Batch-wise Adaptive Pruning: Periodic Neuron Activation-Aware Weight Pruning for Language Reasoning Model

Anonymous ACL submission

Abstract

Large Reasoning Models (LRMs) achieve strong performance on complex tasks through extended chain-of-thought generation, but incur substantial computational costs during inference. In production settings, batched inference is essential for high throughput, yet existing adaptive pruning methods face performance limitations: First, they rely on averaging activations across samples to determine shared pruning masks, which may miss critically activated neurons for some individual samples. Second, after the averaging, they adopt threshold-based selection for pruning neurons, causing sparsity ratio instability. In this work, we propose a training-free adaptive pruning method designed specifically for batched inference in LRMs. Our method adopts max-pooling activations for cross-sample aggregation to better capture the sample-specific important neurons. To stabilize the activation sparsity ratio, our method adopts periodic top-k selection over the aggregated neurons instead of threshold-based selection. Futhremore, based on the observation that important neurons tend to be repeatedly activated, we incorporate an activation memory mechanism to capture periodically important neurons. Experiments on diverse reasoning benchmarks demonstrate that our method achieves a 39.7% improvement over the previous state-of-the-art adaptive pruning method at batch size 4 with 50% sparsity, along with 1.32× speedup over dense inference at batch size 1 and 1.14× at batch size 4, which is comparable to existing pruning methods, demonstrating practical efficiency gains for deployment. [1]

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities across a wide range of tasks (Grattafiori et al., 2024; Yang et al., 2024; Team, 2023), yet they come with substantial computational costs. In particular, recent advances in reasoning capabilities have led to the emergence of Large Reasoning Models (LRMs) (DeepSeek-AI, 2025; OpenAI, 2024), which employ extended chain-of-thought processes to solve complex problems. While effective, these models generate substantially longer outputs, further exacerbating computational costs during inference. With billions of parameters, modern LLMs (Grattafiori et al., 2024; Yang et al., 2024) incur substantial computational and memory costs, making deployment particularly difficult in resource-constrained environments (Gholami et al., 2024).

These computational bottlenecks become particularly critical in production settings, where serving

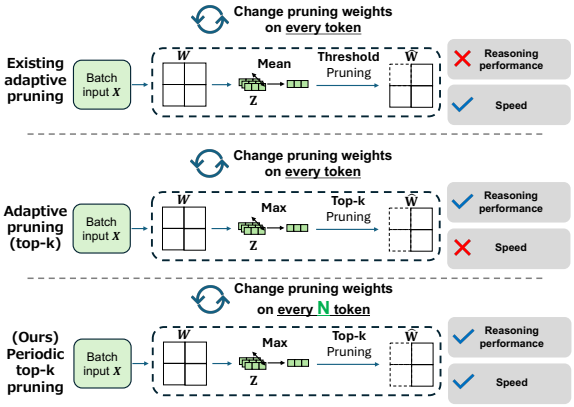

Figure 1: Comparison of existing adaptive pruning methods and our proposed approach for batched inference on reasoning tasks. (Top) Existing adaptive pruning methods (Liu et al., 2025) average activation $\mathbf{Z}$ across samples to enable batched inference, but this causes performance degradation on reasoning tasks. (Middle) Replacing averaging with max aggregation and top-k selection improves reasoning task performance, but computing top-k at every token incurs computational overhead, resulting in slow inference speed. (Bottom) Our method periodically updates the pruning mask using max aggregation with top-k selection, achieving strong performance on reasoning tasks in batched inference while maintaining computational efficiency.

[1] Our code is available at: https://anonymous.4open.science/r/reasoning_eval-F880

Table 1: Comparison of pruning methods across inference settings. Static pruning (Sun et al., 2024; Frantar and Alistarh, 2023) fails on reasoning tasks due to fixed patterns. Existing adaptive methods (Lee et al., 2024; Liu et al., 2025) degrade with batched inference. Our method achieves consistent performance across all settings (*). See Figure 1 for detailed comparison with the existing adaptive pruning and our method.

Task Type	Batch Size	Static Pruning	Existing Adaptive Pruning	Ours
Non-reasoning	1	✓	✓	✓
	≥2	✓	✓	✓
Reasoning	1	✗	✓	✓
	≥2	✗	✗	✓*

multiple requests simultaneously through batched inference is essential for achieving high throughput. For instance, modern inference frameworks such as vLLM (Kwon et al., 2023) employ continuous batching to efficiently handle concurrent requests, dynamically adding new requests to ongoing batches without waiting for previous generations to complete. To further minimize such request latency, we need a more lightweight models that can run on batched inference settings.

A promising approach to reduce inference cost is pruning. Existing pruning methods can be categorized as either *static* or *adaptive*. Static methods (Dong et al., 2024) determine pruning patterns once during the prompt phase; however, we observe that this fixed approach degrades performance on reasoning tasks where activation patterns evolve during chain-of-thought generation. Adaptive methods (Liu et al., 2025; Lee et al., 2024) address this by dynamically selecting neurons at each step based on activation magnitudes, achieving strong performance on complex reasoning tasks.

However, existing adaptive pruning methods such as CATS (Lee et al., 2024) and TEAL (Liu et al., 2025) face a fundamental limitation in batched inference scenarios. On GPUs, batched inference requires a single shared pruning mask across all samples in a batch. TEAL addresses this by averaging activations across samples to determine a common pruning pattern. While this averaging approach can identify neurons that are consistently activated across all samples, it misses important neurons that are occasionally but critically activated for individual samples. While this approach may work adequately for short question-answering tasks where information can be compressed, it causes significant performance degradation in generation tasks that require preserving sample-specific information under batched inference settings, particularly in reasoning tasks. Additionally, after the averaging procedure, these existing methods rely on threshold-based selection rather than top-k selection for pruning, which may cause performance instability because of the fluctuation of the actual sparsity ratio at inference.

In this work, we propose a training-free adaptive pruning method for LRMs that enable efficient batched inference while maintaining strong performance on reasoning tasks. The novelty of our method is as follows: (i) For cross-sample aggregation, our method adopts max-pooling activations rather than averaging, enabling more effective identification of neurons that are critically activated for individual samples. (ii) To stabilize the activation sparsity ratio, our method adopts top-k selection over the aggregated neurons instead of threshold-based selection. We introduce periodic mask updates instead of per-token updates, reducing the computational overhead of top-k selection while maintaining consistent sparsity. (iii) Based on the observation that important neurons tend to be repeatedly activated (Figure 3), we introduce an activation memory mechanism to better capture periodically important neurons.

