DelT5: Dynamic Token Deletion for Efficient Byte-level Language Models

Stanford CS224N Custom Project

Julie Kallini
Department of Computer Science
Stanford University
kallini@stanford.edu

Abstract

Models that rely on subword tokenization have significant drawbacks, such as sensitivity to character-level noise and inconsistent compression rates across different languages and scripts. Although character or byte-level models like ByT5 aim to address these concerns, they have not achieved widespread adoption due to inefficiencies in training and inference times, along with extremely short sequence lengths that make them impractical for real use-cases. In this project, we introduce DelT5, an efficient variant of ByT5 that uses a novel token deletion mechanism in its encoder to dynamically reduce its input sequence length. Additionally, DelT5 utilizes an auxiliary loss function to adjust token deletion rates, balancing computational efficiency with model performance. We find that DelT5 can achieve significant gains in inference runtime with minimal effect on loss. This approach presents a potential solution to the practical limitations of existing byte-level models.

1 Key Information to include

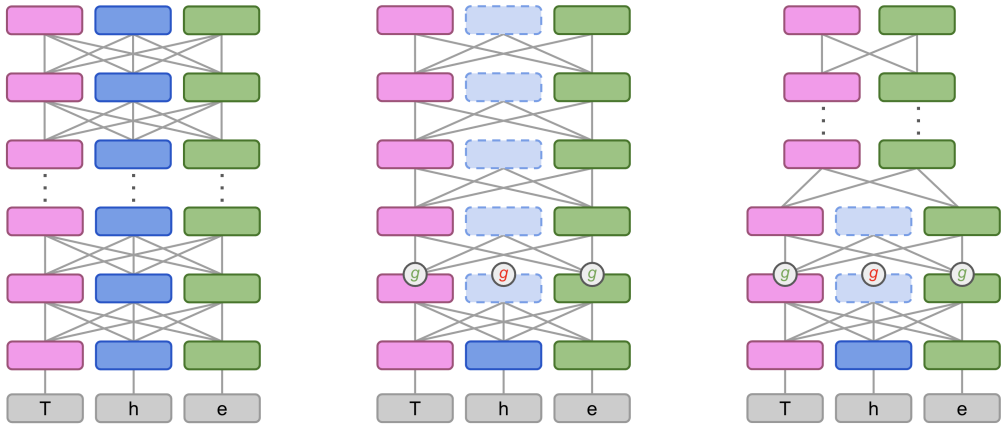
Mentor: Shikhar Murty (internal), Róbert Csordás (external); **External Collaborators**: Christopher Manning, Christopher Potts (research advisors); **Sharing project**: N/A.

2 Introduction

Subword tokenization, using algorithms such as byte-pair encoding (Sennrich et al., 2016) or SentencePiece (Kudo and Richardson, 2018), is a fundamental text preprocessing step that has become ubiquitious in modern language models. Subword tokenizers divide text into meaningful units known as *tokens*, which closely resemble words or parts of words. Tokenization can be seen as a form of compression, since it reduces the sequence length of the input passed to the compute-intensive Transformer (Vaswani et al., 2017). However, subword tokenizers have several drawbacks. For example, they are not very robust to character-level noise and manipulations, such as spelling errors (Kaushal and Mahowald, 2022; Huang et al., 2023); tokenization directly impacts how models process digits and perform arithmetic (Singh and Strouse, 2024); and tokenizers can have disproportionate compression rates for different languages and scripts (Ahia et al., 2023; Petrov et al., 2023). Today, language model APIs charge users per-token, and such discrepancies can cause users of certain languages to be overcharged for poorer results.[1]

As an alternative to subword models, *tokenization-free* models skip the tokenization preprocessing step entirely by passing the raw character or byte stream directly as input. However, character- or byte-level sequences tend to be significantly longer than tokenized text sequences, which is a

[1] See also Andrej Karpathy's tweets on tokenization: https://twitter.com/karpathy/status/1759996551378940395; https://twitter.com/karpathy/status/1657949234535211009



(a) ByT5. (b) DelT5 with soft deletion. (c) DelT5 with hard deletion.

Figure 1: A comparison of the encoder architecture for (a) ByT5, (b) DelT5 with attention masking as a form of soft deletion, and (c) DelT5 with hard deletion. In these diagrams, the DelT5 deletion gate is applied at layer 2. The gating mechanism $\mathbf{g}$ determines whether a token is deleted or not. Soft or hard deletion may be used during training; hard deletion is always applied at inference.

problem for Transformer models, which have quadratic time and space complexity with respect to the input sequence length. For example, ByT5 (Xue et al., 2022), a byte-level counterpart of mT5 (Xue et al., 2021), is competitive with mT5 on a number of tasks, but it has a much slower pre-training and inference runtime, making it impractical for real use-cases. Most other attempts to create character-level or byte-level models perform explicit downsampling or pooling to reduce the sequence length (Clark et al., 2022; Tay et al., 2022). However, relevant units of meaning usually span a variable number of bytes/characters. These methods also introduce significant alterations to the standard Transformer architecture.

In this project, we propose **DelT5**, a variant of the ByT5 architecture that has the potential to address its inefficiencies while allowing more flexibility than fixed-span downsampling methods. DelT5 dynamically deletes tokens in its encoder in order to reduce the sequence length using a gating mechanism at a fixed, early encoder layer. By allowing the first few encoder layers to process the entire sequence, the encoder creates contextualized representations of the tokens. Then, when a subset of the tokens are deleted by the gating mechanism, those that remain already contain contextual information about those that were removed, allowing an implicit merging of information into a shorter sequence. During training, we use an auxiliary loss with a tunable weight that can adjust the amount of deletion DelT5 performs.

We fine-tune the DelT5 gating mechanism on top of a pre-trained ByT5 small, using English data for training. Our results indicate that DelT5 outperforms both random and fixed deletion baselines in terms of span corruption loss while removing an equivalent percentage of tokens. We also conduct zero-shot tests across 15 diverse languages, showing that DelT5 can generalize to new languages and scripts. Through individual sample analysis, we show that DelT5 selectively deletes more tokens in cases where it does not lead to a significant loss increase. Our approach overcomes the limitations of ByT5, presenting a significant step toward the adoption of byte-level models and the elimination of subword tokenization from modern language models.

3 Related Work

There have been several attempts to design character-level or byte-level models in an effort to overcome the pitfalls of subword tokenization. Several architectures have employed explicit downsampling steps to reduce their input sequence lengths. For example, CANINE (Clark et al., 2022) is a character-level model trained on the same languages as mBERT (Devlin et al., 2019), and it uses convolutional downsampling to reduce the sequence before feeding it to a 12-layer Transformer

encoder stack. Charformer (Tay et al., 2022) is another byte-level encoder-decoder model that learns a gradient-based "soft tokenization" for more efficient training and inference.

This paper focuses on ByT5 (Xue et al., 2022), a byte-level sequence-to-sequence Transformer architecture that serves as a counterpart to mT5 (Xue et al., 2021), the multilingual version of T5 (Raffel et al., 2023). ByT5 requires significantly fewer parameters for its vocabulary matrix (which is comprised of only 256 embeddings), but to compensate for the loss of these parameters, ByT5 has a "heavy" encoder with a larger number of layers than the decoder. While ByT5 shows impressive performance on a variety of downstream tasks, its heavy encoder and short input sequence length (1024 bytes) make it quite inefficient. In terms of FLOPs, ByT5 requires about 1.2 times more operations than mT5, resulting in a 33% increase in pre-training wall time. With regard to inference time on downstream tasks, ByT5 is up to 10 times slower than mT5, depending on the length of the input sequence.

