MrT5: Dynamic Token Merging for Efficient Byte-level Language Models

Anonymous authors
Paper under double-blind review

Abstract

Models that rely on subword tokenization have significant drawbacks, such as sensitivity to character-level noise like spelling errors and inconsistent compression rates across different languages and scripts. While character or byte-level models like ByT5 attempt to address these concerns, they have not gained widespread adoption—processing raw byte streams without tokenization results in significantly longer sequence lengths, making training and inference inefficient. This work introduces **MrT5** (**Mer**ge**T5**), a more efficient variant of ByT5 that integrates a token deletion mechanism in its encoder to *dynamically* shorten the input sequence length. After processing through a fixed number of encoder layers, a learnt *delete gate* determines which tokens are to be removed and which are to be retained for subsequent layers. MrT5 effectively "merges" critical information from deleted tokens into a more compact sequence, leveraging contextual information from the remaining tokens. In continued pre-training experiments, we find that MrT5 can achieve significant gains in inference runtime with minimal effect on performance. When trained on English text, MrT5 demonstrates the capability to transfer its deletion feature zero-shot across several languages, with significant additional improvements following multilingual training. Furthermore, MrT5 shows comparable accuracy to ByT5 on downstream evaluations such as XNLI and character-level tasks while reducing sequence lengths by up to 80%. Our approach presents a solution to the practical limitations of existing byte-level models.[1]

1 Introduction

Subword tokenization, typically via algorithms such as byte-pair encoding (Sennrich et al., 2016) or SentencePiece (Kudo & Richardson, 2018), is a fundamental text preprocessing step that has become ubiquitous 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 & Mahowald, 2022; Huang et al., 2023); they directly impact how models process digits and perform arithmetic (Singh & Strouse, 2024); and they have disproportionate compression rates for different languages and scripts (Ahia et al., 2023; Petrov et al., 2023). In addition, current language model APIs charge users per-token, and such discrepancies can cause users of certain languages to be overcharged for poorer compression.[2]

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 limiting factor for Transformer models. 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

[1] We will open-source our code upon acceptance.

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

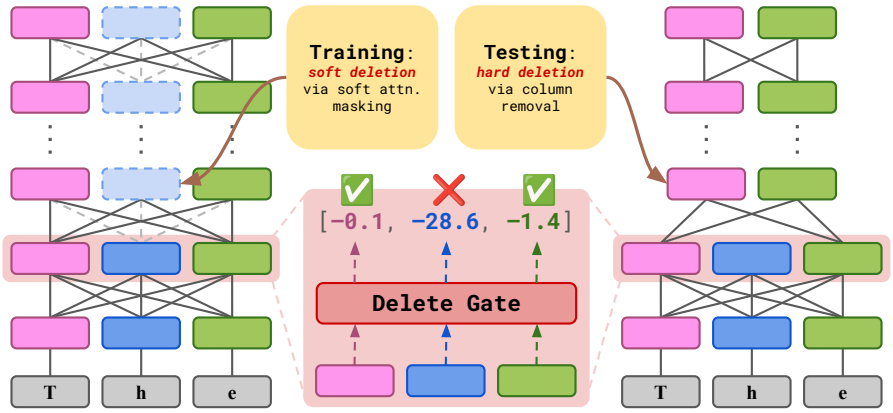


Figure 1: MrT5's encoder during training and testing. During training, fully-differentiable *soft deletion* masks out tokens using the output of MrT5's delete gate. During testing, *hard deletion* removes columns from the computation, which reduces the sequence length and leads to efficiency gains. In this visual, the delete gate is placed at layer 2, but the gate placement may be tuned.

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 that cannot be used to easily adapt existing pre-trained models.

In this work, we propose **MrT5** (**Mer**ge**T5**), a variant of the ByT5 architecture that helps address its inefficiencies while allowing more flexibility than fixed-span downsampling methods (Section 3). MrT5 dynamically merges its encoder's input into a shorter sequence using a token deletion gating mechanism at a fixed, early encoder layer, as shown in Figure 1. By allowing the first few encoder layers to process the entire sequence, the encoder creates contextualized representations of the tokens. When the gating mechanism then deletes a subset of the tokens, those that remain keep the contextual information about those that were removed, allowing information to be implicitly merged into a shorter sequence. During training, we use a deletion regularizer with a tunable weight that can adjust the amount of deletion MrT5 performs. MrT5 effectively learns to merge relevant tokens and eliminate extraneous ones in a completely unsupervised manner, by optimizing the balance between the regularization objective and language modeling.

We first train several MrT5 models on synthetic tasks (Section 4); we find that MrT5 not only drops tokens that are irrelevant to the tasks, but also compresses the relevant context into a shorter sequence by applying meaningful deletion patterns tailored to the tasks. Next, we perform continued pre-training experiments by fine-tuning the MrT5 gating mechanism on top of the pre-trained ByT5 Small (Section 5). Our results indicate that MrT5 outperforms random and fixed token dropping 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 MrT5 trained only on English can apply its deletion mechanism to several languages; we also find that we can further improve MrT5's deletion rates across languages with different scripts through multilingual training. In our final set of experiments, we evaluate MrT5 on XNLI and character-level tasks (Section 6), showing that MrT5 achieves comparable accuracy to ByT5 while significantly improving inference runtimes by reducing the sequence length by up to 80%. Our approach improves on the main 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.

2 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" that learns a block scoring function to select the byte embeddings to pool together for more efficient training and inference.

This paper focuses on ByT5 (Xue et al., 2022), a byte-level sequence-to-sequence Transformer architecture (Vaswani et al., 2017) that serves as a counterpart to mT5 (Xue et al., 2021), the multilingual version of T5 (Raffel et al., 2020). ByT5 requires significantly fewer parameters for its vocabulary matrix (which is comprised of only 256 embeddings). However, 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, large model and feed forward dimensionalities, and short input sequence length (1024 bytes) make it quite inefficient. The architecture itself requires about 1.2 times more operations than mT5, which contributes to a 33% longer pre-training wall clock time for ByT5, even though it was trained on only a quarter of the data used for mT5. In terms of inference speed on downstream tasks, ByT5 can be up to 10 times slower than mT5, depending on the input sequence length.

Our model is closely related to early-exit methods proposed for autoregressive Transformers (Elbayad et al., 2020; Schuster et al., 2022) and BERT (Xin et al., 2020; 2021). In contrast to previous approaches, our method is fully differentiable and does not require special training considerations or calculating the entropy of the final classifier, and the deletion decisions are made in a single layer, making the model easy to use and efficient.

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" that are not determined dynamically and do not necessarily correspond to meaningful units of text. 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 & Van Durme, 2023; Qin et al., 2023), which encodes the whole sentence, but passes only a dynamic subset of the embeddings to the decoder. This approach does not save compute on the encoder side.

Unlike previous work, MrT5's deletion gating mechanism 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 gating can also be applied to models trained from scratch. 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.

3 The MrT5 Model Architecture

The unique aspect of MrT5 is its deletion gating mechanism: after a fixed encoder layer, a *delete gate* determines which tokens in the sequence should be kept to be processed by later layers, and which tokens may be deleted, thereby "merging" information into a shorter sequence. Our choice to delete tokens only in a single layer has three motivations: (1) we want to avoid the overhead of executing the deletion algorithm multiple times; (2) we observe that both the performance and the number of deleted tokens stabilize after a few initial layers (see Section 7); and (3) in terms of minimizing computation costs, the deletion is most beneficial if done in an early layer.

3.1 Deletion Gating Mechanism

The MrT5 deletion gating mechanism is inspired by existing architectures with gating mechanisms such as Long-Short Term Memory (LSTMs, Hochreiter & Schmidhuber, 1997), Gated Recurrent Units (GRUs, Cho et al., 2014), and Mixture-of-Experts (MoEs, Shazeer et al., 2017). MrT5's delete gate is placed after the output of a fixed encoder layer l and is defined by the following function:

$$\mathbf{G} = k\sigma(\mathbf{H}_l\mathbf{W} + \mathbf{1}_N b) \tag{1}$$

where $\mathbf{H}_l \in \mathbb{R}^{N \times d}$ are the hidden states output by layer l; $\mathbf{W} \in \mathbb{R}^{d \times 1}$; $b \in \mathbb{R}$; $\mathbf{G} \in \mathbb{R}^{N \times 1}$; k is a large negative constant; N is the encoder input sequence length; d is ByT5's hidden state/model dimensionality; and $\mathbf{1}_N \in \mathbb{R}^{N \times 1}$ is a vector of ones. The gating activation function is a rescaled and translated sigmoid function, bounded between k and 0. In our experiments, we use $k = -30$.

