SparseSSM: EFFICIENT SELECTIVE STRUCTURED STATE SPACE MODELS CAN BE PRUNED IN ONE-SHOT

Anonymous authors
Paper under double-blind review

ABSTRACT

State-space language models such as Mamba match Transformer quality while permitting linear complexity inference, yet still comprise billions of parameters that hinder deployment. Existing one-shot pruning methods are tailored to attention blocks and fail to account for the time-shared and discretized state-transition matrix at the heart of the selective state-space module (SSM). In this paper, we introduce *SparseSSM*, the first training-free pruning framework that extends the classic optimal brain surgeon (OBS) framework to state space architectures. Our layer-wise algorithm **(i)** derives an approximate second-order saliency score that aggregates Hessian-trace information across time steps, **(ii)** incorporates a component sensitivity analysis to guide feed-forward network (FFN) pruning, which also sheds light on where redundancy resides in mamba architecture, **(iii)** can be easily extended to semi-structured and structured sparsity. Empirically, we prune 50% of SSM weights without fine-tuning and observe no zero-shot accuracy loss, achieving the current state-of-the-art pruning algorithm for Mamba-based LLMs.

1 INTRODUCTION

The rapid expansion of Transformer-based large language models (LLMs), which now scale to hundreds of billions of parameters (Touvron et al., 2023; Zhang et al., 2022; Workshop et al., 2023), has created an urgent demand for efficient model compression. The deployment of such models involves substantial computational cost and environmental impact. Among various compression techniques, network pruning, the removal of redundant weights, remains a classic yet effective method to reduce model size and accelerate inference with minimal performance degradation (LeCun et al., 1990; Hassibi & Stork, 1993; Han et al., 2016; Ma et al., 2023; Frantar & Alistarh, 2023). However, many pruning approaches, especially those based on magnitude or gradient information, require retraining or fine-tuning to recover accuracy (Han et al., 2015; Li et al., 2017; He et al., 2017), which is feasible for smaller models but becomes prohibitively expensive on the scale of modern LLMs. To address this, researchers have introduced **training-free** pruning strategies that induce sparsity in one shot without any additional optimization, in addition to more traditional **training-based** pipelines when budget allows (Lee et al., 2019; Tanaka et al., 2020; Frantar & Alistarh, 2022).

In the Transformer regime, lots of methods have emerged to support one-shot pruning without fine-tuning, achieving surprisingly strong performance. Notably, SparseGPT Frantar & Alistarh (2023) introduced an approximate optimal brain surgeon (OBS) (Hassibi & Stork, 1993) inspired framework that prunes massive LLMs to over 50% sparsity in a single pass with negligible degradation. SparseGPT leverages local second-order information to reconstruct pruned weights and minimize output error. Simpler alternatives such as Wanda (Sun et al., 2023) offer lightweight heuristics based on the product of magnitude and input activation per output neuron, yet still achieve accuracy competitive with more sophisticated OBS-based approaches. These methods exemplify a growing trend in Transformer-based LLMs compression: **OBS-guided pruning** that can operate efficiently at scale while preserving LLM quality, even at high sparsity levels (Meng et al., 2024).

Recently, Mamba (Gu & Dao, 2023; Dao & Gu, 2024) has emerged as a promising state space alternative to Transformers, replacing the attention mechanism with a **selective state space model (SSM)** which enables linear complexity sequence processing and significantly faster inference. Mamba-based LLMs have demonstrated competitive performance compared to Transformer-based LLMs of similar scale (Zuo et al., 2024). Despite their efficiency and effectiveness, Mamba-based

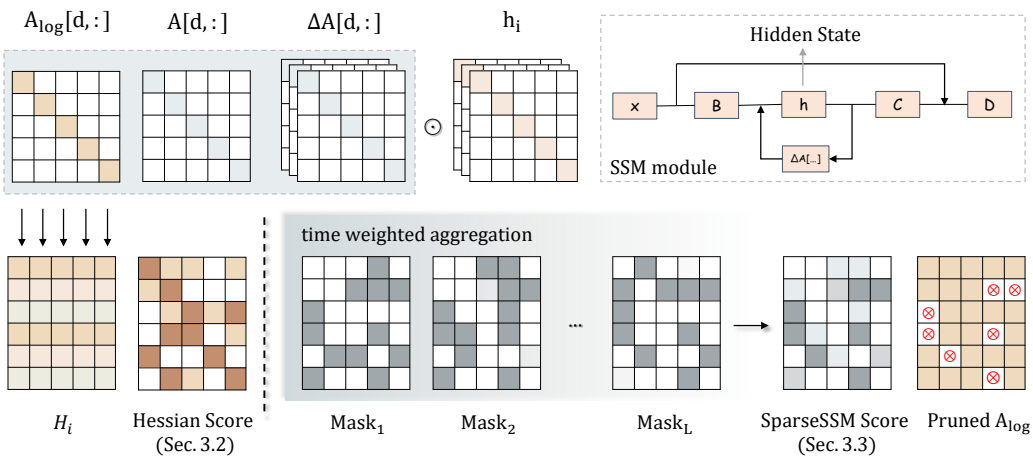

Figure 1: Illustration of *SparseSSM*. The **first row** depicts the evolution of the diagonal parameter matrix A_{log} within the SSM module, together with a schematic of the forward-propagation process. In the **second row**, the **left panel** shows the procedure for obtaining a mask from the Hessian estimate at a single time step (see Section 3.2), while the **right panel** presents our weighted strategy for merging the masks across all time steps, darker background indicates larger weights (see Section 3.3).

LLMs still contain billions of parameters and rely on time-shared parameters in the SSM module, which incur non-trivial inference time, thus stand to benefit greatly from pruning. To date, most pruning research has focused on Transformer-based models, with limited efforts targeting Mamba or other state-space architectures. This motivates a key question: **Can we design a training-free pruning method specifically tailored to the SSM module in Mamba?**

However, most existing pruning techniques developed for feed-forward and attention layers of Transformers cannot be directly transferred to the SSM module of Mamba, due to its time-shared parameters and discretization operation. For example, **(i)** The parameter A in the SSM module is time-shared, implying that any pruning decision must account for importance metrics computed at each time step; unlike spatial aggregation, however, the activations at one step are directly influenced by the previous time step. **(ii)** During execution, A is discretized into ΔA; therefore, pruning must explicitly consider this discretization operation.

To address these issues, we proposed *SparseSSM*, a layer-wise pruning method that generalizes the traditional OBS framework to efficient selective structured state space models (see Fig. 1). Our technical **contribution** can be summarized as follow:

- We introduce *SparseSSM* that adapts the classic optimal brain surgeon framework to the selective SSM module in Mamba. Our method computes approximate second-order weight importance for the time-sharing SSM parameters, enabling principled one-shot pruning of the SSM layers. This is the first application of OBS-based pruning to Mamba's architecture.
- We further improve *SparseSSM* with two complementary techniques. First, we propose a weighted mask aggregation method to address the time-sharing nature of the SSM module. Second, we provide an in-depth analysis of the components of Mamba and compare their pruning tolerance, which informs the FFN pruning strategy, shedding light on where redundancy resides in Mamba and ensuring that these heavy components are also explicitly targeted by our pruning framework.
- *SparseSSM* achieves significantly superior performance compared to current state-of-the-art pruning algorithms for Mamba-based LLMs. Through experiments on language modelling benchmarks, we demonstrate that our method can prune 50% of the SSM weights without performance degradation for Mamba-370M, also without fine-tuning. We also adapt *SparseSSM* to semi-structured and structured sparsity format, and generalized our method to other SSM-based architectures.

2 Related Work

Selective State Space Models. Selective State Space Models (SSMs) have emerged as promising alternatives to the attention layers in Transformers, particularly due to their computational complexity

in linear time and their ability to handle long-range dependencies efficiently (Gu & Dao, 2023; Dao & Gu, 2024). Unlike traditional attention-based mechanisms (Vaswani et al., 2017), whose complexity grows as the square of the sequence length, SSMs operate linearly, allowing efficient processing of exceptionally long sequences (Gu et al., 2020; 2022a; Smith et al., 2023). Mamba allows parameters within SSM layers to dynamically vary based on the input sequence (Gu & Dao, 2023), while Mamba-2 further introduces State Space Duality (SSD) to improve computational parallelism and hardware utilization (Dao & Gu, 2024; Waleffe et al., 2024). Recent hybrid architectures combining SSMs with Transformers have further demonstrated significant empirical gains, exploiting the complementary strengths of both architectures (Lieber et al., 2024; Glorioso et al., 2024; Patro & Agneeswaran, 2024; Dong et al., 2024; Waleffe et al., 2024).

Network Pruning in LLMs. Network pruning is a widely adopted technique to reduce the computational cost and memory footprint of deep neural networks by eliminating redundant parameters (LeCun et al., 1990; Han et al., 2016). Applying pruning techniques to Large Language Models (LLMs) presents unique challenges compared to smaller models like convolutional networks Singh & Alistarh (2020); Benbaki et al. (2023) or even moderate-sized language models like BERT Devlin et al. (2019). To address these challenges, several efficient pruning methods for LLMs have been proposed. Regarding granularity, these pruning methods for LLMs can be either unstructured, targeting individual weights (Frantar & Alistarh, 2023; Sun et al., 2023; van der Ouderaa et al., 2024), or structured, removing entire units like channels, filters, or attention heads (Ma et al., 2023; Tang et al., 2025). Our work mainly focuses on one-shot unstructured pruning due to its efficiency and potential for high sparsity, while it can also be extended to structured patterns.

Layer-wise Unstructured Pruning Methods. To date, layer-wise pruning methods for LLMs are primally based on the optimal brain surgeon (OBS) (Hassibi & Stork, 1993) framework. OBC (Frantar & Alistarh, 2022) proposed the ExactOBS algorithm to reduce computational burden, reformulating layer-wise pruning as a row-wise operation. To address the massive parameters of LLMs, SparseGPT (Frantar & Alistarh, 2023) further tackles the expensive Hessian computation by employing partial weight updates and adaptive mask selection. Other techniques explored more aggressive Hessian estimation (Sun et al., 2023), extension to structured sparsity (Ling et al., 2024; Wei et al., 2024), and methodological improvement for performance (Wu et al., 2024; Yu et al., 2022; Meng et al., 2024).

Pruning Methods for Mamba. While pruning algorithms tailored for Transformer-based LLMs have achieved considerable success, pruning Mamba architectures (Gu & Dao, 2023; Dao & Gu, 2024) still encounter substantial challenges. Gwak et al. reveal the redundancy and compressibility of state space models, thereby motivating the application of pruning techniques to SSM architectures (Kwak et al., 2024). Some early studies have focused on structured pruning of Mamba, such as the coarse-grained removal of SSM modules or whole blocks by Mamba-Shedder (Muñoz et al., 2025) and on unstructured pruning, evaluating a variety of pruning techniques applied to the Mamba architecture (Ghattas et al., 2025). Nearly, Taghibakhshi et al. propose a group-aware pruning strategy tailored for hybrid attention-SSM models (Taghibakhshi et al., 2025). While Shihab et al. introduce an unstructured pruning framework for Mamba that novelly uses an iterative pruning schedule(Shihab et al., 2025). Compared to earlier strategies, our solution departs in two critical aspects: **(i)** *SparseSSM* extends the classic OBS framework to address pruning in the SSM module, providing rigorous theoretical justification and comprehensive experimental validation. **(ii)** We propose a one-shot, unstructured pruning algorithm for Mamba that requires no fine-tuning.

