

Dynamic Token Merging for Efficient Subword Encoder-Decoder Transformers

Stanford CS224N Custom Project

Nathan Zhou, Chris Gu, Marco Andono Sie
Department of Computer Science
Stanford University
{ natezhou, cgu26, marcoas } @stanford.edu

Abstract

Transformer models achieve strong performance across many NLP tasks, but computational costs scale with input length. Dynamic token merging (DTM) addresses this inefficiency by deleting low-importance tokens in an early encoder layer, allowing later layers to operate on a shorter sequence. In this work, we study DTM in a subword encoder-decoder model by integrating a deletion gate into the encoder of T5Gemma2, evaluating the trade-off between deletion rate, accuracy, and efficiency on natural language inference. We find that subword tokenization is less redundant than byte-level tokenization, so moderate deletion rates reduce encoder FLOPs with limited accuracy loss, while aggressive deletion causes rapid performance degradation. A fixed-rate random deletion baseline performs competitively at low deletion rates, suggesting substantial input redundancy. Finally, cross-lingual evaluation reveals that learned deletion policies vary across languages.

1 Key Information to include

- Mentor: Julie Kallini
- External Collaborators: N/A
- Sharing project: N/A
- Late Days:

Team member	Late days remaining
Nathan Zhou	3
Chris Gu	3
Marco Andono Sie	3
Total pooled	9
Team size	3
Late days applied for the report	3

2 Introduction

Transformer models [1] achieve strong performance across many NLP tasks, but computational costs scale with input sequence length. In standard architectures, every token is processed by every encoder layer, even though many tokens contribute little to the final prediction. This redundancy results in unnecessary computation, particularly for long inputs.

Dynamic token merging (DTM) addresses this inefficiency by allowing the model to remove low-importance tokens during the forward pass. By shortening the sequence at an early layer in the encoder, the model can reduce computation while preserving the most informative tokens. Prior

work demonstrates that deletion can reduce sequence length and model cost while maintaining performance in transformer models. However, existing studies primarily evaluate token deletion in byte-level architectures [2]. It remains unclear whether these methods transfer effectively to subword encoder-decoder models, which are more widely used in modern NLP systems. Subword tokenization may exhibit different redundancy patterns, and deletion policies learned under one tokenization scheme may not generalize to another.

In this work, we investigate DTM in a subword encoder-decoder model. We implement a deletion mechanism in the encoder of T5Gemma2 [3] and evaluate the behavior of our new MrT5Gemma2 model on natural language inference tasks. We analyze the trade-off between deletion rate, model accuracy, and computational cost on MNLI [4], and examine cross-lingual generalization using XNLI [5]. Our results show that moderate deletion reduces encoder computation with limited accuracy loss on MNLI, but deletion policies learned on English do not generalize uniformly across languages.

3 Related Work

Improving the efficiency of transformer models is an active area of research because the growing computational demands of large models increase financial costs, energy consumption, and barriers to accessible deployment.

Some approaches reduce the quadratic complexity of self-attention. Methods such as Longformer [6], BigBird [7], and Reformer [8] introduce sparse or structured attention patterns that enable transformers to process longer sequences with reduced computational cost. Other approaches reduce model size through compression techniques such as knowledge distillation, as demonstrated by DistilBERT [9].

Another approach improves efficiency by reducing the number of tokens processed by later layers of the model. Token pruning removes tokens predicted to be uninformative during intermediate layers. For example, PoWER-BERT [10] bases this deletion on attention-derived importance scores, while TR-BERT [11] learns a dynamic token selection policy. However, most token pruning methods are developed for encoder-only architectures and evaluated primarily on English benchmarks. Similar ideas have been explored and become increasingly common in vision transformers. DynamicViT [12] learns token importance scores to progressively prune tokens during inference, while Token Merging (ToMe) [13] reduces sequence length by merging similar token representations rather than removing them entirely.

Dynamic token merging (DTM) has recently been explored as a new approach in byte-level encoder-decoder transformers. MrT5 [2], derived from ByT5 [14], introduces a learned deletion gate during encoding that predicts token importance and removes uninformative representations, allowing later layers to process shorter sequences. This approach reduces sequence length and computational cost while maintaining downstream task performance. However, because MrT5 operates on byte-level inputs, deletion decisions are made over sequences with different properties than subword-tokenized inputs.

Our work studies DTM in a subword encoder-decoder transformer architecture. We integrate a deletion gate into the encoder of T5Gemma2 [3], a recent state-of-the-art encoder-decoder model from the Google Gemma family [15] that uses SentencePiece tokenization of vocabulary size 262k [16] and UL2-style pretraining [17]. It supports both image and text modalities with a shared embedding space between the encoder and decoder parts of the model; we focus on text-only tasks and freeze the vision tower during training and evaluation. Our work therefore provides the first analysis of dynamic token merging transferred to a modern subword encoder-decoder architecture and evaluates its behavior under multilingual evaluation. Multilinguality is particularly important in this context, as efficiency techniques may unfairly affect languages with different tokenization patterns or linguistic structures.

Finally, we adopt parameter-efficient fine-tuning to adapt the model during training. We use Low-Rank Adaptation (LoRA) [18], which introduces low-rank update matrices that enable large transformer models to be fine-tuned with substantially fewer trainable parameters while keeping the base model weights frozen. This approach reduces training cost while isolating the effect of the deletion mechanism.

