FLRC: Fine-grained Low-Rank Compressor for Efficient LLM Inference

Yu-Chen Lu[1,2], Chong-Yan Chen[1],
Chi-Chih Chang[3], Yu-Fang Hu[1], Kai-Chiang Wu[1],
[1]National Yang Ming Chiao Tung University,
[2]Macronix International Co., Ltd., [3]Cornell University,
Correspondence: yuchen.cs11@nycu.edu.tw

Abstract

Although large language models (LLM) have achieved remarkable performance, their enormous parameter counts hinder deployment on resource-constrained hardware. Low-rank compression can reduce both memory usage and computational demand, but applying a uniform compression ratio across all layers often leads to significant performance degradation, and previous methods perform poorly during decoding. To address these issues, we propose the *Fine-grained Low-Rank Compressor (FLRC)*, which efficiently determines an optimal rank allocation for each layer, and incorporates progressive low-rank decoding to maintain text generation quality. Comprehensive experiments on diverse benchmarks demonstrate the superiority of *FLRC*, achieving up to a 17% improvement in ROUGE-L on summarization tasks compared to state-of-the-art low-rank compression methods, establishing a more robust and efficient framework to improve LLM inference.

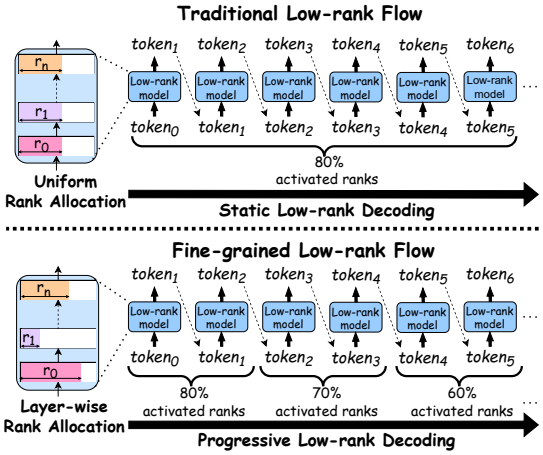

Figure 1: The differences between FLRC and traditional low-rank compression. As shown on the left side of the figure, we can determine the optimal number of ranks to preserve for each layer. On the right side, during the decoding stage, our approach gradually reduces the model's overall activated rank as more tokens are generated, unlike previous static methods, thereby decreasing the parameter usage and computational requirements while maintaining the quality of the generated output.

1 Introduction

In recent years, large language models (LLM) (Zhang et al., 2022; Touvron et al., 2023; Jiang et al., 2023; Liu et al., 2024a) have achieved remarkable progress in text understanding and generation, finding widespread applications in areas ranging from customer service to data analysis. However, the substantial parameter counts and high computational demands of these models pose significant challenges for deployment in resource-constrained environments such as mobile devices and edge servers.

To address these challenges, various model compression techniques have been proposed to reduce the computational and memory requirements of LLM while maintaining performance. Notable methods include model pruning (Ma et al., 2023; Akhauri et al., 2024) and quantization (Shao et al., 2023; Liu et al., 2024b). Among these, low-rank compression methods based on singular value decomposition (SVD) (Yuan et al., 2023; Wang et al., 2024) have shown particular promise in reducing both model size and computational cost.

Despite their potential, low-rank compression methods face several challenges that must be addressed. First, each layer (and even each projection) has its own tolerance for compression (c.f. Appendix A). Previous studies (Lin et al., 2024; Ji et al., 2024; Shao et al., 2024) have attempted to assign different, optimal compression rates to each component, but these methods are often time-consuming or insufficiently precise. Another significant issue is that prior work primarily evaluates compressed models on prefill-centric benchmarks, such as perplexity or common-sense reasoning tasks, which are limited to single-token genera-

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 14957–14967
November 4-9, 2025

tion. Our analysis reveals that even state-of-the-art SVD-based methods suffer from notable accuracy degradation on tasks that require multiple decoding iterations, such as text summarization.

As shown in Figure 1, we propose *Fine-grained Low-Rank Compressor (FLRC)* to overcome current limitations. Our framework introduces two key innovations. First, we develop an efficient, gradient-based rank allocation algorithm that is significantly faster and more accurate than existing methods. Second, we implement a dynamic low-rank compression paradigm that adjusts the rank allocation during each token generation, starting with a conservative compression rate and progressively increasing it to maintain high accuracy at the same overall compression ratio.

Experimental results on popular LLaMA model families further validate our approach. In our experiments, our rank allocation algorithm reduces search time by up to 49× compared to previous methods, and *FLRC* achieves up to a 17.35% higher ROUGE-L score on summarization benchmarks, setting a new standard for efficient and accurate model compression.

