FractalLLM: Lossless Self-Speculative Decoding with Layer Embedded Self-Compression

Anonymous ACL submission

Abstract

Autoregressive decoding in large language models (LLMs) necessitates a full forward pass for each generated token, significantly increasing inference latency. To address this limitation, we propose *Fractal-LLM*, a lossless self-speculative decoding method that embeds a compressed model within selected decoder layers of the original model. Specifically, our approach generates multiple draft tokens in parallel by injecting compressed layers into selected decoder layers. These draft tokens are subsequently verified through a single forward pass of the original model, ensuring the final outputs exactly match those produced by the original model. Experimental results across diverse benchmarks—including GSM8K, XSUM, CNN/DailyMail, and HumanEval—demonstrate that our method achieves substantial inference speed-ups (up to 2.47×) compared to standard autoregressive decoding, without requiring any additional training.

1 Introduction

Recently, large language models (LLMs) have demonstrated remarkable performance across various natural language processing tasks. However, autoregressive decoding requires a full forward pass for each generated token, causing significant latency in practical applications. Speculative decoding methods (Chen et al., 2023; Li et al., 2024) mitigate this issue by proposing multiple draft tokens simultaneously and verifying them with fewer forward passes, thus reducing overall decoding steps. Yet, existing approaches often require either an external draft model (Chen et al., 2023), additional decoding heads integrated within the original model (Cai et al., 2024), or employ self-speculative strategies (Zhang et al., 2024; Liu et al., 2024) that do not fully escape the token-by-token autoregressive decoding constraint.

In this paper, we introduce *Fractal-LLM*, which embeds compressed layers within selected decoder layers, enabling parallel draft token generation in a single forward pass. Generated tokens are subsequently verified by one additional forward pass of the original model, ensuring that outputs precisely match those of standard autoregressive decoding. This approach significantly reduces inference latency without auxiliary models or extra training.

The contributions of this paper are summarized as follows: (1) *Layer Embedded Self-Compression*: We introduce a novel design that embeds a compressed model into selected decoder layers for parallel token generation without removing original layers; (2) *Lossless Self-speculative Decoding*: We generate and verify tokens within a single model in parallel, preserving output quality identical to the base model; (3) *Efficiency Analysis*: We conduct a detailed study of how the number of embedded layers and the size of the draft window jointly affect inference speed-up.

2 Related Work

Speculative Decoding. Early research on speeding up autoregressive generation introduced the idea of speculative decoding, in which a small draft model proposes multiple tokens for each step, and the main (larger) model verifies them in a single forward pass (Chen et al., 2023). This approach can effectively reduce the number of sequential decoding steps, but often requires an additional model that must be trained or at least carefully aligned to the target LLM. Some work, such as MEDUSA (Cai et al., 2024), extends this paradigm by adding multiple look-ahead heads, constructing a tree of candidate continuations in parallel. Recently, EAGLE (Li et al., 2024) further enhanced speculative decoding by introducing feature-level autoregression, which leverages the top-layer outputs of an LLM for predicting token features, and

a context-aware dynamic draft tree for improved token acceptance rates. Ouroboros (Zhao et al., 2024) extends speculative decoding by enabling draft models to generate entire phrases at once, subsequently concatenating high-quality phrases selected during verification to significantly enhance decoding efficiency.

Instead of relying on a second model, recent self-speculative techniques generate drafts by skipping layers or using partial forward passes within the same network, then apply the full model to verify and correct these drafts (Zhang et al., 2024; Liu et al., 2024; Metel et al., 2024). However, these methods commonly retain a strictly token-by-token mechanism, thus limiting potential speedups from token-level parallelism.

Another way to reduce latency is to generate several tokens per iteration. Block-wise or multi-token algorithms (Stern et al., 2018) split the sequence, guess a chunk of tokens in parallel, then verify or refine them before continuing. Look-ahead decoding (Fu et al., 2024) uses a modified attention mask to simultaneously predict multiple future tokens without additional training.

