SPARSEFORGE: EFFICIENT SEMI-STRUCTURED LLM SPARSIFICATION VIA ANNEALING OF HESSIAN-GUIDED SOFT-MASK

Anonymous authors
Paper under double-blind review

ABSTRACT

Semi-structured sparsity provides a practical path to accelerate large language models (LLMs) with native hardware support, but enforcing grouped sparsity constraints can substantially degrade model quality. Recovering this loss through extended sparse retraining incurs significant computational cost, motivating methods that achieve better recovery with fewer retraining tokens. We propose SparseForge, a post-training framework that jointly optimizes model weights and sparsity masks rather than relying solely on longer retraining. SparseForge combines Hessian-aware groupwise importance estimation with progressive soft-mask annealing, allowing the sparse support to adapt during recovery before final projection into a binary 2:4 pattern. On LLaMA-2-7B, SparseForge achieves **57.27%** average zero-shot accuracy on the seven-task CAST-7 benchmark using **5B** retraining tokens. Compared with prior SOTA, CAST, which achieves 54.37% with 7.86B tokens, this represents a **2.90%** accuracy improvement with approximately **36.4%** fewer retraining tokens. Evaluations across multiple model families further examine the applicability of the proposed recovery framework. These results highlight curvature-guided mask optimization and progressive hardening as a promising route to improving the accuracy–retraining-token trade-off. Our code is available at https://anonymous.4open.science/r/SparseForge-Anonymous.

1 INTRODUCTION

Semi-structured sparsity stands out as one of the few sparsity paradigms that deliver both meaningful compression and tangible hardware acceleration for large language models (LLMs). In particular, patterns such as 2:4 sparsity are natively supported by modern NVIDIA Sparse Tensor Core pipelines, making them attractive for practical deployment [22]. However, in post-training LLM sparsification, achieving high accuracy under such structured constraints typically requires substantial retraining, leading to a nontrivial trade-off between performance and training cost.

Existing post-training pruning methods often rely on computing importance scores for weights followed by hard projection to sparse structures [9, 28, 13, 14]. While effective in simpler settings, this paradigm becomes brittle under semi-structured constraints, where pruning decisions are no longer independent. Instead, weights compete within each constrained group, and premature hard decisions can remove weights that are individually important but suboptimal under imperfect scoring. As a result, these approaches either suffer from noticeable accuracy degradation or require large amounts of retraining to recover performance.

We argue that this challenge is fundamentally rooted in *how sparsity masks are learned*. Rather than committing to a discrete mask prematurely, semi-structured sparsification should treat the mask as a continuous and optimizable variable, allowing the model to explore structured alternatives before making irreversible pruning decisions. More importantly, we argue that improving mask optimization quality can reduce reliance on large-scale retraining, offering a more efficient path to sparse recovery. Adopting a continuous mask formulation, however, introduces a new challenge: at inference time the mask must be an exact, hardware-executable binary pattern, creating a gap between the learned soft mask and its deployment-time hard counterpart. In practice, naive soft-mask approaches often lead to

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

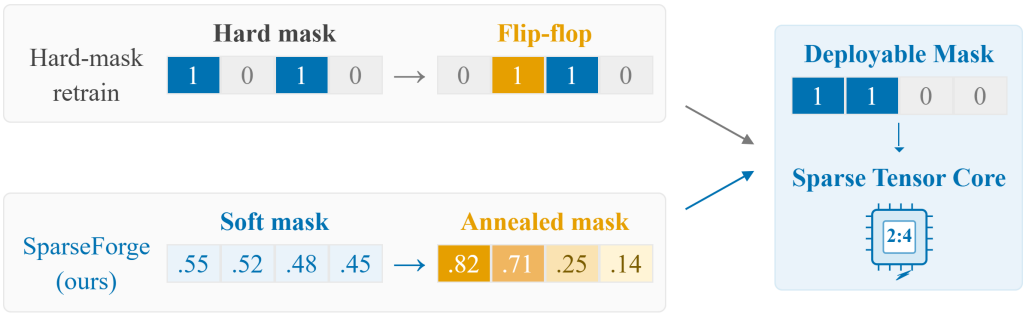

Figure 1: Compared with hard-mask retraining, SparseForge explicitly optimizes a soft mask and progressively anneals it into a deployable binary 2:4 mask.

performance degradation when the mask is binarized, due to insufficient separation between mask values.

Based on this perspective, we propose **SparseForge** (Figure 1), a post-training framework for semi-structured sparsification that leverages *Hessian-aware importance* guideline to derive a *soft mask*, and progressively hardens it by gradually shaping from continuous exploratory values into deployable binary sparse structures. SparseForge improves mask optimization quality via three key techniques: (1) continuous soft-mask optimization for stable structured exploration; (2) Hessian-aware importance estimation to better capture deletion sensitivity under grouped competition; and (3) a progressive annealing mechanism that gradually aligns soft masks with hardware-executable binary patterns. With improved mask quality, SparseForge largely reduces retraining tokens and hence achieves efficient sparse recovery. The main contributions of this work are as follows:

- We identify that under semi-structured constraints, sparse recovery is bottlenecked more by mask quality than by retraining scale. This insight motivates us to prioritize optimizing the mask itself and to keep it soft during retraining, which leads to the proposed *mask annealing* pipeline that efficiently navigates the soft-to-hard transition.
- We present SparseForge, a post-training framework for semi-structured sparsification that explicitly optimizes the mask itself. SparseForge features a Hessian-aware mask annealing algorithm, which smoothly trains the soft mask towards a hardware-executable binary pattern.
- We demonstrate that SparseForge achieves *higher accuracy with fewer retraining tokens*. On LLaMA-2-7B, SparseForge attains 57.27% average zero-shot accuracy on CAST-7 using 5B retraining tokens, compared with 54.37% using 7.86B tokens of CAST [14]. This corresponds to a 2.90-percentage-point improvement with approximately 36.4% fewer retraining tokens. Ablation studies further examine the contributions of the key components.

2 Related Work

Hardware-aligned sparsity patterns. Structured sparsity has long been pursued as a practical route to efficient neural network acceleration, since regular sparse patterns are far easier for hardware and kernels to exploit than irregular ones [31, 18, 19, 10, 21]. Early work explored filter-, channel-, and block-level pruning for real execution benefits. More recently, semi-structured patterns such as 2:4 sparsity have drawn significant attention due to native GPU support [22]. While offering clear deployment gains, such patterns impose tightly coupled local constraints—selecting exactly N survivors within each group of M (in practice N=2, M=4, i.e., 2:4 sparsity)—which makes their accuracy impact more substantial than flexible, unstructured cases.

Unstructured sparsification for LLMs. A large body of recent work studies post-training sparsification for LLMs in the unstructured setting. Building on classical magnitude-based pruning [11], representative methods such as SparseGPT [9] and Wanda [28] estimate weight importance via approximate second-order information or activation statistics, removing parameters with little or no

retraining. OWL [34] further shows that outlier-weighed, layerwise sparsity allocation is critical for pushing LLMs to high sparsity. BESA [32] introduces a blockwise sparsity allocation strategy, but its goal is still not hardware-aligned semi-structured execution. However, unstructured sparsity does not translate into efficient execution on current accelerators, so it does not resolve the deployment problem targeted by hardware-aligned patterns.

Structured sparsification for LLMs. Recent efforts therefore turn to structured or semi-structured sparsification for LLMs. MaskLLM [8] cast $N{:}M$ mask selection as a learnable discrete distribution optimized via Gumbel-Softmax sampling on large-scale corpora, while AST [13] followed a dense-to-sparse adaptive sparse training scheme that progressively annealed the weights toward the 2:4 pattern. More recently, CAST [14] relaxed the hard 2:4 constraint into a continuous, differentiable form so that masks could be optimized jointly with weights. These works show that better mask optimization can substantially narrow the gap of hardware-executable sparse models, yet they all rely on training over a large number of tokens for the mask to converge to a high-quality solution. Our work instead targets *recovery efficiency* by improving mask optimization quality itself, attaining strong recovery with considerably fewer retraining tokens.

3 Motivation

3.1 Rethinking the Bottleneck of Sparse Recovery: Mask Quality over Retraining Scale

As discussed in §2, previous methods such as CAST [14] preserve quality by scaling retraining tokens, which yields gains but with poor efficiency. Observations from CAST show that even with substantial retraining, accuracy is still governed by the selected sparse pattern, implying that *mask quality, rather than retraining scale alone, is the key bottleneck.* We instead choose to directly improve the quality of mask optimization along three axes: (1) keeping the mask soft long enough to avoid brittle early hard decisions, (2) using Hessian-aware importance to resolve intra-group competition, and (3) progressively annealing the mask into the final hardware-executable 2:4 pattern.

