Shaped by the Script: Designing the Representation Around the Writing System for Efficient Low-Resource Transliteration

streetsolider
https://github.com/streetsolider

September 2026

Abstract

Our Latin→Thaana transliteration service runs a 300M-parameter ByT5 fine-tune on a GPU, at about 356 ms per sentence, behind a server. We wanted the same thing on a phone, with no server at all. Encouraged by the small byte-level models that serve grapheme-to-phoneme conversion for a hundred languages in a few million parameters, we trained a 9M-parameter T5-style encoder–decoder from scratch on the same task and the same data, in three stages: public news pairs, then the length augmentations the large model uses, then sequence-level distillation from the large model over a Dhivehi Latin chat text corpus. On the public news test split the small model matches the large one on character accuracy (0.919 vs. 0.919) and beats it on exact match (0.286 vs. 0.247); on a held-out set of Dhivehi Latin chat text it leads on word accuracy (0.804 vs. 0.797) and trails on character accuracy (0.920 vs. 0.955). Exported to ONNX and quantized to 8 bits it is a 9.6 MB download that transliterates a sentence in 35 ms on a laptop CPU inside the browser, with quantization costing nothing measurable. Along the way we found that one- to three-word inputs are a failure mode of any model trained on sentence pairs, that the merged ONNX decoder silently escapes dynamic quantization, and that the small model could not learn chat text from the news corpus alone: distillation from the large model is what closed that gap. So the fair summary is not that small beat large, but that a specialised student overtook its teacher in-domain and runs anywhere. We release the model, the page and the evaluation harness.

1 Introduction

Dhivehi is written in Thaana, but most Dhivehi typed on phones is written in Latin letters, in a loose orthography where *miadhu*, *miyadhu* and *miadu* are the same word. Turning that into Thaana is a transliteration problem, and in earlier work [11, 12] we solved it by fine-tuning a 300M-parameter ByT5-small [14], starting from Neobe's Dhivehi checkpoint [8], to emit an ASCII keymap that a table turns into Thaana. That model works well, and it is the one our service runs. It is also 1.2 GB of weights that need a GPU, and every keystroke a user types travels to a server and back.

We wanted the opposite deployment: the model inside the web page, downloaded once, running on whatever device the user has, with the text never leaving it. That rules out the 300M model on size alone. What made us think a much smaller model might do is the grapheme-to-phoneme literature. Zhu et al. [15] serve a hundred languages from a ByT5 with $d_{\text{model}} = 256$ and a few million parameters, and grapheme-to-phoneme is the same shape of problem as ours: a monotonic, character-level mapping over short inputs, where a byte-level model needs no vocabulary and where most of the capacity of a large pretrained model is spent on things the task never asks for. More generally, there is a pattern across speech, spelling correction and named-entity transliteration [3] of tiny models trained on one narrow task holding their own against general models a hundred times their size. We wanted to know whether that held here.

So this paper asks a narrow question: how small can a Latin→Thaana transliterator be and still match the 300M model, and can it be made to run in a browser? We train a 9M-parameter model from scratch, in three stages that we check one at a time on the same tests the large model has to pass, and we export it to run in the page. We find that it matches or beats the large model on the public news test split, that it needed the large model as a teacher to catch up on Dhivehi Latin chat text, and that it runs in 35 ms per sentence on a CPU in the browser from a 9.6 MB download. We also report three things that went wrong on the way, because each of them cost us a day and would cost the next person the same.

2 Background

2.1 The task and the keymap

Thaana is an abugida written right to left: every consonant carries a vowel sign (*fili*) or a sukun, so މިއަދު ("today") is five consonants and five marks. Its codepoints (U+0780–U+07BF) take three bytes each in UTF-8. Our earlier work [12] sidestepped that cost with the Segha phonetic keyboard layout [4], which maps every Thaana letter and mark to one ASCII byte, so a model emits `miwadu` and a 55-entry table produces މިއަދު. A byte-level decoder then spends one step per Thaana character instead of three. We keep that design unchanged here; it matters even more for a model whose whole point is to be cheap.

