Position-Aware Depth Decay Decoding (D^3): Boosting Large Language Model Inference Efficiency

Siqi Fan[1], Xuezhi Fang[2], Xingrun Xing[2,3], Peng Han[1], Shuo Shang[1*], Yequan Wang[2*]

[1]University of Electronic Science and Technology of China, Chengdu, China
[2]Beijing Academy of Artificial Intelligence, Beijing, China
[3]Institute of Computing Automation, Chinese Academy of Sciences, Beijing, China
`{sqfann, jedi.shang, tshwangyequan}@gmail.com`

Abstract

Due to the large number of parameters, the inference phase of Large Language Models (LLMs) is resource-intensive. Unlike traditional model compression, which needs retraining, recent dynamic computation methods show that not all components are required for inference, enabling a training-free pipeline. In this paper, we focus on the dynamic depth of LLM generation. A token-position aware layer skipping framework is proposed to save 1.5x times operations efficiently while maintaining performance. We first observed that tokens predicted later have lower perplexity and thus require less computation. Then, we propose a training-free algorithm called Position-Aware **D**epth **D**ecay **D**ecoding (D^3), which leverages a power-law decay function, $\left\lfloor L \times (\alpha^i) \right\rfloor$, to determine the number of layers to retain when generating token T_i. Remarkably, without any retraining, the D^3 achieves success across a wide range of generation tasks for the first time. Experiments on large language models (*i.e.,* the Llama) with $7 \sim 70$ billion parameters show that D^3 can achieve an average 1.5x speedup compared with the full-inference pipeline while maintaining comparable performance with nearly no performance drop ($< 1\%$) on the GSM8K and BBH benchmarks.

1 Introduction

LLMs have demonstrated impressive performance on various downstream tasks (*e.g.,* text generation, question & answering, and sentiment analysis) using various evaluation protocols such as zero-shot, few-shot, and fine-tuning (Touvron et al., 2023). Notably, In-context learning ability allows LLMs to adapt to tasks using input-output examples without parameter updates (Brown et al., 2020; Todd et al., 2024). However, their inference phases are very expensive due to the large number of parameters (Liu et al., 2023). LLMs employ multi-layer
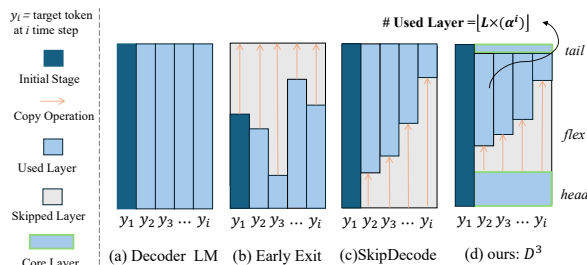

Figure 1: D^3's generation process *vs.* (a) standard implementation, (b) Early Exit, and (c) SkipDecode.

Transformers, focusing much of the computation on decoder blocks. For LLMs like Llama, inference complexity is $LSd(d + S)$ per single inference, where d is the word vector dimension, S is the sequence length, and L represents the number of decoder blocks (Narayanan et al., 2021). This shows that computational cost scales linearly with the number of decoder blocks. Therefore, many works explore the possibility of achieving *dynamic depth* in LLMs during inference (Schuster et al., 2022; Ma et al., 2023).

Early studies in visual neural networks (Bolukbasi et al., 2017; Huang et al., 2017) show that "Easy" instance activates at shallower layers while "hard" ones at deeper layers. *Dynamic depth* was then widely used in classification tasks with encoder-only LLM like BERT (Li et al., 2020; Liu et al., 2020; Kong et al., 2022), mainly through early exiting and skipping layers. The rise of generation tasks in decoder-only LLMs, prompts us to question: "Can we allocate different computational resources for generating tokens at different positions?" However, extending *dynamic depth* to generation tasks such as autoregressive decoding in Transformers presents challenges, as it requires computing the entire Transformer stack for each token (Schuster et al., 2022). This introduces two key challenges:

(i) Handling Missing State. Managing the miss-

---

*Corresponding authors.

ing hidden state and key-value cache for the current token T_i becomes challenging if the previous token T_{i-1} exited at a lower layer than T_i. The common approach (Corro et al., 2024; Schuster et al., 2022) is adding the missing state by copying. However, there are many variable factors affecting this copy operation are unclear (*e.g.,* when to add the missing state).

