8 in Section 4.3 presents WER% of context-aware Latin-to-native script transliteration systems for all languages on the development portion of the Dakshina dataset. The mT5 models on Sinhala performed extremely poorly relative to the ByT5 models—more than 10% absolute higher. This was the only language with higher WER% for mT5 than ByT5, leading us to investigate why this particular language was so much worse for that modeling method.

We first looked to characterize the kinds of errors that were being made relative to other conditions, and to that end Table C.1 presents the raw total number of substitutions, insertions, and deletions for the ByT5 (Base) and mT5 (Large) models for each

Table C.1
Comparison of substitutions, insertions, and deletions between the hypotheses of ByT5 base and mT5 large models on the full-sentence Dakshina dev set, total errors summed over all 5 trials. For each class we also report r defined as the absolute difference for each class divided by the number for the ByT5 base model. The anomalous values of mT5 insertions and the corresponding values of r_{ins} are highlighted in pink. The highest number of insertions corresponds to Sinhala.

Language	Substitutions			Insertions			Deletions		
	ByT5	mT5	r_{sub}	ByT5	mT5	r_{ins}	ByT5	mT5	r_{del}
bn	40,810	40,486	0.0	4,438	4,641	0.0	8,831	2,920	0.7
gu	45,384	49,727	0.1	1,997	2,052	0.0	13,430	307	1.0
hi	49,523	46,800	0.1	1,406	1,544	0.1	17,399	886	0.9
kn	31,081	31,533	0.0	1,182	1,308	0.1	14,779	5,621	0.6
ml	42,775	42,463	0.0	592	1,320	1.2	5,293	506	0.9
mr	25,092	25,984	0.0	322	718	1.2	4,070	507	0.9
pa	62,006	64,429	0.0	1,198	1,317	0.1	14,226	967	0.9
sd	106,174	101,309	0.0	15,989	16,158	0.0	6,274	1,906	0.7
si	52,602	75,176	0.4	3,959	**31,907**	**7.1**	9,922	1,317	0.9
ta	39,685	37,593	0.1	811	891	0.1	5,618	147	1.0
te	30,195	31,384	0.0	1,699	4,266	1.5	6,319	350	0.9
ur	43,233	40,945	0.1	7,001	7,090	0.0	6,136	5,057	0.2

language. In order to detect divergences between the two systems for each class of error, we also calculate r_{class} for each class of error defined as

$$r_{\text{class}} = \frac{\left| N_{\text{class}}^{\text{ByT5}} - N_{\text{class}}^{\text{mT5}} \right|}{N_{\text{class}}^{\text{ByT5}}} \tag{C.1}$$

which yields r_{sub}, r_{ins}, and r_{del} for the substitutions, insertions, and deletions, respectively. As can be seen from the table, insertions stand out, with high r_{ins} values for Malayalam, Marathi, Tamil, and especially Sinhala. The number of insertions in the mT5 output for Sinhala is nearly an order of magnitude higher than for ByT5.

Since the key difference between mT5 and ByT5 is the former's use of a SentencePiece vocabulary,[45] we hypothesized that the relatively high number of insertions by mT5 in some languages may be due to some lack of coverage by mT5 vocabulary. Two Unicode symbols used in the Dakshina dataset are not found in the mT5 vocabulary: zero-width non-joiner (ZWNJ, U+200C) and zero-width joiner (ZWJ, U+200D), which are non-printing characters that are often used in digital representations of Brahmic scripts (Gupta and Sornlertlamvanich 2007; Unicode Consortium 2022). Out of the 12 Dakshina languages, the ZWNJ characters were only found in Malayalam and Telugu, while the ZWJ characters are only present in Marathi and Sinhala. These characters appear not only in the Dakshina dataset but also in the mC4 dataset, but were apparently removed during SentencePiece vocabulary selection. It is important to note that these characters cannot be merely dismissed as text noise, but instead serve an important

45 The mT5 vocabulary is available at https://console.cloud.google.com/storage/browser/t5-data/vocabs/mc4.250000.100extra. It consists of 250K entries (with 100 special tokens).

Table C.2
Comparison of types with Unicode zero-width non-joiner (ZWNJ, U+200C) and zero-width joiner (ZWJ, U+200D) characters between normalized mC4 and Wikipedia corpora. The Malayalam and Telugu corpora only have ZWNJ characters, while Marathi and Sinhala corpora only have ZWJ characters. The counts are rounded to the nearest integer. The measures r are defined as a ratio (expressed in percents) between the count of types or tokens with ZWNJ or ZWJ characters, and the total number of types or tokens, respectively.

		mC4						Wikipedia					
L.	Char	Types			Tokens			Types			Tokens		
		N	N_{ZW}	r_{typ}	N	N_{ZW}	r_{tok}	N	N_{ZW}	r_{typ}	N	N_{ZW}	r_{tok}
ml	ZWNJ	25M	2M	7.6	976M	19M	1.9	1M	42K	3.8	6M	93K	1.4
mr	ZWJ	12M	228K	1.9	1,272M	4M	0.3	335K	4K	1.3	3M	11K	0.3
si	ZWJ	5M	569K	12.3	428M	22M	**5.1**	232K	31K	13.4	3M	213K	**7.6**
te	ZWNJ	13M	2M	13.3	630M	30M	4.7	891K	54K	6.1	9M	133K	1.5

graphemic function. In Sinhala, for example, the ZWJ characters are used among other things to form ligated conjunct consonant clusters (known as *bandi akuru*, බැඳී අකුරු) that otherwise have no atomic Unicode representation as a single Sinhala code-point (Samaranayake et al. 2003; Wijayawardhana et al. 2008).

In addition to the Wikipedia-based Dakshina development set, we also investigate the prevalence of ZWNJ and ZWJ characters in the mC4 corpus that is used for pre-training mT5 model (Xue et al. 2021). For these counts, we preprocess each of the 12 language's corpora in mC4 using an approach similar to Daskhina whitespace normalization described in Section 3.2.4: We treat any character outside of the language's native script Unicode code block as a whitespace but retain the characters from the Unicode general punctuation code block (U+2000 – U+206F). The comparison of counts for types and tokens corresponding to ZWNJ and ZWJ between whitespace-normalized mC4 and Wikipedia data is shown in Table C.2. For both corpora, ZWNJ characters are only found in Malayalam and Telugu, while the ZWJ characters are only present in Marathi and Sinhala. For each corpus (mC4 or Wikipedia) and each word (token or type) three values are shown in Table C.2: the total number of types or tokens in the corpus (N), the number of types or tokens with ZWNJ or ZWJ characters (N_{ZW}), and the measure r (for types or tokens) defined as $r = 100\frac{N_{ZW}}{N}$.

As can be seen from Table C.2, Sinhala mC4 and Wikipedia normalized corpora have the highest percentage of tokens with zero-width non-printing characters (according to r_{tok}) among all the languages in question, and the Wikipedia types and tokens are particularly high, which explains why this issue causes such a noticeable spike in error rate for Sinhala in particular.

The ZWNJ and ZWJ characters are treated as OOVs by mT5, which explains the abnormally high number of insertions: mT5's SentencePiece tokenization produces extra whitespace tokens ("▁", or U+2581, in the mT5 vocabulary) in positions corresponding to ZWJ characters, thus over-segmenting and causing evaluation errors.[46]

Three examples of such segmentation errors are shown in Table C.3. Each input Sinhala string is accompanied by: (1) the number of ZWJ characters it contains; (2) the

46 The degradation in performance of large multilingual neural models on downstream tasks due to insufficient vocabulary coverage has been noted before for models other than mT5 (Wang et al. 2019; Liang et al. 2023).

Table C.3
Examples of bad Sinhala ZWJ tokenization using mT5 SentencePiece tokenizer. The OOV ZWJ is represented by "▁" (U+2581, Lower One Eighth Block).

