NSMLA: Natively Trainable Sparse Multi-Head Latent Attention

Anonymous ACL submission

Abstract

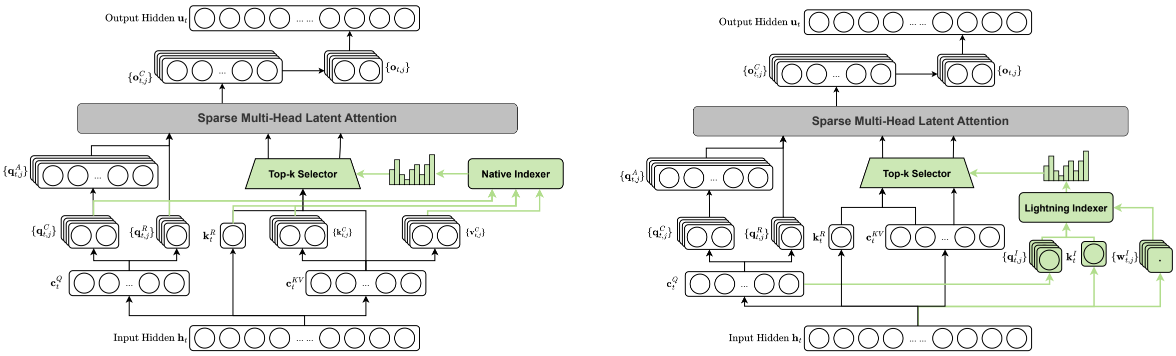
DeepSeek Sparse Attention (DSA) introduces a lightning indexer together with a fine-grained sparse multi-head latent attention (Sparse MLA) mechanism. The indexer efficiently computes relevance scores for each query token and retrieves only the key–value pairs corresponding to the top-k scores. Compared to prior chunk-based sparse attention and sliding-window attention methods, DSA provides greater flexibility and modeling capacity. Despite its effectiveness, DSA has several limitations. Because the lightning indexer is trained by distilling from the main branch as a teacher, DSA only supports continued pre-training and cannot be trained from scratch. Moreover, training DSA is computationally expensive. It requires an explicit dense warm-up stage to align the indexer with dense MLA. During sparse training, the main model is optimized with a cross-entropy loss to adapt from dense to sparse MLA, while a KL-divergence loss is simultaneously applied to continually align the indexer with sparse MLA. To address these limitations, we propose **NSMLA**, which employs a **native indexer** during pre-training. The native indexer introduces no additional parameters, yet enables efficient top-k token selection and more effective optimization of sparse MLA. In a subsequent annealing stage, the native indexer is transformed into a **memory-efficient lightning indexer**, allowing the model to adapt in advance to faster inference-time execution. The native indexer is functionally equivalent to the teacher module in DSA, and therefore more faithfully captures the main model's token preferences. As a result, NSMLA eliminates the need for an expensive dense warm-up stage, requires no KL-divergence loss, and avoids gradient updates to the indexer. The lightning indexer shares a similar architecture with the student module in DSA. Although the conversion introduces a small temporary performance drop, this loss is recovered during the annealing stage. Experiments on DeepSeek-V2-Lite and Youtu-LLM under training-free, continued pre-training, and training-from-scratch settings demonstrate that NSMLA achieves strong performance across all scenarios.

1 Introduction

With rapid progress in complex reasoning, multimodal reasoning, and agentic LLMs, long-context modeling has become increasingly critical. However, at sequence lengths of 128K tokens and beyond, efficient architectures such as linear attention (Gu and Dao, 2024; Katharopoulos et al., 2020) and RNN-based variants (Wu et al., 2025; Bae et al., 2025) still fall short of standard attention in expressiveness. Meanwhile, common sparsification schemes like sliding-window attention (Xiaomi, 2025; Agarwal et al., 2025; AI at Meta, 2025) or block-sparse attention (Lu et al., 2025; Yuan et al., 2025) can be inference-efficient, but their limited flexibility in sparsity patterns often leads to more noticeable quality degradation compared to full attention.

Recently, DeepSeek introduced a fine-grained, token-wise sparse attention mechanism, DeepSeek Sparse Attention (DSA) (Liu et al., 2025). DSA employs a lightning indexer to select the top-2048 most important tokens for each query and retrieves only these important key–value pairs of the selected tokens into KVCache, substantially reducing memory traffic and computation. For instance, on DeepSeek-V3.2 at a context length of 128K, DSA achieves approximately 3.7× and 7.2× cost reductions in prefilling and decoding, respectively, compared to DeepSeek-V3.1 using dense MLA. Relative to sliding-window and block-sparse attention, Sparse MLA offers finer granularity and better quality, while its engineering optimizations also deliver appealing inference speed.

