Herd: Grouping before Pruning for Batch Inference

Yuwei An, Zhuoming Chen, Chenyan Xiong, Beidi Chen
Carnegie Mellon University
yuweia, zhuominc, cx, beidic@andrew.cmu.edu

Abstract

With the rapid development of Large Language Models(LLMs), these technologies are now extensively utilized across a variety of natural language processing applications. On the other hand, the vast number of parameters results in significant computational costs and resource usage. To achieve more efficient model inference, methods such as pruning are employed to capitalize on model sparsity and decrease computational demands during the inference process. However, previous pruning approaches typically face one major problem which is the restriction to the inference scenario with batch size of one. To address this issue, we have developed a new method, *Herd*, which leverages contextual sparsity similarity across inputs to group data into batches and dynamically selects activation parameters for pruning during batch inference. Experiments show that Herd not only maintains the performance of the original model but also significantly improves inference efficiency. From a downstream performance perspective, Herd mitigates the degradation typically observed in batch inference with dynamic pruning. Simultaneously, Herd enables efficient batch inference and achieves multi-fold improvements in throughput.

1 Introduction

Large Language Models (LLMs) have had a significant impact on a wide range of natural language processing domains and applications. However, with billions of parameters, the inference of LLMs typically requires substantial computational resources and time. Pruning is a simple and effective way to reduce computational parameters and time costs. It identifies and removes redundant parameters from the original model, generating a pruned model with fewer parameters and higher computational speed.

Recently, a new method called dynamic pruning has been introduced (Liu et al., 2023). Dynamic pruning leverages **contextual sparsity**, which refers to a small, input-specific set of parameters that can approximate the output of the original model. Contextual sparsity is a widely observed phenomenon in modern LLMs and substantial research has focused on developing more effective dynamic pruning algorithms (Dong et al., 2024; Lee et al., 2024; Liu et al., 2024). However, a key limitation of existing methods is their applicability only in scenarios where the batch size is equal to one. This restriction arises from the variability in contextual sparsity across different inputs within the same batch, which makes it challenging to align sparsity patterns and efficiently perform dynamic pruning in larger batch settings.

In batch inference scenarios, different input data within the same batch may require distinct sets of activated parameters during dynamic pruning inference. There are two primary approaches to address this mismatch of contextual sparsity among data in the same batch. The first approach is the Union strategy, which combines the sets of activated parameters for all data in the batch to meet the requirements of every input. However, result in Appendix B shows that this strategy significantly reduces the sparsity ratio, leading to limited speedup potential in dynamic pruning.

The second approach is Eviction, where all data in the batch share the same set of activated parameters under a fixed pruning ratio. In this scenario, each input sacrifices accuracy because of potentially missing essential parameters. While eviction provides a feasible solution for dynamic pruning in batch inference scenario, it risks performance degradation compared to dynamic pruning with single input because the correct parameters may not always be available. Figure 1 highlights the performance degradation caused by this mismatch, showing that batch inference experiences significantly worse downstream performance with randomly grouped data batch.

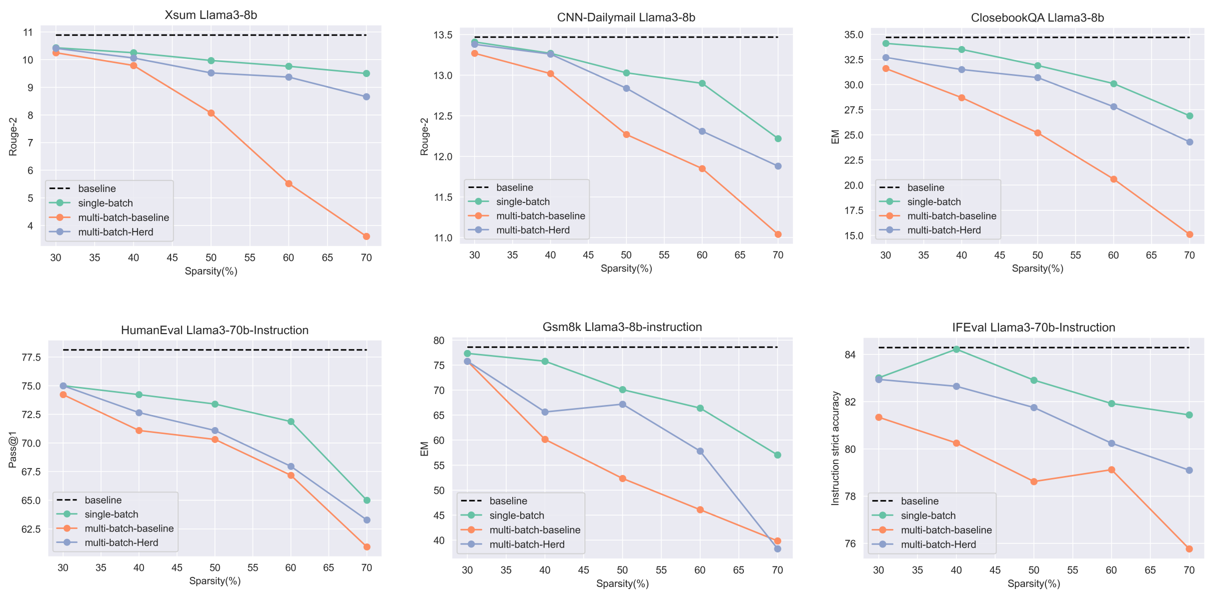


Figure 1: Downstream performance comparison of various inference strategies. The black dashed line indicates the performance of the model without pruning. The green line shows the performance of Dynamic Pruning with batch size = 1. The orange line shows the performance of Dynamic Pruning in a batch inference scenario with randomly grouped batches. The blue line shows the performance of the Herd strategy, which applies CSPs to group data with higher contextual sparsity similarity prior to dynamic pruning.

Previous work (Liu et al., 2023, 2024) introduced the high-level concept that in dynamic pruning, grouping data with similar activated parameters via an eviction strategy can reduce conflicts and ensure that each input achieves the expected activated parameters to the greatest extent possible. This oracle solution minimizes performance degradation in batch inference scenarios. We define this phenomenon as **sparsity similarity** between data. However, sparsity similarity can only be detected or verified after inference is completed, making it impractical for achieving real-time inference speedups. Therefore, a critical problem is:

how to detect data with high sparsity similarity as early as possible and group them to perform batch inference?

To address this problem, we introduced *Herd*, an algorithm designed for batch inference with dynamic pruning. Specifically, there are two main challenges Herd solves:

How to detect data with high sparsity similarity as early as possible: Herd introduces **contextual sparsity similarity**, which can be detected during the pre-filling stage and represents the sparsity similarity between data samples. To achieve this, we define the **contextual sparsity pattern** (CSP)—a bitwise vector that identifies the parameters most likely to be activated in a Feed Forward (FF) block. The CSP is generated during the pre-filling stage of LLM inference, providing a preview of the most frequently activated parameters for subsequent dynamic pruning. The similarity between CSPs across different data samples indicates the contextual sparsity similarity within a given FF block. Moreover, while CSP is inherently tied to individual FF blocks, a key observation reveals that the similarity of CSPs persists consistently across layers in the Transformer architecture (Vaswani et al., 2023). This cross-layer consistency enables grouping of data using CSPs collected only during the first stage, significantly reducing both computational overhead and storage requirements.