Input	N_{ZWJ}		mT5 Tokens		Detokenized
ශ්‍රී	1	→	{ "▁ශ්", "▁රී"}	→	ශ් රී
පරීක්‍ෂිත	1	→	{ "▁ප", "රී", "ක්", "▁ෂ", "ිත" }	→	පරීක් ෂිත
ආශ්‍රිතව	2	→	{ "▁ආ", "ශ", "▁", "්", "▁රි", "තව" }	→	ආශ ් රිතව

resulting mT5 SentencePiece segmentation; and (3) the bad detokenized output string, with the problematic tokens and the corresponding bad outputs shown in dark red. As can be seen from the table, each OOV token is converted to whitespace by the detokenizer. In the first two examples from the table, the corruption results in ungrammatical, but orthographically valid output. For example, the first word in "Sri Lanka" represented by Sinhala akṣara "ශ්‍රී" ("shri") is emitted as two independent akṣara tokens "ශ්" ("sh") and "රී" ("ri"). In the third example, the two ZWJ characters in the input string cause the tokenizer to over-segment twice, resulting in three whitespace-separated outputs, the second of which is orthographically illegal because it violates the rules of representing the akṣara in Brahmic scripts—the modifier character, such as Sinhala *virama* "්" in this example (U+0DCA, Sinhala Sign Al-Lakuna), cannot start a syllable (Salomon 1996; Bright 1999).

While these problems are most noticeable in our Sinhala data, they also occur in the other languages with ZWJ or ZWNJ (ml, mr, te), just to a lesser degree given the lower percentage of Wikipedia tokens in those languages using those characters. As can be seen from the mC4 statistics in Table C.2, a different sample of text could very well have had as many such tokens in Telugu as in Sinhala, leading to similar WER% increases in that language.

Appendix D. Experimental Results for All Languages

Here we present full tables across all languages in the collection for results that are otherwise reported in the main body of the paper just as micro-averaged and/or macro-averaged error rates.

- **Single-word Latin-to-native transliteration:** Table D.1 presents the reductions in single word Latin-to-native transliteration CER% achieved by ensembling five different runs of the same modeling method, versus unensembled runs of the method, using ensembling methods outlined in Section 3.6. Table D.2 presents CER% for ensembles of two distinct methods, and Table D.3 presents ensembles of 3 or more systems.
- **Single-word native-to-Latin transliteration:** Table D.4 presents the reductions in minCER% achieved by ensembling five different runs of the same modeling method, versus unensembled runs of the method. Table D.5 presents minCER% for ensembles of two distinct methods, and Table D.6 presents ensembles of 3 or more systems. Table D.7 presents the reductions in EMDCER% achieved by ensembling five different runs of

Table D.1
Per-language single method ensembling CER% results for all four modeling methods for Latin-to-native script single word transliteration, along with reduction in CER (Δ) versus unensembled systems.

Lang	Pair 6g		Transformer		LSTM		ByT5	
	CER%	Δ	CER%	Δ	CER%	Δ	CER%	Δ
bn	14.2 (.04)	0.1	12.6 (.05)	0.4	12.8 (.07)	0.9	11.9 (.31)	0.5
gu	12.8 (.02)	0.2	11.6 (.08)	0.3	11.7 (.09)	0.7	9.7 (.17)	0.3
hi	14.8 (.05)	0.0	13.2 (.13)	0.2	13.1 (.09)	0.7	10.6 (.20)	0.4
kn	7.2 (.10)	0.1	6.4 (.04)	0.2	6.3 (.06)	0.4	5.3 (.16)	0.3
ml	9.9 (.05)	0.1	8.7 (.09)	0.2	8.6 (.07)	0.5	8.2 (.21)	0.3
mr	12.5 (.13)	0.1	11.3 (.11)	0.3	11.6 (.12)	0.8	9.6 (.31)	0.4
pa	18.0 (.03)	0.0	16.7 (.07)	0.3	16.7 (.08)	0.8	15.2 (.35)	0.4
sd	20.6 (.12)	0.1	19.5 (.16)	0.5	19.4 (.13)	1.0	18.5 (.22)	1.1
si	9.3 (.02)	0.0	8.7 (.03)	0.3	8.7 (.06)	0.5	8.9 (.12)	0.3
ta	9.3 (.02)	0.1	8.0 (.02)	0.2	8.1 (.07)	0.5	7.6 (.08)	0.3
te	6.9 (.04)	0.1	6.0 (.07)	0.2	6.3 (.06)	0.4	5.6 (.15)	0.3
ur	20.0 (.01)	0.0	19.0 (.14)	0.4	18.5 (.08)	1.1	19.8 (.34)	0.7
μ	13.0 (.02)	0.0	11.8 (.02)	0.3	11.8 (.02)	0.7	10.9 (.04)	0.5
all	12.8 (.02)	0.1	11.7 (.03)	0.3	11.7 (.02)	0.7	10.8 (.05)	0.5

Table D.2
Per-language two-model ensembling CER% results for Latin-to-native script single word transliteration.

Lang	LSTM	Pair 6g + transformer	ByT5	LSTM + ByT5	Transformer + LSTM	Transformer + ByT5
bn	12.2 (.05)	12.0 (.10)	11.3 (.14)	11.2 (.15)	12.2 (.05)	11.2 (.14)
gu	11.3 (.07)	11.2 (.06)	9.5 (.16)	9.6 (.18)	11.1 (.05)	9.4 (.08)
hi	12.9 (.06)	12.8 (.11)	11.0 (.20)	10.9 (.16)	12.7 (.07)	10.8 (.15)
kn	6.1 (.07)	6.2 (.03)	5.2 (.12)	5.0 (.08)	6.0 (.05)	5.3 (.12)
ml	8.4 (.04)	8.5 (.03)	8.0 (.07)	7.8 (.13)	8.4 (.07)	7.8 (.12)
mr	11.3 (.08)	11.0 (.07)	9.5 (.23)	9.6 (.21)	11.0 (.03)	9.4 (.19)
pa	16.5 (.09)	16.4 (.06)	15.2 (.17)	15.2 (.16)	16.3 (.07)	15.1 (.13)
sd	18.8 (.11)	19.0 (.07)	17.9 (.18)	17.7 (.20)	19.0 (.11)	17.9 (.24)
si	8.2 (.06)	8.2 (.05)	8.2 (.07)	8.2 (.11)	8.4 (.04)	8.2 (.04)
ta	8.0 (.05)	7.9 (.04)	7.5 (.04)	7.4 (.08)	7.9 (.03)	7.3 (.08)
te	6.0 (.03)	5.9 (.07)	5.4 (.06)	5.4 (.01)	5.9 (.09)	5.3 (.09)
ur	18.0 (.04)	18.4 (.06)	18.3 (.18)	17.8 (.18)	18.1 (.10)	18.6 (.28)
μ	11.5 (.02)	11.5 (.01)	10.6 (.02)	10.5 (.02)	11.4 (.03)	10.5 (.03)
all	11.4 (.02)	11.3 (.01)	10.5 (.03)	10.4 (.03)	11.3 (.02)	10.4 (.03)

the same modeling method, versus unensembled runs of the method. Table D.8 presents EMDCER% for ensembles of two distinct methods, and Table D.9 presents ensembles of 3 or more systems.

- **Full sentence native-to-Latin transliteration:** Table D.10 presents single method ensembled transliteration model WER% on the dev set.

Table D.3
Per-language multi-model ensembling CER% results for Latin-to-native script single word transliteration.

Lang	All models in ensemble except				All
	Pair 6g	Transformer	ByT5	LSTM	models
bn	11.3 (.10)	11.1 (.06)	11.8 (.03)	11.0 (.08)	11.1 (.06)
gu	9.8 (.09)	9.6 (.13)	10.9 (.07)	9.6 (.08)	9.8 (.07)
hi	11.4 (.08)	11.1 (.09)	12.4 (.07)	11.1 (.07)	11.4 (.06)
kn	5.3 (.07)	5.1 (.08)	5.8 (.06)	5.2 (.09)	5.3 (.05)
ml	7.9 (.07)	7.8 (.09)	8.2 (.04)	7.9 (.06)	7.8 (.03)
mr	9.9 (.12)	9.7 (.17)	10.8 (.04)	9.6 (.10)	9.9 (.12)
pa	15.4 (.14)	15.2 (.10)	16.1 (.06)	15.1 (.11)	15.3 (.11)
sd	17.8 (.09)	17.7 (.10)	18.6 (.08)	17.8 (.12)	17.7 (.04)
si	8.1 (.10)	7.9 (.07)	8.0 (.06)	7.8 (.06)	7.9 (.05)
ta	7.4 (.08)	7.4 (.03)	7.8 (.03)	7.3 (.02)	7.4 (.04)
te	5.4 (.06)	5.3 (.03)	5.8 (.04)	5.3 (.08)	5.3 (.06)
ur	17.7 (.10)	17.4 (.06)	17.6 (.03)	17.7 (.17)	17.3 (.11)
μ	10.61 (.03)	10.43 (.03)	11.13 (.01)	10.44 (.02)	10.51 (.02)
all	10.47 (.03)	10.30 (.03)	11.00 (.01)	10.31 (.02)	10.37 (.02)

Table D.4
Per-language single method ensembling minCER% results for all four modeling methods for native-to-Latin script single word transliteration, along with reduction in minCER (Δ) versus unensembled systems.

