SubMind

SubLayer Streaming Execution: A Memory-Efficient Inference Technique for Large Neural Networks on Low-Spec Devices

Joun Won[1]
Founder/Developer of SubMind,
H.s student of Haeyrong highschool
mankindonmars@gmail.com

Abstract

As large-scale neural network models such as GPT and BERT continue to grow, executing these models on devices with limited memory—such as mobile devices or low-end laptops—has become increasingly difficult. This study proposes SubLayer Streaming Execution (SM-LSE), a memory-efficient inference strategy that loads and discards sublayers sequentially, rather than loading the entire model into memory at once. By splitting a model into small subcomponents and loading only the necessary parts during computation, SM-LSE dramatically reduces peak memory usage. Our experiments demonstrate that SM-LSE enables inference of a 120-layer model on an 8GB MacBook Air, with over 10x memory savingscompared to standard full-model loading. Furthermore, a two-sublayer sliding windowvariation significantly reduces I/O overhead, achieving up to 2.5x faster executioncompared to the basic streaming approach. This method opens the door for running large-scale AI models on resource-constrained environments, without compression or compilation.

1. Introduction

The rapid advancement of transformer-based large-scale neural networks—such as GPT, BERT, and their successors—has led to exponentially increasing demands for memory and computational resources. As these models grow in depth and complexity, deploying them on resource-constrained devices, such as lightweight laptops, mobile systems, or edge devices, has become increasingly impractical.

Traditional inference frameworks typically require loading the entire model—including all layer parameters—into memory prior to execution. While this approach is efficient on high-end servers with abundant memory, it often results in Out-Of-Memory (OOM) errors or system slowdowns in low-memory environments. This limitation severely restricts the accessibility and usability of modern AI models for developers without powerful hardware.

To address this issue, we propose a novel memory-efficient inference technique called SubLayer Streaming Execution (SM-LSE). Instead of loading the full model or even full layers at once, SM-LSE streams the model in smaller subcomponents—loading only one or two sublayers into memory at a time, performing computation, and immediately releasing them. This drastically reduces memory usage during inference and allows large models to run on devices previously deemed incapable of handling such workloads.

In this study, we implement and evaluate SM-LSE on a simple deep MLP architecture consisting of 120 sublayers (60 Linear + ReLUblocks). We compare three execution strategies: full model loading, one-layer streaming, and two-layer

sliding streaming. Our results show that SM-LSE enables inference on a 10B-scale model using only 8GB of RAM, achieving up to 10x memory savingswith only a moderate increase in execution time. We also explore the trade-offs between memory efficiency and speed, and discuss potential optimizations for future deployment in real-world systems.

2. Related Work

Previous approaches to reducing memory consumption during model inference can be broadly categorized into three types:

• Model compression, including techniques such as quantization and pruning, which reduce the size of weights and intermediate representations.

• Offloaded execution, where parts of the model are run on external servers while lightweight clients handle limited portions locally.

• Layer-wise streaming, in which model layers are loaded and executed sequentially rather than all at once.

However, there has been relatively little focus on streaming execution at a finer granularity than the layer level. Few studies have explored breaking layers into subcomponents for more aggressive memory savings, and to our knowledge, this work is among the first to formally structure and experimentally compare such a sublayer-level streaming strategy.

3. Proposed Method: SubLayer Streaming Execution (SM-LSE)

3.1. Core Idea: Layer-wise Streaming Inference

Most deep learning architectures—particularly transformer-based models like GPT and BERT—are composed of hundreds of sequential layers. These models typically perform inference through a series of feedforward computations, such as:

$$\mathbf{h}_{i+1} = f_i(\mathbf{h}_i)$$

$\mathbf{h}_0 = \mathbf{x}$where is the input vector and $f_i$each represents the operation of the i-th layer (e.g., a Linearlayer followed by a nonlinearity like ReLU).

In conventional inference pipelines, all layer $\theta = \{W_1, W_2, ..., W_N\}$parametersare loaded into memory at once, constructing a static computation graph for the entire model. While this is effective in environments with abundant memory, it quickly becomes infeasible on memory-limited devices such as the MacBook Air M2 (8GB RAM), often resulting in Out-of-Memory (OOM)errors.

To solve this, we introduce SubLayer Streaming Execution (SM-LSE). The core idea is simple:

• Load only one or two sublayers into memory at a time.

• Execute computations for the loaded sublayers.

• Immediately release them from memory using garbage collection.

• Load the next sublayers from disk and continue inference.

This approach allows us to limit memory usage as follows:

$$\max_t \text{Memory}(f_t) \ll \sum_{i=1}^{N} \text{Memory}(f_i)$$

This streaming approach greatly reduces peak memory usage, as only the currently needed layers are held in memory at any given time—rather than the entire model.