Through extensive experiments on various reasoning tasks, we demonstrate that our method is able to maintain good performance in batched settings, while existing adaptive pruning methods suffer significant performance degradation. Specifically, our approach achieves a 39.7 percentage point improvement over the previous state-of-the-art adaptive pruning method at batch size 4 with 50% sparsity using DeepSeek-R1-Distill-Qwen-7B. Additionally, our approach achieves $1.32\times$ speedup over dense inference at a single batch inference and $1.14\times$ speedup at batch size 4, which is comparable to the existing pruning method, demonstrating practical efficiency gains for deployment.

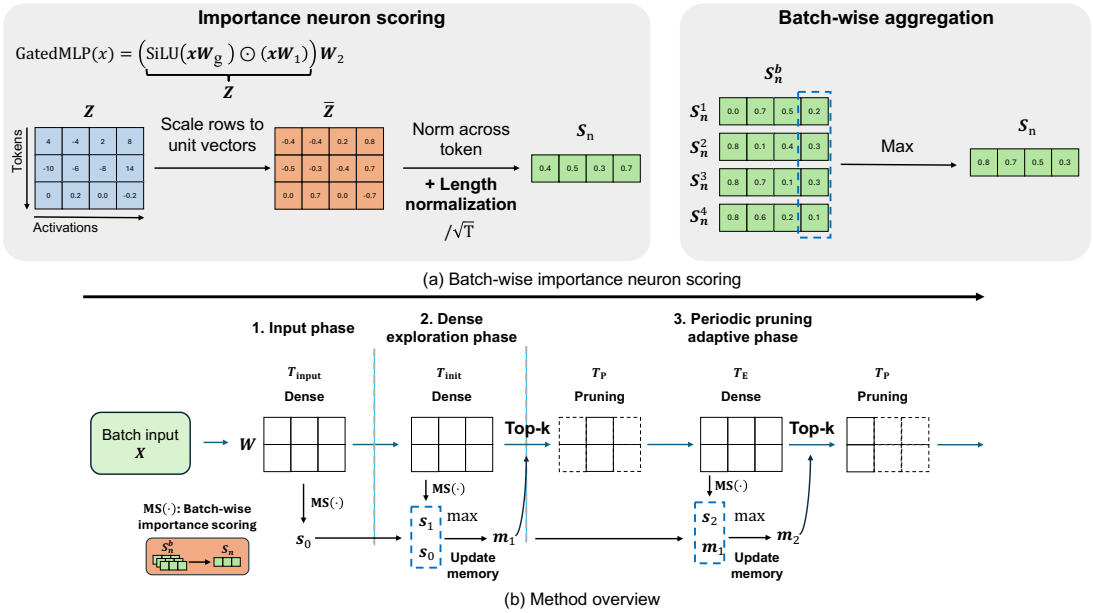

Figure 2: Overview of our batch-wise adaptive pruning approach. (a) We extend the prior importance scoring method (Dong et al., 2024) to support batched inference and periodic pruning. Importance scores are computed by the batch-wise scoring function $\mathrm{MS}(\cdot)$ through normalization along the activation dimension followed by norm aggregation along the token dimension. For batched inference, we aggregate importance scores $\mathbf{s}_n^b$ across the batch dimension using element-wise maximum, enabling the shared pruning mask to capture important neurons from each sample. (b) The method operates in three phases: input phase computes initial importance score $\mathbf{s}_0$, dense exploration phase updates the importance memory $\mathbf{m}$ via element-wise maximum, and periodic adaptive phase alternates between pruning and exploration. This design minimizes pruning switch overhead by maintaining stable pruning patterns, while periodic memory updates enable continuous retention of important neurons.

2 Related Work

Pruning methods for large language models can be categorized along several dimensions: granularity, input dependency, and adaptivity. Table 1 summarizes the key differences between existing pruning approaches and our proposed method in various settings. In contrast to existing approaches that struggle to maintain performance in batched settings or on reasoning tasks, our method is specifically designed to preserve accuracy while enabling efficient batched inference for LRMs.

Pruning Granularity. Pruning reduces model size by removing redundant parameters while preserving performance. Methods can be categorized by granularity: *unstructured pruning* removes individual weights, achieving high sparsity with minimal accuracy loss but requiring specialized hardware or software for acceleration. SparseGPT (Frantar and Alistarh, 2023) performs one-shot weight update during pruning, while Wanda (Sun et al., 2024) is training-free. For *structured pruning*, LLM-Pruner (Ma et al., 2023) and Sheared-LLaMA (Xia et al., 2024) require post-training to recover performance. *Semi-structured pruning* (e.g., N:M sparsity) offers a middle ground but requires specialized CUDA kernels and is limited to specific GPU architectures that support such sparsity patterns (Frantar and Alistarh, 2023; Sun et al., 2024). Our proposed method belongs to structured pruning and does not require specific GPU architectures, as it operates through standard matrix operations without specialized sparse computation kernels.

Input-Independent vs. Input-Dependent Pruning. Pruning methods can also be categorized based on whether the pruning pattern depends on input activations. *Input-independent* methods perform pruning before inference using calibration data, and this pattern remains fixed regardless of the input (Sun et al., 2024; Ashkboos et al., 2024; Kim et al., 2024; Frantar and Alistarh, 2023). While these approaches benefit from batch compatibility, they exhibit strong dependency on the calibration dataset, and cannot exploit the observation that different inputs activate different subsets of neurons. In contrast, *input-dependent* methods dynamically select which neurons to prune based on the acti-

vation distribution during inference (Dong et al., 2024; Liu et al., 2025; Lee et al., 2024). Although some input-dependent methods also utilize calibration data, their dependency on the calibration dataset is generally lower than input-independent approaches. However, input-dependent pruning introduces challenges for efficient batched inference, as different samples may require different pruning patterns. In contrast to prior input-dependent methods that suffer from performance degradation in batched settings, our approach can extend input-dependent pruning to batched inference while maintaining performance on reasoning tasks.

Static vs. Adaptive Pruning. The existing pruning methods can be further categorized based on whether the pruning mask is dynamically updated during decoding. Input-independent methods are inherently *static pruning*, as the pruning pattern is fixed during inference. Within input-dependent methods, *static pruning* determines a fixed sparsity pattern during the prompt phase and reuses it throughout generation (Dong et al., 2024), while *adaptive pruning* dynamically updates the pruning mask at each decoding step (Lee et al., 2024; Liu et al., 2025). Static pruning cannot adapt to evolving activation patterns during generation, leading to performance degradation on reasoning models that produce long outputs. However, existing adaptive pruning methods rely on threshold-based selection rather than top-k selection, because computing top-k at every token incurs prohibitive computational overhead. When aggregating pruning patterns across batches, this threshold-based approach causes the activation distribution to differ from the single-sample setting, leading to performance degradation. In contrast, our method uses top-k selection, which remains unaffected by distribution shifts after batch aggregation. By periodically switching pruning patterns, our approach maintains consistent sparsity ratios while capturing important neurons from each sample, preserving performance in batched inference. Notably, our method is training-free, computing importance scores entirely from activations during inference.