How to perform dynamic pruning after batch grouping: In the context of batch inference, experiments demonstrate that normalization at the batch level is both straightforward and effective. A simple adjustment allows previously established dynamic pruning methods to be applied in batch inference scenarios.

By addressing these two challenges, Herd enables batch inference with dynamic pruning. The contribution of Herd includes:

- Novelty: To the best of our knowledge, Herd is the first approach to explore the batch inference scenario for dynamic pruning.

- Simplicity: Herd is training-free and cost-efficient, requiring minimal external operations that are both straightforward and low in time complexity.
- Performance: As shown in Figure 1, Herd effectively mitigates the degradation caused by parameter eviction during batch inference. Experiments demonstrate that our algorithm significantly improves inference throughput in both on-device and offloading scenarios.
- Compatibility: Herd is compatible with most dynamic pruning algorithms including Griffin (Dong et al., 2024) and CATS (Lee et al., 2024).

2 Problem Definition

In this section we formally define the dynamic pruning problem using mathematical notation.

In the architecture of a transformer, a single transformer layer comprises an Attention block and a Feed Forward block (FF block). Dynamic pruning is typically applied to the Feed Forward block as it contains more parameters and exhibits higher sparsity compared to the Attention block. Presently, the prevalent trend in most popular models involves the adoption of Gated Mechanism within the FF block, which can be defined as:

$$\mathbf{H} = (\sigma(\mathbf{X}\mathbf{W}_{\mathbf{gate}}) \cdot (\mathbf{X}\mathbf{W}_{\mathbf{up}}))\mathbf{W}_{\mathbf{down}}$$

where $\mathbf{X} \in \mathbb{R}^{B \times Seq \times D_H}$ denotes the input of Feed Forward block, σ denotes the activation function, $\mathbf{W}_{\mathbf{gate}} \in \mathbb{R}^{D_H \times D_I}$, $\mathbf{W}_{\mathbf{up}} \in \mathbb{R}^{D_H \times D_I}$ and $\mathbf{W}_{\mathbf{down}} \in \mathbb{R}^{D_I \times D_H}$ denotes the model parameters of FF block. B, Seq, D_H and D_I represents batch size, sequence length, hidden states size and intermediate size respectively.

Typically, the goal of dynamic pruning is to slice parameters along the D_I dimension. Given specific input $\mathbf{X}$ the objective is to identify a subset of activated parameter indices $\mathcal{J}$. Each element $j \in \mathcal{J}$ indicates that the j-th row or column of the matrix $\mathbf{W}$ in the dimension of D_I should not be pruned. Most of recent dynamic pruning method is magnitude-based pruning which operates under the assumption that parameters with higher relative magnitudes are more critical to the model's performance.

3 Observation

In this section, we introduce two key observations that inform the design of Herd. In Section 3.1, we revisit a previous observation indicating that activated neurons during dynamic pruning can be estimated after the pre-filling stage. Based on this insight, we propose the Contextual Sparsity Pattern (CSP) to represent inherent contextual sparsity. Section 3.2 highlights the consistency of CSP similarity across layers, suggesting that calculating CSP similarity for a single FF block provides a reliable indication of similarity in other blocks due to this observed consistency.

3.1 Contextual Sparsity Pattern

Recent work (Dong et al., 2024) makes a key observation showing that neurons producing high relative magnitudes are naturally shared across tokens within a sequence. Due to the alignment in the sequence level, the set of activate parameters $\mathcal{J}$ can be determined in the pre-filling stage and applied along the decoding stage.

We are interested in this phenomenon as the set of activated parameters $\mathcal{J}$ can be viewed as an inherent representation of contextual sparsity. It indicates the most likely activated neurons during the decoding stage. Therefore, when $\mathcal{J}$ is highly similar across data pairs, it is natural to expect that they will share more parameters and reduce competition when batched together. We introduce the contextual sparsity pattern (CSP) as the mathematical representation of $\mathcal{J}$, where CSP is a bit vector: a value of 1 indicates parameters that are retained, while 0 represents those that are pruned.

In Herd, the CSP is computed in the pre-filling stage. Specifically, for a FF block, it is calculated as follows:

$$V_{\text{dot}} = \sigma\left(\mathbf{X}\mathbf{W}_{\text{gate}}\right) \cdot \mathbf{X}\mathbf{W}_{\text{up}}$$

$$t = \mathit{Threshold}\left(\left\|\frac{V_{\text{dot}}}{\|V_{\text{dot}}\|_{\text{dim}=2}}\right\|_{\text{dim}=1}, \mathcal{R}\right)$$

$$V_{CSP} = \left\{ m_i \,\middle|\, m_i = \begin{cases} 1 & \text{if } m_i > t \\ 0 & \text{else} \end{cases} \right\}$$

$\mathbf{X} \in \mathbb{R}^{1 \times Seq \times D_H}$ represents the input prompt, while $\mathcal{R}$ represents the pruning ratio. The output V_{CSP} is the contextual sparsity pattern, which defines the sparsity pattern of the Feed Forward block throughout inference for the given input $\mathbf{X}$.

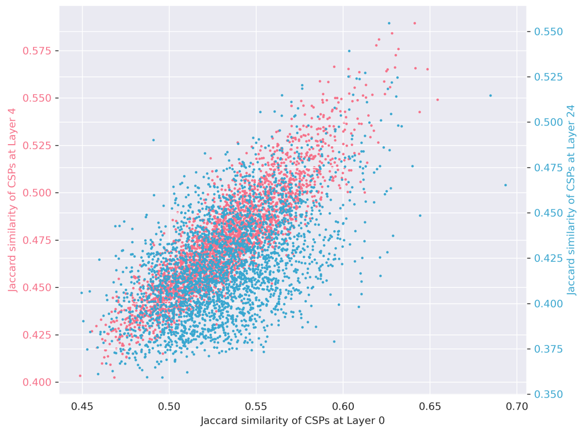


Figure 2: Consistency of similarity in CSP across layers: Each point in the figure represents a pair of data samples chosen from the C4 dataset (Raffel et al., 2023). The X and Y axes correspond to the Jaccard similarity of the CSP at layers i and j of Meta- Llama-3-8B model (AI@Meta, 2024), respectively. For pairs of data that exhibit high Jaccard similarity at one layer, there tends to be a linear correlation in the similarity between these pairs across other layers. The sparsity ratio $\mathcal{R}$ to retrieve CSPs is 0.5.

However, CSP is calculated only for one FF block and high similarity in CSPs of input data can only indicate that these data points are likely to share the same parameters for this specific FF block during decoding steps. After this layer, data would typically need to be regrouped based on the CSPs of the next layer, making it challenging to achieve speedup during inference. Fortunately, we observed consistency in CSP similarity across layers, allowing us to avoid such regrouping and streamline the process further.

Since CSPs are represented as bit vectors, we use the Jaccard similarity (Real and Vargas, 1996) to quantify the similarity between two patterns. Figure 2 demonstrates a clear linear relationship in the Jaccard similarity across different layers, especially in the region with higher Jaccard similarity. Figure 3 illustrates the correlation coefficient of CSPs' Jaccard similarity between consecutive layers. The consistently high values across all layers indicate a strong linear relationship and suggest that the similarity of CSPs remains highly consistent as the layers progress. This indicates that if data exhibits higher similarity at a certain layer, it is highly likely to maintain higher similarity in other layers as well.