(ii) Decision on Skipping Layer Number and Direction. While the concept of "easy" instances activate at shallower layers and "hard" ones at deeper layers is appealing, this approach is incompatible with batching (Huang et al., 2017). The recent SkipDecode (Corro et al., 2024) addresses this by controlling the computational budget with batched exit points. However, choosing which layers to skip remains an open question. Currently, there are differing views on whether to skip head or tail layers at different timesteps. CALM (Schuster et al., 2022) uses a confidence measure to skip tail layers, while SkipDecode designs a batch exit function to skip head layers, but the reason behind this choice is unclear (Figure 1).

In this paper, we propose Position-Aware **D**epth **D**ecay **D**ecoding (D^3) to address the above challenges. D^3 dynamically reduces the number of activated layers per token.

We find that generation performance is sensitive to early decoding steps, as shown by Llama perplexity (PPL) decrease per token. We hypothesize that *"During LLM generation, tokens predicted later have lower perplexity and thus require less computation."* Previous work also (Schuster et al., 2022; Corro et al., 2024) supports this, noting that early incorrect predictions affect subsequent tokens and that average loss decreases over time. These findings guide the design of D^3.

The core of D^3 lies in decision-making strategy for each token (*i.e.,* skip layer number and skip direction). Inspired by the decreasing behavior of Llama PPL per token, we design a power law decay function $\lfloor L \times (\alpha^i) \rfloor$ to decide how many layers to keep when generating token T_i at i time step. Here, L is the total number of layers in the model, and α is a hyperparameter that controls the decay rate. For the skip direction, previous works like (Men et al., 2024; Yang et al., 2024) suggest skipping middle layers based on cosine similarity of hidden states. We further validate this by analyzing the information flow (*e.g.,* MLP, attention activations) across each block during pretraining.

Subsequently, we can perform D^3 to achieve efficient inference without additional learning. It involves only two hyperparameters: the flex layer *start ID* and the *decay rate (α)*. These can be determined through grid search on a validation set on a small model and then directly transferred to larger models. Moreover, the task-specific nature of these hyperparameters allows for adaptive inference, with different parameter settings reflecting the model's ability to adjust to specific tasks.

Experiments on well-known large language models (*i.e.,* the Llama series) with $7 \sim 70$ billion parameters show that our proposed D^3 can achieve 1.07x to 1.94x speedup compared with full-inference pipeline (*i.e.,* HuggingFace implementation) on the GSM8K and BBH benchmarks while maintaining comparable performance with nearly no performance drop ($< 1\%$). More importantly, D^3 is orthogonal to other model acceleration techniques (batch processing and KV caching), offering the potential for further enhancing inference efficiency. We argue that D^3 unlocks a new paradigm for efficient inference alongside existing effective methods.

In summary, our main contributions are as follows:

- A simple but effective framework (D^3) to decide computation resource per token for LLM generation without any model retraining.

- A complement analysis of the token-position wise decoding that motivates the design of power law decay function.

- Experiments demonstrate D^3 achieve average 1.5x speedup on two benchmarks while maintaining comparable performance.

2 Related Work

Dynamic Depth. Dynamic depth involves two methods: Early Exit (EE) and Skip layer. EE first appeared in CNN/DNN networks for visual tasks (Bolukbasi et al., 2017; Huang et al., 2017; Teerapittayanon et al., 2016). Subsequently, it was utilized in accelerating the inference of encoder-only architectures in BERT for classification tasks by (Li et al., 2020; Liu et al., 2020; Li et al., 2021; Kong et al., 2022). Recently, (Schuster et al., 2022; Varshney et al., 2023; Corro et al., 2024) discuss confidence-based or batch exit EE for accelerate LM inference. Meanwhile, skip-layer dynamically omits the execution of middle layers (or modules) for any input token, facilitated by a gate function

Method	Gen.	KV	Batch	Extra Draft	Task Adapt.
Early-EE	✗	✗	✗	✗	✓
CLAM	✓	✗	✗	✗	✗
SkipDecode	✓	✓	✓	✗	✗
SD	✓	✓	✓	✓	✓
Self-SD	✓	✓	✓	✗	✓
Ours: D^3	✓	✓	✓	✗	✓

Table 1: Comparison of related methods. Gen. indicates support for generation, KV and Batch for KV caching and batch processing, Extra Draft for requiring a draft model, and Task Adapt. for task adaptability.