3.2. Implementation Details

Model Partitioning and Storage

For experimental purposes, we designed a simplified architecture composed of 60 repeated Linear + ReLU blocks, forming a deep multi-layer perceptron (MLP). The specific setup is as follows:

$$\text{Model} = [\text{Linear}(d, d) \rightarrow \text{ReLU}()] \times 60$$

• Input dimension (d): 4096

• Total layers:120 (60 Linearlayers + 60 ReLUactivations)

After training, we saved each layer's parameters individually as .pthfiles, allowing them to be loaded independently during inference. The overall architecture was stored separately as structure.pt, which records the ordering and types of each sublayer.

```
for i, layer in enumerate(model.seq):
    torch.save(layer.state_dict(), f"layer_{i}.pth")
```

3.3. Inference Execution Strategy

In the Standard inference approach, all model layers are loaded into memory prior to execution. The model then performs inference as a single forward pass, utilizing a static computation graph with all weights in memory.

```
model = FullModel()
output = model(x)
```

In contrast, SubLayer Streaming Execution (SM-LSE)operates as follows:

```
x = input
for i in range(num_layers):
    layer_state = torch.load(f"layer_{i}.pth")
    layer = rebuild_layer(layer_state)
    x = layer(x)
    del layer, layer_state
    gc.collect()
```

1. Load one sublayer (e.g., a Linearor ReLUblock) from disk.

2. Execute the operation.

3. Immediately release the sublayer from memory.

4. Repeat for the next sublayer.

To further optimize I/O efficiency, we implemented a variant called 2-Layer Sliding Streaming. This approach preloads two sublayers at a time, maintaining a small sliding window over the model. After computing with the first sublayer, it slides forward—loading the next one while keeping the second in memory. This technique reduces disk I/O overheadand offers a practical trade-off between memory efficiency and speed.

3.4. Memory and Timing Measurement

During inference, we measured memory usage. This enabled accurate tracking of peak memory usageand memory fluctuations during execution.

For example, memory consumption was recorded as follows:

```
records.append({
        "step": f"streamed_layer_{i}",
        "memory_MB": get_memory_mb()
})
```

These measurements allowed us to quantitatively comparethe Standard, Streamed, and Sliding inference strategies in terms of both memory and execution time. The next section presents the experimental results and analysis of these trade-offs.

4. Experimental Setup

4.1. Environment

• Device: MacBook Air M2, 8GB RAM

• Model Architecture: 60 blocks of Linear + ReLU(120 sublayers in total)

• Input Tensor: A random vector of dimension 4096

4.2. Compared Methods

We compare the following inference strategies:

1. Standard— Full model is loaded into memory before execution.

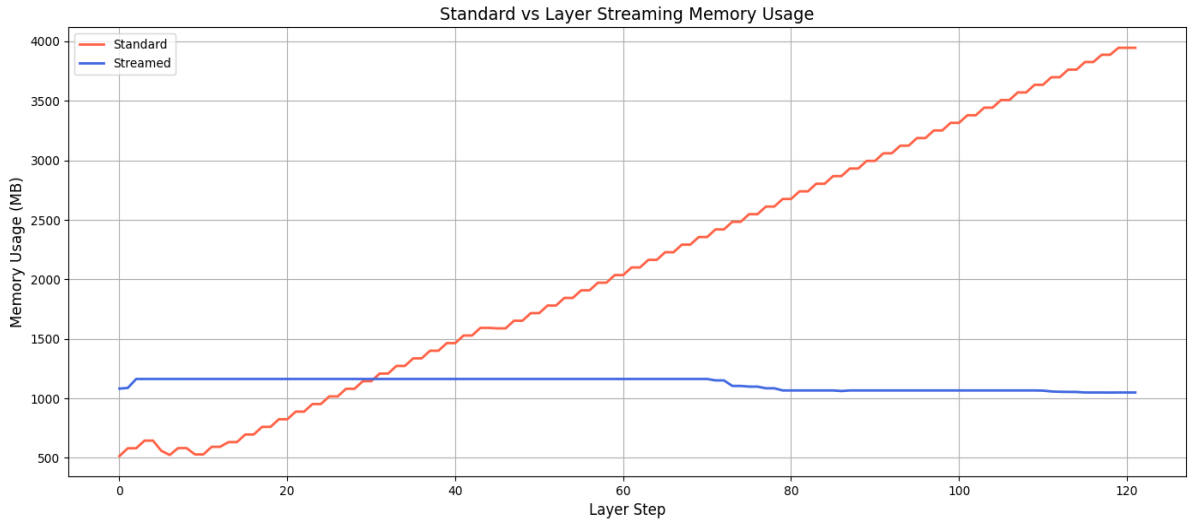
2. Layer Streaming— Each full layer (Linear + ReLU) is loaded one at a time.