3 METHOD

In this section, we demonstrate how the OBS framework can be adapted to the Mamba architecture and present our method, *SparseSSM*. To start, we provide a detailed overview of Mamba's forward-propagation pipeline, emphasizing the internal computations within its SSM modules. We then describe our targeted Hessian-matrix calculation technique and derive the resulting importance metrics. Finally, we explore pruning strategies for the feed-forward network (FFN) layers.

3.1 FORWARD PROPAGATION PIPELINE OF MAMBA

We first dive deeper into the forward propagation of a single mamba block. The Mamba architecture decomposes into two complementary components: a feed-forward network (FFN) that performs

feature projection and preliminary transformation, and a state space model (SSM) that selectively captures and processes sequential dependencies.

State space models provide a sequence modeling paradigm based on latent state dynamics. In Mamba's SSM layer, the state $\mathbf{h}_t \in \mathbb{R}^{B\times D\times N}$ evolves recurrently with input $\mathbf{x}$ as:

$$\mathbf{h}_t := \widehat{A}\mathbf{h}_{t-1} + \widehat{B}\,\mathbf{x}_t, \quad \mathbf{y}_t := C^\top \mathbf{h}_t, \tag{1}$$

where $\widehat{A}$ denotes the state transition matrix , $\widehat{B}$ and C are parameters of SSM, and $\mathbf{y}_t$ is the output. The DSS model (Gu et al., 2022b) first achieves efficient sequence processing by making parameters A, B' and C diagonal, then Mamba achieves selectivity by making the diagonal weight matrices time-dependent. Specifically, the original input and output gate matrices B' and C are expanded to shape $\mathbb{R}^{B\times L\times N}$, and the transition parameter A is held via zero-order-hold and discretized into $\Delta A \in \mathbb{R}^{B\times L\times D\times N}$, where B, L, D, N respectively denote the batch size, sequence length, hidden dimension, and latent state space dimension. The discretization and parameterization procedures for A are, respectively, as follows:

$$(\Delta A)_{b,\ell,d,n} = \exp\big(\delta_{b,\ell,d}\, A_{d,n}\big), \qquad (A_{log})_{d,n} = -\log\big(A_{d,n}\big), \tag{2}$$

where $\delta_{b,\ell,d}$ denotes element of the stride $\Delta \in \mathbb{R}^{B\times L\times D}$.

This selective, input-dependent design addresses the limitations of earlier linear time-invariant SSMs and enables long-context reasoning by dynamically controlling which state dimensions carry information. However, its recurrent structure and discretization operations render prior methods inapplicable to pruning the parameters of the SSM, specifically the A matrix.

In the Mamba layer, we leverage the selective scan algorithm to traverse the sequence and record the internal state contributions. At each token t, we denote by $\mathbf{h}_t \in \mathbb{R}^{B\times D\times N}$ the hidden state tensor at time step t, containing the activation values for every batch and channel. This internal signal $\mathbf{h}_t$ reflects how much each state dimension i remembers its past activation at step t. By collecting these values across all time steps $t = 1 \ldots L$ for a given layer, we obtain token-wise activation statistics for each state dimension. In this way, the selective scan provides a direct window into the layer's memory utilization, which we will exploit to guide pruning.

3.2 Hessian Matrix Estimation of SSM Layer

To formally quantify each parameter's importance in SSM layers, we adopt a Hessian-based analysis inspired by optimal brain surgeon (OBS) pruning (Hassibi & Stork, 1993; Frantar & Alistarh, 2023). The goal of pruning is to identify a sparse weight matrix A_{log} that minimizes the reconstruction error between the original and pruned layer outputs. Let $\mathbf{SSM}$ denote Mamba's SSM layer, then the problem can be formulated as:

$$\arg\min_{A'} \big\|\mathbf{SSM}\big(A,\theta,x\big) - \mathbf{SSM}\big(A',\theta,x\big)\big\|_2^2, \tag{3}$$

where θ denotes the output of the formal linear projection and A' denotes the pruned weight matrix. Based on OBS framework, pruning of parameter A requires no compensation because it is essentially a concatenation of multiple diagonal matrices, whose elements' importance can be defined as $\varepsilon_m = w_m^2 H_{mm}$, where H_{mm} denotes the m-th diagonal element of the Hessian matrix. It actually measures the curvature of the loss concerning the parameter m. However, precisely computing the entries of the Hessian matrix for an SSM module is challenging, because for a given input x, the SSM may be unrolled across time steps as follows:

$$\begin{aligned} h_i &= \Delta A_i \odot h_{i-1} + \Delta B_i \odot x\,, \quad h_{-1} = 0\,, \\ y_i &= h_i^\top C_i = \sum_{j=0}^{i}\Big[\Big(\prod_{k=j+1}^{i} \Delta A_k\Big) \odot \Delta B_j \odot x_j\Big]^\top C_i\,, \end{aligned} \tag{4}$$

where x denotes the input of the SSM module, and y_i denotes the output at each time step i. To address this problem shown in Eq. (4), we consider computing the Hessian matrix at each time step and using the hidden state to assess the importance of elements. For the SSM module, we consider the loss $\mathcal{L}_t$ incurred when it processes input data of time-step t.

$$\mathcal{L}_t = \frac{1}{B}\sum_{b=1}^{B} \ell(\mathbf{y}_{b,t}) = \frac{1}{B}\sum_{b=1}^{B} \|\mathbf{y}_{b,t} - \hat{\mathbf{y}}_{b,t}\|_2^2, \tag{5}$$

where y denotes the output of one single time step. Based on this hypothesis, we propose Theorem 1 to give an estimation of the Hessian matrix, which, to the best of our knowledge, represents a novel theoretical perspective for analyzing and guiding SSM pruning.

Theorem 1 (SSM Hessian Matrix Estimation)**.** *Let $A_{\log} \in \mathbb{R}^{D\times N}$ be the matrix of parameters for SSM diagonals, $\Delta A \in \mathbb{R}^{B\times L\times D\times N}$ the discretized parameter matrix, $\{\delta_{b,i,d}\}$ the discretization increments, and $\{h_{b,i-1,d,n}\}$ the hidden activations before step i. Under the diagonal character of parameter A_{log}, the per-parameter importance score of time step t is:*

$$I^t_{d,n} \;=\; \sum_b \Delta A^2_{b,t,d,n}\, h^2_{b,t-1,d,n} \;=\; e^{\delta_{b,t,d}A_{\log,d,n}} \sum_b h^2_{b,t-1,d,n}. \tag{6}$$

The complete proof of Theorem 1 is provided in Appendix A.

3.3 Importance Estimation for Integrated Time Steps

Our proposed Theorem 1 enables the Hessian matrix of the SSM module to be quickly and accurately estimated. However, pruning A_{log} remains challenging due to its parameter time-sharing property. This implies that while the activation at each time step can produce a pruning mask for A_{log}, pruning at one time step affects the selection of the pruning mask at the subsequent time step. Consequently, merging these masks becomes problematic.

To merge these pruning masks produced by each time step, we propose a hierarchical aggregation protocol. To reconcile the uneven importance of different time steps, we aggregate step-wise saliency with an appropriate weighting and selection scheme. Empirically, in Mamba's selective scan, We logged the hidden state at every time step and computed the expectations $H_t = \mathbb{E}||h_t||_2^2$ across a held-out set of inputs. As illustrated in figure 2, H_t exhibits a highly consistent pattern across layers. Consequently, most of the state information is rapidly accumulated in the early timesteps, and **prefix information** deserves higher emphasis. At the same time, very early measurements can be transient and noisy, and the effective influence should decay with time. We therefore adopt a power-law temporal weighting that is heavy-tailed yet monotone,

$$w_t \propto (t+1)^{-p}, \qquad p > 0, \tag{7}$$

which privileges early steps without overly collapsing mass onto the first few tokens. This prefix-biased, power-law aggregation consistently improves pruning decisions for time-shared parameters such as A_{log}. Algorithm 1 provides the detailed steps of our proposed *SparseSSM* method.

Algorithm 1 Time-Selective One-Shot OBS Pruning for the SSM Matrix $A_{\log}$

Phase 1: Statistic accumulation

$\mathbf{S} \leftarrow \mathbf{0}_{L\times D\times N},\ n \leftarrow 0$
for each mini-batch $b = 1, \dots, B$ **do**
  Run the forward pass of the layer and collect $\mathbf{h}^{(b)}_{1:\mathrm{L}}$
  $n \leftarrow n + 1$
  $\mathbf{S} \leftarrow \dfrac{n-1}{n}\,\mathbf{S} + \dfrac{1}{n}\big(\mathbf{h}^{(b)}_{1:\mathrm{L}}\big)^2$
end for

Phase 2: Time-weighted global scoring

$K \leftarrow \lceil pDN \rceil$
$\boldsymbol{\Delta A}_{1:\mathrm{L}} \leftarrow \sum_{b=1}^{B} \boldsymbol{\Delta A}^{(b)}_{1:\mathrm{L}} \in \mathbb{R}^{L\times D\times N}$ ▷ sum across batches
$w_t \leftarrow (t{+}1)^{-\text{power}}$ for $t = 1, \dots, L$; $\quad w_t \leftarrow \dfrac{w_t}{\sum_{u=1}^{L} w_u}$ ▷ prefix-biased weights
$\mathbf{Q} \leftarrow \sum_{t=1}^{L} w_t \Big(\boldsymbol{\Delta A}_t^2 \odot \mathbf{S}_t\Big)$ ▷ $\odot$: element-wise
$\mathcal{I}_\star \leftarrow \arg\operatorname{smallest}_K\big(\mathbf{Q}\big)$
$\mathbf{M} \leftarrow \mathbf{1}_{D\times N};\ \ \mathbf{M}_{\mathcal{I}_\star} \leftarrow 0$
$\widetilde{A}_{\log} \leftarrow A_{\log} \odot \mathbf{M}$

3.4 Sensitivity-Aware Pruning of the FFN Component

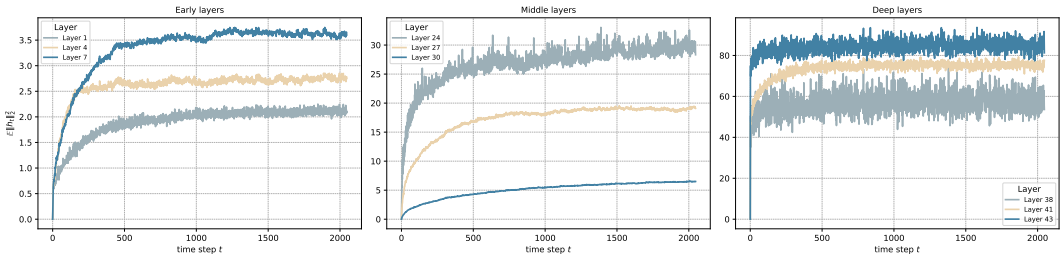

Figure 2: The evolution of H_t over timesteps across different layers of Mamba

While our primary contribution is the OBS-based pruning of Mamba's state space module, we also perform pruning on the standard feed-forward networks (FFNs) in the model to further reduce parameters based on the SparseGPT (Frantar & Alistarh, 2023) framework. The forward-propagation blocks in Mamba are composed primarily of linear layers and one-dimensional convolutional layers. Inspired by (Shao et al., 2024), we conducted a module-wise pruning analysis within the feed-forward network (FFN) and found that their pruneability varies substantially (see Appendix B.2.3). In particular, pruning the `in_proj` and `out_proj` modules incurs a pronounced degradation in overall model performance. Moreover, we empirically observe that the reconstruction error of each module grows as its Hessian trace increases, with the rate of this growth varying across modules, as shown in Fig. 3.