4 Approach

Our method follows the dynamic token deletion framework introduced in MrT5 [2], integrating a deletion gate into the T5Gemma2 encoder [3]. Figure 1 shows the architecture, where after the first k encoder layers, a deletion gate shortens the sequence before it is processed by the remaining $26 - k$ encoder layers and passed to the decoder through cross-attention. We study this mechanism in a fine-tuning setting, where a pretrained T5Gemma2 model is adapted to the downstream task while learning the deletion gate.

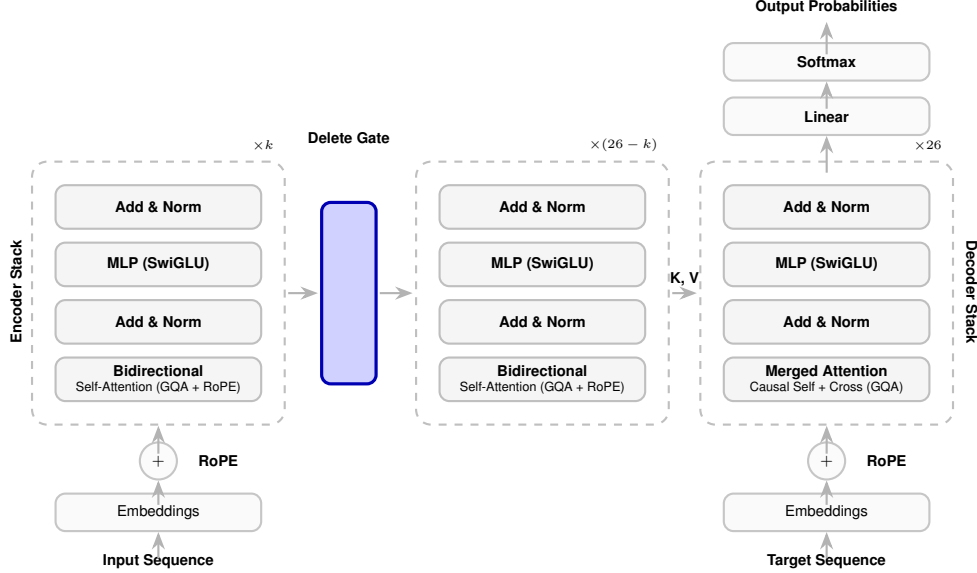


Figure 1: MrT5Gemma2 architecture.

Let $h_i \in \mathbb{R}^d$ denote the hidden representation of token i at encoder layer l . A deletion gate predicts a scalar importance score for each token

$$g_i = c \sigma(-(Wh_i + b)), \quad c < 0$$

where $W \in \mathbb{R}^{1 \times d}$ and b are learned parameters, c is a fixed scaling constant, and $\sigma(\cdot)$ is the sigmoid function. During training, we apply *soft deletion* through attention-logit masking. Let A_{ij} denote the original attention logit between tokens i and j . The masked attention logits are

$$A'_{ij} = A_{ij} + g_j$$

and the attention weights are computed as

$$\alpha_{ij} = \text{softmax}_j(A'_{ij}).$$

Target deletion rates

We adopt the proportional–integral (PI) controller used in MrT5 [2], which dynamically adjusts the gate-regularization coefficient to maintain a target deletion rate δ ; in our experiments, we evaluate target deletion rates $\delta \in \{0.1, 0.2, 0.3\}$. Let r_t denote the observed deletion rate at training step t . The controller updates the regularization coefficient α_t using both the instantaneous error ($\delta - r_t$) and an accumulated error term over previous steps:

$$\alpha_t = \alpha_{t-1} + k_p(\delta - r_t) + k_i \sum_{i=1}^t (\delta - r_i), \quad (1)$$

where k_p and k_i are the proportional and integral gains, respectively. This feedback mechanism encourages the model to match the desired deletion rate during training, resulting in observed deletion rates that closely approximate the specified targets at inference time.

Hard deletion

During inference, we apply *hard deletion* by compacting tokens at the gate layer using a threshold τ . If $S = \{i \mid g_i > \tau\}$ denotes the retained tokens, the encoder sequence becomes $H' = \{h_i \mid i \in S\}$, so later encoder layers and decoder cross-attention operate on the shortened sequence.

Following the performance–efficiency tradeoffs reported in MrT5 [2], the deletion gate is inserted at layer 3 of the encoder. Early encoder layers contextualize tokens before deletion decisions are made, while later layers process the reduced sequence.

LoRA

We adapt the model using LoRA [18], which keeps the pretrained weight matrix fixed and learns only a low-rank update. For a weight matrix $W \in \mathbb{R}^{d \times k}$, the adapted layer is

$$W' = W + \frac{\alpha}{r}BA,$$

where $A \in \mathbb{R}^{r \times k}$ and $B \in \mathbb{R}^{d \times r}$ are trainable matrices with rank $r \ll \min(d, k)$. In our setup, we use $r = 8$ and $\alpha = 16$, so the LoRA branch is scaled by $\alpha/r = 2$. This gives a compact update that adds far fewer trainable parameters than full fine-tuning while still allowing the model to adapt effectively. We also use LoRA dropout $p = 0.05$, which regularizes the low-rank branch during training by randomly dropping a small fraction of its activations.

Softmax1

To stabilize training, we adopt the softmax1 normalization used in MrT5 [2] to prevent degenerate soft deletion behavior (see Appendix). Given logits x_i , softmax1 is defined as

$$\text{softmax}_1(x_i) = \frac{\exp(x_i)}{1 + \sum_j \exp(x_j)}.$$

However, we use regular softmax during evaluation, as we find near-identical hard deletion performance with significantly reduced latency (see Appendix).

