SD^2: Self-Distilled Sparse Drafters

Mike Lasby [† 1 2] **Nish Sinnadurai** [1] **Valavan Manoharararajah** [1] **Sean Lie** [1] **Yani Ioannou** [2] **Vithursan Thangarasa** [1]

Abstract

Speculative decoding is a powerful technique for reducing the latency of Large Language Models (LLMs), offering a fault-tolerant framework that enables the use of highly compressed draft models. In this work, we introduce Self-Distilled Sparse Drafters (SD^2), a novel methodology that leverages self-data distillation and fine-grained weight sparsity to produce highly efficient and well-aligned draft models. SD^2 systematically enhances draft token acceptance rates while significantly reducing Multiply-Accumulate operations (MACs), even in the Universal Assisted Generation (UAG) setting, where draft and target models originate from different model families. On a Llama-3.1-70B target model, SD^2 provides a 1.59× higher Mean Accepted Length (MAL) compared to layer-pruned draft models and reduces MACs by over 43.87% with a 8.36% reduction in MAL compared to dense draft models. Our 1.5B and 3B unstructured sparse drafters outperform both dense and layer-pruned models of equivalent size in terms of end-to-end latency improvements; highlighting the potential of sparsity-aware fine-tuning and compression strategies to improve LLM inference efficiency while maintaining alignment with target models.

1 Introduction

Large Language Models (LLMs) have high utility; however, the causal dependency between preceding and subsequent tokens results in approximately 10× higher latency for sequence generation compared to processing an equivalent length sequence in parallel (Liu et al., 2023). Their high computational cost has motivated methods which improve their efficiency[1].

In contrast, speculative decoding (Stern et al., 2018; Leviathan et al., 2023; Chen et al., 2023) offers a unique framework to accelerate token generation *without* sacrificing accuracy. In speculative decoding, a smaller *draft* model is utilized to auto-regressively generate a sequence of *draft tokens* which are verified in parallel by a *target* model. For speculative decoding to be effective, an efficient draft model which is closely aligned with the target model is required. How best to select and/or train a draft model has been the focus of several recent works (Xia et al., 2024).

Fine-grained sparsity, such as unstructured or 2:4 (Mishra et al., 2021) sparsity, for compressing draft models has yet to be examined. Speculative decoding is a uniquely well-suited setting for using sparse draft models since any errors produced during drafting can be gracefully recovered from during the verification step. Other compression strategies such as quantization succeed in reducing memory overhead, but may not reduce latency (Kurtic et al., 2024). While sparse draft models reduce the total amount of Multiply-Accumulate operations (MACs) required per forward pass, achieving practical acceleration requires specialized software kernels (Neural Magic, 2021; 2024; Schultheis and Babbar, 2023; Lasby et al., 2023; Frantar et al., 2025) and hardware such as Cerebras' wafer-scale engine (Thangarasa et al., 2024; Hall et al., 2023; Lie, 2023; Agarwalla et al., 2024).

Given the practical limitations of fine-grained sparsity, *structured pruning* — the removal of layers, neurons, or other substructures — of LLMs to obtain performant draft models is a more hardware-friendly alternative. In particular, pruning entire transformer *blocks* from the decoder of LLMs provides low latency models which retain their quality under moderate compression ratios (Liu et al., 2023; Gromov et al., 2025; Men et al., 2024; Kim et al., 2024; Sun et al., 2025). A natural extension of these works is to leverage layer-pruning techniques to obtain draft models as examined by Thangarasa et al. (2025). However, it is not clear a priori if the model quality degradation of layer-pruned models can be overcome by their improved latency for use as draft models in the speculative decoding setting.

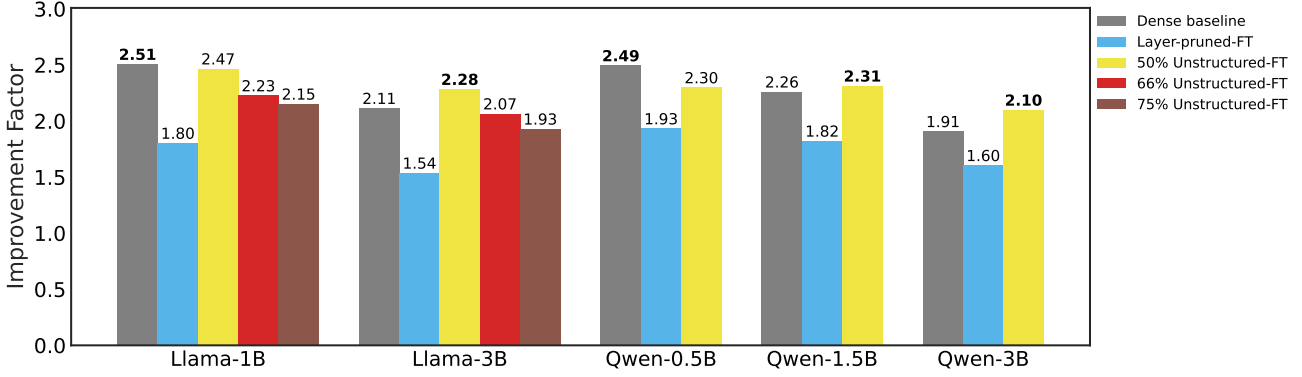[1]See Appendix C for a detailed discussion of related work

Figure 1: **Improvement factor of dense, layer-pruned, and SD^2 unstructured** Llama and Qwen models drafting for Llama-3.1-70B-Instruct and Qwen-2.5-72B-Instruct, respectively. SD^2 drafters outperform layer-pruned draft models and dense drafters in the 1.5B and 3B model size categories.

(a) **UAG Qwen drafting for Llama**
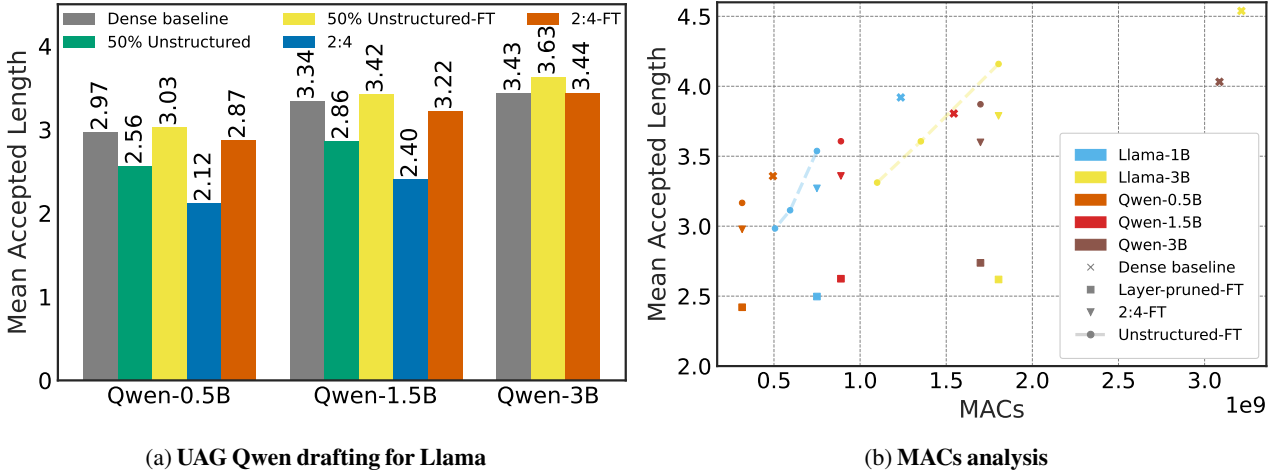
(b) **MACs analysis**

Figure 2: **a: SpecBench MAL for SD^2 Qwen-2.5 models** drafting for Llama-3.1-70B-Instruct in the UAG setting. These results illustrate the benefits of SD^2 for aligning draft models *even across different model families*. SD^2 Qwen drafters achieve a **higher** MAL than their dense counterparts. **b MAL vs. MACs for layer-pruned and sparse draft models.** Particularly notable are the Qwen-2.5 unstructured sparse drafters which approach iso-MAC performance compared to the dense models.

In this work, we propose Self-Distilled Sparse Drafters (SD^2) — a novel methodology for obtaining efficient, well-aligned draft models — by utilizing *self-data distillation fine-tuning* (Yang et al., 2024b; Thangarasa et al., 2025) and weight sparsity. Specifically, we make the following contributions:

- We introduce SD^2, a novel methodology for obtaining fine-grained sparse draft models;
- We demonstrate the superiority of fine-grained sparsity for accelerating speculative decoding and downstream evaluation tasks compared with layer-pruned models;
- We showcase the effectiveness of self-data distillation fine-tuning for model alignment, even when aligning with a different model family with Universal Assisted Generation (UAG);

- When paired with optimized sparse representations, we find that the end-to-end acceleration of speculative decoding with fine-grained sparse draft models in some cases exceeds that of dense draft models.

2 Method

In this section, we provide preliminaries for speculative decoding and introduce the components of SD^2, consisting of self-data distillation, one-shot fine-grained pruning, and sparse fine-tuning. See Appendix D.4 for a discussion of our layer pruning method which serves as a baseline to compare to our sparse drafters.

Speculative decoding. In speculative decoding, draft tokens are produced by auto-regressively sampling from a smaller

draft model, M_d, to produce candidate completions which are verified in parallel by the target model, M_t. Formally, given an input sequence $\{x_1, x_2, ..., x_n\}$, the draft model calculates probability distributions for each token in the completion conditioned on the input sequence and any preceding output tokens: $p_{n+j+1} = M_d(\{x_1, ..., x_n, ...\tilde{x}_{n+1}\}) \forall j \in \{1, ..., k\}$. From these distributions draft tokens are sampled $\tilde{x}_{n+j} \sim p_{n+j}$ to generate a partial completion $\{\tilde{x}_{n+1}, ..., \tilde{x}_{n+k}\}$ of k draft tokens. In the verification stage, the target model M_t computes the output probabilities for each draft token plus one additional token based on its own output distribution: $q_{j+1} = M_t(x_{\leq n}, \tilde{x}_{\leq j}) \forall j \in \{n, ..., n+k+1\}$. For each draft token, $\tilde{x}_j$, the token is accepted if a verification condition based on the draft and target model probabilities is satisfied. In our experiments, we use greedy sampling with a strict top-1 verification criterion which guarantees that generated text matches the output of the original target model precisely, see Appendix D.1. We model the expected *improvement factor* to the overall latency of speculative decoding versus sampling from the target model directly as follows:

$$\text{Improvement Factor} = \frac{\text{MAL}}{kc+1}, \quad (1)$$

where Mean Accepted Length (MAL) is the average number of tokens accepted per round[2], k is the number of draft tokens speculated per round, and c is the cost factor representing the ratio of draft versus target model efficiency. For our practical improvement factor calculations, c represents the wall clock latency ratio, i.e., $c = \frac{\text{timeit}(M_d(x))}{\text{timeit}(M_t(x))}$. For our theoretical improvement factor calculations, we substitute MACs in lieu of latencies.