2 Related Works

Low-rank compression (Kaushal et al., 2023; Hsu et al., 2022) has emerged as an effective strategy for reducing both parameter counts and computational overhead in neural networks. ASVD (Yuan et al., 2023) mitigates the impact of outlier activations by scaling weight matrices based on activation distribution. Additionally, it introduces a rank allocation strategy to assign appropriate parameters ratio to each layer. However, this search method is extremely time-consuming. In contrast, our proposed rank search significantly reduces search time and, under high compression rates, finds rank allocation that deliver superior performance.

Another related work, SVD-LLM (Wang et al., 2024), introduces a truncation-aware data whitening method to better correlate singular values with compression errors, allowing the truncation of smaller singular values with minimal impact on error. However, despite these improvements, many low-rank compression methods still perform suboptimally during the decoding phase of LLM inference. To overcome this limitation, we propose progressive low-rank decoding, which maintains high text generation quality even under aggressive compression, thereby improving the practicality of compressed LLM in real-world generation tasks.

Algorithm 1 Layer-wise Rank Allocation

Input: Model M with layers L,
where each layer $l \in \mathcal{L}$
contains a set of projections P_l;
Calibration dataset $\mathcal{D}$;
Rank budget target R_{budget}.
Output: Rank allocation $\{r_{l,p}\}_{l\in\mathcal{L},p\in P_l}$.

$\{\mathbf{G}_{l,p}\} \leftarrow \text{ComputeGradient}(M, \mathcal{D})$
for each layer $l \in \mathcal{L}$ **do**
  for each projection $p \in P_l$ **do**
    Compute the importance:
    $\alpha_{l,p} = \sum_i \Big(\mathbf{G}_{l,p}[i] \times \mathbf{W}_{l,p}[i]\Big)^2$.
  end for
end for
Compute the total importance:
$S = \sum_{l\in\mathcal{L}} \sum_{p\in P_l} \alpha_{l,p}$.
for each layer $l \in \mathcal{L}$ **do**
  for each projection $p \in P_l$ **do**
    Allocate rank proportionally:
    $r_{l,p} = \text{round}\Big(\frac{\alpha_{l,p}}{S} \times R_{\text{budget}}\Big)$.
  end for
end for
return $\{r_{l,p} \mid l \in \mathcal{L}, p \in P_l\}$.

3 Proposed Method

Our proposed *Fine-grained Low-Rank Compressor (FLRC)* consists of two main components.

3.1 Fisher-based Layer-wise Rank Allocation

In LLM, different weight matrices—and even different projections within the same layer—exhibit varying capacities to tolerate compression. A uniform compression ratio across all layers can thus be suboptimal, as it may overcompress some components while undercompressing others. To address this issue, we propose the *Fisher-based Layer-wise Rank Allocation* algorithm, which computes an optimal rank allocation for each projection, preserving crucial projection ranks while effectively reducing overall model size. An overview of our algorithm is provided in Algorithm 1.

Our method begins by passing a calibration dataset $\mathcal{D}$ through the model M and computing the gradients via backward propagation. Let L denote the set of all layers in the model, and for each layer $l \in L$, let P_l be the set of projections in that layer. For each layer $l \in L$ and each projection $p \in P_l$, we denote the corresponding weight vector

as $\mathbf{W}_{l,p}$ and its gradient as $\mathbf{G}_{l,p}$. We then calculate a fisher-based (Abdelfattah et al., 2021) importance value $\alpha_{l,p}$, defined as:

$$\alpha_{l,p} = \sum_i \Big(\mathbf{G}_{l,p}[i] \times \mathbf{W}_{l,p}[i]\Big)^2, \quad (1)$$

which measures the sensitivity of each projection by incorporating both the gradient and the weight values. Higher $\alpha_{l,p}$ values indicate that the projection is more critical and should be compressed less aggressively (or potentially left uncompressed), whereas lower values suggest that the projection can tolerate more aggressive compression. For further details on Equation 1, please refer to Appendix B.

After computing the importance values for all projections, we sum them to obtain the total importance score S. We then allocate the rank for each projection proportionally to its importance by setting:

$$r_{l,p} = \text{round}\Big(\frac{\alpha_{l,p}}{S} \times R_{\text{budget}}\Big), \quad (2)$$

where R_{budget} is the overall rank budget target, adjustable based on the desired level of overall parameter compression. This yields a layer-wise rank allocation $\{r_{l,p} \mid l \in \mathcal{L}, p \in P_l\}$ that specifies the number of ranks retained for each projection in each layer, reflecting their relative importance.