Layer Compression and Quantization. A complementary approach to boosting decoding efficiency is to reduce the size or depth of the network itself. Early work such as Xu et al. (2020) and Fan et al. (2020) demonstrates that partial or probabilistic layer dropping can preserve performance while lowering computational costs. Likewise, quantizing weights and activations often yields faster inference with minimal accuracy loss (Guo et al., 2023). However, these methods neither address the bottleneck imposed by token-by-token decoding nor guarantee preservation of the original model's output quality.

3 Method

3.1 Problem Definition

Speculative decoding generally consists of the following two steps: (1) *Draft Phase*: From the current input sequence $\{x_1, ..., x_p\}$ generate a new set of K draft tokens, forming $\{x_1, ..., x_{p+k}\}$; (2) *Verification Phase*: Subsequently, use the target model (the original LLM) in a single forward pass to verify these draft tokens in parallel. If a certain draft token is found to be incorrect, that position is replaced with the token predicted by the target LLM, and from that token onward, a new round of the draft phase begins.

Suppose that, until the complete sequence is generated, this pair of draft and verify phases repeats a total of N_{phase} times. Following the draft $\rightarrow$ verify phases, if we denote the time spent in each phase as T_{phase}, the overall generation time can be expressed as

$$T_{total} = N_{phase} \times T_{phase} \tag{1}$$

Our goal is to minimize T_{total}, however, this introduces a trade-off between the accuracy and efficiency of the draft process (Zhang et al., 2024). From the point of view of precision, the closer the draft is to the output of the original model, the greater the number of tokens that pass the verification, potentially reducing the N_{phase}. However, from the point of view of efficiency, ensuring that the quality of the draft matches the original model closely can require significant computational cost, thus increasing T_{phase}.

3.2 Layer Embedded Self-Compression

Fractal Layer. The core idea of our method is to compress the model and inject it between the original decoder layers. Concretely, when we designate the i-th decoder layer as a *Fractal Layer*, we attach these compressed layers (including the compressed LM head and embedding) *before* the original layer L_i. This arrangement facilitates multiple draft-token proposals *mid-forward*, leveraging the partial hidden states up to layer $(i-1)$ without having to re-run all preceding layers for each new token.

Draft Phase. We begin by appending w (draft window) draft tokens at the end of the input sequence. These tokens are newly initialized special tokens without any contextual information. Let h_{i-1} be the hidden state from the $(i-1)$-th original layer.

Fractal Layer forwards h_{i-1} through the *upper* compressed layers $(L'_i \dots L'_l)$, yielding logits that range from the prefix's last token to the w newly appended tokens. We then apply the compressed LM head to select the top token (via $\arg\max$) at each of these positions, resulting in $w+1$ new tokens (one from the prefix's final logit, plus w more from the newly appended slots).

Next, we re-embed the updated sequence (consisting of the original prefix plus the newly generated $w+1$ tokens) using the compressed embedding layer. We then pass this re-embedded sequence through the *lower* compressed layers $(L'_1 \dots L'_{i-1})$,

