MINI-SEQUENCE TRANSFORMER: Optimizing Intermediate Memory for Long Sequences Training

Anonymous Authors[1]

Abstract

We introduce MINI-SEQUENCE TRANSFORMER (MST), a simple and effective methodology for highly efficient and accurate LLM training with extremely long sequences. MST partitions input sequences and iteratively processes mini-sequences to reduce intermediate memory usage. Integrated with activation recomputation, it enables significant memory savings in both forward and backward passes. In experiments with the Llama3-8B model, with MST, we measure no degradation in throughput or convergence even with 12x longer sequences than standard implementations due to our careful memory optimizations. MST is fully general, implementation-agnostic, and requires minimal code changes to integrate with existing LLM training frameworks.

1. Introduction

The development of Transformer (Vaswani et al., 2017) has been a remarkable journey, with each iteration pushing the boundaries of what is possible regarding model size, performance, and efficiency. One of the critical challenges in this journey has been managing the memory requirements of these models, particularly during training. As Transformers have significantly grown in size(Chen et al., 2023) and complexity (Raffel et al., 2020), the memory demand has increased exponentially, necessitating innovative solutions to optimize memory usage while maintaining performance.

A significant milestone in this journey was the introduction of multi-query attention (Shazeer, 2019). This technique dramatically reduced the size of the KV-cache during inference, which uses multiple query heads but single key and value heads. The idea was first adopted in the large-scale training of PaLM (Chowdhery et al., 2023), then adopted and empirically tested in LLaMA (Touvron et al., 2023). As the field progressed, multi-query attention evolved into grouped query attention (GQA) (Ainslie et al., 2023), which relaxes the single key and value head restriction to multiple heads, and each head is coupled with a group of queries. It significantly improves the quality and is adopted by Llama2-70B (Touvron et al., 2023) and Mistral-7B (Jiang et al., 2023).

To further improve model quality, Llama3 (Meta) introduced a tokenizer with a vocabulary of 128K tokens, enabling more efficient language encoding than Llama2's 32K vocabulary. Additionally, Llama3 increased its MLP intermediate size from 11k to 14k. These changes reflect a trend toward more extensive vocabulary and intermediate sizes for better quality. Meanwhile, Llama3 maintains its hidden size of 4k for inference efficiency. This trend is also reflected in the Microsoft development of Phi-3 (Abdin et al., 2024) compared with Phi-2 (Javaheripi et al., 2023).

These advancements have also brought about new memory challenges, particularly in the intermediate value of linear layers of multilayer perception(MLP) and language modeling head (LM-Head). The substantial increase in intermediate variables, which can be nearly ten times larger than the input variables, has severely limited the network's ability to expand sequence length and batch size. This limitation has made it difficult to train large models without restricting sequence length to 8K or relying on gradient accumulation or distributed systems to expand batch size.

Our Approach: Recognizing these challenges, we introduce MINI-SEQUENCE TRANSFORMER (MST), a simple and effective methodology for enabling highly efficient and highly accurate LLM training with extremely long sequence lengths by reducing intermediate memory overhead. MST introduces a per-layer mini-sequence where the input partitions work for each MLP and LM-Head block. MST partitions individual samples along the sequence dimension and iteratively processes each mini-sequence, combining all mini-sequence results to recover full-sequence outputs for these blocks. Our work also adopts activation recomputation (Chen et al., 2016). We find that there is no degradation in throughput or convergence even with sequences up to 12× compared to a standard implementation of Llama3-8B, as

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Under review by the Workshop on Long-Context Foundation Models (LCFM) at ICML 2024. Do not distribute.