3.2 Soft Masks are Necessary under Grouped Semi-structured Constraints

Under semi-structured constraints such as 2:4 sparsity, pruning is no longer an element-wise decision but a grouped selection problem in which weights compete within each constrained block. Committing to a binary mask too early therefore locks the optimizer into a suboptimal sparse subspace before intra-group importance is sufficiently resolved. This effect is also reflected empirically in Figure 3(a): on the 7-task zero-shot benchmark, SparseForge achieves 59.20% mean accuracy, outperforming the hard-mask-style AST baseline at 58.62% [13]. This supports the need to *keep the mask soft* during optimization.

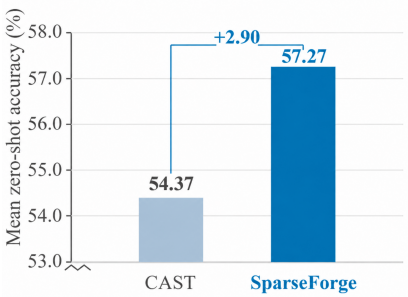

Figure 2: Soft masks improve sparse recovery: compared with the hard-mask-style AST baseline, our soft-mask-style SparseForge improves the mean zero-shot accuracy on the 7-task benchmark from 58.62% to 59.20%, supporting the need for soft mask optimization.

However, a soft mask must eventually be converted into a deployable binary (0/1) form. Combining the two sides, we propose a pipeline that we call ***mask annealing*** in this paper: first, we optimize the mask in a soft, continuous state to allow extensive exploration, analogous to heating metal into a malleable condition (**Heating**); then, we binarize the mask into the hardware-ready pattern, analogous to cooling it into a fixed shape (**Quenching**).

3.3 Insights to Implement Efficient Mask Annealing

With mask annealing, we appear to achieve the best of both worlds: the exploration flexibility of a soft mask and the deployment readiness of a hard mask. Yet, translating this idea into practice requires several key insights, which we detail below.

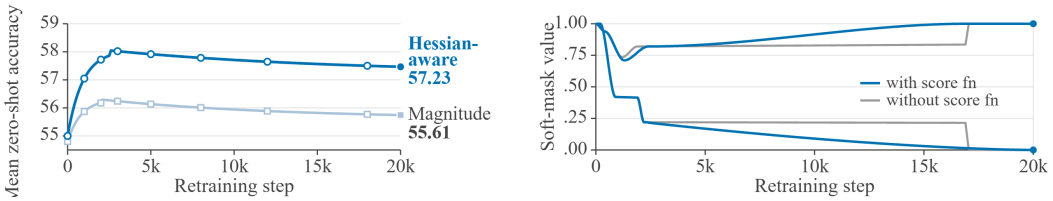

Figure 3: **(a)** Hessian-aware importance provides a better survival signal: replacing it with magnitude-based scoring drops mean zero-shot accuracy from 57.23% to 55.61%. **(b)** Soft masks must also be progressively hardened toward a deployable 2:4 pattern: score structural hardening pushes the top-2 and bottom-2 entries in each group towards 1 and 0, respectively; without such hardening, the mask remains far from binary, leading to a larger soft-to-hard gap at final projection.

(I1) Using second-order Hessian-aware loss signal. Even with a soft mask, the optimizer still needs to decide which entries should survive within each constrained group. A useful intuition comes from the local loss change incurred by removing a weight:

$$\Delta L = L(w_i = 0) - L(w_i) \approx \frac{\partial L}{\partial w_i}(-w_i) + \frac{1}{2}\frac{\partial^2 L}{\partial w_i^2} w_i^2$$

Near a converged solution, the first-order gradient term is typically negligible, so removal sensitivity is dominated by the local curvature term. Under grouped competition, magnitude alone is therefore often insufficient to determine which entries should be preserved. Figure 3(a) directly supports this point: using magnitude-based scoring to replace Hessian-aware importance would drop the mean zero-shot accuracy from 57.23% to 55.61%. This is also consistent with classical second-order pruning criteria [17, 12, 27] and modern LLM pruning methods such as SparseGPT [9].

(I2) The quenching process needs to be progressive. Under 2:4 sparsity, the two surviving entries in each group should approach 1, while the two pruned ones should approach 0. If many mask values remain in the middle of $[0, 1]$ in the late stages during training, the final quenching step must abruptly binarize them, causing a large projection error. This motivates us to optimize soft mask towards a near-binary structured solution during the heating stage, rather than relying on a late hard projection alone in the quenching stage. As in Figure 3(b), with a score function explicitly designed to encourage the mask toward a near-binary state, retraining steadily pushes the entries toward 0/1.

4 Methodology

Building on the insights from §3, we design **SparseForge**, a LLM sparsification framework that efficiently implements the mask annealing pipeline. Figure 4 provides an overview. In §4.1, we present the overall retraining loop of the heating stage, describing how the soft mask is optimized alongside model weights. §4.2 then addresses (I1) by detailing our second-order mask update rule, while §4.3 tackles (I2) and explains how progressive quenching bridges the soft-to-hard gap.

4.1 Dual-track Retraining Loop in the Heating Stage

During the heating stage, SparseForge jointly optimizes the model weights and the soft mask in a unified retraining loop. The weight update follows the paradigm established by prior sparse-training methods such as AST and CAST [13, 14]. Given a linear layer

$$Z = XW^T, \qquad X \in \mathbb{R}^{N\times C},\ W \in \mathbb{R}^{D\times C},\ Z \in \mathbb{R}^{N\times D},$$

a sparse mask $m \in \{0,1\}^{D\times C} \cap \mathcal{M}$ is applied in the forward pass by writing $\tilde{W} = m \odot W$, where $\mathcal{M}$ is the semi-structured feasible set (e.g., 2:4 or block-16). The sparse forward pass, optionally equipped with a sparse low-rank branch (SLoRB) [13], is

$$\hat{Y} = X(\tilde{W}^T + P^T B^T), \tag{1}$$

where P is the fixed projection and B is a learnable low-rank parameter used to compensate for capacity lost during pruning. Our retraining objective builds upon prior work and introduces a

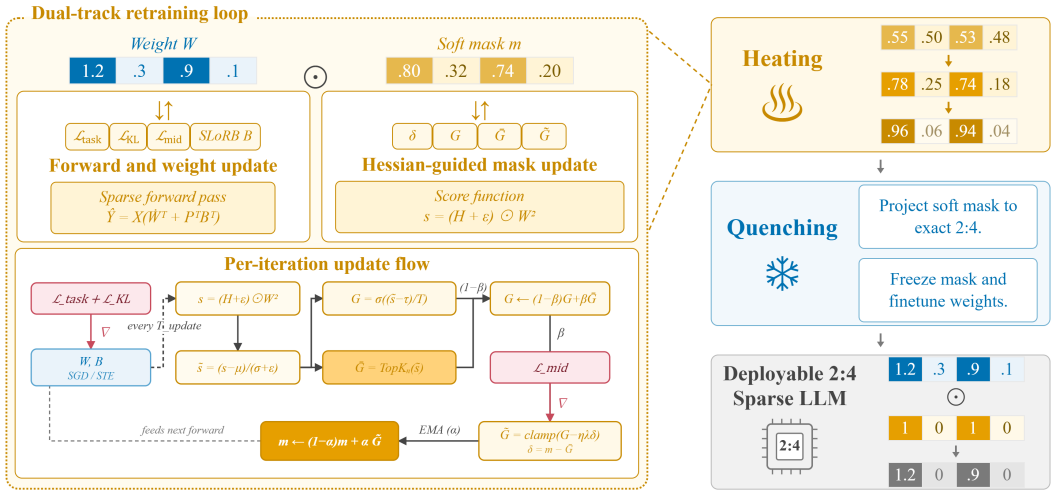

Figure 4: **Overview of SparseForge.** We first co-optimize the weights and the explicit learnable soft mask in a dual-track retraining loop (§4.1) in the heating stage, where the mask is updated with a Hessian-guided signal (§4.2). Then we progressively binarize the mask to a hard, deployable form in the quenching stage (§4.3).

structure-aware guidance term that facilitates progressive quenching:

$$\mathcal{L} = \lambda_{\text{task}}\mathcal{L}_{\text{task}} + \lambda_{\text{KL}}\mathcal{L}_{\text{KL}} + \lambda_{\text{mid}}\mathcal{L}_{\text{mid}}, \tag{2}$$

where $\mathcal{L}_{\text{task}}$ and $\mathcal{L}_{\text{KL}}$ inherit the standard task and distillation losses from prior work, and $\mathcal{L}_{\text{mid}}$ extends the objective with a binary-preference regularizer that we design for the mask (§4.3).