We compare our MrT5Gemma2 DTM method against two baselines to contextualize results. The first baseline is standard T5Gemma2 fine-tuning without any token deletion mechanism. The second baseline uses a fixed-rate random deletion strategy in which random token positions are removed to match the same average deletion rate. These baselines allow us to evaluate whether learned token deletion provides benefits beyond simple sequence reduction by preserving task-relevant tokens.

5 Experiments

5.1 Data

Our experiments focus on the downstream task of natural language inference (NLI), where the model predicts the semantic relationship between a premise-hypothesis sentence pair among three types: entailment, contradiction, or neutral. This task provides a standard benchmark for evaluating general-purpose sentence-level language understanding.

We fine-tune on the Multi-Genre Natural Language Inference (MNLI) [4] dataset. In our setup, MNLI provides 392,702 training examples, and we report validation performance on the matched split of 9,815 examples, each example across the entire dataset containing on average 29.6 words. The examples are distributed evenly among the three types of semantic relationships.

To analyze cross-lingual behavior, we additionally evaluate on the Cross-lingual Natural Language Inference (XNLI) [5] dataset. XNLI provides 2490 validation examples for each of its 15 covered languages: Arabic, Bulgarian, German, Greek, English, Spanish, French, Hindi, Russian, Swahili, Thai, Turkish, Urdu, Vietnamese, and Chinese. This benchmark allows us to test whether deletion policies learned during fine-tuning transfer across languages with different linguistic and tokenization characteristics.

For the continued pretraining experiment, we use a span-corruption objective with a HuggingFace streaming multilingual corpus of non-English data from FineWeb-2 [19] and English data from FineWeb [20].

5.2 Evaluation method

We evaluate models using both task performance and efficiency metrics. Task performance is measured using classification accuracy on the NLI task. Accuracy is computed as the proportion of correctly predicted labels over the evaluation set.

To measure the behavior of the deletion mechanism, we report the average deletion rate, defined as the fraction of non-padding input tokens removed by the deletion gate during inference. This metric reflects how aggressively the model prunes tokens and allows us to compare models trained with different target deletion rates.

We run this evaluation on two variants of this mechanism, soft and hard deletion. In the first setting, tokens are greatly attenuated through the soft deletion gate but remain in the sequence. In the second setting, tokens selected by the gate are removed entirely via hard deletion, reducing the effective sequence length processed by the model.

We also measure computational efficiency using two metrics. First, we estimate encoder floating-point operations (FLOPs), which approximate the theoretical computation required for the encoder forward pass after token deletion. We calculate this as

$$\text{FLOP}_{\text{enc}} = B \cdot L \cdot (2s^2d + 8sd^2),$$

where s is the effective encoder sequence length after deletion, d is the hidden size, L is the number of encoder layers, and B is the batch size (set to 1 for per-example reporting). Second, we measure forward-pass latency per example to capture the practical runtime impact of dynamic token deletion, though this metric exhibits slight variance across runs.

5.3 Experimental details

All models are fine-tuned from pretrained T5Gemma2 weights. Training uses the MNLI training split, and evaluation is performed on the validation sets for both MNLI and XNLI. The maximum input length is set to 512 tokens. Early training runs showed substantial performance degradation when training for more than one epoch (see Appendix). Because of this, we train all models for a single epoch. We evaluate several configurations of the deletion mechanism. Models are trained with target deletion rates of 0.1, 0.2, and 0.3. For each configuration, we compare soft deletion during training with hard deletion during inference.

Optimization is performed using the HuggingFace Seq2SeqTrainer with the AdamW optimizer and a learning rate of 1×10^{-4} . A linear learning rate scheduler without warmup is used. Weight decay is set to 0, gradient accumulation is set to 1, and the maximum gradient norm is set to 1. The batch size is 32 during training and 8 during evaluation. Parameter-efficient fine-tuning is implemented using LoRA adapters applied to the attention layers of T5Gemma2, while the base model weights remain frozen. The deletion gate parameters and the LoRA adapters are optimized jointly during fine-tuning. During training, the loss decreases steadily over the course of the single epoch. The deletion rate also stabilizes over time and converges to the target deletion rate due to the controller. Training curves for loss and deletion rate are reported (see Appendix). All experiments are run on a single NVIDIA L4 GPU (4 vCPUs, 16 GB RAM) through GCP Deep Learning VM. Each training epoch took approximately 2 hours. Additional experiments initializing the deletion gate using continued pretraining on span corruption objective before fine-tuning were conducted on a single NVIDIA GH200 GPU, but we found that this continued pretraining does not produce measurable improvements over the standard fine-tuning setup (see Appendix).

5.4 Results

We evaluate the deletion mechanism along three dimensions. First, we examine task performance under varying deletion rates and compare learned deletion against a fixed deletion baseline. Second, we analyze efficiency using profiling metrics, including encoder FLOPs and forward-pass latency. Finally, we evaluate cross-lingual behavior by applying MNLI-trained deletion models to the XNLI benchmark.

Task performance

We evaluate MNLI validation accuracy as a function of the target deletion rate, as shown in Figure 2. Moderate deletion rates preserve much of the task performance. The baseline model without

deletion achieves 0.863 accuracy on MNLI. Under hard deletion, accuracy is approximately 0.80 at a target deletion rate of 0.1 and 0.78 at a target deletion rate of 0.2. When the deletion rate increases to 0.3, performance declines substantially, indicating that more aggressive token removal reduces task accuracy.