More recently, MegaByte (Yu et al., 2023) has shown promise in scaling byte-level decoders to long context problems, but it also includes a step that segments sequences into fixed-length "patches." SpaceByte (Slagle, 2024) employs a similar solution, but adds larger, global Transformer blocks to certain types of bytes, such as space characters, to improve performance. In a similar vein to these papers, other work has attempted to address issues with long sequences in Transformer models more generally. For example, Hierarchical Transformers (Nawrot et al., 2022) add several layers of downsampling and upsampling to handle long sequences in decoder models. Follow-up work has implemented dynamic pooling using boundary predictors, but these usually involve a supervised training step (Nawrot et al., 2023). Other solutions include Nugget (Qin and Van Durme, 2023; Qin et al., 2023), which encodes representations for dynamic subsets of the input text.

Unlike previous work, the deletion gating mechanism that we add to DelT5 does not require an overhaul of the existing Transformer architecture, so it can be added to a pre-trained model with fine-tuning using a small number of additional parameters. The deletion gating can also be applied to models trained from scratch, which allows for faster pre-training runtimes. While we are particularly interested in byte-level modeling, our approach can also be applied to subword models, complementing the existing line of work on long-context modeling.

4 Approach

4.1 DelT5 Model Architecture

DelT5 is a variant of the ByT5 architecture that has the potential to address its inefficiencies while allowing more flexibility than previous character/byte-level models. After a certain number of layers, DelT5's encoder dynamically deletes tokens that it no longer needs to attend to (in the context of byte-level models, "tokens" refer to bytes). More concretely, after a fixed encoder layer, we add a gating mechanism that determines which tokens in the sequence should be kept to be processed by later layers, and which tokens should be deleted from the sequence.[2]

Deletion Gating Mechanism. The DelT5 gating mechanism is inspired by existing architectures with gating mechanisms such as Long-Short Term Memory (LSTMs, Hochreiter and Schmidhuber, 1997), Gated Recurrent Units (GRUs, Cho et al., 2014), and Mixture-of-Experts (MoEs, Shazeer et al., 2017). DelT5's gating mechanism is placed after the output of a fixed encoder layer l and is defined by the following function:

$$\mathbf{g}_i = k\sigma(-\mathbf{W}\mathbf{h}_{li} + \mathbf{b})$$

where $\mathbf{h}_{li} \in \mathbb{R}^d$ is the hidden state vector output by layer l for token i in the sequence; $\mathbf{W} \in \mathbb{R}^{1\times d}$; $\mathbf{b} \in \mathbb{R}$; k is a large negative constant; and d is ByT5's hidden state dimensionality. The gating activation function is a rescaled and translated version of the sigmoid function; instead of being bounded between 0 and 1, it is bounded between k and 0. The gating values for the entire sequence can be expressed as $\mathbf{G} = [\mathbf{g}_1; \mathbf{g}_2; \ldots; \mathbf{g}_N] \in \mathbb{R}^{1\times N}$. In the experiments we show in the next section, we use $k = -50$ and $l = 3$.

The new parameters for the deletion gate are randomly initialized, other than the bias term, which is initialized to -1. All other parameters are loaded from a pre-trained ByT5 small from HuggingFace.

[2]For details on exact code contributions for this project, see Appendix A.

Hard and Soft Deletion. There are two types of deletion involved in the DelT5 encoder's architecture, shown in Figure 1. The first is *soft deletion*, where the outputs of the gating mechanism are added directly to the standard key-query-value self-attention mechanism of the subsequent layer:

$$\mathbf{H}_{l+1} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}} + \mathbf{G}\right)\mathbf{V}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{H}_{l+1} \in \mathbb{R}^{N \times d}$.

Soft deletion does not truly reduce the sequence length, but it may be used during training to emulate deletion (see Figure 1b). In order to see efficiency gains, we must apply *hard deletion*, where the hidden states are removed from the sequence using a hard threshold; we set this threshold to be $\frac{k}{2}$, half of the range of the delete gate's output. If hard deletion is applied during training, in order to allow gradient flow through the gating mechanism, soft deletion is first applied at layer l, and the sequence length reduction is applied after layer $l + 1$, as shown in Figure 1c.

For different samples in a given batch, different amounts of tokens can be deleted; when applying hard deletion, the new sequence length is determined by the example in the batch with the largest number of remaining tokens, and the other examples are padded to the new sequence length. In addition to resizing the hidden states, since T5 architectures use relative positional embeddings at each layer, the hard deletion is also applied to the positional embeddings. In the main results we present, hard deletion is applied during both training and evaluation. For results on models trained with soft deletion, see Appendix B.

Auxiliary Loss. DelT5 allows deletion rates to be adjusted using a tunable auxiliary loss:

$$\text{loss}_{\mathbf{g}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{g}_i$$

This loss is the average of the delete gate output values, which encourages them to be more negative (i.e. closer to k, the minimum gate value). The total loss is defined as the sum loss $= \text{loss}_{\text{CE}} + \alpha\text{loss}_{\mathbf{g}}$, where loss_{CE} is the cross entropy loss of the standard ByT5 pre-training objective, described in the next section. Varying the hyperparameter α allows the DelT5 model to delete more or fewer tokens. In our experiments, we train models with α values of 0, $5e{-}5$, $1e{-}4$, $2e{-}4$, $4e{-}4$, and $1e{-}3$.

4.2 Baselines

For comparison with DelT5, we use the following baselines:

1. **ByT5 baseline**: An unaltered ByT5 small architecture.
2. **Decoder-only baseline**: We implement and train a decoder-only version of ByT5 by ignoring the input to the encoder and instead providing a single pad token as input.
3. **Random deletion baseline**: We implement and train a set of models with a random deletion gating mechanism, where the choice of how the tokens are deleted is random; some percentage of gating values are set to k, and the rest are set to 0. In our experiments, we try different deletion probabilities: 20%, 50%, 80%, and 90%.
4. **Fixed deletion baseline**: We implement and train a set of models that delete specific tokens, mainly whitespace, punctuation, and the ends of words. We train several models that delete different percentages of the ends of words: 20%, 50%, and 90%.

For the random and fixed deletion baselines, the deletion gate is placed at layer $l = 3$, like the DelT5 models. The ByT5 baseline serves as a lower bound on the best possible span corruption loss, since it does not delete any tokens; the decoder-only baseline is an upper bound on the span corruption loss, since it uses no information from the encoder.

5 Experiments

5.1 Data

We use a subset of the multilingual C4 (mC4) corpus (Xue et al., 2021) for model training and evaluation. For training, we download 1.3B bytes of the English mC4, so that all models and

baselines are trained on the same data. For evaluations, we also compute per-language metrics on a set of 15 typologically diverse languages: English, French, Spanish, German, Greek, Bulgarian, Russian, Turkish, Arabic, Vietnamese, Thai, Chinese, Hindi, Swahili and Urdu. These evaluations are zero-shot for all languages other than English. We precompute and load 16M bytes from each language's validation dataset in mC4.[3] The number of bytes we download for the training and validation samples is calculated to fit the batch size and number of steps described in the experiment configuration in Section 5.3.