Despite these advantages, DSA still has several limitations:



(a) Pre-training or sparse training stage

(b) Annealing and inference stage

Figure 1: Overall framework of NSMLA, which employs a native indexer during the pre-training stage for sparse MLA training from scratch or during the sparse training stage for continued pre-training, and converts it into a DSA-like lightning indexer during annealing and inference to achieve higher inference efficiency.

Method	Pretraining	Dense Warm-up	Sparse Training	Annealing
DSA	Not supported	Required (KL loss)	Heavy (CE + KL loss)	Heavy (CE + KL loss)
NSMLA	Supported	Not required	Efficient (CE loss only)	Efficient (Low Rank + CE loss only)

Table 1: Training-stage comparison between DSA and NSMLA.

- DSA requires a well-trained dense MLA as a teacher, and is therefore limited to post-pretraining adaptation rather than pretraining Sparse MLA from scratch.

- Early in training, the indexer's predicted top-k tokens differ substantially from those preferred by the main branch, making it difficult for Sparse MLA to produce meaningful attention scores. As a result, DSA first freezes the entire model except the indexer, performs a dense warm-up with a relatively large learning rate (e.g., $1\text{e}{-3}$), and distills the $O(n^2)$ dense MLA signals into the indexer using a KL-divergence loss.

- In the sparse-training stage, DSA typically optimizes two objectives simultaneously:

  - a cross-entropy loss to drive the main branch to adapt to the dense-to-sparse paradigm shift and recover next-token prediction capability;
  - a KL-divergence loss that forces the indexer to continuously track the evolving Sparse MLA, reflecting its preference over token importance.

Because Sparse MLA and the indexer are interdependent and both evolve throughout the sparse training stage, the indexer must continually "chase" a drifting teacher. This instability causes frequent changes in the sparsity pattern, making the optimization target fluctuate and thereby reducing training efficiency and stability.

In addition, at the systems level, because operators such as FlashAttention and online softmax do not directly expose attention scores, DSA training requires an extra pass to compute core attention scores for supervision and alignment, further amplifying the overhead. In the training pipeline of DeepSeek-V3.2, dense warm-up and sparse pretraining together consume roughly 1T tokens of training, and each step incurs additional computation to obtain and align attention scores between the indexer and the main model.

To enable native pretraining of Sparse MLA from scratch and to improve the training efficiency and effectiveness of DSA, we propose **NSMLA**. The key idea is to decouple "Sparse MLA training" from "indexer alignment," splitting the tightly coupled objectives in DSA into two more stable and easier-to-optimize stages.

- **Pre-training or Sparse Training Stage.** We directly compute multi-head attention scores from q and k, and obtain per-head importance weights by applying L_2 pooling over v within each head. We then take a weighted average of the head-wise attention scores to produce an "ideal" token-importance ranking. This stage introduces no additional indexer parameters and does not require KL-divergence-based distillation. Consequently, NSMLA can be trained like a standard model—either from scratch during pretraining or by allowing Sparse MLA to focus solely on adapting from dense attention to sparse attention—without being hindered by inaccurate indexer predictions.

- **Annealing Training Stage.** After Sparse MLA has been well trained, we convert the native indexer into a DSA-like lightning indexer. Specifically, we first absorb the multi-head-style native indexer into a multi-query-style indexer and apply low-rank compression. Unlike DSA, we do not force the indexer to align with Sparse MLA; instead, we freeze the indexer and continue training Sparse MLA, using the annealing stage to implicitly align Sparse MLA with the indexer.

It is important to note that selecting top-k tokens using multi-head attention qkv is ideal for training but unsuitable for inference: doing so would require loading the full KV cache to evaluate token importance, which is counterproductive. In NSMLA, the model is converted during the annealing stage into an inference-friendly lightning indexer (closer to MQA and with a smaller cache footprint), such that computing token importance is significantly cheaper than Sparse MLA. This design allows NSMLA to reuse DeepSeek's Sparse MLA inference framework and realize practical speedups.

