Scalable LLM Math Reasoning Acceleration with Low-rank Distillation

Harry Dong [1] **Bilge Acun** [2] **Beidi Chen** [1] **Yuejie Chi** [1 2]

Abstract

Due to long generations, LLM math reasoning demands significant computational resources and time. While many existing efficient inference methods have been developed with excellent performance preservation on language tasks, they often severely degrade math performance. We propose Caprese, a resource-efficient distillation method to recover lost capabilities from deploying efficient inference methods, focused primarily in feedforward blocks. With original weights unperturbed, $\sim$1% of additional parameters, and only 20K synthetic training samples, we can recover much if not all of the math capabilities lost from efficient inference for thinking LLMs and without harm to language tasks for instruct LLMs. Moreover, Caprese slashes the number of active parameters ($\sim$2B cut for Gemma 2 9B and Llama 3.1 8B) and integrates cleanly into existing model layers to reduce latency (>11% reduction to generate 2048 tokens with Qwen 2.5 14B) while encouraging response brevity.

1. Introduction

With the increasing capabilities of large language models (LLMs) (Vaswani et al., 2017), evaluations are also becoming increasingly sophisticated, typically involving multi-step reasoning such as in math problem solving. These tasks tend to demand long generation which drives up latency, making efficiency a dire issue. Fortunately, many efficient LLM inference algorithms have shown great promise, slashing expensive computational bottlenecks with little damage to the original performance on a variety of language-based tasks like reading comprehension and summarization. *However, for math reasoning tasks, many efficient inference algorithms begin to break down, decimating performance, despite their robustness in language settings.* Thus, there is a need to design efficient inference algorithms that simultaneously maintain language and math capabilities.

A critical difference between math reasoning and many language tasks is the generation length. Reasoning typically involves long generation due to improved performance from chain-of-thought (CoT) (Wei et al., 2022), which often surpasses the length of the input query. However, CoT performance falls apart with efficient algorithms that introduce approximation errors which build up over time. For instance, deploying CATS (Lee et al., 2024), a sparse thresholding method, on Gemma 2 2B (Team et al., 2024) has little impact on language generation performance, but knocks GSM8K (Cobbe et al., 2021) accuracy *from 51.02% to 34.42%* (Table 1). Similarly, for thinking models: applying GRIFFIN (Dong et al., 2024a), an adaptive structured pruning method, on the first half of DeepSeek-R1-Distill-Qwen 1.5B (Guo et al., 2025) drops MATH-500 (Lightman et al., 2023) accuracy *from 79.40% to 42.00%* (Table 2). As generation is much more demanding computationally than prefill, there is a dire need to make long reasoning CoTs efficient, especially with the rise of test-time scaling. This poses two key inference challenges with math reasoning:

1. *Instability*: Long CoTs can be sensitive to mistakes. Even single token mistakes can sometimes drive the generation trajectory off course, leading to an incorrect response (Zhou et al., 2024). Hence, efficient inference algorithms should respect the delicacy of CoT.
2. *Inefficiency*: Due to the reliance on CoT, which tend to lead to long generation, the already computationally expensive LLM autoregressive decoding process is accentuated. This problem is further exaggerated as CoTs scale in length (Guo et al., 2025) and quantity (Brown et al., 2024; Wu et al., 2024; Snell et al., 2024) for better performance.

An ideal method should be efficient, performance preserving, and easy to integrate. Thankfully, low-rank structure in the feedforward (FF) output features can help make this possible. We choose to focus on FF blocks since, contributing $\sim$2/3 of an LLM's parameters, detrimental in latency-sensitive applications with small batch sizes such as in personal devices and robotics. Because of the success of works that exploit contextual sparsity in FF blocks on language

[1]Department of Electrical and Computer Engineering, Carnegie Mellon University, USA [2]FAIR at Meta. Correspondence to: Harry Dong <harryd@andrew.cmu.edu>.

Proceedings of the 2^{nd} Workshop on Long-Context Foundation Models, Vancouver, Canada. 2025.

tasks for efficiency, like CATS and GRIFFIN, we peek into the residuals of FF sparsity-based efficient methods. Using an oracle top-k filter on FF nonlinearity output magnitudes, we observe huge reductions in error with a low-rank approximation to the FF output residuals (Figure 1). Since FF intermediate feature sizes can be on the order of 10^5, adding 256 for a low-rank approximation is comparatively mundane. *This observation motivates us to estimate the residual from sparse FF methods with low-rank layers.*

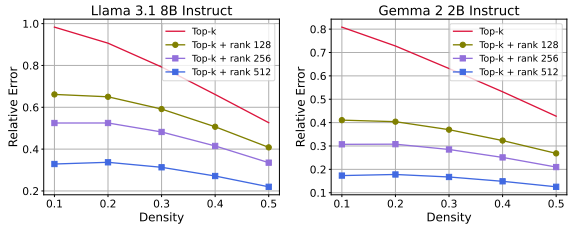

Figure 1: Average relative FF output error of generated tokens with varying top-k densities and low-rank approximations. The density is the fraction of non-zero intermediate FF features maintained by top-k. *A relatively small low-rank approximation to the top-k residual dramatically reduces error much more effectively than just increasing density.*

We introduce **Caprese** (**CAP**ability **RE**covery with **S**calable **E**fficiency) to learn FF residuals from *any* sparse FF algorithm using small low-rank linear layers for improved performance (Figure 2). Through distillation of math knowledge into these low-rank layers, Caprese is able to overcome both challenges of math reasoning and makes significant progress towards an ideal efficient method:

1. **Performance Enhancement:** Demonstrated across instruct and thinking models, Caprese *recovers much if not all of the math performance lost from deploying a sparse FF algorithm without harming language tasks, even with different techniques of test-time scaling*. For example, scaling generations with Caprese on Llama 3.2 3B Instruct improves Pass@100 by 7.0% from the original model while using 15.8% less compute.
2. **Efficiency:** Due to the ability to combine our low-rank layers with existing FF layers, Caprese reduces generation latency. For instance, Caprese reduces latency by 11.7% when generating 2048 tokens using DeepSeek-R1-Distill-Qwen 14B with GRIFFIN.
3. **Low Budget:** We can see significant gains in math performance by training Caprese layers on only 20K synthetic math samples. Moreover, with a low-rank layer size of only 256, this equates to adding roughly 0.8% additional parameters into Llama 3.1 8B and Gemma 2 9B, dwarfed by savings in active parameters ($\sim$2B cut for both models with Caprese (CATS)).