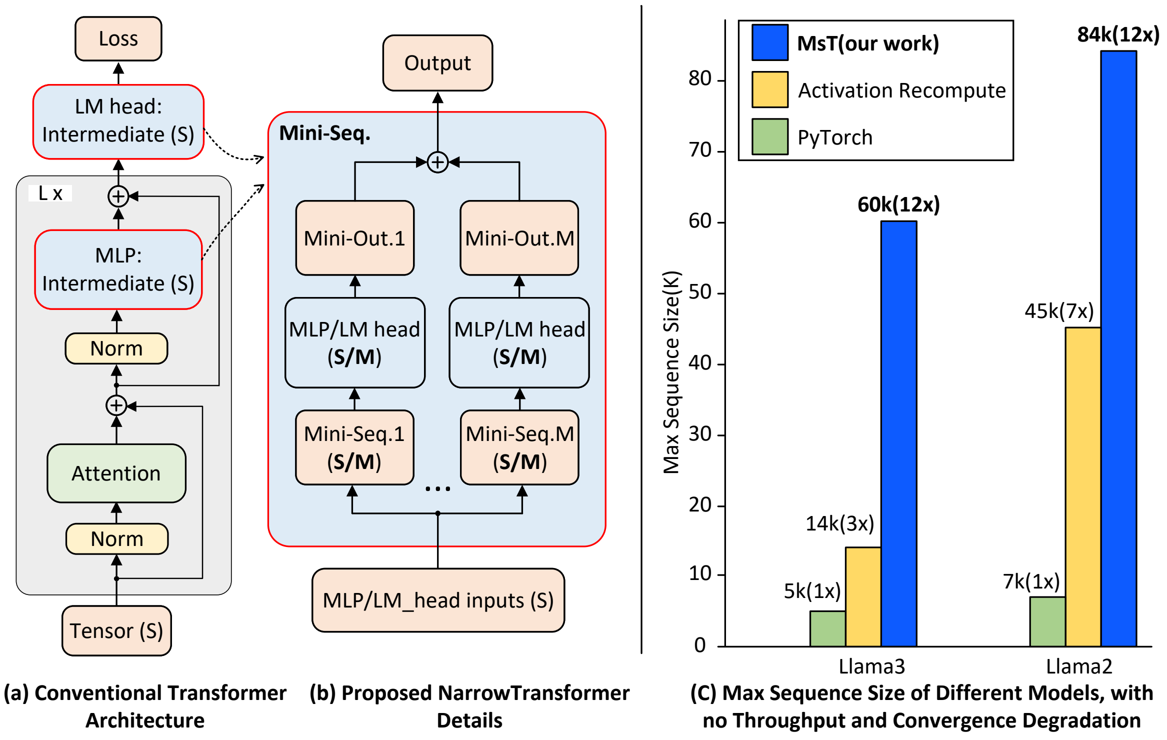

Figure 1: (a) Standard Transformer architecture. MLP's and LM-Head's activation sequence length is annotated with S. (b) MINI-SEQUENCE TRANSFORMER is used to replace MLP blocks and LM-Head block, which splits the input sequence S into M mini-sequences with sequence length S/M, where $M = 2$ on this figure. (c) Max sequence size for training Llama2/Llama3 on A100-80GB GPU, with no degradation of throughput or convergence using our approach.

shown in Figure 1(c).

To summarize, we make the following contributions to advance the state-of-the-art in long-sequence training:

- MST trains 12x longer sequence lengths than existing systems on a single A100 GPU with no degradation in throughput and convergence of training.

- Fully general and implementation agnostic: MST supports most parameter-efficient training as it works independently with attention layers.

2. MINI-SEQUENCE TRANSFORMER (MST)

We present our MINI-SEQUENCE TRANSFORMER (MST) mechanism with the idea of partitioning the input sequence into $M\times$ mini-sequences. We show how to compute the exact transformer block by storing small intermediate matrices for the backward pass. Then, we analyze its IO complexity, showing that our method is throughput-equalized compared to the standard transformer. We focus here on the forward pass for ease of exposition;

2.1. Algorithms: Reducing Intermediate Value With Mini-Sequence Processing

Our idea arises from the observation of large intermediate values from transformer blocks. Given the inputs $X \in \mathbb{R}^{N\times d}$ in HBM, attention blocks and MLP blocks compute the output $O \in \mathbb{R}^{N\times d}$ and loss computation block computes the output $loss \in \mathbb{R}^1$, N equals to sequence size S here. We observe that the intermediate values are always larger than the input X and output O, $loss$. Attention has intermediate values $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}$, which is $(1 + 2 \times d)/G$ larger than input size, where $(1 + 2 \times d/G = 1.5)$ in Llama3 setting. MLP has intermediate value $I_{up}, I_{gate} \in \mathbb{R}^{N\times I}$, where $2 \times I/d = 16$ in Llama3 setting. LM-Head has $logits \in \mathbb{R}^{V\times d}$, where $V/d = 32$ in Llama3 setting.