Notably, the relatively strong performance at low deletion rates verifies that MNLI inputs contain some redundancy under subword tokenization. At higher deletion rates, our learned soft deletion model outperforms the fixed deletion baseline, indicating more effective preservation of task-relevant tokens and better retention of task-relevant information.

However, learned hard deletion does not consistently outperform the fixed deletion baseline, suggesting that reliably identifying entirely dispensable tokens under subword tokenization remains difficult.

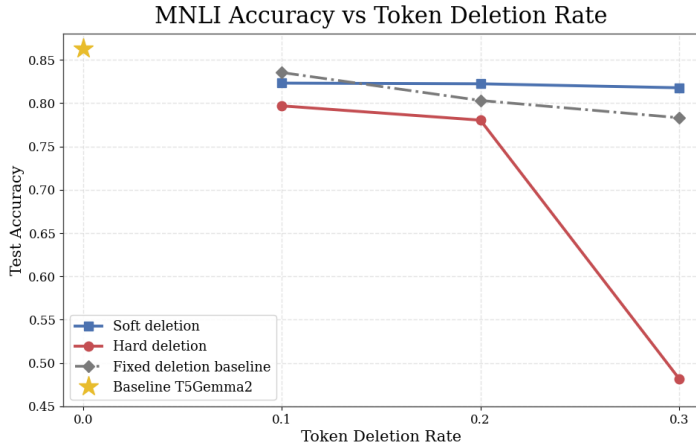


Figure 2: MNLI accuracy vs token deletion rate.

Efficiency metrics

For each configuration, we report achieved deletion rate, estimated encoder FLOPs, and forward-pass latency per example, shown in Table 1.

	Baselines		MrT5Gemma2	
	Baseline	Fixed Deletion	Soft Deletion	Hard Deletion
Accuracy on MNLI	0.863	0.803	0.822	0.780
Deletion Rate (%)	0.0	18.9	19.0	19.0
Encoder FLOPs (B)	36.2	28.4	36.2	28.4
Latency (ms/example)	6.36	6.53	6.37	6.31

Table 1: Efficiency results evaluated with target deletion rate of 0.2 on MNLI.

Forward-pass latency remains largely unchanged across configurations. The fixed deletion baseline exhibits slightly higher latency than the baseline model without deletion. This result suggests that forward-pass latency in our implementation is not determined solely by arithmetic computation. Instead, runtime may be influenced by factors such as GPU overhead, memory movement, and other system-level costs, so theoretical efficiency does not yet translate directly to realized efficiency.

In contrast, the FLOP estimates align with the expected behavior of the architecture. Hard deletion reduces encoder FLOPs from 36.2B to 28.4B, corresponding to a significant 21.5% reduction in encoder computation. This reduction reflects the shorter effective sequence length processed by later encoder layers after tokens are removed by the deletion gate.

Multilingual evaluation

In Figure 3, we compare the cross-lingual behavior of the learned deletion model and fixed deletion baseline, both trained on MNLI with a target deletion rate of 0.1.

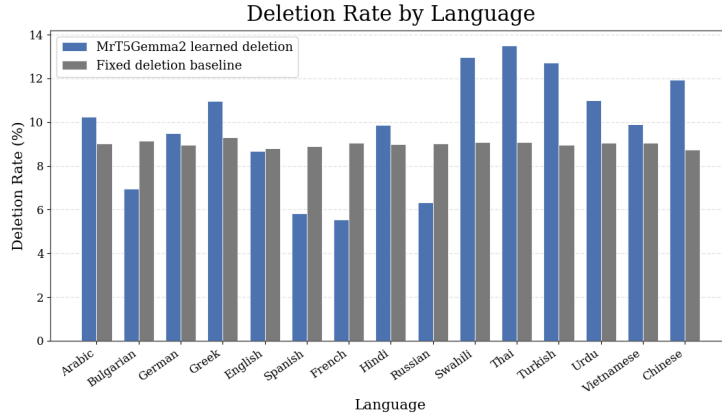


Figure 3: Token deletion rate for different languages.

Evaluation on XNLI shows that deletion behavior varies across languages. The learned deletion model does not realize the same deletion rate across languages, even though it was trained with a target deletion rate on English MNLI. This variation suggests that the learned deletion policy does not transfer uniformly from English to other languages and is sensitive to language-specific tokenization or representation patterns. This also suggests the model could potentially learn language-specific deletion rates. There are some performance discrepancies, with English outperforming other languages, as expected (see Appendix).

6 Analysis

Our results show that dynamic token deletion remains effective in a subword encoder-decoder model. At moderate deletion rates, the model preserves much of the baseline performance while reducing encoder computation. However, performance degrades rapidly once deletion becomes too aggressive, suggesting that although subword token sequences contain redundancy, the margin for safe compression is narrower than in the byte-level setting studied by MrT5.

Table 2 examples how entropy changes between the fixed-deletion baseline and learned soft deletion at 0.2.

Metric	Fixed Deletion (0.2)	Soft Deletion (0.2)
Shannon Entropy (H)	0.7857	0.4896
Gini Impurity	0.4580	0.2717
Brier Score	0.1309	0.2914
Confidence Margin	0.4334	0.6765

Table 2: Comparison of uncertainty-related evaluation metrics for the 0.2 fixed deletion baseline and the learned 0.2 soft deletion model.

The fixed-deletion baseline shows higher Shannon entropy and Gini impurity, while the learned soft deletion model yields substantially lower values, indicating more confident predictions. Despite some variation in calibration reflected in the Brier score, the overall pattern across these metrics suggests sharper prediction distributions under the learned deletion policy.