This adaptive strategy ensures that the available compression budget is efficiently distributed across the model, focusing more resources on the most impactful components. As a result, our rank allocation method achieves a better balance between compression and performance compared to methods that apply a uniform compression ratio across all layers.

3.2 Progressive Low-rank Decoding

In text generation tasks, earlier tokens play a more significant role in shaping the overall coherence and quality of the output compared to later tokens (c.f. Appendix D). Thus, we propose *Progressive Low-rank Decoding*, a dynamic compression strategy that gradually reduces the model's overall activated ranks during decoding. As shown in Figure 1, our method progressively decreases the rank as more tokens are generated, increasing the overall compression rate while preserving strong performance in generation phase.

To adapt the rank allocation during decoding, we design a scheduler that determines the overall rank budget R_{budget} to be used for each token. Our scheduler leverages a calibration dataset to identify the optimal schedule based on different target compression levels. Let $R_{\text{budget}}(t)$ denote the rank budget for token t as determined by the scheduler. Note that $R_{\text{budget}}(t)$ is non-increasing, meaning that while consecutive tokens may share the same budget, the budget for token $t+1$ will never exceed that for token t.

Substituting $R_{\text{budget}}(t)$ for R_{budget} in Equation 2 yields the token-specific rank configuration:

$$r_{l,p}(t) = \text{round}\Big(\frac{\alpha_{l,p}}{S} \times R_{\text{budget}}(t)\Big). \quad (3)$$

This yields the configuration $\{r_{l,p}(t) \mid l \in \mathcal{L}, p \in P_l\}$ for the current token.

This scheduler-based approach dynamically adjusts the rank budget during decoding: early tokens benefit from a larger parameter set, while later tokens are generated with a reduced rank configuration. For supplementary details on our method, please refer to Appendix C.

4 Experiments

4.1 Experiments Setup

For the decoding stage evaluation, we conduct experiments on two summarization benchmarks: DialogSum (Chen et al., 2021) and CNN/DM (Hermann et al., 2015). In addition, to assess performance during the prefilling stage, we measure the perplexity on the Wikitext2 (Merity et al., 2016) dataset and evaluate zero-shot accuracy across seven common tasks provided in the LM-Evaluation-Harness (Gao et al., 2021). For experimental details, please refer to the Appendix F.

4.2 Evaluation on Generation Tasks

As shown in Table 1, our experiments on Llama-3-8B-Instruct (Dubey et al., 2024) reveal that previous low-rank compression methods struggle with generation tasks. In contrast, our approach, which incorporates progressive low-rank decoding, consistently maintains strong performance across various compression ratios. Here, the compression rate represents the overall percentage of parameter usage saved during the entire generation stage. Notably, under a 20% compression rate[1], evaluations on the DialogSum benchmark indicate that while competing methods yield ROUGE-L scores

[1]We define the compression rate as the average percentage reduction in model parameters per token, computed over both the prefilling and decoding stages.

Comp. Rate	Method	Llama-3-8B-Instruct				Llama-2-7B-Chat			
		DialogSum		CNN/DM		DialogSum		CNN/DM	
		ROUGE-L ↑	BERTScore ↑	ROUGE-L ↑	BERTScore ↑	ROUGE-L ↑	BERTScore ↑	ROUGE-L ↑	BERTScore ↑
-	Baseline	24.72	86.79	24.34	86.51	24.56	87.75	24.82	87.23
20%	ASVD	0.10	80.07	0.54	77.09	15.44	80.45	7.94	78.75
	SVD-LLM	0.24	78.12	6.29	76.46	13.62	83.07	19.71	**84.86**
	FLRC	**17.35**	**86.00**	**17.72**	**84.18**	**17.22**	**85.29**	**19.84**	84.83
30%	ASVD	0.53	72.45	0.07	71.81	6.47	80.34	3.44	75.66
	SVD-LLM	0.41	72.06	3.98	74.28	2.34	75.62	15.56	82.20
	FLRC	**8.09**	**81.92**	**10.83**	**79.92**	**14.91**	**83.62**	**17.28**	**83.91**

Table 1: Generative performance comparison (ROUGE-L and BertScore are expressed as percentages).