Self-data distillation. Self-data distillation (SDD) consists of generating a dataset, D_{self}, whose outputs are generated by a model of interest (Yang et al., 2024b). Recently, SDD was extended to speculative decoding for improving the alignment of draft models to a target model; we adopt a similar approach (Goel et al., 2024; Thangarasa et al., 2025). Specifically, given a task specific context $\mathbf{C}$, original input sequence $\mathbf{X}$, and original ground truth label $\mathbf{Y}$, we combine these components into a new input sequence $\tilde{\mathbf{X}}$ which is used to generate the distilled completion, $\tilde{\mathbf{Y}}$, from the target model

$$\tilde{\mathbf{Y}} = M_t(\tilde{\mathbf{X}}) \quad \text{where } \tilde{\mathbf{X}} = \mathbf{C} \| \mathbf{X} \| \mathbf{Y}. \quad (2)$$

In our case, the correctness of the distilled label is irrelevant. The distilled outputs, correct or otherwise, are already aligned with the target model output distribution. See Appendix D.2 for more details.

Fine-grained pruning. To obtain our fine-grained sparse draft models, M_d, we use SparseGPT (Frantar and Alistarh, 2023). However, we emphasize that our method is compat-

---

[2]MAL includes accepted draft tokens and one additional token from the target model per round.

ible with any pruning algorithm. In addition to the original SparseGPT hyperparameter settings, we experiment with non-uniform layer-wise sparsity distributions such as Outlier Weighted Layer-wise sparsity (OWL) (Yin et al., 2024b) and a novel distribution inspired by the angular cosine distance measure from Gromov et al. (2025). See Appendix F for a more thorough discussion of our findings. Ultimately, we use a uniform layer-wise sparsity distribution for all results presented below.

Sparse fine-tuning. In typical supervised fine-tuning settings, a dense LLM M with parameters θ is fine-tuned using a supervised dataset D_f containing input sequences $\mathcal{X}$ and ground truth output sequences $\mathcal{Y}$. For each token, y_j, in the ground truth output sequences, $\mathbf{Y} \in \mathcal{Y}$, the model outputs a probability distribution over its vocabulary conditioned by the input sequences, $\mathbf{X} \in \mathcal{X}$, any output tokens preceding the current token position, and the model parameters: $Q(y_j | \mathbf{X}, \mathbf{Y}_{:j-1}, \theta)$ where $Q(\cdot)$ represents the language modelling head (LM head) logits normalized with the softmax function. The total loss per mini-batch is the average negative log-likelihood across all completion tokens:

$$\mathcal{L}_t = \sum_{n=1}^{N} \frac{1}{S_n} \sum_{j=1}^{S_n} -\log(Q(y_{n,j} | \mathbf{X}_n, \mathbf{Y}_{n,:j-1}, \theta)) \quad (3)$$

where N is the number of samples per batch, and S_n is number of completion tokens per sequence. The model is fine-tuned by minimizing this loss using stochastic gradient descent. In the sparse fine-tuning setting, we simply replace θ with $\theta_s \subset \theta$, the set of non-zero parameters. In our approach, the sparse topology is fixed after one-shot pruning. Pruned parameters remain fixed at zero throughout fine-tuning. For our experiments, we fine-tune our sparse models with a binary mask to initialize pruned parameters to zero and set their gradients to zero during backpropagation via backwards hook. The backwards hook ensures that gradients of pruned parameters do not contribute to the partial derivates of active parameters nor optimizer buffer states. See Algorithm 2 for more details.

3 Results

To empirically evaluate SD^2 and our primary hypothesis that sparse draft models can outperform dense or layer-pruned models, we evaluate a variety of draft model candidates across the Llama-3.2 (Llama Team, AI @ Meta, 2024) and Qwen-2.5 (Yang et al., 2025) model families. For Llama, we evaluate drafting with Llama-3.2 1B-Instruct and 3B-Instruct for Llama-3.1-70B-Instruct. For Qwen-2.5, we evaluate 0.5B-Instruct, 1.5B-Instruct, 3B-Instruct drafting for Qwen-2.5-72B-Instruct. We also consider the UAG setting in which the Qwen draft models are aligned to and evaluated with a Llama-3.1-70B-Instruct target model. We prune and fine-tune fine-grained sparse draft models

with both unstructured and 2:4 sparsity using a uniform layer-wise sparsity distribution and prune all decoder blocks, excluding the embedding and LM head. The sparsity levels reported in our results refer to the overall sparsity of the decoder. As baselines, we compare these candidates with 50% layer-pruned[3] and dense models. See Appendix E for hyperparameters and implementation details.

OpenLLM leaderboard V1 benchmarks. We evaluate our models using the default multi-shot settings on the OpenLLM Leaderboard V1 (HuggingFace, 2024) tasks using the EleutherAI evaluation harness (Gao et al., 2023). These tasks include: 25-shot ARC-C (Clark et al., 2018), 5-shot strict exact match GSM8k (Cobbe et al., 2021), 10-shot HellaSwag (Zellers et al., 2019), 5-shot MMLU (Hendrycks et al., 2021), 5-shot Winogrande (Sakaguchi et al., 2021), and 0-shot multi-true (MC2) TruthfulQA (Lin et al., 2022). We report byte-length normalized accuracies for ARC-C and HellaSwag[4].

See Figure 9 for the mean accuracy for models pruned and fine-tuned using the SD^2 methodology. We find that one-shot pruned models suffer high degradation on the tasks evaluated; however, after fine-tuning the fine-grained sparse models recover much of their accuracy, particularly the 50% unstructured and 2:4 sparse models. For the Qwen-2.5 1.5 and 3B models, we find that the unstructured 50% models approach or *exceed* the accuracy of the dense baseline. Notably, the 50% layer-pruned models suffer high degradation, even after fine-tuning. See Appendix I for more detailed results.

Speculative decoding. SpecBench (Xia et al., 2024) assesses draft token acceptance rate across a variety of tasks including translation, summarization, question answering, mathematical reasoning, multi-turn conversation, and retrieval augmented generation (RAG).

In Figure 4, we report the MAL on SpecBench (Xia et al., 2024) for a variety of models and sparsities. We find that the SD^2 50% unstructured drafters achieve the highest MAL of the sparse models, but do not exceed the dense baselines. For Qwen-2.5, the SD^2 50% unstructured drafter achieve a MAL within 0.2 of the dense models across all model sizes investigated, despite lower overall scores compared to the Llama model family. We speculate that the overall higher MAL for the Llama-3.2 draft models are the result of improved alignment with the target model stemming from the pruning and distillation process used during pretraining (Llama Team, AI @ Meta, 2024).

The importance of the draft model alignment is clear; our self-data distillation and fine-tuning process enables much higher MAL than the one-shot pruned models. Even more strik-

ingly, the SD^2 effectively aligns the draft model in the UAG setting *across model families*, as can be seen in Figure 2a. Remarkably, the sparse Qwen drafters outperform their dense counterparts under this setting, further highlighting the importance of draft model alignment and the benefits of SD^2.

MACs analysis. The prior section demonstrate that SD^2 drafters have comparable quality to dense models, it remains unclear whether their efficiency gains yield offer practical benefits. Unstructured sparsity is not well suited for Graphics Processing Units (GPUs) and tangible latency improvements require specialized kernels. An analysis based on MACs provides a theoretical limit of the benefits one can expect from unstructured sparsity.

In Figure 2b, we plot the MAL of our models trained with SD^2 versus their corresponding MACs. The unstructured sparse Qwen-2.5 drafters nearly achieve iso-MAC performance improvements compared to their dense counterparts. However, additional MACs associated with the LM head prevent the unstructured sparse models from directly outperforming the dense drafters in terms of MAL vs. MACs. In Figures 5 and 6 we demonstrate that our 3B SD^2 draft models provide the highest *theoretical* improvement factor of all draft model candidates examined.

Compressed sparse drafters. As noted in Equation (1), a crucial property to consider when evaluating speculative decoding draft models is c, the ratio of the draft to target model latency. To realize practical benefits of SD^2 on commodity hardware available today, we require efficient sparse representations and kernels which offer lower latency than the dense baselines. To this end, we benchmark our SD^2 unstructured draft models using nm-vLLM (Neural Magic, 2024; Kwon et al., 2023) and compare them with dense and layer-pruned draft models in Figure 1. SD^2 drafters outperform layer-pruned models and even dense drafters in the 3B model size category. While the smallest dense model in each model family achieves a higher overall improvement factor, these results highlight the potential for sparsity to enable the use of larger draft models. As foundation model sizes continue to grow, increasing the capacity of draft models may be necessary to maintain sufficient MAL and support novel use-cases such as scaling test-time compute. See Appendix G for details on our benchmarking methodology and additional benchmark results.

4 Conclusion

We introduce the SD^2 methodology for obtaining fine-grained sparse draft models. SD^2 consists of self-data distillation, one-shot pruning, and sparse fine-tuning. We find that self-data distillation effectively aligns draft models, even if their target is from a different model family such as in the UAG setting. SD^2 shows significant theoretical benefits

---

[3]Specifically, 50% of the decoder blocks are pruned.

[4]Reported as the `acc_norm` field in the EleutherAI evaluation harness outputs. See Gao (2021) for details.

based MACs. When paired with efficient sparse kernels, SD^2 outperforms layer-pruned models at all model sizes and compression ratios evaluated. For the 3B draft models, SD^2 improves end-to-end speculative decoding latency compared to dense draft models, revealing the potential for using larger drafters. See Appendix B for a discussion of limitations and future work. We hope this work will inspire further efforts to develop low-latency sparse kernels and encourage software/hardware co-design for efficient LLM inference.

References

Abhinav Agarwalla, Abhay Gupta, Alexandre Marques, Shubhra Pandit, Michael Goin, Eldar Kurtic, Kevin Leong, Tuan Nguyen, Mahmoud Salem, Dan Alistarh, Sean Lie, and Mark Kurtz. 2024. Enabling High-Sparsity Foundational Llama Models with Efficient Pretraining and Deployment. ArXiv:2405.03594 [cs].

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *Proceedings of the Forty-first International Conference on Machine Learning*, pages 5209–5235. PMLR.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating Large Language Model Decoding with Speculative Sampling. ArXiv:2302.01318 [cs].

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. 2025. Efficientqat: Efficient quantization-aware training for large language models. ArXiv:2407.11062 [cs].

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:1803.05457 [cs].

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

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale. In *Proceedings of the Thirty-Sixth Conference on Neural Information Processing Systems*.

Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. 2024. Maskllm: Learnable semi-structured sparsity for large language models. In *Proceedings of the Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Elias Frantar, Roberto L Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. 2025. Marlin: Mixed-precision auto-regressive parallel inference on large language models. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pages 239–251.

Leo Gao. 2021. Multiple Choice Normalization in LM Evaluation.

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. 2023. A framework for few-shot language model evaluation.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. In *Low-power computer vision*, pages 291–326. Chapman and Hall/CRC.