Hard and Soft Deletion. During training, MrT5 deletes tokens *softly*, where the outputs of the gating mechanism $\mathbf{G}$ are applied as a soft attention mask. The outputs are added directly to the self-attention mechanism of the subsequent encoder layers, as well as the cross attentions between the decoder and encoder. As an example, the hidden states of encoder layer $l + 1$ are defined as:

$$\mathbf{H}_{l+1} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}} + \mathbf{1}_N \mathbf{G}^\top\right)\mathbf{V} \tag{2}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{H}_{l+1} \in \mathbb{R}^{N \times d}$ and $\mathbf{1}_N \in \mathbb{R}^{N \times 1}$ is a vector of ones. A token at sequence position $i \in [1, N]$ and $\mathbf{G}_i \approx 0$ will not be masked, whereas a token at sequence position $j \neq i \in [1, N]$ and $\mathbf{G}_j \approx k$ will be masked, since k is a large negative constant. Though soft deletion does not reduce the sequence length, we apply it during training to emulate the effect of token deletion while being fully differentiable. To see efficiency gains during inference, we apply *hard deletion*, where the hidden states are removed from the sequence, determined by a hard threshold; we set this threshold to be $\frac{k}{2}$, half of the range of the delete gate's output.

For different samples in a given batch, different numbers of tokens may 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 deleting and padding the hidden states, since the T5 architecture uses relative position biases at each layer, the deletion and padding is also performed on the position biases. For a theoretical analysis of the compute savings gained by MrT5's hard deletion, see Appendix B.

3.2 Gate Regularizer

MrT5 allows deletion rates to be adjusted using a tunable regularizer loss:

$$\mathcal{L}_{\mathbf{G}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{G}_i \tag{3}$$

This loss is the average of the gate output values, which encourages them to be more negative (i.e. closer to k, the minimum gate value). In other words, as this loss decreases, the number of deleted tokens increases. The total loss is defined as the sum $\mathcal{L} = \mathcal{L}_{\text{CE}} + \alpha\mathcal{L}_{\mathbf{G}}$, where $\mathcal{L}_{\text{CE}}$ is the cross entropy loss. Varying the hyperparameter α allows the MrT5 model to delete more or fewer tokens.

Optimizing a Specific Deletion Ratio. For most of our experiments, we set α by hand, which allows the model to dynamically set the deletion ratio depending on the difficulty of the task. Alternatively, we can optimize for a specific ratio of deleted tokens. This can be done using an algorithm that resembles the proportional controller (P-controller) from classical control theory. Let's call the proportion of deleted tokens in the current batch $\delta_t \in [0, 1]$, the regularization hyperparameter for the current batch α_t and the target deletion ratio $\hat{\delta}$. We update α after each training step as follows:

$$\alpha_{t+1} = \text{clamp}\left(\alpha_t + k_p(\hat{\delta} - \delta_t)\right) \tag{4}$$

where $\text{clamp}(x) = \max(x, 0)$. We found that $k_p = 10^{-6}$ and $\alpha_0 = 0.0$ work well in practice. This method is easier to use than manually setting α and allows α to change dynamically as the model undergoes phase transitions during training, resulting in more stable learning.

Softmax$_1$. It is possible for all elements of $\mathbf{G}$ to equal the minimum gate value such that $\mathbf{G}_i = k$ for all $i \in [1, N]$. This $\mathbf{G}$ would satisfy the gate regularizer but fail to act as an attention mask, since adding the same value to all elements of the input to a standard softmax function does not affect its output. To help avoid this scenario, we use softmax_1 (Miller, 2023) in the attention mechanism:

$$(\text{softmax}_1(\mathbf{x}))_i = \frac{\exp(\mathbf{x}_i)}{1 + \sum_j \exp(\mathbf{x}_j)} \tag{5}$$

With the softmax_1 function, if $\mathbf{G}_i = k$ for all $i \in [1, N]$, as k becomes negative, the sum of attention scores approaches zero. This eliminates the failure case of using tokens that appear to be all deleted. For consistency, we use softmax_1 for both MrT5 and baseline ByT5 models.

Task	Input	Target
Simple Vowel Removal	zEKRreJcBxGUJQbZSIos	zKRrJcBxGJQbZSs
Contextual Vowel Removal	EOubXgaYVbiOgiIrEnld	EOubXgYVbOgIrnld
Sequence Merge	KjAxIpABCZCxBcniABCs	KjAxIpDZCxBcniDs

Table 1: Synthetic tasks with example input and target sequences. In our experiments, sequences are 128 characters/tokens long, including a start and end token. Legend: vowels to remove, vowels to keep, sequences to replace.

4 Simulations

We first train tiny 31M-parameter MrT5 and T5 models with 9 encoder layers and 3 decoder layers from scratch on three synthetic tasks: a simple vowel removal task, a contextual vowel removal task, and a sequence merge task. The purpose of these experiments is to verify that the architecture behaves as intended, particularly with regard to the merging patterns it learns. Does MrT5 merely drop tokens when they are irrelevant to the output, or does it effectively merge relevant context into a shorter sequence? These results help set the stage for our continued pre-training experiments in Section 5 and our downstream task evaluations in Section 6.

Synthetic Task Specifications. Each of our three synthetic tasks is a variant of a copy task, designed to assess MrT5's ability to identify unimportant or redundant information in the input sequence or merge relevant information from some tokens into other tokens. Input sequences are comprised of random lowercase and uppercase English characters, plus start and end tokens. Example inputs and labels are provided in Table 1.

1. **Simple Vowel Removal**: generate a copy of the input token sequence, except for any vowels. We expect MrT5 to delete vowels, which occur with 19% probability. Thus, the optimal sequence length decrease is 19%.
2. **Contextual Vowel Removal**: generate a copy of the input token sequence, except for any vowels that follow a lowercase consonant. We increase the probability of vowels such that, on average, they comprise 40% of the sequence, and about 18% of the sequence is comprised of vowels that follow a lowercase consonant. If MrT5 learns the relevant deletion pattern, we would expect a sequence length decrease of 18%.
3. **Sequence Merge**: generate a copy of the input token sequence, and translate any occurrence of the character sequence `ABC` into the character `D`. `ABC` sequences are inserted into the input randomly and occur about 10 times per sequence on average. If MrT5 merges `ABC` into a single token, we would expect it to drop 20 tokens on average, which is 15.6% of the sequence.

For the model architecture and training configurations we use for the simulations, see Appendix A.1.

Results. Table 2 presents the performance of several MrT5 models that use different regularizer α values trained on each of our three synthetic tasks. We vary α across model training runs to examine the different deletion patterns that emerge. In each of the three synthetic tasks, several MrT5 models with optimal deletion rates outperformed or nearly matched the T5 model while using shorter sequence lengths. We also observed that some MrT5 models, when tuned with the optimal value of α, developed token-dropping strategies that exactly aligned with the specific requirements of the tasks. For instance, in the simple vowel removal task, the MrT5 models that selectively dropped vowels but kept consonants intact showed improved performance.[3] These findings suggest that MrT5 can create complex deletion strategies that exploit patterns or redundancies in the input.

5 Continued Pre-training

In our main set of experiments, we train MrT5 models on the ByT5 span corruption task. In this pre-training objective, spans of tokens in unlabeled text data are replaced with a single *sentinel* token

[3]For tasks requiring more contextual information, we found that these models have too much capacity for the synthetic tasks, and even the T5 models only started to develop non-trivial attention patterns in very late layers. We found that placing the delete gate at later task layers allowed MrT5 to develop more contextual token deletion solutions.