Qualitative analysis of token-level deletion behavior suggests that the model preferentially removes tokens that contribute little to the core semantic content of the NLI task, such as special markers, formatting tokens, and punctuation (see Appendix). Figure 4 shows two representative examples.

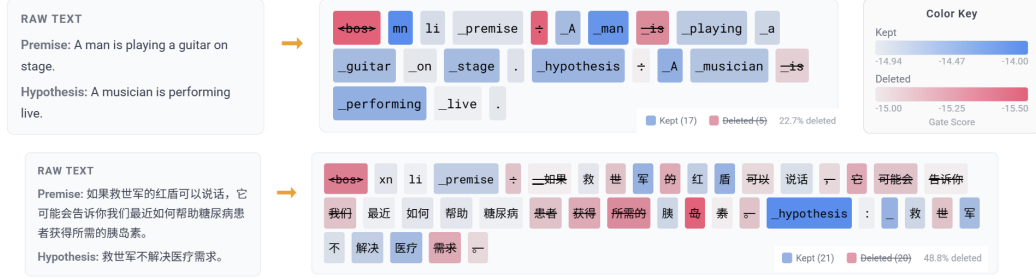


Figure 4: Examples of token deletion patterns. The top example is from English MNLI and the bottom example is from Chinese XNLI. Both are produced by MrT5Gemma2 under hard deletion with target rate 0.2.

In the English example, the model deletes several structural or low-information tokens, including the <bos> token, formatting markers such as *_premise* and *_hypothesis*, punctuation tokens (e.g., “.” and “,”), and auxiliary words such as *_is*. This pattern is consistent with aggregate statistics on the MNLI validation set for MrT5Gemma2 with target deletion rate 0.2. Special marker tokens are deleted on average 57.9% of the time, with the initial <BOS> token deleted in 100% of examples. Punctuation tokens (e.g., “,” and “.”) are deleted 40.1% of the time, and numerical tokens (e.g., “1” and “4”) are deleted 29.2% of the time. These categories appear to play a relatively limited role in determining the semantic relation between the premise and hypothesis.

By contrast, semantically informative tokens in the English example, such as *_man*, *_playing*, *_guitar*, *_stage*, *_musician*, *_performing*, and *_live*, are retained. This tendency also appears in the broader validation set. For example, tokens such as *_government*, *_democratic*, *_friends*, *_researchers*, and *_receive* occur 55, 18, 32, 13, and 48 times, respectively, and are deleted only about 5–7% of the time. Overall, these examples suggest that the learned deletion policy often preserves tokens that are more central to sentence meaning.

A similar pattern is visible in our cross-lingual XNLI evaluation. In the Chinese example, the model deletes several function or modifier tokens, including “可以” (*_can*), “如果” (*_if*), and “它” (*_it*). However, token-level deletion behavior is still variable, and the model sometimes deletes semantically important words, especially in examples where the realized deletion rate substantially exceeds the target. In the example shown in Figure 4, the model deletes 48.8% of tokens, including content-bearing words such as “患者” (*patient/sufferer*) and “告诉你” (*tell you*). This illustrates that although the deletion policy often identifies low-information tokens, it can also over-prune and remove important content in some cases.

Finally, token character length has almost no linear relationship with dataset-wide deletion frequency, with correlation $r = -0.043$. This suggests that the model is not simply favoring shorter or longer tokens for deletion. Additional interpretability analyses are provided in the Appendix.

7 Conclusion

We studied dynamic token deletion in a subword encoder-decoder transformer by integrating an MrT5-style deletion gate into T5Gemma2. Our results show that moderate deletion rates can reduce encoder computation with relatively small accuracy loss, but performance deteriorates quickly once deletion becomes too aggressive. We further find that learned deletion does not consistently outperform a random deletion baseline at low deletion rates, and that gains in theoretical efficiency are larger than gains in measured latency. Cross-lingual evaluation also reveals that deletion behavior varies across languages, suggesting sensitivity to tokenization and language-specific structure. Overall, our findings show that dynamic token deletion remains a promising efficiency mechanism in subword models, but that robust token selection and practical runtime gains remain open challenges. Several directions remain for future work (see Appendix), including additional experimentation and analysis, as well as novel creative directions to further explore the potential of dynamic token merging.

Team contributions

All authors equally contributed to the implementation, experimentation, and writing of this project. Marco and Chris set up and profiled the T5Gemma2 architecture, and Nathan implemented the initial code for DTM in the T5Gemma2 architecture and evaluation based on the MrT5 codebase. All authors contributed to debugging and refining the implementation of the codebase. Nathan and Marco conducted fine-tuning experiments on NVIDIA L4 GPUs, and Chris conducted continued pretraining and fine-tuning experiments on an NVIDIA GH200 GPU.