The remainder of this paper is organized as follows. Section 2 describes two efficient sparse FF algorithms that serve as baselines. Section 3 details Caprese's methodology. Then in Section 4, we showcase the strong performance of Caprese on instruct LLMs, scaling generation outputs, scaling generation length, and efficiency. Although we focus on math in this paper, Caprese is not restricted to math tasks.

2. Adaptively Sparse FF Methods

In this section, we provide descriptions of GRIFFIN and CATS which we use as baselines and backbones to Caprese. Let $\boldsymbol{X} \in \mathbb{R}^{S\times D}$ be the input into the FF block during prefill with sequence length S and feature size D. Define the FF block as $\text{FF}(\boldsymbol{X}) = \text{FF}_2(\text{FF}_1(\boldsymbol{X}))$ such that

$$\boldsymbol{Z} = \text{FF}_1(\boldsymbol{X}) = \sigma(\boldsymbol{X}\boldsymbol{W}_g) \odot \boldsymbol{X}\boldsymbol{W}_1, \tag{1}$$

$$\text{FF}_2(\boldsymbol{Z}) = \boldsymbol{Z}\boldsymbol{W}_2, \tag{2}$$

where $\boldsymbol{W}_g, \boldsymbol{W}_1 \in \mathbb{R}^{D\times D_{\text{FF}}}$, $\boldsymbol{W}_2 \in \mathbb{R}^{D_{\text{FF}}\times D}$, σ is a nonlinear function, $\odot$ is an element-wise multiplication operator, and $D_{\text{FF}} \gg D$ is the FF intermediate feature size.

GRIFFIN. GRIFFIN (Dong et al., 2024a) adaptively prunes columns and rows in the FF block weights per prefill. GRIFFIN finds $[\overline{\boldsymbol{Z}}]_i = |[\boldsymbol{Z}]_i|/\|[\boldsymbol{Z}]_i\|_2$ for each token index i, followed by an aggregation across the FF feature axis: $[\boldsymbol{s}]_j = \|[\overline{\boldsymbol{Z}}]_{\cdot,j}\|_2$. The result $\boldsymbol{s} \in \mathbb{R}^{D_{\text{FF}}}$ gives a metric to perform top-k selection across corresponding columns of $\boldsymbol{W}_g$ and $\boldsymbol{W}_1$, and rows of $\boldsymbol{W}_2$ to produce $\widehat{\boldsymbol{W}}_g, \widehat{\boldsymbol{W}}_1 \in \mathbb{R}^{D\times k}$, and $\widehat{\boldsymbol{W}}_2 \in \mathbb{R}^{k\times D}$. Then, for the generation phase, the following FF block is used for input $\boldsymbol{x} \in \mathbb{R}^D$:

$$\widehat{\boldsymbol{z}} = \widehat{\text{FF}}_1(\boldsymbol{x}) = \sigma(\boldsymbol{x}\widehat{\boldsymbol{W}}_g) \odot \boldsymbol{x}\widehat{\boldsymbol{W}}_1, \tag{3}$$

$$\widehat{\text{FF}}_2(\widehat{\boldsymbol{z}}) = \widehat{\boldsymbol{z}}\widehat{\boldsymbol{W}}_2. \tag{4}$$

CATS. CATS (Lee et al., 2024) uses hard thresholding to skip computation in part of the FF block. Letting T_τ be the hard thresholding function with threshold τ,

$$\widehat{\boldsymbol{z}} = \widehat{\text{FF}}_1(\boldsymbol{x}) = T_\tau(\sigma(\boldsymbol{x}\boldsymbol{W}_g)) \odot \boldsymbol{x}\boldsymbol{W}_1, \tag{5}$$

$$\widehat{\text{FF}}_2(\widehat{\boldsymbol{z}}) = \widehat{\boldsymbol{z}}\boldsymbol{W}_2. \tag{6}$$

$\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ should be sparsified into $\widehat{\boldsymbol{W}}_1$ and $\widehat{\boldsymbol{W}}_2$, respectively, based on the non-zero entries of $T_\tau(\sigma(\boldsymbol{x}\boldsymbol{W}_g))$ for latency improvement, which can vary from token to token and require a custom kernel for wall clock speed-up. We calibrate τ based on prefill features and only threshold during the generation phase, analogous to GRIFFIN.

3. Method: Caprese

Motivated by the low-rank structure of residuals in Figure 1, we introduce Caprese which distills approximation errors in embeddings into low-rank linear layers in FF blocks. See Figure 2 for an illustration of our method.

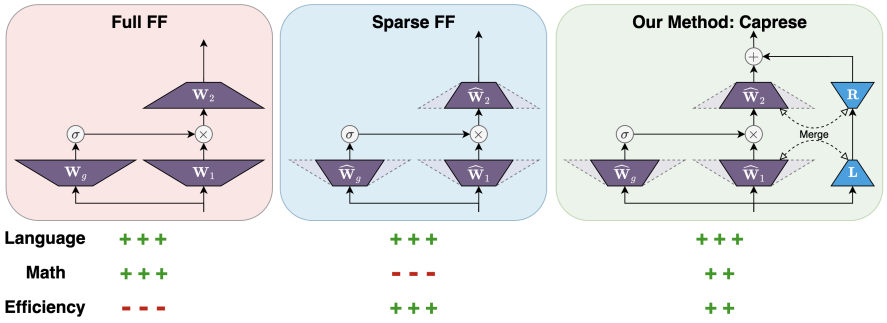

Figure 2: A full FF block will contain the complete performance of the original model without any benefit to efficiency. Sparse FF algorithms can be very efficient by using subsets of the FF block but harm math performance. Our method, Caprese, uses a sparse FF algorithm and a small distilled low-rank linear layer, which can be merged with existing FF weights, for highly performative inference in language and math settings while being efficient.