Besides weight optimization, SparseForge performs explicit soft-mask learning. In prior works, masks are typically treated as hard selections derived from one-shot importance scores or updated through discrete re-selection steps, rather than being continuously optimized as learnable variables. SparseForge instead separates mask learning from weight adaptation and makes the soft mask the central object of optimization.

Specifically, we optimize a continuous mask $m \in [0,1]^{D\times C}$ during retraining and only project it to a hardware-executable binary pattern at the end. Within the same unified loop, the optimization proceeds along two interleaved tracks: the weights W and the low-rank B follow the previous paradigm, updated by the loss at every step, while the soft mask m is updated by its own dedicated learning rule (§4.2) only every T_{update} steps. This decoupled schedule confines sparsification entirely to the mask space, where optimization remains more stable under grouped semi-structured constraints. We briefly sketch this *dual-track retraining loop* in Algorithm 1. The following sections elaborate the details.

4.2 Hessian-Guided Soft-Mask Update

Following (I1) (§3.3), we derive the survival signal for mask updates from a *Hessian-aware* criterion, which yields the classic OBD-style importance score [17]:

$$s = (H + \epsilon) \odot W^2, \qquad s \in \mathbb{R}^{D\times C},$$

where s_{ij} measures the Hessian-aware pruning sensitivity of weight W_{ij}, $H \in \mathbb{R}^{D\times C}$ denotes an approximate Hessian diagonal reshaped to the weight matrix, and ϵ is a small stabilizer.

Since explicitly forming the Hessian is prohibitively expensive for LLMs, we estimate its diagonal with a Hutchinson-style stochastic estimator [15]:

$$\mathbb{E}[(Hv) \odot v] = \text{diag}(H), \qquad v \sim \text{Rademacher}(\pm 1),$$

and maintain an exponential moving average of $(Hv) \odot v$ in practice.

Algorithm 1 SparseForge's Dual-track Retraining Loop

Require: Dense weights W, semi-structured constraint $\mathcal{M}$, training data $\mathcal{D}$, mask update duration T_{update}

Initialize soft mask $m \leftarrow \mathbf{1}$; initialize temperature $T \leftarrow T_0$ and structural-mix factor $\beta \leftarrow 0$
Initialize optional SLoRB branch (P, B)
for $t = 1$ to $T_{\max}$ **do**
  Sample batch $(X, Y) \sim \mathcal{D}$
  Compute sparse forward pass with current soft mask m
  Update W (via STE) and B using $\lambda_{\text{task}}\mathcal{L}_{\text{task}} + \lambda_{\text{KL}}\mathcal{L}_{\text{KL}}$ {$\mathcal{L}_{\text{mid}}$ enters through (4), not autograd}
  if $t \bmod T_{\text{update}} = 0$ **then**
    Compute importance score $s = (H + \epsilon) \odot W^2$; standardize to $\tilde{s}$
    Synthesize soft gate $G = \sigma((\tilde{s} - \tau)/T)$ with group-wise threshold τ, and hard target $\bar{G} = \text{TopK}_N(\tilde{s})$ on $\mathcal{M}$
    Blend in structure: $G \leftarrow (1 - \beta)\, G + \beta\, \bar{G}$
    Inject mid-penalty: $\tilde{G} \leftarrow \text{clamp}\big(G - \eta_{\text{pen}}\lambda_{\text{mid}}(m - \bar{G}),\, 0,\, 1\big)$
    Update soft mask: $m \leftarrow (1 - \alpha) \cdot m + \alpha \cdot \tilde{G}$
    Anneal schedules: $T \leftarrow \gamma T$; advance β, λ_{mid}
  end if
end for
return Near-hard mask m and the trained model

To respect the final semi-structured constraint while keeping the mask continuous, we turn s into a soft group-wise gate rather than directly projecting it onto the feasible set. For an $N{:}M$ pattern, let $\text{Groups}(M)$ denote the collection of non-overlapping groups, each containing M mask entries along the structured sparsity dimension. For a group $\mathcal{G}$, $s_{\mathcal{G}}$ denotes the restriction of the score matrix s to the entries in $\mathcal{G}$. We first standardize the scores layer-wise so that the soft threshold has a stable physical scale, $\tilde{s} \leftarrow (s - \mu_s)/(\sigma_s + \epsilon)$, and then define, within each group, a soft gate and its hard top-N counterpart:

$$G_{\mathcal{G}} \;=\; \sigma\!\left(\frac{\tilde{s}_{\mathcal{G}} - \tau_{\mathcal{G}}}{T}\right), \qquad \bar{G}_{\mathcal{G}} \;=\; \text{TopK}_N(\tilde{s}_{\mathcal{G}}), \qquad \forall\, \mathcal{G} \in \text{Groups}(M), \tag{3}$$

where $\tau_{\mathcal{G}}$ is the N-th largest entry of $\tilde{s}_{\mathcal{G}}$, σ is the sigmoid, and $T > 0$ is a temperature. The soft gate G already encodes the N:M competition: the N largest entries in each group sit above 0.5 and the remaining $M - N$ sit below, with the decisiveness controlled by T. The hard counterpart $\bar{G} \in \{0, 1\}^{D \times C}$ is kept on the side as a curvature-guided structured target that we will mix into G progressively (§4.3).

Directly overwriting the soft mask with $\bar{G}$ would destroy mask continuity and reintroduce premature hard-projection instability. We therefore use $\bar{G}$ only as an update direction, and push G toward it through the mid-penalty residual $\delta = m - \bar{G}$:

$$\tilde{G} \;=\; \text{clamp}(G - \eta_{\text{pen}}\, \lambda_{\text{mid}}\, \delta,\; 0,\; 1)\,, \tag{4}$$

so that entries that the structured target wants to keep ($\bar{G}_{ij} = 1$) but the current mask has dropped (m_{ij} small) are nudged up, and vice versa. This is precisely the gradient of the binary-preference regularizer $\mathcal{L}_{\text{mid}}$ (§4.3) evaluated against the structured target, and it is the only channel through which $\mathcal{L}_{\text{mid}}$ influences the mask. Finally, we update the soft mask with an exponential moving average:

$$m \;\leftarrow\; (1 - \alpha) \cdot m \;+\; \alpha \cdot \tilde{G}. \tag{5}$$

This rule preserves soft-mask continuity while moving the mask toward entries that are important under the Hessian-aware loss signal and already compatible with the final N:M structure.

4.3 Progressive Soft-to-Hard Mask Quenching

(I2) implies that an abrupt projection from soft to binary mask is harmful. We therefore introduce *progressive quenching*, a strategy that incorporates structure-aware regularization and penalty

terms throughout the heating stage, encouraging the mask to progressively approach a near-binary configuration before the final hardening step.

First, we introduce a binary-preference regularizer that penalizes ambiguous mask values:

$$\mathcal{L}_{\text{mid}} = \frac{1}{|m|} \sum_{i,j} m_{ij}(1 - m_{ij}).$$

This term is maximized at $m_{ij} = 0.5$ and minimized at $m_{ij} \in \{0, 1\}$. We anneal its weight λ_{mid} during training, keeping the binary preference weak during early exploration and progressively strengthening it as the mask approaches the final deployment stage.

Second, to inject structure awareness during training, we progressively blend the soft gate G from (3) with its Hessian-guided structured target $\bar{G}$ before the mid-penalty injection of (4):

$$G \;\leftarrow\; (1 - \beta)\, G \;+\; \beta\, \bar{G}, \qquad \beta : 0 \to 1.$$

We schedule β with a smooth-step (cubic Hermite) curve so that the update gradually shifts from soft exploration to structured hardening without a discontinuity in the gradient of the schedule itself. In parallel, we geometrically anneal the sigmoid temperature in (3), $T \leftarrow \gamma\, T$ with $\gamma \in (0, 1)$ applied at every mask-update step, so that the soft threshold sharpens monotonically as retraining proceeds. Among the three handles, the interpolation controlled by β is the primary driver: it continuously morphs the gate from the soft, exploratory G into the hard, group-structured target $\bar{G}$, while T sharpens the decisiveness of G in sync with this interpolation, and λ_{mid} adds a mild element-wise pressure that pushes any residual mid-range values in m toward $\{0, 1\}$. Consequently, the mask approaches a near-binary state while also aligning with the correct top-N pattern within each group.