(Wang et al., 2018) or a binary router (Zeng et al., 2023) and layer pruning (Kim et al., 2024; Yang et al., 2024; Song et al., 2024). In addition, Dola and SLED (Zhang et al., 2024; Chuang et al., 2024) improve model factuality from the perspective of dynamic depth, highlighting the potential of dynamic depth in various applications.

Speculative Decoding (SD). Vanilla speculative decoding (Leviathan et al., 2023; Chen et al., 2023) uses two models: a lightweight draft model for simple tokens and a powerful verification model for complex tokens. The draft model quickly predicts potential tokens, while the verification model checks their accuracy in parallel. To reduce model dependency, self-Speculative Decoding (Zhang et al., 2023; Elhoushi et al., 2024), using a skip-layer version of the verification model as the draft model, thus avoiding additional model loading overhead. These skipped layers are searched via Bayesian optimization, which efficiently identifies the most suitable layer configurations for draft model. Compared to existing methods, our approach eliminates secondary model loading and token verification process, with key differences summarized across four dimensions in Table 1.

3   Methodology

In this section, we introduce D^3. We begin with brief recap the two phases of text generation for decoders-only transformers for convenience (§ 3.1), then bring the challenges when dynamic depth comes up with generation. Last, we investigate the effects of dynamic depth on model performance (§ 3.2) identify the primary reason for performance degradation, and propose strategies to mitigate them (§ 3.3), which guide our designed D^3 for per token-position wise decoding(§ 3.4).

3.1   Preliminary: Dissecting Efficient Inference of LLMs

Mainstream LLMs (*e.g.,* GPT, Llama) are rooted in the Transformer architecture (Vaswani et al., 2017), and pretrained with a full language modeling objective with a decoder-only structure, computing loss on all tokens.

Two phases behind generative tasks. When LLMs engage in generative tasks, they enter a next-token prediction loop until reaching an exit signal (*e.g.,* an EOS token or reaching the maximum sequence length). This process involves two phases. The first, known as the *initiation phase*, entails generating the first token of the completion by processing the tokenized prompt through the network. Subsequently, the generated token is appended to the input token sequence, becoming part of the new input to generate the subsequent token. This iterative process continues until the exit signal is encountered, often referred to as the generation phase.

Practical accelerate methods: Batching and KV cache. Batch processing and Key-Value (KV) cache are practical acceleration methods used during inference in LLMs. Batch processing involves simultaneously handling batches of data during inference. Since the decoder operates causally, during the generation phase, Key-Value (KV) caches prevent the recomputation of keys and values for past tokens to reduce computation costs.

Challenges. (i) Instance-aware dynamic depth poses challenges for batch processing and KV cache. Confidence-based methods like CALM (Schuster et al., 2022; Varshney et al., 2023) when processing batch data, instances within the same batch have to wait until the last token in a batch exits before they can stop computing. This severely restricts the practical application of such techniques (Corro et al., 2024). (ii) Filling missing state has error propagation. At the sequence level, if the depth at which the previous token exits is earlier than that of subsequent tokens, filling of missing layers's hidden state and KV cache for preceding tokens may cause error propagation. (iii) Uncertainty skip layer number and direction. CALM (Schuster et al., 2022) uses a confidence measure to skip tail layers, while SkipDecode designs a batch exit function to skip head layers. However, the rationale behind their choice is unclear.

Model	Layer Num.	SD	Conf
Llama2 7B	32	21	0.8
Llama2 13B	40	23	0.7
Llama2 70B	80	53	0.8

Table 2: LLM Saturation Statistics. "SD" refers to the average Saturation Depth per token, and "conf" indicates the confidence level of the Saturation Depth.