3.2 Consistency of CSP similarity

The consistency of CSPs provides significant advantages for the Herd design. Firstly, it ensures

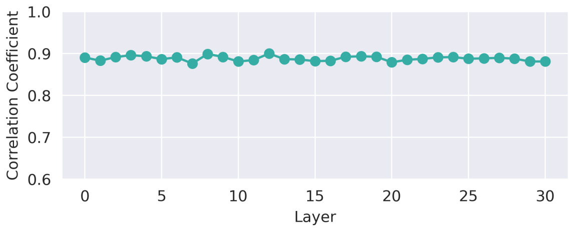


Figure 3: The correlation coefficient of CSPs' Jaccard similarity between consecutive layers. The sparsity ratio $\mathcal{R}$ to retrieve CSPs is 0.5.

that the contextual similarity among data remains consistent across blocks and layers. This means that when data with high similarity in a particular FF block are grouped together, the benefit of sharing similar parameters extends throughout the entire model. Secondly, rather than collecting CSPs from every layer, we can rely on the CSP from the first layer to represent the input's overall contextual similarity, thereby significantly reducing computational time and storage requirements.

4 Herd

In this section, we introduce our algorithm and system design, Herd. In section 3.1 and section 3.2, we introduce two main parts of Herd. In the batch grouping section, Herd collect the CSPs of data from the first FF block and use them to group data with high CSP similarity. In the batch dynamic pruning section, Herd performs the modified Griffin and CATS algorithm designed for batch inference. In section 3.3, we provide the overview of whole Herd design.

4.1 Batch Grouping

Based on previous observation, we define the contextual sparsity similarity as the Jaccard similarity between the CSPs of the data generated from the first FF block. For a given input pool, the CSP is calculated for each input at the first FF block of transformer structure. With the collected CSPs we further apply the K-means algorithm to group data with higher CSP similarity. Based on previous discussion, inputs within the same cluster exhibit higher contextual sparsity similarity and are more likely to share parameters during inference. After the cluster is grouped, we batch the input data from the same cluster for further inference.

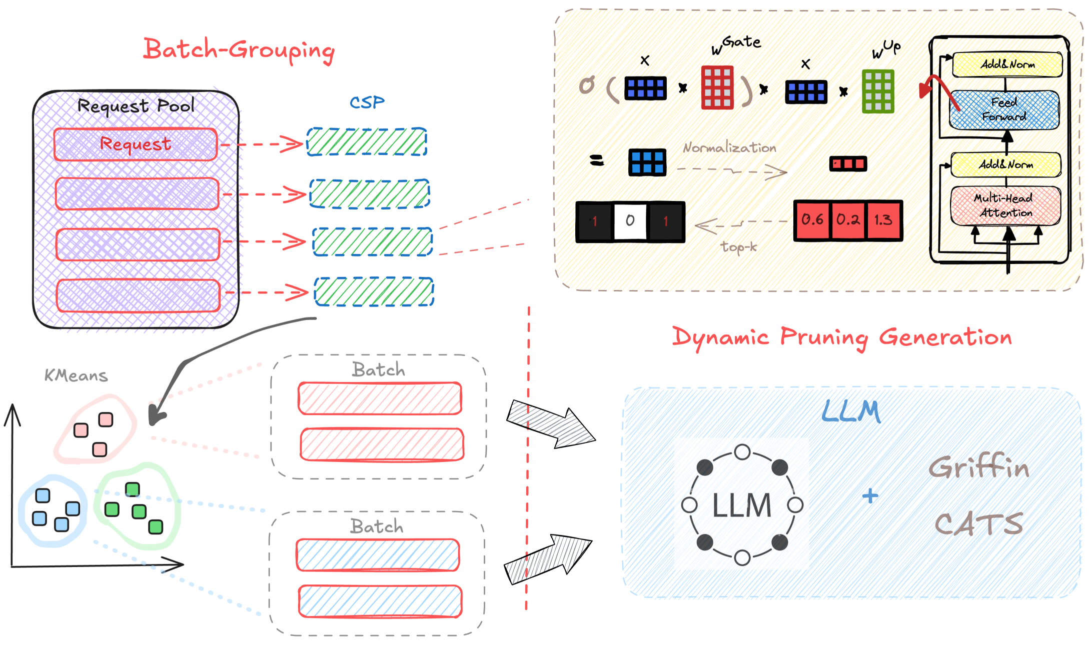


Figure 4: Overview of Herd: Herd consists of two primary stages. In the batch-grouping stage, the CSP is computed for each request, and data with higher similarity in CSPs are grouped into the same batch. In the dynamic pruning stage, Herd leverages adaptive pruning method (e.g. Griffin and CATS) to perform dynamic pruning across multiple batches, optimizing the inference process in a batch inference scenario.

4.2 Batch Dynamic Pruning

After the batch is grouped, we perform batch dynamic inference during the decoding stage. Herd adapts the algorithms from the two most recent and best-performing methods, Griffin and CATS, for dynamic pruning. Specifically, Griffin uses the CSP extracted from the pre-filling stage to determine which parameters are retained and which are pruned then applies it during the decoding stage. In contrast, CATS treats the output of the gate projection in the FF block as a routing mechanism to identify which parameters are most important. It applies a threshold $T_{\mathcal{R}}$ on the output of gate projection to decide which parameters to prune.

In the batch inference scenario, the primary difference is that the input shape during the decoding stage changes from $\mathbb{R}^{1\times1\times D_H}$ to $\mathbb{R}^{B\times1\times D_H}$. Therefore, determining the set of activated parameters for the entire batch becomes important. Our experiments show that applying a simple L1 norm along the B-dimension performs effectively. The absolute value of the magnitude can be interpreted as a measure of confidence for retaining parameters, with higher values being more likely to be preserved. Appendix C illustrates the details of the two dynamic pruning methods. The operations highlighted in red represent the key modifications necessary for their application in batch inference.