Lang	Pair 6g		Transformer		LSTM		ByT5	
	minCER%	Δ	minCER%	Δ	minCER%	Δ	minCER%	Δ
bn	4.1 (.04)	0.1	2.8 (.05)	0.3	2.8 (.03)	0.4	2.4 (.03)	0.1
gu	2.5 (.03)	0.0	1.2 (.04)	0.1	1.1 (.02)	0.2	1.0 (.03)	0.1
hi	4.6 (.03)	0.0	3.2 (.08)	0.2	3.3 (.10)	0.4	2.8 (.08)	0.1
kn	1.4 (.03)	0.0	0.9 (.09)	0.4	0.8 (.06)	0.1	0.8 (.05)	0.1
ml	1.6 (.01)	0.0	2.3 (.53)	0.5	0.9 (.03)	0.3	0.9 (.02)	0.1
mr	2.4 (.03)	0.0	1.6 (.03)	0.1	1.6 (.04)	0.2	1.5 (.12)	0.1
pa	4.3 (.03)	0.0	3.0 (.08)	0.1	3.2 (.07)	0.3	2.9 (.06)	0.1
sd	8.6 (.09)	0.1	6.8 (.08)	0.3	7.0 (.09)	0.5	6.3 (.18)	0.2
si	1.1 (.01)	0.0	0.4 (.03)	0.1	0.5 (.03)	0.1	0.7 (.04)	0.0
ta	3.2 (.02)	0.1	2.6 (.09)	0.5	2.5 (.02)	0.3	2.4 (.05)	0.1
te	2.8 (.02)	0.0	2.2 (.05)	0.3	2.0 (.05)	0.3	2.4 (.07)	0.1
ur	7.7 (.02)	0.0	5.9 (.03)	0.2	6.1 (.07)	0.5	5.8 (.25)	0.1
μ	3.7 (.01)	0.0	2.7 (.04)	0.3	2.7 (.02)	0.3	2.5 (.02)	0.1
all	3.4 (.01)	0.1	2.5 (.05)	0.3	2.4 (.01)	0.3	2.3 (.02)	0.1

Table D.5
Per-language two model ensembling minCER% results for native-to-Latin script single word transliteration.

Lang	LSTM	Pair 6g + Transformer	ByT5	LSTM + ByT5	Transformer + LSTM	Transformer + ByT5
bn	2.7 (.02)	2.8 (.03)	2.3 (.02)	2.2 (.04)	2.5 (.06)	2.1 (.05)
gu	1.3 (.04)	1.7 (.06)	1.2 (.03)	1.0 (.03)	1.0 (.02)	1.0 (.03)
hi	3.2 (.02)	3.3 (.04)	2.7 (.08)	2.7 (.05)	3.0 (.06)	2.6 (.04)
kn	0.7 (.01)	0.9 (.08)	0.8 (.03)	0.6 (.02)	0.7 (.03)	0.7 (.11)
ml	1.0 (.00)	1.1 (.03)	0.9 (.01)	0.8 (.03)	0.9 (.02)	0.9 (.01)
mr	1.7 (.03)	1.9 (.03)	1.6 (.07)	1.3 (.07)	1.6 (.02)	1.4 (.05)
pa	3.1 (.05)	3.2 (.03)	2.9 (.05)	2.7 (.02)	3.0 (.03)	2.8 (.04)
sd	6.8 (.03)	6.8 (.05)	6.3 (.15)	6.1 (.08)	6.6 (.04)	5.9 (.11)
si	0.5 (.01)	0.5 (.01)	0.5 (.03)	0.5 (.01)	0.4 (.01)	0.4 (.01)
ta	2.4 (.03)	2.5 (.04)	2.3 (.04)	2.2 (.01)	2.3 (.02)	2.3 (.03)
te	2.1 (.04)	2.2 (.02)	2.1 (.02)	1.9 (.02)	1.9 (.05)	2.0 (.06)
ur	5.9 (.06)	6.1 (.02)	5.7 (.08)	5.2 (.05)	5.7 (.09)	5.1 (.08)
μ	2.6 (.01)	2.7 (.01)	2.4 (.02)	2.3 (.01)	2.5 (.01)	2.3 (.02)
all	2.4 (.01)	2.5 (.01)	2.2 (.02)	2.1 (.01)	2.2 (.01)	2.1 (.02)

Table D.6
Per-language multi-model ensembling minCER% results for native-to-Latin script single word transliteration.

Lang	All models in ensemble except Pair 6g	Transformer	ByT5	LSTM	All models
bn	2.2 (.03)	2.2 (.06)	2.5 (.03)	2.1 (.04)	2.2 (.03)
gu	1.0 (.02)	1.0 (.01)	1.1 (.02)	1.0 (.04)	1.0 (.02)
hi	2.7 (.07)	2.6 (.04)	3.0 (.03)	2.6 (.06)	2.6 (.05)
kn	0.6 (.03)	0.6 (.03)	0.7 (.02)	0.7 (.04)	0.6 (.03)
ml	0.8 (.03)	0.8 (.01)	0.9 (.02)	0.8 (.02)	0.8 (.03)
mr	1.4 (.02)	1.4 (.01)	1.6 (.03)	1.5 (.05)	1.4 (.02)
pa	2.7 (.05)	2.7 (.03)	2.9 (.04)	2.8 (.04)	2.7 (.03)
sd	6.1 (.11)	6.1 (.06)	6.4 (.02)	6.1 (.07)	6.1 (.03)
si	0.4 (.02)	0.4 (.01)	0.4 (.01)	0.4 (.02)	0.4 (.02)
ta	2.2 (.02)	2.2 (.03)	2.3 (.03)	2.3 (.03)	2.2 (.02)
te	1.9 (.03)	1.9 (.05)	2.0 (.02)	2.0 (.04)	1.9 (.03)
ur	5.1 (.05)	5.1 (.08)	5.5 (.04)	5.1 (.06)	5.0 (.07)
μ	2.2 (.01)	2.3 (.01)	2.4 (.01)	2.3 (.02)	2.2 (.01)
all	2.0 (.01)	2.1 (.01)	2.2 (.01)	2.1 (.02)	2.0 (.01)

Table D.7
Per-language single method ensembling EMDCER% results for all four modeling methods for native-to-Latin script single word transliteration, along with reduction in EMDCER (Δ) versus unensembled systems.