As flash attention and group query attention have minimized the intermediate value of attention, we put our focus on the MLP block and LM-Head block. Therefore, our implementation of MST is general enough to work with any attention: self-attention (Vaswani et al., 2017), cross-attention (Bahdanau et al., 2014), causal attention (Radford et al., 2018), their sparse counterparts (Child et al., 2019; Zaheer et al., 2020; Roy et al., 2021), and their various optimized kernels such as different versions of FlashAttention (Dao et al., 2022; Dao, 2023). Our implementation adopts FlashAttention2 (Dao, 2023) for the experiments.

Algorithm 1 Mini-Sequence MLP

```
Require: Matrices $X \in \mathbb{R}^{N \times d}$, Weights $W_{gate}, W_{up} \in \mathbb{R}^{d \times I}$, $W_{down}, \in \mathbb{R}^{I \times d}$
1: Partition matrices $X$ into $M$ blocks $X_1, \dots, X_m$ of size $N_m \times d$, where $N_m = N/M$
2: for $1 \leq i \leq M$ do
3:     Compute $I'_i = SiLU((X_i W_{gate}) * (X_i W_{up}))$, $I_i \in \mathbb{R}^{N_m \times I}$
4:     Compute $\mathbf{O}'_i = I_i W_{down}$, $\mathbf{O}_i \in \mathbb{R}^{N_m \times d}$
5: end for
6: Contact $\mathbf{O} = \{\mathbf{O}_i, \dots, \mathbf{O}_m\} \in \mathbb{R}^{N \times d}$
7: Return $\mathbf{O}$.
```

Algorithm 2 Mini-Sequence LM-Head

```
Require: Matrices $X \in \mathbb{R}^{N \times d}$, Labels $L \in \mathbb{R}^N$, Weights $W_{out} \in \mathbb{R}^{d \times V}$
1: Partition matrices $X$ into $M$ blocks $X_1, \dots, X_m$ of size $N_m \times d$, where $N_m = N/M$
2: Partition labels $L$ into $M$ sub-label, $L_1, \dots, L_m$ of size $N_m$, where $N_m = N/M$
3: for $1 \leq i \leq M$ do
4:     Compute $logits_i = X_i W_{out}$, $logits_i \in \mathbb{R}^{N_m \times V}$
5:     Compute if $(i-1) * N_m \leq L_i \leq (i-1) * N_m$, $L_i = L_i$ else $L_i = -100$
6:     Compute $loss_i = crossentropyloss(logits_i, L\_)$
7: end for
8: Compute $loss = \sum_1^M loss_i / M$
9: Return $loss$.
```

Input Partition. We apply the mini-sequence technique to overcome the technical challenge of large intermediate values occupying HBM memory. We describe this in Algorithms 1, and 2, which represent MLP blocks and LM-Head from Llama2 (Touvron et al., 2023) and Llama3(Meta). Their MLP block consists of three linear layer and SiLU function (Ramachandran et al., 2017), and their LM-Head block consists of one linear layer and CrossEntropyLoss function(Rubinstein, 1999). The corresponding backward implementations can be referred from Appendix A for more details. The main idea is partition the input X into mini-sequence X_i as Algorithm 1 line 1 and Algorithm 2 line 1, then compute the output with respect to those mini-sequences. We get the exact same result as standard implementation by contacting all mini-sequence outputs.