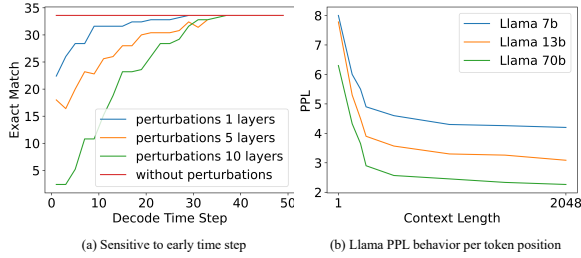(a) Sensitive to early time step   (b) Llama PPL behavior per token position

Figure 2: Error propagation and explanation from perplexity (PPL) behavior in filling missing States.

3.2 Error Propagation on Filling Missing State

In autoregressive decoding, when generated token t, computing the input hidden state h_t^i for layer i depends on $h_{1:t-1}^{i-1}$, which is the output hidden states of the previous layer for all the tokens that have been generated so far. Therefore, if the model has early exited at some layer $j < i-1$ for a token $s < t$, then h_s^{i-1} and KV cache for s is not available.

To handle these missing hidden states and KV cache, methods like CALM (Elbayad et al., 2019; Schuster et al., 2022) adopt the approach of copying hidden states and recomputing KV cache, while SkipDecode (Corro et al., 2024) copies both hidden states and KV cache. Both methods may introduce error propagation. To this end, we first investigate the potential of layer skip in LLMs. Then we will analyze the impact of copied operation on performance, in addition to considering other factors. We use Llama2-7b (Touvron et al., 2023) and the BBH word sorting task (Suzgun et al., 2022) for these experiments.

Dynamic depth potential across scaling laws. We control for the correctness of the predicted tokens to examine the potential of early exit in LLMs. That is, we first explored confidence-based oracles for decoder-only LLMs without regard to cost (Schuster et al., 2022). Due to the exact match
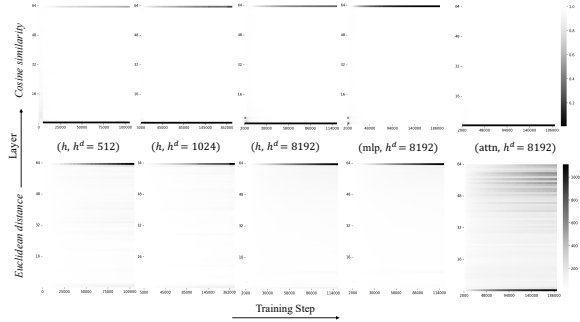

Figure 3: Visualization of input/output information flow, including features hidden state, mlp, and attention activation value, for each block during training.

evaluation metric for word sorting[1], when generating each token t, we can compare each block i's output hidden state h_t^i with the last block hidden state h_t^L after passing lm_head layer[2]. This allows us to identify the hidden state's Saturation Depth and obtain the corresponding confidence. This process can be represented by the formula: for each generated token t in each block $i, i \in [1, L)$, $\arg\max p(y_{t+1} \mid lm\_head(h_t^i)) = \arg\max p(y_{t+1} \mid lm\_head(h_t^L))$.

CALM (Schuster et al., 2022) finds that in an 8-layer T5 encoder-decoder model, exits at an average of 1.53 layers per token without performance degradation. Similar observation also happens in decoder-only LLMs, for the initial phase, the first token required processing through all layers, but during the generation phase, approximately 95% of the tokens were correctly predicted by around 0.7 depth of the model (Refer to Table 2). This suggests that LLMs exhibit significant redundancy in depth, highlighting substantial potential for compute savings through layer skipping.

Copy operation is sensitive to early time step. Note that in the above Oracle experiment, missing hidden states are copied while missing KV caches are recalculated regardless of cost. When KV caches are copied, performance drops significantly (by about 20% on the word sorting task). Although CALM (Schuster et al., 2022) addresses this by recalculating KV states, this approach is impractical for decoder-only architectures due to high computational costs (Corro et al., 2024). There-

---

[1]A given predicted string's exact match score is 1 if it is the exact same as its reference string, and is 0 otherwise.