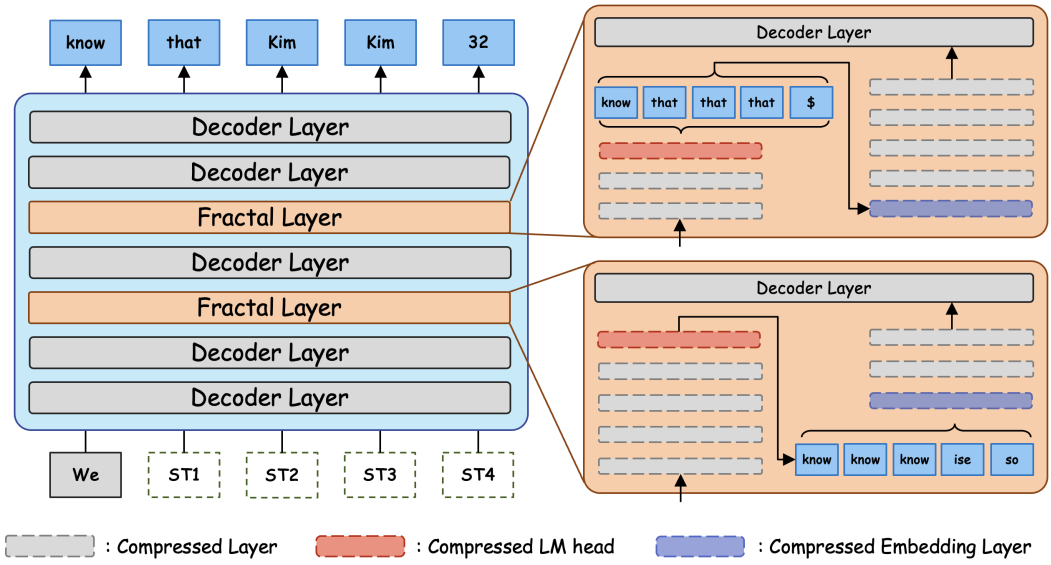

Figure 1: A schematic illustration of our Fractal-LLM approach. The original hidden state passes through the compressed layers, produces provisional logits for draft tokens, re-embeds newly selected tokens, and merges them back into the main sequence.

obtaining an updated hidden state $\hat{h}_{i-1}$. Finally, we merge by replacing only the draft-token components in h_{i-1} with those in $\hat{h}_{i-1}$, retaining the prefix intact. This merged state then feeds into the original layer L_i. Since the prefix's hidden states remain unaltered, the first newly generated token is guaranteed to be aligned with the preceding context. Furthermore, each *Fractal Layer* iteratively refines subsequent draft tokens, inducing the later draft tokens to incorporate updated contextual information. Thus, multiple tokens can simultaneously match the target outputs within a single compressed forward pass, breaking the conventional one-token-per-forward constraint. Fig. 1 provides an overview of our method.

Verify Phase. During verification forward, the injected compressed network is omitted entirely, making this step identical to a standard forward pass of the target model. If we designate n layers in total as *Fractal Layers*, then we can produce $w+n+1$ draft tokens in one forward pass, verifying them against the full model. This guarantees that the final output exactly matches what the original model would produce under vanilla autoregressive decoding.

4 Evaluation

Setup. We benchmark three open Llama checkpoints— Llama-3-3B, Llama-3-8B (Grattafiori et al., 2024), and CodeLlama-13B (Roziere et al., 2023). For each Llama-3-3B, Llama-3-8B we randomly select 300 inputs each from GSM8K (Cobbe et al., 2021), XSUM (Narayan et al., 2018), and CNN/DailyMail (Nallapati et al., 2016). CodeLlama-13B is evaluated on the full HumanEval (Chen et al., 2021) test set. Throughput (tokens / s), relative speed-up, and theoretical FLOPs are measured under identical settings. All experiments were conducted using RTX 3090 GPUs. Detailed implementation and hyperparameters are described in Appendix D.

Results. Table 1 shows that our decoder consistently out-runs the vanilla baseline on all three tasks. On the 3B backbone we reach 14.6 tok/s on GSM8K (↑1.20×) and 8.5 tok/s on XSUM (↑1.48×) with only a 7 % increase in FLOPs, demonstrating that the injected compressed model introduce negligible overhead in total. Our method's effectiveness increases with larger model sizes. For instance, the 8B model achieves a 2.12x speedup on the CNN/DM task, even though its compute budget is approximately 20% larger than the baseline.

We further evaluate our method on the 13B CodeLlama model on HumanEval (Table 2). *Fractal-LLM* achieves a 2.47× throughput improvement, demonstrating effective scaling to larger models and computationally intensive code-generation tasks, while preserving output identical