Once the soft mask is sufficiently separated, the final quenching step interpolates between the soft mask and its hard version:

$$m_{\text{eff}} = x\, m + (1 - x)\, \mathbf{1}[m > \theta],$$

where x is annealed linearly from 1 to 0 over a dedicated hardening window. Once x reaches 0, we apply the final structured projection, freeze the binary mask, and run a short finetuning stage to absorb the remaining projection error.

5 Experiments

5.1 Experimental Setup

Models. We evaluate SparseForge on a diverse set of pretrained language models spanning different architectures and scales, including GPT-2 [24], OPT [36], LLaMA-2 [29], Qwen3 [33], and DeepSeek-MoE [6]. The evaluated model sizes range from 124M to 16B parameters, covering both dense transformer models and mixture-of-experts (MoE) architectures.

Hardware setting. All experiments are conducted on a cluster of 32 NVIDIA L20A GPUs. Training is implemented with distributed data parallelism. For LLaMA-2-7B, the full SparseForge training with 5B tokens requires approximately 50 GPU hours. Unless otherwise stated, all experiments target 2:4 semi-structured sparsity, i.e., 50% sparsity within every group of four weights. This sparsity pattern is directly supported by modern sparse Tensor Core hardware [22], making it a practically relevant deployment setting.

Training setup. For LLaMA-2-7B, we perform sparse retraining on Dolmino-mix-1124 [23, 1], following the data configuration adopted in recent continual pretraining and sparse recovery studies. For all other models, we use the C4 corpus [25, 7]. Unless otherwise specified, we use a total retraining budget of approximately 5B tokens for LLaMA-2-7B and 1.25B tokens for other models, substantially smaller than prior retraining-based methods such as AST [13] and CAST [14]. Unless otherwise specified, each SparseForge configuration is run with three random seeds. We report the mean performance across the three runs.

Evaluation metrics. We report WikiText-2 perplexity (PPL) and zero-shot accuracy on downstream tasks evaluated with `lm-evaluation-harness` [2]. For the cross-model comparison in Table 1,

Table 1: Compact cross-model summary under 2:4 semi-structured sparsity. We report mean zero-shot accuracy (%) for the dense model, SparseForge, and their difference. Detailed task-level results are provided in Appendix Table 3.

Metric	GPT2-M	GPT2-L	GPT2-XL	OPT-2.7B	Qwen3-1.7B	Qwen3-8B	Qwen3-14B	DeepSeek-MoE
Dense	40.97	42.76	45.49	47.76	56.51	65.73	68.36	59.54
SparseForge	40.31	42.10	44.34	46.67	53.33	63.31	65.44	58.57
Diff	-0.66	-0.66	-1.15	-1.09	-3.18	-2.42	-2.93	-0.97

we use a benchmark suite that is consistent across model families and remains aligned with the reporting protocol of recent semi-structured sparsification work. This suite includes ARC-Challenge and ARC-Easy [5], BoolQ [4], HellaSwag [35], OpenBookQA [20], RTE [30], and WinoGrande [26].

For the detailed LLaMA-2-7B comparison in Table 2, we follow the benchmark configuration of CAST [14], adding RACE [16] and PIQA [3] to the above suite. We report mean accuracy over the corresponding task set in each table.

5.2 Results Across Model Families

To facilitate consistent comparison across architectures while staying aligned with recent semi-structured sparsification practice, we summarize the cross-model results in Table 1, while detailed task-level results are deferred to Appendix Table 3. SparseForge preserves model quality across diverse architectures and scales: although degradation varies across families, the drops in mean accuracy remain moderate in most cases, showing that continuous mask optimization with progressive quenching is a robust recipe for semi-structured sparsification beyond a single model family.

Table 2: Comparison on LLaMA-2-7B. Task-level scores and Average use plain accuracy on CAST-7. AST (our eval.) denotes our re-evaluation of the official checkpoint; CAST (our paper-based reimplementation) denotes our implementation based on the paper. For these two rows, WikiText-2 perplexity is evaluated at context length 4096. Other rows retain the original draft results; SparseForge entries remain pending sparsity/export verification. † denotes a longer-budget continual-training variant.

Method	Pattern	Tokens	HellaS.	RACE	PIQA	WinoG.	ARC-e	ARC-c	OBQA	Average	Wiki PPL
Dense	Dense	2T	57.18	39.52	78.07	69.06	76.26	43.52	31.40	56.43	5.12
Wanda	2:4	×	41.05	35.02	70.78	62.67	61.99	27.56	22.80	45.98	11.29
SparseGPT	2:4	×	43.36	36.84	71.38	63.69	62.84	29.18	22.80	47.16	10.42
ALPS (our eval.)	2:4	–	46.44	38.37	73.52	65.75	65.61	32.37	26.60	49.81	10.33
ELSA (our eval.)	2:4	–	50.81	38.79	75.43	65.56	67.26	34.84	26.73	51.35	7.98
ProxSparse (our eval.)	2:4‡	–	47.84	35.41	71.87	63.14	65.78	33.11	24.20	48.76	9.30
MaskLLM	2:4	2B	50.91	40.77	74.92	64.48	69.57	36.00	28.00	52.09	6.72
Naive Retrain	2:4	10B	53.90	38.28	76.61	68.27	75.21	41.21	29.60	54.73	5.78
SR-STE	2:4	10B	54.02	39.02	76.88	68.35	75.58	41.46	29.60	54.99	**5.74**
AST	2:4	7.86B	54.51	39.62	**76.99**	67.32	73.48	39.93	28.60	54.35	5.91
CAST	2:4	7.86B	53.92	39.52	76.33	66.30	74.07	41.64	28.80	54.37	6.14
SparseForge (ours)	2:4	1.25B	52.95	41.15	75.46	69.14	76.35	43.86	32.80	55.96	6.24
SparseForge† (ours)	2:4	5B	**54.42**	**41.63**	76.88	**69.69**	**77.95**	**45.31**	**35.00**	**57.27**	6.09

5.3 Comparison with Prior Methods on LLaMA-2-7B

We compare SparseForge with representative post-training and retraining-based semi-structured sparsification methods, including SparseGPT [9], Wanda [28], MaskLLM [8], AST [13], and CAST [14], under aligned 2:4 sparsity.

As shown in Table 2 and Figure 5, SparseForge achieves 57.27% average accuracy using only 5B retraining tokens, outperforming the 7.86B-token CAST result (54.37%). This indicates that improved mask optimization and structure-aware tempering can substantially reduce the retraining required for high-quality semi-structured sparse recovery. We use 3 different seeds for SparseForge 5B-token experiment, which is reported in Appendix J(Table 14).

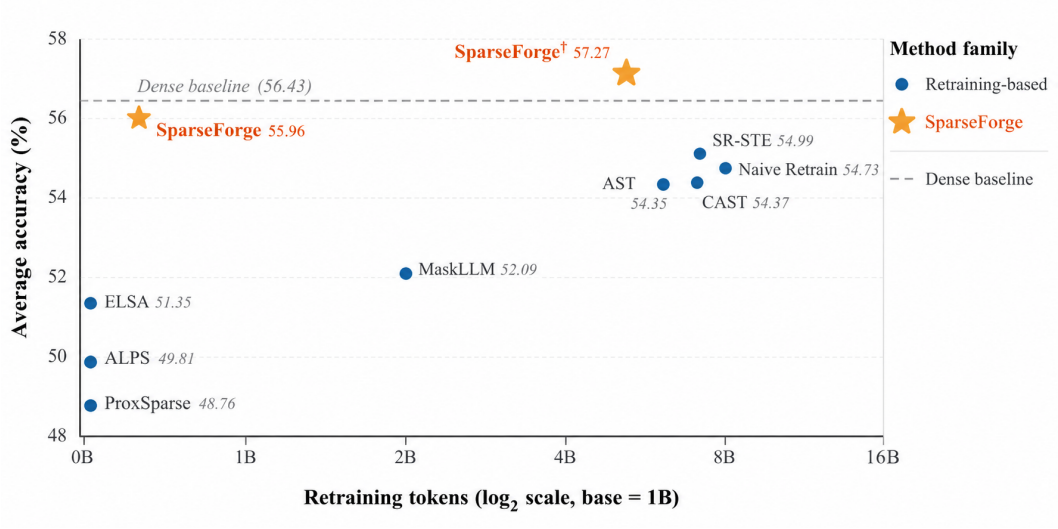

Figure 5: Accuracy vs. retraining tokens on LLaMA-2-7B (2:4 sparsity, log-scale). SparseForge matches the 40B-token CAST† with ∼8× fewer tokens. AST/AST+SLoRB (semi-transparent) report only a 5-task mean.