Raghavv Goel, Mukul Gagrani, Wonseok Jeon, Junyoung Park, Mingu Lee, and Christopher Lott. 2024. Direct alignment of draft model for speculative decoding with chat-fine-tuned LLMs. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Dan Roberts. 2025. The unreasonable ineffectiveness of the deeper layers. In *The Thirteenth International Conference on Learning Representations*.

Stewart Hall, Sean Lie, and Rob Schreiber. 2023. Training giant neural networks using weight streaming on cerebras wafer-scale clusters. *Cerebras Systems, Inc., Tech. Rep.*

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *Proceedings of the Ninth International Conference on Learning Representations*.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with

less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *Proceedings of the Tenth International Conference on Learning Representations*.

HuggingFace. 2024. Open LLM Leaderboard v1.

Ajay Kumar Jaiswal, Zhe Gan, Xianzhi Du, Bowen Zhang, Zhangyang Wang, and Yinfei Yang. 2024. Compressing llms: The truth is rarely pure and never simple. In *Proceedings of the Twelfth International Conference on Learning Representations*.

Ananya Harsh Jha, Tom Sherborne, Evan Pete Walsh, Dirk Groeneveld, Emma Strubell, and Iz Beltagy. 2024. Just CHOP: Embarrassingly Simple LLM Compression. ArXiv:2305.14864 [cs].

Yixin Ji, Yang Xiang, Juntao Li, Qingrong Xia, Ping Li, Xinyu Duan, Zhefeng Wang, and Min Zhang. 2025. Beware of calibration data for pruning large language models. In *Proceedings of the Thirteenth International Conference on Learning Representations*.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. 2024. Shortened LLaMA: Depth Pruning for Large Language Models with Comparison of Retraining Methods. ArXiv:2402.02834 [cs].

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 1317–1327.

Eldar Kurtic, Denis Kuznedelev, Elias Frantar, Michael Goin, and Dan Alistarh. 2023. Sparse Finetuning for Inference Acceleration of Large Language Models. ArXiv:2310.06927 [cs].

Eldar Kurtic, Alexandre Marques, Shubhra Pandit, Mark Kurtz, and Dan Alistarh. 2024. "Give Me BF16 or Give Me Death"? Accuracy-Performance Trade-Offs in LLM Quantization. ArXiv:2411.02355 [cs].

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*.

Mike Lasby, Anna Golubeva, Utku Evci, Mihai Nica, and Yani Ioannou. 2023. Dynamic Sparse Training with Structured Sparsity. In *Proceedings of the Twelfth International Conference on Learning Representations*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Guha, Sedrick Scott Keh, Kushal Arora, et al. 2024a. Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. Eagle-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7421–7432.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024c. Eagle: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the Forty-first International Conference on Machine Learning*.

Sean Lie. 2023. Cerebras Architecture Deep Dive: First Look Inside the Hardware/Software Co-Design for Deep Learning. *IEEE Micro*, 43(3):18–30.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. 2024a. Kangaroo: Lossless Self-Speculative Decoding via Double Early Exiting. ArXiv:2404.18911 [cs].

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024b. Dora: Weight-decomposed low-rank adaptation. In *Proceedings of the Forty-first International Conference on Machine Learning*.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. 2023. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR.

Llama Team, AI @ Meta. 2024. The Llama 3 Herd of Models. ArXiv:2407.21783 [cs].

Ilya Loshchilov and Frank Hutter. 2018. Decoupled Weight Decay Regularization. In *Proceedings of the Seventh International Conference on Learning Representations*.

Haiquan Lu, Yefan Zhou, Shiwei Liu, Zhangyang Wang, Michael W Mahoney, and Yaoqing Yang. 2024. Alphapruning: Using heavy-tailed self regularization theory for improved layer-wise pruning of large language models. *Advances in Neural Information Processing Systems*, 37:9117–9152.

Shi Luohe, Hongyi Zhang, Yao Yao, Zuchao Li, et al. 2024. Keep the cost down: A review on methods to optimize llm's kv-cache consumption. In *Proceedings of the First Conference on Language Modeling*.

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.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. ShortGPT: Layers in Large Language Models are More Redundant Than You Expect. ArXiv:2403.03853 [cs].

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *Proceedings of the Fifth International Conference on Learning Representations*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. 2024. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 932–949.

Seyed Iman Mirzadeh, Keivan Alizadeh-Vahid, Sachin Mehta, Carlo C. del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. 2023. ReLU Strikes Back: Exploiting Activation Sparsity in Large Language Models. In *Proceedings of the Twelfth International Conference on Learning Representations*.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. 2021. Accelerating Sparse Deep Neural Networks. ArXiv:2104.08378 [cs].

Neural Magic. 2021. Deepsparse engine: Sparsity-aware deep learning inference runtime for CPUs.

Neural Magic. 2024. Neural magic nm-vLLM inference engine.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian En-Hsu Yen, Avner May, Tianqi Chen, and Beidi Chen. 2024. MagicDec: Breaking the Latency-Throughput Tradeoff for Long Context Generation with Speculative Decoding. In *Proceedings of the Thirteenth International Conference on Learning Representations*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Mohammad Samragh, Mehrdad Farajtabar, Sachin Mehta, Raviteja Vemulapalli, Fartash Faghri, Devang Naik, Oncel Tuzel, and Mohammad Rastegari. 2023. Weight subcloning: direct initialization of transformers using larger pretrained ones. ArXiv: 2312.09299 [cs].

Erik Schultheis and Rohit Babbar. 2023. Towards memory-efficient training for extremely large output spaces–learning with 670k labels on a single commodity gpu. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 689–704. Springer.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. 2024a. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. In *Proceedings of the First Conference on Language Modeling*.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2024b. A simple and effective pruning approach for large language models. In *Proceedings of the Twelfth International Conference on Learning Representations*.

Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. 2025. Transformer layers as painters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(24):25219–25227.

Vithursan Thangarasa, Abhay Gupta, William Marshall, Tianda Li, Kevin Leong, Dennis DeCoste, Sean Lie, and Shreyas Saxena. 2023. SPDF: sparse pre-training and dense fine-tuning for large language models. In *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*, pages 2134–2146.

Vithursan Thangarasa, Shreyas Saxena, Abhay Gupta, and Sean Lie. 2024. Sparse-IFT: Sparse iso-FLOP transformations for maximizing training efficiency. In

Proceedings of the Forty-first International Conference on Machine Learning.

Vithursan Thangarasa, Ganesh Venkatesh, Mike Lasby, Nish Sinnadurai, and Sean Lie. 2025. Self-Data Distillation for Recovering Quality in Pruned Large Language Models. In *Proceedings of the Eighth Conference on Machine Learning and Systems*.

Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conferenec of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. 2025. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. In *The Thirteenth International Conference on Learning Representations*.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 7655–7671.

Wenda Xu, Rujun Han, Zifeng Wang, Long T. Le, Dhruv Madeka, Lei Li, William Yang Wang, Rishabh Agarwal, Chen-Yu Lee, and Tomas Pfister. 2024a. Speculative Knowledge Distillation: Bridging the Teacher-Student Gap Through Interleaved Sampling. ArXiv:2410.11325 version: 1.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. 2024b. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. ArXiv: 2406.08464 [cs].

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report.

Yifei Yang, Zouying Cao, and Hai Zhao. 2024a. LaCo: Large Language Model Pruning via Layer Collapse. ArXiv:2402.11187 [cs].

Zhaorui Yang, Tianyu Pang, Haozhe Feng, Han Wang, Wei Chen, Minfeng Zhu, and Qian Liu. 2024b. Self-distillation bridges distribution gap in language model fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1028–1043.

Lu Yin, Ajay Jaiswal, Shiwei Liu, Souvik Kundu, and Zhangyang Wang. 2024a. Pruning small pre-trained weights irreversibly and monotonically impairs "difficult" downstream tasks in llms. ArXiv:2310.02277 [cs].

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, Ajay Jaiswal, Mykola Pech-enizkiy, Yi Liang, et al. 2024b. Outlier weighed layerwise sparsity (owl): A missing secret sauce for pruning llms to high sparsity. In *Proceedings of the Forty-First International Conference on Machine Learning*.

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*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2024a. Draft& verify: Loss-less large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11263–11282.

Yingtao Zhang, Haoli Bai, Haokun Lin, Jialin Zhao, Lu Hou, and Carlo Vittorio Cannistraci. 2024b. Plug-and-play: An efficient post-training pruning method for large language models. In *Proceedings of the Twelfth International Conference on Learning Representations*.

Zhenyu Zhang, Shiwei Liu, Runjin Chen, Bhavya Kailkhura, Beidi Chen, and Atlas Wang. 2024c. Q-Hitter: A Better Token Oracle for Efficient LLM Inference via Sparse-Quantized KV Cache. *Proceedings of Machine Learning and Systems*, 6:381–394.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Association for Computational Linguistics.

A   Impact Statement

Training and serving LLMs consume large amounts of energy, even for efficient implementations that leverage techniques discussed in this paper. Speculative decoding requires more VRAM than auto-regressive sampling, necessitating the use of additional hardware devices. There are many potential negative societal consequences of generative AI, we hope that by improving the efficiency of these models we will help distribute the benefits of such systems more equitably to the broader public.

B   Limitations and future work

The effectiveness of SD^2 in improving inference efficiency depends on software and hardware optimized for sparsity. The nm-vLLM unstructured sparse kernel we use to benchmark is optimized for Ampere-series NVIDIA GPUs, and evaluating its performance on alternative hardware, such as TPUs or custom accelerators, is crucial for broader adoption. While SD^2 requires significant compute for fine-tuning and self-data distillation, this investment could further enhance the quality of dense models, particularly in speculative decoding. Several promising research directions emerge from this work. One is integrating quantization-aware training (Chen et al., 2025), draft-token trees, and compressed KV-cache implementations (Luohe et al., 2024) with SD^2 to improve memory and compute efficiency. Another is benchmarking SD^2 on hardware optimized for unstructured sparsity, such as Cerebras' wafer-scale engine. Incorporating advanced fine-tuning strategies like speculative knowledge distillation (Xu et al., 2024a) or square-head distillation (Kurtic et al., 2023) could further improve sparse draft model quality. Finally, combining structured pruning with fine-grained sparsity and quantization presents an opportunity to reduce inference-time latency while maintaining accuracy, making sparsity-aware methods like SD^2 more widely applicable across diverse hardware architectures.

C   Related work

The high computational cost of LLMs has motivated significant research into methods which improve their efficiency, including: quantization (Gholami et al., 2022; Kurtic et al., 2024), pruning (Ma et al., 2023; Gromov et al., 2025), weight sparsity (Frantar and Alistarh, 2023; Sun et al., 2024b; Yin et al., 2024b), activation sparsity (Mirzadeh et al., 2023), KV-cache compression (Zhang et al., 2024c), distillation (Kim and Rush, 2016; Hsieh et al., 2023), and matrix decomposition (Hu et al., 2021; Liu et al., 2024b). However, these methods typically trade improved efficiency for decreased model quality. (Yin et al., 2024a; Jaiswal et al., 2024).