3 Method

In this section, we present our batch-wise adaptive pruning approach for efficient LLM inference. Our method is training-free, requiring no fine-tuning; instead, importance scores are computed entirely from activations during inference. We first introduce the preliminaries on gated MLP structures, then describe our batch-wise importance scoring mechanism, and finally detail our three-phase pruning algorithm.

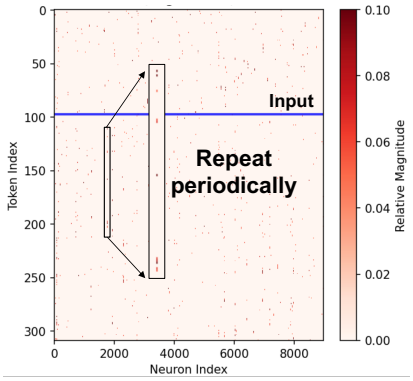

Figure 3: Activation visualization of $\mathbf{Z}$ along the token dimension in an FFN layer of DeepSeek-R1-Distill-Qwen-1.5B. Neurons with high magnitude exhibit periodic patterns. Based on this observation, our method adopts periodic mask updates rather than updating pruning positions at every token.

3.1 Preliminaries

Modern transformer architectures employ gated MLP structures in their feedforward blocks (Grattafiori et al., 2024; Yang et al., 2024). Given a sequence of input hidden states $\mathbf{X} \in \mathbb{R}^{T \times D}$ where T is the number of tokens, each hidden state $\mathbf{x} \in \mathbb{R}^{D}$ is processed by the gated MLP:

$$\mathbf{Z} = \sigma(\mathbf{W}_g \mathbf{x}) \odot (\mathbf{W}_1 \mathbf{x}) \tag{1}$$

where σ denotes the SiLU activation function and $\odot$ signifies element-wise multiplication. For all weight matrices, $\mathbf{W}_1, \mathbf{W}_g \in \mathbb{R}^{D_{\text{FF}} \times D}$ and $\mathbf{W}_2 \in \mathbb{R}^{D \times D_{\text{FF}}}$ where typically $D_{\text{FF}} \gg D$. We refer to $\mathbf{Z} = \text{FF}_1(\mathbf{x}) \in \mathbb{R}^{D_{\text{FF}}}$ as the FF activations, and the output is computed as $\text{FF}_2(\mathbf{Z}) = \mathbf{W}_2 \mathbf{Z}$.

3.2 Batch-wise Importance Neuron Scoring

We define an *importance score* $\mathbf{s} \in \mathbb{R}^{D_{\text{FF}}}$ as a measure of how critical each neuron in the FF layer is for the current task. Following prior work on activation-based pruning (Dong et al., 2024), we compute importance scores by aggregating normalized activation magnitudes across tokens.

For a sequence of T tokens, let $\mathbf{Z} = \text{FF}_1(\mathbf{X}) \in \mathbb{R}^{T \times D_{\text{FF}}}$ denote the FF activations for the full sequence. We first compute the relative activations $\overline{\mathbf{Z}}$ by applying row-wise ℓ_2 normalization: $[\overline{\mathbf{Z}}]_t = [\mathbf{Z}]_t / \|[\mathbf{Z}]_t\|_2$ for each token t. This normalization ensures that we capture the relative impor-

tance of each neuron within a token, rather than being dominated by tokens with large overall activation magnitudes. As described in Section 3.3, our method operates in three distinct phases, and we compute importance scores at each phase separately. The importance score at each phase n is then computed by taking the column-wise ℓ_2-norm:

$$\mathbf{s}_n = \mathrm{MS}(\mathbf{Z}) = \frac{1}{\sqrt{T}} \left[\|[\overline{\mathbf{Z}}]_{\cdot,1}\|_2, \ldots, \|[\overline{\mathbf{Z}}]_{\cdot,D_{\mathrm{FF}}}\|_2 \right]^\top \tag{2}$$

where n denotes the phase index, and $\mathrm{MS}(\cdot)$ denotes the batch-wise scoring function. Intuitively, neurons with consistently high relative activations across multiple tokens receive higher importance scores, capturing the "persistently important" neurons that contribute meaningfully to the model's computation. Unlike prior work (Dong et al., 2024) that computes importance scores over the entire sequence in a single pass, our method computes importance scores at each phase separately. Since different phases may have varying numbers of tokens, we normalize by $\sqrt{T}$ (the square root of the number of tokens) to eliminate length bias, enabling fair comparison of importance scores across phases.

Batch Aggregation. For batched inference with B samples, we aggregate importance scores across the batch, excluding padding tokens and EOS tokens from the aggregation. For a single sample ($B=1$), we directly use $\mathbf{s}_n$. For multiple samples ($B > 1$), we aggregate using element-wise maximum:

$$\bar{\mathbf{s}}_n = \max_{b \in \{1,\ldots,B\}} \mathbf{s}_n^{(b)} \tag{3}$$

where $\mathbf{s}_n^{(b)}$ denotes the importance score for sample b at phase n. We then construct a *pruning mask* $\mathbf{M} \in \{0,1\}^{D_{\mathrm{FF}}}$, a binary vector indicating which neurons to prune. Specifically, we select the top-k neurons with the highest importance scores, that ensures consistent sparsity regardless of the aggregated activation distribution, where $k = \lfloor (1-\rho) \cdot D_{\mathrm{FF}} \rfloor$ and $\rho \in (0,1)$ is the target sparsity ratio. The shared pruning mask is then applied uniformly across all samples in the batch. This aggregation method ensures that neurons important to any sample are retained, while the top-k selection maintains the target sparsity ratio regardless of the aggregated distribution.

3.3 Batch-wise Adaptive Pruning Algorithm

Our batch-wise adaptive approach operates in three distinct phases during autoregressive generation, as illustrated in Figure 2. The key insight is that we balance computational efficiency with adaptivity by alternating between sparse computation and dense exploration phases.

Phase 1: Input Phase. During the initial prompt processing with T_{input} input tokens, we compute the FF activations $\mathbf{Z}_t$ for each token $t = 1, \ldots, T_{\text{input}}$. Using these activations, we compute the initial importance score $\mathbf{s}_0$ using the batch-wise scoring function (Equation 2).