In summary, NSMLA offers the following advantages:

- It supports pretraining Sparse MLA-based models from scratch;
- It removes the need for dense warm-up and costly step-wise KL-based alignment distillation, enabling standard end-to-end training;
- Through two-stage decoupling, it substantially improves training stability and efficiency without requiring extra alignment-only steps;
- After training, it can reuse DSA's inference framework with minimal inference-side engineering;
- Our training code is built on tilelang[1], and our inference code is based on vLLM[2]. All code will be open-sourced to facilitate reproduction and further research by the community.

[1] https://github.com/tile-ai/tilelang
[2] https://github.com/vllm-project/vllm

2 Related Works

With attention as their core mechanism, transformers have achieved remarkable success in language modeling. However, the computational complexity of standard attention is $O(N^2)$ with respect to sequence length, which poses a significant scalability bottleneck. To address this issue, several categories of efficient attention mechanisms have been proposed (Zhang et al.), all of which aim to reduce KV cache memory consumption and improve computational efficiency.

Sparse Attention Method. The core objective of sparse attention is to reduce the computational cost and GPU memory consumption of attention computation. Most existing methods are based on token pruning, which reduces the number of tokens participating in matrix multiplications to lower computational overhead and KV cache usage, and can be further categorized into token-level and block-level pruning. Token-level pruning operates on individual tokens and includes approaches such as StreamingLLM (Xiao et al., 2023), H2O (Zhang et al., 2023), and InfLLM (Xiao et al., 2024). In contrast, block-level pruning removes tokens in contiguous blocks; while generally less accurate, this strategy is more efficient and easier to implement in practice. Representative methods in this category include Long-Heads (Lu et al., 2024), RazorAttention (Tang et al., 2024), Ada-KV (Feng et al., 2024), DuoAttention (Xiao et al.), MoBA (Lu et al., 2025), and NSA (Yuan et al., 2025).

Compact Attention Method. Rather than caching full-size keys and values, compact attention methods typically compress the hidden states using techniques such as low-rank approximation. This compression reduces KV cache memory consumption while simultaneously lowering the computational cost of attention. Such methods generally achieve these benefits by reducing either the number of attention heads or the head dimension of the hidden states. Representative examples include MLA (Liu et al., 2024), MHA2MLA (Ji et al., 2025), TransMLA (Meng et al., 2025), and CLOVER (Meng et al., 2024).

3 Preliminary

3.1 DeepSeek Sparse Attention

DeepSeek Sparse Attention (DSA) is a sparse attention method applied in DeepSeek-V3.2 (Liu et al., 2025) and was originally introduced through continued training. The core algorithm of DSA consists of a lightning indexer and a fine-grained token selection mechanism.

In DSA, the lightning indexer computes an index score $I_{t,s}$ between a query token $\mathbf{h}_t \in \mathbb{R}^d$ and all preceding tokens $\mathbf{h}_s \in \mathbb{R}^d$ to determine which tokens should be selected by the query token:

$$I_{t,s} = \sum_{j=1}^{H^I} w_{t,j}^I \cdot \mathrm{ReLU}\left(\mathbf{q}_{t,j}^I \cdot \mathbf{k}_s^I\right), \qquad (1)$$

where H^I denotes the number of indexer heads; $\mathbf{q}_{t,j}^I \in \mathbb{R}^{d^I}$ and $w_{t,j}^I \in \mathbb{R}$ are derived from the query token $\mathbf{h}_t$, and $\mathbf{k}_s^I \in \mathbb{R}^{d^I}$ is derived from the preceding token $\mathbf{h}_s$. The lightning indexer is implemented in FP8.

Given the index scores $\{I_{t,s}\}$ for each query token $\mathbf{h}_t$, the fine-grained token selection mechanism retrieves only the key–value entries $\{\mathbf{c}_s\}$ corresponding to the top-k index scores. The attention output $\mathbf{u}_t$ is then computed by applying the attention mechanism between the query token $\mathbf{h}_t$ and the sparsely selected key–value entries $\{\mathbf{c}_s\}$:

$$\mathbf{u}_t = \mathrm{Attn}(\mathbf{h}_t, \{\mathbf{c}_s \mid I_{t,s} \in \text{Top-}k(I_{t,:})\}). \qquad (2)$$