Inference efficiency algorithms often introduce feature approximation errors in favor of faster generation, which we mitigate with distillation. Let the efficient and approximate FF block be $\widehat{\text{FF}}(\boldsymbol{x}) = \widehat{\text{FF}}_2(\widehat{\text{FF}}_1(\boldsymbol{x}))$. In our design of Caprese, we do not constrain $\widehat{\text{FF}}$ to be any specific method or architecture. For instance, $\widehat{\text{FF}}$ could be an FF block with GRIFFIN or CATS. Then, the error is $\|\text{FF}(\boldsymbol{x}) - \widehat{\text{FF}}(\boldsymbol{x})\|_2^2$. We choose to reduce this residual with a low-rank linear layer, meaning we want to solve

$$\min_{\boldsymbol{L},\boldsymbol{R}} \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{x}\in\mathcal{X}} \|\text{FF}(\boldsymbol{x}) - \widehat{\text{FF}}(\boldsymbol{x}) - \boldsymbol{x}\boldsymbol{L}\boldsymbol{R}\|_2^2 \tag{7}$$

for input set $\mathcal{X}$, $\boldsymbol{L} \in \mathbb{R}^{D\times r}$, and $\boldsymbol{R} \in \mathbb{R}^{r\times D}$ where $r \ll D_{\text{FF}}$. The optimal solution can be computed analytically since this is a reduced rank regression problem, but the size of $|\mathcal{X}|$ and D may make it prohibitively expensive. Therefore, we opt to learn $\boldsymbol{L}$ and $\boldsymbol{R}$ independently for every FF block (i.e., previous FF blocks are assumed to be from the original model), allowing for parallel layer-wise (LW) training, taking inspiration from Dong et al. (2024b). Each $\boldsymbol{R}$ is initialized as a zero matrix since the efficient approximation is assumed to be of good quality, and original model weights are frozen, similar to LoRA (Hu et al., 2022).

We also distill end-to-end (E2E) to further improve the performance. Using the learned low-rank layers as an initialization, we put them all together to distill the final model embedding into the efficient model, again using MSE:

$$\min_{(\boldsymbol{L}_i,\boldsymbol{R}_i)_{i=1,\dots,L}} \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{x}\in\mathcal{X}} \|\text{M}(\boldsymbol{x}) - \text{M}_{\text{student}}(\boldsymbol{x})\|_2^2 \tag{8}$$

where M and $\text{M}_{\text{student}}$ are the original L-layered LLM and LLM with Caprese layers, respectively, excluding the final linear head. Again, only each FF block's $\boldsymbol{L}$ and $\boldsymbol{R}$ are tuned.

4. Experiments

We showcase the effectiveness of Caprese at recovering much, if not all, of the math performance lost from efficient inference algorithms without sacrificing efficiency or performance on language tasks. When ambiguous, we denote Caprese (CATS) to be our method with CATS as the underlying sparse method and similarly for GRIFFIN. Otherwise, we use "Caprese" for brevity. Unless specified, the FF intermediate feature sparsity is set at 50% and $r = 256$. *For a more meaningful baseline, we only apply GRIFFIN to the first half of the model for all experiments since we observed much steeper drops in math performance if used for all layers. Regardless, Caprese still improves the performance of GRIFFIN on all layers. CATS is applied to all layers.* Additional experiments are found in Appendix J.

Table 1: Instruct models' 0-shot accuracies on mathematical reasoning (GSM8K and MATH) and language generation tasks (CoQA and QASPER). More results in Table 7.

MODEL	GSM8K	MATH	COQA	QASP.
Llama 3.2 3B IT	51.55	14.32	63.95	12.45
GRIFFIN	28.96	10.98	64.52	**12.52**
LW CAPRESE	40.18	13.70	64.33	11.60
E2E CAPRESE	**44.66**	**16.96**	**64.83**	12.35
CATS	41.24	12.04	58.87	11.36
LW CAPRESE	45.49	13.62	60.53	12.26
E2E CAPRESE	**46.85**	**14.40**	**61.72**	**12.36**
Gemma 2 2B IT	51.02	16.06	63.77	10.96
GRIFFIN	33.74	11.32	63.28	**11.07**
LW CAPRESE	42.53	12.32	**63.77**	10.75
E2E CAPRESE	**48.14**	**13.70**	63.37	11.05
CATS	34.42	10.56	61.53	10.11
LW CAPRESE	**46.32**	13.90	61.92	10.82
E2E CAPRESE	46.17	**14.16**	**63.92**	**11.03**

Instruct Models. We show Caprese is able to preserve math performance without sacrificing performance on pure language tasks like question answering with instruct LLMs. We test Llama 3 (Dubey et al., 2024) and Gemma 2 (Team et al., 2024) models on zero-shot GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), CoQA (Reddy et al., 2019), and QASPER (Dasigi et al., 2021), using LM Evaluation Harness (Gao et al., 2023) and CoT prompts for math.

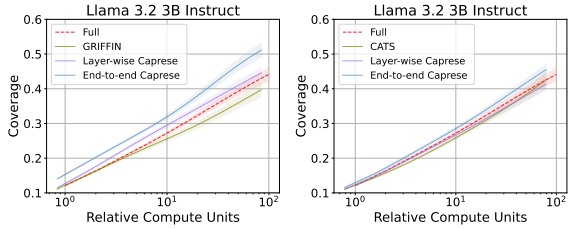

Figure 3: Coverage and standard deviation of 140 samples from MATH as the number of generation attempts, N, scales. We define Relative Compute Units $= N \times A$ where A is the fraction of total parameters activated per input. More results in Figure 9.

Table 1 shows that *Caprese is able to preserve most if not all of the math capabilities in the original models without damaging performance on the language tasks, despite the distillation dataset being all math.* In most cases, CATS and GRIFFIN severely harm GSM8K and MATH accuracy, but Caprese is able to effectively recover the lost performance. In addition, Caprese's performance is consistent at different sparsity levels and r (Appendix B). Best performers in CoQA and QASPER are a toss-up, but all methods have little impact on the accuracy of these tasks (the main purpose of these tasks is to show no degradation in language tasks).

Scaling Best-of-N. Now, we see the generalizability of Caprese on the first axis of test-time scaling: sampling multiple responses and selecting the best one, known as Best-of-N (BoN). To evaluate, we find the coverage (Pass@K) on 140 samples from MATH. We use an oracle verifier to accurately assess the quality of the pool of generated responses. To combat the high variance when K is close to N, we calculate the average coverage for $K = 1, \ldots, 100$ across 10 independent pools of 100 generations for each sample. *Factoring in the saved compute, E2E Caprese is able to have similar or better coverage scaling compared to the original model*, as shown in Figure 3. Notably, E2E Caprese (GRIFFIN) on Llama 3.2 3B Instruct improves Pass@100 by 7.0% from the original model with 15.8% less compute.

Scaling Length. Thinking models provide another axis of test-time scaling by augmenting the CoT length before giving a final answer. Because this entails very long generation lengths, it is critical that error accumulation across tokens be minimized. We test on DeepSeek-R1-Distill-Qwen models (Yang et al., 2024; Guo et al., 2025) using configurations from Open R1 (Face, 2025). We evaluate on MATH-500 (Hendrycks et al., 2021; Lightman et al., 2023), AIME 2024, and AMC 2023 for a max generation length of 32768. Due to the small number of problems in AIME 2024 and AMC 2023, we evaluate these 3 independent times, reporting the average accuracy and sample standard deviation. From Table 2, Caprese is the most performative. In Appendix E, we show a way to recover even more with reselection.