Algorithm 1 Herd Algorithm

```
1:  **Input:** Data pool $\mathcal{D}$, Model $M$, Number of clusters $C$, Batch size $B$
2:  **Output:** Decoded output from model $M$
3:  **Step 1: Batch Grouping**
4:  **for** each data point $d$ in $\mathcal{D}$ **do**
5:      collect CSP from Feed Forward Block 0 of $M$ for $d$
6:  **end for**
7:  Clusters $\hat{C} \leftarrow$ **KMeans**(CSPs, $C$)
8:  **Step 2: Generation**
9:  **for** each cluster $\mathcal{C}_i$ in **Clusters do**
10:     Batches $\hat{B} \leftarrow$ **GroupIntoBatches**($\mathcal{C}_i$, $B$)
11:     **for** each batch $\mathcal{B}$ in $\hat{B}$ **do**
12:         Perform prefilling on $\mathcal{B}$
13:         Decode $\mathcal{B}$ with Griffin or CATS
14:     **end for**
15: **end for**
```

Comparatively, Griffin employs a fixed dynamic pruning pattern based on the CSP, which allows it to prune more parameters overall by pruning across W_{gate}, W_{up}, and W_{down} with a fixed pruning ratio. In contrast, CATS prunes only W_{up} and W_{down}, but it dynamically selects the most appropriate parameters at each decoding step, leading to better downstream performance.

mode/$\mathcal{R}$	Xsum		CNN-Dailymail		CoQA		ClosebookQA	
	Rouge-2	Conv	Rouge-2	Conv	EM	F1	EM	F1
baseline	10.89	76.38	13.47	89.78	72.30	81.52	34.70	42.62
S/30%	10.43	74.50	13.41	87.72	71.60	80.83	34.10	41.94
S/40%	10.25	72.98	13.27	87.23	71.10	79.62	33.50	40.82
S/50%	9.97	71.24	13.03	86.81	70.80	78.19	31.90	39.28
S/60%	9.76	69.86	12.90	86.39	69.40	76.74	30.10	38.65
S/70%	9.50	68.12	12.22	84.96	67.70	75.88	26.90	36.74
R/30%	10.22	72.96	13.29	87.63	70.25	80.01	31.25	38.56
R/40%	9.71	63.13	13.01	86.63	67.65	77.58	28.55	36.87
R/50%	8.15	56.86	12.46	85.38	65.15	75.20	25.10	34.14
R/60%	5.59	40.67	11.92	84.67	63.95	75.12	20.50	30.72
R/70%	3.76	31.00	11.01	81.53	59.50	71.38	15.60	25.99
H/30%	10.41	71.64	13.38	87.43	70.90	80.56	32.70	40.14
H/40%	10.06	70.01	13.26	86.80	69.40	78.66	31.50	39.11
H/50%	9.52	68.82	12.84	86.27	67.90	76.80	30.70	39.05
H/60%	9.37	67.53	12.31	85.79	66.60	75.84	27.80	36.82
H/70%	8.66	65.98	11.88	83.91	65.30	73.42	24.30	33.41

Table 1: The result of In-context generation category tasks on Llama3-8B. The inference mode S, R and H is mentioned in section 5.2 and $\mathcal{R}$ represents the pruning ratio. The pruning algorithm is Griffin.

4.3 Overview

Based on the discussion above, we introduce Herd, a system designed for batch inference with dynamic pruning for LLM generation. Figure 4 and algorithm 1 illustrates the overview of Herd. The input data pool will be grouped with calculated CSPs from first FF block and then perform batch dynamic pruning algorithm to get the final outcome.

5 Downstream Task Evaluation

In this section, we perform multi-topic tasks to demonstrate the downstream performance improvements achieved by Herd in the scenario of batch inference.

5.1 Task Selection

Since Herd is applied during the generation stage, the downstream tasks we have chosen are mainly related to the evaluation of natural language generation. These tasks can be broadly divided into two categories:

In-context Learning Generation: These tasks involve generating answers or summaries based on a provided long context and are generally easier and more robust to the effects of pruning. The tasks in this category include Xsum (Narayan et al., 2018), CNN-Dailymail (See et al., 2017), CoQA (Reddy et al., 2019) and Natural QA(closebook version) (Kwiatkowski et al., 2019).

Instruction-based Generation: These tasks involve solving realistic problems such as code generation which are generally more difficult and less robust to pruning. The tasks we selected for this category include Gsm8k (Cobbe et al., 2021), HumanEval (Chen et al., 2021) and IFEval (Zeng et al., 2024).

Details are shown in Table 3 and 4.

5.2 Experiment Setup and Procedure

To show the performance improvement provided by Herd, we mixed the input request of the same task category into one request pool and apply different inference modes for text generation.

- Single-batch inference(S): The inference of this mode is under the condition of batch size = 1.
- Batch inference with random grouping(R): The inference of this mode is under the condition of batch size > 1. It does not apply Batch Grouping approach and randomly select the requests of the same task category to form a batch.
- Herd(H): The inference of this mode is under the condition of batch size > 1.

The detail of experiment setup is in Appendix D.

Model Type	Llama3-8B-Inst			Llama3-70B-Inst		
mode/$\mathcal{R}$	Gsm8k	HumanEval	IFEval	Gsm8k	HumanEval	IFEval
	EM	Pass@1	I-acc	EM	Pass@1	I-acc
baseline	78.60	61.30	75.24	89.80	78.13	84.29
S/30%	77.34	56.25	75.14	89.80	75.00	83.01
S/40%	77.34	56.25	73.12	89.80	74.22	84.22
S/50%	70.10	48.43	71.84	89.06	73.40	82.91
S/60%	66.41	43.75	70.96	87.50	71.87	81.92
S/70%	57.03	20.31	70.66	87.50	65.00	81.44
R/30%	75.39	50.78	73.22	89.06	73.81	81.34
R/40%	60.15	43.35	70.98	88.23	71.09	80.25
R/50%	53.13	39.84	68.79	87.50	70.70	78.62
R/60%	45.31	25.39	64.27	85.54	67.18	79.12
R/70%	40.62	16.40	60.20	84.37	61.32	75.77
H/30%	75.78	53.13	71.35	89.80	75.00	82.94
H/40%	65.63	47.65	70.87	88.23	72.65	82.65
H/50%	67.19	42.97	70.38	87.50	71.09	81.75
H/60%	57.81	28.90	69.90	86.72	67.96	80.24
H/70%	38.28	18.75	68.44	84.38	63.28	79.10

Table 2: The result of Instruction-based generation category tasks on Llama3-8B-Instruction and Llama3-70B-Instruction. The inference mode S, R and H is mentioned in section 5.2 and and $\mathcal{R}$ represents the pruning ratio. P-acc represents prompt level strict accuracy metric in IFEval evaluation. The pruning algorithm is CATS.

Task	Shot	Type
Xsum	3	Summarization
CNN-Dailymail	3	Summarization
CoQA	0	Question Answer
Natural QA	3	Question Answer

Table 3: In-context Learning Generation Tasks List

Task	Shot	Type
Gsm8k	8	Math
HumanEval	0	Code
IFEval	0	Instruction

Table 4: Instruction-based Generation Tasks List

5.3 Experiment Performance

Table 1 and table 2 shows the downstream performance comparison of single-batch inference S, multi-batch inference with random grouping R and Herd H. The results of our experiments demonstrate that with batch grouping, Herd achieves significantly better downstream performance than the baseline batch inference mode. In most tasks, when the sparsity ratio less than 50%, Herd's performance is comparable to that of single input inference, and even approaches the performance of the original LLM.

6 Efficiency Experiment

In this section we are going to introduce the benchmark experiment and shows the efficiency improvement of Herd. In section 6.1, we present the decoding latency using different dynamic pruning algorithms within Herd. In section 6.2, we conduct an end-to-end benchmark test and analyze the time consumption of each component in Herd.