Model	Comp. Rate	Method	Perplexity ↓	Zero-shot Task Accuracy (%) ↑							
			Wiki2	ARC-e	ARC-c	Hella	OBQA	Wino	MathQA	PIQA	Avg.
Llama-3-8B	-	Baseline	6.14	80.13	50.51	60.17	34.80	72.61	40.50	79.71	59.78
	20%	ASVD	3206.80	30.81	19.54	27.06	13.80	52.41	21.04	56.37	31.58
		SVD-LLM	14.72	**55.64**	27.30	37.22	21.60	60.54	24.39	64.69	41.63
		FLRC	**12.53**	54.42	**28.58**	**38.95**	**23.80**	**68.27**	**25.03**	**66.54**	**43.66**
	30%	ASVD	28566.03	25.58	**22.78**	25.84	12.40	51.22	18.26	52.29	29.77
		SVD-LLM	33.13	**40.07**	20.99	30.30	16.80	55.33	**22.75**	57.94	34.88
		FLRC	**25.46**	38.34	20.39	**30.84**	**19.00**	**59.51**	21.68	**60.55**	**35.76**

Table 2: Perplexity and zero-shot accuracy of low-rank compression methods.

of less than 1%, our method achieves an impressive 17.35%.

Although earlier low-rank compression techniques have shown relatively better performance on Llama-2-7B-Chat, our method still delivers significantly higher ROUGE-L and BertScore metrics at high compression rates across both benchmarks. An ablation study of our proposed approach is presented in Appendix E. We also evaluated our method on different model sizes to demonstrate its generalization; see Appendix H for details.

4.3 Evaluation on Understanding Tasks

In addition to generation tasks, we evaluate our approach on perplexity and zero-shot accuracy using Llama-3-8B, as shown in Table 2. On the Wikitext2 dataset, our method achieves significantly lower perplexity compared to other low-rank compression techniques. Moreover, the average zero-shot accuracy across various compression ratios consistently outperforms that of previous methods. These results indicate that our proposed layer-wise rank allocation effectively mitigates the performance loss typically associated with model compression, ensuring robust language understanding even under aggressive parameter reduction.

4.4 Rank Allocation Search Time

We compare our proposed rank allocation search with the ASVD approach. The ASVD method, being perplexity-based, requires substantially more time for the search process compared to our approach. On an A100 GPU, the ASVD method takes approximately 147 minutes to complete the search, whereas our method requires only 3 minutes, representing a 49-fold improvement in speed. This significant reduction in search time demonstrates that our approach can quickly and efficiently determine an optimal rank configuration for the model, thereby facilitating faster deployment. For additional performance comparison experiments, please refer to Appendix G.

5 Conclusion

In this study, we propose the *Fine-grained Low-Rank Compressor (FLRC)* to rapidly determine the optimal compression ratio for each layer, thereby mitigating the performance degradation that arises from applying a uniform compression rate across all layers. Additionally, we introduce progressive low-rank decoding to address the poor performance of existing low-rank compression methods during the generation phase. Experimental results demonstrate that, under the same parameter utilization, our approach outperforms other methods on both generation and understanding tasks, indicating a significant performance improvement in low-rank compression.

Limitation

In this study, we rely on a calibration dataset to perform layer-wise rank allocation and design the scheduler for *FLRC*. However, the model's performance on different benchmarks may vary depending on the choice of calibration dataset, which can lead to discrepancies. To ensure fairness, we use the same calibration dataset for all methods in our experiments.

Additionally, our experimental results show that dynamically specifying the number of model parameters used per token can greatly enhance LLM inference efficiency. Nevertheless, optimizing the scheduler for dynamic rank allocation remains a crucial challenge, as it may introduce additional overhead. Consequently, our future work will focus on engineering optimizations and kernel design, specifically reducing the overhead associated with dynamic rank allocation, to further improve the overall efficiency of our approach.

Acknowledgment

We would like to express our gratitude to all organizations that provided the computational resources necessary to complete the experiments in this study. Additionally, we acknowledge the use of ChatGPT for assisting with paraphrasing and polishing, and not for any other illegal purposes.

References

Mohamed S Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D Lane. 2021. Zero-cost proxies for lightweight nas. *arXiv preprint arXiv:2101.08134*.

Yash Akhauri, Ahmed F AbouElhamayed, Jordan Dotzel, Zhiru Zhang, Alexander M Rush, Safeen Huda, and Mohamed S Abdelfattah. 2024. Shadowllm: Predictor-based contextual sparsity for large language models. *arXiv preprint arXiv:2406.16635*.

Aida Amini, Saadia Gabriel, Peter Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. *Preprint*, arXiv:1905.13319.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Yulong Chen, Yang Liu, Liang Chen, and Yue Zhang. 2021. Dialogsum: A real-life scenario dialogue summarization dataset. *arXiv preprint arXiv:2105.06762*.

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. 2021. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*, 10:8–9.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28.

Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. 2022. Language model compression with weighted low-rank factorization. *arXiv preprint arXiv:2207.00112*.