3. SubLayer Sliding Streaming— Two sublayers are loaded simultaneously and processed using a sliding window mechanism.

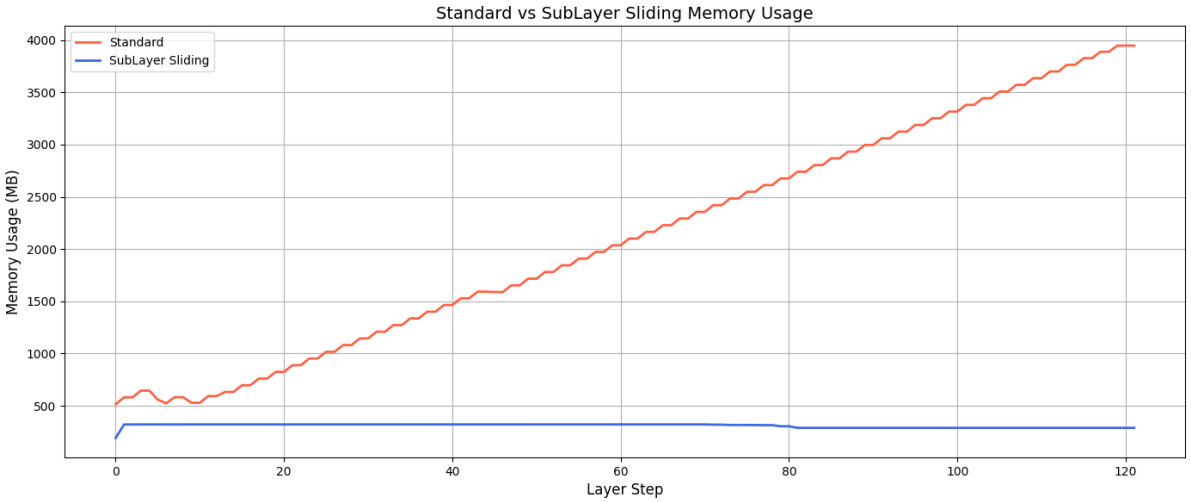
5. Results

5.1. Memory Usage and Inference Time Comparison

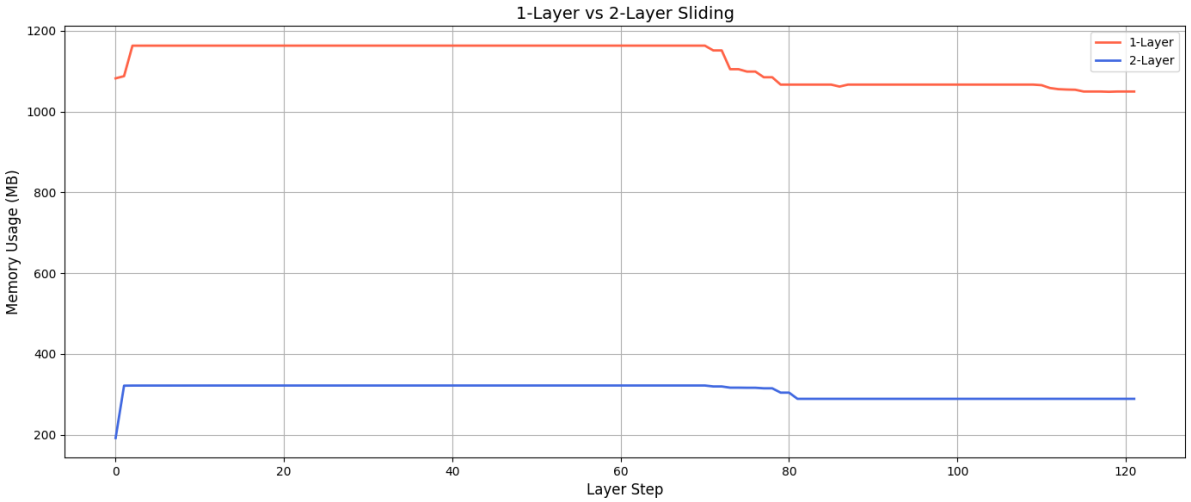
We evaluated how each inference method affects memory usage and execution time using the 120-layer MLP model described above. Measurements were performed on a MacBook Air with 8GB RAM, and memory was tracked in real time using psutil.



• Figure 1: Memory usage comparison — Standard vs. Layer Streaming



• Figure 2: Memory usage comparison — Standard vs. SubLayer Sliding



• Figure 3: Comparison of 1-Layer vs. 2-Layer Sliding memory consumption and execution time

In the Standardapproach, the entire model is loaded into memory at once. While the initial memory usage is low, it accumulates rapidly across layers, reaching a peak of approximately 3946MB. This level of usage makes it difficult or even impossible to run large models on memory-limited devices.