5.4 Ablation Studies

We conduct a full ablation of SparseForge on LLaMA-2-7B under 2:4 sparsity, using Dolmino-mix-1124 [23, 1] as the default retraining corpus with 1.25B retraining tokens (20k steps). Table 4 studies the effects of retraining data, training budget, importance criterion, SLoRB, teacher scale, and distillation. The full ablation results are provided in Appendix B (Table 4).

Several trends are clear. First, SparseForge remains relatively token-efficient: halving the budget from 20k to 10k steps reduces mean accuracy by only 1.38%. Second, Hessian-aware importance is important for resolving intra-group competition: replacing it with magnitude-based scoring lowers mean accuracy by 1.66% and worsens perplexity. Finally, the remaining components also make nontrivial contributions: removing SLoRB causes the largest drop among these auxiliary ablations (-2.23%), while removing distillation (-1.55%) or replacing the teacher with a larger 13B model (-2.40%) also degrades recovery. Overall, these results support the view that SparseForge benefits from both better mask optimization and careful training design, rather than from retraining scale alone.

6 Conclusion

We introduced **SparseForge**, a post-training framework for semi-structured sparsification that follows a mask-annealing pipeline: SparseForge explicitly learns and progressively quenches the sparsity mask toward a deployable 2:4 pattern, driven by a dual-track retraining loop that jointly updates weights and the soft mask under a Hessian-guided structured target. Our results suggest that under grouped semi-structured constraints, sparse recovery is governed more by mask quality than by retraining scale alone. By combining explicit soft-mask optimization, Hessian-guided structured targets, and progressive quenching within this dual-track loop, SparseForge achieves strong sparse recovery across model families and a favorable accuracy–efficiency trade-off under semi-structured sparsity, matching or surpassing prior state-of-the-art recovery while using substantially fewer retraining tokens. More broadly, these findings highlight mask-centric optimization as a promising direction for hardware-aligned sparse LLM compression.

AI Use Statement

Generative AI tools were used to assist with manuscript drafting and editing, translation. The authors take full responsibility for the final manuscript, including all reported results, claims, figures, and other AI-assisted content.

Reproducibility Statement

The main text describes the SparseForge algorithm and experimental setup. The appendix provides training configurations, implementation details, and baseline reproduction settings. We distinguish checkpoint re-evaluations from paper-based reimplementations when reporting baseline results. Our anonymous code repository is available at https://anonymous.4open.science/r/SparseForge-Anonymous.

Ethics Statement

This work studies LLM sparsification to improve deployment efficiency. Reducing computational cost should not be interpreted as resolving potential biases, privacy risks, or harmful behaviors of the underlying models. Deployment should respect the licenses and usage conditions of the models and datasets and include application-specific assessments of quality, fairness, and safety.

References

[1] Allen Institute for AI. allenai/dolmino-mix-1124. https://huggingface.co/datasets/allenai/dolmino-mix-1124, 2024. Hugging Face dataset card, accessed: 2026-04-30.

[2] Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sid Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin Wang, Genta Indra Winata, Franccois Yvon, and Andy Zou. Lessons from the trenches on reproducible evaluation of language models. *ArXiv*, abs/2405.14782, 2024. URL https://api.semanticscholar.org/CorpusID:269982020.

[3] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*, 2019. URL https://api.semanticscholar.org/CorpusID:208290939.

[4] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *ArXiv*, abs/1905.10044, 2019. URL https://api.semanticscholar.org/CorpusID:165163607.

[5] 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*, abs/1803.05457, 2018. URL https://api.semanticscholar.org/CorpusID:3922816.

[6] Damai Dai, Chengqi Deng, Chenggang Zhao, Runxin Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://api.semanticscholar.org/CorpusID:266933338.

[7] Jesse Dodge, Ana Marasovic, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, Matt Gardner, and William Agnew. Documenting large webtext corpora: A case study on the colossal

clean crawled corpus. In *Conference on Empirical Methods in Natural Language Processing*, 2021. URL https://api.semanticscholar.org/CorpusID:237568724.

[8] Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. Maskllm: Learnable semi-structured sparsity for large language models. *ArXiv*, abs/2409.17481, 2024. URL https://api.semanticscholar.org/CorpusID:272910976.

[9] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. *ArXiv*, abs/2301.00774, 2023. URL https://api.semanticscholar.org/CorpusID:255372747.

[10] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *ArXiv*, abs/1902.09574, 2019. URL https://api.semanticscholar.org/CorpusID:67855585.

[11] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural network. In *Neural Information Processing Systems*, 2015. URL https://api.semanticscholar.org/CorpusID:2238772.

[12] Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems*, 1993.

[13] Weiyu Huang, Guohao Jian, Yuezhou Hu, Jun Zhu, and Jianfei Chen. Pruning large language models with semi-structural adaptive sparse training. In *AAAI Conference on Artificial Intelligence*, 2024. URL https://api.semanticscholar.org/CorpusID:271544038.

[14] Weiyu Huang, Yuezhou Hu, Jun Zhu, and Jianfei Chen. Cast: Continuous and differentiable semi-structured sparsity-aware training for large language models. *ArXiv*, abs/2509.25996, 2025. URL https://api.semanticscholar.org/CorpusID:281682355.

[15] M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2):433–450, 1990. doi: 10.1080/03610919008812866. URL https://doi.org/10.1080/03610919008812866.

[16] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard H. Hovy. Race: Large-scale reading comprehension dataset from examinations. *ArXiv*, abs/1704.04683, 2017. URL https://api.semanticscholar.org/CorpusID:6826032.

[17] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, 1990.

[18] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ArXiv*, abs/1608.08710, 2016. URL https://api.semanticscholar.org/CorpusID:14089312.

[19] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2755–2763, 2017. URL https://api.semanticscholar.org/CorpusID:5993328.

[20] 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 *Conference on Empirical Methods in Natural Language Processing*, 2018. URL https://api.semanticscholar.org/CorpusID:52183757.

[21] Asit K. Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *ArXiv*, abs/2104.08378, 2021. URL https://api.semanticscholar.org/CorpusID:233296249.

[22] NVIDIA. Nvidia ampere architecture in-depth. https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/, 2020. Accessed: 2026-04-30.

[23] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James Validad Miranda, Jacob Daniel Morrison, Tyler C. Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke S. Zettlemoyer, Ali Farhadi, Noah Smith, and Hanna Hajishirzi. 2 olmo 2 furious. *ArXiv*, abs/2501.00656, 2024. URL https://api.semanticscholar.org/CorpusID:275213098.

[24] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[25] 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. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2019. URL https://api.semanticscholar.org/CorpusID:204838007.

[26] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande. *Communications of the ACM*, 64:99 – 106, 2019. URL https://api.semanticscholar.org/CorpusID:198893658.

[27] Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximations for model compression. *ArXiv*, abs/2004.14340, 2020. URL https://api.semanticscholar.org/CorpusID:216641895.

[28] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. *ArXiv*, abs/2306.11695, 2023. URL https://api.semanticscholar.org/CorpusID:259203115.

[29] Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Niko lay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony S. Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel M. Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, R. Subramanian, Xia Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zhengxu Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melissa Hall Melanie Kambadur, Sharan Narang, Aur'elien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288, 2023. URL https://api.semanticscholar.org/CorpusID:259950998.

[30] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *ArXiv*, abs/1905.00537, 2019. URL https://api.semanticscholar.org/CorpusID:143424870.

[31] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Helen Li. Learning structured sparsity in deep neural networks. In *Neural Information Processing Systems*, 2016. URL https://api.semanticscholar.org/CorpusID:2056019.

[32] Peng Xu, Wenqi Shao, Mengzhao Chen, Shitao Tang, Kai-Chuang Zhang, Peng Gao, Fengwei An, Yu Qiao, and Ping Luo. Besa: Pruning large language models with block-wise parameter-efficient sparsity allocation. *ArXiv*, abs/2402.16880, 2024. URL https://api.semanticscholar.org/CorpusID:268032346.

[33] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Jingren Zhou, Junyan Lin, Kai Dang, Keqin Bao, Ke-Pei Yang, Le Yu, Li-Chun Deng, Mei Li, Min Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shi-Qiang Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. 2025. URL https://api.semanticscholar.org/CorpusID:278602855.