Speculative decoding with greedy verification was initially proposed by Stern et al. (2018). Speculative sampling was introduced concurrently by Leviathan et al. (2023) and Chen et al. (2023). Since these initial investigations, several works have proposed modifications and refinements to the original framework. In Medusa (Cai et al., 2024), the authors used a single model with multiple lm-head layers, each of which predicts multiple draft token candidates across each draft token position, introducing the first instance of draft *token-trees*. Specifically related to our work, Medusa also used self-distillation to obtain a fine-tuning dataset. Sadhukhan et al. (2024) examined speculative decoding for long context lengths, finding that constant KV-cache sizes lead to improved acceleration. EAGLE (Li et al., 2024c;b) proposed an efficient draft model consisting of a single layer auto-regression head trained from scratch and the frozen embedding and lm-head layers from the target model. Token trees were further investigated by Miao et al. (2024) in which a parallel decoding algorithm was also proposed. Sun et al. (2024a) introduced a hierarchical framework in which efficient KV-cache implementations are used for drafting. Speculative decoding was combined with early-exit mechanisms for drafting by Zhang et al. (2024a) and Liu et al. (2024a). We note that optimizations such as draft token-trees and KV-cache compression are orthogonal to SD^2 and future work could consider their integration with our method.

The surprising result that a significant fraction of decoder blocks can be pruned from LLMs without incurring excessive quality reduction was examined in detail in several works (Jha et al., 2024; Sun et al., 2025; Yang et al., 2024a; Men et al., 2024). In particular, these works found that the "middle" decoder blocks are the most amenable to pruning. However, abstract reasoning tasks were noted to be disproportionality affected by layer-pruning. We specifically highlight Gromov et al. (2025) which introduced the angular cosine distance metric that is used for our layer-pruned baselines.

Fine-grained sparsity for LLMs has received significant attention from the research community. Efficient one-shot pruning methods such as SparseGPT (Frantar and Alistarh, 2023), Wanda (Sun et al., 2024b), and Plug-and-Play (Zhang et al., 2024b) proposed re-framing the pruning procedure as a layer-wise reconstruction of the original model weights, using the magnitudes and/or activations to determine the saliency of individual weights. Fang et al. (2024) proposed freezing the model weights and training the mask alone to efficiently obtain 2:4 sparse LLMs. OWL (Yin et al., 2024b) and AlphaPruning

---

Algorithm 1 Self-data distillation for speculative decoding

1: **Input:** Pretrained target model M_t, fine-tuning dataset D_{sft}, and self-data distillation (SDD) context $\mathbf{C}$.
2: Initialize $D_{sdd} = \emptyset$
3: **for** $\mathbf{X}_i, \mathbf{Y}_i \in D_{\text{sft}}$ **do**
4:     $\tilde{\mathbf{X}}_i \leftarrow \mathbf{C}||\mathbf{X}_i||\mathbf{Y}_i$ `# Combine into new prompt`
5:     $\tilde{\mathbf{Y}}_i \leftarrow M_t(\tilde{\mathbf{X}}_i)$ `# Generate new label`
6:     $D_{sdd}.\text{append}((\mathbf{X}_i, \tilde{\mathbf{Y}}_i))$ `# Accept `$\tilde{\mathbf{Y}}_i$` w/o verification`
7: **end for**
8: **Output:** D_{sdd}

(Lu et al., 2024) proposed non-uniform layer-wise sparsity distributions which outperform uniform distributions in terms of perplexity and downstream task accuracy. Despite these efforts, sparse LLMs still fall short of their dense counterparts in many respects (Jaiswal et al., 2024; Thangarasa et al., 2023; Yin et al., 2024a).

D   Extended method discussion

D.1   Speculative decoding

Equations (4) and (5) formally define our greedy sampling and verification schemes, respectively:

$$\tilde{x}_{n+j} = \text{argmax}\, p_j \quad \forall j \in \{1, ..., k\}, \tag{4}$$

$$x_{n+j} = \begin{cases} \tilde{x}_{n+j} & \text{if } \tilde{x}_{n+j} = \text{argmax}\, q_{n+j} \\ \text{argmax}\, q_{n+j} & \text{otherwise.} \end{cases} \tag{5}$$

D.2   Self-data distillation

In the original formulation of self-data distillation, Yang et al. (2024b) only extract the distilled label $\tilde{\mathbf{Y}}$ if and only if it aligns with the original ground truth label, $\mathbf{Y^t}$. For instance, only accepting the distilled label if it is mathematically equivalent to the ground truth label. However, for tasks that do not yield a closed-form solution, it can be challenging to assess the validity of the distilled label.

The speculative decoding setting gracefully eliminates the need to consider the distilled label verification process. Since our fundamental goal is aligning the draft with the target model, the correctness of the distilled label is irrelevant. We accept the distilled labels even if they contain easily identifiable errors; the distilled output, correct or otherwise, is already aligned with the target model output distribution. See Algorithm 1 for a summary of the self-data distillation process.

D.3   Sparse fine-tuning

See Algorithm 2 for a formal description of our end-to-end algorithm including sparse fine-tuning.

D.4   Layer pruning

We obtain our layer-pruned baselines as follows. Consider a draft model M_d with N decoder blocks. Each decoder block consists of a multi-headed self-attention module followed by a feed-forward module. The decoder blocks are stacked sequentially such that the output of the preceding block is the input to the following block. The final block's output is typically used for downstream tasks such as language modelling or sequence classification. The goal of layer pruning is to identify and prune n sequential decoder blocks such that the resultant quality degradation on relevant downstream tasks is minimized.

Various approaches have been proposed for identifying the most important blocks in a transformer (Samragh et al., 2023; Men et al., 2024). Following Thangarasa et al. (2025), we elect to use the angular cosine distance measure as proposed by Gromov et al. (2025). This metric uses the angular cosine distance between the input and output of a group of n sequential decoder blocks to define block importance. The group of blocks with the highest similarity between their input and output are considered to be more redundant than other group candidates and therefore can be pruned with the smallest impact on the overall model output. Formally, we define the angular cosine distance measure as follows:

Algorithm 2 SD^2: Self-distilled sparse drafters

1: **Input:** Draft model M_d with parameters θ, target model M_t, calibration dataset D_{cal}, supervised fine-tuning dataset D_{sft}, self-data distillation (SDD) context $\mathbf{C}$, optimizer $\mathcal{O}$, learning rate α, number of iterations T, and batch size N.
2: **Output:** Fine-tuned sparse draft model, M_d'.
3: **Define** `SparsityHook`$(\nabla_\theta \mathcal{L}_t, \theta, \theta_p)$:
4:   **for** $p_i, \frac{\partial \mathcal{L}_t}{\partial p_i} \in \{\theta, \nabla_\theta \mathcal{L}_t\}$ **do**
5:     **if** $p_i \in \theta_p$ **then**
6:       $\frac{\partial \mathcal{L}_t}{\partial p_i} \leftarrow 0$
7:     **end if**
8:   **end for**
9:   **return** $\nabla_{\theta_s} \mathcal{L}_t$
10: $M_d' \leftarrow \texttt{SparseGPT}(M_d, D_{\text{cal}})$
11: $\theta_p \leftarrow \{p_i \in \theta \mid p_i = 0\}$
12: $D_{sdd} \leftarrow \texttt{SDD}(M_t, D_{sft}, \mathbf{C})$ # Algorithm 1
13: $M_d.\texttt{register}(\texttt{partial}(\texttt{SparsityHook}(\theta, \theta_p)))$
14: **for** $t = 1$ **to** T **do**
15:   $\{(\mathbf{X}_n, \tilde{\mathbf{Y}}_n)\}_{n=1}^N \sim D_{sdd}$
16:   $C_t \leftarrow \sum_{n=1}^N \texttt{len}(\tilde{\mathbf{Y}}_n)$
17:   $\mathcal{L}_t \leftarrow \sum_{n=1}^N \sum_{j=1}^{\texttt{len}(\tilde{\mathbf{Y}}_n)} -\log M_d(\tilde{y}_{n,j} \mid \mathbf{X}_n, \tilde{\mathbf{Y}}_{n,:j-1})$
18:   $\mathcal{L}_t \leftarrow \mathcal{L}_t / C_t$
19:   $\nabla_\theta \mathcal{L}_t \leftarrow \mathcal{L}_t.\texttt{backwards}()$ # Triggers SparsityHook
20:   $\theta \leftarrow \mathcal{O}(\theta, \nabla_\theta \mathcal{L}_t, \alpha)$
21: **end for**
22: **Return:** M_d'

$$d(x_{D_t}^i, x_{D_t}^{i+n}) = \frac{1}{\pi} \arccos\left(\frac{x_{D_t}^i \cdot x_{D_t}^{i+n}}{\|x_{D_t}^i\| \|x_{D_t}^{i+n}\|} \right), \tag{6}$$

where $x_{D_t}^i$ and $x_{D_t}^{i+n}$ are vectors representing the inputs to blocks i and $i+n$, respectively, for the last token t in each sequence across a representative calibration dataset D. The dot product of these vectors is normalized using the L^2 norm ($\|\cdot\|$) to facilitate comparisons between different groups of blocks. To identify the optimal i to begin pruning, we simply identify the group of blocks with the smallest angular distance:

$$i^*(n) = \underset{i}{\arg\min}\, d(x_{D_t}^i, x_{D_t}^{i+n}), \tag{7}$$

where $i^*(n)$ is the starting block index for the block group of length n with the minimal angular cosine distance for a given calibration dataset subset. Once identified, decoder blocks i^* to $i^* + (n-1)$ are pruned and the outputs of layer i^* connect as input to block $i^* + n$ to obtain the pruned draft model M_d'.

E   Hyperparameter settings and implementation details

The core components of SD^2 consist of: one-shot pruning, self-data distillation, and sparse fine-tuning.

E.1   One-shot pruning

For one-shot pruning, we use the default SparseGPT hyperparameters. Explicitly, we use 0.01, 128, and 16 bits for the Hessian damping, block size, and model precision, respectively. For calibration, we randomly select 128 samples from a subset[5] of the DCLM dataset (Li et al., 2024a) with a sequence length of 2048 tokens. We selected the DCLM calibration dataset based on the results of Ji et al. (2025).

We prune our draft model candidates to 0.5, 0.66, and 0.75 unstructured sparsity in addition to 2:4 sparsity. For our experiments

---

[5] https://huggingface.co/datasets/robbiegwaldd/dclm-micro

with non-uniform layer-wise sparsity distributions, we explore both OWL and our proposed angular distance layer-wise distribution, the results of which are presented in Appendix F.

However, despite apparent advantages in terms of perplexity when measured on WikiText-V2 (Merity et al., 2017), we find that the non-uniform layer-wise sparsity distributions considered offer little benefit to downstream evaluation tasks or MAL during speculative decoding. Furthermore, since non-uniform layer-wise sparsity distribution pose a challenge for inference systems that leverage pipelining with continuous batching, we opt to use the more straightforward uniform layer-wise sparsity distribution for all results included in Section 3.