Table 2: Thinking models' 0-shot performance. More results in Table 8. Further improvements with reselection are shown in Table 4.

MODEL	MATH-500	AMC 2023
DeepSeek-R1-Distill-Qwen 1.5B	79.40	57.50 ± 5.00
GRIFFIN	42.00	16.67 ± 1.44
LW CAPRESE	47.20	27.50 ± 2.50
E2E CAPRESE	**60.40**	**35.83** ± 7.22
CATS	72.00	38.33 ± 3.82
LW CAPRESE	73.80	46.67 ± 2.89
E2E CAPRESE	**74.80**	**48.33** ± 3.82
DeepSeek-R1-Distill-Qwen 7B	90.20	76.67 ± 2.89
GRIFFIN	80.60	58.33 ± 3.82
LW CAPRESE	84.80	60.83 ± 1.44
E2E CAPRESE	**85.40**	**70.00** ± 5.00
CATS	89.20	62.50 ± 2.50
LW CAPRESE	87.00	65.83 ± 7.22
E2E CAPRESE	**90.00**	**79.17** ± 5.77

Efficiency. Caprese reduces end-to-end generation latency. Table 3 shows the latencies of different setups and models. Caprese cuts latency by >11% when generating 2048 or fewer tokens from a prompt of 2048 tokens using DeepSeek-R1-Distill-Qwen 14B (Qwen 2.5 14B architecture) or Gemma 2 9B. Latency improvements are smaller with generation length 8192 but still faster than the full model. Moreover, the latency differences between GRIFFIN and Caprese are fairly small, suggesting that Caprese has *minimal overhead*. Metrics were collected on an NVIDIA L40 GPU using BF16 precision and pre-allocated memory for the KV cache. More metrics included in Appendix H.

Table 3: End-to-end generation latency (s) for GRIFFIN and Caprese (GRIFFIN). For the "Setup" column, $P + G$ indicates input and generation lengths of P and G tokens, respectively. Percentages show the reduction in latency.

SETUP	FULL	GRIFFIN	CAPRESE
Qwen 2.5 14B			
2048+256	12.8	11.1 (−13.3%)	11.3 (−11.7%)
2048+2048	113.3	98.6 (−13.0%)	100.1 (−11.7%)
2048+8192	661.9	603.2 (−8.9%)	603.7 (−8.8%)
Gemma 2 9B			
2048+256	9.6	8.4 (−12.5%)	8.5 (−11.5%)
2048+2048	84.4	74.5 (−11.7%)	75.1 (−11.0%)
2048+8192	483.6	444.4 (−8.1%)	445.8 (−7.8%)

5. Conclusion

To combat the inefficiency of the long and brittle generation process associated with math reasoning, we introduce Caprese, a highly performant and efficient method with strong math and language capabilities that is compatible with a broad class of efficient FF algorithms. In the future, it would be interesting to evaluate its benefit in other reasoning domains besides math and explore the use of input-dependent low-rank layers.

Acknowledgments

The work of H. Dong is supported in part by the Wei Shen and Xuehong Zhang Presidential Fellowship at Carnegie Mellon University.

References

Arora, D. and Zanette, A. Training language models to reason efficiently. *arXiv preprint arXiv:2502.04463*, 2025.

Aytes, S. A., Baek, J., and Hwang, S. J. Sketch-of-thought: Efficient llm reasoning with adaptive cognitive-inspired sketching, 2025. URL https://arxiv.org/abs/2503.05179.

Brown, B., Juravsky, J., Ehrlich, R., Clark, R., Le, Q. V., Ré, C., and Mirhoseini, A. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dai, D., Deng, C., Zhao, C., Xu, R., Gao, H., Chen, D., Li, J., Zeng, W., Yu, X., Wu, Y., et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

Dasigi, P., Lo, K., Beltagy, I., Cohan, A., Smith, N. A., and Gardner, M. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*, 2021.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

Dong, H., Chen, B., and Chi, Y. Towards structured sparsity in transformers for efficient inference. In *Workshop on Efficient Systems for Foundation Models@ ICML2023*, 2023.

Dong, H., Chen, B., and Chi, Y. Prompt-prompted adaptive structured pruning for efficient llm generation. In *First Conference on Language Modeling*, 2024a.

Dong, H., Yang, X., Zhang, Z., Wang, Z., Chi, Y., and Chen, B. Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference. In *Forty-first International Conference on Machine Learning*, 2024b.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Face, H. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL https://github.com/huggingface/open-r1.

Frantar, E. and Alistarh, D. Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*, 2023.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Geva, M., Schuster, R., Berant, J., and Levy, O. Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*, 2020.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Lee, D., Lee, J.-Y., Zhang, G., Tiwari, M., and Mirhoseini, A. Cats: Contextually-aware thresholding for sparsity in large language models. *arXiv preprint arXiv:2404.08763*, 2024.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Li, Z., You, C., Bhojanapalli, S., Li, D., Rawat, A. S., Reddi, S. J., Ye, K., Chern, F., Yu, F., Guo, R., et al. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *The Eleventh International Conference on Learning Representations*, 2022.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Re, C., et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023.

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

Manvi, R., Singh, A., and Ermon, S. Adaptive inference-time compute: Llms can predict if they can do better, even mid-generation. *arXiv preprint arXiv:2410.02725*, 2024.

Nayab, S., Rossolini, G., Simoni, M., Saracino, A., Buttazzo, G., Manes, N., and Giacomelli, F. Concise thoughts: Impact of output length on llm reasoning and cost. *arXiv preprint arXiv:2407.19825*, 2024.

Qu, Y., Yang, M. Y., Setlur, A., Tunstall, L., Beeching, E. E., Salakhutdinov, R., and Kumar, A. Optimizing test-time compute via meta reinforcement finetuning. In *Workshop on Reasoning and Planning for Large Language Models*, 2025.

Reddy, S., Chen, D., and Manning, C. D. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019.

Renze, M. and Guven, E. The benefits of a concise chain of thought on problem-solving in large language models. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, pp. 476–483. IEEE, 2024.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Sun, H., Haider, M., Zhang, R., Yang, H., Qiu, J., Yin, M., Wang, M., Bartlett, P., and Zanette, A. Fast best-of-n decoding via speculative rejection. *arXiv preprint arXiv:2410.20290*, 2024.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Team, G., Riviere, M., Pathak, S., Sessa, P. G., Hardin, C., Bhupatiraju, S., Hussenot, L., Mesnard, T., Shahriari, B., Ramé, A., et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Wu, Y., Sun, Z., Li, S., Welleck, S., and Yang, Y. An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

Xu, S., Xie, W., Zhao, L., and He, P. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*, 2025.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Zhang, Z., Lin, Y., Liu, Z., Li, P., Sun, M., and Zhou, J. Moefication: Transformer feed-forward layers are mixtures of experts. *arXiv preprint arXiv:2110.01786*, 2021.