[2]The classification layer (*lm_head*) transforms decoder logits into a vocabulary-wide probability distribution using linear transformation and *softmax*, enabling word prediction by selecting the top probability option.

$$S_{kipped}(T_i) = \{l \in \mathbb{Z}, \text{head} \le l \le \text{head} + L - \lfloor L \times (\alpha^i) \rfloor \}$$

Skipped layer ID set · Decay function · i-th generated token · Start skipped layer ID · Model max Layer ID

(a) power-law decay function with different decay rate

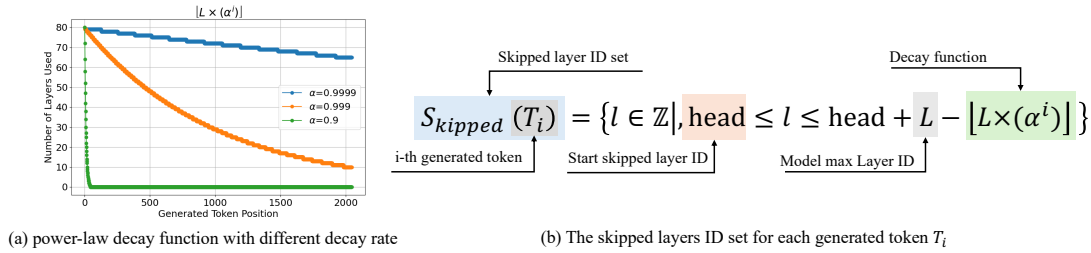(b) The skipped layers ID set for each generated token T_i

Figure 4: The layer usage for the current generated token T_i follows a power-law decay function with decode time steps.

fore, we examine how the copy operation affects generative tasks in Figure 2(a). We consider two variables: (i) The effect of decoder time steps (one token decoded at a time) on final performance when the same layers are copied, and (ii) the number of layers copied. For convenience, only tail layers are copied. The results are depicted in Figure 2(a), which shows that earlier perturbations lead to more significant performance degradation while copying more layers exacerbates performance decline. We attribute this outcome to perplexity (PPL) behavior per token in Figure 2(b). It provides a measure of uncertainty or "surprise" about the prediction. For a token T_i with probability $p(T_i)$, it is calculated as $PPL(T_i) = \frac{1}{p(T_i)}$. At the start of the generation, limited information causes higher PPL. As more tokens are decoded, richer context lowers PPL for later tokens. Thus, we hypothesize that *"During LLM generation, tokens predicted later have lower perplexity and thus require less computation."*

3.3 Core & Flex Layer

Modern LLMs build coarse-grained features in their head layers and develop more detailed, fine-grained representations in deeper layers, facilitated by multi-head attention mechanisms (Vaswani et al., 2017) and residual connections (He et al., 2016). To investigate representation changes in each decoder block, we visualize the input-output flow during training, as shown in Figure 3. Specifically, we calculate Cosine similarity and Euclidean distances for hidden state, perception, and attention activation. For instance, given output hidden states h_1, h_2 from block 1 and block 2 at training step t, the cosine similarity is computed as $\frac{h_1 \cdot h_2}{\|h_1\| \|h_2\|}$, and the Euclidean distance is $\sqrt{\sum_{i=1}^{d}(h_{1,i} - h_{2,i})^2}$, where d is the hidden size. Other metrics, such as perception (MLP) and attention activation (attn), follow similar calculation processes. Detailed data is provided in the supplementary materials.

Results show that while middle layers exhibit minimal changes in input-output flow over time, the head and tail layers remain distinct. We propose that the head and tail layers can be called core layers, with specific roles: (i) head layers handle abstract, fundamental features close to the embedding layer, and (ii) final layers align with the output near the classification layer. Middle layers, termed flex layers, are more adaptable. Dynamic depth like CALM (Schuster et al., 2022) skip tail layers, and SkipDecode (Corro et al., 2024) skip head layers. We argue that the middle flex layers should be skipped first, and the optimal starting point for skipping can be determined through parameter search in smaller models.

3.4 D^3: Token Position Decay Strategy

Previous early exit methods require a stop signal for each token, such as training early exit classifiers (Li et al., 2020; Liu et al., 2020) or using *softmax* response (Schuster et al., 2022). However, we believe these methods significantly increase computational load, especially since the *softmax* response projects the hidden state to a large output vocabulary size. Therefore, after identifying the core layers, we propose a decay method that does not require training or *softmax* response. SkipDecode (Corro et al., 2024) uses linear decay given a target speedup ratio. In contrast, we design a power-law decay function $\lfloor L \times (\alpha^i) \rfloor$ based on the power-law trend of PPL behavior in Figure 2(b). The decay rate is adjusted by controlling the decay coefficient α, as illustrated in Figure 4(a). Given a model with L layers, the skipped layers for each generated token T_i are depicted in Figure 4(b).