5.2 Training and Evaluation

Training Task. The main task we use to train the DelT5 model is the span corruption pre-training task, as it is defined in the ByT5 paper. In this pre-training objective, spans of tokens in unlabeled text data are replaced with a single *sentinel* token ID, and the model must fill in the missing spans. For ByT5, these are spans of bytes, and the masks can potentially interfere with word boundaries. Below is an example, where <X>, <Y>, and <Z> represent sentinel tokens:

- Text: `CS224n is the best class taught at Stanford.`
- Masked input: `CS224n is t<X>ght a<Y>ord.`
- Target: `<X>he best class tau<Y>t Stanf<Z>`

Evaluation Methods. The main evaluation metrics we use are the span corruption cross entropy loss defined above, the percentage of deleted tokens from the sequence, and the inference runtime. All of these are measured on the span corruption task and averaged over the validation set. The goal is to achieve the lowest possible loss on the span corruption objective while maintaining a substantial amount of deletion in the DelT5 encoder using the gating mechanism.

5.3 Experimental Details

We train each DelT5 and baseline model on the span-corruption task for 1,000 steps over batches of 2^{20} tokens (i.e. an encoder sequence length of 1024 with an effective batch size of 1024). For the span corruption loss, we calculate the corrupted spans such that the average mask span length is 20 tokens with a noise density of 15% (i.e. 15% of tokens in the sequence are masked out), following the specification in the ByT5 paper. We use a per-device training batch size of 16 with 64 gradient accumulation steps to emulate the effective batch size of 1024 examples. We do not repeat epochs over the sample of mC4 data. We use the AdamW optimizer with an initial learning rate of 1e-3 with linear decay and no warmup. All other hyperparameters are the same as the default training arguments of the HuggingFace Trainer class.[4] Each per-language evaluation is 1,000 steps with a batch size of 2^{14} tokens (i.e. an encoder sequence length of 1024 with a batch size of 16). We use the last model checkpoint at step 1,000 for all evaluations.

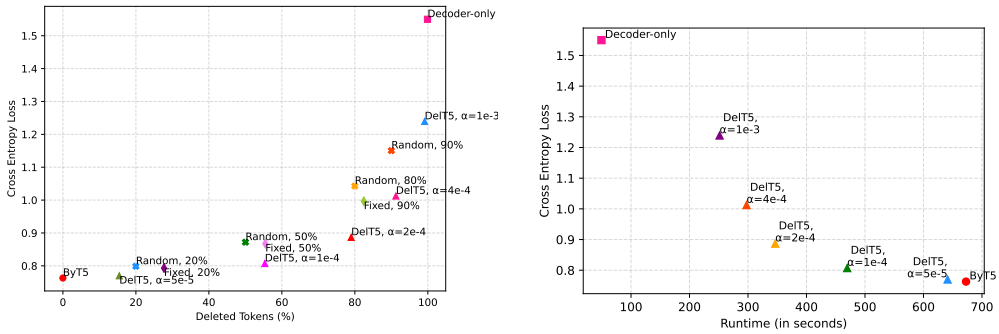
On average, training takes about 7 hours for each model (training time varies depending on the amount of deletion). We train 14 models on NVIDIA RTX A6000 (48 GB) GPUs, for a total of about 100 GPU hours.

5.4 Results

Loss vs. Deletion Rate. Figure 2a shows the span corruption cross entropy loss over the percentage of deleted tokens for each DelT5 and baseline model. As expected, ByT5 has the best loss, but it does not delete any tokens; the decoder-only baseline deletes all tokens, but has the worst loss. The DelT5 models as well as the fixed and random deletion baselines fall in-between, but notably, the DelT5 models have better losses than the other baseline models with the same deletion rate. For instance, DelT5 with $\alpha = 5e{-}5$ deletes about 55% of tokens, which is about the same deletion rate as the 50% fixed and 50% random deletion baselines, but the DelT5 model has the lowest loss. We see similar trends for the other DelT5 models and baselines. These results indicate that the DelT5 gating mechanism effectively deletes tokens in a way that minimizes the loss, compared to the random and fixed deletion baselines.

[3] Certain languages (namely, Swahili and Urdu) lack enough validation data in the mC4 dataset. For these languages, we instead sample from the their training sets.

[4] Trainer class: `https://huggingface.co/docs/transformers/en/main_classes/trainer`



(a) Span corruption cross entropy loss vs. percentage of deleted tokens for each DelT5 and baseline model.

(b) Span corruption cross entropy loss vs. inference runtime for each DelT5 model as well as ByT5 and the decoder-only baseline.

Figure 2: Evaluation results on a sample of the English mC4 validation dataset.

Table 1: Average span corruption loss on the validation set across languages and models.

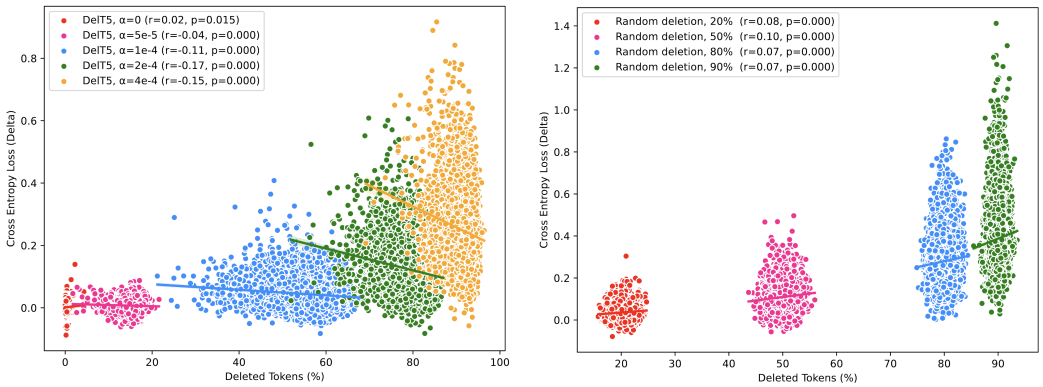
Language	ByT5	Decoder-only	Random				Fixed			DelT5					
			20%	50%	80%	90%	20%	50%	90%	$\alpha = 0$	$\alpha = 5e{-5}$	$\alpha = 1e{-4}$	$\alpha = 2e{-4}$	$\alpha = 4e{-4}$	$\alpha = 1e{-3}$
English	0.7630	1.5500	0.7986	0.8718	1.0428	1.1506	0.7932	0.8675	0.9964	0.7646	0.7698	0.8070	0.8867	1.0124	1.2392
French	0.8802	2.3792	0.9706	1.1082	1.3657	1.5142	0.9842	1.1803	1.4048	0.8905	0.9029	0.9775	1.1023	1.2643	1.7206
Spanish	0.9076	2.4307	0.9930	1.1414	1.3685	1.4973	1.0130	1.2021	1.4137	0.9177	0.9400	1.0130	1.1429	1.2982	1.6738
German	0.9181	2.6071	1.0068	1.1758	1.4396	1.5940	1.0389	1.2507	1.4949	0.9310	0.9445	1.0348	1.1566	1.3247	1.8211
Greek	0.6278	1.7775	0.7447	0.9745	1.2463	1.3797	0.8150	1.1055	1.3676	0.6449	0.6739	0.7547	0.8730	1.0251	1.4235
Bulgarian	0.6155	1.7313	0.7259	0.9282	1.2058	1.3411	0.7864	1.0629	1.4126	0.6357	0.6424	0.7135	0.8463	1.0790	1.4380
Russian	0.5620	1.6309	0.6737	0.8857	1.1590	1.2762	0.7500	1.0196	1.2821	0.5730	0.5855	0.6526	0.7802	0.9226	1.2819
Turkish	1.0129	2.8941	1.1192	1.3017	1.6101	1.8153	1.1151	1.3923	1.8030	1.0209	1.0876	1.1299	1.3052	1.4530	2.2062
Arabic	0.7506	1.7771	0.8662	1.0570	1.3017	1.4130	0.9227	1.2019	1.4201	0.7694	0.7743	0.8434	0.9717	1.1137	1.4420
Vietnamese	0.9191	2.8380	1.0404	1.2400	1.4986	1.6529	1.0993	1.3362	1.6196	0.9418	0.9637	1.0741	1.1875	1.3477	1.8817
Thai	0.5326	1.6621	0.6656	0.8610	1.1246	1.2545	0.7361	0.9468	1.2265	0.5618	0.5746	0.6353	0.7700	0.8868	1.3451
Chinese	1.0920	2.6098	1.2419	1.5089	1.8771	2.0763	1.3394	1.6206	2.0951	1.1137	1.1261	1.1903	1.3535	1.5411	2.3053
Hindi	0.6564	1.7341	0.7860	0.9865	1.2408	1.3663	0.8430	1.0810	1.3758	0.6705	0.6832	0.7508	0.8868	0.9990	1.4451
Swahili	1.1835	2.7789	1.2948	1.4968	1.7920	1.9820	1.3099	1.5479	1.9259	1.2065	1.2288	1.3303	1.4815	1.7220	2.3490
Urdu	0.7952	2.0185	0.9214	1.1227	1.3816	1.4846	0.9579	1.2458	1.5334	0.8100	0.8271	0.9193	1.0565	1.2131	1.5058