Yixin Ji, Yang Xiang, Juntao Li, Wei Chen, Zhongyi Liu, Kehai Chen, and Min Zhang. 2024. Feature-based low-rank compression of large language models via bayesian optimization. *arXiv preprint arXiv:2405.10616*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Ayush Kaushal, Tejas Vaidhya, and Irina Rish. 2023. Lord: Low rank decomposition of monolingual code llms for one-shot compression. *arXiv preprint arXiv:2309.14021*.

Quentin Lhoest, Albert Villanova Del Moral, Yacine Jernite, Abhishek Thakur, Patrick Von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, et al. 2021. Datasets: A community library for natural language processing. *arXiv preprint arXiv:2109.02846*.

Chi-Heng Lin, Shangqian Gao, James Seale Smith, Abhishek Patel, Shikhar Tuli, Yilin Shen, Hongxia Jin, and Yen-Chang Hsu. 2024. Modegpt: Modular decomposition for large language model compression. *arXiv preprint arXiv:2408.09632*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024a.

Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2024b. Spinquant–llm quantization with learned rotations. *arXiv preprint arXiv:2405.16406*.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

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

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

Hang Shao, Bei Liu, and Yanmin Qian. 2024. One-shot sensitivity-aware mixed sparsity pruning for large language models. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 11296–11300. IEEE.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. 2024. Svd-llm: Truncation-aware singular value decomposition for large language model compression. *arXiv preprint arXiv:2403.07378*.

Thomas Wolf. 2020. Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Zhihang Yuan, Yuzhang Shang, Yue Song, Qiang Wu, Yan Yan, and Guangyu Sun. 2023. Asvd: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv:2312.05821*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

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

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

A Importance Score of Different Layers

Different layers within a model often exhibit varying degrees of "compressibility", implying that uniform compression ratio can lead to suboptimal results. We can calculate the importance score of each component in the model based on our proposed method. As shown in Figure 2, the importance score of the projection in each layer varies significantly. Identifying which layers can tolerate more aggressive compression and which layers require a more careful approach is crucial to maximizing efficiency while minimizing performance degradation.

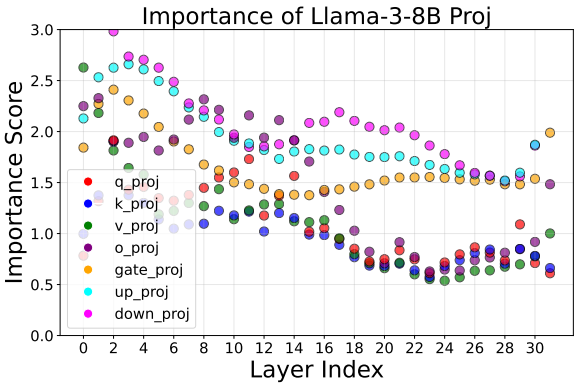

Figure 2: The importance score of various projections in Llama-3-8B across different layer indices. Each point represents a projection's score; higher scores (e.g., "down_proj") indicate that less compression should be applied, while lower scores allow for more aggressive compression.

B Sensitivity Metrics for Each Projection

We use a small calibration dataset and perform back propagation to compute the gradient for each projection. We observed that parameters with larger gradients tend to be more sensitive, and that larger weight values typically indicate higher importance. Thus, we multiply the weight and its corresponding

Comp. Rate	Method	DialogSum ROUGE-L ↑
-	Baseline	24.72
10%	Eq. 4	0.44
	Eq. 5	15.74
	Eq. 1	**20.23**
20%	Eq. 4	0.07
	Eq. 5	2.16
	Eq. 1	**17.35**

Table 3: Generative performance comparison of different sensitivity metrics on Llama-3-8B-Instruct (ROUGE-L is expressed as percentages).

gradient and then square the product to derive an importance value.

In addition to Equation 1, we evaluated two alternative metrics. First, we considered only the weight magnitudes:

$$\alpha_{l,p} = \sum_i \Big(\mathbf{W}_{l,p}[i]\Big)^2, \tag{4}$$

and second, we considered only the gradient values:

$$\alpha_{l,p} = \sum_i \Big(\mathbf{G}_{l,p}[i]\Big)^2. \tag{5}$$

Using each metric, we computed the importance of every projection and performed rank allocation accordingly. Table 3 presents generative performance comparison on Llama-3-8B-Instruct. It is clear that the metric combining both gradient and weight magnitudes is the most accurate. Consequently, we adopt Equation 1 as our chosen method for estimating the importance of projection.