Model	Token-level Accuracy (%)	Seq.-level Accuracy (%)	Seq. Length Reduction (%)	Delete Gate Layer	Description of Deleted Tokens
T5	99.97	96.44	0.00	—	—
MrT5 ($\alpha = 0$)	99.99	99.64	18.94	3	Most vowels, and no consonants.
MrT5 ($\alpha =$ 1e–4)	99.75	77.51	51.13	3	All vowels and many consonants.

(a) Models trained on the simple vowel removal task. MrT5 solves this task effectively with a non-trivial deletion rate, but too much deletion—-beyond the optimal rate of 19%-—results in a drop in performance.

Model	Token-level Accuracy (%)	Seq.-level Accuracy (%)	Seq. Length Reduction (%)	Delete Gate Layer	Description of Deleted Tokens
T5	99.99	99.19	0.00	—	—
MrT5 ($\alpha =$ 1e–3)	99.99	98.54	1.56	3	Only start and end tokens.
MrT5 ($\alpha =$ 1e–3)	99.97	96.51	18.50	7	Vowels after lowercase consonants.
MrT5 ($\alpha =$ 1e–3)	99.96	95.19	18.50	8	Vowels after lowercase consonants.

(b) Models trained on the contextual vowel removal task. Due to the task's contextual nature, deletion works best in higher task layers. MrT5 learns to delete around the optimal 18%, with only a slight performance drop.

Model	Token-level Accuracy (%)	Seq.-level Accuracy (%)	Seq. Length Reduction	Delete Gate Layer	Description of Deleted Tokens
T5	99.99	98.41	0.00	—	—
MrT5 ($\alpha =$ 1e–3)	99.99	98.90	1.56	3	Only start and end tokens.
MrT5 ($\alpha =$ 1e–3)	99.98	97.68	8.30	6	'B' within 'ABC' sequences.
MrT5 ($\alpha =$ 1e–3)	99.85	85.34	16.93	7	All 'B's, and 'BC' within 'ABC' sequences.

(c) Models trained on the sequence merge task. MrT5 can learn to merge characters within ABC sequences at higher task layers, but exceeding the optimal 15.6% deletion rate degrades performance.

Table 2: Synthetic task performance for T5 and MrT5 with different deletion strategies. Token-level accuracy measures the percentage of correctly predicted tokens, averaged across sequences; sequence-level accuracy measures the percentage of sequences with all tokens predicted correctly.

ID per span, and the model must fill in the missing tokens. For ByT5 and MrT5, these are spans of bytes, and the masks can potentially interfere with word boundaries.

We use the English C4 corpus (Raffel et al., 2020) as well as the multilingual C4 (mC4) corpus (Xue et al., 2021) for continued pre-training. We first train several English-only MrT5 and baseline models on C4, and we analyze the performance differences for models with different deletion rates. We also train a multilingual MrT5 and ByT5 model on 15 typologically diverse languages from the mC4 corpus: English, French, Spanish, German, Greek, Bulgarian, Russian, Turkish, Arabic, Vietnamese, Thai, Chinese, Hindi, Swahili, and Urdu. We then compare monolingual and multilingual MrT5 in cross-lingual evaluations, analyzing the zero-shot transfer of English MrT5 and evaluating the performance gains achieved through multilingual training.

Models. We train several MrT5 models, varying the α hyperparameter of the deletion regularizer. In our experiments, we found that $\alpha \in [5\text{e–}3, 1.5\text{e–}2]$ resulted in a broad range of deletion rates. This is a continued pre-training setup; we load the pre-trained weights and use the architecture settings of ByT5 Small (300M parameters) and only randomly initialize the weights of MrT5's gating mechanism. Based on a sweep of different layers, we place the gating mechanism after encoder layer $l = 3$ (see Section 7). In addition to the MrT5 models, we also train several baselines:

1. **ByT5 baseline**: A ByT5 Small architecture with $\mathrm{softmax}_1$, but without deletion.
2. **Random baseline**: We implement and train a set of models with a random gating mechanism, where the choice of how the tokens are deleted is random; some number of gate values are set to k, and the rest are set to 0. In our experiments, we train five random models with different average deletion rates: 15%, 40% 50%, 60%, and 70%.
3. **Fixed baseline**: We implement and train a set of models that delete the ends of words. We train five models that delete different percentages of the ends of words: 15%, 40% 50%, 60%, and 70%. For details on the implementation of this baseline, see Appendix A.4.

For the random and fixed baselines, the gating mechanism is placed at layer $l = 3$, like the MrT5 models. All models use $\mathrm{softmax}_1$ in their attention mechanisms. The ByT5 baseline serves as a

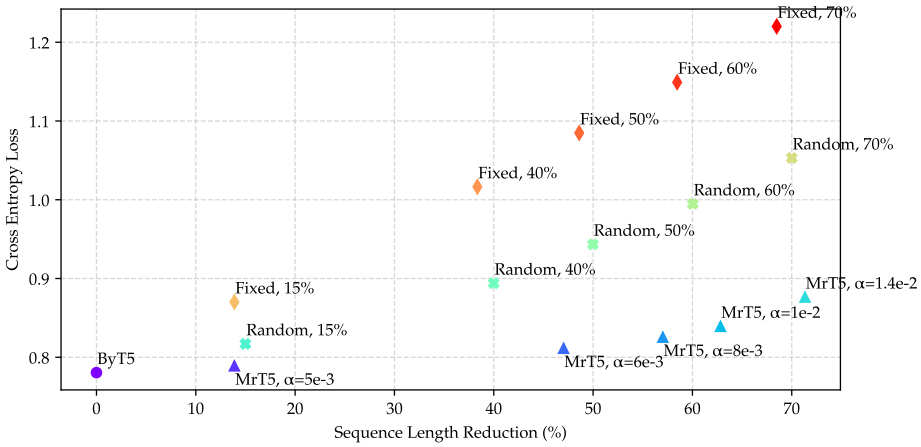


Figure 2: Span corruption cross entropy loss vs. sequence length reduction for each MrT5 and baseline model. MrT5 models consistently have much lower loss than the baselines, and are generally competitive with unmodified ByT5, even where they achieve very large sequence length reductions.

Model	ByT5	MrT5 $\alpha = 5\text{e–}3$	MrT5 $\alpha = 6\text{e–}3$	MrT5 $\alpha = 8\text{e–}3$	MrT5 $\alpha = 1\text{e–}2$	MrT5 $\alpha = 1.4\text{e–}2$
Runtime (ms)	56.27	53.98 (↓ 4.08%)	43.92 (↓ 21.96%)	40.78 (↓ 27.53%)	39.62 (↓ 29.59%)	33.81 (↓ 39.92%)

Table 3: Average inference runtime for a single sequence (in milliseconds) for ByT5 and each MrT5 model. Percentage decrease in runtime relative to ByT5 is displayed in parentheses.

lower bound on the best possible span corruption loss, since it does not reduce the sequence length. For further details on model architectures, dataset preparation, and optimization, see Appendix A.2.

Monolingual Results. Our first set of results are for the monolingual MrT5 models trained only on English data. We evaluate span corruption loss on a held-out test set of 10,000 examples. Figure 2 illustrates the relationship between span corruption cross entropy loss and sequence length reduction for each MrT5 model compared to their respective baselines. As anticipated, ByT5 achieves the lowest loss; however, it does not reduce sequence length. While the MrT5 models, along with the fixed and random baselines, exhibit higher losses overall, the MrT5 models consistently achieve lower losses than the other baselines at comparable deletion rates. For instance, MrT5 with $\alpha = 8\text{e–}3$ removes approximately 57% of tokens, which is a similar deletion rate to the 60% fixed and random baselines, yet MrT5 achieves the lowest loss. This pattern is consistent across all MrT5 models and baselines, showing the effectiveness of MrT5's gating mechanism in reducing the sequence length while minimizing loss.

MrT5 models with a higher deletion rate also have a faster inference runtime, as shown in Table 3. In particular, MrT5 models that reduce the sequence length by more than 50% can achieve 25% speedup or greater, with our implementation. These results show that hard deletion improves the efficiency of MrT5 when compared to ByT5, and this speedup can be tuned.