6.1 Latency Benchmark

The application scenario for Herd is mainly considered in the on-device inference and offloading inference. For the on-device inference, model's parameters are completely loaded in the GPU device and Herd primarily aims to reduce the time required to loading parameters from GPU to tensor kernel as well as the computation time. For the offloading inference, the hardware resource(e.g. GPU memory) is limited thus the whole model can not be loaded once. Instead, GPU needs to frequently swap parameters with DRAMs and the loading of parameters is the main bottleneck. Nat-

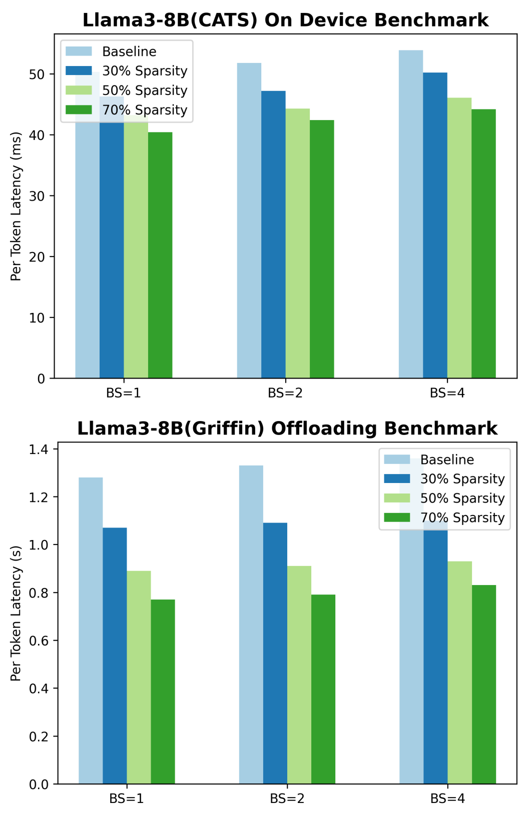


Figure 5: Latency benchmark result for on device scenario with CATS(top) and offloading scenario with Griffin(bottom)

urally, Herd allows less parameter loading of FF block during inference stage thus can bring higher benchmark performance.

Figure 5 presents the benchmark performance comparison between the on-device and offloading scenarios for the Meta- Llama-3-8B (AI@Meta, 2024) model. While higher sparsity naturally leads to reduced latency in both scenarios, Herd primarily contributes to throughput improvement. Through effective batch grouping, Herd enables multi-fold increases in the throughput of dynamic pruning inference. The experiment setup is in Appendix D.

6.2 End to End Benchmark

We conducted an end-to-end experiment using a fixed input data pool and benchmarked the total processing time across various scenarios. Figure 6 presents a detailed time breakdown of different processing stages. Notably, the overhead associated with generating CSPs and performing KMeans clustering is negligible compared to the time spent on LLM inference, particularly the decoding stage. The proposed Herd method enables batch inference, resulting in significantly improved through-

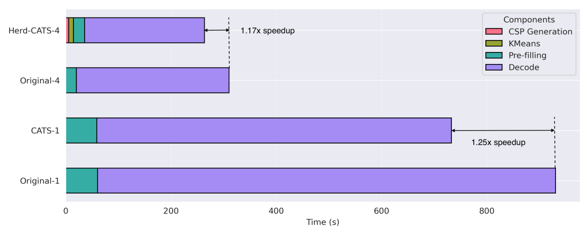


Figure 6: Time breakdown of different scenarios under a fixed request pool with 200 requests. Each request has the input length of 1024 and generation steps of 128. *Original* refers to no pruning, with -1 and -4 indicating batch sizes of 1 and 4, respectively. We use CATS as dynamic pruning algorithm. The processing time is decomposed into four components: CSP Generation, KMeans clustering, Pre-filling, and LLM Decoding. The sparsity ratio is 0.5. The experiment is performed on A6000 GPU.

put while maintaining the efficiency of the dynamic pruning algorithm.

7 Conclusion

In this paper, we proposed Herd, a novel dynamic pruning method for efficient LLM inference. By wisely grouping data with contextual sparsity pattern, Herd can batch data with similar contextual sparsity thus reduce the parameter competition and alleviate the degradation of downstream performance.

Limitation

In this section we will discuss the limitation of Herd design.

- **Lack of kernel optimization** Recently, many frameworks have achieved significantly better inference latency through compiler optimizations and specialized kernel designs. However, these implementations are primarily designed for dense models and such optimizations cannot be directly applied in dynamic pruning scenarios. For instance, while tools like Torch Compile (Ansel et al., 2024) provide substantial inference speedups, they lack the flexibility needed for diverse and dynamic pruning operations, making them unsuitable for this use case.

  Due to the lack of mature support from compilers and kernel designs, some of the previous dynamic pruning algorithm like CATS does not support the introduction of torch compile. Consequently, it is unable to compare benchmark performance directly with popular inference frameworks such as vLLM (Kwon et al., 2023) or SGLang (Zheng et al., 2024), which benefit from advanced optimizations. Unfortunately, Herd does not provide a solution to this challenge and continues to face the same issue. Addressing this limitation will require further kernel optimizations specifically tailored for dynamic pruning.

- **Challenges in Adapting to Online Service Applications** In our discussion of Herd, the scenario aligns more closely with offline inference rather than online inference. In an online service application, numerous variables must be taken into account. For example, the size of the request pool before performing CSP calculations and batch grouping is a critical factor. If the pool size is too large, the time required to fill the pool increases, leading to higher service latency. Conversely, if the pool size is too small, the benefits gained from Herd become uncertain. Therefore, applying our system design to real-world online service applications requires further exploration of system trade-offs.

- **Constraints of pruning format** Although most of the dynamic pruning algorithm is performed as we described in section 2, some new work propose new dynamic pruning format. For example, in Liu et al. (2024)'s work, TEAL introduce the activation-based dynamic pruning and perform it on both attention block and FF block. Our experiment shows that due to the mismatch of problem definition, Herd is not compatible with TEAL and results in similar outcome with random grouped batch.

Ethics Statement

Potential Risks

As an algorithm towards efficient inference, we introduced dynamic pruning to approximate the output of original model. Such approximation can potentially bring risks since it is not the original LLMs and it is unpredictable for its behavior.

Use of Scientific Artifacts

We utilize the evaluation platform such as lm-evaluation-harness and helm under the licence of MIT License and Apache License 2.0 respectively.

Model Size, Budget and Hyperparameters

In the Appendix D the details of model size and the amount of experiment data are listed. The budget of experiment is decided by the hardware usage.

Ai Assistants In Research Or Writing

We do use the Ai Assistants like ChatGPT for paper polishment.

References

AI@Meta. 2024. Llama 3 model card.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. 2024. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 929–947, New York, NY, USA. Association for Computing Machinery.

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*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen

Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

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. *Preprint*, arXiv:2110.14168.

Harry Dong, Beidi Chen, and Yuejie Chi. 2024. Prompt-prompted mixture of experts for efficient llm generation. *Preprint*, arXiv:2404.01365.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. 2017. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Preprint*, arXiv:1702.03118.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2024. A framework for few-shot language model evaluation.

Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *Preprint*, arXiv:1510.00149.

Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 1135–1143, Cambridge, MA, USA. MIT Press.

Dan Hendrycks and Kevin Gimpel. 2023. Gaussian error linear units (gelus). *Preprint*, arXiv:1606.08415.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *Preprint*, arXiv:1503.02531.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*.