Lang	Pair 6g		Transformer		LSTM		ByT5	
	EMDCER%	Δ	EMDCER%	Δ	EMDCER%	Δ	EMDCER%	Δ
bn	12.6 (.02)	0.1	11.5 (.10)	0.5	11.1 (.03)	0.5	11.2 (.05)	0.2
gu	10.0 (.10)	0.1	8.9 (.07)	0.2	8.4 (.07)	0.5	8.4 (.08)	0.1
hi	10.2 (.02)	0.0	9.4 (.18)	0.2	8.4 (.05)	0.4	8.2 (.10)	0.1
kn	4.7 (.06)	0.0	5.4 (.23)	0.4	4.0 (.01)	0.2	3.8 (.03)	0.1
ml	5.8 (.01)	0.1	7.1 (.28)	0.4	5.0 (.11)	0.4	5.4 (.08)	0.1
mr	7.8 (.05)	0.0	8.1 (.14)	0.3	6.7 (.03)	0.3	6.9 (.13)	0.1
pa	12.5 (.04)	0.0	11.8 (.05)	0.4	11.0 (.02)	0.5	11.2 (.10)	0.1
sd	17.1 (.07)	0.0	15.2 (.06)	0.4	15.2 (.04)	0.6	15.2 (.29)	0.3
si	4.1 (.00)	0.0	3.7 (.07)	0.4	3.4 (.02)	0.2	3.7 (.09)	0.1
ta	8.3 (.01)	0.1	9.1 (.11)	0.5	7.3 (.01)	0.4	7.6 (.10)	0.1
te	7.0 (.01)	0.0	7.0 (.14)	0.3	6.4 (.01)	0.4	6.5 (.09)	0.1
ur	17.8 (.00)	0.0	16.0 (.04)	0.3	16.4 (.03)	0.7	16.5 (.22)	0.2
μ	9.8 (.02)	0.1	9.4 (.03)	0.4	8.6 (.01)	0.4	8.7 (.02)	0.1
all	9.2 (.02)	0.1	9.0 (.03)	0.4	8.1 (.01)	0.4	8.2 (.02)	0.1

Table D.8
Per-language two model ensembling EMDCER% results for native-to-Latin script single word transliteration.

Lang	LSTM	Pair 6g + transformer	ByT5	LSTM + ByT5	Transformer + LSTM	Transformer + ByT5
bn	11.2 (.02)	11.4 (.06)	11.2 (.02)	10.7 (.02)	10.9 (.06)	11.0 (.06)
gu	8.6 (.05)	8.9 (.04)	8.6 (.05)	8.0 (.05)	8.3 (.06)	8.3 (.04)
hi	8.9 (.02)	9.3 (.08)	8.8 (.04)	8.0 (.06)	8.6 (.11)	8.5 (.13)
kn	4.0 (.03)	4.7 (.12)	4.0 (.02)	3.7 (.01)	4.4 (.10)	4.4 (.11)
ml	5.1 (.04)	6.0 (.11)	5.3 (.03)	4.9 (.07)	5.7 (.15)	5.8 (.12)
mr	6.9 (.01)	7.6 (.08)	7.0 (.06)	6.5 (.05)	7.2 (.07)	7.2 (.08)
pa	11.2 (.02)	11.6 (.03)	11.3 (.06)	10.6 (.04)	11.0 (.03)	11.1 (.06)
sd	15.5 (.05)	15.5 (.04)	15.5 (.14)	14.7 (.13)	14.8 (.04)	14.7 (.10)
si	3.5 (.01)	3.6 (.03)	3.7 (.04)	3.4 (.04)	3.3 (.03)	3.5 (.05)
ta	7.4 (.01)	8.1 (.05)	7.6 (.04)	7.2 (.03)	7.8 (.06)	8.0 (.04)
te	6.3 (.01)	6.6 (.05)	6.4 (.03)	6.2 (.03)	6.4 (.05)	6.5 (.08)
ur	16.3 (.02)	16.1 (.03)	16.3 (.09)	15.9 (.09)	15.8 (.03)	15.7 (.07)
μ	8.7 (.01)	9.1 (.02)	8.8 (.02)	8.3 (.01)	8.7 (.02)	8.7 (.02)
all	8.2 (.01)	8.6 (.02)	8.3 (.02)	7.8 (.01)	8.2 (.02)	8.3 (.02)

Table D.9
Per language multi-model ensembling EMDCER% results for native-to-Latin script single word transliteration.

Lang	All models in ensemble except				All
	Pair 6g	Transformer	ByT5	LSTM	models
bn	10.7 (.04)	10.7 (.02)	10.9 (.04)	10.9 (.05)	10.7 (.04)
gu	8.1 (.04)	8.2 (.04)	8.4 (.04)	8.4 (.03)	8.1 (.03)
hi	8.2 (.09)	8.3 (.03)	8.7 (.07)	8.7 (.08)	8.4 (.06)
kn	4.1 (.06)	3.8 (.02)	4.2 (.07)	4.2 (.08)	4.0 (.05)
ml	5.3 (.10)	5.0 (.04)	5.4 (.09)	5.5 (.07)	5.2 (.07)
mr	6.8 (.05)	6.6 (.03)	7.0 (.05)	7.1 (.06)	6.8 (.04)
pa	10.8 (.03)	10.8 (.03)	11.1 (.02)	11.1 (.05)	10.8 (.03)
sd	14.5 (.07)	14.9 (.09)	15.0 (.03)	14.9 (.07)	14.7 (.06)
si	3.3 (.04)	3.5 (.03)	3.4 (.02)	3.5 (.04)	3.4 (.03)
ta	7.5 (.02)	7.2 (.02)	7.6 (.04)	7.7 (.03)	7.4 (.02)
te	6.3 (.04)	6.2 (.02)	6.3 (.03)	6.4 (.04)	6.2 (.03)
ur	15.6 (.04)	15.8 (.05)	15.8 (.02)	15.7 (.05)	15.5 (.03)
μ	8.4 (.02)	8.4 (.01)	8.6 (.01)	8.7 (.02)	8.4 (.01)
all	8.0 (.01)	7.9 (.01)	8.1 (.01)	8.2 (.02)	7.9 (.01)

Table D.10
Per-language full string Latin-to-native script transliteration WER% achieved with single-word (non-contextual) single-system ensembles.

Language	Single-word Single-system Ensembles		
	Pair 6g	LSTM	ByT5
bn	34.7 (.05)	32.9 (1.17)	36.8 (.38)
gu	33.6 (.71)	29.6 (1.19)	29.7 (1.04)
hi	25.2 (.06)	27.4 (1.11)	25.7 (1.00)
kn	23.7 (.13)	21.4 (.10)	24.7 (.76)
ml	38.7 (.13)	39.5 (.51)	37.2 (1.28)
mr	29.8 (.52)	29.0 (.25)	28.1 (.84)
pa	38.1 (.10)	35.0 (1.60)	37.2 (1.19)
sd	55.5 (.03)	53.5 (.55)	54.5 (1.57)
si	37.7 (.01)	34.9 (.40)	39.2 (.45)
ta	30.1 (.10)	29.0 (.02)	30.2 (.62)
te	27.6 (.02)	26.3 (.18)	26.2 (.82)
ur	34.3 (.03)	31.4 (2.47)	33.6 (2.21)
μ	34.1 (.09)	32.5 (.19)	33.6 (.37)
all	34.9 (.09)	33.2 (.24)	34.4 (.37)

Appendix E. Transliteration Cache Coverage

Here we provide two plots of cache coverage in each of the Dakshina languages: type coverage and token coverage in Figure E.1. Type coverage measures the fraction of unique words found in the dev set that were also found in the cache. Token coverage is the fraction of all tokens in the dev set that were found in the cache. Token coverage rises above 75% for all languages, meaning that frequent words are relatively well covered even for the highly inflected Dravidian languages.