E.2 Self-data distillation

Our self-data distillation follows the original implementation[6] of Yang et al. (2024b). We generate the distilled labels using 16 bit float precision for the model weights and a maximum generation length of 4096 tokens. We sample from the target model with top-p sampling with $p = 1.0$ and a temperature of 0.9. Using Llama-3.1-70B, we use self-data distillation to produce aligned fine-tuning datasets for Opus-100 translation (Tiedemann and Thottingal, 2020)[7] and MathInstruct-V2 (Toshniwal et al., 2025) datasets. Using Qwen-2.5-72B-Instruct, we distil MathInstruct-V2.

We create the distillation inputs by combining the original input, original output, and context with the default chat templates. For MathInstruct-V2 we applied the following template:

```
1   _register_template(
2       name="llama3_orig_math_distill",
3       format_user=StringFormatter(
4           slots=[
5               (
6                   "<|start_header_id|>user<|end_header_id|>\nQuestion
                        :\n\n{{content}}\nAnswer:\n\n{{resp}}\n\nGreat! Let's think step by step.<|eot_id|>"
7                   "<|start_header_id|>assistant<|end_header_id|>\n\n"
8               )
9           ]
10      ),
11      format_system
                =StringFormatter(slots=["<|start_header_id|>system<|end_header_id|>\n\n{{content}}<|eot_id|>"]),
12      format_observation=StringFormatter(
13          slots=[
14              (
15                  "<|start_header_id|>tool<|end_header_id|>\n\n{{content}}<|eot_id|>"
16                  "<|start_header_id|>assistant<|end_header_id|>\n\n"
17              )
18          ]
19      ),
20      format_prefix=EmptyFormatter(slots=[{"bos_token"}]),
21      default_system=(
22          "You are a math expert tasked with generating
                high-quality responses. You will be provided with a math question and a reference answer. Your
                goal is to rewrite the reference answer in a clear and accurate manner, ensuring it thoroughly
                addresses the question. Maintain mathematical rigor while improving clarity where necessary."
23      ),
24      stop_words=["<|eot_id|>"],
25      replace_eos=True,
26      replace_jinja_template=False,
27  )
```

For Opus, we applied the following template:

```
1   _register_template(
2       name="llama3_multilingual_distill",
3       format_user=StringFormatter(
4           slots=[
5               (
6                   "<|start_header_id|>user<|end_header_id|>\n\n{{content}}\n\n{{resp}}<|eot_id|>"
7                   "<|start_header_id|>assistant<|end_header_id|>\n\n"
8               )
9           ]
10      ),
11      format_system=StringFormatter(
12          slots=[{"bos_token"}, "<|start_header_id|>system<|end_header_id|>\n\n{{content}}<|eot_id|>"]
13      ),
14      default_system=(
```

---

[6]https://github.com/sail-sg/sdft
[7]https://huggingface.co/datasets/Helsinki-NLP/opus-100

```
15            "You are a language translation expert tasked ↘
                  with generating high-quality translations. You will be provided with a sentence or passage in the ↘
                  source language and a reference translation. Your goal is to rewrite the reference translation to ↘
                  ensure it is both accurate and fluent, preserving the original meaning while improving clarity and↘
                   readability. Ensure cultural nuances and context are respected during the translation process."
16         ),
17         stop_words=["<|eot_id|>"],
18         replace_eos=True,
19     )
```

In addition to these datasets, we leverage publicly available datasets produced using the self-synthesis method Magpie (Xu et al., 2024b) for both Llama-3.1-70B-Instruct[8] and Qwen-2.5-72B-Instruct[9] target models. As the Magpie datasets are generated by the target models, further alignment with self-data distillation is redundant.

E.3   Sparse fine-tuning

For sparse fine-tuning, we extend Llama-Factory (Zheng et al., 2024) to incorporate our sparse fine-tuning method. We fine-tune our models using a maximum sequence length of 8,192 tokens and truncate any tokens which exceed this limit. For Llama models, we randomly interleave samples from all three fine-tuning datasets with sampling probabilities 75, 12.5, and 12.5% from Magpie, distilled Opus translation, and distilled MathInstruct-V2, respectively. For Qwen, we randomly interleave samples from Magpie and distilled MathInstruct-V2 with probabilities 80 and 20%, respectively.

We optimize our models using the default AdamW optimizer (Loshchilov and Hutter, 2018) with 1.0e-08, 0.9, and 0.999 for the ε, β_1, and β_2 hyperparameters, respectively. For most of our models, we train them with 16,000 optimizer steps with a batch size of 8 for a total of 128,000 samples. The one exception to the above are the 75% sparse models, which we find benefit from an extended training duration of 32,000 optimizer steps with a batch size of 8 for a total of 256,000 samples.

We use a grid search optimize the learning rate for all Llama draft model and sparsity combinations. In general, we find that smaller and more sparse models required higher learning rates. Rather than performing a second grid search, we simply re-use the optimal learning rates found for Llama models when fine-tuning Qwen models, using the Llama-3.2-1B-Instruct learning rates for both 0.5B and 1.5B Qwen models. See Table 1 for the various learning rates used based on the model size and sparsity type. For learning rate schedule, we use a linear schedule with a linear warm-up using 5% of the total steps. We also experimented with a cosine schedule but found no significant difference in the validation loss.

E.4   Computational Resources

We use an internal cluster to run our experiments on a node with four Nvidia H100-80GB-HBM3 GPUs, 192 Intel Xeon Platinum 8468 CPUs, and 2TB of RAM. Pruning was completed using a single GPU in less than 30 minutes per model. Our fine-tuning datasets were generated in approximately 6-8 hours per dataset. For sparse fine-tuning, we use a single GPU with the following approximate runtimes: 10 and 21 hours for 1B and 3B, respectively, for the 50% and 66% sparse models. The 75% sparse model fine-tuning took approximately 20 and 42 hours for 1B and 3B, respectively, due to the increased number of optimization steps.

F   Non-uniform layer-wise sparsity distributions

This section highlights our analysis of non-uniform layer-wise sparsity distributions such as OWL (Yin et al., 2024b) and a novel distribution inspired by the angular cosine distance measure from Gromov et al. (2025). Ultimately, we found that the non-uniform layer-wise sparsity distributions examined resulted in improved perplexity on WikiText-V2 and a modest improvement on the OpenLLM Leaderboard V1 suite of benchmarks; however, we did not find a statistically significant improvement to MAL during speculative decoding.

OWL allocates sparsity to layers proportional to their *outlier ratio*. The outlier ratio was inspired by the observation that the activations of LLMs often contain large outliers (Dettmers et al., 2022). Specifically, for a weight matrix $\mathbf{W}$ of shape (C_{out}, C_{in}) and input X, OWL defines the outlier score of $\mathbf{W}_{i,j}$ as $A_{i,j} = \|X_j\| \cdot |\mathbf{W}_{i,j}|$ across all tokens in a calibration dataset. The sparsity ratio is defined as $S^l \propto 1 - D^l \quad S^l \in \{S - \lambda, S + \lambda\}$ where $S \in [0,1]$ is the uniform sparsity target, λ is a hyperparameter which constraints the maximum and minimum layer sparsities, and D^l is the outlier distribution for a single layer calculated as

---

[8]https://huggingface.co/datasets/Magpie-Align/Magpie-Llama-3.1-Pro-MT-300K-Filtered
[9]https://huggingface.co/datasets/Magpie-Align/Magpie-Qwen2.5-Pro-300K-Filtered

Table 1: **Learning rates** used for sparse fine-tuning of the various draft model candidates.

Model	Sparsity	LR
Llama-3.2-1B-Instruct	0.5	1.25e-05
	2:4	1.25e-05
	50% layer-pruned	1.25e-05
	0.66	2.0e-05
	0.75	5.0e-05
Llama-3.2-3B-Instruct	0.5	7.5e-06
	2:4	7.5e-06
	50% layer-pruned	7.5e-06
	0.66	1.25e-05
	0.75	2.0e-05
Qwen-2.5-0.5B-Instruct	0.5	1.25e-05
	2:4	1.25e-05
	50% layer-pruned	1.25e-05
Qwen-2.5-1.5B-Instruct	0.5	1.25e-05
	2:4	1.25e-05
	50% layer-pruned	1.25e-05
Qwen-2.5-3B-Instruct	0.5	7.5e-06
	2:4	7.5e-06
	50% layer-pruned	7.5e-06

$$D^l = \frac{\sum_{i=1}^{C_{out}} \sum_{j=1}^{C_{in}} \mathbf{1}_\zeta(A_{i,j}^l)}{C_{in} C_{out}}. \tag{8}$$

In the above, $\bar{A}^l$ is the mean of A^l, M is a hyperparameter which defines the magnitude of outliers compared to the average activation, and $\mathbf{1}_\zeta$ is the indicator function defined as

$$\mathbf{1}_\zeta = \begin{cases} 1 & A_{i,j} > M \cdot \bar{A}^l \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

For our experimental results, we use the default hyperparameters setting $M = 5$ and $\lambda = 0.08$.

Our *angular distance layer-wise distribution* is based on the intuition that parameter allocation to decoder blocks should be based on the propensity of each block to modify the residual stream. In essence, decoder blocks which greatly modify their inputs should be allocated more parameters compared to blocks which only slightly modify their inputs.