The Latin side has no standard. The same word is spelled several ways, English words and names are mixed in, and word boundaries do not line up with Thaana's: chat writes *male gai* where Thaana writes މާލޭގައި, with the stem's final vowel changing when the suffix attaches. This is why the task is a sequence-to-sequence problem rather than a lookup, and why word-level dictionaries plateau early.

2.2 Byte-level models and small specialised ones

ByT5 [14] is the T5 architecture [10] applied to UTF-8 bytes (Figure 1): no tokenizer, a 384-entry "vocabulary" of the 256 byte values plus special tokens, and a deliberately encoder-heavy layout. The published sizes start at 300M parameters, because they are meant to be pretrained on a large corpus and fine-tuned for anything. Nothing about the architecture requires that scale. Zhu et al. [15] showed that the same layout at $d_{\text{model}} = 256$, with 8 to 16 layers, learns grapheme-to-phoneme conversion for 100 languages, and that it does so trained from scratch on the task data. We took their configuration as our starting point and made it smaller still.

2.3 Distillation

When a small student is trained on a large teacher's outputs rather than on gold labels, it usually learns more from the same amount of text, because the teacher's outputs are consistent in a way that human labels are not [5]. For sequence models the simplest form is sequence-level distillation [7]: run the teacher, keep its outputs, train the student on them as if they were references. We use exactly that, with one addition described in Section 3.2: we had already found and corrected the teacher's systematic errors on the distillation corpus, so the student learns from a better teacher than the one we serve.

3 Method

3.1 Architecture

The model (Figure 2) is a T5 encoder–decoder with the ByT5 byte tokenizer: 6 encoder and 2 decoder layers, $d_{\text{model}} = 256$, $d_{\text{ff}} = 1024$, 4 heads of width 64, gated-GELU feed-forward blocks, T5's relative position bias, RMS normalisation, and input and output embeddings tied to one 384×256 matrix. That is 9.0M parameters. The 6-to-2 split follows ByT5's argument that byte-level models should be encoder-heavy, because the decoder's job (which byte comes next given a good encoding) is the easy half. We also prepared a 12-plus-4 layer variant at 18M parameters as a fallback and never needed it.

We deliberately kept everything else identical to the 300M model: the same tokenizer, the same corpus format, the same keymap, the same evaluation scripts. This is what let us run the small model through every check the large one has to pass without writing anything new, and it is why we can compare the two honestly.

3.2 Data and stages

Training data came in three stages, and we checked each stage before adding the next.

Stage 1, news only. The 150,326 training pairs of the public `dhivehi-transliteration-pairs` corpus [1]: news headlines and sentences with a reviewed Latin transliteration, the Thaana side converted to keymap. This is the only fully trusted parallel data we have, so stage 1 measures the architecture, not the data.

Stage 2, augmentation. The same length-diverse augmentations the 300M fine-tune uses: single words mined from parity rows and from the project lexicon, short two-to-three-word phrases, long concatenations, place names (islands, atoll codes, house names) each with two or three chat misspellings mapped to the same Thaana, and English loanwords with the spelling a Dhivehi lexicographer gives them. 173,353 pairs.

Stage 3, distillation. The service has a corpus of Dhivehi typed in Latin letters in messages, which we call the Dhivehi Latin chat text corpus, and the 300M model had already transliterated all of it. In separate work we had built a token-level error detector for that output and corrected its systematic errors: English words read by Dhivehi Latin rules (*to* → ތޯ instead of ޓު), merged forms, and a few hundred spellings a person had rejected. Split into sentences and filtered to parallel pairs under 250 bytes, that gave 73,265 teacher-labelled sentence pairs, which we added to the stage 2