Model	Method	GSM8K			XSUM			CNN/DM		
		Tok/s	Acc.	FLOPs	Tok/s	Acc.	FLOPs	Tok/s	Acc.	FLOPs
Llama-3 3B	Baseline	12.20	1.00×	$\mathbf{2.5 \times 10^{14}}$	5.74	1.00×	$\mathbf{4.6 \times 10^{14}}$	3.13	1.00×	$\mathbf{7.8 \times 10^{14}}$
	Fractal	**14.60**	**1.20×**	2.7×10^{14}	**8.49**	**1.48×**	5.0×10^{14}	**5.51**	**1.69×**	8.3×10^{14}
Llama-3 8B	Baseline	4.26	1.00×	$\mathbf{7.6 \times 10^{14}}$	2.25	1.00×	$\mathbf{1.1 \times 10^{15}}$	1.01	1.00×	$\mathbf{1.8 \times 10^{15}}$
	Fractal	**7.47**	**1.75×**	9.0×10^{14}	**4.06**	**1.80x**	1.4×10^{15}	**2.15**	**2.12x**	2.2×10^{15}

Table 1: Throughput (tokens / sec), relative acceleration (Acc.), and estimated FLOPs of our decoding method versus the vanila autoregressive baseline.

Model	Method	HumanEval		
		Tok/s	Acc.	FLOPs
CodeLlama-13B	Baseline	1.79	1.00×	4.1×10^{15}
	Fractal	**4.43**	**2.47×**	$\mathbf{4.0 \times 10^{15}}$

Table 2: Throughput (*tokens/s*), relative acceleration (Acc.), and average FLOPs over the first 300 decoding steps on HumanEval.

to baseline autoregressive decoding. At this scale, our method even slightly reduces total FLOPs compared to baseline, as computational savings from fewer full forward passes outweigh overhead from compressed layers.

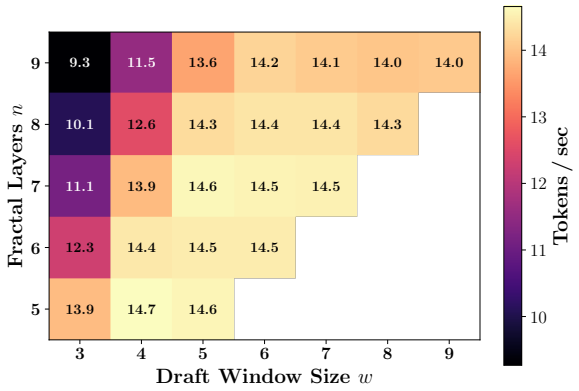

Figure 2: Inference acceleration (tokens/s) as a function of number of layers (n) and the draft window size (w).

Analysis Our method's efficiency depends significantly on two hyperparameters: (1) the number of layers designated as *Fractal Layers* (n), and (2) the number of draft tokens proposed simultaneously (w). To clearly understand their effects, we conducted an analysis using the Llama-3 3B model on a random subset of 100 samples from the GSM8K dataset, across different combinations of these parameters (Figure 2).

As discussed in Eq 1, total generation time (T_{total}) depends on the number of phases (N_{phase}) and the duration per phase (T_{phase}). Increasing Fractal Layers (n) progressively refines draft tokens within each forward pass, enhancing accuracy and reducing N_{phase}, but simultaneously increases internal computations, raising T_{phase}. Conversely, fewer layers limit refinement quality, lowering draft accuracy and thus potentially increasing N_{phase}.

The draft window size (w) determines the number of tokens the model attempts to predict in parallel during each draft phase. Setting w too large introduces a high number of inaccurate tokens, which unnecessarily increases computational overhead through additional attention and FFN computations. Furthermore, incorrect tokens result in more corrections during the verification phase, thereby extending the total decoding time.

Experimental results indicate that setting the draft window size equal to or slightly smaller than the number of *Fractal Layers* achieves an optimal balance between computational overhead and draft accuracy. Additional ablation studies (Appendix A) confirm that this configuration consistently provides stable improvements in both draft token accuracy and overall decoding speed.

5 Conclusion

In this paper, we introduced *Fractal-LLM*, a novel lossless self-speculative decoding framework that integrates a compressed model into selected original decoder layers. Our approach facilitates the parallel generation of multiple draft tokens within a single forward pass, significantly reducing inference latency. Experimental evaluations on diverse benchmarks demonstrated substantial improvements in inference speed (up to 2.47×) while ensuring that outputs remain identical to original decoding. Crucially, our method offers valuable insights by demonstrating that autoregressive decoding need not strictly correspond to one token per model forward pass, enabling the completion of multiple tokens simultaneously.