C Supplementary Details on Progressive Low-Rank Decoding

Increasing the compression rate gradually during the generation phase is highly compatible with low-rank compression. After decomposing each projection's parameter matrix into two smaller matrices using singular value decomposition, the channels in these matrices are automatically ordered by importance. In other words, rows or columns at lower indices contain the most critical information, while those at higher indices can be safely truncated. This allows us to dynamically decide, at each token generation step, how many of the top k rows or columns to retain, where a smaller k corresponds to a higher compression rate. This inherent property makes our approach ideally suited for dynamic rank allocation, leading to an efficient implementation of progressive low-rank decoding, as we only need to decrement k during token generation.

Method	DialogSum ROUGE-L ↑
Static rank decoding	14.71
Increased rank decoding	8.59
Decreased rank decoding	**19.87**

Table 4: Comparison of different dynamic rank decoding methods on Llama-3-8B-Instruct (ROUGE-L is expressed as percentages).

Although dynamic rank adjustment introduces some overhead, when the savings in computation and data transfer are substantial, the overhead of dynamically changing the rank becomes negligible. Moreover, users can also evaluate what level of performance degradation is acceptable in exchange for the corresponding acceleration, as this will vary depending on the specific use case.

In our work, the term "schedule" refers to the points during the generation process at which the LLM switches to a higher compression rate. This means we can generate many schedule candidates, each corresponding to an overall compression rate (i.e., the average compression rate used for every token), which we denote as the overall rank budget (R_{budget}). We then use a calibration dataset to evaluate the performance of each schedule (using metrics like BERTScore), ultimately selecting the schedule that best meets our desired overall rank budget R_{budget} while achieving optimal performance for running our FLRC.

D Progressive Low-rank Decoding Forms

In this study, we propose dynamically adjusting the number of ranks used for each generated token, and we compare three approaches for doing so. The first approach, *Static Rank Decoding*, applies a fixed rank for every token. The second, *Increased Rank Decoding*, uses fewer ranks for early tokens and more for later ones. The third, *Decreased Rank Decoding*, assigns more ranks to early tokens and fewer to later tokens. In Table 4, we compare these methods on the DialogSum summarization task, ensuring that each approach uses the same average number of parameters. Our results demonstrate that *Decreased Rank Decoding* achieves superior performance, which is why we adopt it as our method for *Progressive Low-Rank Decoding*.

Ablation Settings			DialogSum ROUGE-L ↑
SVD-LLM	FLRA	PLRD	
✓	✗	✗	0.24
✓	✓	✗	13.28
✓	✓	✓	**17.35**

Table 5: Ablation study on generative performance (ROUGE-L is expressed as percentages). "FLRA" denotes Fisher-based Layer-wise Rank Allocation, and "PLRD" denotes Progressive Low-Rank Decoding.

E Ablation Study

Our proposed Fine-grained Low-Rank Compressor consists of two key components: Fisher-based Layer-wise Rank Allocation (FLRA) and Progressive Low-Rank Decoding (PLRD). To quantify the impact of each component, we conducted an ablation study on Llama-3-8B-Instruct with a 20% compression rate, measuring generative performance. As shown in Table 5, SVD-LLM alone delivers poor results. In contrast, applying either our FLRA or PLRD individually yields substantial gains in generation quality. These findings demonstrate that both components of our method effectively enhance the performance of low-rank compressed models.

F Experimental Details

Our used datasets and base models were sourced from the HuggingFace (Lhoest et al., 2021) and Transformers (Wolf, 2020) libraries, and all usage complied with the respective terms and conditions. For evaluating zero-shot accuracy, we employed seven common tasks: ARC-Easy, ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), WinoGrande (Sakaguchi et al., 2019), MathQA (Amini et al., 2019) and PIQA (Bisk et al., 2020). For summarization tasks, we used ROUGE-L (Lin, 2004) and BertScore (Zhang et al., 2019) as evaluation metrics.

For the *FLRC* layer-wise rank allocation, we sampled 256 sequences (each with a length of 2048) from the Wikitext2 training set as our calibration dataset, while the scheduler's calibration dataset was drawn from 500 samples from the DialogSum training set. For perplexity evaluation, the input sequence length was set to 2048. The compression rate is computed by first establishing a baseline based on the number of parameters in the `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj` matrices of the LLaMA model, and then determining the percentage of parameters omitted during each inference.

Our experimental pipeline follows the SVD-LLM procedure. First, the model weights are decomposed using SVD-LLM's truncation-aware data whitening method, after which we apply our proposed layer-wise rank allocation and progressive low-rank decoding modules. Notably, since our compression strategy is orthogonal to PEFT fine-tuning, we deliberately omit the weight updating steps typically included in the SVD-LLM framework. This design choice was made to ensure a fair comparison with SVD-LLM.