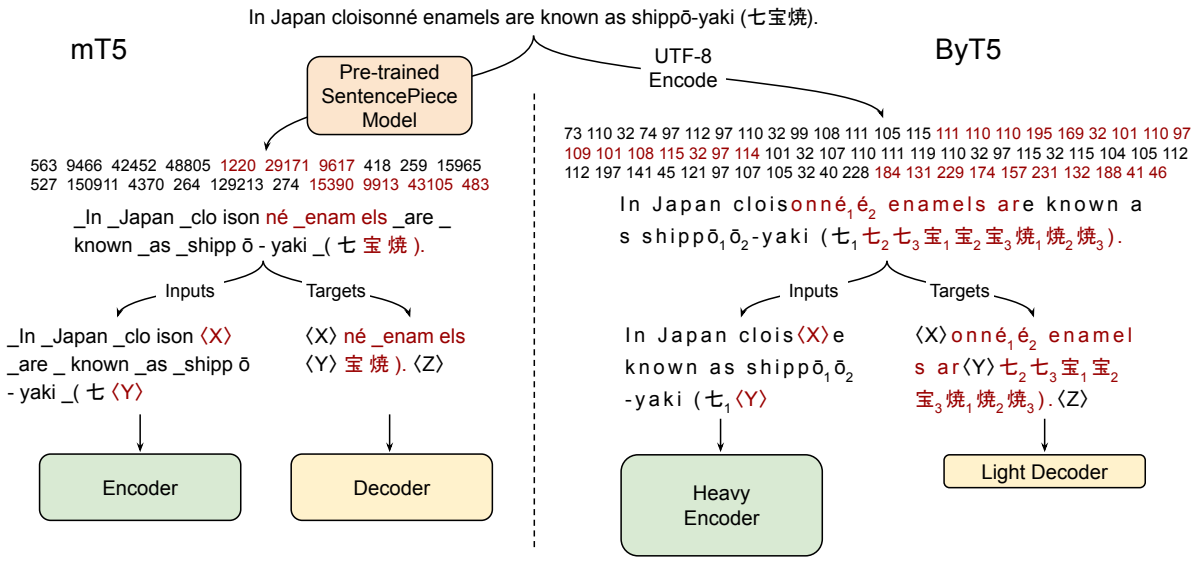

Figure 1: How mT5 and ByT5 read text, reproduced from Xue et al. [14] (Figure 1 there; CC BY 4.0). mT5 splits text into SentencePiece tokens and uses encoder and decoder stacks of equal depth; ByT5 reads raw UTF-8 bytes with no tokenizer, and makes the encoder three times deeper than the decoder. Our model keeps the byte input and the heavy-encoder, light-decoder shape, at a tiny fraction of the size.

corpus for 246,618 pairs in total. Because the corrections had already been applied, this is distillation from a teacher slightly better than the one in production. The corpus itself is not redistributed.

3.3 Training

All runs use the same recipe: from scratch, AdamW, learning rate 10^{-3} with 5% linear warm-up and linear decay, batch 64, bfloat16, dropout 0.1, weight decay 0.01, 20 epochs, evaluation and checkpoint at every epoch boundary on a fixed 1,000-row subset of the test split, best checkpoint by character accuracy, early stopping with patience 3. Stage 1 took 49 minutes and stage 3 took 1 h 53 min on one RTX 5070 Ti; the small model trains at about 15 optimizer steps per second. We also trained one warm-started variant of stage 3, continuing from the stage 2 checkpoint at a lower learning rate for 8 epochs, to see whether a curriculum beat training from scratch on everything.

3.4 Evaluation

Every checkpoint is scored the same four ways as the 300M model.

1. The full 37,582-row news test split, beam 4 and greedy. We report exact match, character accuracy (one minus the edit distance between output and reference, normalised by the longer of the two), and the *in-lexicon rate*, the share of output words that appear in a Dhivehi word list, which catches a model inventing words.
2. A *chat test set* of 102 sentences from the Dhivehi Latin chat text corpus whose Thaana a person has verified, reporting character and word accuracy. This is the deployment domain.
3. Four one-word inputs (*bohkuraa*, *kuru*, *karu*, *bas*) that the original ByT5 checkpoint got wrong and the 300M fine-tune exists to fix. We report how many of the four come out right.
4. The exported ONNX model scored on 1,000 test rows through the same JavaScript runtime the page uses, which is the number that matters for deployment.