Multilingual Results. In our second set of results, we evaluate MrT5 and ByT5 models on 15 languages. For the monolingual models trained on English only, these are zero-shot evaluations. Each language's test set is sampled from its mC4 validation split and contains 10,000 examples, except for Swahili and Urdu, which only have 2,800 and 9,300 examples, respectively. Figure 3 shows the test set span corruption loss versus sequence length reduction for a monolingual MrT5 (with constant $\alpha = 1\text{e–}2$) and a multilingual MrT5 model (with constant $\alpha = 1.2\text{e–}2$) across 15 languages. We choose to compare these two models because they learned similar deletion rates for English ($\approx$63%). We also include the corresponding ByT5 models' losses, trained on the same data, as a baseline.

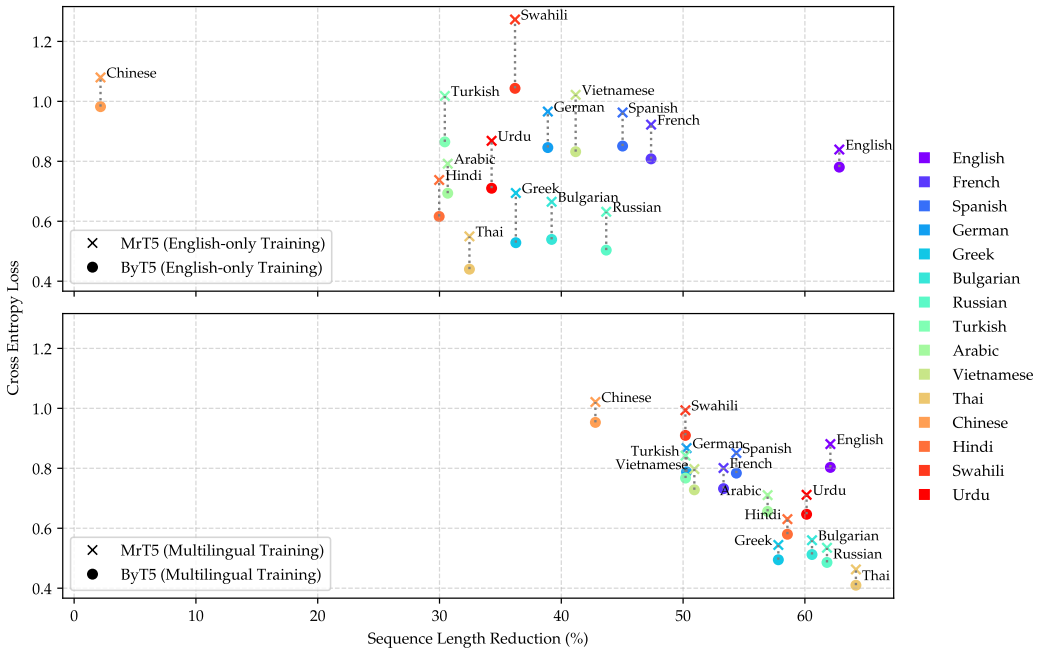


Figure 3: Average test set span corruption loss vs. sequence length reduction for monolingual (top) and multilingual (bottom) MrT5 models across 15 languages. ByT5 is shown for loss comparison only (it does not reduce the sequence length). MrT5 can transfer its deletion feature to new languages when trained in English, but performs best with multilingual training. A multilingual MrT5 model achieves over 50% sequence length reduction in most languages with minimal effect on the loss.

The monolingual MrT5 model achieves the highest sequence length reduction for English, as expected, and performs moderately well in zero-shot settings for Latin-script languages (e.g., Spanish, French, Vietnamese, and Russian), with reduction rates above 40%. However, it struggles with sequence length reduction for Chinese, which uses a completely different script. In contrast, the multilingual MrT5 significantly improves the sequence length reduction across all languages, with most languages achieving 60% reduction; at the same time, the loss increase compared to ByT5 shrinks across all languages. These results demonstrate that the MrT5 architecture can learn to perform contextual deletions and transfer this capability across languages, but its effectiveness in doing so, especially across different scripts, is greatly enhanced when trained with multilingual data.

6 Downstream Task Evaluations

We next assess MrT5's performance on downstream tasks, specifically XNLI and two character-level tasks. XNLI evaluates MrT5's ability to understand semantic relationships between sentences, while the character-level tasks test whether it retains its sensitivity to character-level manipulations.

Cross-lingual Natural Language Inference. We first test our multilingual MrT5 model using the Cross-lingual Natural Language Inference (XNLI) corpus (Conneau et al., 2018), a benchmark for cross-lingual sentence classification with 5,000 parallel examples in 15 languages. These are the same 15 languages our multilingual MrT5 model was trained on. We selected XNLI for testing because the ByT5 authors noted it as one of the downstream tasks with the worst inference runtimes, primarily due to the long input sequences; ByT5 was 6.4 to 9.5 times slower than mT5. We fine-tune two models on the English MultiNLI corpus (Williams et al., 2018): our multilingual MrT5 model ($\alpha = 1.2\text{e}{-2}$) and the baseline multilingual ByT5 model. For training details, see Appendix A.3.

Table 4 presents the evaluation metrics for the ByT5 and MrT5 models on the XNLI English test split, as well as their average performance across all languages. On English, MrT5 outperforms ByT5's accuracy while reducing the sequence length by 52.56% and cutting down the inference runtime by 38.0%. Across languages, MrT5 significantly decreases both the sequence length and

Language	Accuracy (%)		Average Runtime (ms)		MrT5 Runtime Decrease (%)	MrT5 Seq. Len. Reduction (%)
	ByT5	MrT5	ByT5	MrT5		
English	76.47	78.88	8.95	5.55	38.00	52.56
All Languages	51.34	49.63	14.25	8.19	42.50	46.82

Table 4: XNLI evaluation metrics for ByT5 and MrT5 models. Except for English, all evaluations are zero-shot. Chance accuracy is 33%. MrT5 achieves significantly reduced sequence lengths and faster runtimes compared to ByT5 while maintaining comparable accuracy, and it even outperforms ByT5 on English. See Table 6 in Appendix C for metrics on each language individually.

Task	Seq.-level Accuracy (%)		Average Runtime (ms)		MrT5 Runtime Decrease (%)	MrT5 Seq. Len. Reduction (%)
	ByT5	MrT5	ByT5	MrT5		
Spelling Correction	58.19	56.07	3.25	2.18	32.90	78.88
Word Search	75.37	74.30	4.95	2.22	55.15	73.91

Table 5: Character-level task metrics for ByT5 and MrT5 models. MrT5 significantly reduces runtime and sequence lengths across both tasks, with a minimal accuracy trade-off in spelling correction and comparable accuracy in word search. See Table 7 and Table 8 in Appendix C for metrics on individual test splits for each task.

runtime, with a small accuracy decrease of just 1.7% compared to ByT5. The XNLI task illustrates the efficiency of MrT5's fast encoder; a 50% deletion rate results in substantial gains in runtime. This improvement can be attributed to the setup of the XNLI task, which requires large input sequences (up to 1,024 tokens) and short decoder sequences (a single token for classification).[4]

Character-level Tasks. We fine-tune and evaluate an English MrT5 model on the *Spelling Correction with Context* and *Word Search* character-level tasks from Huang et al. (2023). In the Spelling Correction task, the input is a sentence containing a spelling error, and the goal is to generate the same sentence with the error corrected. In the Word Search task, the input follows the format `definition: letters`, and the objective is to identify the substring in `letters` that, when reversed, matches the given `definition`. We selected these two tasks from the suite because they both require understanding meaning and context and involve processing longer sequence lengths. For each character-level task, we fine-tune two models: our English-only MrT5 model (α = 1e–2) and the baseline English-only ByT5 model. See Appendix A.3 for fine-tuning optimization details. We evaluate on all test splits from Huang et al. (2023), which are designed to rigorously assess a model's ability to integrate meaning and context in its predictions.