Gradient Accumulation. One of our goals is to reduce intermediate values for backward passes. The backward pass typically requires the matrices $X \in \mathbb{R}^{N \times d}$, $I \in \mathbb{R}^{N \times I}$, $logits \in \mathbb{R}^{N \times V}$ to compute the gradients with respect to weights. However, by input partition the $X \in \mathbb{R}^{N_m \times d}$, we can reduce the intermediate value as $I \in \mathbb{R}^{N_m \times I}$, $logits \in \mathbb{R}^{N_m \times V}$ by $M\times$ in the backward pass in HBM. With gradient accumulation for all mini-sequences, all gradients are generated in the same way as standard implementation by introducing more memory loading time.

2.2. Analysis: IO Complexity of MINI-SEQUENCE TRANSFORMER (MST)

We analyze the IO complexity of MST, compared with consistent compute complexity, which can affect its compute-bound or memory-bound performance characteristics.

Theorem 2.1. *Let S be the sequence length, d be the hidden dimension, I be the intermediate size, and V be the voice size. Standard MLP returns $O = act((XW_{gate}) * (X_i W_{up})) * W_{down}$ with $O(SdI)$ FLOPS and* MST *MLP returns $O(SdI/M * M) = O(SdI)$ FLOPS. Standard LM-Loss returns $loss = crossentropyloss(XW, L)$ with $O(SdV + SV)$ FLOPS, and* MST *LM-Loss returns $O((SdV + SV)/M * M) = O(SdV + SV)$ FLOPS.*

Theorem 2.2. *Standard MLP requires $\Theta(Sd + SI + dI)$ HBM accesses, while* MST *(1) requires $\Theta(Sd + SI + dIM)$ HBM accesses. Standard LM-Head requires $\Theta(Sd + SV + dV)$ HBM accesses, while* MST *(2) requires $\Theta(Sd + SV + dVM)$ HBM accesses.*

For Llama3 values of d (4096), I (14336) and V (128256), SI, Sv is many time larger than Sd. For long sequence cases like $S = 100k, S >> d$, the compute complexity and IO complexity are dominated by SI and SV, where MST is close to standard implementation. In this scenario, the intermediate value can be saved by $M\times$ while maintaining the same throughput performance. However, for small sequence cases where $S << d$, the compute complexity and IO complexity are dominated by dI and dV while MST needs dIM and dVM. Therefore, MST would cause throughput downgrades.

3. Experiment

We evaluate the impact of using MINI-SEQUENCE TRANSFORMER (MST) on Llama3(Meta), a state-of-the-art model for many NLP tasks. We validate our claims about scaling sequence length, reporting training time, and memory overhead.

- **Maximun Sequence Length.** MST scales transformer to longer sequences. MST can train Llama3-8B with context length 60k and Llama3-7B with context length 84k on a single A100 GPU. MST outperforms the activation recomputation sequence length by $1.8 - 4\times$ for Llama3-8B, Llama2-7B and Llama2-13b, and it achieves $12 - 20\times$ than standard implementation.
- **Training throughput.** MST maintains the same training throughput compared with standard long-sequence training.

3.1. Longer Sequence Length with MINI-SEQUENCE TRANSFORMER (MST)

Llama3. We train a Llama3-8B(Meta) MST by exploring the sequence length on a single A100 GPU with lossless training strategies, such as activation recomputation and MST. Lossy strategies like sliding windows (Dai et al., 2019) and LoRA (Hu et al., 2021) are not included in our experiments. Table 1 compares our maximum sequence and training time to the PyTorch standard implementation and Huggingface PEFT with activation recomputation. Our implementation trains 4× longer sequence compared with activation recomputation and 12× longer sequence compared with standard implementation.

Table 1: Maximum sequence length of Llama3-8B, running on single A100 80G GPU.

Llama3-8B-hf Implementation	Maximum Sequence Length (K)
vanilla	5
Activation Recomputation	14
MST	60

Llama2. We also train Llama2 models(Radford et al., 2019) with MST, in order to compare the performance of NaT on different models. Table 2 compares our maximum sequence to the PyTorch standard implementation and Huggingface PEFT with activation recomputation. Our implementation trains 1.8× longer sequence compared with activation recomputation and 12× longer sequence compared with standard implementation.