3.5 Export and the page

The checkpoint is exported with optimum [6] to ONNX as an encoder and a merged decoder (the with-cache and without-cache decoders combined into one graph behind an `If` node), quantized to 8-bit integers for WebAssembly and converted to fp16 for WebGPU, and loaded in the page with Transformers.js [13] on ONNX Runtime [9]. Transformers.js has no ByT5 tokenizer, and it does not need one: encoding is `byte + 3` with an end-of-sequence token, decoding is the reverse, ten lines of JavaScript. The page ports the service's chunking unchanged: paragraphs, then sentences on `.!?`, then phrases on `,;`, then ten-word chunks, one batched `generate` per paragraph, stitched back with

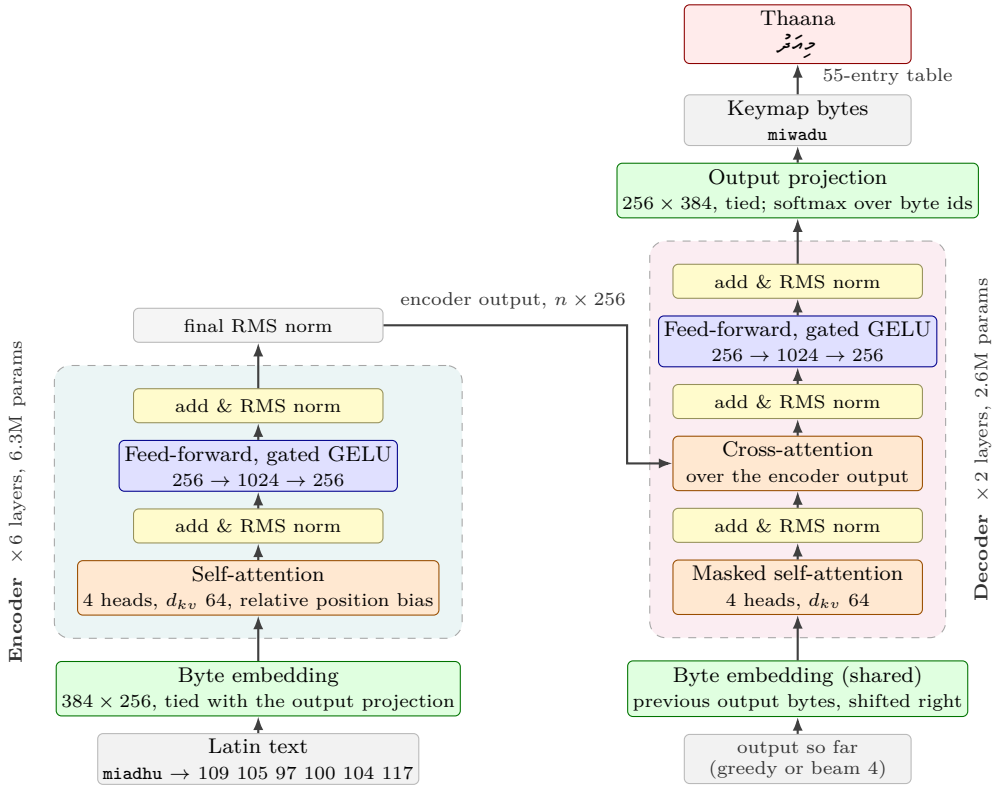

Figure 2: The model, drawn layer by layer. Bytes go in offset by three (pad, eos, unk); six identical encoder layers of self-attention and a gated-GELU feed-forward block build a 256-wide representation; two decoder layers attend over it and predict one keymap byte at a time through an output projection tied to the input embedding. A 55-entry table turns the ASCII keymap into Thaana. 9.0M parameters in total: 6.3M in the encoder, 2.6M in the decoder, 0.1M in the shared embedding.

	news test		chat test		
System	exact	char	char	word	1-word
300M, served [12]	0.247	0.919	**0.955**	0.797	4/4
9M stage 1 (news)	**0.302**	**0.925**	0.884	0.784	1/4
9M stage 2 (+augment.)	0.282	0.912	0.880	0.768	4/4
9M stage 3 (+distil.)	0.286	0.919	0.920	**0.804**	4/4
warm-started	0.279	0.914	0.908	0.805	4/4

Table 1: Accuracy, beam 4. "news test" is the 37,582-row public split; "chat test" the 102 verified sentences of Dhivehi Latin chat text; "1-word" the four one-word inputs. The released model is the stage 3 row. In-lexicon rates on the news split: 300M 0.952, stage 1 0.974, stage 2 0.969, stage 3 0.970.

the original punctuation and the Arabic comma, semicolon and question mark.