3.2 Continued Pre-Training

DeepSeek-V3.2 is continued pre-trained based on DeepSeek-V3.1-Terminus. The continued pre-training process consists of two stages: a dense warm-up stage and a sparse training stage. Since the indexer is introduced to assist Sparse MLA in token selection, it brings in newly added parameters that are randomly initialized at the beginning. Directly training with these randomly initialized parameters would disrupt the distribution learned by the original MLA. Therefore, an additional dense warm-up stage is required. During the dense warm-up stage, all model parameters are frozen except for the lightning indexer, and training is performed using dense attention. The training objective is defined as the following KL-divergence loss:

$$\mathcal{L}^I = \sum_t \mathbb{D}_{\mathrm{KL}}(p_{t,:} \,\|\, \mathrm{Softmax}(I_{t,:})). \qquad (3)$$

Once the indexer can sufficiently capture the model's preference of queries over key–value tokens, sparse MLA training can be enabled, where all model parameters are activated and training is conducted using top-k DSA attention. The training objective remains the KL-divergence loss defined above.

$$\mathcal{L}^I = \sum_t \mathbb{D}_{\mathrm{KL}}(p_{t,\mathcal{S}_t} \,\|\, \mathrm{Softmax}(I_{t,\mathcal{S}_t})). \qquad (4)$$

In DSA, the majority of training—approximately 1T tokens—is carried out during this stage. However, in this phase, the model must simultaneously adapt the main branch from dense MLA to sparse MLA and continuously align the indexer with sparse MLA for accurate top-k token selection. These two processes are tightly coupled, and jointly optimizing them under the same loss often leads to training instability. In contrast, NSMLA first employs a native indexer and focuses solely on training sparse MLA. The alignment between the indexer and sparse MLA is deferred to the annealing stage. This decoupled training strategy results in improved training stability and significantly reduced training cost.

4 Native Sparse MLA

4.1 Native Indexer & Lightning Indexer

To achieve token-wise fine-grained attention sparsification, following DSA, we introduce a lightweight indexer module that precomputes attention scores $I_{t,:}$ between each query token $q_{t,j}^I$ and all key tokens $k_{:,j}^I$. Based on these scores, the indexer selects the top-k key–value tokens, which are then passed to the main branch for sparse attention computation. Since the indexer is only responsible for ranking tokens, the softmax operation—which does not affect the relative ordering of attention scores—is unnecessary. Moreover, online softmax requires more complex GEMM operations. Therefore, we replace softmax with a ReLU activation in the indexer to reduce computational overhead. During inference, MLA is absorbed into an MQA formulation, where all attention heads share a single KV cache. Consequently, all heads must adopt the same sparsity pattern. To this end, the attention scores produced by different heads in the indexer are aggregated via a weighted average, with weights denoted by $w_{t,j}^I$. The overall formulation is given by:

$$I_{t,s} = \sum_{j=1}^{H} w_{t,j}^{I} \cdot \mathrm{ReLU}\left(\mathbf{q}_{t,j}^{I} \cdot \mathbf{k}_{s,j}^{I}\right), \tag{5}$$

The representations $q_{t,j}^I$, $k_{s,j}^I$, and $w_{t,j}^I$ take different forms at different stages. During pretraining or sparse training, we directly use the MHA formulation of MLA, which is shown in Equation 6, as the native indexer, introducing no additional parameters. This design enables training from scratch without distillation training indexer parameters, thereby reducing per-step training overhead. More importantly, it ensures tight alignment between the indexer and sparse attention, improving the accuracy of token selection during training.

$$\begin{cases} \mathbf{q}_{t,j}^{I} = \left[\mathbf{q}_{t,j}^{C};\, \mathbf{q}_{t,j}^{R}\right] = \mathbf{q}_{t,j} \in \mathbb{R}^{d}, \\ \mathbf{k}_{s,j}^{I} = \left[W_j^{UK}\mathbf{c}_s^{KV};\, \mathbf{k}_s^{R}\right] \in \mathbb{R}^{d}, \\ w_{t,j}^{I} = \|W_j^{UV}\mathbf{c}_s^{KV}\| \in \mathbb{R}, \end{cases} \tag{6}$$

Because pretraining is compute-intensive, queries and keys are expanded into a multi-head representation with head dimension $d = 192$ (128 dimensions for the NoPE component and 64 dimensions for the RoPE component). As the original model does not include an explicit mechanism to measure head importance, we apply L2 pooling across heads on the value representations, which naturally serves as a proxy for per-head token importance.

In practice, we design a specialized kernel for the multi-head indexer that supports variable-length sequences. Each query in the batch is partitioned into query blocks, and each thread block processes one query block against all keys within the corresponding sequence. Within a thread block, key blocks are traversed sequentially in a pipelined manner. For each key block, the index score between the query block and the key block is computed serially across heads according to Equation 5. After the index score of each head is computed, it is accumulated into a running total to form the final index score for the query–key block pair.