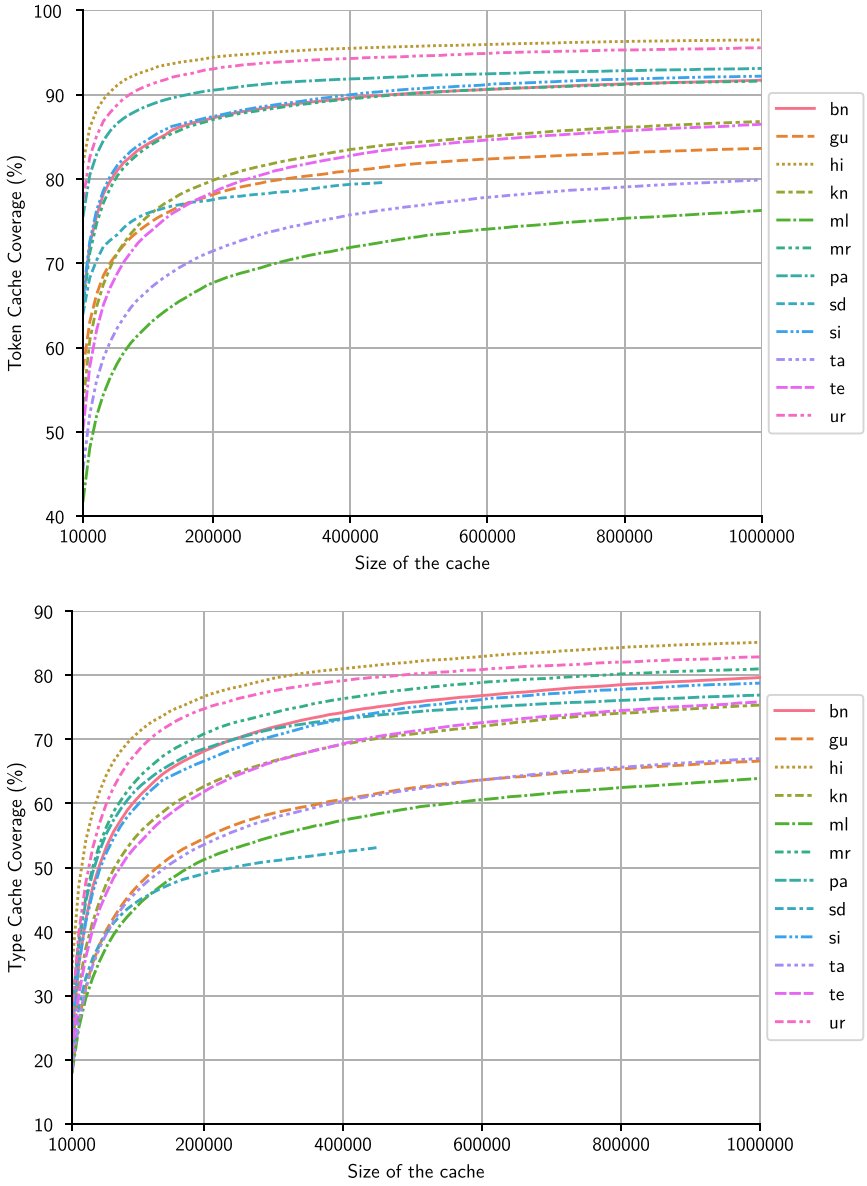


Figure E.1
Cache coverage plots for tokens (top) and types (bottom) displaying the coverage estimate (percentage) vs. cache size.

Half of the languages achieve greater than 90% token coverage, with Hindi and Urdu topping 95% coverage. Type coverage is, of course, lower, indicating that we are doing a better job covering frequent words than infrequent words, as was our intent.

Acknowledgments
The authors thank Işın Demirşahin, Raiomond Doctor, and Shankar Kumar for useful discussions, and anonymous reviewers for helpful comments and suggestions.

References

Ahmadi, Sina and Antonios Anastasopoulos. 2023. Script normalization for unconventional writing of under-resourced languages in bilingual communities. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14466–14487. https://doi.org/10.18653/v1/2023.acl-long.809

Al-Badrashiny, Mohamed, Ramy Eskander, Nizar Habash, and Owen Rambow. 2014. Automatic transliteration of romanized dialectal Arabic. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 30–38. https://doi.org/10.3115/v1/W14-1604

Allauzen, Cyril, Mehryar Mohri, and Brian Roark. 2003. Generalized algorithms for constructing statistical language models. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 40–47. https://doi.org/10.3115/1075096.1075102

Allauzen, Cyril, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of 12th International Conference on Implementation and Application of Automata (CIAA)*, pages 11–23. https://doi.org/10.1007/978-3-540-76336-9_3

Amrhein, Chantal and Rico Sennrich. 2020. On Romanization for model transfer between scripts in neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2461–2469. https://doi.org/10.18653/v1/2020.findings-emnlp.223

Andronov, Mikhail Sergeevich. 2004. *A Reference Grammar of the Tamil Language*, LINCOM Language Research. LINCOM Academic Publishers, Munich, Germany.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Baum, Leonard E. and Ted Petrie. 1966. Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563. https://doi.org/10.1214/aoms/1177699147

Bisani, Maximilian and Hermann Ney. 2002. Investigations on joint-multigram models for grapheme-to-phoneme conversion. In *Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP 2002)*, pages 105–108. https://doi.org/10.21437/ICSLP.2002-78

Bisani, Maximilian and Hermann Ney. 2008. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451. https://doi.org/10.1016/j.specom.2008.01.002

Bright, William. 1999. A matter of typology: Alphasyllabaries and abugidas. *Written Language & Literacy*, 2(1):45–55. https://doi.org/10.1075/wll.2.1.03bri

Celisse, Alain. 2008. *Model Selection via Cross-validation in Density Estimation, Regression, and Change-points Detection*. Ph.D. thesis, Faculté des Sciences d'Orsay, Université Paris Sud XI, Paris, France.

Chae, Moon-Jung, Kyubyong Park, Jinhyun Bang, Soobin Suh, Jonghyuk Park, Namju Kim, and Longhun Park. 2018. Convolutional sequence to sequence model with non-sequential greedy decoding for grapheme to phoneme conversion. In *Proceedings of 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2486–2490. https://doi.org/10.1109/ICASSP.2018.8462678

Chen, Hsin-Hsi, Sheng-Jie Huang, Yung-Wei Ding, and Shih-Chung Tsai. 1998. Proper name translation in cross-language information retrieval. In *COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics*, pages 232–236. https://doi.org/10.3115/980451.980883

Chen, Mia Xu, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. *arXiv preprint arXiv:1804.09849*. https://doi.org/10.18653/v1/P18-1008

Chen, Stanley F. 2003. Conditional and joint models for grapheme-to-phoneme conversion. In *Proceedings of the 8th European Conference on Speech Communication and Technology (Eurospeech 2003)*, pages 2033–2036. https://doi.org/10.21437/Eurospeech.2003-584

Choksi, Nishaant. 2020. From transcript to "trans-script": Romanized Santali across semiotic media. *Signs and Society*, 8(1):62–92. https://doi.org/10.1086/706549

Choudhury, Monojit, Anupam Basu, and Sudeshna Sarkar. 2004. A diachronic approach for schwa deletion in Indo Aryan languages. In *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology: Current Themes in Computational Phonology and Morphology*, pages 20–26. https://doi.org/10.3115/1622153.1622156

Conneau, Alexis, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451. https://doi.org/10.18653/v1/2020.acl-main.747

Damerau, Fred J. 1964. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176. https://doi.org/10.1145/363958.363994

Datta, Arindrima, Bhuvana Ramabhadran, Jesse Emond, Anjuli Kannan, and Brian Roark. 2020. Language-agnostic multilingual modeling. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8239–8243. https://doi.org/10.1109/ICASSP40776.2020.9053443

Demirsahin, Isin, Cibu Johny, Alexander Gutkin, and Brian Roark. 2022. Criteria for useful automatic Romanization in South Asian languages. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 6662–6673.

Deri, Aliya and Kevin Knight. 2016. Grapheme-to-phoneme models for (almost) any language. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 399–408. https://doi.org/10.18653/v1/P16-1038

Dhamecha, Tejas, Rudra Murthy, Samarth Bharadwaj, Karthik Sankaranarayanan, and Pushpak Bhattacharyya. 2021. Role of language relatedness in multilingual fine-tuning of language models: A case study in Indo-Aryan languages. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8584–8595. https://doi.org/10.18653/v1/2021.emnlp-main.675

Doctor, Raiomond, Alexander Gutkin, Cibu Johny, Brian Roark, and Richard Sproat. 2022. Graphemic normalization of the Perso-Arabic script. *arXiv preprint arXiv:2210.12273*. https://doi.org/10.48550/arXiv.2210.12273

Doddapaneni, Sumanth, Rahul Aralikatte, Gowtham Ramesh, Shreya Goyal, Mitesh M. Khapra, Anoop Kunchukuttan, and Pratyush Kumar. 2023. Towards leaving no Indic language behind: Building monolingual corpora, benchmark and models for Indic languages. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12402–12426. https://doi.org/10.18653/v1/2023.acl-long.693

Edunov, Sergey, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500. https://doi.org/10.18653/v1/D18-1045

Eskander, Ramy, Mohamed Al-Badrashiny, Nizar Habash, and Owen Rambow. 2014. Foreign words and the automatic processing of Arabic social media text written in Roman script. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, pages 1–12. https://doi.org/10.3115/v1/W14-3901

Finch, Andrew and Eiichiro Sumita. 2010. Transliteration using a phrase-based statistical machine translation system to re-score the output of a joint multigram model. In *Proceedings of the 2010 Named Entities Workshop*, pages 48–52.