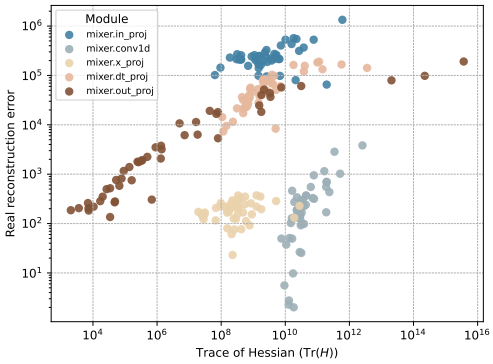

Figure 3: The Hessian matrices and corresponding reconstruction errors for each module of the Mamba-370M FFN at a sparsity level of 50%.

Motivated by these findings, we adopt the sensitivity-aware pruning framework, treating the `in_proj` and `out_proj` modules independently. We use the trace of the Hessian matrix of the weights as the sensitivity score, and define the sparsity ratio as:

$$sparsity = 1 - p - \alpha + \frac{2\alpha\, id}{N-1}, \tag{8}$$

where p is the target global sparsity, α defines the allowable deviation interval $\big[1-p-\alpha,\ 1-p+\alpha\big]$, N is the total number of weights, and id is the sensitivity-rank index of the given weight after sorting by Hessian-trace importance. This formulation ensures that higher-sensitivity weights (larger id) are assigned lower sparsity, while exactly satisfying the overall sparsity budget p.

4 Experiments

In this section, we benchmark *SparseSSM* against leading pruning algorithms for SSM-based LLMs. The complete experimental protocol and reproducibility details appear in Appendix B.1.

Models and Datasets. We evaluate *SparseSSM* on the public Mamba-1 checkpoints ranging from 130 million to 2.8 billion parameters (Gu & Dao, 2023), and Mamba-2 checkpoints ranging from 370 million to 2.7 billion parameters. Additionally, we include the 8B-parameter Mamba-2 model trained with the Megatron framework and released as part of the empirical study of Mamba-based LLMs, in order to stress-test the robustness and scalability of our method on a large production-scale model. For all models, we follow the standard calibration protocol on WikiText-2: we randomly sample 128 contiguous segments of 2048 tokens each from the first data shard, as in (Frantar & Alistarh, 2023). Perplexity is computed as the exponential of the negative log-likelihood per token, consistent with (Hugging Face, 2022). Downstream evaluation uses the raw WikiText-2 validation set (Merity et al., 2017), the Penn Treebank corpus (Marcus et al., 1994), and a 10000-document slice of the C4 validation split (Raffel et al., 2020). In addition, to probe behavior on a modern,

information-dense web corpus, we report supplementary results on a validation subset of the FineWeb dataset (Penedo et al., 2024) in Appendix B.1. Zero-shot generalization is measured on PIQA (Bisk et al., 2020), OpenBookQA (Mihaylov et al., 2018), Winogrande (Sakaguchi et al., 2019), ARC-Easy, and ARC-Challenge (Clark et al., 2018) without any fine-tuning. This suite covers both language modeling and reasoning benchmarks, allowing a comprehensive assessment of model performance and generalization. During implementation, we also referred to mamba-minimal (Ma) for guidance.

Baselines. We compare *SparseSSM* against three representative pruning methods under identical calibration and sparsity budgets. First, global magnitude pruning follows the classical heuristic of removing the smallest-magnitude weights (Han et al., 2015). Second, SparseGPT applies a Hessian-aware one-shot pruning strategy (Frantar & Alistarh, 2023). However, SparseGPT is not inherently suited to the structural characteristics of SSM modules. Here, we present the results of its naive application. Third, Mamba-Shedder is a recent selective state space variant tailored for Mamba architectures (Muñoz et al., 2025). Additionally, we also include Wanda (Sun et al., 2023) as an extra baseline, which is a recent pruning method developed under the OBS framework. *All baselines and our method share the same configuration to ensure fairness.*

4.1 Results of Pruning SSM Modules

We first isolate the SSM blocks and prune only the learnable diagonal A_{log} matrices. Within the state-space module (SSM), Mamba reparameterizes A via its negative logarithm to enforce $A < 0$, thus preserving the module's robustness. Indeed, the parameter A_{log} plays a role analogous to the forget gate in LSTM (Hochreiter & Schmidhuber, 1997) networks and has a profound impact on the predictive capacity of the language model.

Table 1 reports detailed token-level perplexity and zero-shot accuracies at 50% sparsity. As demonstrated, our pruning strategy shows excellent efficacy in maintaining SSM stability. Across all model scales, *SparseSSM*consistently outperforms MP, Mamba-Shedder, and SparseGPT at the same sparsity, matching or exceeding the dense baseline on most zero-shot tasks while retaining competitive perplexities. For example, on the Mamba-370M model, our method attains no degradation on the majority of zero-shot evaluations and yields a $\sim$5.4% average accuracy gain over alternative pruners, without any fine-tuning. The gains stem from our second-order importance metric combined with a time-weighted mask, as detailed in Section 3. We further observe that larger models exhibit lower tolerance to pruning, as evidenced by more pronounced declines in zero-shot performance.

Additionally, to assess pruning effectiveness at higher sparsity, we evaluate models of different sizes across a range of sparsity levels and examine their performance, see Appendix B.2.9. We provide additional results on more baselines, datasets, and model scales in the Appendix B.2.6.

4.2 Results of Pruning the Whole Mamba Architecture

We then apply one-shot unstructured pruning across all trainable weights except the input embedding and output head. In this setting, each model typically incorporates an `nn.Conv1d` layer for feature preprocessing, `in_proj` and `out_proj` linear layers for dimensionality transformation, and—immediately before the selective scan operation, a learnable `x_proj` mapping that produces the parameters Δ, B, C, concurrently, the temporal stride parameter Δ is reparameterized via `dt_proj`. Empirical analysis reveals that these modules exhibit markedly heterogeneous pruning tolerances: pruning of the `in_proj` and `out_proj` layers induces substantially larger degradations than other linear modules, detailed comparison results are in Appendix B.2.3.

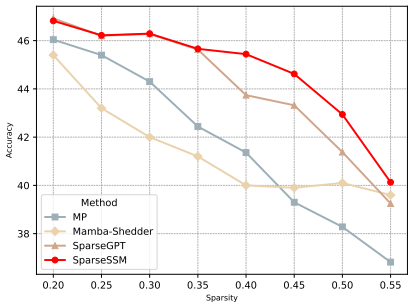

Figure 4: Performance of the full Mamba architecture at multiple sparsity levels by measuring zero-shot task accuracy

However, when we jointly prune both the SSM modules and the FFN branches, our proposed method *SparseSSM* outperforms all baselines, achieving lower perplexity and higher zero-shot accuracy across every model scale, as

Table 1: Performance analysis for one-shot unstructured pruning of SSM modules in Mamba models at 50% sparsity. Here, ↓ lower metrics reflect better outcomes, and ↑ denotes higher metrics reflect better outcomes.

Model	Method	Wiki. ↓	PTB ↓	C4 ↓	OBQA ↑	PIQA ↑	ARC-e ↑	ARC-c ↑	WinoG ↑	Avg. ↑
Mamba-1 model										
Mamba-130M	Dense	20.60	32.75	25.66	28.60	63.28	48.02	24.40	52.5	43.36
	MP (Han et al., 2015)	740.3	1109	273.0	26.80	58.05	39.69	22.35	**52.33**	39.84
	Wanda (Sun et al., 2023)	77.84	111.7	52.32	28.60	60.88	39.69	23.21	51.78	40.83
	Mamba-Shedder (Muñoz et al., 2025)	698.7	1544	532.6	28.00	54.73	30.00	23.72	49.88	37.27
	SparseGPT (Frantar & Alistarh, 2023)	2.4e7	6.1e6	3.9e5	27.60	55.28	30.64	23.98	49.25	37.35
	SparseSSM	**21.26**	**33.82**	**26.31**	**30.60**	**63.33**	**45.62**	**24.66**	50.75	**42.99**
Mamba-370M	Dense	14.32	23.46	19.37	31.00	68.34	54.97	27.90	55.25	47.49
	MP (Han et al., 2015)	291.2	535.9	105.0	**30.40**	61.70	44.23	22.61	51.38	42.06
	Wanda (Sun et al., 2023)	40.91	79.89	37.49	27.60	64.20	37.71	25.85	53.35	41.74
	Mamba-Shedder (Muñoz et al., 2025)	334.5	446.6	221.81	23.80	54.19	29.42	23.12	52.25	36.56
	SparseGPT (Frantar & Alistarh, 2023)	2696	7570	613.2	**30.40**	65.23	49.16	25.60	52.41	44.56
	SparseSSM	**15.04**	**24.63**	**20.13**	30.20	**68.34**	**55.26**	**27.90**	**55.64**	**47.47**
Mamba-790M	Dense	11.96	18.45	16.62	33.80	72.63	61.07	29.44	56.27	50.64
	MP (Han et al., 2015)	179.0	377.0	79.43	30.20	64.74	47.81	25.09	54.14	44.40
	Wanda (Sun et al., 2023)	27.63	50.45	27.83	31.40	66.00	52.02	26.19	53.04	45.73
	Mamba-Shedder (Muñoz et al., 2025)	225.48	256.32	195.47	28.20	56.47	33.29	21.50	51.07	38.11
	SparseGPT (Frantar & Alistarh, 2023)	110.5	242.19	81.87	**32.80**	68.34	54.42	27.47	54.93	47.59
	SparseSSM	**12.47**	**19.43**	**17.19**	32.60	**70.40**	**57.11**	**30.46**	**56.27**	**49.37**
Mamba-1.4B	Dense	10.75	17.05	15.17	36.40	73.88	65.57	32.85	61.17	53.98
	MP (Han et al., 2015)	100.7	190.8	54.49	30.60	67.95	53.28	24.06	52.49	45.68
	Wanda (Sun et al., 2023)	21.17	38.19	21.86	32.20	68.55	56.69	27.73	51.62	47.36
	Mamba-Shedder (Muñoz et al., 2025)	223.1	293.7	190.5	27.20	56.86	34.09	23.04	51.46	38.53
	SparseGPT (Frantar & Alistarh, 2023)	49.77	88.20	40.74	34.40	71.38	60.10	30.03	54.78	50.14
	SparseSSM	**11.89**	**18.93**	**16.49**	**35.00**	**72.42**	**61.99**	**31.23**	**56.83**	**51.49**
Mamba-2.8B	Dense	9.45	14.79	13.61	39.60	75.84	69.70	36.35	63.06	56.91
	MP (Han et al., 2015)	61.08	70.75	39.13	**35.40**	70.57	59.13	29.35	60.30	50.95
	Wanda (Sun et al., 2023)	15.31	21.10	17.53	34.40	72.36	63.05	30.12	57.93	51.57
	Mamba-Shedder (Muñoz et al., 2025)	104.7	126.6	112.1	25.80	59.00	40.20	23.50	51.20	39.95
	SparseGPT (Frantar & Alistarh, 2023)	56.83	92.31	44.43	34.80	**73.94**	**65.74**	**34.04**	58.56	53.42
	SparseSSM	**10.63**	**16.98**	**15.14**	**35.40**	73.45	65.07	33.45	**60.54**	**53.58**
Mamba-2 model										
Mamba2-370M	Dense	14.17	22.01	19.07	32.40	69.26	54.84	26.71	55.41	47.72
	MP (Han et al., 2015)	15.28	23.69	20.22	**32.00**	**70.13**	**54.88**	25.85	53.99	**47.37**
	Mamba-Shedder (Muñoz et al., 2025)	38.87	95.31	32.87	29.80	66.65	46.97	25.00	53.83	44.45
	SparseGPT (Frantar & Alistarh, 2023)	15.42	24.33	20.90	31.20	69.26	53.79	26.54	**54.38**	47.03
	SparseSSM	**14.89**	**23.48**	**19.92**	31.60	68.93	54.55	**27.13**	**54.38**	47.32
Mamba2-780M	Dense	11.81	18.20	16.46	36.20	71.82	61.03	28.50	60.22	51.55
	MP (Han et al., 2015)	**12.70**	20.42	18.26	35.40	**71.65**	**59.51**	**28.50**	56.83	**50.38**
	Mamba-Shedder (Muñoz et al., 2025)	27.06	52.07	24.86	33.00	68.01	46.17	27.13	53.04	45.47
	SparseGPT (Frantar & Alistarh, 2023)	12.81	19.59	18.28	**35.80**	71.44	58.29	27.56	56.83	49.98
	SparseSSM	**12.70**	**19.20**	**18.13**	35.40	71.60	59.34	28.07	**57.38**	50.36