References

- [1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023. arXiv:1706.03762 [cs].
- [2] Julie Kallini, Shikhar Murty, Christopher D. Manning, Christopher Potts, and Róbert Csordás. MrT5: Dynamic Token Merging for Efficient Byte-level Language Models, April 2025. arXiv:2410.20771 [cs].
- [3] Biao Zhang, Paul Suganthan, Gaël Liu, Ilya Philippov, Sahil Dua, Ben Hora, Kat Black, Gus Martins, Omar Sanseviero, Shreya Pathak, Cassidy Hardin, Francesco Visin, Jiageng Zhang, Kathleen Kenealy, Qin Yin, Xiaodan Song, Olivier Lacombe, Armand Joulin, Tris Warkentin, and Adam Roberts. T5Gemma 2: Seeing, Reading, and Understanding Longer, December 2025. arXiv:2512.14856 [cs].
- [4] Adina Williams, Nikita Nangia, and Samuel R. Bowman. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference, February 2018. arXiv:1704.05426 [cs].
- [5] Alexis Conneau, Guillaume Lample, Ruty Rinott, Adina Williams, Samuel R. Bowman, Holger Schwenk, and Veselin Stoyanov. XNLI: Evaluating Cross-lingual Sentence Representations, September 2018. arXiv:1809.05053 [cs].
- [6] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The Long-Document Transformer, December 2020. arXiv:2004.05150 [cs].
- [7] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for Longer Sequences, January 2021. arXiv:2007.14062 [cs].
- [8] Nikita Kitaev, ukasz Kaiser, and Anselm Levskaya. Reformer: The Efficient Transformer, February 2020. arXiv:2001.04451 [cs].
- [9] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter, March 2020. arXiv:1910.01108 [cs].
- [10] Saurabh Goyal, Anamitra R. Choudhury, Saurabh M. Raje, Venkatesan T. Chakaravarthy, Yogish Sabharwal, and Ashish Verma. PoWER-BERT: Accelerating BERT Inference via Progressive Word-vector Elimination, September 2020. arXiv:2001.08950 [cs].
- [11] Deming Ye, Yankai Lin, Yufei Huang, and Maosong Sun. TR-BERT: Dynamic Token Reduction for Accelerating BERT Inference, May 2021. arXiv:2105.11618 [cs].
- [12] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: Efficient Vision Transformers with Dynamic Token Sparsification, October 2021. arXiv:2106.02034 [cs].
- [13] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token Merging: Your ViT But Faster, March 2023. arXiv:2210.09461 [cs].
- [14] Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models, March 2022. arXiv:2105.13626 [cs].

- [15] Biao Zhang, Fedor Moiseev, Joshua Ainslie, Paul Suganthan, Min Ma, Surya Bhupatiraju, Fede Lebron, Orhan Firat, Armand Joulin, and Zhe Dong. Encoder-Decoder Gemma: Improving the Quality-Efficiency Trade-Off via Adaptation, April 2025. arXiv:2504.06225 [cs].
- [16] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing, August 2018. arXiv:1808.06226 [cs].
- [17] Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Siamak Shakeri, Dara Bahri, Tal Schuster, Huaixiu Steven Zheng, Denny Zhou, Neil Houlsby, and Donald Metzler. UL2: Unifying Language Learning Paradigms, February 2023. arXiv:2205.05131 [cs].
- [18] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models, October 2021. arXiv:2106.09685 [cs].
- [19] Guilherme Penedo, Hynek Kydlíek, Vinko Sabolec, Bettina Messmer, Negar Foroutan, Amir Hossein Kargaran, Colin Raffel, Martin Jaggi, Leandro Von Werra, and Thomas Wolf. FineWeb2: One Pipeline to Scale Them All – Adapting Pre-Training Data Processing to Every Language, June 2025. arXiv:2506.20920 [cs].
- [20] Guilherme Penedo, Hynek Kydlíek, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale, October 2024. arXiv:2406.17557 [cs].

A Appendix

Training Dynamics

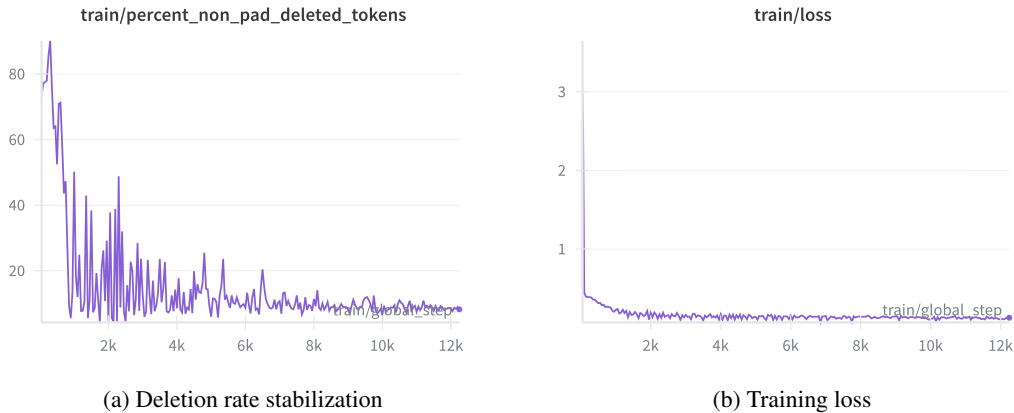


Figure 5: Training dynamics for the learned 0.1 deletion policy.

Continued pretraining

Deletion Type	Number of Epochs Pretrained					
	None	1	2	3	5	10
Hard	0.7804	0.2858	0.4875	0.6561	0.6905	0.3099
Soft	0.8224	0.7348	0.7504	0.8196	0.7419	0.8094

Table 3: MNLI test accuracy for 0.1 deletion models with different degrees of continued pretraining, softmax used for both pretraining and fine-tuning. Continued pretraining does not offer clear performance improvements.

Training epochs

Epochs	Deletion Type	Test Accuracy	Latency (ms)	Avg. Delete Rate (%)
1	Hard	0.7804	6.31	19.03
1	Soft	0.8224	6.37	19.03
2	Hard	0.2094	7.09	18.25
2	Soft	0.7923	7.06	18.25

Table 4: MNLI results for MrT5Gemma2 with target deletion rate 0.2 after 1 and 2 training epochs. Training for 2 epochs does not improve performance. Deletion rates remain similar across epochs, suggesting the degradation is not due to substantially different pruning behavior.