[34] Lu Yin, You Wu, Zhenyu (Allen) Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Mykola Pechenizkiy, Yi Liang, Zhangyang Wang, and Shiwei Liu. Outlier weighed layerwise sparsity (owl): A missing secret sauce for pruning llms to high sparsity. *ArXiv*, abs/2310.05175, 2023. URL https://api.semanticscholar.org/CorpusID:263829692.

[35] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Annual Meeting of the Association for Computational Linguistics*, 2019. URL https://api.semanticscholar.org/CorpusID:159041722.

[36] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models. *ArXiv*, abs/2205.01068, 2022. URL https://api.semanticscholar.org/CorpusID:248496292.

A Detailed Cross-Model Results Under 2:4 Sparsity

For readability, the main text reports a compact cross-model summary using mean zero-shot accuracy only. In this appendix, we provide the full task-level results corresponding to the cross-model comparison in Table 1. Table 3 reports the dense-model performance, SparseForge performance, and their differences on each benchmark under 2:4 semi-structured sparsity, together with the mean accuracy and WikiText-2 perplexity. The run-specific configuration in Appendix F documents a separate LLaMA-2-7B run and is not assigned retrospectively to all of the cross-model results reported here.

Table 3: Detailed cross-model results under 2:4 semi-structured sparsity. We report zero-shot accuracy (%) on downstream tasks, the mean accuracy across tasks, and WikiText-2 perplexity (PPL) for the dense model, SparseForge, and their differences.

Model	Variant	ARC-c	ARC-e	BoolQ	Hella.	OBQA	RTE	Wino.	Mean	Wiki PPL
GPT-2-Medium	Dense	21.67	49.07	58.41	33.23	18.60	52.71	53.12	40.97	21.80
	SparseForge	21.16	46.04	60.09	31.70	16.40	53.43	53.35	40.31	25.90
	Diff	-0.51	-3.03	+1.68	-1.53	-2.20	+0.72	+0.24	-0.66	+4.10
GPT-2-Large	Dense	21.84	53.16	60.55	36.39	19.40	52.71	55.25	42.76	18.95
	SparseForge	22.10	50.29	61.90	34.68	19.40	52.71	53.59	42.10	23.57
	Diff	+0.26	-2.86	+1.35	-1.70	+0.00	+0.00	-1.66	-0.66	+4.62
GPT-2-XL	Dense	25.09	58.29	61.68	40.00	22.60	52.35	58.41	45.49	17.12
	SparseForge	23.29	55.98	61.77	38.37	21.40	54.15	55.41	44.34	20.53
	Diff	-1.79	-2.31	+0.09	-1.63	-1.20	+1.81	-3.00	-1.15	+3.41
OPT-2.7B	Dense	26.79	60.77	60.40	45.86	25.00	54.51	61.01	47.76	12.45
	SparseForge	25.17	58.67	63.06	44.44	21.80	52.71	60.85	46.67	15.14
	Diff	-1.62	-2.10	+2.66	-1.41	-3.20	-1.81	-0.16	-1.09	+2.69
Qwen3-1.7B	Dense	39.85	72.26	77.58	46.09	28.20	70.76	60.85	56.51	15.56
	SparseForge	37.97	70.08	74.53	46.30	27.00	56.68	60.77	53.33	10.54
	Diff	-1.88	-2.19	-3.06	+0.21	-1.20	-14.08	-0.08	-3.18	-5.02
Qwen3-8B	Dense	55.80	83.50	86.61	57.14	31.00	78.34	67.72	65.73	8.97
	SparseForge	50.85	81.52	82.35	54.34	31.60	72.92	69.30	63.31	8.03
	Diff	-4.95	-1.98	-4.25	-2.80	+0.60	-5.42	+1.58	-2.42	-0.94
Qwen3-14B	Dense	58.45	84.18	89.33	60.96	35.00	77.62	73.01	68.36	7.89
	SparseForge	53.67	82.24	84.92	57.94	34.40	72.20	72.69	65.44	7.17
	Diff	-4.78	-1.94	-4.40	-3.03	-0.60	-5.42	-0.32	-2.93	-0.72
DeepSeek-MoE	Dense	45.05	75.88	72.72	58.09	32.00	62.82	70.24	59.54	6.28
	SparseForge	41.47	74.12	73.82	57.45	32.00	62.09	69.06	58.57	6.92
	Diff	-3.58	-1.77	+1.10	-0.64	+0.00	-0.72	-1.18	-0.97	+0.64

B Full Ablation on LLaMA-2-7B

Table 4 reports the full ablation results on LLaMA-2-7B, varying the training budget, importance criterion, SLoRB, teacher size, and distillation.

Table 4: Full ablation on LLaMA-2-7B under 2:4 sparsity. Unless otherwise noted, variants are trained on dolmino with 1.25B retraining tokens (20k steps). We report WikiText-2 perplexity, mean zero-shot accuracy, and the accuracy difference relative to the full SparseForge configuration in percentage points (pp).

Variant	**Steps / Tokens**	**Wiki PPL**	**Mean Acc.**	**Diff vs Full (pp)**
Dense	2T	5.1983	59.71%	+2.44
SparseForge (full)	20k / 1.25B	6.2423	57.27%	–
SparseForge w/ 10k steps	10k / 0.625B	6.3355	55.89%	−1.38
SparseForge w/ magnitude	20k / 1.25B	6.3875	55.61%	−1.66
SparseForge w/o SLoRB	20k / 1.25B	6.2217	55.04%	−2.23
SparseForge w/ larger teacher (13B)	20k / 1.25B	6.4689	54.87%	−2.40
SparseForge w/o distillation	20k / 1.25B	6.5194	55.72%	−1.55

C Extension to Block-16 Sparsity

Although our main experiments focus on the practically important 2:4 semi-structured pattern, the SparseForge formulation is not inherently restricted to this setting. In particular, the score-based projection and progressive quenching mechanism can be adapted to other structured sparsity patterns by redefining the feasible group and projection operator. To examine this generality, we additionally evaluate SparseForge under a block-16 sparsity pattern. Concretely, in our block-16 setting, each 16×16 weight block is constrained to contain only 50% nonzero entries. Compared with 2:4 sparsity, this is a relatively looser form of structured sparsity: the sparsity pattern is still constrained at the block level, but the competition among weights is imposed over a larger local region rather than by a strict fine-grained $n:m$ rule within every group of four.

Table 5 reports results on several representative model families under block-16 sparsity. Overall, SparseForge remains effective beyond 2:4, with moderate degradation on most model families and benchmarks. In particular, LLaMA-2-7B retains strong performance under block-16, with mean accuracy dropping from 59.70% to 59.02% (a 0.68-point decrease). OPT-2.7B remains nearly unchanged, showing a slight mean improvement from 47.76% to 47.93%. Similar trends are also observed on GPT-2-XL, where the mean accuracy only decreases from 45.49% to 45.14%.

At the same time, the block-16 setting appears to be more challenging for some architectures, especially smaller or more sensitive models. For example, Qwen3-1.7B shows a larger drop in mean accuracy (from 56.47% to 53.49%), and DeepSeek-MoE-16B also exhibits a noticeable decrease (from 59.54% to 58.20%). These results suggest that while the proposed mask optimization strategy transfers beyond 2:4, the degree of recoverability under block-structured sparsity still depends on model architecture and training dynamics.

Overall, these results support the view that SparseForge is a general mask optimization framework rather than a method tied specifically to 2:4 sparsity. We leave a more systematic study of different structured sparsity patterns and their hardware–accuracy trade-offs to future work.

D End-to-End Inference Speedup

Beyond accuracy recovery, a practical motivation for adopting the 2:4 semi-structured pattern is that it is directly supported by modern NVIDIA sparse Tensor Cores [22], enabling tangible end-to-end acceleration on real hardware. To verify that our SparseForge-trained 2:4 sparse models translate into actual deployment gains, we evaluate their inference throughput using TensorRT-LLM on an NVIDIA H800 GPU, under four different input/output sequence length settings. We use the 2:4 sparse LLaMA-2-7B model produced by SparseForge, and compare its decoding throughput (tokens/s) and memory footprint against the corresponding dense model under identical inference configurations.

Table 5: SparseForge under block-16 sparsity across model families. We report mean zero-shot accuracy before and after sparsification.

Model	Base Mean	Block-16 Mean	Diff
LLaMA-2-7B	59.70	59.02	-0.68
OPT-2.7B	47.76	47.93	+0.17
GPT-2	37.13	37.58	+0.45
GPT-2-Medium	41.01	39.45	-1.56
GPT-2-Large	42.65	41.87	-0.81
GPT-2-XL	45.49	45.14	-0.35
Qwen3-1.7B	56.47	53.49	-2.98
DeepSeek-MoE-16B	59.54	58.20	-1.34

Table 6: Detailed results of SparseForge under block-16 sparsity on LLaMA-2-7B.