4 Results

4.1 The small model matches the large one on news

Table 1 is the main result. On the public news test split the stage 1 model, trained from scratch on nothing but the news pairs, is already ahead of the 300M fine-tune on every metric: exact match 0.302 against 0.247, character accuracy 0.925 against 0.919, and a higher share of real lexicon words. The released stage 3 model gives a little of that back (0.919 character accuracy, equal to the large model's) in exchange for the chat-text gains below, and it gets all four one-word inputs right where stage 1 gets one.

We should say what exact match means here. Many reference rows in this test split are a different headline from their source rather than a transliteration of it, so exact match has a low ceiling for any model and small differences in it are not meaningful. Character accuracy is the number we trust, and on that number the small and large models are tied on news.

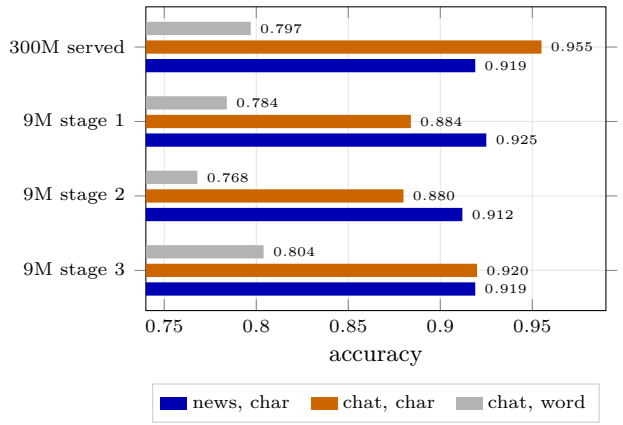

Figure 3: Character accuracy on the news test split and on the chat test set, and word accuracy on the chat test set, for the served 300M model and the three stages of the 9M model (beam 4). The axis starts at 0.74 so the differences are visible. Stage 1 wins on news and loses on chat text; stage 3 closes most of the chat gap without giving up news.

4.2 Chat text is where the small model needed a teacher

The chat columns tell the other half of the story. Trained on news alone, the 9M model reaches 0.884 character accuracy on the chat test set against the large model's 0.955. The augmentations of stage 2 do not help here at all (0.880); they fix short inputs, which is a different problem. Distillation is what moves the number: stage 3 reaches 0.920, and on word accuracy it is ahead of the large model, 0.804 against 0.797.

That result needs a careful reading. The student was trained on the teacher's outputs over exactly this kind of text, with the teacher's known errors corrected, so the student is not discovering Dhivehi Latin chat text from news; it is compressing what the teacher already knew, plus the corrections. The remaining gap in character accuracy (0.920 vs. 0.955) is the part of the teacher's knowledge that 73k sentences did not transfer, and it is character-level detail: vowel length and a sukun here and there, rarely a wrong word, which is why word accuracy is at parity while character accuracy is not.

The warm-started variant, which continued from stage 2 on the same stage 3 corpus, came out about a point lower on every test than training from scratch on the full corpus. We had expected the curriculum to help and it did not; the model is small enough that a fresh run over everything is the better use of two hours.

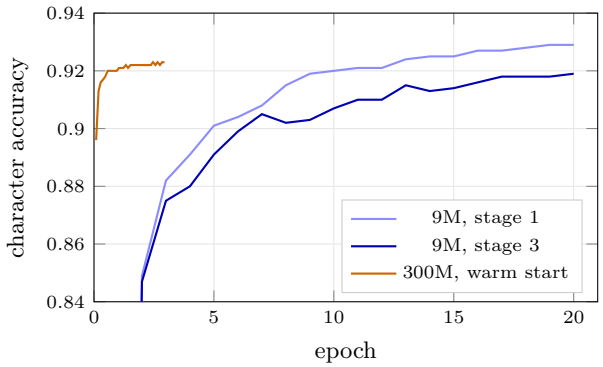

Figure 4: Character accuracy on the fixed 1,000-row evaluation subset after each epoch. The 9M runs start from random weights and need about eight epochs to reach where the 300M model, warm-started from a pretrained checkpoint, is after a fraction of one. Early stopping never fired: the curves keep creeping up until the learning rate reaches zero.