Table 2: Average percentage of deleted tokens on the validation set across languages and models.

Language	ByT5	Decoder-only	Random				Fixed			DelT5					
			20%	50%	80%	90%	20%	50%	90%	$\alpha = 0$	$\alpha = 5e{-5}$	$\alpha = 1e{-4}$	$\alpha = 2e{-4}$	$\alpha = 4e{-4}$	$\alpha = 1e{-3}$
English	0.0000	0.9990	0.2000	0.5002	0.7998	0.9000	0.2770	0.5546	0.8245	0.0020	0.1545	0.5533	0.7899	0.9125	0.9912
French	0.0000	0.9990	0.2002	0.5001	0.7999	0.9000	0.2845	0.5593	0.8297	0.0042	0.1401	0.4679	0.7126	0.8382	0.9890
Spanish	0.0000	0.9990	0.1999	0.5003	0.8000	0.9000	0.2821	0.5546	0.8291	0.0055	0.1542	0.4933	0.7456	0.8717	0.9903
German	0.0000	0.9990	0.1999	0.5000	0.8000	0.9000	0.2736	0.5475	0.8406	0.0036	0.1252	0.4780	0.7002	0.8276	0.9893
Greek	0.0000	0.9990	0.2001	0.4999	0.8000	0.9000	0.2482	0.5464	0.8701	0.0201	0.1030	0.4473	0.7629	0.8548	0.9842
Bulgarian	0.0000	0.9990	0.2000	0.4999	0.7999	0.9001	0.2463	0.5507	0.8645	0.0137	0.0935	0.4697	0.7651	0.8618	0.9796
Russian	0.0000	0.9990	0.2000	0.5002	0.8001	0.9001	0.2466	0.5450	0.8703	0.0061	0.0744	0.4364	0.7718	0.8584	0.9808
Turkish	0.0000	0.9990	0.2000	0.5001	0.8001	0.9001	0.2686	0.5439	0.8546	0.0129	0.1213	0.4150	0.6973	0.7819	0.9865
Arabic	0.0000	0.9990	0.2000	0.5001	0.8001	0.9000	0.2486	0.5506	0.8694	0.0031	0.0850	0.3345	0.6996	0.8161	0.9847
Vietnamese	0.0000	0.9990	0.2000	0.5002	0.7999	0.9001	0.2836	0.5491	0.8350	0.0060	0.1470	0.4520	0.6909	0.7973	0.9738
Thai	0.0000	0.9990	0.2000	0.5002	0.8000	0.9001	0.2211	0.5138	0.8847	0.0121	0.0497	0.4204	0.7598	0.8168	0.9809
Chinese	0.0000	0.9990	0.2000	0.5000	0.7998	0.8999	0.2170	0.5120	0.8891	0.0068	0.0288	0.2342	0.5636	0.7022	0.9855
Hindi	0.0000	0.9990	0.2000	0.5001	0.8001	0.9000	0.2365	0.5254	0.8741	0.0064	0.0717	0.3866	0.7041	0.8083	0.9797
Swahili	0.0000	0.9990	0.1999	0.5001	0.8000	0.9000	0.2780	0.5582	0.8372	0.0102	0.1563	0.4185	0.6627	0.8127	0.9876
Urdu	0.0000	0.9990	0.2000	0.5001	0.8000	0.8998	0.2449	0.5574	0.8640	0.0054	0.1215	0.3964	0.7223	0.8438	0.9630

We also evaluate loss and deletion rates across 15 typologically diverse languages, as detailed in Table 1 and Table 2. (To view these results as graphs, see Figure 8 in Appendix C.) Generally, the trend observed in English, where higher values of α lead to increased deletion, holds across these languages. However, the DelT5 models consistently demonstrate lower losses compared to both random and fixed baselines at comparable deletion rates. Notably, some languages exhibit lower deletion rates than others. For example, Chinese consistently shows the lowest deletion rates at most values of α. This might be due to the language's use of single characters to represent words in its script.

Loss vs. Inference Runtime. The models that delete more tokens also have a faster runtime, as shown by the results in Figure 2b. In particular, DelT5 with $\alpha = 1e{-4}$ has about a 30% decrease in inference runtime when compared to ByT5, with only a small increase in cross entropy loss. These results show that we can tune DelT5 to be more efficient with minimal effects on performance. These results also hold on our zero-shot evaluations across languages, showing that DelT5 can improve



(a) DelT5 models.

(b) Random deletion models.

Figure 3: Delta of the span corruption loss vs. the percentage of deleted tokens for each (a) DelT5 model and (b) random deletion baseline model. Delta is calculated by subtracting ByT5's loss from the model's loss for a particular sample. Each point represents a single sample.

efficiency in multilingual settings as well. See Appendix C for the loss, deletion, and runtime results across languages.

6 Analysis

Per-sample Analysis In this section, we present a per-sample analysis of the cross entropy loss and deletion rates for each DelT5 model and random deletion baseline. Figure 3a plots the increase in cross entropy loss for individual samples for each DelT5 model, using the ByT5 loss as the baseline (i.e. the delta between the DelT5 model's loss and ByT5's loss for individual samples). For each DelT5 model, we see a weak negative correlation between the delta of the loss and the percentage of tokens deleted. This reflects what we would expect from the DelT5 models; for an individual sample, DelT5 learns when it can delete more tokens without incurring a large increase in the loss.

Figure 3b shows the same analysis for the random deletion models. We can see that there is a weak positive correlation between the delta of the loss and the percentage of tokens deleted; when the random deletion model removes more tokens, it is more likely to cause an increase in loss. These results further support the observation that the DelT5 models more strategically remove tokens compared to the baselines. Appendix D includes additional per-sample analyses with loss and deletion histograms for each model.