Softmax1 in training

Training	Deletion Type	Avg. Delete Rate (%)	Test Accuracy
Softmax1	Hard	19.03	0.7804
	Soft	19.03	0.8224
Softmax	Hard	84.07	0.0411
	Soft	84.07	0.8637

Table 5: Effect of removing softmax1 and using regular softmax during training. Without this normalization, the model learns a degenerate deletion policy that removes $\sim 84\%$ of tokens, causing severe degradation in MNLI performance under hard deletion.

Softmax1 in evaluation

Target Deletion	Deletion Type	Standard Eval	Softmax1 Eval
0.1	Hard	0.7969	0.7972
0.1	Soft	0.8232	0.8533
0.2	Hard	0.7804	0.7793
0.2	Soft	0.8224	0.8536

Table 6: MNLI accuracy under standard evaluation and evaluation with softmax1. Soft deletion benefits noticeably from softmax1, while hard deletion remains largely unchanged.

Target Deletion	Deletion Type	Standard Eval (ms)	Softmax1 Eval (ms)
0.1	Hard	6.28	7.27
0.1	Soft	6.32	7.01
0.2	Hard	6.31	7.26
0.2	Soft	6.37	6.89

Table 7: Forward-pass latency for standard evaluation and evaluation with softmax1. Applying softmax1 introduces a modest latency overhead.

Full XNLI evaluation results

Language	T5Gemma2		MrT5Gemma2			Fixed Deletion	
	Acc	Lat (ms)	Acc	Lat (ms)	Del (%)	Acc	Lat (ms)
Arabic	0.6847	6.50	0.5743	6.39	10.24	0.6715	6.44
Bulgarian	0.7438	6.46	0.6365	6.42	6.95	0.7124	6.41
German	0.7430	6.44	0.5996	6.48	9.49	0.7365	6.50
Greek	0.7273	6.44	0.6157	6.40	10.95	0.7185	6.37
English	0.8606	6.41	0.7827	6.39	8.68	0.8349	6.38
Spanish	0.7904	6.56	0.6916	6.41	5.83	0.7643	6.52
French	0.7867	6.50	0.6739	6.43	5.56	0.7570	6.40
Hindi	0.6667	6.57	0.5494	6.60	9.87	0.6554	6.34
Russian	0.7201	6.65	0.6000	6.41	6.33	0.7056	6.42
Swahili	0.5149	6.46	0.4462	6.50	12.98	0.5028	6.40
Thai	0.6771	6.45	0.5554	6.37	13.50	0.6711	6.39
Turkish	0.6851	6.46	0.5542	6.41	12.71	0.6486	6.40
Urdu	0.5980	6.53	0.5040	6.50	10.99	0.5892	6.50
Vietnamese	0.7289	6.53	0.6092	6.42	9.89	0.7217	6.39
Chinese	0.7490	6.46	0.6000	6.38	11.93	0.7088	6.37

Table 8: XNLI transfer evaluation across 15 languages reporting accuracy, latency, and average token deletion rates. MrT5Gemma2 uses hard deletion.

Entropy distributions

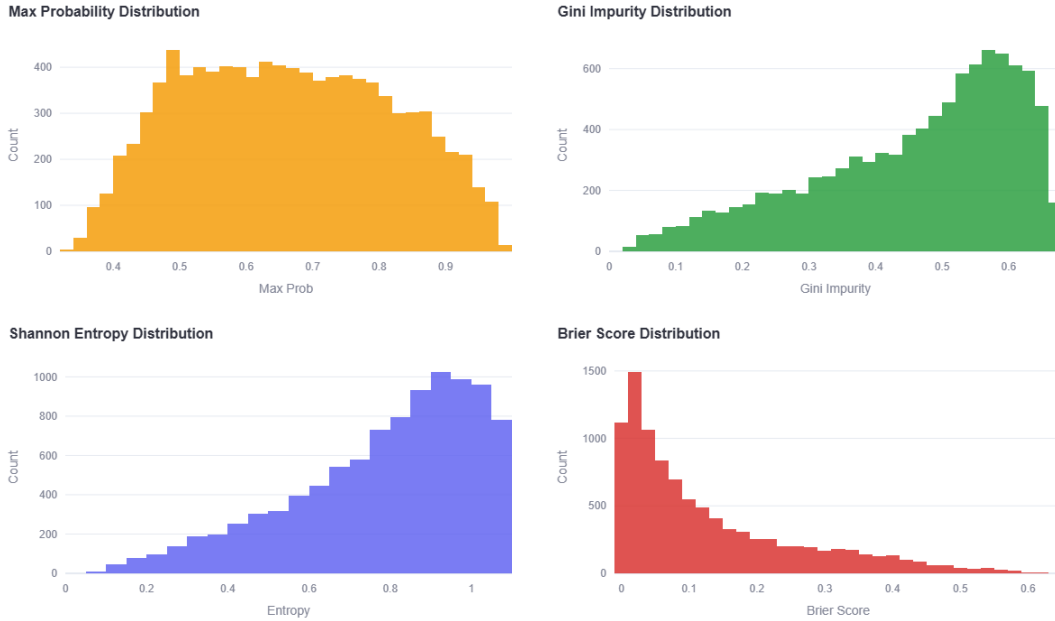


Figure 6: Entropy visualizations for the 0.2 fixed deletion policy.