4.3 Short inputs are a failure mode of sentence-trained models

The stage 1 model gets one of the four one-word inputs right. The three failures are the same failure: given a single word it emits the word twice, *bas* → `basq, basq` (ބަސް، ބަސް). The cause is plain once seen. The news corpus has no one-word rows and only 538 rows of two or three words, so the model has never seen an input end that early and has learned that outputs are longer than one word. The fix is the stage 2 augmentation, after which all four come out right, and it is worth stating as a rule: any transliterator trained on sentence pairs needs single-word and short-phrase pairs added on purpose, or it will fail exactly on the inputs a user types first.

4.4 Quantization costs nothing, once it is actually applied

Scored through the browser runtime on 1,000 test rows, the released model gives 0.920 character accuracy at 8-bit and 0.920 at fp32: the quantization is free. It was not free to get right. Dynamic quantization of the merged ONNX decoder produced a file of the same 11.1 MB as the fp32 original, because every matrix multiplication in the merged graph lives inside the branches of the `If` node that selects the cached or uncached path, and ONNX Runtime's dynamic quantizer does not descend into subgraphs. Quantizing the two decoder halves separately and merging the quantized halves gives a 3.0 MB decoder. With the 6.6 MB int8 encoder, the whole download is 9.6 MB, against 1.2 GB for the served model's weights.

Configuration	size	per sentence
300M, GPU, service path, beam 4	1.2 GB	356 ms
300M, GPU, batch 1, greedy	1.2 GB	72 ms
9M, GPU, batch 1, greedy	36 MB	53 ms
9M, browser CPU, WASM, int8	9.6 MB	35 ms

Table 2: Latency per sentence. GPU rows on one RTX 5070 Ti; the browser row in Node's WebAssembly runtime on a laptop CPU, which is what Chrome runs. Sizes are what has to be loaded.

4.5 Speed

Table 2 gives the latencies. On a GPU at batch 1 the 9M model is only modestly faster than the 300M (53 vs. 72 ms): at this size a GPU decode loop is overhead-bound and the weights are not what it waits for. The number that changes the deployment is the last row. In the browser, on a laptop CPU, with no server, the model transliterates a sentence in 35 ms from a 9.6 MB download. Enabling cross-origin isolation, which lets the WASM runtime use threads, halved the per-sentence time in our testing, so the demo server sends those headers.

5 Memorisation

The stage 3 corpus is 30% teacher-labelled sentences from the Dhivehi Latin chat text corpus, which we do not redistribute, so before releasing the weights we checked whether the model emits text from it. The question is not whether the model can transliterate one of those sentences when given it, which is its job and which reveals nothing the caller does not already hold, but whether it produces text from the corpus that it was not given [2]. We built the set of word 4-grams and character 15-grams that occur in the corpus's Thaana and in none of the public corpora we train or evaluate on, then ran two probes against it: 60 short public prompts with a 256-byte generation budget under greedy and beam 4 decoding, and the first two Latin words of 300 corpus sentences, scored only on words the model added beyond its prompt and checked against the true continuation. Nothing was emitted in the first probe. In the second, six of 300 outputs added words beyond the prefix, and none of them matched the sentence's continuation. We release the probe alongside the model, and we note that a first version of it reported hits until we realised a 15-character window fits inside the transliteration of the two prompt words themselves.

6 Discussion and limitations

What the result is and is not. On the public news split a 9M model trained from scratch equals a 300M pretrained-and-fine-tuned model, and on exact match it is ahead. That is the specialised-small-model result we set out to test, and it held. On Dhivehi Latin chat text the small model could not get there from public data; it got there by distilling the large model's outputs, which means the large model was still necessary, once, to produce the corpus. We think that is the general shape of the answer for tasks like this: the small model can replace the large one in production, but the large one earns its keep as a teacher.