Table 5 displays the test set results for each character-level task. For both tasks, MrT5 reduces the sequence length by over 70%, speeding up the inference runtime by 32.9% for the Spelling Correction task and and 55.15% for the Word Search task. At the same time, MrT5 maintains accuracy scores that are competitive with ByT5. These results show that MrT5's method of sequence merging effectively preserves its sensitivity to character-level information.

7 Analysis

Per-sample Sequence Length Reduction. We present a per-sample analysis of the cross entropy loss and sequence length reduction for the English MrT5 models and random baselines. We take a sample of 1,000 English sentences from the mC4 test set and calculate the percent increase in loss on a per-sample basis, using the English ByT5's loss as the baseline (i.e. the percent increase between the MrT5 model's loss and ByT5's loss for individual samples). For each sample, we also get the sequence length reduction. Across five MrT5 models with different deletion rates, we found no correlation between the percent increase in the loss and the percentage of tokens deleted (average correlation of $r = -0.014$).[5] This reflects what we would expect from the MrT5 models; for an

[4] We note that the accuracy of our ByT5 Small model on XNLI (51.3%) falls short of the accuracy reported in the ByT5 paper (69.1%). We attribute this discrepancy primarily to the significantly longer training duration used in their fine-tuning setup—262,144 steps, compared to our 4,000 steps.

[5] When averaging correlation coefficients, we apply Fisher's Z transformation to stabilize the variance.

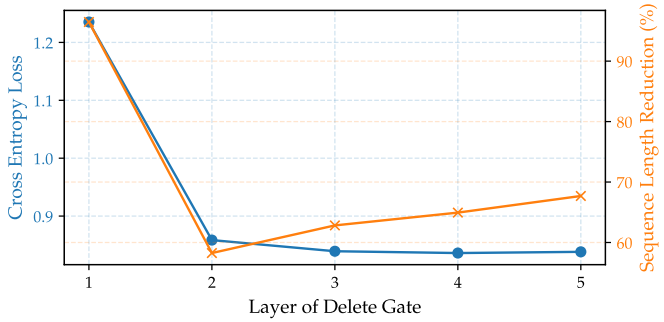


Figure 4: Cross-entropy loss and sequence length reduction for MrT5 models with delete gates at different layers. Loss is higher in layers 1 and 2, but stabilizes from layer 3 onward. Layers 4 and 5 only enable slightly more deletion than layer 3, so we select layer 3 as optimal for gate placement.

individual sample, MrT5 learns when it can delete more tokens without incurring a large increase in the loss. In contrast, for five random models with different sequence length reduction rates, we found a moderate positive correlation (average correlation of $r = 0.298$). When the random model removes more tokens, it is more likely to cause an increase in loss. These results further support the observation that the MrT5 models more strategically and contextually delete tokens compared to the baselines. See Figure 6 in Appendix D for plots presenting the correlations for each individual MrT5 and random baseline model.

Gate Placement. We present an analysis of MrT5 models with delete gates placed at various encoder layers. To maximize efficiency, it is ideal to position the delete gate in the earliest encoder layer possible. However, placing the gate too early reduces the contextual information in the token representations, leading to more significant performance degradation compared to ByT5. We trained several English-only MrT5 models, all using the same training setup and architecture as described in Section 5, with a fixed regularizer $\alpha = 1\text{e}{-2}$. The only variable was the placement of the delete gate, as shown in Figure 4. Our findings indicate that the loss is higher when the gate is placed in early layers but stabilizes after layer 3. Although higher layers remove slightly more tokens, our goal is to place the gate as early as possible. Therefore, we selected layer 3 as the optimal point for gate placement. This analysis provides further evidence that MrT5 merges information into fewer tokens, since deletion at earlier, less contextual layers results in a higher loss.

The gate placement sets an upper bound on the compute savings achievable with our model. We provide a detailed analysis of MrT5's theoretical compute savings in Appendix B. With typical hyperparameters from our tasks, we can achieve a maximum speedup of around three times with reasonable deletion rates.

8 Conclusion

In this paper, we introduce **MrT5** (**Mer**ge**T5**), a variant of the ByT5 architecture designed to address the inefficiencies of byte-level language modeling. MrT5's token deletion mechanism forces the model to merge input tokens into a more compact sequence, allowing for computational savings while preserving model performance. Our synthetic experiments demonstrate that MrT5 effectively merges relevant context into a shorter sequence using strategies that align with task-specific objectives. In our continued pre-training experiments, MrT5 outperforms both random and fixed deletion baselines when trained in English, and with multilingual training, MrT5 achieves over 50% reduction in sequence length across multiple languages with minimal impact on the loss. Our model learns very fast: the continued pre-training requires only a few thousand additional training steps. Furthermore, MrT5 maintains competitive accuracy with ByT5 on downstream tasks such as XNLI and character-level manipulations while improving inference runtimes. This demonstrates MrT5's capacity to handle tasks requiring semantic information, while still effectively processing character-level details–the main advantage of byte-level modeling. Our work takes a significant step toward the viability of byte-level language models and eliminating the need for subword tokenization.

Reproducibility Statement

Steps for reproducing each of our experiments are detailed in Appendix A. Descriptions of the model architectures and training configurations/hyperparameters for our synthetic task experiments are provided in Appendix A.1; details of the model architectures, span corruption data preprocessing steps, and training configurations/hyperparameters for continued pre-training are provided in Appendix A.2; and training configurations/hyperparameters for fine-tuning on the XNLI and character-level downstream tasks are provided in Appendix A.3. We provide anonymized source code as part of the submission, and we will open-source our code upon acceptance.

References

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David Mortensen, Noah Smith, and Yulia Tsvetkov. Do all languages cost the same? tokenization in the era of commercial language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9904–9923, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.614. URL https://aclanthology.org/2023.emnlp-main.614.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans (eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://aclanthology.org/D14-1179.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91, 2022. doi: 10.1162/tacl_a_00448. URL https://aclanthology.org/2022.tacl-1.5.

Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel R. Bowman, Holger Schwenk, and Veselin Stoyanov. Xnli: Evaluating cross-lingual sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *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)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, Virtual only, May 2020.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Jing Huang, Zhengxuan Wu, Kyle Mahowald, and Christopher Potts. Inducing character-level structure in subword-based language models with type-level interchange intervention training. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 12163–12180, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.770. URL https://aclanthology.org/2023.findings-acl.770.

Ayush Kaushal and Kyle Mahowald. What do tokens know about their characters and how do they know it? In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2487–2507, Seattle, United

States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.179. URL https://aclanthology.org/2022.naacl-main.179.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In Eduardo Blanco and Wei Lu (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL https://aclanthology.org/D18-2012.

Evan Miller. Attention is off by one, Jul 2023. URL https://www.evanmiller.org/attention-is-off-by-one.html.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Lukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. Hierarchical transformers are more efficient language models. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 1559–1571, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.117. URL https://aclanthology.org/2022.findings-naacl.117.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. Efficient transformers with dynamic token pooling. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6403–6417, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.353. URL https://aclanthology.org/2023.acl-long.353.

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

Guanghui Qin and Benjamin Van Durme. Nugget: Neural agglomerative embeddings of text. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 28337–28350. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/qin23a.html.

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

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. *Jorunal of Machine Learning Research (JMLR)*, 21:140:1–140:67, 2020.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022*, New Orleans, LA, USA, November 2022.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

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

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

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

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

Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1112–1122. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/N18-1101.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL*, pp. 2246–2251, Virtual only, July 2020.

Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. Berxit: Early exiting for BERT with better fine-tuning and extension to regression. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 91–104, Virtual only, April 2021.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mT5: A massively multilingual pre-trained text-to-text transformer. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 483–498, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.41. URL https://aclanthology.org/2021.naacl-main.41.

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. doi: 10.1162/tacl_a_00461. URL https://aclanthology.org/2022.tacl-1.17.

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

A Experimental Details

A.1 Simulation Details

Model Architectures. We train our synthetic models with 9 encoder layers and 3 decoder layers, following the 3:1 ratio of encoder to decoder layers in ByT5, and we use $d_{\text{ff}} = 1024$ and $d_{\text{model}} = 512$. We use softmax_1 for all T5 and MrT5 models. Other architectural settings match the standard ByT5 Small, resulting in an architecture with 31M parameters (10% of ByT5 Small's parameter count).