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 ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Je-Yong Lee, Donghyun Lee, Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini. 2024. Cats: Contextually-aware thresholding for sparsity in large language models. *Preprint*, arXiv:2404.08763.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Alexander Cosgrove, Christopher D Manning, Christopher Re, Diana Acosta-Navas, Drew Arad Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue WANG, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Andrew Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2023. Holistic evaluation of language models. *Transactions on Machine Learning Research*. Featured Certification, Expert Certification.

James Liu, Pragaash Ponnusamy, Tianle Cai, Han Guo, Yoon Kim, and Ben Athiwaratkun. 2024. Training-free activation sparsity in large language models. *Preprint*, arXiv:2408.14690.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, and Beidi Chen. 2023. Deja vu: Contextual sparsity for efficient LLMs at inference time. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 22137–22176. PMLR.

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*, volume 36, pages 21702–21720. Curran Associates, Inc.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *Preprint*, arXiv:1808.08745.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer. *Preprint*, arXiv:1910.10683.

Raimundo Real and Juan M. Vargas. 1996. The Probabilistic Basis of Jaccard's Index of Similarity. *Systematic Biology*, 45(3):380–385.

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2019. Coqa: A conversational question answering challenge. *Preprint*, arXiv:1808.07042.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, and Maosong Sun. 2024. Prosparse: Introducing and enhancing intrinsic activation sparsity within large language models. *Preprint*, arXiv:2402.13516.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2023. Powerinfer: Fast large language model serving with a consumer-grade gpu. *Preprint*, arXiv:2312.12456.

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*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need. *Preprint*, arXiv:1706.03762.

BigScience Workshop, :, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, Dragomir Radev, Eduardo González Ponferrada, Efrat Levkovizh, Ethan Kim, Eyal Bar Natan, Francesco De Toni, Gérard Dupont, Germán Kruszewski, Giada Pistilli, Hady Elsahar, Hamza Benyamina, Hieu Tran, Ian Yu, Idris Abdulmumin, Isaac Johnson, Itziar Gonzalez-Dios, Javier de la Rosa, Jenny Chim, Jesse Dodge, Jian Zhu, Jonathan Chang, Jörg Frohberg, Joseph Tobing, Joydeep Bhattacharjee, Khalid Almubarak, Kimbo Chen, Kyle Lo, Leandro Von Werra, Leon Weber, Long Phan, Loubna Ben allal, Ludovic Tanguy, Manan Dey, Manuel Romero Muñoz, Maraim Masoud, María Grandury, Mario Šaško, Max Huang, Maximin Coavoux, Mayank Singh, Mike Tian-Jian Jiang, Minh Chien Vu, Mohammad A. Jauhar, Mustafa Ghaleb, Nishant Subramani, Nora Kassner, Nurulaqilla Khamis, Olivier Nguyen, Omar Espejel, Ona de Gibert, Paulo Villegas, Peter Henderson, Pierre Colombo, Priscilla Amuok, Quentin Lhoest, Rheza Harliman, Rishi Bommasani, Roberto Luis López, Rui Ribeiro, Salomey Osei, Sampo Pyysalo, Sebastian Nagel, Shamik Bose, Shamsuddeen Hassan Muhammad, Shanya Sharma, Shayne Longpre, Somaieh Nikpoor, Stanislav Silberberg, Suhas Pai, Sydney Zink, Tiago Timponi Torrent, Timo Schick, Tristan Thrush, Valentin Danchev, Vassilina Nikoulina, Veronika Laippala, Violette Lepercq, Vrinda Prabhu, Zaid Alyafeai, Zeerak Talat, Arun Raja, Benjamin Heinzerling, Chenglei Si, Davut Emre Taşar, Elizabeth Salesky, Sabrina J. Mielke, Wilson Y. Lee, Abheesht Sharma, Andrea Santilli, Antoine Chaffin, Arnaud Stiegler, Debajyoti Datta, Eliza Szczechla, Gunjan Chhablani, Han Wang, Harshit Pandey, Hendrik Strobelt, Jason Alan Fries, Jos Rozen, Leo Gao, Lintang Sutawika, M Saiful Bari, Maged S. Al-shaibani, Matteo Manica, Nihal Nayak, Ryan Teehan, Samuel Albanie, Sheng Shen, Srulik Ben-David, Stephen H. Bach, Taewoon Kim, Tali Bers, Thibault Fevry, Trishala Neeraj, Urmish Thakker, Vikas Raunak, Xiangru Tang, Zheng-Xin Yong, Zhiqing Sun, Shaked Brody, Yallow Uri, Hadar Tojarieh, Adam Roberts, Hyung Won Chung, Jaesung Tae, Jason Phang, Ofir Press, Conglong Li, Deepak Narayanan, Hatim Bourfoune, Jared Casper, Jeff Rasley, Max Ryabinin, Mayank Mishra, Minjia Zhang, Mohammad Shoeybi, Myriam Peyrounette, Nicolas Patry, Nouamane Tazi, Omar Sanseviero, Patrick von Platen, Pierre Cornette, Pierre François Lavallée, Rémi Lacroix, Samyam Rajbhandari, Sanchit Gandhi, Shaden Smith, Stéphane Requena, Suraj Patil, Tim Dettmers, Ahmed Baruwa, Amanpreet Singh, Anastasia Cheveleva, Anne-Laure Ligozat, Arjun Subramonian, Aurélie Névéol, Charles Lovering, Dan Garrette, Deepak Tunuguntla, Ehud Reiter, Ekaterina Taktasheva, Ekaterina Voloshina, Eli Bogdanov, Genta Indra Winata, Hailey Schoelkopf, Jan-Christoph Kalo, Jekaterina Novikova, Jessica Zosa Forde, Jordan Clive, Jungo Kasai, Ken Kawamura, Liam Hazan, Marine Carpuat, Miruna Clinciu, Najoung Kim, Newton Cheng, Oleg Serikov, Omer Antverg, Oskar van der Wal, Rui Zhang, Ruochen Zhang, Sebastian Gehrmann, Shachar Mirkin, Shani Pais, Tatiana Shavrina, Thomas Scialom, Tian Yun, Tomasz Limisiewicz, Verena Rieser, Vitaly Protasov, Vladislav Mikhailov, Yada Pruksachatkun, Yonatan Belinkov, Zachary Bamberger, Zdeněk Kasner, Alice Rueda, Amanda Pestana, Amir Feizpour, Ammar Khan, Amy Faranak, Ana Santos, Anthony Hevia, Antigona Unldreaj, Arash Aghagol, Arezoo Abdollahi, Aycha Tammour, Azadeh HajiHosseini, Bahareh