Zhou, Y., Chen, Z., Xu, Z., Lin, V., and Chen, B. Sirius: Contextual sparsity with correction for efficient llms. *arXiv preprint arXiv:2409.03856*, 2024.

A. Related Works

Efficient Inference for FF Blocks. To improve the efficiency of FF blocks in LLMs, various methods leverage existing sparse structures in FF features (Geva et al., 2020; Dettmers et al., 2022; Li et al., 2022; Dong et al., 2023; Liu et al., 2023). Pruning sets a parameter subset to zero to enforce static sparsity (LeCun et al., 1989; Sun et al., 2023; Frantar & Alistarh, 2023; Ma et al., 2023). Mixtures of experts (MoEs) adaptively select predefined parameter subsets in FF blocks, though they tend to require significant fine-tuning or training from scratch (Jacobs et al., 1991; Zhang et al., 2021; Dai et al., 2024; Liu et al., 2023). Similar to MoEs, another line of work aims at exploiting contextual sparsity to dynamically select FF neurons for each input without training. GRIFFIN (Dong et al., 2024a) is a calibration-free method that uses FF activations from the prefill phase to adaptively prune FF neurons for generation. CATS (Lee et al., 2024) uses hard thresholding to skip computation in parts of the FF block with a custom kernel. More details of GRIFFIN and CATS can be found in Section 2.

Shorter CoTs. Several works aim to improve reasoning efficiency by directly reducing the number of generated tokens (Renze & Guven, 2024; Nayab et al., 2024; Arora & Zanette, 2025; Aytes et al., 2025; Xu et al., 2025; Qu et al., 2025). Since we aim to reduce the per-token latency, this line of research to encourage brevity is orthogonal to ours and could be used alongside our method.

Test-time Scaling. To enhance LLM performance, the priority has historically been to scale training with more data and bigger models (Kaplan et al., 2020; Hoffmann et al., 2022). More recently, there is an increasing effort towards test-time scaling to boost performance on difficult tasks like math. Two ways of test-time scaling that have seen great success. First, for a single prompt, multiple responses can be sampled from the LLM (Brown et al., 2024; Snell et al., 2024; Wu et al., 2024; Manvi et al., 2024; Sun et al., 2024). This increases the probability that a desired response lies in the pool of responses. While this method of scaling is highly parallelizable, it also relies on a verifier model to select the best one. Second, CoTs can be lengthened for each response (Guo et al., 2025). By extrapolating the success of CoTs to extreme lengths (e.g., DeepSeek-R1 generates up to 32K tokens per response), the accuracy of the final answer improves significantly, but this method has low parallelizability due to the autoregressive nature of generation.

B. Effect of Rank & Sparsity

Here, we ablate the relationship between varying ranks in Caprese and sparsity levels in CATS. Using Llama 3.2 1B Instruct, we show the test performance of MATH in Figure 4. The same training procedure and data are used as outlined in Section 3. For all ablated ranks, Caprese consistently outperforms pure CATS by a large margin when CATS performs poorly relative to the full model. With a couple of exceptions (perhaps due to randomness in the generation process), greater performance is correlated with higher rank.

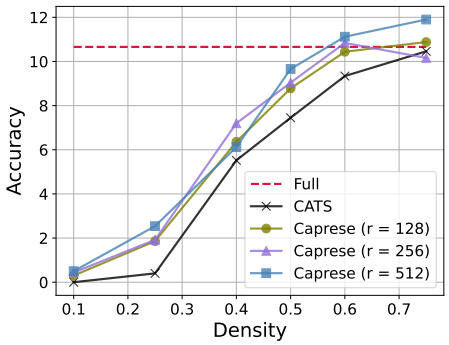

Figure 4: Llama 3.2 1B Instruct's performance on MATH with varying densities of CATS and ranks in Caprese with end-to-end training.

C. Caprese Parameters

Caprese has a tiny parameter footprint. In Figure 5, we see that Caprese adds roughly 1% new parameters relative to the full model with a trend downwards as model size increases.

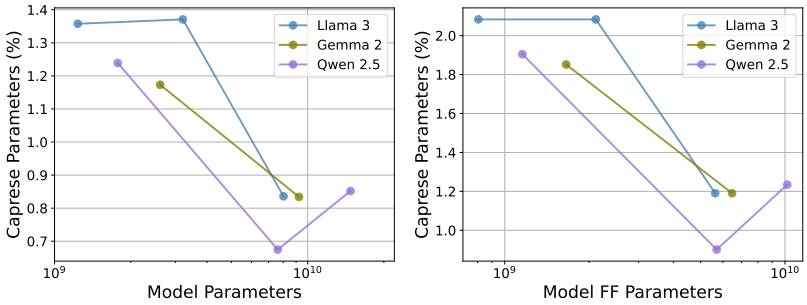

Figure 5: The percent of new parameters that Caprese ($r = 256$) adds relative to the entire model (left) and relative to only the FF parameters (right) for the Llama 3, Gemma 2, and Qwen 2.5 model families.

D. Parallel Inference Computation

The computation with Caprese parameters, $\boldsymbol{xLR}$, can be done in parallel with the original FF operations. In fact, $\boldsymbol{L}$ and $\boldsymbol{R}$ can be concatenated with the up and down projection matrices, respectively. In other words, the Caprese FF block is $\widehat{\text{FF}}^+(\boldsymbol{x}) = \widehat{\text{FF}}_2^+(\widehat{\text{FF}}_1^+(\boldsymbol{x}))$, such that

$$\widehat{\boldsymbol{z}}^+ = \widehat{\text{FF}}_1^+(\boldsymbol{x}) = \left[\sigma(\boldsymbol{x}\widehat{\boldsymbol{W}}_g) \quad \mathbf{1}_r\right] \odot \boldsymbol{x}\left[\widehat{\boldsymbol{W}}_1 \quad \boldsymbol{L}\right], \tag{9}$$

$$\widehat{\text{FF}}_2^+(\widehat{\boldsymbol{z}}^+) = \widehat{\boldsymbol{z}}^+\left[\widehat{\boldsymbol{W}}_2^\top \quad \boldsymbol{R}^\top\right]^\top, \tag{10}$$

where $\mathbf{1}_r$ is a one-vector with length r. In practice, to save memory and time, we do not materialize $\mathbf{1}_r$ but directly assign the product with $\sigma(\boldsymbol{x}\widehat{\boldsymbol{W}}_g)$ to corresponding entries of $\boldsymbol{x}\widehat{\boldsymbol{W}}_1^+$. Recall that the prefill stage still just uses the original model.