shown in Table 6 in Appendix B.2.1. Fig. 4 further illustrates our method's performance across multiple sparsity levels. As demonstrated, for each downstream task, the pruned models exhibit consistent improvements, with gains becoming especially pronounced under higher sparsity regimes.

4.3 Results of Semi-Structure and Structure Sparsity Extension

Our approach admits a straightforward extension to $N : M$ and fully structured pruning. In fact, during unstructured pruning experiments, we observed that the pruned entries in the parameter A_{log} overwhelmingly cluster within particular columns. Certain hidden state channels in the state space model exhibit markedly higher redundancy. This empirical finding underpins the strong performance of our structured pruning scheme.

Table 4 reports results on the Mamba-370M model under 2:4 and 4:8 sparsity patterns. At the same overall sparsity, our method delivers smaller performance degradation in $N : M$ pruning.

To implement structured pruning, we target the second axis of A_{log}. We employ the weight itself to guide the pruning and aggregate the importance of each column by computing its L_1 norm. Simultaneously, we resize the output dimension of the linear `x_proj` layer. As shown in Table 5, this structured pruning on Mamba-370M induces only negligible accuracy loss at 50% sparsity without

Table 2: Performance and efficiency analysis for one-shot structure pruning of SSM modules in Mamba-370M models.

Overall Sparsity	$A_{\log}$ Sparsity	Avg. Zero-shot Acc. ↑	WikiText ppl ↓	Speedup (Parallel Scan)	Speedup (Recurrent)
Dense	Dense	47.49	14.32	—	—
0.1%	25%	46.85	15.22	1.07×	1.57×
0.2%	50%	46.31	18.13	1.09×	1.68×
0.3%	75%	41.87	31.39	1.12×	1.76×

Table 3: Total CUDA time and speedup in **parallel scan** vs. **recurrent** modes.

Model	Parallel Scan Mode			Recurrent Mode		
	Dense (ms)	Pruned (ms)	Speedup	Dense (ms)	Pruned (ms)	Speedup
Mamba-130m	6.270(±0.07)	5.790(±0.08)	1.08(±0.02)×	251.4(±0.20)	232.5(±0.18)	1.08(±0.00)×
Mamba-370m	17.59(±0.17)	16.03(±0.23)	1.09(±0.02)×	546.7(±0.35)	329.4(±0.27)	1.66(±0.00)×
Mamba-790m	30.78(±0.41)	29.11(±0.31)	1.06(±0.02)×	661.9(±0.64)	388.5(±0.51)	1.70(±0.00)×
Mamba-1.4b	48.39(±0.70)	46.32(±0.65)	1.05(±0.02)×	1090(±0.02)	656.1(±1.39)	1.66(±0.00)×
Mamba-2.8b	94.49(±0.93)	90.46(±1.40)	1.05(±0.02)×	1648(±0.04)	737.6(±2.31)	2.23(±0.01)×

any fine-tuning, while accelerating the SSM module by a factor of 1.72×. Although the number of parameters in A_{log} is relatively small, its contribution to acceleration is substantial because it is repeatedly utilized, especially in scenarios where specialized CUDA operators cannot be employed, such as on edge devices or non-NVIDIA GPUs, see Table 2.

As reported in Table 3, our pruning method yields consistent runtime acceleration across both parallel scan and recurrent execution modes. Therefore, pruning directly translates into significant end-to-end runtime gains in practical inference scenarios. Detailed configurations are provided in the Appendix B.2.4. In addition, our method also leads to substantial memory savings, See Appendix B.2.5 for memory results.

Table 4: Performance analysis for one-shot pruning of the SSM module in Mamba-370M at $2:4$ and $4:8$ sparsity patterns.

Sparsity	Method	Wiki. ↓	PTB ↓	C4 ↓	OBQA ↑	PIQA ↑	ARC-e ↑	ARC-c ↑	WinoG ↑	Avg. ↑
2 : 4	MP	77.20	135.9	59.74	25.60	56.80	34.85	21.67	51.07	38.00
	SparseSSM	**16.90**	**26.91**	**21.61**	**30.80**	**68.28**	**54.08**	**26.37**	**52.96**	**46.50**
4 : 8	MP	81.56	148.25	63.76	26.00	55.44	37.54	21.84	50.12	38.19
	SparseSSM	**16.56**	**26.44**	**21.40**	**31.20**	**68.44**	**54.88**	**26.45**	**53.20**	**46.83**

Table 5: Performance analysis for one-shot structured pruning of the SSM module in Mamba-370M.

Sparsity	Method	Wiki. ↓	PTB ↓	C4 ↓	OBQA ↑	PIQA ↑	ARC-e ↑	ARC-c ↑	WinoG ↑	Avg. ↑
25%	MP	35.27	71.12	33.85	27.40	60.94	43.43	24.91	50.75	41.49
	SparseSSM	**15.22**	**24.80**	**20.38**	**30.60**	**68.44**	**53.66**	**27.30**	**54.22**	**46.85**
50%	MP	117.0	162.7	66.74	26.00	55.82	35.69	23.29	49.09	37.98
	SparseSSM	**18.13**	**28.82**	**22.65**	**30.40**	**67.68**	**53.53**	**27.30**	**52.64**	**46.31**

4.4 Generalize to other SSM architecture

Beyond the vanilla Mamba-1/2 language models considered in the main experiments, recent work has extended SSMs to diverse domains, including vision backbones and large-scale hybrid Transformer–Mamba language models.

MambaVision (Hatamizadeh & Kautz, 2025) is a recent hybrid Mamba–Transformer vision backbone that redesigns the Mamba formulation for 2D feature maps and interleaves SSM blocks with ViT-style attention, providing an efficient and accurate alternative to pure ViT architectures on image

classification and detection tasks. We adopt the official pretrained checkpoints released by NVLabs and directly apply our pruning algorithm to the SSM modules. Empirically, the pruned MambaVision models largely preserving the original accuracy. Jamba (Lieber et al., 2024) interleaves Transformer attention layers with Mamba SSM layers and incorporates sparse Mixture-of-Experts blocks. The pruned Jamba models retain most of the baseline performance at moderate sparsity level. The detailed experiment setting and results are in the Appendix B.2.7.

4.5 Ablation Study

We conduct a systematic ablation study on the key components of *SparseSSM* to isolate their contributions to pruning efficacy. In particular, we find that accurate Hessian matrix estimation is instrumental to our method's superior performance, while incorporating a temporal pruning-frequency metric yields additional gains. As shown in Table 8, our full strategy significantly outperforms a simpler baseline that applies our Hessian estimate via a simple L_2 norm aggregation over time steps. We further conduct additional ablation studies on the choice of the exponent p in our timestep aggregation scheme, see Appendix B.2.8 for details.

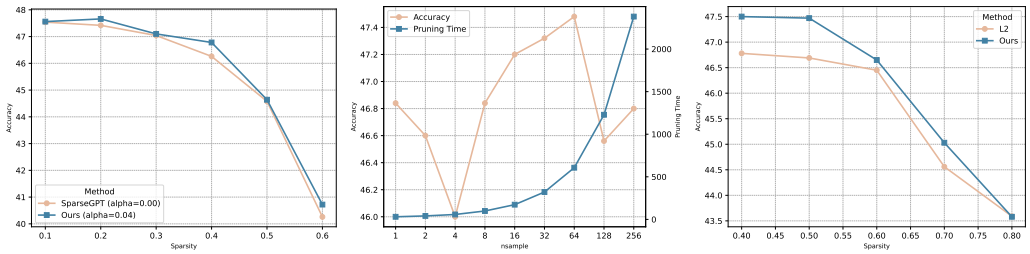

Figure 5: Effects of calibration sample size, sparsity interval, and aggregation method. **(Left)** shows a performance analysis of pruning the FFN components of Mamba-370M under varying sparsity settings, **(middle)** shows a performance analysis and pruning efficiency analysis of pruning the SSM modules as the calibration sample size is varied, while **(right)** shows a performance analysis of different aggregation methods.

We further assess the effects of sensitivity pruning width and calibration data volume on final results. We change the super parameter α that controls pruning width and N_{sample} that controls calibration data volume. Our experiments reveal that selecting an appropriately sized width parameter substantially improves pruning outcomes in the FFN components, surpassing the performance of SparseGPT. As for calibration data, we observe that fewer than 16 samples degrade the performance of the pruned model. However, a sampling count of 64 strikes the best trade-off between pruning quality and computational efficiency.

5 Conclusion

In this work, we introduce *SparseSSM*, a one-shot, training-free unstructured pruning framework that extends the classic OBS paradigm to selective state-space modules in Mamba-based LLMs. By incorporating time-sharing parameter saliency and explicitly accounting for the discretization of the state-transition matrix, our layer-wise algorithm computes local second-order importance scores and reconstructs remaining weights to minimize output error. Furthermore, our module sensitivity analysis reveals distinct pruning tolerances between input and output projections, offering new insights into redundancy within state-space architectures. Our results establish that state-space LLMs like Mamba can be compressed as effectively as their Transformer counterparts via principled, OBS-guided pruning, paving the way for more efficient deployment within resource-restricted contexts. In future work, we plan to further extend *SparseSSM* to structured pruning of the entire Mamba architecture. We also aim to generalize our time weighted aggregation and insights on time-shared parameters to other time-varying architectures and investigate hardware-aware optimizations that further accelerate sparse state-space inference.