Galescu, Lucian and James F. Allen. 2001. Bi-directional conversion between graphemes and phonemes using a joint *n*-gram model. In *Proceedings of the 4th ISCA Tutorial and Research Workshop (ITRW) on Speech Synthesis*, 6 pages.

Gella, Spandana, Kalika Bali, and Monojit Choudhury. 2014. "ye word kis lang ka hai bhai?" Testing the limits of word level language identification. In *Proceedings of the 11th International Conference on Natural Language Processing*, pages 368–377.

Gow-Smith, Edward, Mark McConville, William Gillies, Jade Scott, and Roibeard Ó Maolalaigh. 2022. Use of transformer-based models for word-level transliteration of the Book of the Dean of Lismore. In *Proceedings of the 4th Celtic Language Technology Workshop within LREC2022*, pages 94–98.

Gupta, Renu and Virach Sornlertlamvanich. 2007. Text entry in South and Southeast Asian scripts. In I. Scott MacKenzie and Kumiko Tanaka-Ishii, editors, *Text Entry Systems: Mobility, Accessibility, Universality*. Morgan Kaufmann, chapter 12, pages 227–250. https://doi.org/10.1016/B978-012373591-1/50012-7

Gutkin, Alexander, Cibu Johny, Raiomond Doctor, Brian Roark, and Richard Sproat. 2022a. Beyond Arabic: Software for Perso-Arabic script manipulation. In *Proceedings of the Seventh Arabic Natural Language Processing Workshop (WANLP)*, pages 381–387. https://doi.org/10.18653/v1/2022.wanlp-1.36

Gutkin, Alexander, Cibu Johny, Raiomond Doctor, Lawrence Wolf-Sonkin, and Brian Roark. 2022b. Extensions to Brahmic script processing within the Nisaba library: New scripts, languages and utilities. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 6450–6460.

Hellsten, Lars, Brian Roark, Prasoon Goyal, Cyril Allauzen, Françoise Beaufays, Tom Ouyang, Michael Riley, and David Rybach. 2017. Transliterated mobile keyboard input via weighted finite-state transducers. In *Proceedings of the 13th International Conference on Finite State Methods and Natural Language Processing (FSMNLP 2017)*, pages 10–19. https://doi.org/10.18653/v1/W17-4002

Hochreiter, Sepp and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735, PubMed: 9377276

Irvine, Ann, Jonathan Weese, and Chris Callison-Burch. 2012. Processing informal, romanized Pakistani text messages. In *Proceedings of the Second Workshop on Language in Social Media*, pages 75–78.

ISO. 2001. ISO 15919: Transliteration of Devanagari and related Indic scripts into Latin characters. https://www.iso.org/standard/28333.html. International Organization for Standardization.

ISO. 2002. ISO 639-1: Codes for the representation of names of languages—part 1: Alpha-2 code. International Organization for Standardization, Geneva, Switzerland.

Izacard, Gautier and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880. https://doi.org/10.18653/v1/2021.eacl-main.74

Jelinek, Frederick. 1998. *Statistical Methods for Speech Recognition*. MIT Press.

Jelinek, Frederick, Lalit Bahl, and Robert Mercer. 1975. Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory*, 21(3):250–256. https://doi.org/10.1109/TIT.1975.1055384

Jia, Ye, Ron J. Weiss, Fadi Biadsy, Wolfgang Macherey, Melvin Johnson, Zhifeng Chen, and Yonghui Wu. 2019. Direct speech-to-speech translation with a sequence-to-sequence model. In *Proceedings of Interspeech 2019*, pages 1123–1127. https://doi.org/10.21437/Interspeech.2019-1951

Jiampojamarn, Sittichai, Colin Cherry, and Grzegorz Kondrak. 2010. Integrating joint n-gram features into a discriminative training framework. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 697–700.

Johny, Cibu and Martin Jansche. 2018. Brahmic schwa-deletion with neural classifiers: Experiments with Bengali. In *Proceedings of the 6th International Workshop on Spoken Language Technologies for Under-Resourced Languages (SLTU)*, pages 264–268. https://doi.org/10.21437/SLTU.2018-55

Johny, Cibu, Lawrence Wolf-Sonkin, Alexander Gutkin, and Brian Roark. 2021. Finite-state script normalization and processing utilities: The Nisaba Brahmic library. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 14–23. https://doi.org/10.18653/v1/2021.eacl-demos.3

Karimi, Sarvnaz, Falk Scholer, and Andrew Turpin. 2011. Machine transliteration survey. *ACM Computing Surveys*, 43(3):1–46. https://doi.org/10.1145/1922649.1922654

Khakhmovich, Aleksandr, Svetlana Pavlova, Kira Kirillova, Nikolay Arefyev, and Ekaterina Savilova. 2020. Cross-lingual named entity list search via transliteration. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 4247–4255.

Khare, Shreya, Ashish Mittal, Anuj Diwan, Sunita Sarawagi, Preethi Jyothi, and Samarth Bharadwaj. 2021. Low resource ASR: The surprising effectiveness of high resource transliteration. In *Proceedings of Interspeech 2021*, pages 1529–1533. https://doi.org/10.21437/Interspeech.2021-2062

Khayrallah, Huda and Philipp Koehn. 2018. On the impact of various types of noise on neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 74–83. https://doi.org/10.18653/v1/W18-2709

Kingma, Diederik P. and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kneser, Reinhard and Hermann Ney. 1995. Improved backing-off for *m*-gram language modeling. In *Proceedings of 1995 International Conference on Acoustics, Speech, and Signal Processing (ICASSP '95)*, volume 1, pages 181–184. https://doi.org/10.1109/ICASSP.1995.479394

Knight, Kevin and Jonathan Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24(4):599–612.

Kreutzer, Julia, Isaac Caswell, Lisa Wang, Ahsan Wahab, Daan van Esch, Nasanbayar Ulzii-Orshikh, Allahsera Tapo, Nishant Subramani, Artem Sokolov, Claytone Sikasote, Monang Setyawan, Supheakmungkol Sarin, Sokhar Samb, Benoît Sagot, Clara Rivera, Annette Rios, Isabel Papadimitriou, Salomey Osei, Pedro Ortiz Suarez, Iroro Orife, Kelechi Ogueji, Andre Niyongabo Rubungo, Toan Q. Nguyen, Mathias Müller, André Müller, Shamsuddeen Hassan Muhammad, Nanda Muhammad, Ayanda Mnyakeni, Jamshidbek Mirzakhalov, Tapiwanashe Matangira, Colin Leong, Nze Lawson, Sneha Kudugunta, Yacine Jernite, Mathias Jenny, Orhan Firat, Bonaventure F. P. Dossou, Sakhile Dlamini, Nisansa de Silva, Sakine Çabuk Ballı, Stella Biderman, Alessia Battisti, Ahmed Baruwa, Ankur Bapna, Pallavi Baljekar, Israel Abebe Azime, Ayodele Awokoya, Duygu Ataman, Orevaoghene Ahia, Oghenefego Ahia, Sweta Agrawal, and Mofetoluwa Adeyemi. 2022. Quality at a glance: An audit of web-crawled multilingual datasets. *Transactions of the Association for Computational Linguistics*, 10:50–72. https://doi.org/10.1162/tacl_a_00447

Kudo, Taku 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: System Demonstrations*, pages 66–71. https://doi.org/10.18653/v1/D18-2012

Kumar, Ankit, Piyush Makhija, and Anuj Gupta. 2020. Noisy text data: Achilles' heel of BERT. In *Proceedings of the Sixth Workshop on Noisy User-generated Text (W-NUT 2020)*, pages 16–21. https://doi.org/10.18653/v1/2020.wnut-1.3

Kumar, Arun, Ryan Cotterell, Lluís Padró, and Antoni Oliver. 2017. Morphological analysis of the Dravidian language family. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 217–222. https://doi.org/10.18653/v1/E17-2035