E. Enhancing Thinking Models with Reselection

We can push the performance of Caprese with neuron reselection. For a sample, GRIFFIN and CATS calculate metrics ($\boldsymbol{s}$ and τ) to determine subsets of the FF block to use, but these metrics are fixed during the generation phase. After generating many tokens, these selection metrics can benefit from regular updates mid-generation.

For GRIFFIN, updating the metric $\boldsymbol{s}$ entails integrating the FF feature statistics of generated tokens into $\boldsymbol{s}$. While this can be done by re-running prefill on all tokens, there is a more efficient way. This can be done by passing in the generated tokens following the last reselection through the full model. As these tokens propagate through each layer, we find the selection metric for the generated tokens $\boldsymbol{s}_G$ and update corresponding KV pairs. This is similar to verification in speculative decoding (Chen et al., 2023; Leviathan et al., 2023). Since $\boldsymbol{s}$ and $\boldsymbol{s}_G$ are ℓ_2-norms along the token axis, we can define the updated metric as $\sqrt{(\boldsymbol{s} \odot \boldsymbol{s}) + (\boldsymbol{s}_G \odot \boldsymbol{s}_G)}$ and use that to reselect different subsets of the FF block to use. *This setup updates the pruned layers yet avoids prefill for all tokens.* Table 4 shows a clear benefit of reselection (even if infrequent) in Caprese by pushing the performance much closer to the full model.

Table 4: E2E Caprese AMC 2023 accuracies and standard deviations when recalculating the GRIFFIN pruning metric during generation. ρ is the rate at which we reselect pruned neurons in terms of generated tokens. No reselection and the full model are special cases where $\rho = \infty$ and $\rho = 1$, respectively.

MODEL	NO RESELECT	$\rho = 1024$	$\rho = 256$	FULL
DEEPSEEK-R1-DISTILL-QWEN 1.5B	35.83 ± 7.22	**40.83** ± 5.77	40.00 ± 7.50	57.50 ± 5.00
DEEPSEEK-R1-DISTILL-QWEN 7B	70.00 ± 5.00	68.17 ± 1.44	**74.17** ± 5.20	76.67 ± 3.82
DEEPSEEK-R1-DISTILL-QWEN 14B	82.50 ± 6.61	87.50 ± 4.33	**89.17** ± 3.82	90.83 ± 2.89

Reselection is also possible with CATS but requires recomputing prefill for all tokens. The parameter to update is the hard thresholding parameter τ, but because this requires finding the desired percentile of intermediate values in the FF block, we would need to access to all values, which likely cannot be fully stored in memory. Consequently, prefill will need to be redone anytime we want to update τ. Additionally, since τ represents a cutoff for a desired percentile (e.g., median), it is fairly robust to new observations. Given this and the lack of computational incentive, we focus primarily on reselection for GRIFFIN.

F. Natural Response Lengths

Caprese implicitly encourages brevity, often even producing shorter responses than from the full thinking model. Intriguingly, this behavior arises despite any enforcement or regularization on response lengths anywhere during training or inference (except for cutting off generation at 32K tokens). Shown in Figure 6 with MATH-500, the shortest mean response length for all problem difficulties and subjects is either the full model or Caprese. Meanwhile, CATS consistently outputs the longest responses, averaging roughly 1K more across every sample. It is also interesting to note that with increasing problem difficulty, all models and methods naturally allocate more inference tokens towards answering the question. Given CATS and Caprese (CATS) achieve similar MATH-500 accuracies to the full DeepSeek-R1-Distill-Qwen 7B and 14B models, the ability of Caprese to cut down the lengthy responses of CATS down to the response lengths of the original model or shorter without compromising performance is a direct memory and latency benefit.

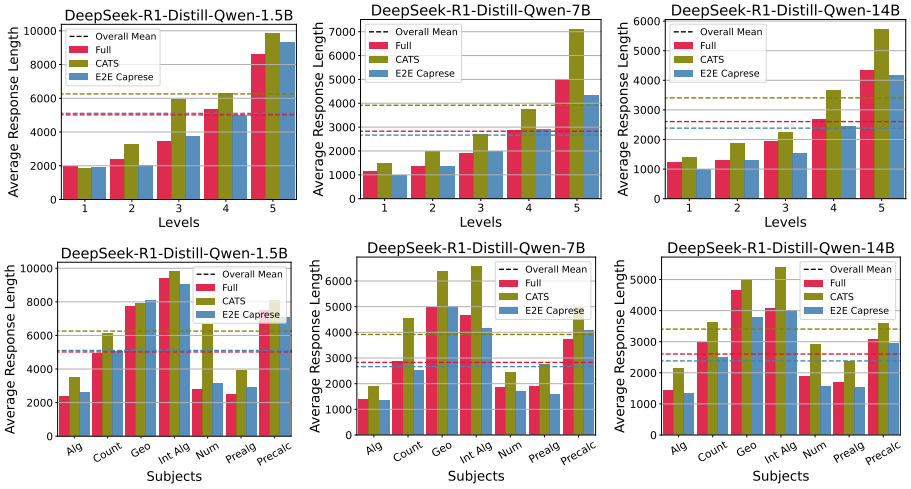

Figure 6: Average number of response tokens for MATH-500 queries across different problem difficulties (top) and subjects (bottom). The subjects are algebra, counting & probability, geometry, intermediate algebra, number theory, prealgebra, and precalculus. The global averages are indicated by the dashed lines. Sparsity is set at 50%.

Complementing Figure 6 for CATS, we show the natural response lengths to MATH-500 samples in Figure 7. Here, we see the similar observations, though the difference between Caprese (GRIFFIN) and the full model is slightly larger but decreasing with model size. Even so, response lengths of Caprese is still significantly shorter than GRIFFIN.