Qualitative Evaluation of Deletion Patterns During training, we notice that the DelT5 models for most values of α gradually learn to delete more tokens over time, as shown in Figure 4. By loading a DelT5 model at different checkpoints and analyzing the attention maps after the gating mechanism, we can see the types of bytes the model keeps and deletes. We show the heatmaps from different model checkpoints for two example sentences in Figure 5, using the DelT5 model with $\alpha = 1e{-}4$. This DelT5 model appears to delete spaces in between words initially, at around 100 and 200 training steps. It then deletes subparts of words at around 500 training steps. After 1,000 steps, it deletes entire words or clauses altogether. In models with greater than 60% deletion, most of the bytes that are deleted are near the end of the sequence. These deletion trends, especially in early training steps, where word boundaries and subparts of words are deleted, have interesting parallels in cognitive science and psycholinguistics; humans are able to process text with only partial word information remarkably well (Rayner et al., 2006; Grainger and Whitney, 2004; Johnson and Eisler, 2012).

7 Conclusion

In this work, we introduce DelT5, a modified version of ByT5 that includes a novel token deletion mechanism in its encoder to reduce the input sequence length, allowing for significant inference

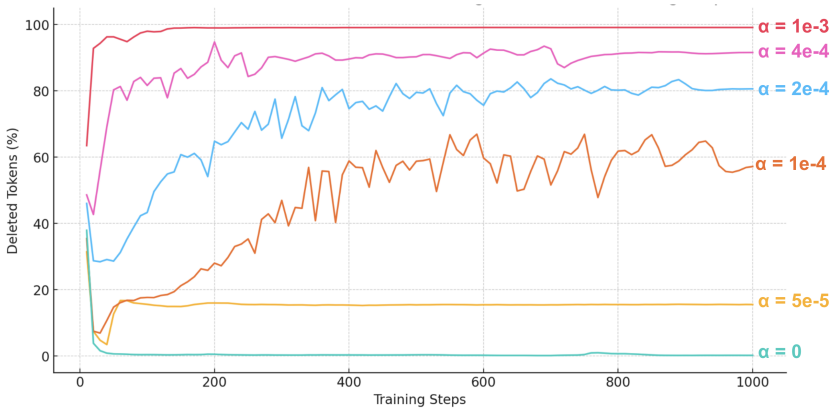


Figure 4: Percentage of deleted tokens over training steps for several DelT5 models that use different α hyperparameter values for the auxiliary loss. Larger α values result in more deletion.

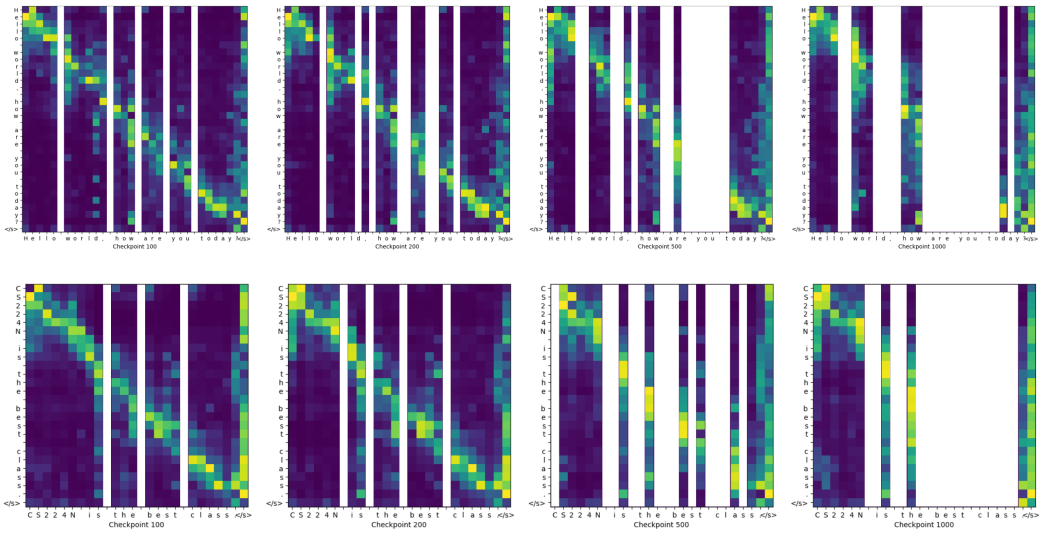


Figure 5: Visualization of DelT5 ($\alpha = 1e{-}4$) attention maps over different training steps. White bands represent bytes that were deleted.

speedups while maintaining model performance. We show that DelT5 achieves a lower span corruption loss compared to random and fixed deletion baselines while deleting the same percentage of tokens. We train DelT5 models on English data, but also perform zero-shot evaluations on 15 typologically diverse languages, showing that DelT5 can generalize to new languages and scripts. We perform per-sample analyses, finding that DelT5 meaningfully deletes more tokens for individual samples when it will not incur a large increase in loss. We finally visualize attention maps showing deletion patterns for a few examples, demonstrating avenues for interpretability work involving the DelT5 architecture. Overall, our contribution addresses the practical limitations of traditional byte-level models, and we see this as a step toward the removal of subword tokenization from modern language models.

Future Work. In this paper, we trained the deletion gating mechanism by fine-tuning a pre-trained ByT5 model. In the future, we plan to pre-train DelT5 models from scratch; we anticipate that this will close the performance gap between DelT5 and ByT5 while allowing for more token deletion and inference gains. We also plan to further fine-tune and evaluate our DelT5 models on downstream long-context modeling tasks, to show that DelT5 can feasibly handle much larger input sequence lengths. We also plan to try alternative gating mechanisms that include explicit merging or pooling of variable-length spans of tokens.

8 Ethics Statement

The current project only trains the DelT5 model on English data. While our evaluations on 15 languages show promise that the model can generalize to multilingual settings, performance on different sets of languages might differ from our evaluation results. This can be concerning if our models are applied to languages that we have not evaluated. In future work, we plan to include multilingual data during training and evaluate on a larger set of languages, but in general, we encourage users to run their own thorough tests before applying our models to tasks that involve new languages and scripts. We do not intend to make claims about performance on all languages based on our results.

Another potential ethical concern is the usage of GPUs. Training large language models can have negative environmental impacts due to energy consumption. While this work involves training a large language model, our hope is to contribute a much more efficient and performant variant of ByT5. We also perform small experiments using fine-tuning rather than training models from scratch, which would be a next step for this project. By starting with small experiments, we minimize the environmental impact of our research.

References

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David Mortensen, Noah Smith, and Yulia Tsvetkov. 2023. Do all languages cost the same? tokenization in the era of commercial language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9904–9923, Singapore. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. 2022. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Jonathan Grainger and Carol Whitney. 2004. Does the huamn mnid raed wrods as a wlohe? *Trends in Cognitive Sciences*, 8(2):58–59.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Jing Huang, Zhengxuan Wu, Kyle Mahowald, and Christopher Potts. 2023. Inducing character-level structure in subword-based language models with type-level interchange intervention training. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 12163–12180, Toronto, Canada. Association for Computational Linguistics.

Rebecca L. Johnson and Morgan E. Eisler. 2012. The importance of the first and last letter in words during sentence reading. *Acta Psychologica*, 141(3):336–351.