Table 2: Maximum sequence length of Llama2 models, running on single A100 80G GPU.

Model Implementation	Maximum Sequence Length (K)
Vanilla	7
Activation Recomputation	45
MST	84
Vanilla	2
Activation Recomputation	25
MST	40

Combination with gradient accumulation. Gradient Accumulation has been used during training Llama2 and Llama3, which helps them train larger batch sizes given limited available GPU memory. However, in Gradient Accumulation, instead of updating the model parameters after processing each batch of training data, the gradients are accumulated over multiple batches before updating. This means that the memory usage for gradients would occupy the memory used for activation. Therefore, using gradient accumulation during training would significantly constrain the maximum sequence size.

Table 3 summarizes the maximum sequence length with gradient accumulation. The activation recomputation technology can train up to 8K sequences, which happens to be the default setting of Llama3 training. Then MST can train up to 30k sequence length, which is 3.75× longer sequence length compared with activation recomputation plus gradient accumulation. MST can also scale 1.3× sequence for Llama2-7B's training

Table 3: Maximum sequence length training on single A100 80G GPU with gradient accumulation.

Llama3-8B-hf Implementation	Maximum Sequence Length (K)
Activation recomputation	8
MST	30
Activation recomputation	38
MST	52

3.2. Faster Long Sequence Training with MINI-SEQUENCE TRANSFORMER (MST)

We evaluate the training performance of MST on Llama3-8B and Llama2-7B models using a single A100 80G GPU. Table 4 compares the training time per step and TFLOPS achieved by MST with the vanilla PyTorch implementation and activation recomputation technique from Hugging Face PEFT.

Table 4: Training performance using MST on single A100 80G GPU.

Model Implementation	Batch Size	TFLOPS
Llama3-8B-hf vanilla	1	OOM
Llama3-8B-hf activation recomputation	2	3271.42
Llama3-8B-hf MST	2	2825.12
Llama3-8B-hf MST	8	3336.14
Llama2-7B-hf vanilla	1	3290.88
Llama2-7B-hf activation recomputation	8	3703.48
Llama2-7B-hf MST	8	3452.19
Llama2-7B-hf MST	16	3639.93

For Llama3-8B, the vanilla implementation runs out of memory (OOM) with a batch size of 1. Activation recomputation allows training with a batch size of 2, achieving 3271.42 TFLOPS and a training time of 5.01 seconds per step. MST, with the same batch size of 2, achieves a comparable 2825.12 TFLOPS with a slightly longer training time of 5.80 seconds per step. However, MST's memory efficiency allows scaling the batch size to 8, resulting in an improved 3336.14 TFLOPS and a training time of 19.64 seconds per step.

In the case of Llama2-7B, the vanilla implementation can train with a batch size of 1, achieving 3290.88 TFLOPS and a training time of 1.24 seconds per step. Activation recomputation enables a batch size of 8, yielding 3703.48 TFLOPS and a training time of 8.85 seconds per step. MST further increases the batch size to 16, maintaining a similar 3639.93 TFLOPS with a training time of 18.00 seconds per step.

References

Abdin, M., Jacobs, S. A., Awan, A. A., Aneja, J., Awadallah, A., Awadalla, H., Bach, N., Bahree, A., Bakhtiari, A., Behl, H., et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Chen, Z., Ma, M., Li, T., Wang, H., and Li, C. Long sequence time-series forecasting with deep learning: A survey. *Information Fusion*, 97:101819, 2023.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Javaheripi, M., Bubeck, S., Abdin, M., Aneja, J., Bubeck, S., Mendes, C. C. T., Chen, W., Del Giorno, A., Eldan, R., Gopi, S., et al. Phi-2: The surprising power of small language models. *Microsoft Research Blog*, 2023.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Meta. Introducing meta llama 3: The most capable openly available llm to date. https://ai.meta.com/blog/meta-llama-3/.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21 (140):1–67, 2020.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Roy, A., Saffar, M., Vaswani, A., and Grangier, D. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

Rubinstein, R. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1:127–190, 1999.