G Rank Allocation Method Comparison

In order to compare our rank allocation method with ASVD's, we first whiten the model weights and then apply different rank allocation strategies. We evaluate the resulting models on Wikitext2 by measuring perplexity. As shown in Table 6, under the same compression rate, our method achieves lower perplexity, demonstrating that our approach not only speeds up the search process but also finds a more optimal rank allocation for the compressed model.

Comp. Rate	Rank Allocation Method	Wiki2 Perplexity ↓
20%	ASVD	22.69
	FLRC	**12.53**
30%	ASVD	128.96
	FLRC	**25.46**

Table 6: Rank allocation method comparison on Llama-3-8B.

The rank allocation method we employ is both fast (as detailed in Section 4.4) and yields superior results. Unlike techniques that rely on iterative updates (such as Bayesian Optimization (Ji et al., 2024)) or memory-intensive and slow Hessian-based methods (Shao et al., 2024), our approach avoids these drawbacks.

Previous works (Lin et al., 2024; Ji et al., 2024; Shao et al., 2024) have proposed estimating the importance of various model components; however, these approaches are often inefficient or inaccurate and unsuitable for our method. MoDeGPT (Lin et al., 2024) evaluates the importance of different blocks (or layers) using block influence, which requires the input and output dimensions to be the same. In contrast, our rank allocation method is

Comp. Rate	Rank Allocation Method	Dialogsum ROUGE-L ↑
-	Baseline	24.56
20%	MoDeGPT	3.91
	PrunerGPT	16.28
	FLRC	**17.22**
30%	MoDeGPT	2.43
	PrunerGPT	10.81
	FLRC	**14.91**

Table 7: Generative performance comparison of different allocation methods on Llama-2-7B-Chat (ROUGE-L is expressed as percentages).

Method	Comp. Rate	DialogSum ROUGE-L ↑ Llama3.2-3B	Llama-2-13B
Baseline	-	12.84	17.23
SVD-LLM	10%	7.09	16.94
FLRC		**13.98**	**17.99**
SVD-LLM	20%	3.55	0.18
FLRC		**9.94**	**17.43**

Table 8: Generative performance comparison on 3B and 13B models (ROUGE-L is expressed as percentages).

more fine-grained and can evaluate the importance of each projection within every block, making it better suited for our progressive low-rank decoding. Bolaco (Ji et al., 2024) uses Bayesian optimization for rank allocation, which requires multiple iterations to converge. Our approach, on the other hand, only needs a single iteration, making it significantly more efficient. PrunerGPT (Shao et al., 2024) uses a Hessian-based approach to identify the importance of each component, which consumes substantial memory and computation time. As a result, these methods are less efficient than our proposed method.

We integrated the allocation methods from prior works with our progressive low-rank decoding and conducted a generative performance comparison. As shown in Table 7, our fisher-based rank allocation outperforms the other methods and remains highly efficient.

H Evaluation on Models of Different Sizes

We evaluated our method on 3B and 13B models to demonstrate its generalization capability. Table 8 clearly shows that FLRC continues to outperform SVD-LLM by a significant margin. The results demonstrate that, although SVD-LLM experiences a significant performance drop, FLRC substantially mitigates the performance degradation at the same compression rate.

We also conducted a zero-shot evaluation on the Llama-2-13B model. As shown in Table 9, our method consistently outperforms prior approaches across diverse tasks at the same compression rate, highlighting the superiority of FLRC efficacy in preserving models performance.

We further evaluated our approach on the Llama-30B model (i.e., models exceeding 20B parameters), as presented in Table 10. On this larger scale, our method continues to outperform prior techniques at identical compression rates. Moreover, we observe that our technique achieves even greater compression efficiency on larger models, yielding a smaller accuracy drop.

I Speedup of End-to-end Decoding

We conducted practical speedup experiments on our method. Table 11 is our current acceleration result using the Llama-3-8B-Instruct model with a batch size of 512, a sequence length of 32, and 128 tokens generated. These results still show a tangible speedup. Typically, benchmarks for such work increase the batch size to make the model compute-bound and achieve higher throughput.

However, we believe that our proposed progressive low-rank decoding is particularly effective for alleviating memory-bound issues as well as situations characterized by low throughput. To further validate this, we conducted an additional experiment under offloading conditions. In this setup, using the same Llama-3-8B-Instruct model, our GPU is limited to approximately 8GB of VRAM; hence, the remaining parameter matrices are offloaded to host DRAM and transferred to GPU VRAM when needed for computation. The experimental settings in this case are: a batch size of 1, sequence length of 32, and generating 128 tokens. Table 12 is our experimental result for offloading. Our approach alleviates the memory transfer requirements, thereby accelerating the overall process. Our results clearly demonstrate that our method yields even more significant acceleration when the system is memory-bound. Additionally, in data transfers, larger data tend to experience increased fragmentation compared to smaller ones. This fragmentation means that the data is divided into more segments or fragments, and each fragment often incurs its own processing overhead. Therefore, in strongly memory-bound situations, FLRC may deliver even better acceleration than theoretically predicted.