Ayush Kaushal and Kyle Mahowald. 2022. What do tokens know about their characters and how do they know it? In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2487–2507, Seattle, United States. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference*

on Empirical Methods in Natural Language Processing: System Demonstrations, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. 2023. Efficient transformers with dynamic token pooling. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6403–6417, Toronto, Canada. Association for Computational Linguistics.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Lukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. 2022. Hierarchical transformers are more efficient language models. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1559–1571, Seattle, United States. Association for Computational Linguistics.

Aleksandar Petrov, Emanuele La Malfa, Philip H. S. Torr, and Adel Bibi. 2023. Language model tokenizers introduce unfairness between languages.

Guanghui Qin, Corby Rosset, Ethan C. Chau, Nikhil Rao, and Benjamin Van Durme. 2023. Nugget 2d: Dynamic contextual compression for scaling decoder-only language models.

Guanghui Qin and Benjamin Van Durme. 2023. Nugget: Neural agglomerative embeddings of text. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 28337–28350. PMLR.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer.

Keith Rayner, Sarah J. White, Rebecca L. Johnson, and Simon P. Liversedge. 2006. Raeding wrods with jubmled lettres: There is a cost. *Psychological Science*, 17(3):192–193. PMID: 16507057.

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

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*.

Aaditya K. Singh and DJ Strouse. 2024. Tokenization counts: the impact of tokenization on arithmetic in frontier llms.

Kevin Slagle. 2024. Spacebyte: Towards deleting tokenization from large language modeling.

Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. 2022. Charformer: Fast character transformers via gradient-based subword tokenization. In *International Conference on Learning Representations*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Linting Xue, 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.

Linting Xue, 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, Online. Association for Computational Linguistics.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. 2023. Megabyte: Predicting million-byte sequences with multiscale transformers.

Table 3: Inference runtime on the validation set, measured as a percentage of ByT5's inference time, across languages and models.

Language	ByT5	Decoder-only	DelT5					
			$\alpha = 0$	$\alpha = 5e{-}5$	$\alpha = 1e{-}4$	$\alpha = 2e{-}4$	$\alpha = 4e{-}4$	$\alpha = 1e{-}3$
English	1.0000	0.0736	1.0488	0.9532	0.6981	0.5152	0.4423	0.3738
French	1.0000	0.0732	1.0364	0.9753	0.7774	0.5731	0.4994	0.3785
Spanish	1.0000	0.0734	1.0352	0.9626	0.7572	0.5510	0.4808	0.3765
German	1.0000	0.0735	1.0345	0.9831	0.7707	0.5795	0.5035	0.3776
Greek	1.0000	0.0738	1.0384	0.9964	0.7811	0.5401	0.4885	0.3834
Bulgarian	1.0000	0.0738	1.0404	1.0016	0.7616	0.5356	0.4831	0.3879
Russian	1.0000	0.0738	1.0352	1.0120	0.7841	0.5341	0.4842	0.3867
Turkish	1.0000	0.0740	1.0420	0.9851	0.8089	0.5811	0.5348	0.3814
Arabic	1.0000	0.0743	1.0408	1.0066	0.8452	0.5810	0.5129	0.3841
Vietnamese	1.0000	0.0742	1.0399	0.9644	0.7873	0.5929	0.5328	0.3929
Thai	1.0000	0.0741	1.0416	1.0341	0.7895	0.5375	0.5086	0.3858
Chinese	1.0000	0.0740	1.0377	1.0476	0.9438	0.6905	0.6095	0.3824
Hindi	1.0000	0.0740	1.0402	1.0225	0.8209	0.5829	0.5183	0.3868
Swahili	1.0000	0.0739	1.0418	0.9579	0.7899	0.5957	0.5195	0.3801
Urdu	1.0000	0.0738	1.0341	0.9775	0.7922	0.5555	0.4871	0.3955

A Code Contributions

To build the DelT5 architecture, we implement PyTorch modules that extend the T5 modules provided by HuggingFace, altering the necessary components while allowing unaltered T5 functionality to remain via inheritance.[5] We also adapt HuggingFace's span corruption training script, which was only compatible with JAX and the original T5 objective. We made the script compatible with Pytorch and adapted how the sentinel tokens are assigned, since ByT5 reuses the last 100 bytes in its vocabulary as sentinels.[6] All code contributions for this project were written by Julie Kallini.

B Soft Deletion Models

In the main paper, we present results for DelT5 models in which hard deletion is applied during both training and inference. We also train as set of models where soft deletion is applied during training, and hard deletion is applied at inference. We call this set of models the MaskedDelT5 models. Figure 6 shows the span corruption loss vs. inference runtime results for these models.

C Additional Per-language Evaluations

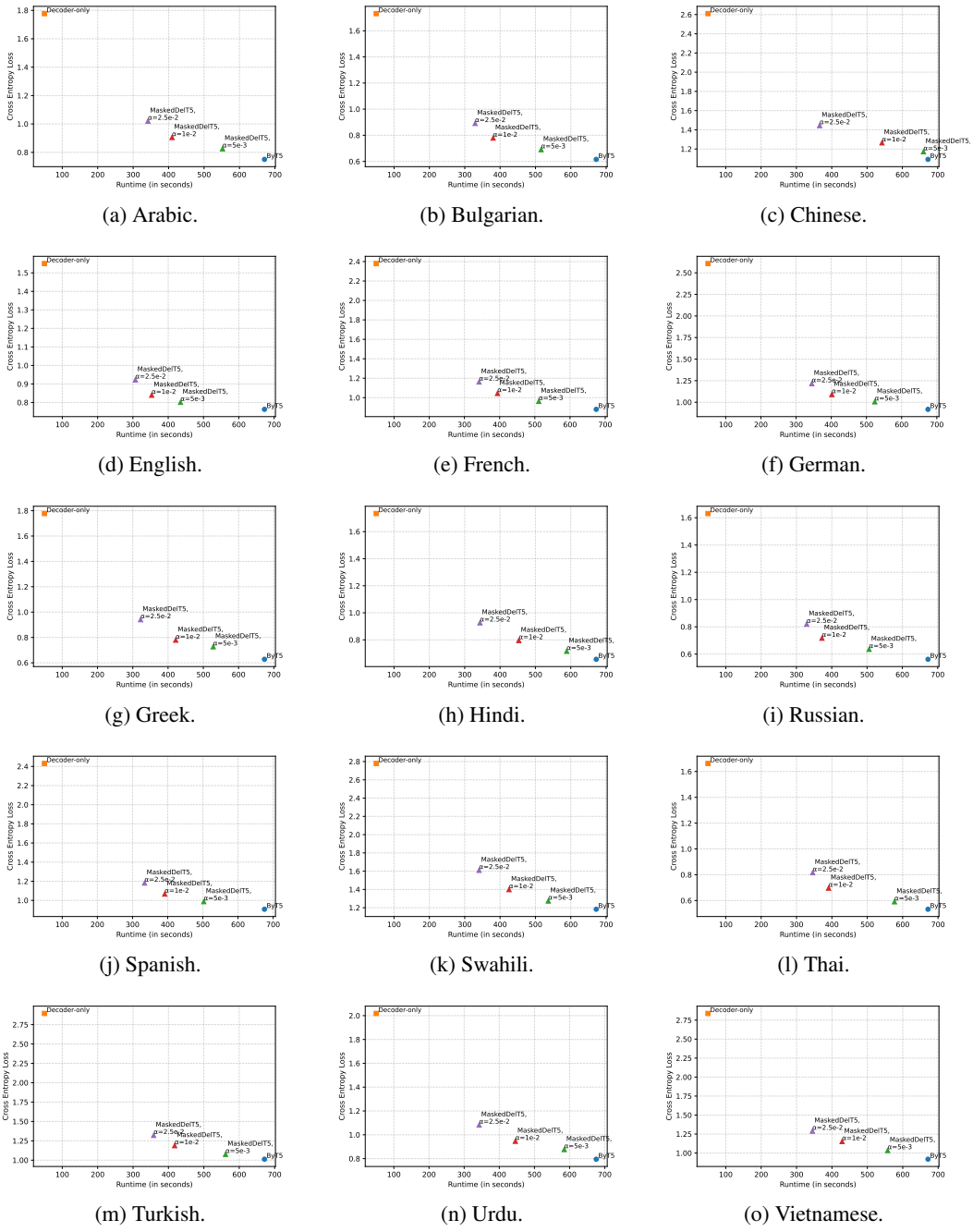
Figure 7 shows the span corruption cross entropy loss vs. the percentage of deleted tokens for each DelT5 and baseline model, for each of the 15 languages. Figure 8 shows similar plots for the loss vs. inference runtime across languages.