Formally, for a decoder with n layers, target global decoder sparsity $S \in [0,1]$, and angular distances $\mathbf{D} \in [D^1,...,D^N]$ where D^i is defined by Equation (6) with $n=1$, we allocate the sparsity of each block as

$$S^i = 1 - \left((1-S) * \frac{D^i}{\sum_{j=1}^n |D^j|} \right). \tag{10}$$

G   Benchmarking details

Benchmarking was completed on a 4 × NVIDIA A6000 GPU node with 88 Intel Xeon E5-2699 v4 CPUs using nm-vLLM (Neural Magic, 2024) version 0.5.0. The latency of our draft and target models is measured using a single request of 300 input tokens and 256 output tokens, the average token counts from the SpecBench benchmark. We repeat the request 10 times to minimize variance in the measurement. The cost factor, c, is calculated as the ratio of the draft and target model

(a) Llama-3.2-1B-Instruct

(b) Llama-3.2-3B-Instruct

(c) Llama-3.2-8B-Instruct

(d) Llama-3.2-1B-Instruct

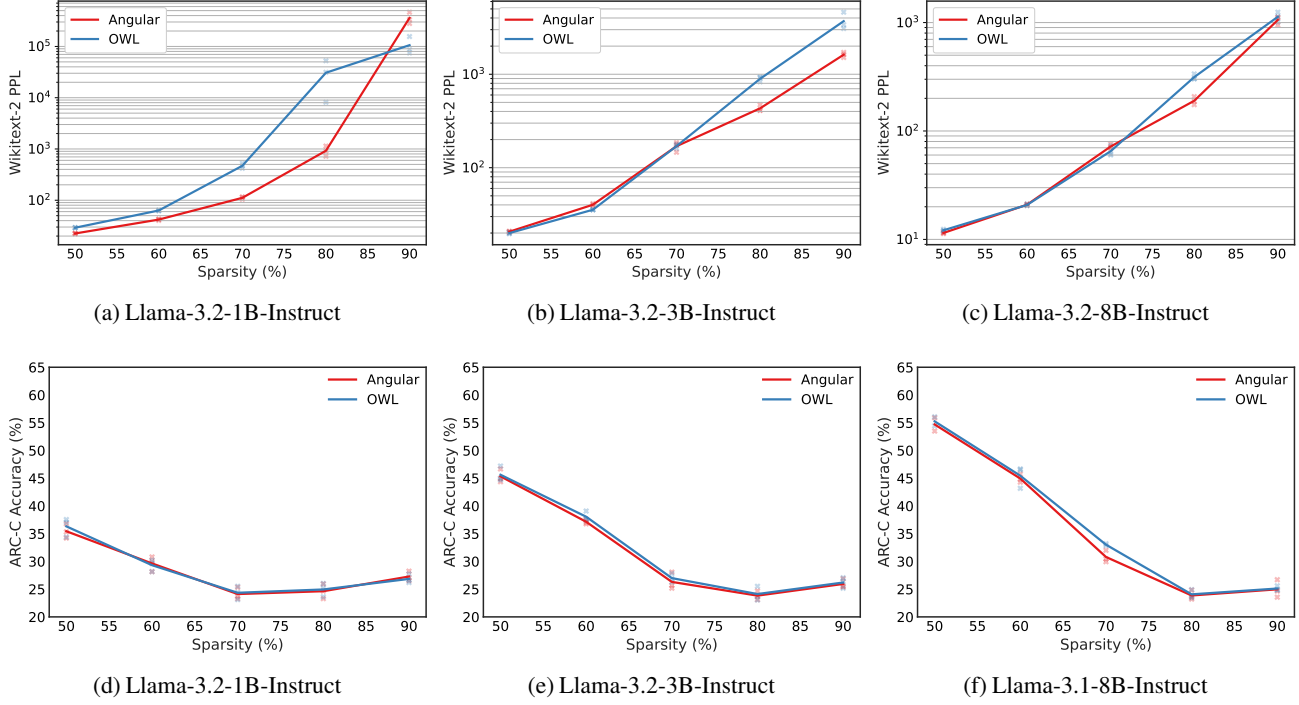(e) Llama-3.2-3B-Instruct

(f) Llama-3.1-8B-Instruct

Figure 3: **Comparison between non-uniform layer-wise sparsity distributions OWL and our proposed angular distance distribution**. In Figures 3a to 3c, we report the WikiText-V2 Perplexity for Llama-3.2 1B and 3B and Llama-3.1 8B at 50, 60, 70, 80, and 90% sparsity. At high sparsities, the angular distance distribution outperforms OWL, particularly for the 1B model. In Figures 3d to 3f, we report the ARC-C accuracy for the same models and sparsities. Despite outperforming OWL in terms of perplexity, we find that OWL achieves slightly higher scores on this benchmark. Further, neither distribution yields statistically significant improvements to acceptance rate in our limited experiments.

wall-time require to process both the input and output tokens. Using MAL from our SpecBench results, we derive the improvement factor per Equation (1). We report the improvement factor and throughput for dense, layer-pruned, and SD^2 unstructured drafters in Figures 1 and 7, respectively. The SD^2 1.5B and 3B drafters outperform both dense and layer-pruned models in terms of end-to-end latency improvements.

In Figure 8 we report the improvement factor for 10 *concurrent* requests following a similar benchmarking approach as per the above. Notably, SD^2 still provides the highest improvement factor for the 3B draft model category, but the 1.5B dense drafter slightly outperforms SD^2 in this setting.

H   Detailed SpecBench results

In this section, we report the detailed SpecBench (Xia et al., 2024) results for draft models fine-tuned with SD^2 (Table 2), one-shot pruned models (Table 3) and SD^2 Qwen-2.5 draft models in the UAG setting (Table 4). Figure 6 depicts the theoretical improvement factor for Qwen-2.5 draft models, in which SD^2 Qwen-2.5 is found to have the highest improvement factor across all candidate draft models.

I   Detailed OpenLLM Leaderboard V1 results

In Tables 5 and 6, we report the benchmark results for our model's after fine-tuning and after one-shot pruning, respectively. In Table 7, we report the benchmark suite results for the Qwen draft models used in our UAG experiments.
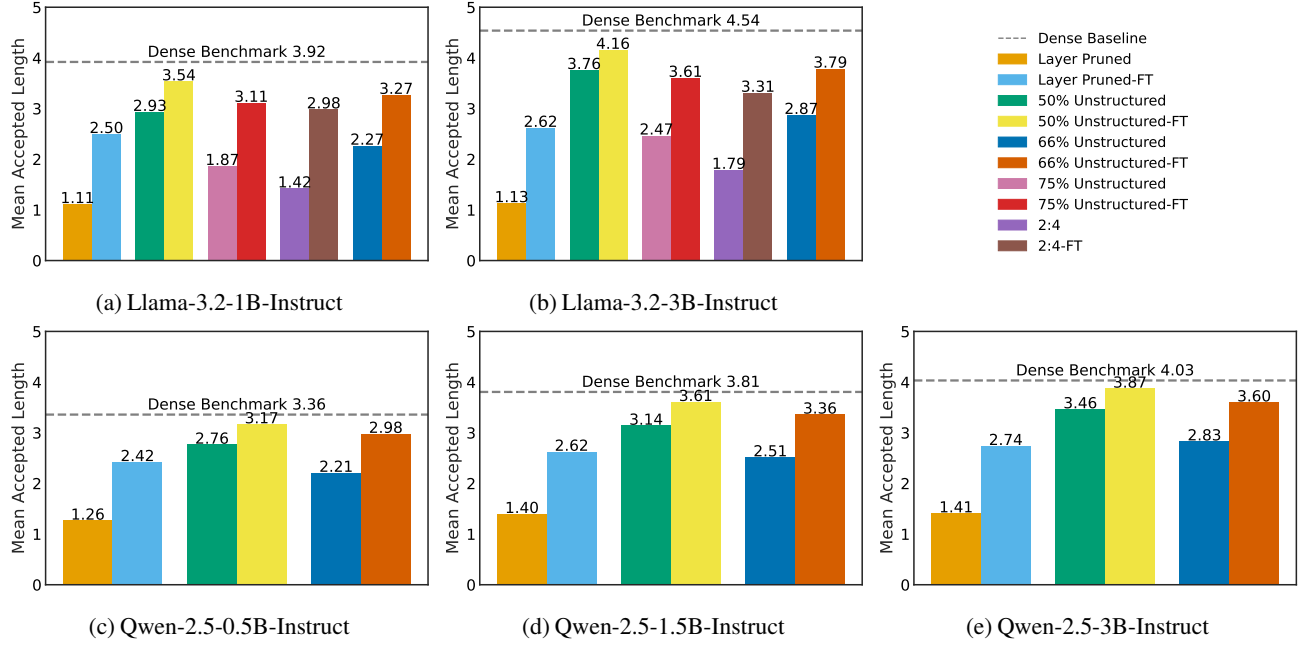
Figure 4: **MAL for Llama-3.2 and Qwen-2.5 model families** on SpecBench (Xia et al., 2024) for layer-pruned, unstructured sparse, and 2:4 sparse draft models including results after one-shot pruning and SD^2 fine-tuning. Dense model baseline are depicted in horizontal grey lines on each plot. **Top row:** Llama-3.2 draft models speculating for a Llama-3.1-70B-Instruct target model. **Bottom row:** Qwen-2.5 draft models speculating for Qwen-2.5-72B-Instruct. Across both model families, we observe that the fine-grained sparse draft models achieve significantly higher MALs than the layer-pruned models.
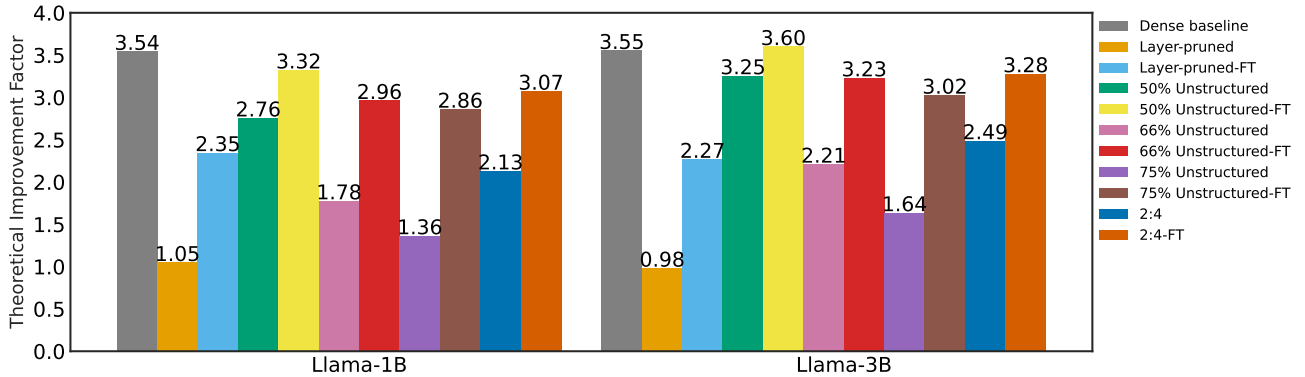
Figure 5: **Theoretical improvement factor for SD^2 Llama-3.2 models** drafting for a Llama-3.1-70B-Instruct based on MAL and MACs. Our SD^2 Llama-3B draft model offers the highest improvement factor compared to dense or layer-pruned models.
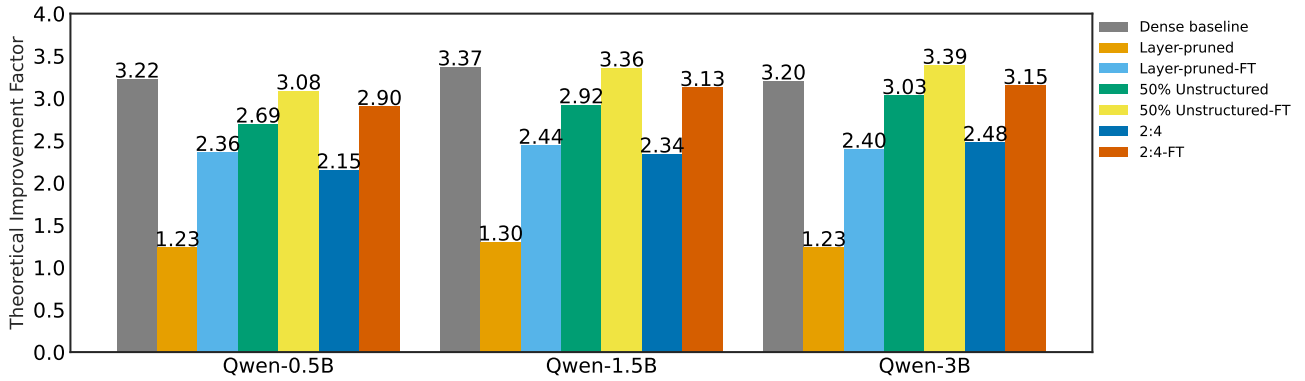
Figure 6: **Theoretical improvement factor for SD^2 Qwen-2.5 models** drafting for Qwen-2.5-72B-Instruct based on MAL and MACs. Similar to our Llama results, SD^2 Qwen-3B offers the highest improvement factor compared to dense or layer-pruned models.
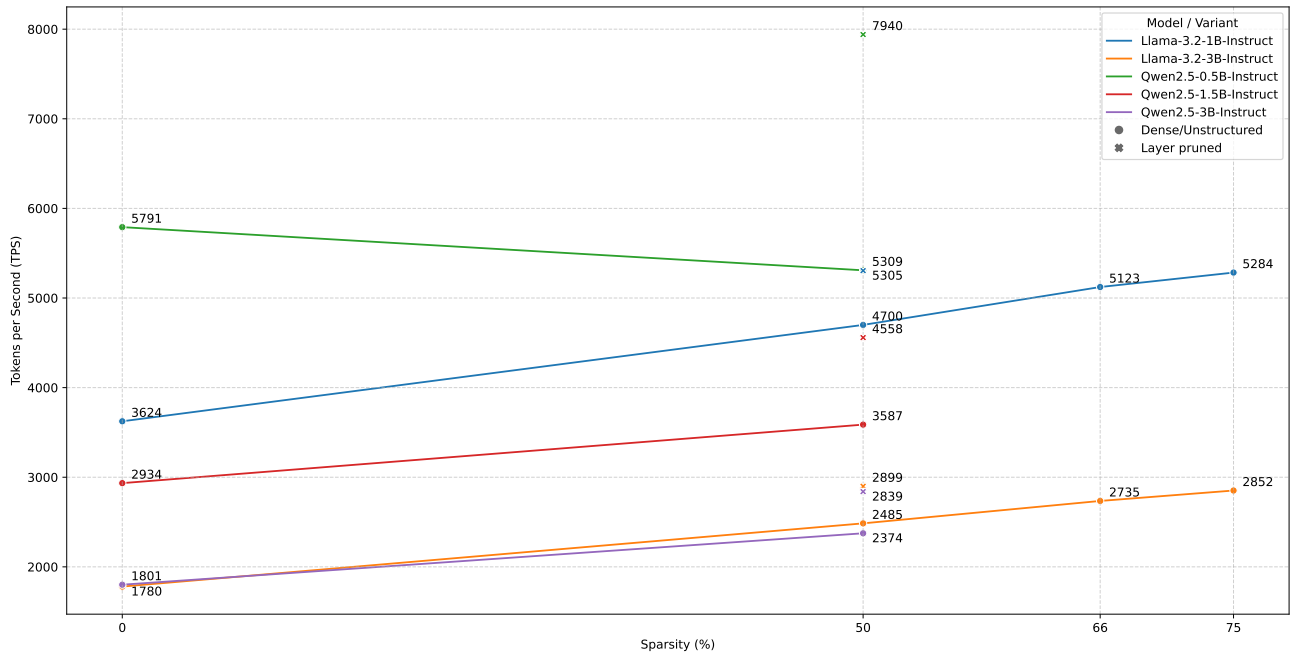
Figure 7: **Throughput (tok/s) vs. sparsity by model and variant.** Layer-pruned draft models offer ultra-low latency; however, their reduced MAL results in an overall lower improvement factor compared to SD^2 unstructured sparse models. All models are benchmarked using nm-vLLM. Throughput values include both input and output tokens.
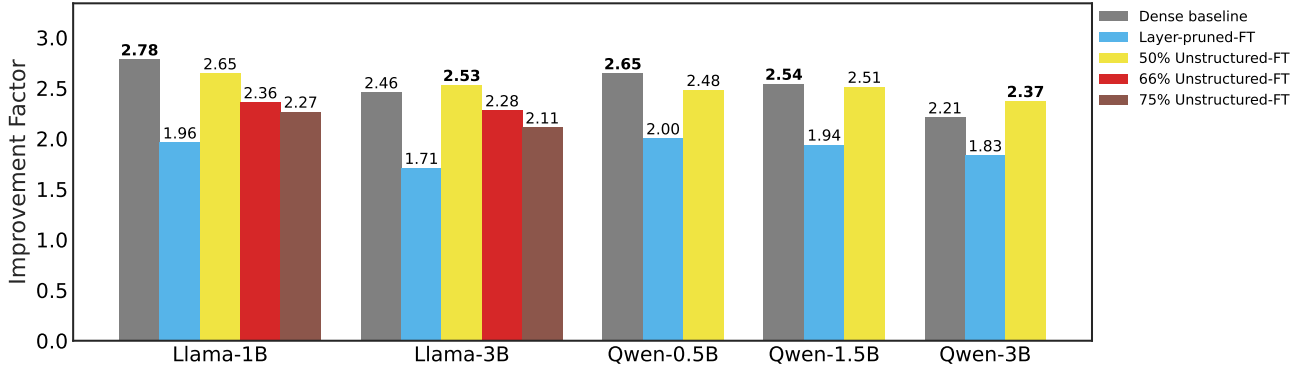
Figure 8: **Improvement factor of dense, layer-pruned, and SD^2 unstructured draft models when benchmarked with 10 concurrent requests**. Llama and Qwen models drafting for Llama-3.1-70B-Instruct and Qwen-2.5-72B-Instruct, respectively. Even in this setting, SD^2 drafters outperform layer-pruned draft models and dense drafters in the 3B model size category.