Phase 2: Dense Exploration Phase. After the prompt, we perform T_{init} steps of dense computation. This dense exploration phase serves two purposes: (1) it allows the model to generate initial tokens using full capacity, which is particularly important for establishing the direction of reasoning, and (2) it enables us to collect activation statistics that better reflect the generation context rather than just the input. At the end of this phase, we compute a new importance score $\mathbf{s}_1$ from the collected activations and update our *importance memory* $\mathbf{m} \in \mathbb{R}^{D_{\mathrm{FF}}}$ using an element-wise maximum: $\mathbf{m}_1 \leftarrow \max(\mathbf{s}_0, \mathbf{s}_1)$. The importance memory $\mathbf{m}$ serves as a buffer that accumulates neuron importance information across phases, ensuring that neurons identified as important in earlier phases remain candidates for selection while incorporating new information from subsequent phases. We then generate the initial pruning mask $\mathbf{M}_1 = \text{top-}k(\mathbf{m}_1, k)$ and immediately begin sparse computation in the next phase.

Phase 3: Periodic Adaptive Pruning Phase. As shown in Figure 3, neurons with high activation magnitudes exhibit periodic patterns during autoregressive generation. Based on this observation, we periodically update the pruning mask to adapt to these evolving activation patterns, enabling effective performance even for reasoning tasks with long output sequences.

After the initial dense exploration, we alternate between cycles of sparse and dense computation. Each cycle n consists of two stages: (1) *pruning*, where we perform T_p steps of sparse forward passes using the current pruning mask, and (2) *exploration*, where we perform T_e steps of dense computation while collecting activations to update the importance scores. At the end of each explo-

Table 2: Pruning performance comparison at 50% target sparsity with batch size 4. Our method significantly outperforms all pruning baselines (Wanda, Griffin, TEAL) across multiple reasoning benchmarks (GSM8K, MATH500, MINERVA, AMC23, GPQA-DIAMOND) on both DeepSeek-R1-Distill-Qwen-7B and DeepSeek-R1-Distill-Llama-8B models. Wanda applies 2:4 structured sparsity. Bold values indicate the highest performance among all pruning methods.

Model	Method	Tasks					AVG
		GSM8K	MATH500	MINERVA	AMC23	GPQA-DIAMOND	
DeepSeek-R1-Distill-Qwen-7B	Dense	92.0	91.8	39.7	87.5	52.5	72.7
	Wanda	59.0	24.4	7.4	12.5	15.7	23.8
	Griffin	22.0	14.8	11.0	10.0	15.2	14.6
	TEAL	30.0	11.2	4.8	10.0	15.7	14.3
	Ours	**89.0**	**71.0**	**29.4**	**50.0**	**30.8**	**54.0**
DeepSeek-R1-Distill-Llama-8B	Dense	96.0	91.0	33.8	90.0	45.5	71.3
	Wanda	9.0	4.8	0.7	5.0	7.1	5.3
	Griffin	16.0	11.8	4.8	2.5	16.7	10.4
	TEAL	28.0	5.4	2.2	0.0	16.2	10.4
	Ours	**75.0**	**38.8**	**12.1**	**25.0**	**21.7**	**34.5**

ration stage, we update the importance memory using the maximum of the current score and the previous memory: $\mathbf{m}_n \leftarrow \max(\mathbf{m}_{n-1}, \mathbf{s}_n)$, and generate a new pruning mask $\mathbf{M}_n = \text{top-}k(\mathbf{m}_n, k)$. This update rule accumulates importance information across multiple cycles, ensuring that persistently important neurons are retained while allowing the mask to adapt to evolving activation patterns. Specifically, the pruning mask is updated every $T_{\text{trans}} = T_e + T_p$ steps, alternating between sparse pruning and dense exploration phases.

During the pruning stage, we leverage the pruning mask to reduce computational costs. Given a mask $\mathbf{M}$, we define the set of retained neuron indices as $\mathcal{I} = \{i : \mathbf{M}[i] = 0\}$, where $|\mathcal{I}| = (1-\rho) \cdot D_{\text{FF}}$. The pruned weight matrices are obtained by selecting the corresponding rows: $\widehat{\mathbf{W}}_g, \widehat{\mathbf{W}}_1 \in \mathbb{R}^{k \times D}$ and $\widehat{\mathbf{W}}_2 \in \mathbb{R}^{D \times k}$. The sparse forward pass then computes:

$$\widehat{\mathbf{Z}} = \sigma(\widehat{\mathbf{W}}_g \mathbf{x}) \odot (\widehat{\mathbf{W}}_1 \mathbf{x}), \quad \mathbf{y} = \widehat{\mathbf{W}}_2 \widehat{\mathbf{Z}} \quad (4)$$

where $\widehat{\mathbf{Z}} \in \mathbb{R}^k$. This structured pruning approach enables efficient matrix operations on modern GPUs, as the reduced dimensions lead to proportionally smaller matrix multiplications.

4 Experiment

4.1 Models and Datasets

We evaluate our proposed batch-wise adaptive pruning method on two reasoning models: DeepSeek-R1-Distill-Llama-8B and DeepSeek-R1-Distill-Qwen-7B (DeepSeek-AI, 2025).

We evaluate on diverse reasoning benchmarks using the prior work evaluation framework (Yue et al., 2025). For mathematical reasoning, we use TinyGSM8K (Cobbe et al., 2021; Polo et al., 2024), a 100-sample subset of GSM8K (grade school math word problems) selected to reduce computational costs while maintaining evaluation reliability. We also include MATH500 (Hendrycks et al., 2021) for competition-level mathematics, MINERVA Math (Lewkowycz et al., 2022) for scientific reasoning, and AMC23 (Hendrycks et al., 2021) from the 2023 American Mathematics Competition. For general reasoning, we use GPQA-DIAMOND (Rein et al., 2024), a graduate-level science QA benchmark, following Open R1 (Hugging Face, 2025) evaluation settings.