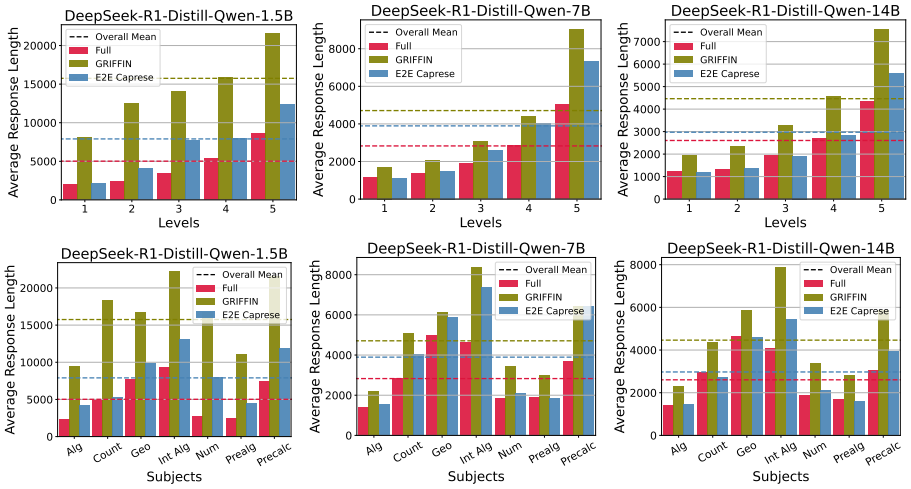

Figure 7: Average number of response tokens for MATH-500 queries across different problem difficulties (top) and subjects (bottom) when using GRIFFIN. The subjects are algebra, counting & probability, geometry, intermediate algebra, number theory, prealgebra, and precalculus. The global averages are indicated by the dashed lines. Sparsity is set at 50%.

G. Robustness to Test-time Density

Trained with a CATS density of 50%, Caprese shows great robustness to deviations from this percentage at test-time, consistently outperforming CATS at across a wide range (Figure 8).

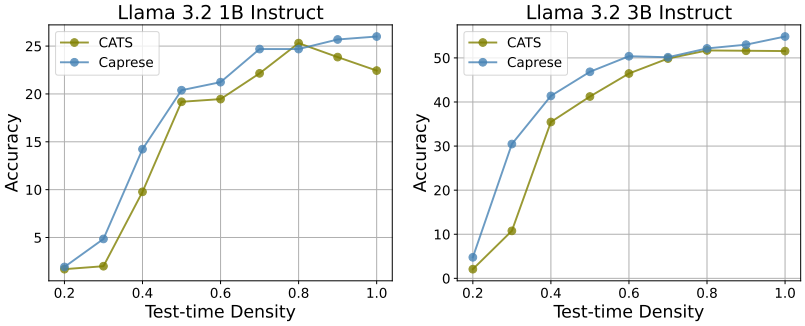

Figure 8: GSM8K performance at different test-time density levels for CATS. The Caprese parameters are trained once at 50% CATS and tested with different densities without fine tuning.

H. Additional Efficiency Metrics

We also report the average time to first token (TTFT) and time to next token (TTNT) in Table 5. The GRIFFIN and Caprese add a small overhead during the prefill phase but generally improve TTNT by 7ms and 5ms for Qwen 2.5 14B and Gemma 2 9B, respectively. There is hardly any difference in TTFT and TTNT between GRIFFIN and Caprese, reinforcing the fact that Caprese adds negligible overhead on top of GRIFFIN.

Table 5: Time to first and next token (s) for GRIFFIN and Caprese (GRIFFIN). For the "Setup" column, $P + G$ indicates input and generation lengths of P and G tokens, respectively. As before, GRIFFIN is applied to the first half of the model, sparsity is set at 50%, and $r = 256$.

Model	Setup	Avg. Time to First Token			Avg. Time to Next Token		
		Full	GRIFFIN	Caprese	Full	GRIFFIN	Caprese
Qwen 2.5 14B	2048+256	0.64	0.67	0.67	0.048	0.041	0.042
	2048+2048	0.67	0.68	0.68	0.055	0.048	0.049
	2048+8192	0.77	0.80	0.80	0.081	0.074	0.074
Gemma 2 9B	2048+256	0.42	0.46	0.46	0.036	0.031	0.032
	2048+2048	0.45	0.49	0.50	0.041	0.036	0.036
	2048+8192	0.50	0.53	0.53	0.059	0.054	0.054

I. Training Details

We set the inner dimension of our low-rank layer to $r = 256$ (to see the effect of different r, see Appendix B). In comparison to the enormous inner dimension of FF layers (e.g., $D_{\text{FF}} = 14336$ for Llama 3.1 8B and Gemma 2 9B), our choice of r is relatively miniscule, adding only $\sim 1\%$ new parameters for all tested models. We use a 20K subset of a common synthetic math training set for training. For a fair comparison, the same subset is used for both layer-wise and E2E distillation for every model. Each training sample is prepended with a CoT instruction: "Please reason step by step." At test time, the actual instructions may be vastly different. Layer-wise and E2E training consists of 20 epochs and 3 epochs, respectively. Training and inference are done in BF16. Table 6 lists the hyperparameter settings for training Caprese layers. The E2E learning rates lie in the interval [4e-6, 2e-4], where larger models tend to learn better with smaller learning rates.

Table 6: Caprese LW and E2E training hyperparameters.

	LAYER-WISE	END-TO-END
OPTIMIZER	ADAM	ADAM
LEARNING RATE	1E-3	[4E-6, 2E-4] (VARIES)
BATCH SIZE	128	16
EPOCHS	20	3
TRAINING SAMPLES	2E5	2E5
SCHEDULER	LINEAR	LINEAR
WARMUP	2%	2%

J. Further Evaluations

This section contains more accuracy evaluations to supplement Section 4.

Table 7: Instruct models' 0-shot accuracies on mathematical reasoning (GSM8K and MATH) and language generation tasks (CoQA and QASPER).