After Sparse MLA training is completed, the multi-head indexer becomes impractical for inference due to its high memory access cost and thus fails to provide inference speedups. To address this, we absorb the indexer into a multi-query formulation:

$$\begin{cases} \mathbf{q}_{t,j}^{I} = \left[W_j^{UK^\top}\mathbf{q}_{t,j}^{C};\, \mathbf{q}_{t,j}^{R}\right] \in \mathbb{R}^{d^C}, \\ \mathbf{k}_{s}^{I} = \left[\mathbf{c}_s^{KV};\, \mathbf{k}_s^{R}\right] \in \mathbb{R}^{d^C}, \end{cases} \tag{7}$$

Specifically, the key up-projection matrices $W_i^{UK^\top}$ are folded into the corresponding query heads. This transformation is mathematically equivalent and preserves the functionality of the indexer. However, the resulting head dimension increases to $d^C = 576$, which still incurs significant computational and memory overhead. To further reduce cost, we apply low-rank compression:

$$\begin{cases} \mathbf{q}_{t,j}^{I} = \left[R_1^\top W_j^{UK^\top}\mathbf{q}_{t,j}^{C};\, \mathbf{q}_{t,j}^{R}\right] \in \mathbb{R}^{d^I}, \\ \mathbf{k}_{s}^{I} = \left[R_1\mathbf{c}_s^{KV};\, \mathbf{k}_s^{R}\right] \in \mathbb{R}^{d^I}, \\ w_{t,j}^{I} = \|R_2 W_j^{UV}\mathbf{c}_s^{KV}\| \in \mathbb{R}, \end{cases} \tag{8}$$

Concretely, we feed a small calibration dataset through the model and perform PCA on the key–value outputs, obtaining projection matrices $R_1 \in \mathbb{R}^{512\times 64}$ and $R_2 \in \mathbb{R}^{512\times 1}$, which are used to reduce the head dimension of the indexer to $d^I = 128$.

Since orthogonal transformations preserve the L2 norm, even aggressive compression of the value representations retains a one-dimensional feature that effectively captures the relative importance of different heads. Furthermore, because only the relative magnitude of attention scores matters, the weight projection can bypass RMSNorm and directly compute w_t^I from h_t using a linear layer, similar to DSA. Although this transformation is inherently lossy, the induced error is minimal because precise attention values are not required—only relative token importance. As demonstrated in our experiments, the resulting degradation is small. To further mitigate this loss, we introduce an annealing stage, during which a small amount of additional training is performed to recover performance.

4.2 Sparse Multi-Head Latent Attention

The design of Sparse MLA follows that of DSA. Given the index scores $\{I_{t,s}\}$ for each query token $\mathbf{q}_t$, Sparse MLA retrieves only the key–value entries corresponding to the top-k index scores. The attention output for head j is then computed as

$$\mathbf{u}_{t,j} = \sum_{s\in\mathcal{S}_t} \mathrm{softmax}\left(\frac{\mathbf{q}_{t,j}^\top \mathbf{k}_s}{\sqrt{d}}\right)\mathbf{v}_s. \tag{9}$$

Attention	Top-k	ARC-E	ARC-C	HS	OBQA	PIQA	WG
MLA	–	0.8342	0.5640	0.6327	0.396	0.8128	0.7522
	512	0.8342	0.5623	0.6321	0.398	0.8107	0.7459
NSMLA	256	0.8342	0.5648	0.6283	0.398	0.8128	0.7466
	128	0.8350	0.5708	0.6233	0.396	0.8096	0.7490

Attention	Top-k	Code	Few-shot	MDQA	SDQA	Summary
MLA	–	0.3951	0.5429	0.2915	0.3883	0.2305
	2048	0.3770	0.4972	0.2579	0.3582	0.2226
NSMLA	1024	0.3803	0.4638	0.2523	0.3414	0.2198
	512	0.3656	0.4339	0.2207	0.3125	0.2175

Table 2: Comparison between dense MLA and our native sparse MLA, where NSMLA is directly converted from MLA **without any additional training**. The upper part shows commonsense reasoning results (5-shot, 4K context), and the lower part shows LongBench results (zero-shot, 128K context). Top-k indicates the maximum number of tokens selected per query.

where $\mathcal{S}_t = \text{top-}k(I_{t,:})$, the selected entries are defined as $\mathbf{q}_s = [\mathbf{c}_s, \mathbf{k}_s^{\text{rope}}]$ and $\mathbf{k}_s = \mathbf{c}_s$

In the following, we explain why the proposed NSMLA adopts the Multi-Query Attention (MQA) formulation rather than the standard Multi-Head Attention (MHA) as the main branch.