Table 2: **SpecBench results for SD^2 layer-pruned and fine-grained sparse draft models.**

| Model	Variant	Sparsity %	Overall	MT Bench	Translation	Summarization	QA	Math Reasoning	RAG
	Dense	0	3.92	3.99	2.97	3.43	3.55	5.65	3.66
	Layer-pruned	50	2.50	2.58	1.61	2.11	2.18	3.91	2.49
Llama-1B		50	3.54	3.68	2.70	3.05	3.00	5.22	3.46
	SparseGPT	2:4	3.27	3.39	2.27	2.87	2.71	5.00	3.21
		66	3.11	3.23	2.16	2.73	2.57	4.92	3.01
		75	2.98	3.12	1.97	2.50	2.49	4.84	3.01
	Dense	0	4.54	4.58	3.59	4.12	4.26	5.88	4.27
	Layer-pruned	50	2.62	2.70	1.80	2.26	2.25	4.02	2.65
Llama-3B		50	4.16	4.29	3.30	3.74	3.62	5.60	3.94
	SparseGPT	2:4	3.79	3.91	2.98	3.46	3.15	5.42	3.61
		66	3.61	3.74	2.70	3.21	2.97	5.30	3.55
		75	3.31	3.47	2.27	2.85	2.71	5.14	3.24
	Dense	0	3.36	3.44	2.99	2.81	2.58	5.70	2.85
	Layer-pruned	50	2.42	2.51	1.44	1.91	2.00	3.96	2.09
Qwen-0.5B	SparseGPT	50	3.17	3.28	2.15	2.65	2.45	5.15	2.71
		2:4	2.98	3.09	1.80	2.50	2.32	4.87	2.55
	Dense	0	3.81	3.90	3.72	3.21	2.96	6.02	3.26
	Layer-pruned	50	2.62	2.73	1.59	2.08	2.16	4.16	2.31
Qwen-1.5B	SparseGPT	50	3.61	3.75	2.83	3.07	2.82	5.35	3.13
		2:4	3.36	3.50	2.35	2.84	2.58	5.21	2.90
	Dense	0	4.03	4.17	3.86	3.45	3.15	6.09	3.43
	Layer-pruned	50	2.74	2.87	1.58	2.16	2.24	4.38	2.37
Qwen-3B	SparseGPT	50	3.87	4.04	3.47	3.36	2.97	5.46	3.41
		2:4	3.60	3.73	3.03	3.13	2.76	5.36	3.13

Table 3: **SpecBench results for one-shot pruned layer-pruned and fine-grained sparse draft models.**

Model	Variant	Sparsity %	Overall	MT Bench	Translation	Summarization	QA	Math Reasoning	RAG
Llama-1B	Dense	0	3.92	3.99	2.97	3.43	3.55	5.65	3.66
	Layer-pruned	50	1.11	1.12	1.05	1.09	1.11	1.15	1.11
	SparseGPT	50	2.93	3.00	2.12	2.68	2.42	4.52	2.82
		2:4	2.27	2.26	1.58	2.23	1.91	3.29	2.33
		66	1.87	1.86	1.29	1.84	1.65	2.47	1.85
		75	1.42	1.41	1.14	1.40	1.37	1.64	1.43
Llama–3B	Dense	0	4.54	4.58	3.59	4.12	4.26	5.88	4.27
	Layer-pruned	50	1.13	1.13	1.07	1.10	1.14	1.20	1.12
	SparseGPT	50	3.76	3.82	2.79	3.54	3.17	5.35	3.61
		2:4	2.87	2.85	2.04	2.92	2.32	4.35	2.93
		66	2.47	2.44	1.72	2.51	2.02	3.65	2.54
		75	1.79	1.75	1.27	1.86	1.62	2.23	1.88
	Layer-pruned	50	1.26	1.26	1.17	1.20	1.24	1.38	1.25
	SparseGPT	50	2.76	2.79	2.19	2.51	2.17	4.32	2.41
		2:4	2.21	2.21	1.63	2.18	1.84	2.90	2.04
Qwen-1.5B	Dense	0	3.81	3.90	3.72	3.21	2.96	6.02	3.26
	Layer-pruned	50	1.40	1.39	1.27	1.33	1.35	1.52	1.39
	SparseGPT	50	3.14	3.18	2.84	2.87	2.47	4.69	2.75
		2:4	2.51	2.53	1.96	2.43	2.08	3.41	2.24
Qwen-3B	Dense	0	4.03	4.17	3.86	3.45	3.15	6.09	3.43
	Layer-pruned	50	1.41	1.41	1.26	1.31	1.35	1.56	1.38
	SparseGPT	50	3.46	3.51	3.27	3.13	2.74	5.12	3.04
		2:4	2.83	2.82	2.63	2.78	2.30	3.88	2.54

Table 4: **SpecBench results for SD^2 layer-pruned and fine-grained sparse draft Qwen models drafting for Llama-3.1-70B-Instruct in the UAG setting.**