Limitations

Our proposed method was evaluated under a simplified setting that does not employ Key-Value (KV) caching. In a typical autoregressive decoding process, using KV caching allows reusing intermediate computations from previous tokens, thus speeding up inference. As part of future research, we plan to incorporate cache management of prefix part in practical manner.

Another limitation is that the speed-up from *Fractal Layers* may vary based on their number and positions within the decoder stack (e.g., closer to the input or the output). Therefore we intend to investigate more systematic strategies for selecting *Fractal Layer* positions to maximize efficiency gains.

References

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 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean–Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Rishi Puri, Gretchen Krueger, Mihai Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 13 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, and Others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. 2023. OliVe: Accelerating large language models via hardware–friendly outlier–victim pair quantization. In *Proceedings of the 50th International Symposium on Computer Architecture (ISCA)*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. EAGLE: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. 2024. Kangaroo: Lossless self–speculative decoding via double early exiting. *arXiv preprint arXiv:2404.18911*.

Michael R. Metel, Peng Lu, Boxing Chen, Mehdi Rezagholizadeh, and Ivan Kobyzev. 2024. Draft on the fly: Adaptive self-speculative decoding using cosine similarity. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2267–2272, Miami, Florida, USA. Association for Computational Linguistics.

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807.

Baptiste Roziere, Gautier Izacard, Jan Leike Botha, and Others. 2023. Code LLaMA: Large language models for code. *Preprint*, arXiv:2308.08545.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020. Bert–of–theseus: Compressing bert by progressive module replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2024. Draft & verify: Lossless 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, Bangkok, Thailand. Association for Computational Linguistics.

Weilin Zhao, Yuxiang Huang, Xu Han, Wang Xu, Chaojun Xiao, Xinrong Zhang, Yewei Fang, Kaihuo Zhang, Zhiyuan Liu, and Maosong Sun. 2024. Ouroboros: Generating longer drafts phrase by

phrase for faster speculative decoding. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 13378–13393, Miami, Florida, USA. Association for Computational Linguistics.

A Ablation Study

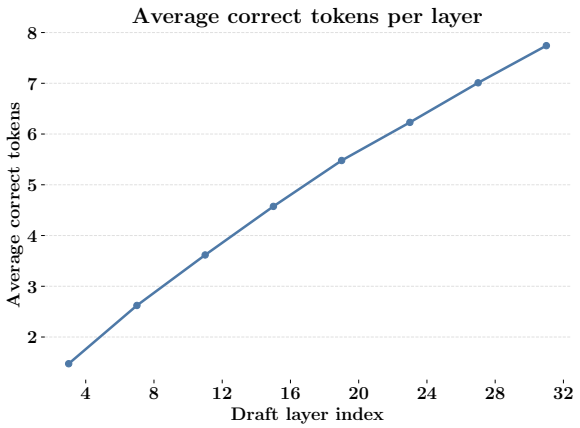

Figure 3: Average correct tokens per layer.

Effect of Fractal Layer Fig 3 analyzes the impact of distributing N Fractal Layers uniformly within a 28-layer decoder architecture. We set $N = 7$, corresponding to one Fractal Layer every four layers, and highlight two critical insights. To validate these observations, we randomly sampled 10 instances from the GSM8K dataset for detailed analysis.

Embedding Fractal Layers *within* the decoder leverages high-quality intermediate representations from earlier full-precision layers. This arrangement consistently yields an average of $N + \delta$ tokens accepted per verification step, with $\delta \approx 0.74$. Specifically, in our setup ($N = 7$), we observed approximately 7.74 tokens accepted per verification pass.

In comparison, employing the same number of layers N in an external draft model theoretically limits the accepted tokens per verification pass to exactly N. Thus, embedding layers internally yields additional tokens per pass without increasing the total number of layers, clearly demonstrating an advantage over external model-based approaches.

B Dataset

To benchmark decoding speed under heterogeneous workloads we employ four public corpora—GSM8K (Cobbe et al., 2021), XSUM (Narayan et al., 2018), CNN/DailyMail (Nallapati et al., 2016), and HumanEval (Chen et al., 2021)—covering arithmetic reasoning, short/long summarization, and code generation.