The key motivation arises from the execution model of modern Tensor Cores, whose underlying computation is based on fixed-shape matrix multiply–accumulate (MMA) micro-tiles, such as `m16n16k16`, `m16n8k16`, and `m8n8k4`. These MMA instructions require the operands to be organized as

$$\mathbf{A} \in \mathbb{R}^{m \times k}, \quad \mathbf{B} \in \mathbb{R}^{k \times n},$$

so that the computation can be decomposed and mapped onto hardware-supported tile shapes for efficient execution.

Under the fine-gained sparse attention setting, the query tensor $\mathbf{q} \in \mathbb{R}^{\mathcal{S} \times \mathcal{H} \times \mathcal{D}}$ exhibits token-dependent sparsity patterns, as each token typically selects a different set of top-k key/value entries. For efficient scheduling and memory access, the implementation commonly partitions the computation by token, assigning each token to a separate computation block. Consequently, each block contains all heads of a single token,

$$\mathbf{q}_t \in \mathbb{R}^{\mathcal{H} \times \mathcal{D}}.$$

For each token, the corresponding top-k key/value block $\mathbf{K}_b \in \mathbb{R}^{\mathcal{B} \times \mathcal{D}}$ is loaded, and attention scores for multiple heads are computed simultaneously. To form matrix multiplications compatible with MMA execution, a single key–value group must therefore be shared across $\mathcal{H}$ query heads.

If MHA is used, where each query head is paired with a distinct key/value head, the computation within each block degenerates into

$$\mathbf{q}_{t,j} \in \mathbb{R}^{1 \times \mathcal{D}}, \quad \mathbf{K}_{b,j} \in \mathbb{R}^{\mathcal{B} \times \mathcal{D}},$$

which corresponds to a matrix–vector multiplication (GEMV). Due to the low arithmetic intensity of GEMV and its incompatibility with fixed-shape MMA tiles, this execution path cannot efficiently utilize Tensor Core acceleration.

Therefore, similar to DSA, Sparse MLA is implemented in the MQA form. By allowing multiple query heads to attend to a shared key–value group, the computation can be restructured into higher-rank GEMM operations that better match MMA tile requirements, thereby enabling efficient Tensor Core execution.

5 Experiment

In this section, we conduct three groups of experiments to validate the effectiveness of NSMLA. First, we directly convert the MLA in DeepSeek-V2-Lite-Chat (Liu et al., 2024) into NSMLA and evaluate its performance without any additional training. Second, we evaluate the performance of NSMLA under continued pre-training: we convert Youtu-LLM-2B(Lab, 2025) into NSMLA and train the converted model on data with a maximum context length of 128K, then observe its performance. Finally, we train NSMLA from scratch to verify its performance under native pre-training.

5.1 Training-Free Conversion

The indexer of NSMLA reuses the parameters of the main branch during the pre-training or sparse

Attention	Tokens	Code	Few-shot	MDQA	SDQA	Summary
MLA	5T	0.1824	0.4674	0.1891	0.2417	0.1822
NSMLA	0B	0.1187	0.1468	0.0697	0.1768	0.1247
	80B	0.2033	0.4035	0.1410	0.2075	0.1437

Table 3: Comparison between dense MLA and our native sparse MLA, where NSMLA is first converted from dense MLA to sparse MLA and then **continually pre-trained**. The model is trained on long-context data to extend its long-context capability, and its performance is evaluated using LongBench results (zero-shot, 128K context). *Tokens* denotes the amount of training data used to obtain the long-context model; *Tokens* = 0B indicates the performance of the converted model without any additional training.

training stages, and applies PCA to the main branch during the annealing and inference stages to obtain the lightning indexer. Compared with DSA, which randomly initializes the indexer, our indexer has a much higher similarity to the main branch. To examine the loss incurred when directly converting Dense MLA to Sparse MLA, we conduct experiments using DeepSeek-V2-Lite-Chat. Following the conversion method described in Section 4, we first absorb MLA into an MQA form and then apply low-rank compression. Without any further training, we directly evaluate the converted model.