Comp. Rate	Method	Zero-shot Task Accuracy (%) ↑							
		ARC-e	ARC-c	Hella	OBQA	Wino	MathQA	PIQA	Avg.
-	Baseline	79.42	48.29	60.05	35.20	72.30	32.13	79.05	58.06
20%	SVD-LLM	67.89	32.76	44.28	29.00	67.56	25.59	71.11	48.31
	FLRC	**70.33**	**38.05**	**47.49**	**31.20**	**69.53**	**27.91**	**72.91**	**51.06**
30%	SVD-LLM	58.71	25.94	37.80	**26.60**	64.56	24.49	66.54	43.52
	FLRC	**63.64**	**30.12**	**41.52**	**26.60**	**66.14**	**24.72**	**68.39**	**45.88**

Table 9: Zero-shot comparison results on Llama2-13B.

Comp. Rate	Method	Wiki2 Perplexity ↓	Dialogsum Rouge-L ↑
-	Baseline	4.10	17.25
20%	SVD-LLM	5.55	16.77
	FLRC	**5.21**	**18.95**
30%	SVD-LLM	6.27	16.31
	FLRC	**5.75**	**18.98**
40%	SVD-LLM	7.58	0.00
	FLRC	**6.62**	**18.19**

Table 10: Performance comparison on 30B model.

Method	Comp. Rate	Throughput (tokens/sec)	Speedup
Baseline	-	3646.62	1x
FLRC	20%	3856.99	1.06x
	30%	4051.53	1.11x
	40%	5290.33	1.45x

Table 11: Speedup of FLRC on Llama-3-8B-Instruct.

As models grow larger and context windows increase, GPU VRAM demand will rise, making offloading scenarios increasingly common for single user and edge device. Under these conditions, the model's inherent throughput can become very low. Therefore, FLRC is particularly beneficial in environments that require model offloading.

J FLRC on Low-precision Model

We conducted our experiments primarily in FP16 precision. As shown in Table 13, our method remains equally effective at lower precisions. We evaluated generation task on both Llama-3-8B-Instruct and Llama-2-7B-Chat models using our approach. The results demonstrate that there is no drop in accuracy across various compression rates, even when using lower-precision models. This confirms that our parameter-reduction technique and low-precision quantization work synergistically.

Method	Comp. Rate	Throughput (tokens/sec)	Speedup
Baseline	-	1.20	1x
FLRC	20%	1.40	1.17x
	30%	1.83	1.53x
	40%	2.54	2.12x

Table 12: Offloading speedup of FLRC on Llama-3-8B-Instruct.

Comp. Rate	Precision	Dialogsum Rouge-L ↑	
		Llama-3-8B-Instruct	Llama-2-7B-Chat
-	FP16	24.72	24.56
20%	FP16	17.35	17.22
	INT8	17.48	17.47
30%	FP16	8.09	14.91
	INT8	7.81	15.19

Table 13: Generative performance comparison on low-precision models.

K Sensitivity Analysis to Calibration Datasets

Most existing SVD-based methods rely on calibration datasets. Table 14 shows experimental results obtained by calibrating on different datasets for compressing Llama-3.2-3B. Notably, when calibrated on Wikitext2, the model exhibits improved perplexity on Wikitext2 but performs worse on C4; conversely, calibration on C4 yields better results on C4 but poorer performance on Wikitext2. This behavior is expected, as models tend to perform better on data that closely resembles the calibration set. Importantly, our results indicate that FLRC consistently achieves lower perplexity than SVD-LLM across different calibration datasets. Therefore, as long as all compared methods are calibrated using the same dataset, the experiments remain fair. In all our experiments, FLRC and the previous methods (ASVD, SVD-LLM) have been calibrated on the identical dataset.

Method	Comp. Rate	Calibration on Wikitext2		Calibration on C4	
		Wiki2 ↓	C4 ↓	Wiki2 ↓	C4 ↓
Baseline	-	7.81	11.33	7.81	11.33
SVD-LLM	10%	14.72	48.03	38.01	29.63
FLRC		**11.39**	**25.79**	**18.99**	**18.55**
SVD-LLM	20%	26.95	120.92	117.74	53.78
FLRC		**19.12**	**58.92**	**42.92**	**27.41**

Table 14: Perplexity on different calibration datasets on Llama-3.2-3B.