REFERENCES

Riade Benbaki, Wenyu Chen, Xiang Meng, Hussein Hazimeh, Natalia Ponomareva, Zhe Zhao, and Rahul Mazumder. Fast as chita: Neural network pruning with combinatorial optimization. In *ICML*, 2023.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *AAAI*, 2020.

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

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *ICML*, 2024.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

Xin Dong, Yonggan Fu, Shizhe Diao, Wonmin Byeon, Zijia Chen, Ameya Sunil Mahabaleshwarkar, Shih-Yang Liu, Matthijs Van Keirsbilck, Min-Hung Chen, Yoshi Suhara, Yingyan Lin, Jan Kautz, and Pavlo Molchanov. Hymba: A hybrid-head architecture for small language models. *arXiv preprint arXiv:2411.13676*, 2024.

Elias Frantar and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. In *ICML*, 2022.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *ICML*, 2023.

T. Ghattas, M. Hassid, and R. Schwartz. On pruning state-space llms. *arXiv preprint arXiv:2502.18886*, 2025.

P. Glorioso, Q. Anthony, Y. Tokpanov, et al. The zamba2 suite: Technical report. *arXiv preprint arXiv:2411.15242*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Re. Hippo: Recurrent memory with optimal polynomial projections. In *NeurIPS*, 2020.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *ICLR*, 2022a.

Albert Gu, Ankit Gupta, Karan Goel, and Christopher Ré. On the parameterization and initialization of diagonal state space models. In *NeurIPS*, 2022b.

Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *NeurIPS*, 1993.

A. Hatamizadeh and J. Kautz. Mambavision: A hybrid mamba-transformer vision backbone. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Hugging Face. Perplexity of fixed-length models. 2022.

Minsunu Kwak, Seungrok Moon, Joohwan Ko, and POOGYEON PARK. Layer-adaptive state pruning for deep state space models. In *NeurIPS*, 2024.

Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *NeurIPS*, 1990.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2019.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid transformer-mamba language model. In *ICLR*, 2024.

Gui Ling, Ziyang Wang, Yuliang Yan, and Qingwen Liu. Slimgpt: Layer-wise structured pruning for large language models. In *NeurIPS*, 2024.

John Ma. mamba-minimal: A minimal pytorch implementation of mamba. https://github.com/johnma2006/mamba-minimal.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *NeurIPS*, 2023.

Mitch Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: Annotating predicate argument structure. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994.

Xiang Meng, Kayhan Behdin, Haoyue Wang, and Rahul Mazumder. Alps: Improved optimization for highly sparse one-shot pruning for large language models. In *NeurIPS*, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR*, 2017.

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

J. Pablo Muñoz, Jinjie Yuan, and Nilesh Jain. Mamba-shedder: Post-transformer compression for efficient selective structured state space models. In *NAACL*, 2025.

Badri N Patro and Vijay S Agneeswaran. Simba: Simplified mamba-based architecture for vision and multivariate time series. *arXiv preprint arXiv:2403.15360*, 2024.

G. Penedo, H. Kydlíček, L. Ben Allal, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*, 2024.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21(140):1–67, 2020.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019.

Hang Shao, Bei Liu, and Yanmin Qian. One-shot sensitivity-aware mixed sparsity pruning for large language models. In *ICASSP*, 2024.

Ibne Farabi Shihab, Sanjeda Akter, and Anuj Sharma. Efficient unstructured pruning of mamba state-space models for resource-constrained environments. In *EMNLP*, 2025.

Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximations for model compression. In *NeurIPS*, 2020.

Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for sequence modeling. In *ICLR*, 2023.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

A. Taghibakhshi, S. T. Sreenivas, S. Muralidharan, et al. Efficient hybrid language model compression through group-aware ssm pruning. *arXiv preprint arXiv:2504.11409*, 2025.

Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *NeurIPS*, 2020.

Shengkun Tang, Oliver Sieberling, Eldar Kurtic, Zhiqiang Shen, and Dan Alistarh. Darwinlm: Evolutionary structured pruning of large language models. *arXiv preprint arXiv:2502.07780*, 2025.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, MarieAnne Lachaux, Timothe Lacroix, Baptiste Rozire, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Tycho F. A. van der Ouderaa, Markus Nagel, Mart van Baalen, Yuki M. Asano, and Tijmen Blankevoort. The llm surgeon. In *ICLR*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Roger Waleffe, Miles Solbrig, Ryan Turner, et al. An empirical study of mamba-based language models. *arXiv preprint arXiv:2406.07887*, 2024.

Jiateng Wei, Quan Lu, Ning Jiang, Siqi Li, Jingyang Xiang, Jun Chen, and Yong Liu. Structured optimal brain pruning for large language models. In *NeurIPS*, 2024.

BigScience Workshop, :, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, and Daniel Hesslow et al. Bloom: A 176b-parameter open-access multilingual language model, 2023.

Diyuan Wu, Ionut-Vlad Modoranu, Mher Safaryan, Denis Kuznedelev, and Dan Alistarh. The iterative optimal brain surgeon: Faster sparse recovery by leveraging second-order information. In *NeurIPS*, 2024.

Xin Yu, Thiago Serra, Srikumar Ramalingam, and Shandian Zhe. The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. In *ICML*, 2022.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, and Xi Victoria Lin et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

J. Zuo, M. Velikanov, D. E. Rhaiem, et al. Falcon mamba: The first competitive attention-free 7b language model. *arXiv preprint arXiv:2410.05355*, 2024.

A PROOFS OF THEOREM 1

Proof. We begin with a trained network's parameters, where $A_{\log}$ is near a local minimum of the loss L. In this setting, small perturbations of the parameters cause a loss increase dominated by the quadratic term of the second-order Taylor expansion.

Lemma 1 (OBS Parameter Importance)**.** *Under the second-order OBS pruning framework, let* $\mathbf{H} = \nabla^2 \mathbf{L}(\theta)$ *denote the Hessian of the loss L with respect to the full parameter vector* θ*. Then the saliency of the individual parameter* $\mathbf{A}_{\log,\mathbf{d},\mathbf{n}}$ *is given by*

$$I^t_{d,n} \;=\; \frac{\big(A_{t,d,n}\big)^2}{2\,\big[H^{-1}\big]_{(d,n),(d,n)}} \;=\; \frac{1}{2}\,H_{(d,n),(d,n)}\,\big(A_{t,d,n}\big)^2. \tag{9}$$

Returning to the proof of the main theorem. We analyze the single scalar coordinate

$$w \;\equiv\; \Delta A_{b,t,d,n}, \tag{10}$$

which multiplies its corresponding hidden coordinate at step t:

$$h_{b,t,d,n} \;=\; w\,h_{b,t-1,d,n} \;+\; (\Delta B_{b,t} u_{b,t})_{d,n}. \tag{11}$$

Hessian as sample covariance of Jacobians. Consider a per-step mean–squared error (MSE) objective at time t:

$$E \;=\; \frac{1}{2B}\sum_{b=1}^{B} \big\|F_{b,t}(w) - T_{b,t}\big\|_2^2, \tag{12}$$

where $F_{b,t}(w)$ is the network output at step t and $T_{b,t}$ is the target (or the teacher output). Its gradient with respect to a scalar parameter w is

$$\frac{\partial E}{\partial w} \;=\; \frac{1}{B}\sum_{b=1}^{B}\left(\frac{\partial F_{b,t}(w)}{\partial w}\right)^{\top}\Big(F_{b,t}(w) - T_{b,t}\Big). \tag{13}$$

The exact Hessian is

$$\frac{\partial^2 E}{\partial w^2} \;=\; \frac{1}{B}\sum_{b=1}^{B}\left[\left(\frac{\partial F_{b,t}(w)}{\partial w}\right)^{\top}\frac{\partial F_{b,t}(w)}{\partial w} \;+\; \Big(F_{b,t}(w) - T_{b,t}\Big)^{\top}\frac{\partial^2 F_{b,t}(w)}{\partial w^2}\right]. \tag{14}$$

Near a trained solution, the residuals $F_{b,t}(w) - T_{b,t}$ are small, hence the second term is negligible. Thus, the Gauss–Newton (GN) / diagonal-OBS approximation gives

$$H_{mm} \;\approx\; \frac{1}{B}\sum_{b=1}^{B}\left\|\frac{\partial F_{b,t}(w)}{\partial w}\right\|_2^2, \tag{15}$$

The Hessian is approximately the sample covariance of the output Jacobians with respect to w. Because w acts multiplicatively and *component-wise* on $h_{b,t-1,d,n}$, cross-partials across different (b,t,d,n) are negligible. The Hessian is effectively block-diagonal in these coordinates.

By direct differentiation of the local update,

$$\frac{\partial h_{b,t,d,n}}{\partial w} \;=\; h_{b,t-1,d,n}. \tag{16}$$

Therefore,

$$H_{mm} \;\approx\; \frac{1}{B}\sum_{b=1}^{B} h^2_{b,t-1,d,\,n} \;\propto\; \sum_{b=1}^{B} h^2_{b,t-1,d,\,n}, \tag{17}$$

where the proportionality absorbs positive, step-dependent factors independent of w.

According to lemma 1, the (diagonal) OBS saliency for a scalar parameter w is,

$$S(w) \;=\; \frac{w^2}{2\,[H^{-1}]_{mm}} \;=\; \frac{1}{2}\,H_{mm}\,w^2, \tag{18}$$

and using the estimate of H_{mm} above with $w = \Delta A_{b,t,d,n}$, the per-sample contribution satisfies

$$S\big(\Delta A_{b,t,d,n}\big) \;\propto\; \big(\Delta A_{b,t,d,n}\big)^2 \, h^2_{b,t-1,d,n}. \tag{19}$$

Summing over the batch at fixed (t, d, n) yields the per-step importance

$$I^t_{d,n} \;=\; \sum_b \Delta A^2_{b,t,d,n} \, h^2_{b,t-1,d,n} \;=\; e^{\delta_{b,t,d} A_{\log,d,n}} \sum_b h^2_{b,t-1,d,n}. \tag{20}$$

which completes the proof. □

A.1 Proof of Lemma 1

Let θ denote the vector of all parameters and $H = \nabla^2 L(\theta)$ the Hessian at the optimum. For a perturbation $\Delta\theta$, the Taylor expansion gives:

$$\Delta L \;\approx\; \frac{1}{2} \Delta\theta^T H \, \Delta\theta \,. \tag{21}$$

In the SSM module, Over a small time increment $\delta_{b,t,d}$ at step t, the state update is:

$$h_{b,t,d,n} \;=\; \Delta A_{t,d,n} \; h_{b,t-1,d,n} \;+\; \Delta\big(B_u\big)_t \,, \tag{22}$$

where $\Delta\big(B_u\big)_i$ is independent with parameter A. The only way $A_{\log,d,n}$ affects the network's forward pass is through this scalar multiplier $e^{A_{d,n},\delta_{b,i,d}}$ at each time step. Crucially, because A is diagonal, each parameter $A_{\log,d,n}$ influences only its corresponding state dimension d in SSM n, independently of other dimensions, which implies

$$\frac{\partial^2 L}{\partial A_{t,d,n} \partial A_{t,d',n'}} \;=\; 0 \,, \;\; \big(d', n'\big) \neq \big(d, n\big). \tag{23}$$

Therefore, the Hessian matrix H has the characteristic

$$\big[H^{-1}\big]_{(d,n),(d,n)} = \frac{1}{H_{(d,n),(d,n)}}. \tag{24}$$

where $H_{(d,n),(d,n)} = \partial^2 L \,/\, \partial A^2_{\log,d,n}$ is the Hessian's diagonal entry for that parameter. Combining with the classic OBS saliency definition $\varepsilon_m = w_m^2 \,/\, [\mathbf{H}^{-1}]_{mm}$,then we define the OBS saliency of parameter $A_{log,d,n}$ as

$$I^t_{d,n} \;=\; \frac{\big(A_{t,d,n}\big)^2}{2\,\big[H^{-1}\big]_{(d,n),(d,n)}} \;=\; \frac{1}{2} \, H_{(d,n),(d,n)} \, \big(A_{t,d,n}\big)^2. \tag{25}$$

□

B Experiments Details

B.1 Experiments Setup

We performed all experiments on a dedicated server using dual Intel Xeon Platinum 8457C processors (48 cores / 96 threads each), 512 GB of DDR5 memory, and eight NVIDIA GeForce RTX 4090 GPUs (24 GB each). We used the PyTorch library to implement the Mamba model and pruning methods for our experiments.