MODEL	GSM8K	MATH	COQA	QASP.
Llama 3.1 8B IT	27.90	13.94	63.88	15.16
GRIFFIN	17.44	6.16	63.37	12.63
LW CAPRESE	27.14	9.72	65.05	14.21
E2E CAPRESE	**40.49**	**12.64**	**65.50**	**15.35**
CATS	40.56	11.80	58.85	12.26
LW CAPRESE	37.68	13.66	63.92	14.34
E2E CAPRESE	**51.86**	**13.88**	**64.27**	**14.42**
Llama 3.2 1B IT	22.44	10.66	55.43	14.43
GRIFFIN	7.13	5.42	56.05	**14.11**
LW CAPRESE	13.72	6.62	56.07	13.40
E2E CAPRESE	**21.00**	**8.44**	**56.55**	13.88
CATS	19.18	7.54	54.40	13.88
LW CAPRESE	18.65	8.28	55.58	**14.35**
E2E CAPRESE	**20.39**	**9.04**	**56.12**	13.75
Llama 3.2 3B IT	51.55	14.32	63.95	12.45
GRIFFIN	28.96	10.98	64.52	**12.52**
LW CAPRESE	40.18	13.70	64.33	11.60
E2E CAPRESE	**44.66**	**16.96**	**64.83**	12.35
CATS	41.24	12.04	58.87	11.36
LW CAPRESE	45.49	13.62	60.53	12.26
E2E CAPRESE	**46.85**	**14.40**	**61.72**	**12.36**
Gemma 2 2B IT	51.02	16.06	63.77	10.96
GRIFFIN	33.74	11.32	63.28	**11.07**
LW CAPRESE	42.53	12.32	**63.77**	10.75
E2E CAPRESE	**48.14**	**13.70**	63.37	11.05
CATS	34.42	10.56	61.53	10.11
LW CAPRESE	**46.32**	13.90	61.92	10.82
E2E CAPRESE	46.17	**14.16**	**63.92**	**11.03**
Gemma 2 9B IT	78.17	27.64	63.78	9.91
GRIFFIN	59.21	25.22	63.82	**10.14**
LW CAPRESE	**76.72**	**25.84**	64.20	9.82
E2E CAPRESE	76.65	25.30	**64.42**	9.92
CATS	76.50	27.32	64.37	9.78
LW CAPRESE	**77.18**	**28.16**	64.52	**10.46**
E2E CAPRESE	**77.18**	28.00	**64.87**	10.02

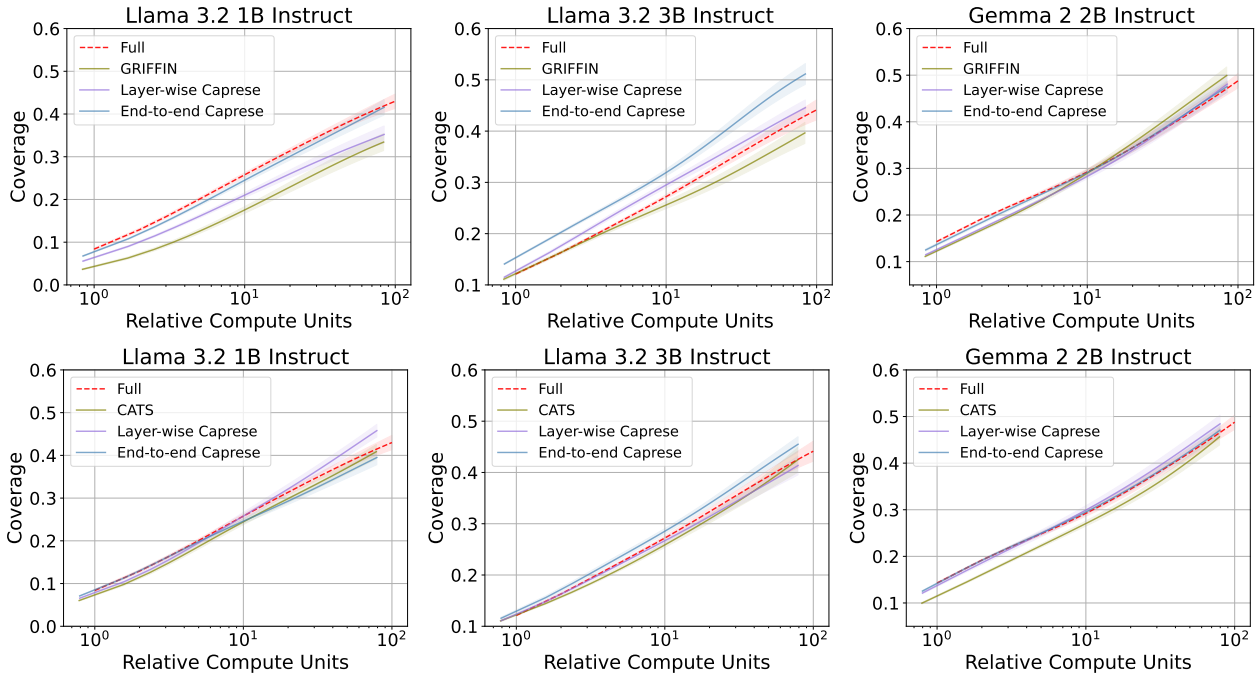

Figure 9: Coverage and standard deviation of 140 samples from MATH as the number of generation attempts, N, scales. We define Relative Compute Units $= N \times A$ where A is the fraction of total parameters activated per input.

Table 8: Thinking models' 0-shot accuracies on math reasoning tasks. Sparsity is set at 50%, and $r = 256$. AIME 2024 and AMC 2023 columns also include the sample standard deviation across 3 runs. Further improvements with reselection are shown in Table 4.

MODEL	MATH-500	AIME 2024	AMC 2023
DeepSeek-R1-Distill-Qwen 1.5B	79.40	31.11 ± 1.92	57.50 ± 5.00
GRIFFIN	42.00	2.22 ± 1.92	16.67 ± 1.44
LW CAPRESE	47.20	3.33 ± 0.00	27.50 ± 2.50
E2E CAPRESE	**60.40**	**6.67** ± 3.33	**35.83** ± 7.22
CATS	72.00	14.44 ± 5.09	38.33 ± 3.82
LW CAPRESE	73.80	16.67 ± 3.33	46.67 ± 2.89
E2E CAPRESE	**74.80**	**21.11** ± 1.92	**48.33** ± 3.82
DeepSeek-R1-Distill-Qwen 7B	90.20	51.11 ± 3.85	76.67 ± 2.89
GRIFFIN	80.60	20.00 ± 3.33	58.33 ± 3.82
LW CAPRESE	84.80	27.78 ± 1.92	60.83 ± 1.44
E2E CAPRESE	**85.40**	**28.89** ± 6.94	**70.00** ± 5.00
CATS	89.20	**37.78** ± 5.77	62.50 ± 2.50
LW CAPRESE	87.00	35.56 ± 5.09	65.83 ± 7.22
E2E CAPRESE	**90.00**	32.22 ± 5.09	**79.17** ± 5.77
DeepSeek-R1-Distill-Qwen 14B	92.80	63.33 ± 3.33	90.83 ± 2.89
GRIFFIN	89.80	30.00 ± 6.67	79.17 ± 2.89
LW CAPRESE	**90.80**	**38.89** ± 1.92	**87.50** ± 5.00
E2E CAPRESE	89.20	37.78 ± 10.72	82.50 ± 6.61
CATS	**92.80**	57.78 ± 1.92	**91.67** ± 5.20
LW CAPRESE	91.00	**60.00** ± 5.77	88.33 ± 1.44
E2E CAPRESE	92.00	58.89 ± 11.71	85.83 ± 3.82