Kunchukuttan, Anoop, Siddharth Jain, and Rahul Kejriwal. 2021. A large-scale evaluation of neural machine transliteration for Indic languages. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3469–3475. https://doi.org/10.18653/v1/2021.eacl-main.303

Kunchukuttan, Anoop, Mitesh Khapra, Gurneet Singh, and Pushpak Bhattacharyya. 2018. Leveraging orthographic similarity for multilingual neural transliteration. *Transactions of the Association for Computational Linguistics*, 6:303–316. https://doi.org/10.1162/tacl_a_00022

Kunchukuttan, Anoop, Ratish Puduppully, and Pushpak Bhattacharyya. 2015. Brahmi-net: A transliteration and script conversion system for languages of the Indian subcontinent. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 81–85. https://doi.org/10.3115/v1/N15-3017

Kundu, Soumyadeep, Sayantan Paul, and Santanu Pal. 2018. A deep learning based approach to transliteration. In *Proceedings of the Seventh Named Entities Workshop*, pages 79–83. https://doi.org/10.18653/v1/W18-2411

Lafferty, John, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In

Proceedings of the 18th International Conference on Machine Learning (ICML), pages 282–289.

Lee, En-Shiun, Sarubi Thillainathan, Shravan Nayak, Surangika Ranathunga, David Adelani, Ruisi Su, and Arya McCarthy. 2022. Pre-trained multilingual sequence-to-sequence models: A hope for low-resource language translation? In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 58–67. https://doi.org/10.18653/v1/2022.findings-acl.6

Lehal, Gurpreet Singh and Tejinder Singh Saini. 2012. Conversion between scripts of Punjabi: Beyond simple transliteration. In *Proceedings of COLING 2012: Posters*, pages 633–642.

Lehal, Gurpreet Singh and Tejinder Singh Saini. 2014. Sangam: A Perso-Arabic to Indic script machine transliteration model. In *Proceedings of the 11th International Conference on Natural Language Processing*, pages 232–239.

Lehmann, Thomas. 1993. *A Grammar of Modern Tamil*. Pondicherry Institute of Linguistics and Culture, Pondicherry, India.

Levenshtein, Vladimir I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics—Doklady*, 10(8):707–710.

Li, Haizhou, Min Zhang, and Jian Su. 2004. A joint source-channel model for machine transliteration. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 159–166. https://doi.org/10.3115/1218955.1218976

Liang, Davis, Hila Gonen, Yuning Mao, Rui Hou, Naman Goyal, Marjan Ghazvininejad, Luke Zettlemoyer, and Madian Khabsa. 2023. XLM-V: Overcoming the vocabulary bottleneck in multilingual masked language models. *arXiv preprint arXiv:2301.10472*. https://doi.org/10.48550/arXiv.2301.10472, https://doi.org/10.18653/v1/2023.emnlp-main.813

Luong, Thang, 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. https://doi.org/10.18653/v1/D15-1166

Madhani, Yash, Mitesh M. Khapra, and Anoop Kunchukuttan. 2023. Bhasa-abhijnaanam: Native-script and romanized language identification for 22 Indic languages. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 816–826. https://doi.org/10.18653/v1/2023.acl-short.71

Madhani, Yash, Sushane Parthan, Priyanka Bedekar, Ruchi Khapra, Vivek Seshadri, Anoop Kunchukuttan, Pratyush Kumar, and Mitesh M. Khapra. 2022. Aksharantar: Towards building open transliteration tools for the next billion users. *arXiv preprint arXiv:2205.03018*. https://doi.org/10.18653/v1/2023.findings-emnlp.4

Maleki, Jalal and Lars Ahrenberg. 2008. Converting romanized Persian to the Arabic writing systems. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*.

Markewich, Logan, Yubin Xing, Roy Ka-Wei Lee, Zhi Li, and Seokbum Ko. 2022. DReD—A descriptive relation dataset for expanding relation extraction. In *IEEE Transactions on Artificial Intelligence*, pages 1–10. https://doi.org/10.1109/TAI.2022.3205567

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

Mhaiskar, Rahul. 2015. Romanagari an alternative for modern media writings. *Bulletin of the Deccan College Post-Graduate and Research Institute*, 75:195–202.

Moezzi, Seyed Ali Reza, Abdolrahman Ghaedi, Mojdeh Rahmanian, Seyedeh Zahra Mousavi, and Ashkan Sami. 2023. Application of deep learning in generating structured radiology reports: A transformer-based technique. *Journal of Digital Imaging*, 36:80–90. https://doi.org/10.1007/s10278-022-00692-x PubMed: 36002778

Mohri, Mehryar. 2002. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350.

Moosa, Ibraheem Muhammad, Mahmud Elahi Akhter, and Ashfia Binte Habib. 2023. Does transliteration help multilingual language modeling? In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 670–685. https://doi.org/10.18653/v1/2023.findings-eacl.50

Moradi, Milad and Matthias Samwald. 2021. Evaluating the robustness of neural language models to input perturbations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1558–1570. https://doi.org/10.18653/v1/2021.emnlp-main.117

Moran, Molly and Constantine Lignos. 2020. Effective architectures for low resource multilingual named entity transliteration. In *Proceedings of the 3rd Workshop on Technologies for MT of Low Resource Languages*, pages 79–86.

Motlani, Raveesh. 2016. Developing language technology tools and resources for a resource-poor language: Sindhi. In *Proceedings of the NAACL Student Research Workshop*, pages 51–58. https://doi.org/10.18653/v1/N16-2008

Muller, Benjamin, Antonios Anastasopoulos, Benoît Sagot, and Djamé Seddah. 2021. When being unseen from mBERT is just the beginning: Handling new languages with multilingual language models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 448–462. https://doi.org/10.18653/v1/2021.naacl-main.38

Murikinati, Nikitha, Antonios Anastasopoulos, and Graham Neubig. 2020. Transliteration for cross-lingual morphological inflection. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 189–197. https://doi.org/10.18653/v1/2020.sigmorphon-1.22

Murphy, Anne. 2018. Writing Punjabi across borders. *South Asian History and Culture*, 9(1):68–91. https://doi.org/10.1080/19472498.2017.1411049

Nagoudi, El Moatez Billah, AbdelRahim Elmadany, and Muhammad Abdul-Mageed. 2022. AraT5: Text-to-text transformers for Arabic language generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 628–647. https://doi.org/10.18653/v1/2022.acl-long.47

Najafi, Saeed, Bradley Hauer, Rashed Rubby Riyadh, Leyuan Yu, and Grzegorz Kondrak. 2018. Comparison of assorted models for transliteration. In *Proceedings of the Seventh Named Entities Workshop*, pages 84–88. https://doi.org/10.18653/v1/W18-2412

Ney, Hermann, Dieter Mergel, Andreas Noll, and Annedore Paeseler. 1987. A data-driven organization of the dynamic programming beam search for continuous speech recognition. In *Proceedings of the IEEE 1987 International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 12, pages 833–836. https://doi.org/10.1109/ICASSP.1987.1169844

Nicolai, Garrett, Bradley Hauer, Mohammad Salameh, Adam St Arnaud, Ying Xu, Lei Yao, and Grzegorz Kondrak. 2015. Multiple system combination for transliteration. In *Proceedings of the Fifth Named Entity Workshop*, pages 72–77. https://doi.org/10.18653/v1/W15-3911

Nielsen, Elizabeth, Christo Kirov, and Brian Roark. 2023. Distinguishing romanized Hindi from romanized Urdu. In *Proceedings of the Workshop on Computation and Written Language (CAWL 2023)*, pages 33–42. https://doi.org/10.18653/v1/2023.cawl-1.5

Pele, Ofir and Michael Werman. 2008. A linear time histogram metric for improved SIFT matching. In *Computer Vision–ECCV 2008*, pages 495–508. https://doi.org/10.1007/978-3-540-88690-7_37

Pele, Ofir and Michael Werman. 2009. Fast and robust earth mover's distances. In *2009 IEEE 12th International Conference on Computer Vision*, pages 460–467. https://doi.org/10.1109/ICCV.2009.5459199

Rabiner, Lawrence R. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286. https://doi.org/10.1109/5.18626

Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(1):5485–5551.