In Table 2, we report results on commonsense reasoning benchmarks in the 5-shot setting with a maximum context length of 4K, including ARC-E (ARC-Easy), ARC-C (ARC-Challenge) (Clark et al., 2018), HS (HellaSwag) (Zellers et al., 2019), OBQA (OpenBookQA) (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), and WG (WinoGrande) (Sakaguchi et al., 2021). We also evaluate on LongBench tasks (Bai et al., 2024) in the zero-shot setting with a maximum context length of 128K, including Code (Code Completion), Few-shot (Few-shot Learning), MDQA (Multi-Document Question Answering), SDQA (Single-Document Question Answering), and Summary (Summarization). The system performance is evaluated under different context lengths.

From the table, we observe that after converting MLA to NSMLA, when the KV cache compression ratio reaches 96.875% (128/4096), NSMLA can still maintain strong performance on commonsense tasks, and even outperforms the dense model on ARC-Easy and ARC-Challenge, while matching the dense model on OpenBookQA. On the more challenging LongBench tasks, when the compression ratio reaches 98.4375% (2048/131072), the model is still able to produce meaningful outputs.

It is worth emphasizing that this conversion loss consists of two components: one from converting Dense MLA to Sparse MLA, and the other from converting the native indexer to the lightning indexer. Subsequent experiments on continued pre-training and training from scratch demonstrate that the loss introduced by converting the native indexer to the lightning indexer is much smaller than that incurred by converting Dense MLA to Sparse MLA. Due to resource constraints, we were unable to directly convert DeepSeek-V3.2 to NSMLA. Since DeepSeek-V3.2 has already completed training from Dense MLA to Sparse MLA, we expect that converting it to NSMLA would yield very promising results, which we leave as future work.

5.2 Continued Pre-Training

Similar to DSA, NSMLA can also be further trained on top of a dense MLA model to obtain a sparse MLA model. Due to resource constraints, in this section we choose Youtu-LLM-2B-Base, a dense MLA model with only 2B parameters, for continued pre-training. Considering the training cost, we directly convert the native indexer into the lightning indexer and transform dense MLA into sparse MLA, followed by annealing training.

As shown in Table 3, because this model has a relatively small number of parameters, the performance degradation after conversion is relatively large. We then continue training the converted model using the 128K-length extended data that is homogeneous to Youtu-LLM-2B-Base. Unlike DSA, which requires a warm-up phase and indexer updates, NSMLA does not incur these overheads and can be trained as a sparse MLA model in the same manner as a standard model. The performance after training is also reported in Table 3.

From Table 3, we can see that with only a small amount of training (compared to DSA, which uses approximately 1T tokens in total for dense-to-sparse training), NSMLA is able to rapidly recover performance. Notably, its code completion capability even surpasses that of the dense MLA model by 2.1%. We believe that with increased training data,

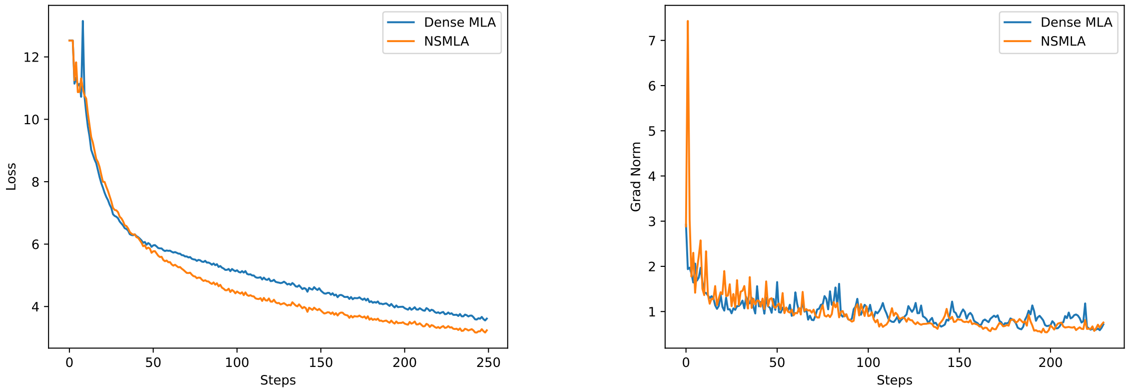


Figure 2: Comparison of the loss and gradient norm when fitting data during training from scratch with NSMLA and dense MLA.

NSMLA can achieve even better performance.