Model	Variant	Sparsity %	Overall	MT Bench	Translation	Summarization	QA	Math Reasoning	RAG
Qwen-0.5B	Dense	0	2.97	3.09	2.04	2.71	2.48	4.24	2.70
	SparseGPT	50	3.03	3.18	2.04	2.68	2.52	4.44	2.78
		2:4	2.87	3.01	1.77	2.54	2.38	4.35	2.68
Qwen-1.5B	Dense	0	3.34	3.47	2.34	3.09	2.82	4.59	2.97
	SparseGPT	50	3.42	3.58	2.28	3.08	2.92	4.70	3.04
		2:4	3.22	3.40	2.04	2.89	2.69	4.59	2.88
Qwen-3B	Dense	0	3.43	3.58	2.40	3.25	2.91	4.55	2.92
	SparseGPT	50	3.63	3.82	2.39	3.36	3.05	4.83	3.28
		2:4	3.44	3.60	2.28	3.17	2.87	4.73	3.13
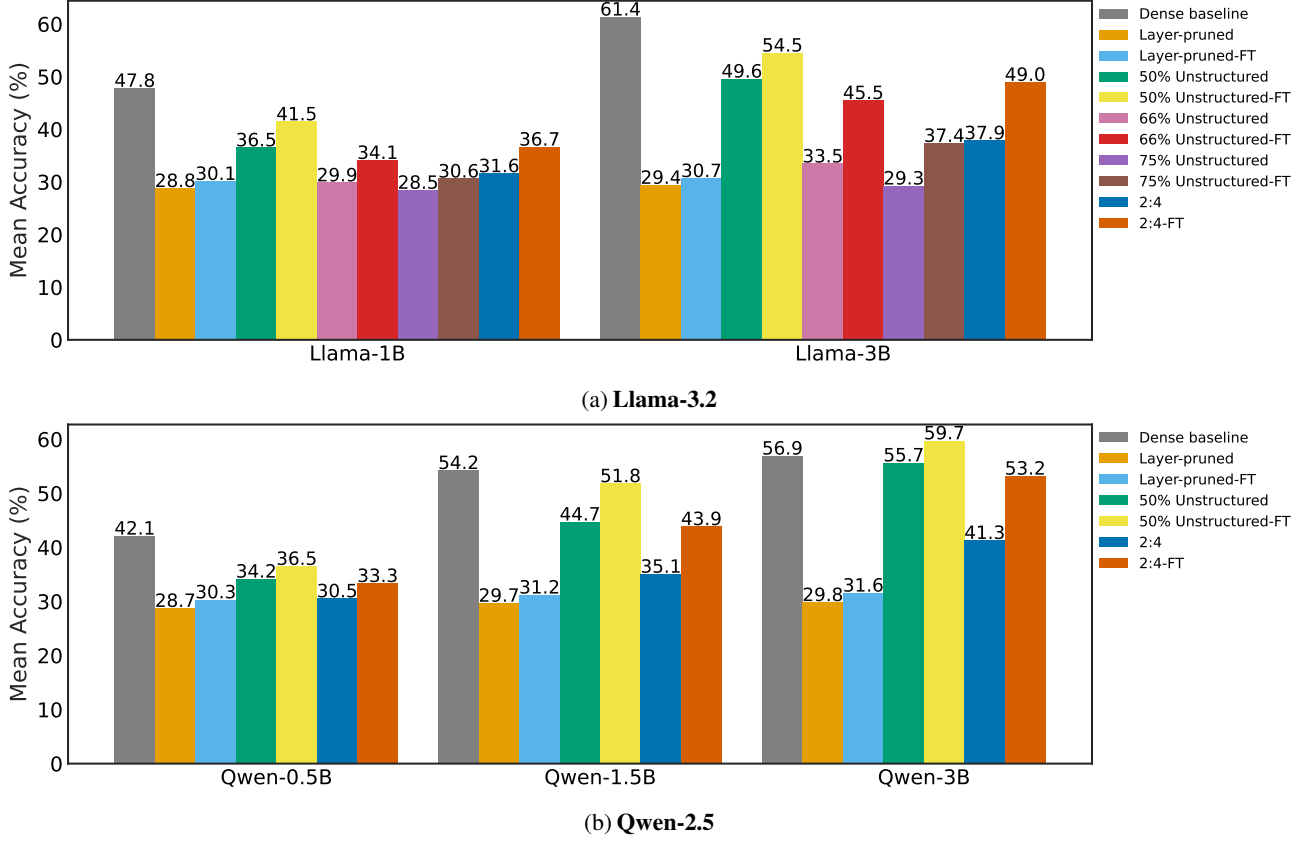
(a) **Llama-3.2**

(b) **Qwen-2.5**

Figure 9: **Average accuracy on OpenLLM Leaderboard V1** benchmarks for dense, layer-pruned, and sparse Llama-3.2 (Figure 9a) and Qwen-2.5 (Figure 9b) draft models. Lighter and darker shades show results after one-shot pruning and SD^2 fine-tuning, respectively. Self-data distillation with sparse fine-tuning enables recovery of accuracy post-pruning. Notably, Qwen-2.5-3B-Instruct at 50% unstructured sparsity *exceeds the accuracy of the original dense model* after preparation with SD^2

Table 5: **SD^2 layer-pruned and fine-grained sparse draft model results on the OpenLLM Leaderboard V1 benchmarks.**

Model	Variant	Sparsity %	ARC-C	GSM8K	Hella-Swag	MMLU	Wino-grande	TruthfulQA MC2	Mean Accuracy
Llama–1B	Dense	0	42.58	33.13	59.70	45.46	62.19	43.85	47.82
	Layer-pruned	50	25.85	0.08	34.74	25.42	52.25	42.50	30.14
	SparseGPT	50	37.03	21.99	52.29	33.40	58.72	45.35	41.46
		2:4	32.34	11.98	46.23	28.36	56.99	44.04	36.66
		66	30.38	7.73	40.94	26.81	55.17	43.72	34.12
		75	25.51	1.59	34.15	26.08	51.07	45.19	30.60
Llama–3B	Dense	0	51.54	64.59	73.09	59.71	69.53	49.72	61.37
	Layer-pruned	50	27.39	0.38	36.91	25.12	50.04	44.22	30.68
	SparseGPT	50	43.60	50.19	65.56	52.81	65.67	49.32	54.53
		2:4	42.75	38.21	59.93	44.50	62.59	45.77	48.96
		66	40.19	29.72	54.70	39.52	60.77	48.12	45.50
		75	33.45	13.50	45.51	32.35	57.46	41.90	37.36
Qwen-0.5B	Dense	0	36.26	21.53	51.40	46.99	54.54	41.95	42.11
	Layer-pruned	50	25.51	0.15	31.88	25.54	51.93	46.94	30.33
	SparseGPT	50	32.25	5.84	44.69	38.33	54.78	43.16	36.51
		2:4	30.03	4.09	39.98	28.24	54.22	43.50	33.35
Qwen-1.5B	Dense	0	53.92	31.31	67.70	60.35	65.59	46.61	54.25
	Layer-pruned	50	27.65	0.76	38.48	25.71	50.91	43.73	31.21
	SparseGPT	50	47.27	42.76	60.79	50.51	61.56	47.98	51.81
		2:4	39.59	26.84	54.44	40.46	58.25	43.64	43.87
Qwen-3B	Dense	0	60.24	10.54	75.18	66.37	70.40	58.76	56.92
	Layer-pruned	50	30.29	1.06	40.53	23.57	50.67	43.29	31.57
	SparseGPT	50	50.34	61.11	68.85	58.27	65.82	54.05	59.74
		2:4	46.93	44.12	62.83	49.68	65.04	50.62	53.20

Table 6: **One-shot pruned layer-pruned and fine-grained sparse draft model results on the OpenLLM Leaderboard V1 benchmarks.**

Model	Variant	Sparsity %	ARC-C	GSM8K	Hella -Swag	MMLU	Wino -grande	TruthfulQA MC2	Mean Accuracy
Llama-1B	Dense	0	42.58	33.13	59.70	45.46	62.19	43.85	47.82
	Layer-pruned	50	26.62	0.00	27.21	22.97	48.54	47.39	28.79
	SparseGPT	50	34.98	4.62	48.50	31.01	59.35	40.69	36.53
		2:4	26.71	1.06	38.15	26.15	55.72	41.85	31.61
		66	24.74	0.99	32.93	26.66	52.49	41.35	29.86
		75	22.53	0.30	27.68	25.45	47.91	46.84	28.45
Llama–3B	Dense	0	51.54	64.59	73.09	59.71	69.53	49.72	61.37
	Layer-pruned	50	25.00	0.00	28.14	24.04	49.41	49.77	29.39
	SparseGPT	50	42.75	31.99	62.44	49.22	63.46	47.66	49.59
		2:4	33.62	5.46	47.78	37.48	58.17	44.72	37.87
		66	28.24	1.44	39.68	30.05	57.38	43.90	33.45
		75	23.98	0.15	29.52	25.61	50.67	45.64	29.26
Qwen-0.5B	Dense	0	36.26	21.53	51.40	46.99	54.54	41.95	42.11
	Layer-pruned	50	21.33	0.00	27.61	22.98	50.83	49.69	28.74
	SparseGPT	50	29.35	3.03	42.74	33.04	53.91	42.85	34.15
		2:4	25.09	1.44	34.61	25.92	52.57	43.50	30.52
Qwen-1.5B	Dense	0	53.92	31.31	67.70	60.35	65.59	46.61	54.25
	Layer-pruned	50	24.06	0.00	32.25	24.37	49.33	48.14	29.69
	SparseGPT	50	43.94	14.25	57.65	48.60	61.01	42.87	44.72
		2:4	31.23	2.65	43.99	35.71	56.04	40.74	35.06
Qwen-3B	Dense	0	60.24	10.54	75.18	66.37	70.40	58.76	56.92
	Layer-pruned	50	23.29	0.08	30.87	24.78	50.51	49.39	29.82
	SparseGPT	50	48.98	43.44	66.22	55.64	66.93	52.77	55.66
		2:4	39.08	8.64	53.25	43.95	60.93	42.01	41.31

Table 7: **OpenLLM Leaderboard V1 benchmarks for SD^2 layer-pruned and fine-grained sparse Qwen-2.5 draft models aligned with a Llama-3.1-70B-Instruct target model for the UAG setting.**

Model	Variant	Sparsity %	ARC-C	GSM8K	Hella -Swag	MMLU	Wino -grande	TruthfulQA MC2	Mean Accuracy
Qwen-0.5B	Layer-pruned	50	26.88	0.00	32.19	25.35	50.12	45.33	29.98
	SparseGPT	50	32.08	6.29	44.83	38.07	55.88	44.56	36.95
		2:4	28.84	4.55	40.50	28.00	54.70	44.34	33.49
Qwen-1.5B	Layer-pruned	50	28.75	0.08	39.21	25.57	52.25	43.67	31.59
	SparseGPT	50	45.48	42.53	60.32	51.30	61.33	46.25	51.20
		2:4	38.65	28.20	53.92	40.09	58.64	43.76	43.88
Qwen-3B	Layer-pruned	50	31.48	0.99	40.55	22.98	50.28	42.17	31.41
	SparseGPT	50	48.98	58.07	67.88	57.61	65.04	49.09	57.78
		2:4	44.71	45.94	61.77	48.72	63.93	48.03	52.18