We based our implementation on the SparseGPT code framework (Frantar & Alistarh, 2023), performing pruning on a per-module basis by registering forward hooks to capture each module's inputs during the forward pass. After pruning a given layer, we update its inputs to maintain correct activation propagation. For each pruned module, we remove the designated parameters to realize the prescribed sparsity.

In our Mamba implementation, we adopted the mamba-minimal (Ma) code framework and loaded the official Mamba checkpoint (Gu & Dao, 2023) for pretrained weights. To meet our experimental objectives, we introduced a small set of modifications to the mamba-minimal implementation.

Hyperparameters. For SSM-module pruning, we set $N_{sample} = 64$, which we found yields the best trade-off between pruning quality and computational cost. In the FFN pruning stage, we chose $\alpha = 0.04$, implying that each FFN submodule is assigned a sparsity rate of

$$S_{\text{FFN},i} = \begin{cases} 0.96 - p + \dfrac{0.08\, id}{N-1}, & \text{if } i \in \{\texttt{in\_proj}, \texttt{out\_proj}\}, \\ S_{\text{global}}, & \text{otherwise}, \end{cases} \tag{26}$$

where N is the total number of weights, and id is the sensitivity-rank index of the given weight after sorting by Hessian-trace importance. It means that for the modules `in_proj` and `out_proj`, the allowable deviation interval $[0.96 - S_{global},\ 1.04 - S_{global}]$. For hierarchical aggregation protocol, we set $p = 1$, which we empirically found to achieve the best performance. The remaining hyperparameters governed the logging and pruning module configuration.

Implementation Details. Below, we summarize the precise configurations used for each selected baseline:

- MP (Han et al., 2015): The weight matrix of each module is sorted by absolute value, retaining the $top - k$ entries and zeroing out all others. For SSM modules, the same procedure is applied to the state-transition matrix A.
- Mamba-Shedder (Muñoz et al., 2025): We employed the authors' published implementation and default settings, without fine-tuning. Since the authors built upon the official Mamba model implementation and introduced their own modifications, we reproduced this baseline by employing the Mamba model definition as provided by the authors.
- SparseGPT (Frantar & Alistarh, 2023): We extended the original `SparseGPT` framework to support Mamba pruning via two key adaptations: (1) when pruning `nn.Conv1d` modules, We applied the SparseGPT processing pipeline for `transformer.Conv1d` directly to the `nn.Conv1d` modules; and (2) when pruning the SSM parameter matrix A, we enable direct matrix-level pruning and use the hidden state h as calibration data.

B.2 Additional Experiments Results

B.2.1 Pruning the Whole Mamba Architecture

At 50% unstructured sparsity over all trainable weights, *SparseSSM* consistently surpasses MP, Mamba-Shedder, and SparseGPT, yielding lower language–modeling perplexities on WikiText-2, PTB, and C4, and higher zero-shot accuracies on OBQA, PIQA, ARC-E, ARC-C, and Winogrande, see Table 6. These gains hold without any fine-tuning and narrow the pruned–vs–dense gap, indicating that our joint treatment of SSM and FFN branches produces robust, scale-transferable improvements.

B.2.2 Pruning Efficiency Analysis

Our proposed method can prune Mamba-based large language models in an extremely short time. Specifically, thanks to our efficient Hessian matrix estimation method and fully parallelized implementation, the time required to compute pruning scores is virtually negligible; the primary time overhead instead stems from processing the calibration data.

B.2.3 Pruning Different modules

In Section 3.4, we note that pruning different modules exerts heterogeneous effects on the overall performance of Mamba-based LLMs, with sensitivity varying markedly across modules. Specifically, pruning the `in_proj` module precipitates a precipitous decline in model performance, and pruning the `out_proj` module similarly induces significant degradation, whereas remaining modules demonstrate higher resilience to parameter removal.

Within the Mamba architecture, the `in_proj` and `out_proj` modules serve as the principal input projection and output transformation layers, respectively, endowing them with high coupling and low redundancy that limit their prunability. Conversely, other modules are characterized by extensive overparameterization, enabling redundant representations of analogous functionalities and yielding comparatively low Hessian curvature across their parameters.

Table 6: Performance analysis for one-shot unstructured pruning of the whole Mamba models (130M $\sim$ 1.4B) at 50% sparsity. Here, $\downarrow$ lower metrics reflect better outcomes, and $\uparrow$ denotes higher metrics reflect better outcomes.

Model	Method	Wiki. ↓	PTB ↓	C4 ↓	OBQA ↑	PIQA ↑	ARC-e ↑	ARC-c ↑	WinoG ↑	Avg. ↑
Mamba-130M	Dense	20.60	32.75	25.66	28.60	63.28	48.02	24.40	52.5	43.36
	MP (Han et al., 2015)	7.2e13	1.6e13	3.9e12	27.00	50.82	25.88	**27.47**	49.41	36.12
	Mamba-Shedder (Muñoz et al., 2025)	364.8	476.9	231.4	24.4	54.46	34.68	22.95	49.41	37.18
	SparseGPT (Frantar & Alistarh, 2023)	6.2e7	1.8e7	7.5e5	**27.40**	53.42	28.24	23.63	**52.49**	37.04
	SparseSSM	**59.17**	**100.9**	**68.60**	25.80	**58.54**	**39.31**	23.46	49.80	**39.38**
Mamba-370M	Dense	14.32	23.46	19.37	31.00	68.34	54.97	27.90	55.25	47.49
	MP (Han et al., 2015)	3.7e10	8.3e10	3.1e10	**30.40**	55.71	31.40	24.15	49.96	38.32
	Mamba-Shedder (Muñoz et al., 2025)	192.3	196.2	120.1	27.60	57.94	39.90	22.78	**52.33**	40.11
	SparseGPT (Frantar & Alistarh, 2023)	3.8e4	3.8e6	7717	28.60	59.14	42.00	**24.83**	**52.33**	41.38
	SparseSSM	**36.89**	**60.74**	**49.00**	30.00	**62.24**	**45.96**	24.49	52.01	**42.94**
Mamba-790M	Dense	11.96	18.45	16.62	33.80	72.63	61.07	29.44	56.27	50.64
	MP (Han et al., 2015)	6.6e57	4.5e53	2.5e46	26.40	54.24	28.11	**25.60**	48.54	36.58
	Mamba-Shedder (Muñoz et al., 2025)	121.9	150.9	112.7	25.80	57.78	38.47	22.61	49.64	38.86
	SparseGPT (Frantar & Alistarh, 2023)	201.1	361.35	156.31	**29.60**	62.62	46.17	25.00	51.86	43.05
	SparseSSM	**22.76**	**37.65**	**31.21**	29.00	**64.58**	**50.04**	25.51	**53.67**	**44.56**
Mamba-1.4B	Dense	10.75	17.05	15.17	36.40	73.88	65.57	32.85	61.17	53.98
	MP (Han et al., 2015)	468.4	743.2	198.5	30.60	**67.95**	53.28	24.06	52.49	45.68
	Mamba-Shedder (Muñoz et al., 2025)	83.70	122.3	81.35	24.20	59.41	42.21	22.87	51.70	40.08
	SparseGPT (Frantar & Alistarh, 2023)	59.16	95.14	55.09	**31.40**	67.74	53.03	24.66	54.70	46.30
	SparseSSM	**19.65**	**45.91**	**25.81**	30.80	66.10	**56.06**	**26.62**	**56.59**	**47.24**

Table 7: Performance analysis of pruning time overhead. Specifically, we conduct experiments on multiple model variants and across different calibration-data sample sizes.

Model	Layers	Hidden size	Nsample	Pruning time
Mamba-130M	24	768	32	164.4378 s
			64	311.3634 s
			128	624.6192 s
Mamba-370M	48	1024	32	319.0448 s
			64	602.5500 s
			128	1203.028 s
Mamba-790M	48	1536	32	326.1090 s
			64	630.0898 s
			128	1239.914 s
Mamba-1.4B	48	2048	32	348.4770 s
			64	662.2011 s
			128	1272.2396 s

Table 8: Performance analysis results for pruning different modules. In each row, the Module column denotes the component being pruned, with 50% sparsity applied to the Mamba-370M model.

Module	Wiki. ↓	PTB ↓	C4 ↓	OBQA ↑	PIQA ↑	ARC-e ↑	ARC-c ↑	WinoG ↑	Avg. ↑
`conv1d`	14.46	23.78	19.52	30.80	68.61	55.13	27.30	55.01	47.37
`in_proj`	16.28	27.23	22.68	30.40	66.43	51.56	26.88	55.25	46.10
`x_proj`	14.35	23.55	19.39	30.40	68.55	54.59	27.90	55.64	47.42
`dt_proj`	14.49	23.88	19.56	30.80	68.39	54.50	28.75	54.78	47.44
`out_proj`	15.19	25.45	21.47	31.00	66.87	54.08	27.56	56.12	47.13

B.2.4 Results of Speedup

As reported in Table 3, our pruning method yields consistent runtime acceleration across both parallel scan and recurrent execution modes. In the parallel scan setting, the measured improvements are

Table 9: Memory results of structured pruning across model sizes.

Model	Sparsity	Memory (parallel scan)	Memory (recurrent)
Mamba-130M	Dense	900.26	1195.1
	50%	896.88	1017.4
Mamba-370M	Dense	1826.2	2354.0
	50%	1817.2	1552.2
Mamba-790M	Dense	3444.6	4433.0
	50%	3428.7	2926.6
Mamba-1.4B	Dense	5651.4	7099.6
	50%	5642.1	4702.2
Mamba-2.8B	Dense	10982	12897
	50%	10958	6872.2

Table 10: FineWeb perplexity under one-shot unstructured pruning of SSM modules at 50% sparsity. Lower perplexity (↓) indicates better performance.