4.2 Baselines

Because our proposed method is training-free, we compare against training-free pruning methods. Wanda (Sun et al., 2024) is an input-independent method that applies 2:4 structured sparsity after calibrating on the C4 dataset with sequences of 2048 tokens. Griffin (Dong et al., 2024) employs input-dependent but static pruning, capturing important neurons from activations during the prompt phase and using the determined pruned weights throughout decoding. TEAL (Liu et al., 2025) is an adaptive pruning method that uses threshold-based selection, calibrating on C4 to extract activation thresholds and pruning weights with activations below the threshold during inference while applying uniform sparsity across all layers. For batched evaluation, following the original approach (Liu et al., 2025), we aggregate activations across samples within a batch using mean, excluding padding

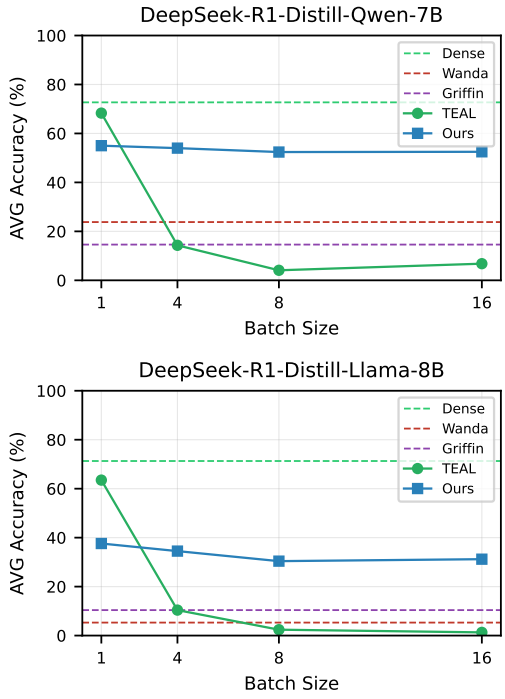

Figure 4: Average accuracy across five reasoning tasks (GSM8K, MATH500, MINERVA, AMC23, GPQA-DIAMOND) with varying batch sizes at 50% target sparsity.

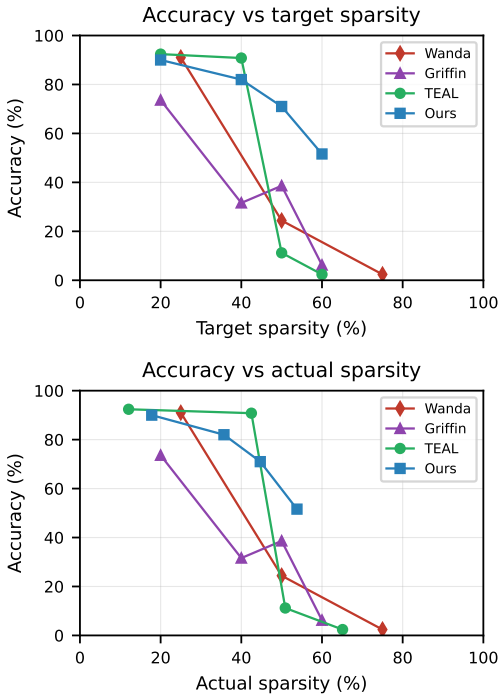

Figure 5: Accuracy on MATH500 across different target sparsity ratios with batch size 4. Target sparsity refers to the intended pruning ratio, a hyperparameter set before inference, while actual sparsity indicates the proportion of neurons that were actually pruned.

tokens and EOS tokens from the aggregation, then apply the single-sample algorithm, resulting in all samples sharing the same pruned weights. Since Dense and Wanda performances are batch-size independent, we report their results using batch size 8 for all experiments.

4.3 Results

Our method outperforms baselines in batched settings. Table 2 presents the performance comparison at 50% target sparsity with batch size 4. Our proposed method significantly outperforms all baseline approaches across multiple reasoning benchmarks on both DeepSeek-R1-Distill-Qwen-7B and DeepSeek-R1-Distill-Llama-8B models. On the Qwen-7B model, our method achieves an average accuracy of 54.0% across the five tasks, compared to TEAL's 14.3%, representing a 39.7 percentage point improvement. Similarly, on the Llama-8B model, our approach attains 34.5% average accuracy versus TEAL's 10.4%, a 24.1 percentage point improvement. Static pruning methods such as Wanda and Griffin exhibit poor performance on reasoning tasks, as they cannot adapt to the evolving activation patterns during long chain-of-thought generation. Griffin achieves only 14.6% and 10.4% average accuracy on Qwen-7B and Llama-8B respectively, while Wanda obtains 23.8% and 5.3%. Although TEAL employs adaptive threshold-based pruning, its performance degrades significantly in batched settings due to distribution shifts when aggregating activations across multiple samples.

Our method maintains consistent performance across all batch sizes. Figure 4 illustrates how average accuracy across the five reasoning tasks varies with batch size at 50% target sparsity. TEAL demonstrates high performance at batch size 1 but experiences substantial degradation as the batch size increases. This performance drop stems from the distribution shift introduced by sample-wise activation aggregation: as batch size grows, the aggregated activation patterns during batched inference diverge from the single-sample calibration settings used during threshold extraction. In contrast, our proposed method maintains robust and consistent performance across all batch sizes, demonstrating its practical advantage for batched deployment scenarios where batching is essential for throughput optimization. Detailed numerical results across all batch sizes are provided in Table 4 in Appendix.

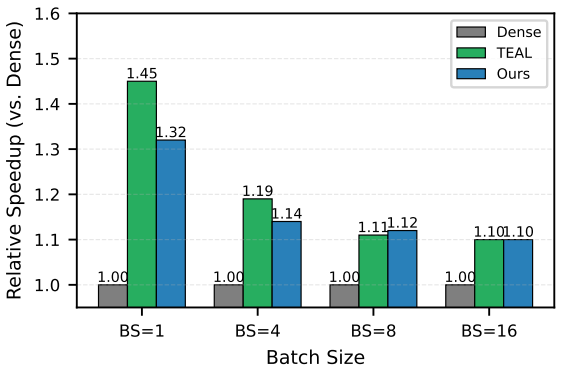

Figure 6: Relative generation throughput speedup on DeepSeek-R1-Distill-Qwen-7B across batch sizes compared to the dense baseline.

Our method maintains strong performance up to high sparsity levels. Figure 5 shows performance on MATH500 across different target sparsity levels with batch size 4. Notably, TEAL exhibits a significant discrepancy between target and actual sparsity: at 20% target sparsity, it achieves only 12.07% actual sparsity, while at 60% target, it overshoots to 65.10% actual sparsity. This unpredictable sparsity control leads to performance collapse at higher target sparsity levels, dropping to 11.2% accuracy at 50% target sparsity and 2.4% at 60%. In contrast, our method maintains actual sparsity closely aligned with the target (17.8%, 35.7%, 44.7%, and 53.8% for 20%, 40%, 50%, and 60% targets respectively), enabling stable performance across all sparsity levels with 71.0% accuracy at 50% and 51.6% at 60% target sparsity.

