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 Languege 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 these issues, 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 demonstrate that Herd not only preserves the performance level of the original model but also enhances inference efficiency, achieving better latency and throughput in batch inference scenario.

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. However, as models become more well-pretrained, the sparsity space and redundant parameters decrease and these static pruning methods are proved to significantly impair the language abilities of LLMs.

To mitigate the performance degradation in large language models (LLMs), a new method called dynamic pruning has been introduced. Unlike traditional static pruning, 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 techniques. However, a key limitation of existing methods is their applicability only in the scenario of batch size equals to one due to variability in contextual sparsity across different inputs.

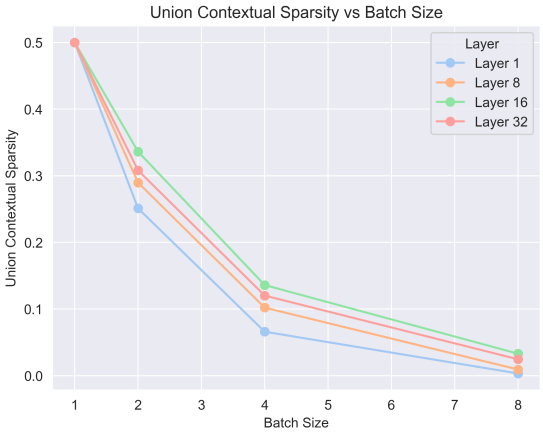


Figure 1: Union of contextual sparsity with batch inference. The experiment is conducted using the Meta-Llama-3-8B (AI@Meta, 2024) model during the inference phase on the ShareGPT dataset. For each data in the batch sparsity ratio = 50%

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 ensure that the requirements of each input are met during inference. However, this strategy significantly reduces

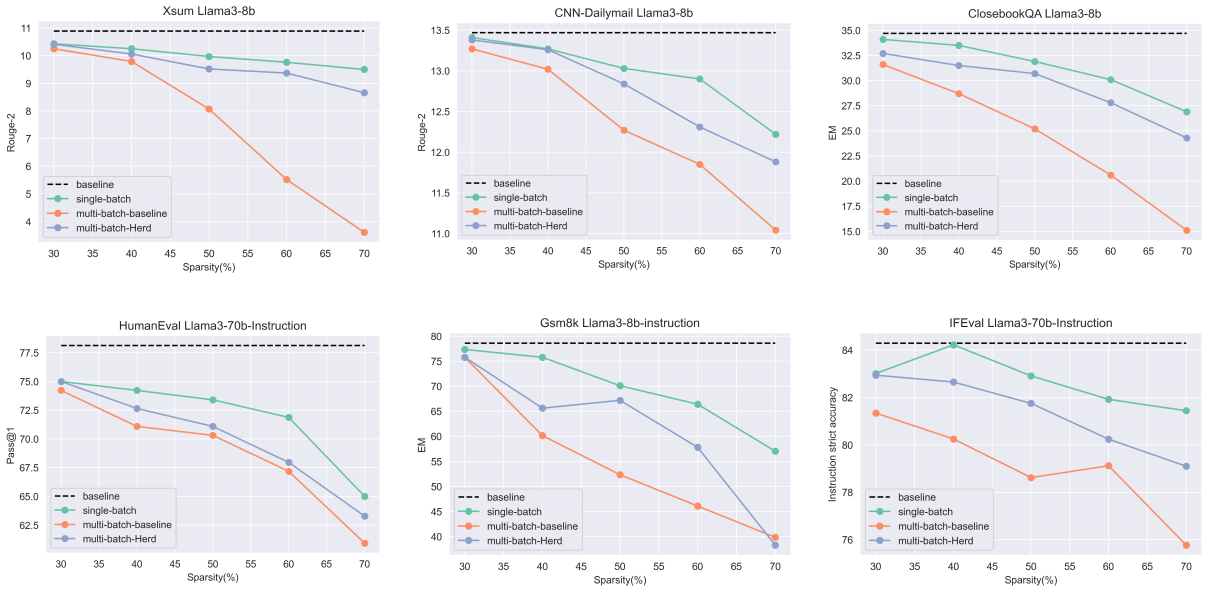


Figure 2: Downstream performance comparison of various inference strategies. The black dashed line indicates the performance of the model without pruning. The green line represents the performance of Dynamic Pruning in a single-instance-batch inference scenario. The orange line shows the performance of Dynamic Pruning in a batch inference scenario with randomly grouped batches. The blue line illustrates the performance of the Herd strategy, which applies flocking patterns to group data with higher contextual sparsity similarity prior to dynamic pruning.