Optimization. We use a batch size of 128 examples and a sequence length of 128 tokens, and we train each model for a total of 20,000 gradient steps. We use the AdamW optimizer with a learning rate that linearly warms up to 1e–4 over 1,000 steps and linearly decays. For MrT5 models, the delete gate's regularizer is enabled half-way through training, at 10,000 steps. We set a constant regularizer α throughout training.

A.2 Continued Pre-training Details

Model Architectures. All MrT5 and baseline models use the model configuration of a standard ByT5 Small, which has $d_{\text{ff}} = 3584$, $d_{\text{model}} = 1472$, 12 encoder layers, 4 decoder layers, and 300M total parameters. The only difference for MrT5 and the baselines is the additional delete gate and the use of softmax_1 in the attention mechanisms.

Data. When training on the span corruption objective, we calculate the corrupted spans such that the average masked 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. For both monolingual and multilingual model training, we ensure that the samples of the mC4 corpus are sufficiently large to avoid training the models for multiple epochs. In the case of multilingual training, we extract equal-sized samples for each language from the mC4 training split.

Optimization. We train each model for 3,000 gradient steps over batches of 2^{20} tokens (i.e. an encoder sequence length of 1024 with an effective batch size of 1024). We use the AdamW optimizer with an initial learning rate of 1e–4 with linear decay and no warmup. As with the simulations from the previous section, we keep the gate regularizer α constant throughout training to allow the model to discover its own deletion rate, but we vary α across runs. Since we are continuing to train on ByT5's pre-training objective, we do not delay the regularizer.

At test time, we use an eval batch size of 2^{14} tokens (i.e. an encoder sequence length of 1024 with a batch size of 16). We use use the last model checkpoint at step 3,000 for all evaluations.

A.3 Downstream Task Details

XNLI Training Details. We train all models for 4,000 gradient steps (≈10.43 epochs) with a batch size of 1,024 and a maximum sequence length of 1,024 tokens. We use the AdamW optimizer with an initial learning rate of 1e–3 that linearly decays with no warmup. For MrT5, we apply a P-controller to achieve a target deletion ratio $\hat{\delta} = 0.5$, aiming for a sequence length reduction of ≈50%, and we delay the regularizer until 500 steps into fine-tuning. The controller parameters are $k_p = 10^{-6}$ and $\alpha_0 = 0.0$.

We evaluate both models on the XNLI dataset, testing them in English and conducting zero-shot evaluations in the 14 additional languages. For evaluation, we use a batch size of 16 and a maximum sequence length of 1,024 tokens.

Character-level Task Training Details. For the Spelling Correction task, we train for 200,000 gradient steps (≈33.7 epochs); for the Word Search task, we train for 300,000 steps (≈ 61.0 epochs). When training MrT5, we apply a P-controller to achieve a target deletion ratio. On the Spelling Correction task, we set the target deletion ratio $\hat{\delta} = 0.6$, aiming for a sequence length reduction of ≈60% (although the model learned to delete closer to ≈80% of tokens). On the Word Search task, we set a target deletion ratio $\hat{\delta} = 0.7$, aiming for a sequence length reduction of ≈70%. We delay the regularizer until 10,000 steps into fine-tuning for both tasks.

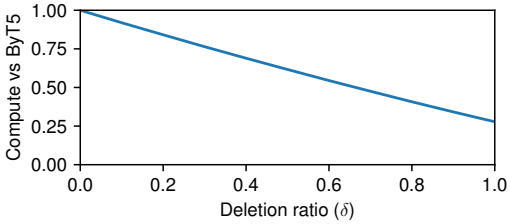


Figure 5: Reduction in the total amount of compute as a function of the deletion ratio δ.

For both tasks, we use a batch size of 16 examples, and we use the AdamW optimizer with an initial learning rate of 5e–4 that linearly decays with no warmup. For evaluation, we use a batch size of 16. The Spelling Correction and Word Search tasks have a maximum input sequence length of 64 and 128 tokens, respectively.

A.4 Description of Fixed Deletion Baselines

The fixed deletion baseline deletes tokens at layer $l = 3$ using deterministic rules based on the token identity. All tokens/columns corresponding to whitespace, punctuation, and symbolic characters are identified: `\t`, `\n`, `␣`, `!`, `"`, `#`, `$`, `%`, `&`, `'`, `(`, `)`, `,`, `+`, `,,` `-`, `.`, `/`, `:`, `;`, `<`, `=`, `>`, `?`, `@`, `[`, `\`, `]`, `;` `_`, `;` `{`, `|`, `}`, `~`, `</s>`. These separator tokens are used to locate word boundaries. Then, based on the fixed deletion percentage, the delete gate will drop the tokens/columns corresponding to the ends of words. For example, if the target percentage is 50%, the delete gate will remove the tokens corresponding to the final two characters of a five letter word, and the final three characters of a six character word.

B Theoretical Compute Savings

Here we analyze the theoretical amount of compute used by MrT5 given a deletion rate. Let's call the average length of the input sequence N_E and the average length of the output sequence N_D. The width of the residual is d_{model}, the dimension of the up-projection in the MLP is d_{ff}, our encoder has L_E and the decoder has L_D layers. The deletion occurs after the L_{del} layer and the average proportion of deleted tokens is δ. We assume that the total size of the head projections is equals to d_{model}, as typical for Transformers ($d_{\text{head}} * N_{\text{heads}} = d_{\text{model}}$). Then, we can approximate the total number of multiply-accumulate operations (MACs) for the model as follows. Before deletion, the self attention in the encoder uses $N_E d_{\text{model}}^2$ MACs for both the Q, K, V and the output projections and $N_E^2 d_{\text{model}}$ MACs for both the $\mathbf{A} = \mathbf{QK}^T$ and $\mathbf{AV}$ projections. The MLP layer uses $N_E d_{\text{model}} d_{\text{ff}}$ additional MACs. Thus, the total number of MACs used per layer is $4N_E d_{\text{model}}^2 + 2N_E^2 d_{\text{model}} + N_E d_{\text{model}} d_{\text{ff}}$. This much compute is used for the first L_{del} layers, after which the sequence length is reduced to $N_E(1-\delta)$ for the remaining $L_E - L_{\text{del}}$ layers. Thus, the encoder uses

$$\begin{aligned} N_{\text{MACs}}^{\text{encoder}} = (L_E - L_{\text{del}})\,(1-\delta)\left(4N_E d_{\text{model}}^2 + 2N_E^2 d_{\text{model}}(1-\delta) + N_E d_{\text{model}} d_{\text{ff}}\right) + \\ L_{\text{del}}\left(4N_E d_{\text{model}}^2 + 2N_E^2 d_{\text{model}} + N_E d_{\text{model}} d_{\text{ff}}\right) \end{aligned} \tag{6}$$

The MACs used by the decoder can be calculated similarly, but additionally the cross attention has to be taken into account. The cross attention uses $N_E(1-\delta)d_{\text{model}}^2$ MACs for the K and V projections, $N_D d_{\text{model}}^2$ MACs for the Q and output projections, and $(1-\delta)N_E N_D d_{\text{model}}$ MACs for the attention matrix itself.

$$\begin{aligned} N_{\text{MACs}}^{\text{decoder}} = L_D\left(4N_D d_{\text{model}}^2 + 2N_D^2 d_{\text{model}} + N_D d_{\text{model}} d_{\text{ff}} + 2N_E(1-\delta)d_{\text{model}}^2 + \right. \\ \left. 2N_D d_{\text{model}}^2 + 2(1-\delta)N_E N_D d_{\text{model}}\right) \end{aligned} \tag{7}$$

Note that $\delta = 0$ corresponds to the baseline ByT5. For MrT5, $d_{\text{model}} = 1472, d_{\text{ff}} = 3584, L_E = 12, L_D = 4, L_{\text{del}} = 3, N_E = 1024, N_D = 189$. Given these numbers, we plot the total reduction in compute (compared to ByT5) as a function of δ in Fig. 5.