D Additional Per-sample Analysis Results

Here, we provide histograms of the cross entropy loss deltas and deletion percentages for the DelT5 models and random deletion baseline models. Figure 9 displays the results for the DelT5 models, and Figure 10 displays the results for the random baseline models.

[5]For HuggingFace's T5 implementation, see `https://github.com/huggingface/transformers/blob/v4.39.0/src/transformers/models/t5/modeling_t5.py`.

[6]We adapt the following FlaxT5 language modeling script: `https://github.com/huggingface/transformers/blob/main/src/transformers/models/t5/modeling_flax_t5.py`.



(a) Arabic. (b) Bulgarian. (c) Chinese.

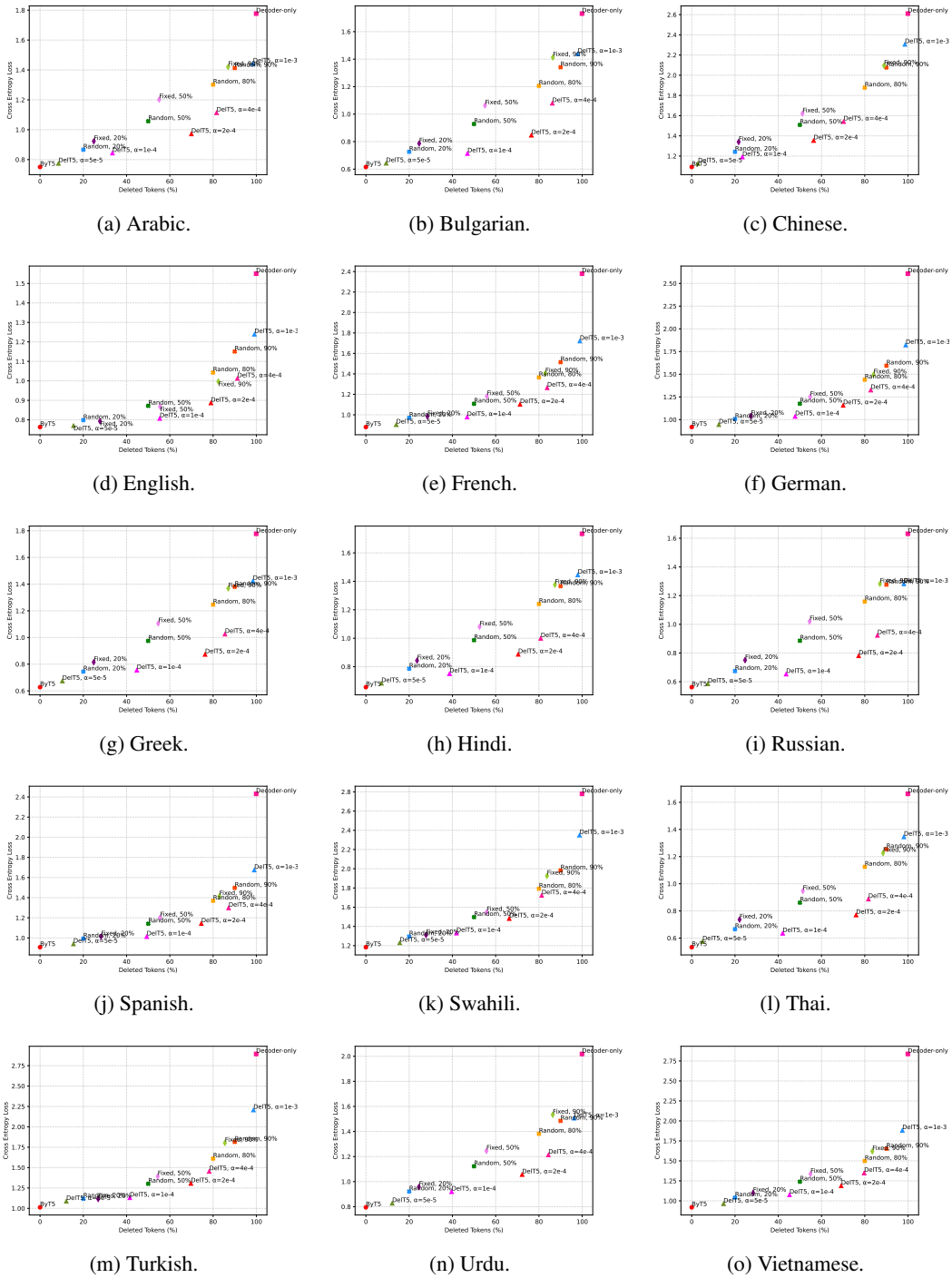
(d) English. (e) French. (f) German.

(g) Greek. (h) Hindi. (i) Russian.

(j) Spanish. (k) Swahili. (l) Thai.

(m) Turkish. (n) Urdu. (o) Vietnamese.

Figure 6: Per-language evaluations of span corruption cross entropy loss vs. inference runtime for each MaskedDelT5 and baseline model. MaskedDelT5 models are trained with soft deletion.



(a) Arabic. (b) Bulgarian. (c) Chinese.

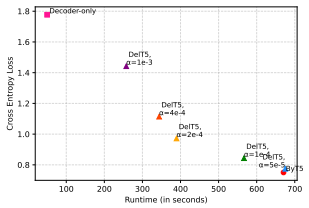
(d) English. (e) French. (f) German.

(g) Greek. (h) Hindi. (i) Russian.

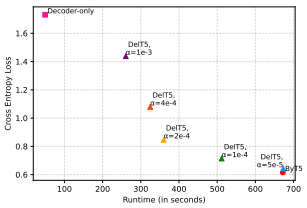
(j) Spanish. (k) Swahili. (l) Thai.

(m) Turkish. (n) Urdu. (o) Vietnamese.

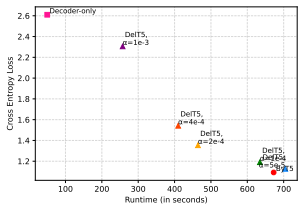
Figure 7: Per-language evaluations of span corruption cross entropy loss vs. the percentage of deleted tokens for each DelT5 and baseline model.



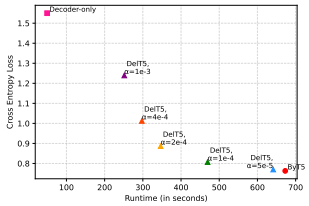
(a) Arabic.