the sparsity ratio, leading to limited speedup potential in dynamic pruning. Figure 1 illustrates the dramatic decline in sparsity with the union strategy, showing that it severely undermines the efficiency of 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 some accuracy by potentially missing essential parameters. While eviction provides a feasible solution for dynamic pruning in batch inference scenario, it risks performance degradation compared to single input dynamic pruning as the correct parameters may not always be available. Figure 2 highlights the performance degradation caused by this mismatch, showing that batch inference experiences significantly worse downstream performance especially at higher sparsity ratios compared to single input inference.

To address this problem, we introduced *Herd*, an algorithm designed for batch inference with dynamic pruning. Our key observation is that while there is diversity in contextual sparsity across different data samples, certain samples intuitively share more activated parameters due to inherent similarities in their sparsity patterns. We call this phenomenon **contextual sparsity similarity**. By batching these similar data together, it becomes more likely that all the samples in the batch will have access to their most relevant parameters, thereby mitigating the performance degradation caused by the eviction strategy. Specifically, there are two main challenges Herd solves:

1. How can we know which data pair share more activate parameters during dynamic inference: Inspired by previous research, Herd introduces the concept of a contextual sparsity pattern (CSP), a bit-wise vector that represents the parameters most likely to be activated in a Feed Forward block. The CSP can be obtained during the prefilling stage of LLM inference, previewing the most frequently activated parameters for subsequent dynamic pruning. The similarity between CSPs across different data samples indicates the contextual sparsity similarity within this Feed Forward block. Additionally, while CSP is inherently a property of individual Feed Forward blocks, we observe that the similarity of CSPs consists across layers in the Transformer architecture(Vaswani et al., 2023). This consistency across layers allows us to group data more efficiently, reducing computational cost and storage requirements. With CSPs, it is easy to cluster data with higher contextual sparsity similarity and the batches formed within these clusters tend

to share more parameters and reduce competition during eviction.

2. How to perform dynamic pruning after batch grouping: In the context of batch inference, experiments demonstrate that normalization at the batch size 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. Figure 2 shows that compared with baseline, Herd mitigates the degradation caused by parameter eviction during batch inference and provides chances to significant inference speedups in both on-device and offloading scenarios.

2 Related Work

In this section, we introduce related work in the field of pruning. In Section 3.3, we formally define the dynamic pruning problem using mathematical notation.

2.1 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). These methods can achieve better results than the classical magnitude-based static pruning methods(Han et al., 2015) on the downstream tasks but still drops heavily compared with original model.

2.2 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 only in this round. In the LLMs field, some works (Liu et al., 2023) (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 and Llama 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 value-based pruning, which assumes that calculations corresponding to low-magnitude normalized values in the output can be removed. Dong et al. (2024)'s recent work shows the consistency of low-magnitude normalized values between prompt tokens and generate tokens and utilize the sparsity information during prompt procedure to decide activate neurons for generation procedure. Lee et al. (2024)'s work introduces a training-free methods which decide the activation parameters based on the output of gate projection.

2.3 Problem Definition

Based on discussion above, we formally define the dynamic pruning problem using mathematical notation in this section.

In the architecture of a transformer, a single transformer layer comprises an Attention block and a Feed Forward 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 Feed Forward 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. 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 Feed Forward 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 prefilling 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 prefilling stage. Specifically, for a Feed Forward block, it is calculated as follows:

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

$$V_{CSP} = \arg\max\left(\left\| \frac{V_{\text{dot}}}{\|V_{\text{dot}}\|_{\text{dim}=-1}} \right\|_{\text{dim}=1}, \mathcal{R}\right)$$

$\mathbf{X} \in \mathbb{R}^{1\times Seq\times D_H}$ 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 $\mathbf{X}$.