Variant	ARC-c	ARC-e	BoolQ	Hella.	OBQA	RTE	Wino.	Mean
Dense	43.43	76.26	77.71	57.17	31.40	62.82	69.14	59.70
SparseForge	41.89	75.25	76.24	56.78	32.20	61.73	69.06	59.02
Diff	-1.54	-1.01	-1.47	-0.39	+0.80	-1.08	-0.08	-0.68

Table 7: End-to-end inference throughput (tokens/s) of SparseForge 2:4 sparse LLaMA-2-7B vs. dense baseline, measured with TensorRT-LLM on an NVIDIA H800 GPU under different input/output sequence lengths. The last row reports the model weight memory footprint. "Efficiency" denotes the sparse-over-dense ratio (speedup for throughput, compression ratio for memory).

(Inp Len, Out Len)	Sparse	Dense	Efficiency
(128, 128)	108.72	75.41	1.44×
(128, 1024)	106.18	74.63	1.42×
(1024, 128)	103.87	73.62	1.41×
(1024, 1024)	101.29	72.38	1.40×
Memory Consumption	7.31 GB	12.55 GB	0.58×

Table 7 reports the results. Across all input/output length configurations, the 2:4 sparse model consistently outperforms its dense counterpart, delivering $1.40\times$–$1.44\times$ throughput improvements on H800. In addition, the memory footprint of model weights is reduced to roughly 58% of the dense baseline (7.31 GB vs. 12.55 GB for LLaMA-2-7B), which directly lowers the GPU memory requirement for deployment. These measurements describe the previously reported TensorRT-LLM deployment setting. The training configuration in Appendix F does not identify the exported checkpoint used for this benchmark; these results should therefore not be interpreted as measurements of that SLoRB-enabled training run.

E Discussion and Limitations

Scaling retraining tokens is not the only path. The experiments investigate mask optimization together with recovery training, rather than treating the number of retraining tokens as the only design choice. The ablations in Appendix B (Table 4) examine the importance criterion, SLoRB, teacher size, distillation, and the training budget. The cross-model results in Table 3 further examine how recovery varies across model families. Comparisons in Table 2 should distinguish paper-reported numbers, checkpoint re-evaluations, and paper-based reimplementations, and use explicitly identified task suites and metrics. A smaller token budget is not, by itself, an equal reduction in training FLOPs or wall-clock time.

Limitations. Recovery remains sensitive to the retraining corpus and model family. The larger degradation on Qwen3 in Table 1 may reflect a mismatch between its pretraining mixture and the generic C4 recovery corpus; this explanation remains a hypothesis rather than a controlled attribution. We focus primarily on 2:4 sparsity, with only a preliminary block-16 study in Appendix C, and evaluate a fixed set of model families. The configurations in Appendices F–I document separate runs with different models, corpora, and optimization settings; they do not constitute a matched experimental comparison. In particular, the documented 7.864B-token SLoRB-enabled run does not establish the configuration of earlier 1.25B- or 5B-token checkpoints. A configured 2:4 mask alone does not establish exact 2:4 sparsity of the full effective weights after export.

F SparseForge Training Configuration

This section describes the LLaMA-2-7B SLoRB-tp configuration on Dolmino-Mix-1124. Both student and teacher are LLaMA-2-7B. The configuration enables SLoRB with a trainable projection and is separate from SLoRB-disabled runs and earlier 5B-token results.

F.1 Model, Data, and Training Budget

Training uses eight NVIDIA L20D GPUs on one node and an 8.000B-token Dolmino-Mix-1124 corpus tokenized with the LLaMA-2 vocabulary and stored as `uint32`. The global batch contains 64 sequences of 4,096 tokens. The configured budget consists of 27,000 main-phase steps followed by 3,000 hard-mask finetuning steps:

$$\begin{aligned} B_{\text{step}} &= 64 \times 4{,}096 = 262{,}144, \\ B_{\text{total}} &= (27{,}000 + 3{,}000) \times 262{,}144 \\ &= 7{,}864{,}320{,}000 \text{ tokens}. \end{aligned} \tag{6}$$

The final finetuning phase accounts for 10% of this configured budget. These are configuration-derived counts, not a statement that training has completed.

F.2 Training Hyperparameters

Table 8 lists the training settings. The structural-interpolation schedule is disabled (`beta_structural_start=0`, `beta_structural_end=0`); the configured final mask target remains `nm_2_4`.

Table 8: SparseForge training settings for the LLaMA-2-7B SLoRB-tp configuration.

Parameter	Value
Model and data	
`student_model`	`LLaMA-2-7B`
`teacher_model`	`LLaMA-2-7B`
`dataset`	`Dolmino-Mix-1124 (LLaMA-2 tokenizer)`
`block_size`	`4096`
`dtype`	`auto`
Optimization and schedule	
`learning_rate`	`0.0001`
`min_lr`	`1e-05`
`warmup_iters`	`1350`
`lr_decay_iters`	`27000`
`max_iters`	`27000`
`final_finetune_iters`	`3000`
`batch_size`	`1`
`global_batch_size`	`64`
`weight_decay`	`0.1`
`srste_decay`	`0.0`
`increase_step`	`10000`
Mask and sparsity	
`mode`	`sparse_forward`
`mask_type`	`unstructured`
`hard_mask_type`	`nm_2_4`
`mask_metric`	`hessian_ratio`
`sparsity_ratio`	`0.5`
`change_mask`	`True`
`beta`	`0.98`
`structured_exact`	`False`

Continued on next page

Table 8 (continued)

Parameter	Value
structured_n	2
structured_m	4
glu_joint_mask	False
weight_scaling	False
Hutchinson and mask updates	
enable_hutchinson	True
mask_update_period	10
mask_update_period_before	30
mask_update_period_after	10
mask_update_switch_step	2700
mask_lr	0.1
score_ema_beta	0.99
Temperature annealing	
temp_init	1.0
temp_decay	0.97
temp_min	0.05
Sparsity penalty	
sparsity_warmup_steps	675
tau_sample_size	262144
mask_penalty_lr	0.1
mask_penalty_lr_min	None
mask_penalty_lr_schedule	constant
mask_penalty_mode	nm_2_4
sparsity_alpha	0.2
lambda_mid_max	0.1
adaptive_l1_decay	0.0
Hardening	
mask_hardening_start	2700
mask_hardening_duration	20250
hardening_period	0
hardening_fraction	0.0
hardening_band_low	0.4
hardening_band_high	0.6
freeze_low	0.0
freeze_high	1.0
beta_structural_start	0
beta_structural_end	0
SLoRB	
SLoRB	True
SLoRB_init_type	sum
SLoRB_k	64
trainable_projection	True
Distillation	
distill_model	True
hardness_task	1.0
hardness_kldiv	1.0
hardness_squarehead	0.0
Distributed training and precision	
use_fsdp	True
fsdp_mode	hybrid_sharded
fsdp_cpu_offload	False
fsdp_mixed_precision	True
gradient_checkpointing	False
use_deepspeed	False
eager_attention	True

Continued on next page

Table 8 (continued)

Parameter	Value
Additional training settings	
distill_temperature	2.0
freeze_non_slorb	False
hybrid_alpha	1.0
mask_transition_steps	0
mask_warmup_steps	0
seed	1337
temperature	1.0

At the transition to final hard-mask finetuning, the optimizer is rebuilt at learning rate 10^{-4} with reset moment state, and the gradient scaler is rebuilt. The supplied settings do not specify the optimizer class, its moment coefficients or epsilon, the clipping threshold, the exact learning-rate decay function, or the number of Hutchinson probes; no values are inferred for these fields.

G AST Training Configurations

The three AST configurations use LLaMA-2-7B, Qwen3-8B-Base, and Qwen3.5-9B-Base-text with Dolmino-Mix-1124 tokenized for the corresponding model. These training configurations are distinct from the earlier re-evaluation of an official AST checkpoint.

G.1 Shared Training Hyperparameters

All three runs use `uint32` token storage, `static_seeded_sequence_permutation` sampling with seed 17, and FP32 master weights, BF16 computation, and FP32 reduction (`fp32_master_bf16_compute_fp32_reduce`). Their shared training parameters are listed in Table 9.

Table 9: Shared AST training hyperparameters.