Behroozi, Benjamin Ajibade, Bharat Saxena, Carlos Muñoz Ferrandis, Daniel McDuff, Danish Contractor, David Lansky, Davis David, Douwe Kiela, Duong A. Nguyen, Edward Tan, Emi Baylor, Ezinwanne Ozoani, Fatima Mirza, Frankline Ononiwu, Habib Rezanejad, Hessie Jones, Indrani Bhattacharya, Irene Solaiman, Irina Sedenko, Isar Nejadgholi, Jesse Passmore, Josh Seltzer, Julio Bonis Sanz, Livia Dutra, Mairon Samagaio, Maraim Elbadri, Margot Mieskes, Marissa Gerchick, Martha Akinlolu, Michael McKenna, Mike Qiu, Muhammed Ghauri, Mykola Burynok, Nafis Abrar, Nazneen Rajani, Nour Elkott, Nour Fahmy, Olanrewaju Samuel, Ran An, Rasmus Kromann, Ryan Hao, Samira Alizadeh, Sarmad Shubber, Silas Wang, Sourav Roy, Sylvain Viguier, Thanh Le, Tobi Oyebade, Trieu Le, Yoyo Yang, Zach Nguyen, Abhinav Ramesh Kashyap, Alfredo Palasciano, Alison Callahan, Anima Shukla, Antonio Miranda-Escalada, Ayush Singh, Benjamin Beilharz, Bo Wang, Caio Brito, Chenxi Zhou, Chirag Jain, Chuxin Xu, Clémentine Fourrier, Daniel León Periñán, Daniel Molano, Dian Yu, Enrique Manjavacas, Fabio Barth, Florian Fuhrimann, Gabriel Altay, Giyaseddin Bayrak, Gully Burns, Helena U. Vrabec, Imane Bello, Ishani Dash, Jihyun Kang, John Giorgi, Jonas Golde, Jose David Posada, Karthik Rangasai Sivaraman, Lokesh Bulchandani, Lu Liu, Luisa Shinzato, Madeleine Hahn de Bykhovetz, Maiko Takeuchi, Marc Pàmies, Maria A Castillo, Marianna Nezhurina, Mario Sänger, Matthias Samwald, Michael Cullan, Michael Weinberg, Michiel De Wolf, Mina Mihaljcic, Minna Liu, Moritz Freidank, Myungsun Kang, Natasha Seelam, Nathan Dahlberg, Nicholas Michio Broad, Nikolaus Muellner, Pascale Fung, Patrick Haller, Ramya Chandrasekhar, Renata Eisenberg, Robert Martin, Rodrigo Canalli, Rosaline Su, Ruisi Su, Samuel Cahyawijaya, Samuele Garda, Shlok S Deshmukh, Shubhanshu Mishra, Sid Kiblawi, Simon Ott, Sinee Sang-aroonsiri, Srishti Kumar, Stefan Schweter, Sushil Bharati, Tanmay Laud, Théo Gigant, Tomoya Kainuma, Wojciech Kusa, Yanis Labrak, Yash Shailesh Bajaj, Yash Venkatraman, Yifan Xu, Yingxin Xu, Yu Xu, Zhe Tan, Zhongli Xie, Zifan Ye, Mathilde Bras, Younes Belkada, and Thomas Wolf. 2023. Bloom: A 176b-parameter open-access multilingual language model. *Preprint*, arXiv:2211.05100.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding. *Preprint*, arXiv:2309.07597.

Zhiyuan Zeng, Jiatong Yu, Tianyu Gao, Yu Meng, Tanya Goyal, and Danqi Chen. 2024. Evaluating large language models at evaluating instruction following. In *The Twelfth International Conference on Learning Representations*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona 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. 2022. Opt: Open pre-trained transformer language models. *Preprint*, arXiv:2205.01068.

Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. 2024. Relu^2 wins: Discovering efficient activation functions for sparse llms. *Preprint*, arXiv:2402.03804.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. Sglang: Efficient execution of structured language model programs. *Preprint*, arXiv:2312.07104.

A Related Work

A.1 Static Pruning

Several classic neural network compression techniques, including pruning (Han et al., 2015), distillation (Hinton et al., 2015) and quantization (Han et al., 2016), have been demonstrated to be useful across various domains of machine learning. Among these methods static pruning is proved to be an effective neural network compression method for efficient LLMs inference. In the LLMs field, there are several works to do the static pruning based on hessian estimation (Frantar and Alistarh, 2023), weights and activations (Sun et al., 2024), grouping parameters (Ma et al., 2023) and sliced parameters (Ashkboos et al., 2024).

A.2 Dynamic Pruning

Compared with static pruning, dynamic pruning does not eliminate parameters permanently. Instead, it predicts activate neurons based on the current input and prunes certain parameters. Deja Vu (Liu et al., 2023) was the first attempt in this field and achieved success with the OPT (Zhang et al., 2022) and Bloom (Workshop et al., 2023) models. In the LLMs field, some works (Song et al., 2023) successfully leverage the feature of ReLU activation function and achieve near-lossless results in the downstream tasks. However, most of the popular models such as GPT (Brown et al., 2020) and Llama (AI@Meta, 2024) do not adapt with ReLU activation function but with more complicated activation functions like GELU (Hendrycks and Gimpel, 2023) and SiLU (Elfwing et al., 2017). There are main two ways to apply dynamic pruning on these models. The first one is ReLUfication which introduces a new ReLU-based model through retraining original model (Zhang et al., 2024; Song et al., 2024). The second method is magnitude-based pruning, which assumes that calculations corresponding to low-magnitude normalized values in the output can be removed (Dong et al., 2024; Lee et al., 2024; Liu et al., 2024).

B Union Strategy for batch inference

Figure 7 illustrates the dramatic decline in sparsity with the union strategy, showing that it severely undermines the efficiency of dynamic pruning.

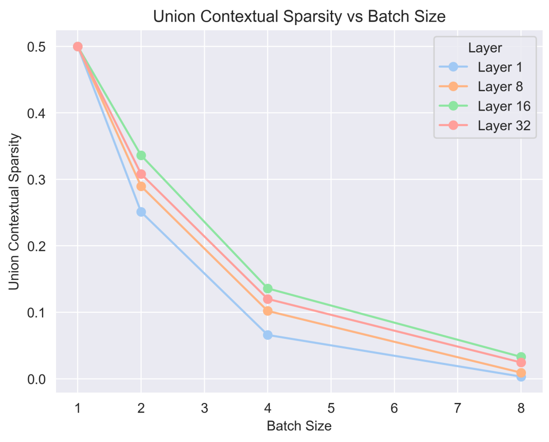


Figure 7: Union of contextual sparsity with batch inference. The experiment is conducted using the Meta-Llama-3-8B (AI@Meta, 2024) model during the inference phase on the ShareGPT dataset. For each data in the batch sparsity ratio = 50%

C Algorithm of Dynamic Pruning for Herd

Algorithm 2 CATS Algorithm

```
1: Input: threshold $t$, hidden state $x$, weights $W_{\text{gate}}$, $W_{\text{down}}$, and $W_{\text{up}}$
2: $v \leftarrow \text{SiLU}(xW_{\text{gate}})$
3: $v \leftarrow \|v\|_1$
4: Mask $\leftarrow$ 1 if $|v| \geq t$ else 0
5: $x_1 \leftarrow (xW_{\text{up}}[\text{Mask}] * v[\text{Mask}])$
6: $y \leftarrow x_1 W_{\text{down}}[\text{Mask}]$
```

Algorithm 3 Griffin Algorithm

```
1: Input: sparsity ratio $\mathcal{R}$, prompt hidden state $x_p$, hidden state $x$, weights $W_{\text{gate}}$, $W_{\text{down}}$, and $W_{\text{up}}$
2: Prefilling:
3:     $v \leftarrow \text{SiLU}(x_p W_{\text{gate}}) * x_p W_{up}$
4:     $v \leftarrow \|v\|_1$
5:     $t \leftarrow \text{Threshold}(|v|, \mathcal{R})$
6:     Mask $\leftarrow$ 1 if $|v| \geq t$ else 0
7: Decoding:
8:     $x_1 \leftarrow xW_{\text{up}}[\text{Mask}] * \text{SiLU}(xW_{\text{gate}}[\text{Mask}])$
9:     $y \leftarrow x_1 W_{\text{down}}[\text{Mask}]$
```

D Experiment Details

D.1 Downstream Task Evaluation Experiment

For the In-context learning generation category, we applied Meta-Llama-3-8B model with batch size

= 8 during the batch inference. For four different tasks, the number of input requests is 1k each. Additionally, we added 4k input requests from C4 dataset to increase pool's uncertainty. The cluster number of Kmeans in the Batch Grouping approach is set to 10. Since the tasks are relatively easy, we choose Griffin for downstream generation.