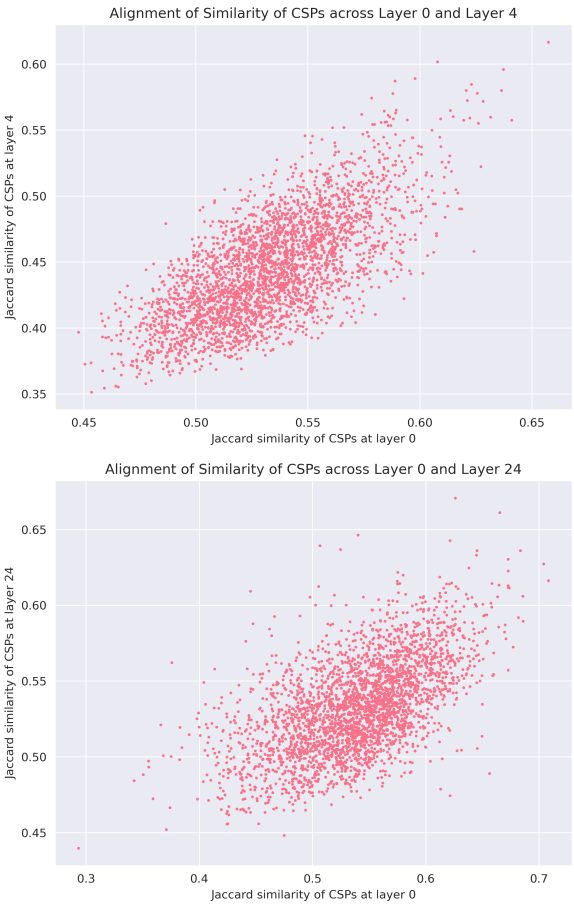


Figure 3: Alignment of Similarity in CSP Across Layers: Each point in the figure represents a pair of data samples chosen from the C4 dataset. The X and Y axes correspond to the Jaccard similarity of the CSP at layers i and j of Meta- Llama-3-8B model, respectively. For pairs of data that exhibit high Jaccard similarity at one layer, there tends to be a linear correlation in the similarity between these pairs across other layers. The sparsity ratio $\mathcal{R} = 0.5$.

3.2 Consistency of CSP similarity across Layers

However, CSP is calculated only for one Feed Forward 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 Feed Forward Block during decoding steps when applying dynamic pruning. 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

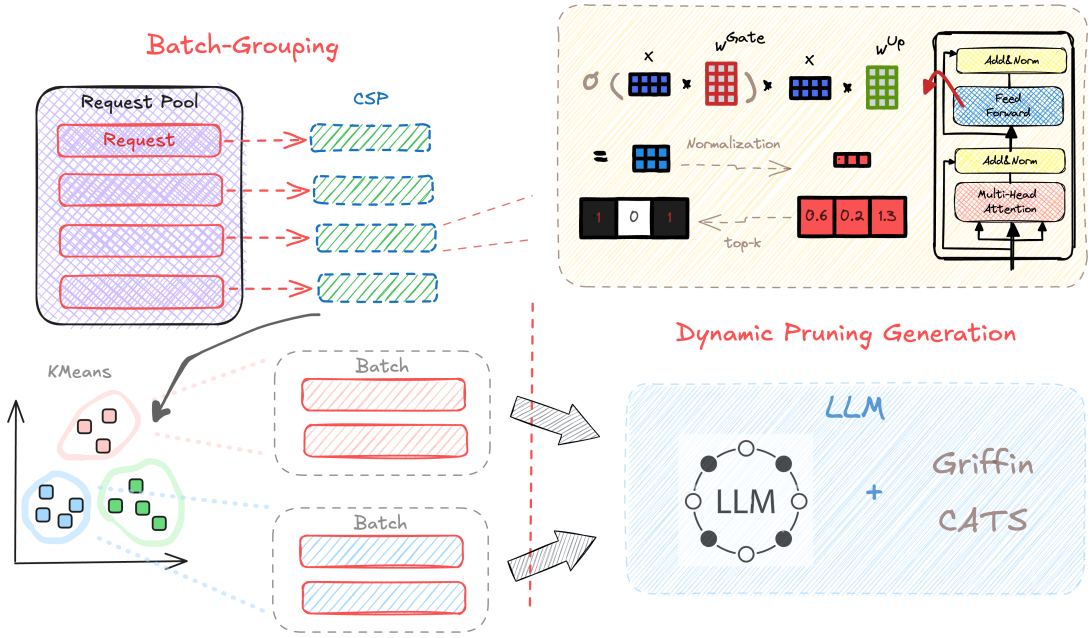


Figure 4: Overview of Herd: Herd consists of two primary stages. In the batch-grouping stage, the CSP is computed for each request, and data with higher similarity in CSPs are grouped into the same batch. In the dynamic pruning stage, Herd leverages either the Griffin or CATS algorithm to perform dynamic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

quantify the similarity between two patterns. Figure 3 demonstrates a clear linear relationship in the Jaccard similarity across different layers, epecially in the region with higher Jaccard similarity. 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.