Riyadh, Rashed Rubby and Grzegorz Kondrak. 2019. Joint approach to deromanization of code-mixed texts. In *Proceedings of the Sixth Workshop on NLP for Similar Languages, Varieties and Dialects*, pages 26–34. https://doi.org/10.18653/v1/W19-1403

Roark, Brian, Richard Sproat, Cyril Allauzen, Michael Riley, Jeffrey Sorensen, and Terry Tai. 2012. The OpenGrm open-source finite-state grammar software libraries. In *Proceedings of the ACL 2012 System Demonstrations*, pages 61–66.

Roark, Brian, Lawrence Wolf-Sonkin, Christo Kirov, Sabrina J. Mielke, Cibu Johny, Isin Demirsahin, and Keith Hall. 2020. Processing South Asian languages written in the Latin script: The Dakshina dataset. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 2413–2423.

Roberts, Adam, Hyung Won Chung, Anselm Levskaya, Gaurav Mishra, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, Curtis Hawthorne, Aitor Lewkowycz, Alex Salcianu, Marc van Zee, Jacob Austin, Sebastian Goodman, Livio Baldini Soares, Haitang Hu, Sasha Tsvyashchenko, Aakanksha Chowdhery, Jasmijn Bastings, Jannis Bulian, Xavier Garcia, Jianmo Ni, Andrew Chen, Kathleen Kenealy, Jonathan H. Clark, Stephan Lee, Dan Garrette, James Lee-Thorp, Colin Raffel, Noam Shazeer, Marvin Ritter, Maarten Bosma, Alexandre Passos, Jeremy Maitin-Shepard, Noah Fiedel, Mark Omernick, Brennan Saeta, Ryan Sepassi, Alexander Spiridonov, Joshua Newlan, and Andrea Gesmundo. 2022. Scaling up models and data with `t5x` and `seqio`. *arXiv preprint arXiv:2203.17189*. https://doi.org/10.48550/arXiv.2203.17189

Ruder, Sebastian, Noah Constant, Jan Botha, Aditya Siddhant, Orhan Firat, Jinlan Fu, Pengfei Liu, Junjie Hu, Dan Garrette, Graham Neubig, and Melvin Johnson. 2021. XTREME-R: Towards more challenging and nuanced multilingual evaluation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10215–10245. https://doi.org/10.18653/v1/2021.emnlp-main.802

Russell, David James. 2014. *Multiple Sequence Alignment Methods*, volume 1079 of *Methods in Molecular Biology*. Springer. https://doi.org/10.1007/978-1-62703-646-7

Salomon, Richard G. 1996. Brahmi and Kharoshthi, Peter T. Daniels and William Bright, editors, *The World's Writing Systems*, Oxford University Press, pages 373–383.

Samaranayake, V. K., S. T. Nandasara, J. B. Disanayaka, A. R. Weerasinghe, and H. Wijayawardhana. 2003. An introduction to UNICODE for Sinhala characters. Technical Report UCSC 03/01, University Of Colombo, School of Computing, Colombo, Sri Lanka.

Schiffman, Harold F. 2008. The Ausbau issue in the Dravidian languages: The case of Tamil and the problem of purism. *International Journal of the Sociology of Language*, 2008(191):45–63. https://doi.org/10.1515/IJSL.2008.024

Schoch, Stephanie, Ritwick Mishra, and Yangfeng Ji. 2023. Data selection for fine-tuning large language models using transferred Shapley values. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 266–275. https://doi.org/10.18653/v1/2023.acl-srw.37

Schuster, Mike and Kaisuke Nakajima. 2012. Japanese and Korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152. https://doi.org/10.1109/ICASSP.2012.6289079

Sennrich, Rico, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96. https://doi.org/10.18653/v1/P16-1009

Sodhar, Irum Naz, Akhtar Hussain Jalbani, Muhammad Ibrahim Channa, and Dil Nawaz Hakro. 2019. Identification of issues and challenges in romanized Sindhi text. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 10(9):229–233. https://doi.org/10.14569/IJACSA.2019.0100929

Spohrer, James C., Peter F. Brown, P. H. Hochschild, and James K. Baker. 1980. Partial traceback in continuous speech recognition. In *Proceedings of the IEEE 1980 International Conference on Cybernetics and Society (ICCS)*, pages 36–42.

Steever, Sanford B. 1987. Tamil and the Dravidian languages. In Bernard Comrie, editor, *The World's Major Languages*, Oxford University Press, pages 725–746. https://doi.org/10.4324/9780203214961-36

Steever, Sanford B. 2019. *The Dravidian Languages*, 2nd edition. Routledge Language Family Series. Routledge. https://doi.org/10.4324/9781315722580, PubMed: 31431302

Taylor, Paul. 2009. *Text-to-Speech Synthesis*. Cambridge University Press. https://doi.org/10.1017/CBO9780511816338

Unicode Consortium. 2022. South and Central Asia - I. In *The Unicode Standard*

(Version 15.0.0). Unicode Consortium, chapter 12, pages 461–532.

United Nations. 2007. Technical reference manual for the standardization of geographical names. Technical Report ST/ESA/STAT/SER.M/87, United Nations, Department of Economic and Social Affairs, Statistics Division, New York. United Nations Group of Experts on Geographical Names. URL https://unstats.un.org/unsd/geoinfo/ungegn/docs/pubs/UNGEGN%20tech%20ref%20manual_m87_combined.pdf.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Virga, Paola and Sanjeev Khudanpur. 2003. Transliteration of proper names in cross-lingual information retrieval. In *Proceedings of the ACL 2003 Workshop on Multilingual and Mixed-language Named Entity Recognition*, pages 57–64. https://doi.org/10.3115/1119384.1119392

Voutilainen, Atro. 2003. Part-of-speech tagging. In Ruslan Mitkov, editor, *The Oxford Handbook of Computational Linguistics*. Oxford University Press, chapter 11, pages 219–232.

Wang, Hai, Dian Yu, Kai Sun, Jianshu Chen, and Dong Yu. 2019. Improving pre-trained multilingual model with vocabulary expansion. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 316–327. https://doi.org/10.18653/v1/K19-1030

Wellisch, Hans H. 1978. *The Conversion of Scripts: Its Nature, History, and Utilization*. Information Sciences Series. John Wiley & Sons.

Wijayawardhana, Harsha, Asanka Wasala, Ruvan Weerasinghe, and Chamila Liyanage. 2008. Implementation of Internet domain names in Sinhala. In *Proceedings of International Symposium on Country Domain Governance (CDG)*, pages 20–23.

Winkler, William E. 1990. String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. In *Proceedings of the Section on Survey Research of American Statistical Association (ASA)*, pages 354–359.

Witten, Ian H. and Timothy C. Bell. 1991. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094. https://doi.org/10.1109/18.87000

Wolf-Sonkin, Lawrence, Vlad Schogol, Brian Roark, and Michael Riley. 2019. Latin script keyboards for South Asian languages with finite-state normalization. In *Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing*, pages 108–117. https://doi.org/10.18653/v1/W19-3114

Wu, Chun Kai, Chao-Chuang Shih, Yu-Chun Wang, and Richard Tzong-Han Tsai. 2022. Improving low-resource machine transliteration by using 3-way transfer learning. *Computer Speech & Language*, 72:Article 101283. https://doi.org/10.1016/j.csl.2021.101283

Xue, Linting, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306. https://doi.org/10.1162/tacl_a_00461

Xue, Linting, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498. https://doi.org/10.18653/v1/2021.naacl-main.41

Yu, Dong and Li Deng. 2015. *Automatic Speech Recognition: A Deep Learning Approach*, volume 1 of *Signals and Communication Technology*. Springer. https://doi.org/10.1007/978-1-4471-5779-3

Zhang, Hao, Richard Sproat, Axel H. Ng, Felix Stahlberg, Xiaochang Peng, Kyle Gorman, and Brian Roark. 2019. Neural models of text normalization for speech applications. *Computational Linguistics*, 45(2):293–337. https://doi.org/10.1162/coli_a_00349