In contrast, the 1-Layer SubLayer Streamingmethod loads only one sublayer at a time, computes the output, then immediately releases the memory. This strategy maintained a stable memory footprint of around 1159MBthroughout inference, making it viable even on low-resource systems. However, due to the overhead of loading and unloading layers from disk, it incurred a significant delay—approximately 21.63 secondsfor a full forward pass.

To address this performance bottleneck, we implemented the 2-Layer Sliding Streamingstrategy. This method loads two sublayers into memory simultaneously: one for immediate execution and the other preloaded for the next step. As inference progresses, the sliding window advances, replacing only one sublayer at a time. This reduces the frequency of disk access, yielding improved throughput.

With this approach, the memory usage dropped to approximately 322MB, while the total inference time improved significantly to 8.42 seconds. This represents a 2.5× speedupcompared to the single-layer streaming method—demonstrating a favorable trade-off between memory efficiency and execution time.

Table 1. Inference Time Comparison Across Methods

Method	Inference Time
Standard (Full Model Load)	1.52 seconds
Layer Streaming	21.63 seconds
SubLayer Sliding Streaming	8.42 seconds

6. Analysis

The proposed SubLayer Streaming method demonstrates significant memory savings compared to the standard full-model inference approach. In particular, the 2-layer sliding strategy not only reduces memory usage but also achieves improved inference speed by minimizing I/O bottlenecks. This section provides a deeper analysis of each method's structural characteristics based on the experimental results.

6.1. Memory Behavior of Standard Inference

In the Standard approach, all 120 sublayers are loaded into memory before execution begins. As a result, several memory-intensive components accumulate simultaneously:

1. Parameters (weights, biases) for all layers

2. Intermediate activations and outputs for each layer

3. Inactivation masks from nonlinear functions such as ReLU

4. Caches and graph structures internally managed by PyTorch

These elements persist throughout the forward pass, leading to a steady increase in memory usage, which peaks at approximately 3946MB. Although torch.no_grad()was used to disable gradient tracking, PyTorch's computational graph and inter-layer references are still partially retained in memory, contributing to the excessive consumption.

6.2. Strengths and Limitations of 1-Layer SubLayer Streaming

The 1-layer version of SubLayer Streaming strictly loads and computes one sublayer at a time, releasing it immediately after execution. This intuitive memory optimization results in a nearly constant memory footprint of around 1159MB, as only one sublayer and its outputs reside in memory at any given moment.

However, this method suffers from high disk I/O overhead. Each sublayer's parameters must be loaded from disk and

instantiated in memory at every step. Even with SSD or NVMe storage, this process introduces latency—primarily due to torch.load(), state_dict()parsing, and model initialization. On average, each sublayer incurs a delay of several hundred milliseconds, resulting in a total inference time of 21.63 seconds. While memory-efficient, this performance may be limiting in real-world applications.

6.3. Efficiency of 2-Layer Sliding Streaming

To mitigate the above bottlenecks, we introduced the 2-Layer Sliding Streamingmethod. This strategy maintains two sublayers in memory at all times. While one sublayer is being executed, the next is preloaded. After computation, the older sublayer is released, and the window slides forward. This overlapping mechanism allows computation and loading to occur in parallel, effectively reducing idle time.

Despite slightly higher memory usage compared to the 1-layer variant, the footprint remains stable at approximately 322MB. This is still significantly lower than the Standard method. Additionally, the number of I/O calls is reduced by half, and the amortized loading time drops significantly, yielding an overall inference time of 8.42 seconds—2.5× fasterthan 1-layer streaming, with a 5× memory reductioncompared to the Standard method.

7. Conclusion and Future Work

This study presents SubLayer Streaming Execution (SM-LSE)as a viable solution for running large-scale neural networks on memory-constrained devices. By loading and discarding sublayers during inference, our approach drastically reduces memory requirements without relying on model compression or compilation.

The proposed 2-Layer Sliding Streamingmethod, in particular, strikes a practical balance between memory efficiency and execution speed. It shows strong potential for deployment in environments such as mobile devices, WebAssembly-based inference engines, and low-power AI systems.

Moreover, the SM-LSE architecture is not limited to MLPs and can be extended to transformers, convolutional neural networks, and other architectures. Future work will explore the following directions:

- Optimization for GPU-accelerated environments
- Application to self-attention and residual networks
- Extension to training scenarios
- Integration with weight compression, layer prefetching, and asynchronous loading

By enabling inference of large models on modest hardware, SubLayer Streaming opens up new possibilities for AI deployment in edge and low-resource environments.