The consistency of CSPs provides significant advantages for the Herd design. Firstly, it ensures that the contextual similarity among data remains consistent across blocks and layers. This means that when data with high similarity in a particular Feed Forward block are grouped together, the benefit of sharing similar neurons 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 based on previous discussion. 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 Feed Forward 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

For a given input pool, the CSP is calculated for each input at the first Feed Forward Block of transformer structure. With the collected CSPs we further apply the classic K-means algorithm to group data with higher CSP similarity. Based on previous discussion, inputs within the same cluster exhibit similar contextual sparsity 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.

4.2 Batch Dynamic Pruning

After the batch is grouped, we perform batch dynamic inference during the decoding stage. Herd modifies the algorithms from Griffin and CATS for dynamic pruning. Specifically, Griffin uses the CSP extracted from the prefilling 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 Feed Forward block as a routing mechanism to identify which parameters are most impor-

Inference 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%	10.25	72.41	13.27	87.52	70.30	80.49	31.60	38.97
R/40%	9.79	63.17	13.02	86.63	68.20	78.48	28.70	37.25
R/50%	8.07	56.28	12.27	85.21	66.40	76.21	25.20	34.37
R/60%	5.52	40.67	11.85	83.99	64.10	75.19	20.60	31.02
R/70%	3.61	29.29	11.04	81.66	60.90	72.28	15.10	25.74
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 4.2 and $\mathcal{R}$ represents the pruning ratio. Conv is the abbreviation of Convergence metric

tant. 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 B 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.

Comparatively, Griffin employs a fixed dynamic pruning pattern based on the CSP, which allows it to prune more parameters overall by pruning across $W_{\text{gate}}, W_{\text{up}}$, 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 despite pruning fewer parameters.

Algorithm 1 Herd Algorithm

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

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 Feed Forward Block and then perform

Model Type	Llama3-8B-Inst			Llama3-70B-Inst		
Inference mode/$\mathcal{R}$	Gsm8k	HumanEval	IFEval	Gsm8k	HumanEval	IFEval
	EM	Pass@1	P-acc	EM	Pass@1	P-acc
baseline	78.60	61.30	64.06	89.80	78.13	74.22
S/30%	77.34	56.25	64.06	89.80	75.00	74.18
S/40%	77.34	56.25	64.06	89.80	74.22	73.43
S/50%	70.10	48.43	66.40	89.06	73.40	72.86
S/60%	66.41	43.75	63.28	87.50	71.87	70.30
S/70%	57.03	20.31	59.37	87.50	65.00	72.86
R/30%	75.78	50.78	61.78	89.80	74.22	71.09
R/40%	60.15	42.97	60.94	88.23	71.09	65.62
R/50%	52.34	39.84	56.25	88.23	70.31	67.18
R/60%	46.09	25.78	53.12	85.93	67.18	67.96
R/70%	39.84	15.62	50.0	85.15	60.93	63.28
H/30%	75.78	53.13	60.94	89.80	75.00	73.43
H/40%	65.63	47.65	63.28	88.23	72.65	73.43
H/50%	67.19	42.97	58.59	87.50	71.09	71.85
H/60%	57.81	28.90	58.59	86.72	67.96	71.09
H/70%	38.28	18.75	56.25	84.38	63.28	71.88

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 4.2 and and $\mathcal{R}$ represents the pruning ratio. P-acc represents prompt level strict accuracy metric in IFEval evaluation.

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 primarily applied during the generation stage rather than the prompt stage, the downstream tasks we have chosen are mainly related to the evaluation of long text 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.

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

5.2 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 A.

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 multi-batch inference mode. In most tasks, when the sparsity ratio less than 50%, Herd's performance is comparable to that of single-batch inference, and even approaches the performance of the original LLM.