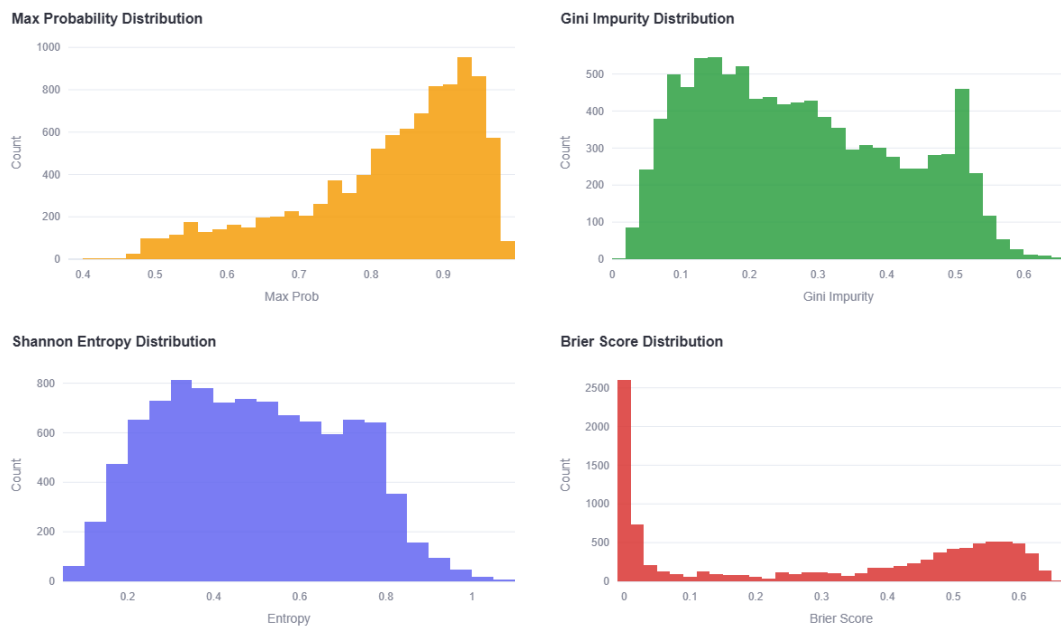


Figure 7: Entropy visualizations for the learned 0.2 soft deletion policy.

Deletion by token type

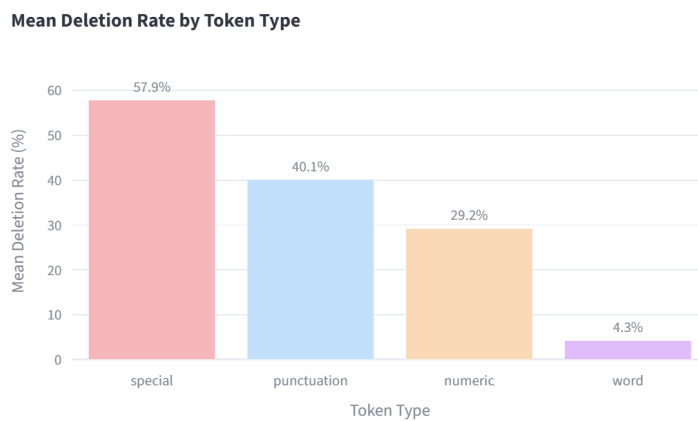


Figure 8: Deletion rates by token type for MrT5Gemma2.

Deletion by character length

Character Length vs Deletion Rate ($r = -0.043$)

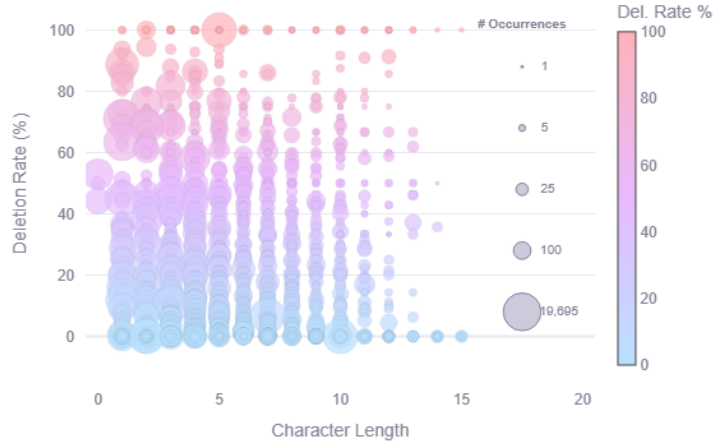


Figure 9: Deletion rates by character length for MrT5Gemma2.

Future work

Training stability appears to be an important factor, as performance degraded when training was extended beyond a single epoch; exploring alternative optimization settings, such as lower learning rates or different controller parameters, may improve stability. It may also be beneficial to place the deletion gate later in the encoder, since subword tokenization may require additional contextualization before reliable deletion decisions can be made. Evaluating the method on tasks with longer inputs, such as question answering benchmarks like TyDiQA-GoldP or long-context retrieval settings, may further clarify the benefits of sequence shortening. Extending the approach to multimodal tasks, such as visual question answering or image-text understanding, may also help evaluate how deletion mechanisms behave in multimodal encoder-decoder architectures. Additional architectural variations, including applying deletion mechanisms to decoder layers or decoder-only models, also warrant investigation. Future work could also explore alternative parameter-efficient fine-tuning strategies or different LoRA configurations to better understand how adaptation methods interact with learned deletion policies. Finally, more detailed profiling of encoder computation and runtime behavior may help better understand when reductions in theoretical computation translate into practical efficiency gains.