Shazeer, N. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

A. Algorithm Details

We describe the full details of MINI-SEQUENCE TRANSFORMER (MST) backward pass. Algorithm 3 shows the MLP backward, and Algorithm 4 shows the LM-Head backward.

Algorithm 3 Mini-Sequence MLP Backward

```
Require: Gradients of output $\nabla O \in \mathbb{R}^{N\times d}$, Matrices $X \in \mathbb{R}^{N\times d}$, Activations $I \in \mathbb{R}^{N\times I}$, Weights $W_{gate}, W_{up} \in \mathbb{R}^{d\times I}$, $W_{down} \in \mathbb{R}^{I\times d}$
 1: Partition matrices $\nabla O$ into $M$ blocks $\nabla O_1, \dots, \nabla O_m$ of size $N_m \times d$, where $N_m = N/M$
 2: for $1 \le i \le M$ do
 3:     Compute $\nabla I_i = \nabla O_i W_{down}^T$, $\nabla I_i \in \mathbb{R}^{N_m\times I}$
 4:     Compute $\nabla W_{down} += I_i^T \nabla O_i$
 5:     Compute $\nabla I_i' = \nabla I_i \odot \delta(I_i)$, where $\delta$ is the activation function derivative of SiLU
 6:     Compute $\nabla X'i = \nabla I'i W_{up}^T$, $\nabla X_i'' = \nabla I_i' W_{gate}^T$, $\nabla X_i', \nabla X_i'' \in \mathbb{R}^{N_m\times d}$
 7:     Compute $\nabla W_{up} += X_i^T \nabla I_i'$, $\nabla W_{gate} += X_i^T \nabla I' i$
 8:     Compute $\nabla X_i = \nabla X_i' + \nabla X_i''$, $\nabla X_i \in \mathbb{R}^{N_m\times d}$
 9: end for
10: Concatenate $\nabla X = \nabla X_1, \dots, \nabla X_m \in \mathbb{R}^{N\times d}$
11: Return $\nabla X$, $\nabla W_{gate}$, $\nabla W_{up}$, $\nabla W_{down}$.
```

Algorithm 4 Mini-Sequence LM-Head Backward

```
Require: Loss gradients $\nabla loss \in \mathbb{R}^1$, Logits $\in \mathbb{R}^{N\times V}$, Labels $L \in \mathbb{R}^N$, Weights $W_{out} \in \mathbb{R}^{d\times V}$
 1: Partition labels $L$ into $M$ sub-labels $L_1, \dots, L_m$ of size $\frac{N}{m}$, where $\frac{N}{m} = \frac{N}{M}$
 2: Each mini-sequence forward
 3: for $1 \le i \le M$ do
 4:     Compute $\nabla logits_i = \text{CrossEntropyLossBackward}(Logits_i, L_i)$
 5:     Delete $logits_i$
 6:     Compute $\nabla X_i = \nabla logits_i W_{out}^T$, $\nabla X_i \in \mathbb{R}^{\frac{N}{m}\times d}$
 7:     Compute $\nabla W_{out} += X_i^T \nabla logits_i$
 8: end for
 9: Each mini-sequence backward
10: for $1 \le i \le M$ do
11:     Compute $\nabla X_i = \nabla X_i \odot \nabla loss$
12:     Compute $\nabla W_{out} = \nabla W_{out} \odot \nabla loss$
13: end for
14: Concatenate $\nabla X = \nabla X_1, \dots, \nabla X_m \in \mathbb{R}^{N\times d}$
15: Return $\nabla X$, $\nabla W_{out}$.
```

We now observe about MST backward pass that when computing the gradients of MLP and LM-Head, we do not need to use full input and intermediates data. Instead, we can use $1/M$ reduced data with mini-sequence, significantly reducing the intermediate value memory overhead.