Language	Accuracy (%)		Average Runtime (ms)		MrT5 Runtime Decrease (%)	MrT5 Seq. Len. Reduction (%)
	ByT5	MrT5	ByT5	MrT5		
English	76.47	78.88	8.95	5.55	38.0	52.56
French	53.90	52.25	10.75	6.66	38.0	47.58
Spanish	55.99	53.84	10.13	6.09	39.9	50.67
German	45.68	46.20	10.42	6.59	36.8	47.61
Greek	52.01	52.33	19.08	9.77	48.8	54.79
Bulgarian	56.31	53.16	17.58	9.68	44.9	47.82
Russian	55.35	55.85	18.02	9.95	44.8	47.95
Turkish	44.86	43.31	9.73	6.03	38.0	46.72
Arabic	51.89	47.29	14.12	8.31	41.1	43.78
Vietnamese	48.63	47.39	12.60	7.67	39.2	46.51
Thai	48.35	44.57	25.28	11.77	53.5	61.07
Chinese	50.70	50.50	7.94	5.83	26.7	24.96
Hindi	43.87	39.99	24.50	13.68	44.2	43.56
Swahili	40.64	38.42	9.08	5.62	38.1	48.48
Urdu	45.48	40.45	15.63	9.77	37.5	38.32
All Languages	51.34	49.63	14.25	8.19	42.5	46.82

Table 6: Per-language XNLI evaluation metrics for ByT5 and MrT5 models. Except for English, all evaluations are zero-shot.

Spelling Correction Test Split	Seq.-Level Accuracy (%)		Average Runtime (ms)		MrT5 Runtime Decrease (%)	MrT5 Seq. Len. Reduction (%)
	ByT5	MrT5	ByT5	MrT5		
Dependent	37.63	35.05	3.28	2.26	31.21	78.77
Independent	76.45	74.76	3.21	2.11	34.44	78.98
All Splits	58.19	56.07	3.25	2.18	32.90	78.88

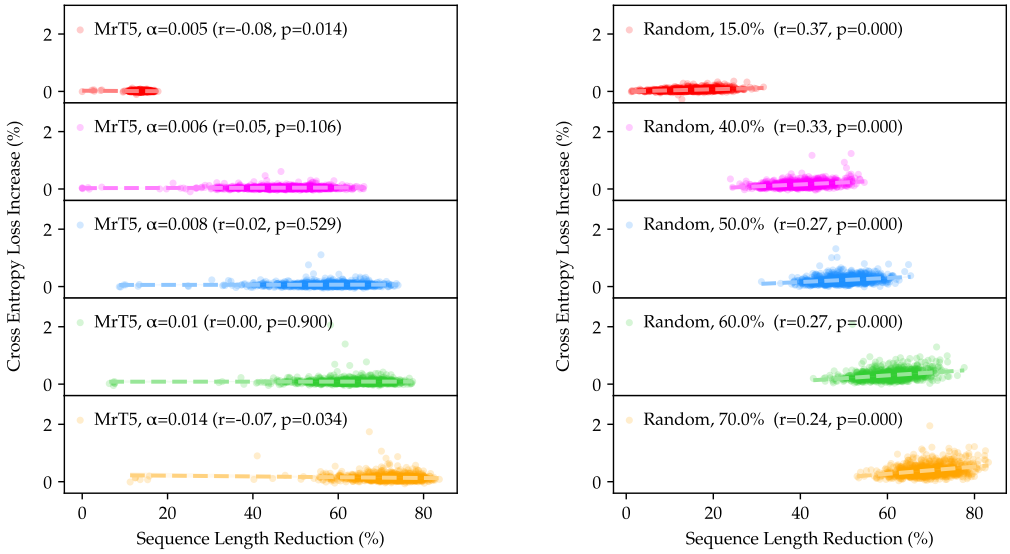
Table 7: Evaluation metrics for all splits for the Spelling Correction with Context character-level task. The "Dependent" split requires incorporating context to correct the spelling error; the "Independent" split does not.

C Additional Downstream Task Evaluations

Table 6 contains XNLI evaluation metrics for ByT5 and MrT5 for each of the 15 languages. Table 7 contains evaluations on all test splits for the Spelling Correction with Context task. Table 8 contains evaluation on all test splits for the Word Search task.

Word Search Test Split	Seq.-Level Accuracy (%)		Average Runtime (ms)		MrT5 Runtime Decrease (%)	MrT5 Seq. Len. Reduction (%)
	ByT5	MrT5	ByT5	MrT5		
OOV	78.05	79.99	5.20	2.55	50.95	72.52
Paraphrase	83.67	82.06	4.89	2.17	55.58	72.27
Overlap	77.58	76.22	4.94	2.20	55.55	76.14
Paraphrase + Overlap	58.46	57.60	4.95	2.21	55.41	73.95
All Splits	75.37	74.30	4.95	2.22	55.15	73.91

Table 8: Evaluation metrics for all splits for the Word Search character-level task. The "OOV" split contains hidden words with mT5 tokenization not seen in the training split (this does not apply to our work, since we only train byte-level models, not subword models); the "Paraphrase" split contains definitions from *The Online Plain Text English Dictionary*, testing the ability to understand context; the "Overlap" split contains overlapping hidden words; and the "Paraphrase + Overlap" split contains both paraphrased definitions and overlapping hidden words.



(a) MrT5 models. All models show no correlation between loss increase and sequence length reduction, showing the MrT5 can deletes at different percentages depending on the sequence, without incurring a loss increase.

(b) Random baseline models. All models show a weak to moderate positive correlation between loss increase and sequence length reduction.

Figure 6: Percent increase in the span corruption loss vs. the percentage of deleted tokens for each (a) MrT5 model and (b) random baseline model. Percent increase is calculated using ByT5's loss as a baseline for a particular sample. Each point represents a single sample.

D Additional Per-sample Analyses

Figure 6 shows per-sample correlation plots between the cross entropy loss percent increase and the sequence length reduction for five MrT5 models and five random baseline models with different deletion rates.

E Additional Experiments with Pooling Baselines

As an additional baseline, we implemented the boundary predictor with pooling method of Nawrot et al. (2023) and report its performance relative to MrT5 in this section.

It is important to note that the boundary predictor method from Nawrot et al. (2023) required several updates in order to be adapted to our experimental setting, as it was originally designed for decoder architectures. Specifically, we made the following modifications to enable its use with an encoder-only model:

1. **Removal of null-group representations**: Null-group representations, which are padded at the beginning of sequences, were excluded as they are unnecessary in an encoder-only context. We found that they hurt model performance.
2. **Handling relative position biases**: The original method does not support models with relative position biases, which are used in the attention mechanism of each layer. To address this limitation, we devised a solution that uses the position biases associated with the initial boundary as the position bias for the span of pooled tokens.

We integrated the boundary predictor and pooling module after the third layer, consistent with the placement of the delete gate for MrT5 in both span corruption and downstream task experiments.

Target Compression (%)	Cross Entropy Loss		Seq. Len. Reduction (%)	
	MrT5	Nawrot et al. (2023)	MrT5	Nawrot et al. (2023)
15	0.79	1.01	15.71	18.77
30	0.80	1.06	30.50	33.64
40	0.81	1.08	41.28	44.10
60	0.83	1.16	60.80	64.11
70	0.87	1.19	72.24	73.43

Table 9: Comparison of cross entropy loss and sequence length reduction for MrT5 models and models using the boundary predictor with pooling method of Nawrot et al. (2023).

We follow Nawrot et al. and use a temperature of 0.5 for the boundary predictor. The results of this comparison are presented below.

Overall, in both the continued pre-training and downstream task experiments, MrT5 outperforms Nawrot et al.'s method in terms of task loss or accuracy while achieving comparable or better compression rates. These results highlight MrT5's ability to effectively balance performance and efficiency, offering a superior alternative to the pooling-based approach.

E.1 Continued Pre-training

We first test the cross entropy loss on the English span corruption task for MrT5 models with models trained using Nawrot et al.'s boundary predictor. All models are trained with the same hyperparameter/training configuration and the same number of gradient steps, as described in Appendix A.2.