The student is bounded by the teacher. The residual gap in chat character accuracy is the teacher's own imperfection on that text plus what did not transfer. A better teacher, or a human-verified chat corpus, is the only way to close it; more news data will not.

Evaluation noise. The chat test set is 102 sentences, which is enough to separate stage 1 from stage 3 but not to resolve tenths of a point between the two stage 3 variants. The news exact-match numbers carry the reference noise described in Section 4. All runs use a single seed.

Runaway generation. The decoder has no length limit of its own, and a rare unseen name can send it to the hard cap. The page limits generation to 2.5× the input bytes plus 16, which is generous for any real output and stops the loop early; a length-ratio check is also the right detector for this in evaluation, as it was for the quantized translation models in our earlier work.

What we did not try. The 18M variant, a larger distillation corpus, WebGPU on phones, and beam search in the page (we ship greedy; the gap on this task is small). Each is a day's work with the released harness.

7 Conclusion

A 9M-parameter byte-level transliterator, trained from scratch in under two hours on one GPU, matches the 300M ByT5 fine-tune our service runs on the public news test split, gets the same four one-word inputs right, and, after distillation from that model over Dhivehi Latin chat text, comes within three and a half points of it on chat character accuracy while leading on word accuracy. Exported to ONNX and quantized to 8 bits it is a 9.6 MB download that transliterates a sentence in 35 ms on a laptop CPU inside the browser, with no server and no text leaving the device. Three findings travel beyond this task: sentence-

trained transliterators fail on one-word inputs unless short pairs are added on purpose; ONNX's merged decoder graph escapes dynamic quantization unless the halves are quantized before merging; and a small specialised model can overtake its teacher in-domain, but on the text it never saw it still needs the teacher's outputs to learn from.

Artifacts. The model, the page, the byte tokenizer, the chunk-and-stitch pipeline, the memorisation probe and the browser harness are at `https://github.com/streetsolider/div-transliteration-web`. The served 300M model this work compares against is `https://huggingface.co/str33t/dhivehi-byt5-latin2thaana-keymap-v1`, and the keymap and evaluation scripts shared with it are in `https://github.com/streetsolider/div-transliteration`.

References

[1] alakxender. dhivehi-transliteration-pairs dataset. `https://huggingface.co/datasets/alakxender/dhivehi-transliteration-pairs`, 2025.

[2] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In *Proceedings of the 30th USENIX Security Symposium*, 2021.

[3] Nancy Chen, Rafael E. Banchs, Xiangyu Duan, Min Zhang, and Haizhou Li. Report of NEWS 2018 named entity transliteration shared task. In *Proceedings of the Seventh Named Entities Workshop*, 2018.

[4] Jawish Hameed. jtk: JavaScript Thaana keyboard. `https://github.com/jawish/jtk`, 2014.

[5] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[6] Hugging Face. Optimum: ONNX export and quantization for transformers models. `https://github.com/huggingface/optimum`, 2025.

[7] Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Proceedings of EMNLP*, 2016.

[8] Neobe. dhivehi-byt5-latin2thaana-v1. `https://huggingface.co/Neobe/dhivehi-byt5-latin2thaana-v1`, 2026.

[9] ONNX Runtime developers. ONNX runtime. `https://onnxruntime.ai`, 2025.

[10] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[11] streetsolider. div-transliteration: the Segha keymap for Thaana. `https://github.com/streetsolider/div-transliteration`, 2026.

[12] streetsolider. dhivehi-byt5-latin2thaana-keymap-v1. `https://huggingface.co/str33t/dhivehi-byt5-latin2thaana-keymap-v1`, 2026.

[13] Xenova and Hugging Face. Transformers.js: State-of-the-art machine learning for the web. `https://github.com/huggingface/transformers.js`, 2025.

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

[15] Jian Zhu, Cong Zhang, and David Jurgens. ByT5 model for massively multilingual grapheme-to-phoneme conversion. In *Proceedings of Interspeech*, 2022.