Parameter	Value
seq_len	4096
lr	2e-05
lr_floor	0.1
decay_max	6e-05
decay_ramp_frac	0.25
mask_every	10
slorb_k	0
optimizer_mode	paper_reindexed
ce_weight	0.6666666666666666
kl_weight	0.3333333333333333
kl_temp	1.0
clip	1.0
beta1	0.9
beta2	0.95
eps	1e-08
seed	17

G.2 Model-Specific Training Settings

Table 10: Model-specific AST settings. Corpus sizes and training-token budgets are distinct quantities.

Parameter	LLaMA-2-7B	Qwen3-8B	Qwen3.5-9B
global_batch	128	64	64
micro_batch	4	2	4
steps	15000	30000	30000
warmup	500	1000	1000
no_activation_checkpointing	True	True	False
Corpus tokens	8,000,013,916	6,986,950,988	7,056,026,407
Sequence count	1,953,128	1,705,798	1,722,662
Configured training tokens	7.86432B	7.86432B	7.86432B

The configured training-token totals follow from $15{,}000 \times 128 \times 4{,}096$ for LLaMA-2-7B and $30{,}000 \times 64 \times 4{,}096$ for each Qwen run.

H CAST Training Configurations

The C4 and Dolmino-Mix-1124 configurations below describe two separate paper-based CAST reimplementations. Their corpora, micro-batches, mask-refresh intervals, and optimizer settings differ and are not combined into one recipe.

H.1 C4 Configuration

The C4 configuration uses a global batch of 256 sequences, sequence length 4,096, and a token budget of 7,864,320,000, corresponding to 7,500 full global batches. The literal `max_steps=0` setting is retained below. The CAST algorithm is enabled; the separate mask and SLoRB subsystems are disabled. Their configured defaults are listed under explicitly marked headings and do not indicate that these subsystems were active during training.

Table 11: Training configuration of the C4 CAST reimplementation. Disabled subsystems are explicitly marked.

Parameter	Value
Objective and run controls	
arm	cast
kl_temperature	1.0
lambda_kl	0.3333333333333333
lambda_task	1.0
loss_chunk_tokens	0
max_steps	0
seed	1234
Training budget	
global_batch_sequences	256
micro_batch_size	2
seq_len	4096
total_tokens	7864320000
CAST algorithm (enabled)	
decay_max	4e-07
decay_mode	grad_l1
decay_warmup_frac	0.5
dense_forward	True
enabled	True

Continued on next page

Table 11 (continued)

Parameter	Value
mask_refresh_interval	100
scaling_granularity	out_channel
scaling_lr	0.0001
terminal_prune_frac	0.02
weight_scaling	True
Training data	
corpus	C4 (LLaMA-2 tokenizer)
dtype	uint32
seed	1234
seq_len	4096
shuffle	bijection
synthetic	False
synthetic_tokens	16777216
vocab_size	0
Target model structure	
num_hidden_layers	32
roles_per_layer	q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj
target_element_count	6476005376
target_tensor_count	224
Mask subsystem (disabled)	
beta_smoothstep_frac	0.8
ema_alpha	0.1
enabled	False
eta_pen	0.1
hardening_frac	0.1
hessian_beta	0.99
hessian_eps	1e-08
hessian_source	grad_sq_ema
init_from	score
lambda_mid_max	0.1
mask_dtype	float32
metric	hessian
recovery_steps	1000
temperature0	1.0
temperature_gamma	0.99
theta	0.5
update_interval	100
Model and precision	
activation_checkpointing	True
attn_implementation	sdpa
group_axis	1
group_keep	2
group_size	4
master_dtype	float32
param_dtype	bfloat16
reduce_dtype	bfloat16
student_model	LLaMA-2-7B
target_roles	[]
teacher_dtype	bfloat16
teacher_mode	replicate
teacher_model	LLaMA-2-7B
Optimizer and schedule	
betas	[0.9, 0.95]
eps	1e-08
grad_clip	1.0

Continued on next page

Table 11 (continued)

Parameter	Value
lr	2e-05
min_lr_ratio	0.05
schedule	cosine
warmup_frac	0.01
weight_decay	0.0
SLoRB (disabled)	
enabled	False
init	mean
k	16
lr	0.0001
materialize_projection	False
trainable_projection	False

H.2 DOLMINO-MIX-1124 CONFIGURATION

This separate LLaMA-2-7B configuration uses Dolmino-Mix-1124 and a cosine learning-rate schedule. Its 7,500 steps, global batch of 256 sequences, and sequence length of 4,096 give the same nominal 7,864,320,000-token budget as above, but do not make the two configurations equivalent.

Table 12: Training configuration of the separate Dolmino-Mix-1124 CAST reimplementation.

Parameter	Value
Model	LLaMA-2-7B
Training corpus	Dolmino-Mix-1124
Token storage	uint32
Stored corpus tokens	77,721,665,859
Training steps	7500
Global batch sequences	256
Sequence length	4096
Micro-batch sequences	1
Gradient accumulation	32
World size	8
Hardware	8 NVIDIA L20A GPUs
Tokens per step	1,048,576
Total tokens	7,864,320,000
Learning-rate schedule	cosine
Peak learning rate	2e-5
Final learning rate	2e-6
Warmup steps	375
L1 decay coefficient	4e-7
Mask refresh period T1	10
Scale groups	2
Scaling parameter eta	1/3
KL temperature	1.0
Gradient clipping	1.0
Optimizer betas	[0.9, 0.999]
Optimizer epsilon	1e-8
Gradient checkpointing	enabled
Training seed	1234
Master-weight precision	float32
Parallel implementation	DDP with optimizer-state sharding (ZeRO-1)
Target linear tensors	224
Target weight elements	6,476,005,376
Masked elements	3,238,002,688

I Additional Baseline Configurations

The ALPS, ProxSparse, and ELSA configurations in this section use Sheared-LLaMA-1.3B, not the LLaMA-2-7B checkpoints reported elsewhere. Table 13 groups the supplied pruning and optimization settings by method.

ALPS uses C4 English validation for calibration, with effective $\rho = 0.1$. ELSA's calibration corpus and optimizer moment coefficients are not specified. For ProxSparse, only mask-application scope and output precision are supplied; its original optimizer, learning rate, optimization-step count, and calibration-sample count are unspecified. No missing settings are inferred.

Table 13: Pruning and optimization settings on Sheared-LLaMA-1.3B.

Parameter	Value
ALPS	
base_model	Sheared-LLaMA-1.3B
nm	2:4
sp	0.5
nsamples	128
seed	0
calibration_corpus	C4 English validation, shard 0 of 8
rho_effective	0.1
seqlen	2048
in_scope_linears	168
saved_dtype	float16
ProxSparse	
base_model	Sheared-LLaMA-1.3B
params_masked	168
saved_dtype	float16
ELSA	
base_model	Sheared-LLaMA-1.3B
sparsity_type	2:4
sparsity_ratio	0.5
seed	0
admm_steps	4096
admm_batch_size	2
admm_gradient_accumulation_steps	4
admm_lr	0.0002
admm_lmda	0.01
admm_interval	32
seqlen	2048
saved_dtype	float16
in_scope_linears	168

J Random-Seed Variability of the 5B-Token Experiment

We report the final exported-model results of SparseForge on LLaMA-2-7B with a retraining budget of 5B tokens for three random seeds: 1234, 1235, and 1236. Table 14 gives the CAST-7 mean zero-shot accuracy and WikiText-2 perplexity for each seed, together with the arithmetic mean and sample standard deviation across the three runs. CAST-7 accuracy is the macro-average of plain accuracy over HellaSwag, RACE, PIQA, WinoGrande, ARC-Easy, ARC-Challenge, and OpenBookQA. These results correspond to the 5B-token setting in Table 2 and are separate from the 1.25B-token ablation results in Appendix B.

Table 14: Three-seed results of SparseForge on LLaMA-2-7B with 5B retraining tokens per run. Accuracy is reported in percent; lower perplexity is better. The summary row reports the mean and sample standard deviation ($n = 3$, denominator $n - 1$), rounded to two decimal places.

Seed	**CAST-7 Acc. (%)**	**Wiki PPL**
1234	57.18	6.05
1235	57.27	6.09
1236	57.36	6.12
Mean $\pm$ SD	57.27 ± 0.09	6.09 ± 0.04

Across these three seeds, mean zero-shot accuracy ranges from 57.18% to 57.36%, and WikiText-2 perplexity ranges from 6.05 to 6.12. The reported standard deviations characterize variability across training seeds rather than uncertainty due to evaluation-set sampling.