Our method achieves practical speedup in batched scenarios. We evaluate end-to-end decoding speed on an NVIDIA H200 GPU following the GPT-Fast (PyTorch, 2024) benchmarking setup. Both TEAL and our method employ custom Triton kernels; implementation details are available in our code repository. Figure 6 illustrates the generation throughput speedup across different batch sizes at 50% target sparsity. At batch size 1, our method achieves 1.32x speedup over the dense baseline. As batch size increases and the workload becomes more computation-bound, our method maintains comparable speed to the dense baseline while exhibiting less throughput degradation than TEAL, confirming its practical efficiency advantage in batched deployment scenarios. Detailed throughput values and speed-accuracy tradeoffs are provided in Appendix.

Table 3: Hyperparameter ablation study on MATH500 dataset with batch size 4, examining the effects of T_{init} (initial dense steps), T_E (exploration steps), and T_{trans} (update period) on accuracy and inference speed.

T_{init}	T_E	T_{trans}	Accuracy	Speed (token/sec)
0	2	20	68.6	925.3
32	2	20	70.8	925.3
64	2	20	71.0	925.3
128	2	20	71.4	925.3
64	1	20	67.4	941.0
64	2	20	71.0	925.3
64	4	20	77.2	896.8
64	2	10	77.4	806.3
64	2	20	71.0	925.3
64	2	30	69.6	958.7

Hyperparameter ablation study. Table 3 presents ablation results on MATH500 with batch size 4. Increasing T_{init} (initial dense steps) from 0 to 128 improves accuracy by 2.8% with minimal speed impact, as it captures post-prompt activation shifts. Increasing T_E (exploration steps) from 1 to 4 yields 9.8% accuracy gain but reduces throughput from 941.0 to 896.8 tokens/sec. Conversely, extending T_{trans} (update period) from 10 to 30 degrades accuracy by 7.8% while increasing throughput from 806.3 to 958.7 tokens/sec.

In summary, T_{init} primarily affects accuracy without speed penalty, while T_E and T_{trans} present accuracy-speed trade-offs. Hyperparameter values can be selected based on deployment requirements.

5 Conclusion

In this work, we presented a training-free batch-wise adaptive pruning method specifically designed for batched inference scenarios in LRMs. Through periodic pruning and importance memory mechanism, our approach maintains strong performance on reasoning tasks even in batched settings while enabling faster inference of reasoning models.

Our experiments revealed that static pruning methods fail to adapt to the evolving activation patterns inherent in reasoning tasks, while existing adaptive pruning methods suffer significant performance degradation in batched settings. In contrast, our proposed method demonstrates robust performance across varying batch sizes and sparsity levels.

Limitation

Our proposed method achieves efficient inference for reasoning models and tasks in batched settings

by periodically pruning weights based on activation patterns observed during generation. However, this approach requires direct access to intermediate activations within the model, which limits its applicability to closed-source or API-based models where internal states are not exposed.

References

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. Slicegpt: Compress large language models by deleting rows and columns. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Harry Dong, Beidi Chen, and Yuejie Chi. 2024. Prompt-prompted adaptive structured pruning for efficient llm generation. In *First Conference on Language Modeling*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 10323–10337. PMLR.

Amir Gholami, Zhewei Yao, Sehoon Kim, Michael W. Mahoney, and Kurt Keutzer. 2024. Ai and memory wall. *arXiv preprint arXiv:2403.14123*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. In *Advances in Neural Information Processing Systems*, volume 34, pages 9380–9392.

Hugging Face. 2025. Open r1: A fully open reproduction of deepseek-r1.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. 2024. Shortened llama: A simple depth pruning for large language models. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models (ME-FoMo)*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023*. ACM.

Je-Yong Lee, Donghyun Lee, Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini. 2024. Cats: Contextually-aware thresholding for sparsity in large language models. In *Conference on Language Modeling (COLM) 2024*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 3843–3857.

James Liu, Pragaash Ponnusamy, Tianle Cai, Han Guo, Yoon Kim, and Ben Athiwaratkun. 2025. Training-free activation sparsity in large language models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net. Spotlight.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*.

OpenAI. 2024. Learning to reason with llms.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. 2024. tinybenchmarks: evaluating llms with fewer examples. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Team PyTorch. 2024. Accelerating generative ai with pytorch ii: Gpt, fast. PyTorch Blog.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *The Twelfth International Conference on Learning Representations*.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2024. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024*. OpenReview.net.

Gemini Team. 2023. Gemini: A family of highly capable multimodal models.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024. Sheared llama: Accelerating language model pre-training via structured pruning. In *The Twelfth International Conference on Learning Representations, ICLR 2024*. OpenReview.net.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, and 1 others. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. 2025. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? In *Advances in Neural Information Processing Systems 38*. Oral.

A Full Performance Comparison

Table 4 extends the main results (Table 2) to all batch sizes. On Distill-Qwen-7B, TEAL's average accuracy drops from 68.3% (BS=1) to 6.8% (BS=16), while our method maintains 55.0% to 52.5%. On Distill-Llama-8B, TEAL degrades from 63.5% to 1.3%, whereas ours remains between 37.6% and 31.2%.

B Generation Throughput Analysis

All throughput experiments in this section are conducted on NVIDIA H200 GPUs. Tables 5 and 6 report the exact throughput values corresponding to Figure 6 in the main text. On Distill-Qwen-7B, our method achieves 1.32× speedup at BS=1 and 1.10× at BS=16. On Distill-Llama-8B, our method achieves 1.30× at BS=1 and 1.15× at BS=16. Figure 7 visualizes the Distill-Llama-8B speedup trends.

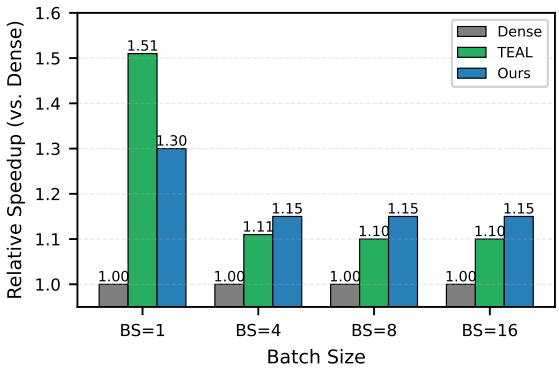

Figure 7: Relative generation throughput speedup on DeepSeek-R1-Distill-Llama-8B across batch sizes compared to the dense baseline.

C Speed-Accuracy Trade-off

Figures 8 and 9 visualize the throughput versus accuracy trade-off across batch sizes. TEAL shows high accuracy at BS=1 but loses both accuracy and relative speedup as batch size increases. Our method maintains consistent accuracy across all batch sizes with competitive throughput.

D Dataset and Model Details

We summarize the evaluation datasets and models used in this work below.