Method	FineWeb ppl. ↓			
	Mamba-130M	Mamba-370M	Mamba-790M	Mamba-1.4B
Dense	17.80	13.45	13.45	10.80
SparseGPT (Frantar & Alistarh, 2023)	6.5e6	4115	68.68	31.14
SparseSSM	**18.22**	**14.14**	**12.12**	**11.94**

modest (around 1.05–$1.09\times$) due to the already high parallel efficiency of dense operations. By contrast, in the recurrent mode, the state transition matrix A_{log} is invoked repeatedly. The speedup becomes much more pronounced, reaching up to $2.23\times$ on Mamba-2.8B. This observation highlights that even though A_{log} comprises a relatively small number of parameters, its repeated usage amplifies the benefits of pruning, especially in the absence of specialized CUDA kernels. Therefore, pruning directly translates into significant end-to-end runtime gains in practical inference scenarios.

All runtime experiments above were conducted on a single NVIDIA GeForce RTX 4090 GPU. We used `torch.profiler` to measure latency by directly benchmarking the pruned models, without relying on any additional acceleration frameworks.

B.2.5 Results of Memory Footprint

Table 9 reports the peak device memory before and after structured pruning across model scales and execution modes. In the *parallel scan* setting, the memory footprint changes marginally (typically $\leq 1\%$), reflecting that peak usage is dominated by activation buffers and fused kernel workspaces that are insensitive to weight sparsity. In stark contrast, the *recurrent* setting benefits substantially: pruning compacts the SSM state and associated working sets, yielding large reductions in peak memory, e.g., from $12{,}897$ MB to $6{,}872$ MB on Mamba-2.8B, approaching a near-$2\times$ decrease, with consistent 30–45% savings on medium and large models. These results indicate that, beyond compute savings, structured sparsity in A_{log} directly translates into materially lower memory headroom for recurrent inference, enabling larger batch sizes or longer contexts under fixed VRAM budgets.

B.2.6 Results of Additional SSM Pruning Experiments

To better demonstrate the robustness of our method, we additionally conduct more SSM pruning experiments. We ran additional experiments on a validation split of FineWeb (Penedo et al., 2024), treating it analogously to the first three perplexity-based benchmarks. We used exactly the same evaluation pipeline as in our existing LM evaluations to ensure a fair comparison. On this large-scale web corpus, we consistently observe that *SparseSSM* yields lower perplexity than the training-free baselines at the same sparsity levels, see Table 10. We also provide evaluation results in the long-

Table 11: Model performance at 50% sparsity on long-sequence inputs with Mamba-130M. Lower perplexity (↓) indicates better performance.

Method	Wiki. ↓	PTB ↓	C4 ↓
Dense	20.60	32.75	25.75
SparseGPT (Frantar & Alistarh, 2023)	792.9	556.2	249.4
SparseSSM	**21.34**	**33.89**	**26.46**

sequence setting, where we measure model performance with a context length of 4K tokens, as Table 11 shows.

To further assess the scalability of our approach, we evaluate it on the Mamba2-8B model (Waleffe et al., 2024), a large-scale checkpoint trained with the Megatron framework. The results in Table 12 confirm that our method remains effective at this larger model size.

Table 12: Performance analysis of the `mamba2-8b-3t-4k` checkpoint at 50% sparsity. Lower perplexity (↓) indicates better performance.

Method	Wiki. ↓	PTB ↓	C4 ↓	FineWeb ↓
Dense	7.25	11.26	9.96	7.91
SparseSSM	7.52	11.85	10.30	8.22

B.2.7 Results of Pruning Other SSM architecture

To test applicability beyond language, we apply *SparseSSM* to a vision SSM architecture, MambaVision (Hatamizadeh & Kautz, 2025), on the ImageNet-1K validation set. Using the same training-free pipeline, *SparseSSM* preserves top-1/top-5 accuracy almost perfectly and compares favorably to magnitude pruning at the same sparsity level (see Table 13), showing that our method extends naturally to a different modality and data distribution. To probe hybrid architectures, we validate our method on a small open-source checkpoint, Jamba-tiny-dev, with results reported in Table 14.

Table 13: Performance analysis on the Imagenet-1K validation set of the MambaVision-B checkpoint at 50% sparsity. Higher accuracy (↑) and lower error (↓) indicate better performance.

Method	Top-1 (%) ↑	Top-1 error (%) ↓	Top-5 (%) ↑	Top-5 error (%) ↓
Dense	84.68	15.31	97.34	2.66
Magnitude pruning	84.36	15.64	97.12	2.88
SparseSSM	**84.65**	**15.35**	**97.37**	**2.62**

Table 14: Performance analysis on the Jamba-tiny-dev checkpoint. Higher scores (↑) indicate better performance.

Method	OBQA ↑	PIQA ↑	ARC-e ↑	ARC-c ↑	WinoG ↑	Avg. ↑
Dense	25.4	**57.89**	**33.21**	17.49	50.04	36.81
SparseSSM	**25.8**	57.40	32.62	**18.01**	**50.36**	**36.84**

B.2.8 Ablation Study of Aggregation Method

We have explicitly compared power-law aggregation with uniform and exponential alternatives in our pruning experiments and found that the power-law choice consistently yields the best perplexity

Table 16: Performance analysis for one-shot unstructured pruning of SSM modules in Mamba models (130M $\sim$ 1.4B) at 40% sparsity. Here, $\downarrow$ lower metrics reflect better outcomes, and $\uparrow$ denotes higher metrics reflect better outcomes.

Model	Method	Wiki. ↓	PTB ↓	C4 ↓	OBQA ↑	PIQA ↑	ARC-e ↑	ARC-c ↑	WinoG ↑	Avg. ↑
Mamba-130M	Dense	20.60	32.75	25.66	28.60	63.28	48.02	24.40	52.5	43.36
	MP (Han et al., 2015)	218.7	304.86	107.77	28.20	60.72	40.57	23.29	**51.85**	40.93
	Mamba-Shedder (Muñoz et al., 2025)	275.3	506.6	222.8	25.00	55.11	34.89	22.10	49.72	37.37
	SparseGPT (Frantar & Alistarh, 2023)	165.0	211.3	87.22	28.80	59.96	40.66	**24.74**	50.43	40.92
	SparseSSM	**20.84**	**33.04**	**25.85**	**30.40**	**63.49**	**45.79**	24.49	50.75	**42.98**
Mamba-370M	Dense	14.32	23.46	19.37	31.00	68.34	54.97	27.90	55.25	47.49
	MP (Han et al., 2015)	149.8	264.8	70.17	31.00	65.89	51.22	25.77	51.85	45.15
	Mamba-Shedder (Muñoz et al., 2025)	195.5	310.6	137.9	26.20	56.80	30.60	22.10	49.64	37.07
	SparseGPT (Frantar & Alistarh, 2023)	2.8e4	4.6e6	6367	**31.80**	65.89	50.84	26.54	53.04	45.62
	SparseSSM	**14.52**	**23.85**	**19.58**	30.40	**69.10**	**55.35**	**27.90**	**54.78**	**47.50**
Mamba-790M	Dense	11.96	18.45	16.62	33.80	72.63	61.07	29.44	56.27	50.64
	MP (Han et al., 2015)	97.37	150.4	53.35	32.40	68.66	54.17	27.65	55.33	47.64
	Mamba-Shedder (Muñoz et al., 2025)	75.51	109.5	78.93	33.60	71.06	56.57	27.39	55.72	48.87
	SparseGPT (Frantar & Alistarh, 2023)	36.14	81.62	34.13	32.80	68.34	54.42	27.47	54.93	47.59
	SparseSSM	**12.15**	**18.82**	**16.81**	**34.00**	**70.13**	**57.11**	**29.95**	**55.88**	**49.41**
Mamba-1.4B	Dense	10.75	17.05	15.17	36.40	73.88	65.57	32.85	61.17	53.98
	MP (Han et al., 2015)	49.99	84.70	34.14	34.60	70.35	59.68	27.82	56.04	49.70
	Mamba-Shedder (Muñoz et al., 2025)	120.6	179.5	109.7	26.40	60.45	39.86	22.95	52.41	40.41
	SparseGPT (Frantar & Alistarh, 2023)	32.39	49.87	28.86	**36.20**	72.36	61.49	31.48	57.30	51.77
	SparseSSM	**11.24**	**17.94**	**15.78**	35.40	**73.01**	**63.30**	**32.08**	**57.38**	**52.24**

and zero-shot accuracy at matched sparsity levels. In our main experiments, we use an exponent of $p = 1.0$ for the power-law aggregation, which we found to be a simple and effective default. We also ran a sensitivity study around this choice, varying in a moderate range of $0.5 \sim 2.5$. The results in Table 15 show that *SparseSSM* is reasonably robust to the exact value of p. Perplexity and zero-shot accuracy change only marginally within this range, and the overall ranking against baselines remains unchanged. Small p and large p value lead to mildly worse performance.

Table 15: Ablation of temporal aggregation strategies and the exponent p for power-law aggregation on Mamba-370M at 50% sparsity. Here, $\downarrow$ lower metrics reflect better outcomes, and $\uparrow$ denotes higher metrics reflect better outcomes.

Method / p	Wiki. ↓	PTB ↓	C4 ↓	OBQA ↑	PIQA ↑	ARC-e ↑	ARC-c ↑	WinoG ↑	Avg. ↑
(a) Ablation of aggregation methods									
Linear aggregation	15.28	24.93	20.47	29.20	68.61	53.75	26.45	54.70	46.54
Exponential aggregation	15.16	24.78	20.35	30.00	**69.31**	**55.64**	27.13	54.85	47.39
Power-law aggregation (ours)	**15.04**	**24.63**	**20.13**	**30.20**	68.34	55.26	**27.90**	**55.64**	**47.47**
(b) Effect of exponent p in power-law aggregation									
0.5	15.12	24.78	20.31	29.60	68.66	55.01	26.71	55.01	47.00
1.0	**15.04**	**24.63**	20.13	30.20	68.34	55.26	**27.90**	**55.64**	47.47
1.5	15.06	24.64	**20.11**	**30.80**	69.04	55.77	27.22	54.93	**47.55**
2.0	15.17	24.81	20.28	30.20	**69.15**	**56.27**	27.22	54.70	47.51
2.5	15.30	25.15	20.45	30.00	68.66	55.81	26.96	54.93	47.27

B.2.9 Results of Pruning SSM Module at High Sparsity

We further compare our method against magnitude pruning (MP), Mamba-Shedder, and SparseGPT across a range of sparsity levels. The pruning results for these methods are reported on Mamba-130M, Mamba-370M, Mamba-790M, and Mamba-1.4B. We evaluate the perplexity of each pruned model on WikiText-2, PTB, and C4, and measure task accuracy on OpenBookQA, PIQA, ARC-Easy, ARC-Challenge, and Winogrande. As summarized in Table 16, 17, 18, 19, our approach consistently outperforms all baselines at every sparsity level, thereby demonstrating its robustness.

Table 17: Performance analysis for one-shot unstructured pruning of SSM modules in Mamba models (130M $\sim$ 1.4B) at 60% sparsity. Here, $\downarrow$ lower metrics reflect better outcomes, and $\uparrow$ denotes higher metrics reflect better outcomes.