It's worth noting that our approach is effective yet simple and easy to implement. Besides the token skipping policy, it does not necessitate any additional modifications to the transformer architecture, either during training or generation.

2994

Model	Params	Layer Num.	GQA
Llama 2	7B	32	✗
Llama 2	13B	40	✗
Llama 2	70B	80	✓

Table 3: LLMs statistics.

Task	output length
BBH	182(2-182)
GSM8K	1230(50-1230)

Table 4: Tasks statistics

HP	Grid Search Space	Description
$start$	[0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]	Drop layer start position
α	[0.8, 0.9, 0.999, 0.9999]	Decay rate

Table 5: Details of the computation layer ID set used for generating token T_i at time step i, along with the search range and descriptions for the formula's hyperparameters.

4 Experiments

4.1 Experiment Settings

Evaluation Tasks. We assess our method from two common text generation benchmarks from Hugging Face Open LLM Leaderboard with varying target lengths and domins: **BBH** (Suzgun et al., 2022), comprising 23 challenging BIG-Bench tasks (Srivastava et al., 2022), where previous language models fell short compared to human raters, necessitating multi-step reasoning and few-shot prompting without CoT; **GSM8K** (Cobbe et al., 2021), designed for question answering on basic mathematical problems requiring multi-step reasoning. The average number of tokens in reference targets of evaluation datasets is detailed in Table 4.

Evaluation Metrics. For performance evaluation, we report exact match score from the lm-harness evaluation framework (Gao et al., 2023), this serves as the backend for Open LLM Leaderboard and is utilized by numerous open-source LLMs (Biderman et al., 2023; Touvron et al., 2023). We evaluate our approach under few-shot scenarios, using sample sizes of 3. Training set examples are added to x_q. For in-context learning prompts, we use a default template: Q : $\{x_k\}$ \nA : $\{y_k\}$ \n\n, concatenating random x_k and y_k samples from task-specific training sets.

Following CALM (Schuster et al., 2022), our main efficiency metric is the average number of decoder layers used per output token, as it directly measures complexity reduction without conflating with implementation or infrastructure specific details. For reference, we convert it to average FLOPs reduction per output token (Elbayad et al., 2019; Narayanan et al., 2021; Corro et al., 2024). Taking into account the conditional checks and redundant parameter passing (*e.g.,* token position) involved in D^3, we also compared the actual speed of D^3 in real-world scenarios compared with Hugging Face implementation, reporting wall-clock time (Dehghani et al., 2021).

Large Language Models. For D^3's backbone, we choose widely recognized Llama 2 series, de-

tailed in Table 3. These models vary in terms of the number of parameters, ranging from 7 billion to 70 billion, and the number of layers, ranging from 32 layers to 80 layers. Compared with Llama 2 7/13B version, the 70B version employs Grouped Query Attention (Ainslie et al., 2023), enhancing its inference capabilities.

Hyperparameter Settings We conducted a grid search using 10% of the training set to determine the optimal values for $start$ and α. The hyperparameter ranges explored are listed in Table 5. Importantly, the optimal hyperparameters identified for smaller models were directly applied to larger models, significantly reducing the time and effort required for the search. For GSM8k, we found $\alpha = 0.9999$ and $start = 0.2$. For BBH, the values were $\alpha = 0.99$ and $start = 0.6$. Notably, these hyperparameters, optimized for the LLaMA 7B model, were efficiently transferred to the LLaMA 13B and 70B models, demonstrating a low-cost, high-efficiency approach.

Comparison Methods. We select Early Exit (Schuster et al., 2022) (*i.e.,* CALM-DEC) and SkipDecode (Corro et al., 2024) for comparison with our D^3. The original CALM (skip deep layers) was designed for the encoder-decoder architecture of T5 and does not support batch processing due their adaptive hidden state saturation policy. To facilitate comparison, we adapted the CALM early exit concept to suit decoder-only models and enable batch processing. SkipDecode (skip shallow layers) applies linear decay within the specified upper and lower bounds of the model's executable layers, given a target speedup ratio. For a fair comparison, we ensure that SkipDecode and our method have a comparable or greater average layer number, with other parameters (*e.g.,* warm-up layer number and passing prompt inputs through all layers in the initial stage) consistent with the original paper. To ensure reproducibility, all experiments were conducted with a fixed random seed, controlling for randomness and enabling accurate comparisons.