D.1 Dataset Details

We report the split, number of examples, and license for each dataset.

Table 4: Pruning performance comparison at 50% target sparsity across different batch sizes. Our method significantly outperforms all pruning baselines (Wanda, Griffin, TEAL). Dense, Wanda, and Griffin results are batch-size independent. Bold values indicate superior performance compared to TEAL at the same batch size.

Model	Method	Batch Size	Tasks					AVG
			GSM8K	MATH500	MINERVA	AMC23	GPQA-DIAMOND	
DeepSeek-R1-Distill-Qwen-7B	Dense	–	92.0	91.8	39.7	87.5	52.5	72.7
	Wanda	–	59.0	24.4	7.4	12.5	15.7	23.8
	Griffin	–	22.0	14.8	11.0	10.0	15.2	14.6
	TEAL	1	91.0	86.8	40.1	80.0	43.4	68.3
		4	30.0	11.2	4.8	10.0	15.7	14.3
		8	9.0	3.2	3.3	0.0	5.0	4.1
		16	0.0	1.6	0.7	30.0	1.5	6.8
	Ours	1	85.0	72.0	26.1	57.5	34.3	55.0
		4	89.0	71.0	29.4	50.0	30.8	54.0
		8	84.0	70.8	25.7	55.0	26.3	52.4
		16	86.0	71.8	30.1	45.0	29.8	52.5
DeepSeek-R1-Distill-Llama-8B	Dense	–	96.0	91.0	33.8	90.0	45.5	71.3
	Wanda	–	9.0	4.8	0.7	5.0	7.1	5.3
	Griffin	–	16.0	11.8	4.8	2.5	16.7	10.4
	TEAL	1	95.0	81.8	30.9	70.0	39.9	63.5
		4	28.0	5.4	2.2	0.0	16.2	10.4
		8	2.0	3.8	2.6	2.5	1.0	2.4
		16	1.0	3.2	2.2	0.0	0.0	1.3
	Ours	1	65.0	48.0	14.0	35.0	25.8	37.6
		4	75.0	38.8	12.1	25.0	21.7	34.5
		8	69.0	39.0	8.8	12.5	22.7	30.4
		16	65.0	39.6	9.6	25.0	16.7	31.2

Table 5: Generation throughput (tokens/sec) and speedup comparison on DeepSeek-R1-Distill-Qwen-7B relative to Dense baseline across different batch sizes.

Method	BS=1	BS=4	BS=8	BS=16
Dense	200.9 (1.00×)	813.0 (1.00×)	1596.0 (1.00×)	3040.5 (1.00×)
TEAL	290.7 (1.45×)	971.1 (1.19×)	1769.1 (1.11×)	3335.9 (1.10×)
GRIFFIN	323.5 (1.61×)	1084.5 (1.33×)	2111.0 (1.32×)	3854.3 (1.27×)
Ours	265.6 (1.32×)	925.3 (1.14×)	1794.7 (1.12×)	3331.8 (1.10×)

- **TinyGSM8K**: split = test, #examples = 100, license = MIT.
- **MATH500**: split = test, #examples = 500, license = MIT.
- **MINERVA Math**: split = test, #examples = 272, license = MIT.
- **AMC23**: split = test, #examples = 40, license = unspecified (MAA copyrighted).
- **GPQA-DIAMOND**: split = test, #examples = 198, license = CC BY 4.0.

D.2 Model Details

We evaluate on two reasoning models and use a consistent zero-shot inference protocol.

- **DeepSeek-R1-Distill-Qwen-7B**: a distilled reasoning model (7B). License = MIT (model weights/repository). Base lineage = Qwen-2.5 series (Apache 2.0). Evaluation uses the official DeepSeek-R1 chat template, greedy decoding with temperature 0 (zero-shot), accuracy as the metric, and maximum generation length of 16,000 tokens.
- **DeepSeek-R1-Distill-Llama-8B**: a distilled reasoning model (8B). License = MIT (model weights/repository). Base lineage = Llama-3.1-8B-Base (Llama 3.1 license). We use the same evaluation protocol: official DeepSeek-R1 chat template, greedy decoding with temperature 0 (zero-shot), accuracy as the metric, and maximum generation length of 16,000

Table 6: Generation throughput (tokens/sec) and speedup comparison on DeepSeek-R1-Distill-Llama-8B relative to Dense baseline across different batch sizes.

Method	BS=1	BS=4	BS=8	BS=16
Dense	175.0 (1.00×)	755.2 (1.00×)	1471.7 (1.00×)	2713.0 (1.00×)
TEAL	263.8 (1.51×)	837.2 (1.11×)	1618.2 (1.10×)	2997.4 (1.10×)
GRIFFIN	269.5 (1.54×)	1010.5 (1.34×)	1957.1 (1.33×)	3747.6 (1.38×)
Ours	227.3 (1.30×)	868.1 (1.15×)	1685.9 (1.15×)	3124.9 (1.15×)

tokens.

E Use of Large Language Models

We employed a large language model (ChatGPT; "GPT-5") for English-language polishing, light copy-editing, and assisting with experimental code implementation. The model was not used to generate research ideas or experimental design. All technical content, claims, and experimental code were authored and verified by the authors to ensure correctness and reproducibility. No non-public data, confidential information, or personally identifiable information was provided to the model.