5.3 Train from Scratch

One of the major advantages of NSMLA is its native support for training from scratch. In this subsection, we adopt the architecture of the Youtu-LLM-2B-Base model, initialize all parameters from scratch, and pre-train the model using both dense MLA and our NSMLA. Due to resource constraints, we conduct pre-training only on data with a context length of 8K, and set the top-k of NSMLA to 128. The training loss and gradient norm during training are shown in Figure 2.

As illustrated in the figure, NSMLA exhibits a convergence speed that is even better than that of dense MLA. This experiment demonstrates the strong potential of NSMLA. In future work, we plan to scale up the training data and conduct more extensive training to further investigate the training-from-scratch capabilities of NSMLA.

6 Conclusion

To address the limitations of DSA, which does not support training from scratch and incurs high training costs, this paper proposes a native sparse MLA (NSMLA) that enables efficient pre-training, sparse training, annealing, and inference. The core innovations lie in the newly introduced native indexer and the conversion method from the native indexer to the lightning indexer. We implement the training components using TileLang kernels and reuse the vLLM implementation from DSA for inference. The effectiveness of our approach is validated under three settings: training-free conversion, continued pre-training, and training from scratch. We believe that this native fine-grained sparse attention training paradigm can significantly improve both model training and inference efficiency.

Limitations

Due to limitations in computational resources and time, this work does not validate the proposed method on larger-scale models or with substantially more training data. We plan to supplement these experiments in future work to better understand the upper performance limits of NSMLA. Since the MLA implementations used in DeepSeek-V2-Lite and Youtu-LLM-2B have only 16 attention heads, the room for further compression of the indexer is limited; therefore, we did not conduct speed benchmarks. Experiments on larger models would allow us to address this issue and demonstrate practical speed improvements. In addition, this paper does not compare NSMLA with a broader range of sparse attention methods; incorporating such comparisons would further strengthen the credibility of the work.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

AI at Meta. 2025. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. https://ai.meta.com/blog/llama-4-multimodal-intelligence/. AI at Meta Blog. Accessed: 2025-12-18.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and 1 others. 2025. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 3119–3137.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. 2024. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *arXiv preprint arXiv:2407.11550*.

Albert Gu and Tri Dao. 2024. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*.

Tao Ji, Bin Guo, Yuanbin Wu, Qipeng Guo, Lixing Shen, Zhan Chen, Xipeng Qiu, Qi Zhang, and Tao Gui. 2025. Towards economical inference: Enabling deepseek's multi-head latent attention in any transformer-based llms. *arXiv preprint arXiv:2502.14837*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR.

Tencent Youtu Lab. 2025. Youtu-llm: Unlocking the native agentic potential for lightweight large language models.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, and 1 others. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, and 1 others. 2025. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*.

Yi Lu, Xin Zhou, Wei He, Jun Zhao, Tao Ji, Tao Gui, Qi Zhang, and Xuan-Jing Huang. 2024. Longheads: Multi-head attention is secretly a long context processor. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7136–7148.

Fanxu Meng, Pingzhi Tang, Xiaojuan Tang, Zengwei Yao, Xing Sun, and Muhan Zhang. 2025. Transmla: Multi-head latent attention is all you need. *arXiv preprint arXiv:2502.07864*.

Fanxu Meng, Pingzhi Tang, Muhan Zhang, and 1 others. 2024. Clover: Cross-layer orthogonal vectors pruning and fine-tuning. *arXiv preprint arXiv:2411.17426*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. 2024. Razorattention: Efficient kv cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*.

Bohong Wu, Mengzhao Chen, Xiang Luo, Shen Yan, Qifan Yu, Fan Xia, Tianqi Zhang, Hongrui Zhan, Zheng Zhong, Xun Zhou, and 1 others. 2025. Parallel loop transformer for efficient test-time computation scaling. *arXiv preprint arXiv:2510.24824*.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2024. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. *Advances in Neural Information Processing Systems*, 37:119638–119661.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Shang Yang, Haotian Tang, Yao Fu, Song Han, and 1 others. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. In *The Thirteenth International Conference on Learning Representations*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

LLM-Core Xiaomi. 2025. Mimo-v2-flash technical report.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, and 1 others. 2025. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Jintao Zhang, Rundong Su, Chunyu Liu, Jia Wei, Ziteng Wang, Pengle Zhang, Haoxu Wang, Huiqiang Jiang, Haofeng Huang, Chendong Xiang, and 1 others. A survey of efficient attention methods: Hardware-efficient, sparse, compact, and linear attention.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.