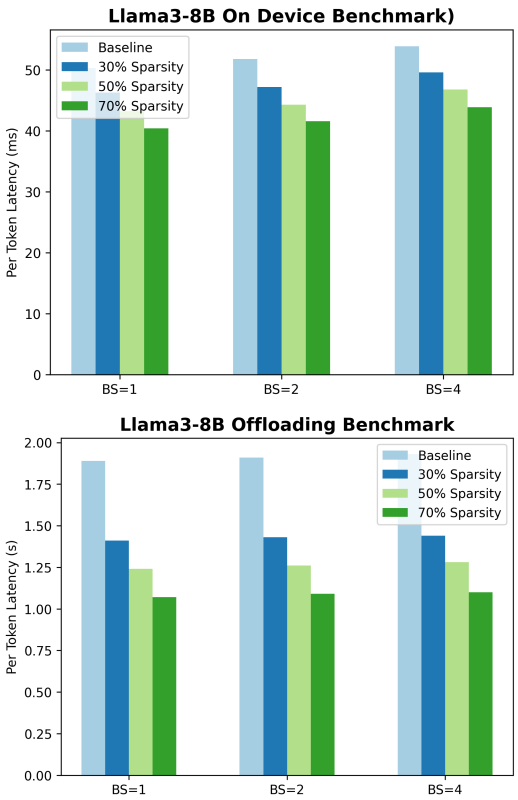


Figure 5: Benchmark result for on device scenario(top) and offloading scenario(bottom)

6 Efficiency Experiment

In this section we are going to introduce the benchmark experiment and shows the efficiency improvement of Herd. 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. Naturally, Herd allows less parameter loading of Feed Forward 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 model. While higher sparsity naturally leads to reduced latency in both scenarios with a more significant impact in the offloading case, 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 A.

One notable point is that our reported latency excludes the time required for calculating CSPs and performing KMeans clustering, as these are treated as precomputation steps. The average time cost for this precomputation is equivalent to the inference time of a single Feed Forward Block, which is minimal compared to the latency involved in subsequent multi-step decoding.

7 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).

8 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 spasity thus recude the parameter competition and alleviate the degration of downstream performance.

Limitation

The main limitation that Herd encounters is the speedup in application scenarios. 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 for dynamic pruning, our benchmarking compares the inference performance using a basic, non-optimized implementation. Consequently, we are unable to compare Herd directly with popular inference frameworks such as vLLM(Kwon et al., 2023) or SGLang(Zheng et al., 2024), which benefit from advanced optimizations.

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.c

Model Size, Budget and Hyperparameters

In the appendix A 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*.

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.

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.

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.

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*.

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 Experiment Details

A.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 multi-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).

A.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 also used CATS 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.

B Algorithm of Dynamic Pruning for Herd

Algorithm 2 CATS Algorithm

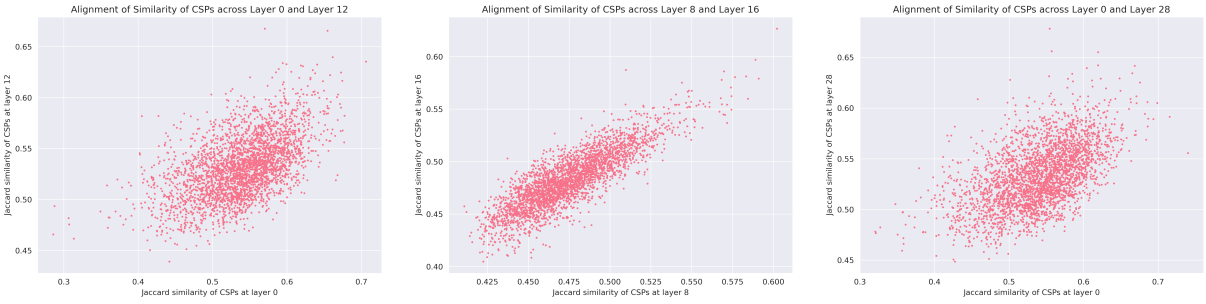
Input: threshold t, hidden state x, weights W_{gate}, W_{down}, and W_{up}
$v \leftarrow \text{SiLU}(xW_{\text{gate}})$
$v \leftarrow \|v\|_1$
Mask $\leftarrow 1$ if $|v| \geq t$ else 0
$x_1 \leftarrow (xW_{\text{up}}[\text{Mask}] * v[\text{Mask}])$
$y \leftarrow x_1 W_{\text{down}}[\text{Mask}]$

Algorithm 3 Griffin Algorithm

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

C 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 6: An overview of the results. Each subfigure highlights different aspects of the experiment.