For the Instruction-based generation category, we applied Meta-Llama-3-8B-Instruct model and Meta-Llama-3-70B-Instruct model for downstream testing with batch size = 4 during the multi-batch inference. The number of input request is equally 128 for different tasks and we also added 128 input request of Natural QA dataset to increase uncertainty. The cluster number of Kmeans in the Batch Grouping approach is set to 8. Since the tasks are relatively hard, we choose CATS for downstream generation.

For the few-shot request, to make sure the diversity of request context, we used different shots for each request. To evaluate the downstream tasks performance, we use the existing github repo lm-evaluation-harness (Gao et al., 2024) and helm (Liang et al., 2023).

D.2 Efficiency Experiment

For the efficiency experiments, we evaluated the Llama3-8B model in both on-device and offloading scenarios for generation tasks. In the on-device scenario, we conducted the benchmark using CATS as the backend dynamic pruning algorithm, running on a single NVIDIA A6000 GPU. The batch sizes tested were 1, 2, and 4.

In the offloading scenario, we used Griffin as the backend dynamic pruning algorithm, but the experiment was performed on a single NVIDIA 2080Ti GPU. Similarly, the batch sizes tested were 1, 2, and 4. Figure 8 shows the on device benchmark of Griffin. We do not show the offloading benchmark of CATS since it is heavily influenced by cpu function.

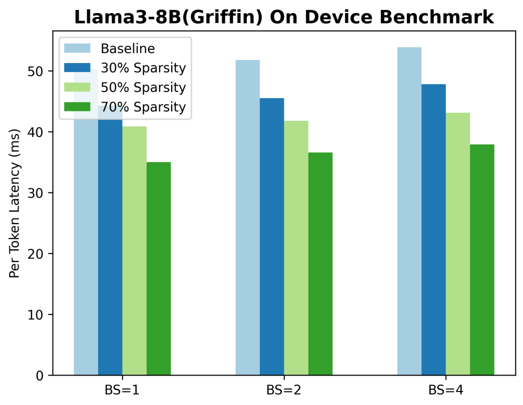


Figure 8: Griffin On device benchmark

E Ablation Study

We compare the clustering outcomes based on the CSPs of different layers. To evaluate the similarity between these outcomes, we utilize the Adjusted Rand Index (ARI) as the metric. ARI ranges from -1 to 1, where higher values indicate greater agreement between clustering results. Furthermore, recent research suggests that the first layer of large language models (LLMs) typically functions as an interpreter of low-level and lexical information. Therefore, we also compare the K-means clustering results with those obtained from a semantic representation. In this analysis, we use the bge-large-en(Xiao et al., 2023) model as a sentence embedding model to quantify the semantic meaning of the input.

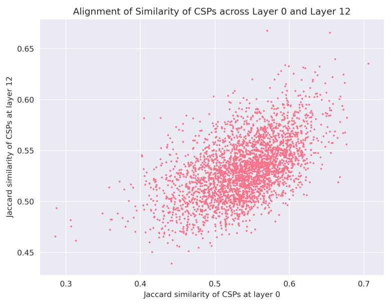
	L_8	L_{16}	L_{24}	L_{32}	Semantic
ARI	0.64	0.57	0.53	0.54	0.79

Table 5: ARI between Kmeans outcome with CSP of layer 1(L_1) and other Kmeans clustering outcome. L_i represents the Kmeans clustering with CSP of layer i and Sematic represents the clustering outcome with sentence embedding model. The experiment is performed on ShareGPT dataset with Meta-Llama-3-8B

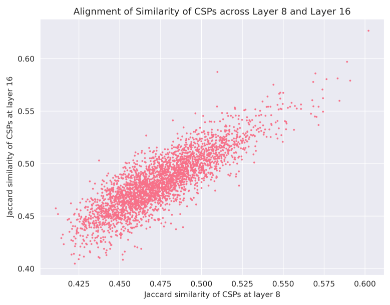
Table 5 presents the ARI metrics for different clustering outcomes. The results show that clustering outcomes based on CSPs from different layers consistently achieve ARI values above 0.5, indicating a strong agreement. This provides evidence of alignment in CSPs across layers. Notably, we observe that the ARI between the clustering outcomes based on CSPs and those based on semantic embeddings is remarkably high, suggesting that the similarity in contextual sparsity corresponds closely to how humans understand topics. This finding indicates that LLMs activate expert neurons aligned with human topic understanding, even in models that are not explicitly trained as mixtures of experts (MoE).

F Results for Consistency of CSP similarity across Layers

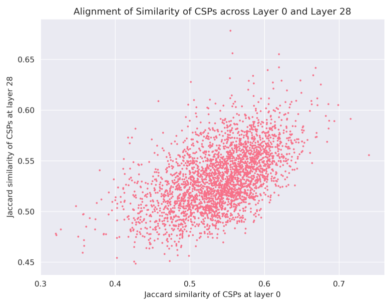
Here we provide more results to show the consistency of similarity in CSPs across layers:



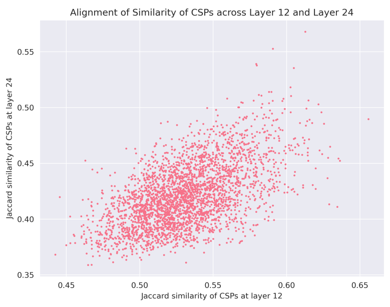
a: Subfigure 1



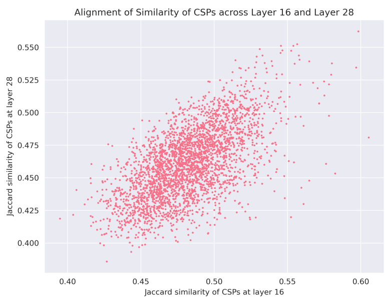
b: Subfigure 2



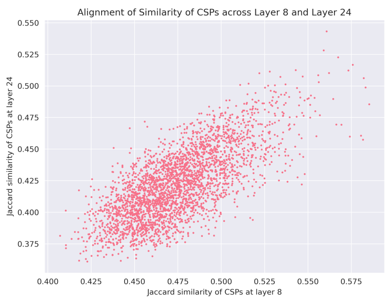
c: Subfigure 3



d: Subfigure 4



e: Subfigure 5



f: Subfigure 6

Figure 9: An overview of the results. Each subfigure highlights different aspects of the experiment.