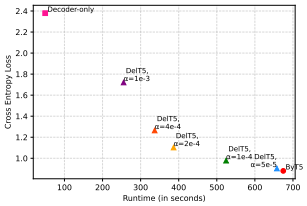
(b) Bulgarian.



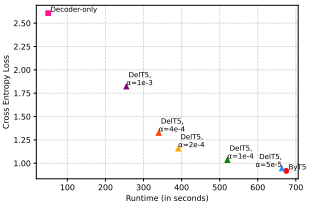
(c) Chinese.



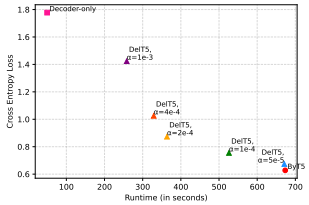
(d) English.



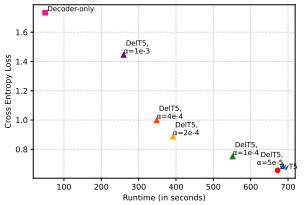
(e) French.



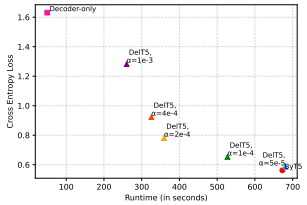
(f) German.



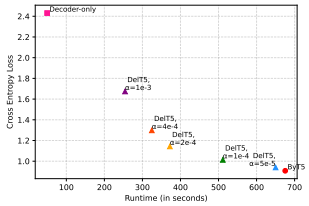
(g) Greek.



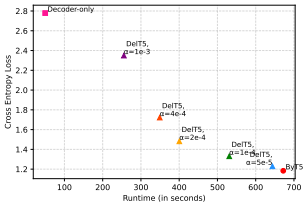
(h) Hindi.



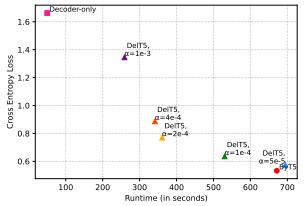
(i) Russian.



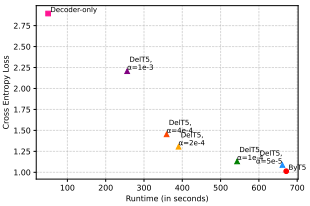
(j) Spanish.



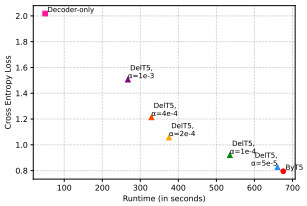
(k) Swahili.



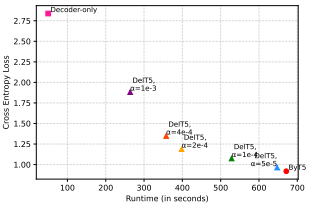
(l) Thai.



(m) Turkish.

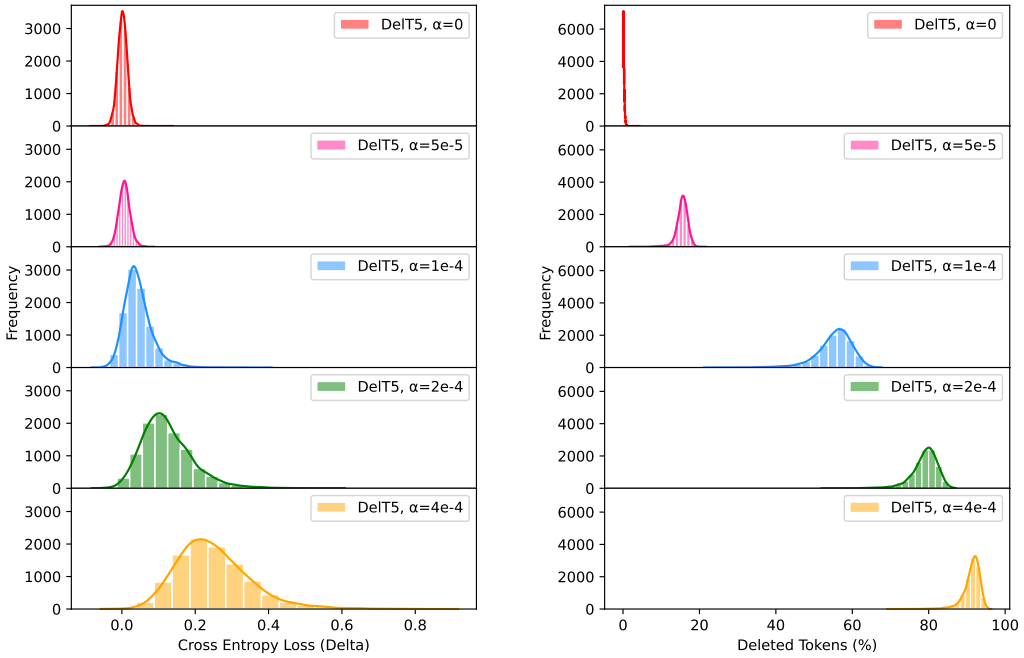


(n) Urdu.



(o) Vietnamese.

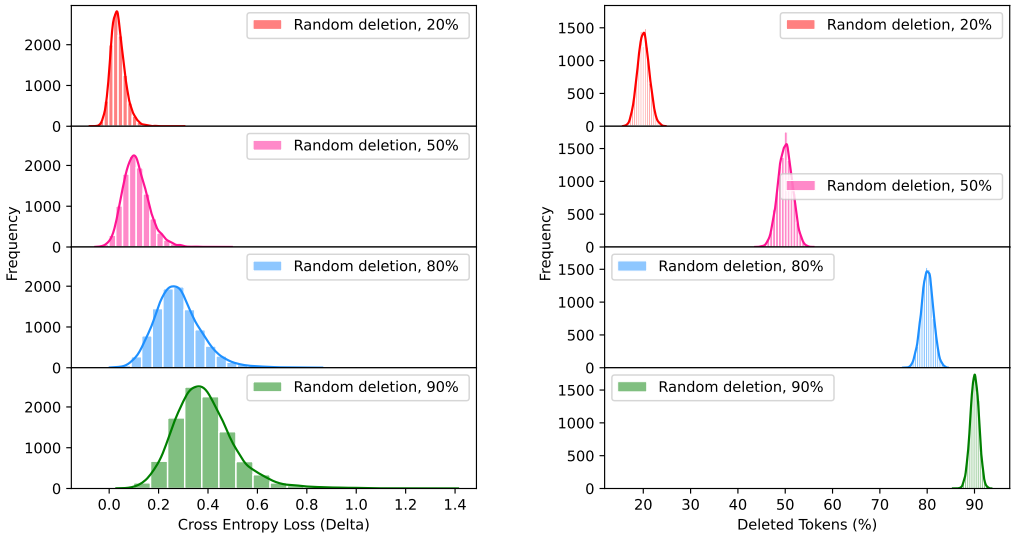
Figure 8: Per-language evaluations of span corruption cross entropy loss vs. inference runtime for each DelT5 and baseline model.



(a) Delta of cross entropy losses.

(b) Percentage of deleted tokens.

Figure 9: Histograms of individual sample cross entropy loss deltas and deletion percentages for each DelT5 model.



(a) Delta of cross entropy losses.

(b) Percentage of deleted tokens.

Figure 10: Histograms of individual sample cross entropy loss deltas and deletion percentages for each random baseline model.