Model	Method	Wiki. ↓	PTB ↓	C4 ↓	OBQA ↑	PIQA ↑	ARC-e ↑	ARC-c ↑	WinoG ↑	Avg. ↑
Mamba-130M	Dense	20.60	32.75	25.66	28.60	63.28	48.02	24.40	52.50	43.36
	MP (Han et al., 2015)	1034	1605	351.7	26.00	55.55	33.42	22.10	49.96	37.41
	Mamba-Shedder (Muñoz et al., 2025)	3219	4998	1503	25.80	54.46	29.00	23.72	50.04	36.60
	SparseGPT (Frantar & Alistarh, 2023)	5.0e4	1.4e4	2.4e4	26.20	52.45	26.85	23.55	49.80	35.77
	SparseSSM	**22.31**	**35.78**	**27.45**	**30.20**	**63.38**	**45.92**	**24.40**	**52.17**	**43.21**
Mamba-370M	Dense	14.32	23.46	19.37	31.00	68.34	54.97	27.90	55.25	47.49
	MP (Han et al., 2015)	386.2	747.6	141.6	26.40	58.05	38.64	21.59	49.64	38.86
	Mamba-Shedder (Muñoz et al., 2025)	463.3	561.6	307.0	25.00	54.03	28.91	23.63	49.72	36.26
	SparseGPT (Frantar & Alistarh, 2023)	360.2	1455	324.7	30.00	58.87	40.07	23.89	53.28	41.22
	SparseSSM	**16.44**	**26.47**	**21.62**	29.60	**68.06**	**55.05**	**26.54**	**53.99**	**46.65**
Mamba-790M	Dense	11.96	18.45	16.62	33.80	72.63	61.07	29.44	56.27	50.64
	MP (Han et al., 2015)	255.6	502.5	108.4	28.40	60.61	41.92	23.29	51.85	41.22
	Mamba-Shedder (Muñoz et al., 2025)	353.5	358.3	283.5	26.60	54.95	32.58	23.04	49.96	37.43
	SparseGPT (Frantar & Alistarh, 2023)	1033	3630	897.5	31.40	65.40	51.26	24.74	53.67	45.29
	SparseSSM	**13.24**	**21.21**	**17.96**	**31.80**	**70.29**	**55.01**	**28.67**	**55.41**	**48.24**
Mamba-1.4B	Dense	10.75	17.05	15.17	36.40	73.88	65.57	32.85	61.17	53.98
	MP (Han et al., 2015)	150.9	322.3	67.64	30.20	62.73	47.47	25.43	50.99	43.36
	Mamba-Shedder (Muñoz et al., 2025)	370.4	481.4	281.4	26.80	55.55	33.67	23.29	50.67	38.00
	SparseGPT (Frantar & Alistarh, 2023)	110.3	209.2	70.36	**34.60**	69.91	58.59	27.99	53.75	48.97
	SparseSSM	**13.51**	**21.10**	**18.34**	31.60	**70.95**	**58.80**	**29.44**	**54.70**	**49.10**

Table 18: Performance analysis for one-shot unstructured pruning of SSM modules in Mamba models (130M $\sim$ 1.4B) at 70% sparsity. Here, $\downarrow$ lower metrics reflect better outcomes, and $\uparrow$ denotes higher metrics reflect better outcomes.

Model	Method	Wiki. ↓	PTB ↓	C4 ↓	OBQA ↑	PIQA ↑	ARC-e ↑	ARC-c ↑	WinoG ↑	Avg. ↑
Mamba-130M	Dense	20.60	32.75	25.66	28.60	63.28	48.02	24.40	52.50	43.36
	MP (Han et al., 2015)	1248	1802	407.0	24.80	54.13	30.68	24.32	**52.49**	37.28
	Mamba-Shedder (Muñoz et al., 2025)	5845	1.2e4	3775	26.80	51.85	26.56	**24.57**	50.67	36.09
	SparseGPT (Frantar & Alistarh, 2023)	1.1e5	6.7e4	1.8e5	24.20	51.47	25.59	24.40	50.36	35.20
	SparseSSM	**25.71**	**40.20**	**30.75**	**29.00**	**62.40**	**44.61**	24.15	52.33	**42.50**
Mamba-370M	Dense	14.32	23.46	19.37	31.00	68.34	54.97	27.90	55.25	47.49
	MP (Han et al., 2015)	497.3	925.2	174.4	25.60	56.91	36.70	20.05	51.30	38.11
	Mamba-Shedder (Muñoz et al., 2025)	1029	933.0	625.7	26.80	52.67	27.95	23.63	50.28	36.26
	SparseGPT (Frantar & Alistarh, 2023)	7.8e4	5.5e4	7.3e4	27.80	59.30	39.23	22.61	50.36	39.86
	SparseSSM	**19.74**	**30.88**	**25.18**	**28.40**	**67.30**	**52.61**	**24.57**	**52.25**	**45.03**
Mamba-790M	Dense	11.96	18.45	16.62	33.80	72.63	61.07	29.44	56.27	50.64
	MP (Han et al., 2015)	341.2	655.9	139.6	27.00	57.83	38.85	24.57	51.22	39.90
	Mamba-Shedder (Muñoz et al., 2025)	353.5	358.3	283.5	26.60	54.95	32.58	23.04	49.96	37.43
	SparseGPT (Frantar & Alistarh, 2023)	1.9e5	2.4e7	2.7e5	27.60	61.32	39.10	24.49	52.96	41.09
	SparseSSM	**14.92**	**24.49**	**19.92**	**31.20**	**69.26**	**55.05**	**28.07**	**54.85**	**47.69**
Mamba-1.4B	Dense	10.75	17.05	15.17	36.40	73.88	65.57	32.85	61.17	53.98
	MP (Han et al., 2015)	180.8	378.5	80.16	28.80	59.58	41.67	23.55	51.07	40.93
	Mamba-Shedder (Muñoz et al., 2025)	805.1	796.6	541.7	25.40	54.08	29.50	24.06	49.09	36.43
	SparseGPT (Frantar & Alistarh, 2023)	452.5	602.9	253.7	**31.20**	66.27	54.00	24.40	50.36	45.24
	SparseSSM	**17.91**	**28.14**	**23.72**	29.20	**68.72**	**54.04**	**26.96**	**53.75**	**46.53**

C Further Discuss

Limitations. Our proposed method represents the first work to extend the OBS framework to Mamba-based LLMs. While it can be naturally extended to structured pruning of the SSM module, further work is required to develop a one-shot, second-order information–based structured pruning strategy that effectively accelerates the entire model. In our preliminary structured-pruning extension, we achieved a 1.72× speed-up on the SSM module, yet the end-to-end inference acceleration of the full model remains modest. Moreover, since our experiments were conducted on open-source Mamba model series, their deployment may inherently entail ethical and safety risks.

Broader Impact. Our proposed method effectively reduces parameter redundancy in Mamba-based LLMs, yielding a leaner network representation that requires fewer floating-point operations during inference. As a result, these pruned models can be deployed with lower computational cost, both in terms of GPU hours and energy consumption, thereby democratizing access to state-of-the-art LLM

Table 19: Performance analysis for one-shot unstructured pruning of SSM modules in Mamba models (130M $\sim$ 1.4B) at 80% sparsity. Here, $\downarrow$ lower metrics reflect better outcomes, and $\uparrow$ denotes higher metrics reflect better outcomes.

Model	Method	Wiki. ↓	PTB ↓	C4 ↓	OBQA ↑	PIQA ↑	ARC-e ↑	ARC-c ↑	WinoG ↑	Avg. ↑
Mamba-130M	Dense	20.60	32.75	25.66	28.60	63.28	48.02	24.40	52.50	43.36
	MP (Han et al., 2015)	1297	1870	420.4	24.00	52.18	31.31	**24.32**	50.51	36.46
	Mamba-Shedder (Muñoz et al., 2025)	2.6e4	5.9e4	2.2e4	26.20	51.69	28.03	23.89	52.01	36.36
	SparseGPT (Frantar & Alistarh, 2023)	2.6e21	5.7e22	2.7e23	24.80	55.98	30.60	23.38	51.30	37.21
	SparseSSM	**38.61**	**55.81**	**41.88**	**27.80**	**61.32**	**42.59**	23.04	**52.09**	**41.37**
Mamba-370M	Dense	14.32	23.46	19.37	31.00	68.34	54.97	27.90	55.25	47.49
	MP (Han et al., 2015)	538.2	983.0	191.0	25.20	53.16	31.99	22.61	49.49	36.49
	Mamba-Shedder (Muñoz et al., 2025)	3191	933.0	1848	**27.80**	52.34	26.52	**24.06**	51.14	36.37
	SparseGPT (Frantar & Alistarh, 2023)	1.1e5	1.2e5	1.0e5	27.40	56.26	34.93	23.38	**53.83**	39.16
	SparseSSM	**28.99**	**40.74**	**34.45**	27.40	**65.72**	**49.12**	23.72	51.93	**43.58**
Mamba-790M	Dense	11.96	18.45	16.62	33.80	72.63	61.07	29.44	56.27	50.64
	MP (Han et al., 2015)	402.7	738.5	160.1	25.80	56.80	36.66	22.70	49.41	38.27
	Mamba-Shedder (Muñoz et al., 2025)	1891	2121	1277	25.40	51.69	28.28	24.40	48.15	35.58
	SparseGPT (Frantar & Alistarh, 2023)	1.7e8	4.3e8	2.1e8	**27.40**	56.80	36.70	23.12	50.91	38.99
	SparseSSM	**19.92**	**31.09**	**25.19**	**27.40**	**67.14**	**54.29**	**27.73**	**52.49**	**45.81**
Mamba-1.4B	Dense	10.75	17.05	15.17	36.40	73.88	65.57	32.85	61.17	53.98
	MP (Han et al., 2015)	227.4	438.6	101.4	25.00	56.04	34.81	22.61	**53.12**	38.31
	Mamba-Shedder (Muñoz et al., 2025)	2260	2236	1405	26.80	51.20	28.87	**27.13**	51.14	37.03
	SparseGPT (Frantar & Alistarh, 2023)	5.7e11	2.6e13	3.1e14	**28.20**	59.36	43.22	23.38	48.93	40.62
	SparseSSM	**29.79**	**58.01**	**36.38**	26.60	**64.42**	**48.32**	24.15	49.72	**42.64**

capabilities for academic, industrial, and edge computing environments. Moreover, by curtailing the extensive resource demands traditionally associated with LLM inference, our approach contributes to a reduction in the cumulative electricity usage and associated carbon emissions of LLM workloads. In doing so, it supports the broader agenda of sustainable AI by mitigating the environmental and climate impacts of deploying LLMs at scale.

The Use of Large Language Models. We used a commercial large language model strictly as a general-purpose assistant for surface-level editing and figure preparation. Specifically, it was employed to polish the grammar and phrasing of author-written text, and to draft plotting templates for visualizing results from our own data. All technical content, claims, and analyses were authored by the authors. All suggestions and scripts produced by the LLM were manually reviewed, edited, and independently verified; figures were regenerated from raw logs by the authors. The model had no access to confidential artifacts beyond aggregated numbers required for visualization.