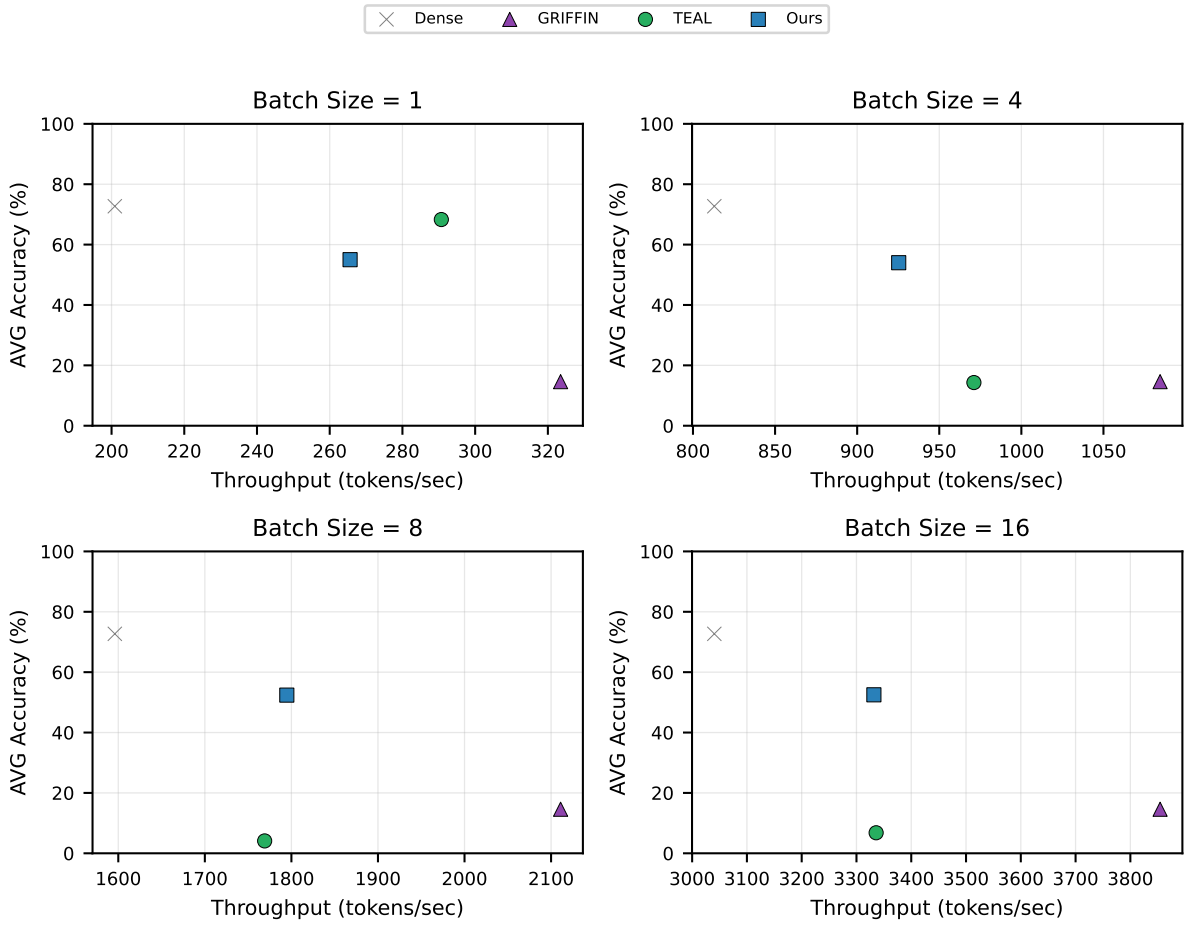

Figure 8: Throughput versus average accuracy trade-off on DeepSeek-R1-Distill-Qwen-7B across different batch sizes at 50% target sparsity. Each subplot shows the speed-accuracy relationship for a specific batch size. Our method maintains consistent accuracy across batch sizes while achieving competitive throughput, whereas TEAL's accuracy degrades significantly as batch size increases.

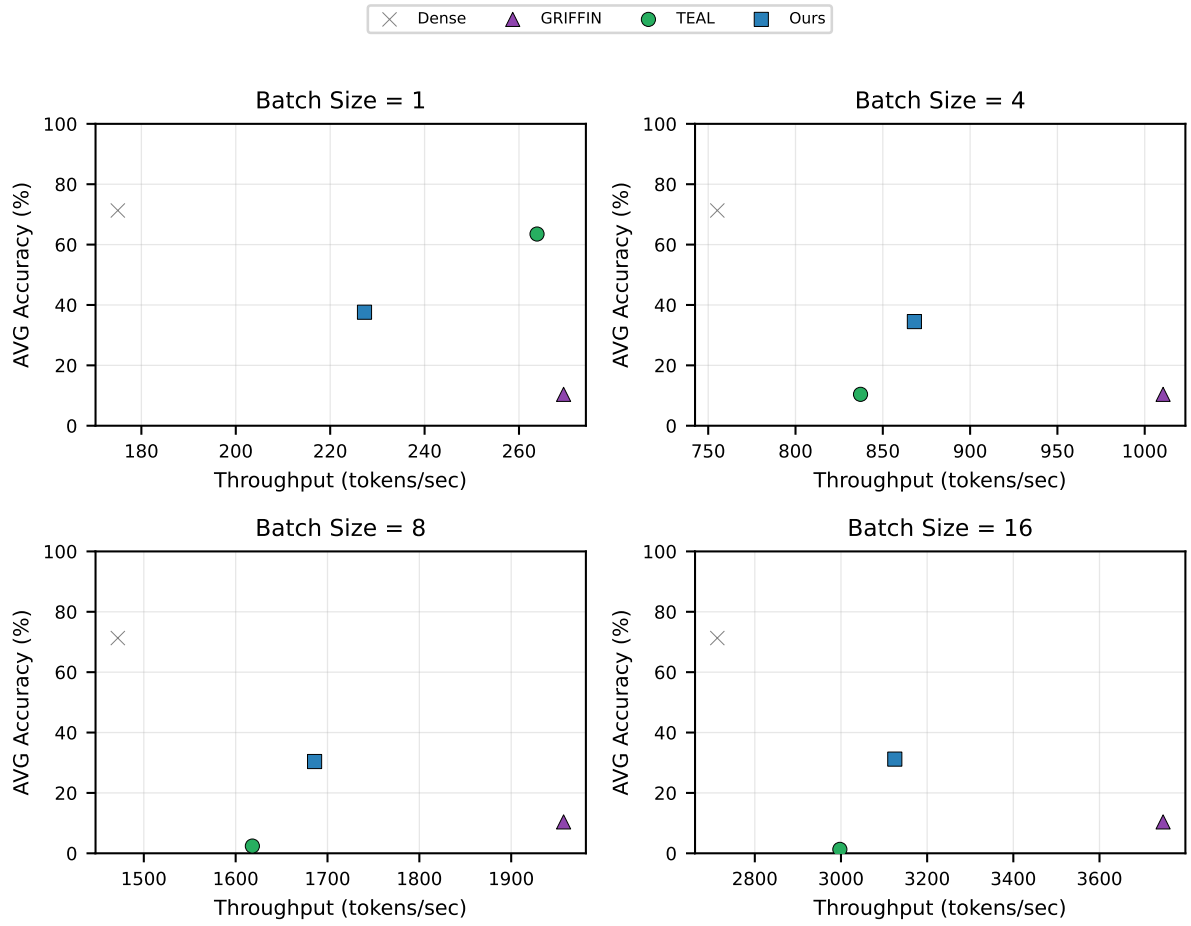

Figure 9: Throughput versus average accuracy trade-off on DeepSeek-R1-Distill-Llama-8B across different batch sizes at 50% target sparsity. Each subplot shows the speed-accuracy relationship for a specific batch size. Our method maintains consistent accuracy across batch sizes while achieving competitive throughput, whereas TEAL's accuracy degrades significantly as batch size increases.