- **GSM8K**: Grade-school math problems, consisting of approximately 7.5K training examples and 1K test examples; throughput only, max 256 generated tokens.
- **XSUM**: BBC news summarization dataset containing approximately 204K training, 11K validation, and 11K test samples; input truncated to 512 tokens, generation capped at 128 tokens.
- **CNN/DailyMail**: Long-form summarization dataset with approximately 287K training, 13K validation, and 11K test articles paired with multi-sentence summaries; same truncation and generation cap as XSUM.
- **HumanEval**: Code generation benchmark consisting of 164 Python synthesis tasks; input up to 512 tokens, throughput measured on the first 300 generated tokens.

C Quantization

We applied 8-bit precision to quantize the injected model components utilized by each *Fractal Layer*, while the original model remained in full precision. This quantization strategy significantly reduces memory and computational demands and crucially preserves loss-less output parity with the baseline decoder. In terms of efficiency, the use of 8-bit matrix multiplications (mat-muls) speeds up parallel draft generation and lowers VRAM usage. Regarding quality, the final FP16 verification step corrects any minor quantization errors, thereby ensuring that end-to-end outputs precisely match those of the baseline decoder.

D Hyperparameters and Settings

Model	Params	Layers	n	w
Llama-3 3B	3 B	28	7	7
Llama-3 8B	8 B	32	8	8
CodeLlama-13B	13 B	40	10	10

Table 3: Model specifications and hyperparameters used in our experiments.

Hyperparameters The hyperparameters used in our experiments are listed in Table 3. We set n to be one-quarter of the total number of layers and define

Model & Dataset	GPUs	Baseline (h)	Fractal (h)	Δ
Llama-3 3B (GSM8K)	2×3090	1.5	1.1	–26.7%
Llama-3 3B (XSUM)	2×3090	2.5	1.3	–48.0%
Llama-3 3B (CNN/DM)	2×3090	3.5	2.1	–40.0%
Llama-3 8B (GSM8K)	2×3090	5.2	2.9	–44.2%
Llama-3 8B (XSUM)	2×3090	6.1	3.2	–47.5%
Llama-3 8B (CNN/DM)	2×3090	10.3	5.1	–50.5%
CodeLlama-13B (HumanEval)	3×3090	12.1	4.5	–62.8%

Table 4: Wall-clock inference time per dataset. Δ is the relative reduction of Fractal vs. baseline.

$w = n$. We set the gap between *Fractal Layer* insertions as the total number of layers divided by the number of insertions.

Generation proceeded via greedy decoding, and no KV cache is utilized. Maximum output lengths follow the task budgets in Appendix B: 128 tokens (XSUM, CNN/DailyMail), 256 (GSM8K), and 512 (HumanEval).

Hardware and Software Environment. Experiments were run on Ubuntu 22.04 with two RTX 3090 (24 GB) GPUs for the 3B/8B checkpoints and three RTX 3090s for the 13B checkpoint. Key libraries: Python 3.11, PyTorch 2.5 (+CUDA 12.1, cuDNN 8.9),Transformers 4.48, BitsAndBytes 0.45 (8-bit mode), Xformers 0.0.28.post3. The enviroment settings are available in the artifact repository.

Compute Budget. All experiments are inference-only (training budget 0 GPU-h). Across all datasets (Table 4) the *baseline* decoder consumes 94.5 GPU-h, whereas our Fractal decoder needs only 44.9 GPU-h (-53 %).

Code Availability. All inference scripts, log-parsing utilities, and plotting recipes are publicly released at `https://anonymous.4open.science/r/FractalLLM-B445/`.

External Resources and Licensing. All checkpoints are distributed under the *Meta Llama Non-Commercial License*; datasets are released under *CC-BY 4.0* or comparably permissive terms. We use them strictly for non-commercial research and do not redistribute derivatives.

E Use of AI Assistant

Translation work in this paper was assisted by an AI tool. All generated output was subsequently reviewed and revised by the authors to ensure accuracy and clarity.