We first evaluate the cross-entropy loss on the span corruption task, comparing MrT5 models to those trained with Nawrot et al.'s boundary predictor method. For MrT5, we use a controller to target specific sequence length reduction rates and train five models with target deletion ratios of $\delta = 0.15, 0.30, 0.40, 0.60$, and 0.70. Similarly, we train models using Nawrot et al.'s method with priors of $0.85, 0.7, 0.6, 0.4$, and 0.3, which correspond to the same target reduction rates of 15%, 30%, 40%, 60%, and 70%, respectively.

For all pairs of models with similar compression ratios, MrT5 achieves much lower cross-entropy loss compared to the models using Nawrot et al.'s method (Table 9).

E.2 Downstream Tasks

For the downstream tasks, all models are trained with the same hyperparameter/training configurations and for the same number of epochs, as specified in Appendix A.3 of the paper. We adjust the prior for the models that use Nawrot's boundary predictor in order to test different compression rates.

XNLI. For the XNLI task, which requires a multilingual base model, we first train a model with multilingual span corruption data that uses Nawrot et al.'s method before fine-tuning it on XNLI, as we had done for the ByT5 and MrT5 models. We use a prior of 0.5, which will give it the same approximate compression rate as the MrT5 model (50%). We maintain this prior when fine-tuning the model on NLI.

Across all languages, MrT5 outperforms Nawrot et al.'s method (Table 10). We also note that the MrT5 model has more dynamic compression rates for different languages, adapting to the specific information density of a particular language. In Nawrot et al.'s paper, their experiments required setting different explicit priors for different languages, which is a limitation in terms of flexibility when compared to our approach.

Character-level Tasks. For the spelling correction task, we try two settings for Nawrot et al.'s method: one with prior = 0.5, and one with prior = 0.3. MrT5 outperforms Nawrot et al.'s method in terms of both task accuracy and sequence length reduction rates (Table 11).

Language	Accuracy (%)			Seq. Len. Reduction (%)	
	ByT5	MrT5	Nawrot	MrT5	Nawrot
English	76.47	78.84	71.48	52.56	52.02
French	53.90	51.89	47.17	47.58	52.89
Spanish	55.99	53.96	51.17	50.67	53.16
German	45.68	45.94	41.88	47.61	52.80
Greek	52.01	52.75	43.79	54.79	55.63
Bulgarian	56.31	53.62	46.12	47.82	47.81
Russian	55.35	55.97	43.61	47.95	48.57
Turkish	44.86	42.91	40.68	46.72	51.86
Arabic	51.89	47.13	42.26	43.78	56.65
Vietnamese	48.63	47.51	46.08	46.51	51.93
Thai	48.35	44.07	40.23	61.07	48.54
Chinese	50.70	50.44	40.72	24.96	53.76
Hindi	43.87	40.25	39.91	43.56	46.30
Swahili	40.64	38.61	42.38	48.48	52.99
Urdu	45.48	40.88	38.97	38.32	55.78
All Languages	51.34	49.65	45.10	46.82	52.05

Table 10: Per-language XNLI results for ByT5, MrT5, and a model that uses the boundary predictor with pooling method of Nawrot et al. (2023).

Split	Accuracy (%)				Seq. Len. Reduction (%)		
	ByT5	MrT5	Nawrot (prior=0.5)	Nawrot (prior=0.3)	MrT5	Nawrot (prior=0.5)	Nawrot (prior=0.3)
Dependent	37.63	35.05	34.92	33.21	78.77	50.24	70.19
Independent	76.45	74.76	73.30	71.29	78.98	50.20	70.25
All Splits	58.19	56.07	55.24	53.37	78.88	50.22	70.22

Table 11: Spelling Correction with Context task results for ByT5, MrT5, and two models that use the boundary predictor with pooling method of Nawrot et al. (2023).

For the word search task, we try two settings for Nawrot et al.'s method: one with prior = 0.3, and one with prior = 0.7. Even with much lower sequence length reduction rates, Nawrot et al.'s method has very poor accuracy and is vastly outperformed by MrT5 (Table 12).

F Additional Experiments with Large Model Sizes

Here, we show the results for continued pre-training experiments using ByT5 Large as the base model. ByT5 Large has 1.23B parameters, 36 encoder layers, 12 decoder layers, $d_{\text{model}} = 1536$, and $d_{\text{model}} = 3840$. We train for 1500 gradient steps over batches of 2^{20} tokens. All other data and optimization settings match the continued pre-training experiments for ByT5 Small detailed in A.2.

For MrT5, we place the deletion gate at layer $l = 3$, and we use a P-controller to target different deletion rates ($\delta = 0.15, 0.3, 0.4, 0.5, 0.6, 0.7$). We also train a ByT5 baseline model as well as random deletion baseline models with different deletion rates (15%, 30%, 40%, 50%, 60%, 70%).

Split	Accuracy (%)				Seq. Len. Reduction (%)		
	ByT5	MrT5	Nawrot (prior=0.7)	Nawrot (prior=0.3)	MrT5	Nawrot (prior=0.7)	Nawrot (prior=0.3)
OOV	78.05	79.99	59.70	38.82	72.52	20.75	78.37
Paraphrase	83.67	82.06	19.16	3.68	72.27	21.50	78.32
Overlap	77.58	76.22	74.09	54.71	76.14	22.47	75.84
Paraphrase + Overlap	58.46	57.60	14.75	3.14	73.95	21.53	78.20
All Splits	75.37	74.30	38.68	22.45	73.91	21.76	77.51

Table 12: Word Search task results for ByT5, MrT5, and two models based on the boundary predictor method of Nawrot et al. (2023) with different priors.

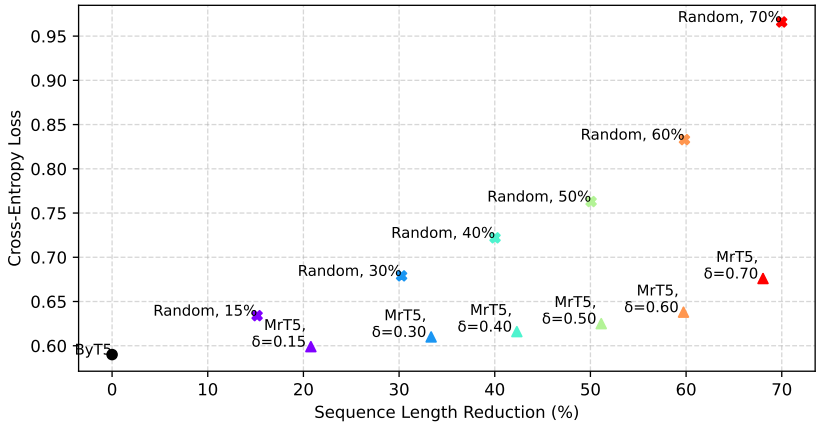


Figure 7: Span corruption cross entropy loss vs. sequence length reduction for each MrT5 Large and large baseline model. MrT5 models consistently have much lower loss than the baselines, and are generally competitive with unmodified ByT5, even where they achieve very large sequence length reductions.

Model	ByT5	MrT5					
		$\delta = 0.15$	$\delta = 0.3$	$\delta = 0.4$	$\delta = 0.5$	$\delta = 0.6$	$\delta = 0.7$
Runtime (ms)	1890.15	1650.61	1388.89	1187.09	1058.60	968.63	817.66
Percent Decrease (%)	–	12.67	26.52	37.20	43.99	48.75	56.73

Table 13: Average inference runtime for a single sequence (in milliseconds) for ByT5 Large and each MrT5 Large model.

With a larger model size, MrT5 consistently outperforms the random baselines and achieves a loss closer to the ByT5 baseline compared to the smaller model experiments (Figure 7). Additionally, the larger MrT5 model demonstrates significantly greater runtime improvements because ByT5 Large includes 36 encoder layers, as opposed to just 12 in ByT5 Small. Placing the gate at layer 3 allows the sequence length to be reduced for a much larger portion of the encoder layers in the large model experiments, resulting in notable runtime gains (Table 13).