Methods	BBH			GSM8K		
	EM	#Avg.Layer	FLOPs r.	EM	#Avg.Layer	FLOPs r.
Llama2 7B	*maximum length 200*			*maximum length 1024*		
Full Depth	31.22	32.00	-	10.61	32.00	-
Early Exit	31.22	19.73	1.60x	10.61	29.90	1.07x
SkipDecode	31.65	24.68	1.28x	0.00	29.90	1.07x
Ours: D^3	**31.88**	**19.17**	**1.64x**	**12.05**	29.90	1.07x
Llama2 13B	*maximum length 200*			*maximum length 1024*		
Full Depth	37.77	40.00	-	22.67	40.00	-
Early Exit	37.72	33.50	1.19x	22.74	37.51	1.06x
SkipDecode	38.98	30.99	1.29x	9.32	37.51	1.06x
Ours: D^3	**39.07**	**20.33**	**1.94x**	**23.43**	37.51	1.06x
Llama2 70B	*maximum length 200*			*maximum length 1024*		
Full Depth	50.90	80.00	-	52.91	80.00	-
Early Exit	50.02	50.14	1.59x	22.74	75.53	1.05x
SkipDecode	**51.87**	62.48	1.27x	26.68	75.53	1.05x
Ours: D^3	51.42	**48.43**	**1.65x**	**54.51**	75.53	1.05x

Table 6: Main Results: Performance and efficiency across model scales.

Actual Time	Llama2 7B	Llama2 13B	Llama2 70B
HF Full Depth	6218	10428	12893
Ours	5559	8427	11294
Speed Up	1.12x	1.23x	1.14x

Table 7: Wall-clock time(s) and actual speed up on GSM8K.

4.2 D^3: Performance and Efficiency Across Scaling Laws

The main experimental results of D^3 are summarized in Tables 6 and 8. These experiments were conducted in few-shot settings, showcasing performance and computational efficiency compared to HuggingFace's full-depth implementation and early exit method, SkipDecode. From a perspective of performance and computational efficiency, we can draw the following experimental conclusions.

Performance is Comparable with Minimal Loss($< 1\%$). Tables 6 and Figure 5 show that exact match remains within a narrow margin of $< 1\%$, when compared to HuggingFace's full-depth implementation. D^3 maintains mainstream LLM capabilities and in-context learning abilities without modifying model parameters. This finding is promising, especially in light of our observation in Table 2, where we demonstrate the feasibility of implementing early exit strategies within LLM middle layers while preserving accuracy. For certain tasks, D^3 surpasses the last layer (Full Depth). This hints at a tendency for deep layers to potentially over-represent certain tasks, which could impede performance during LLM inference.

Over 1.5x Speed Up. We convert the average skipped layers per token for each task to FLOPs
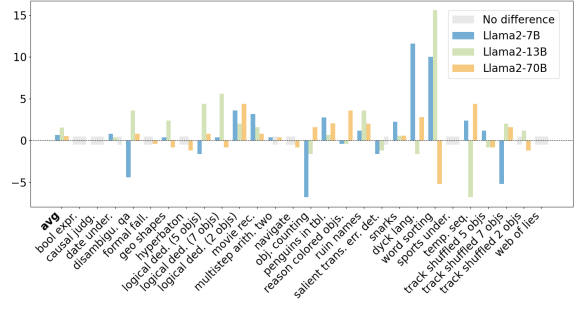

Figure 5: D^3 *vs.* Full Depth performance difference (±) on BBH benchmarks

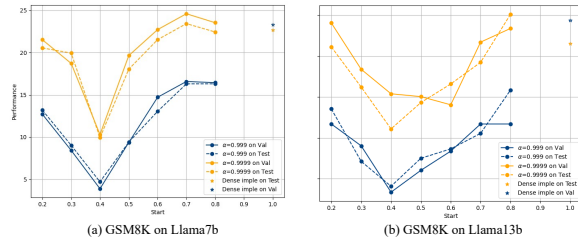(a) GSM8K on Llama7b

(b) GSM8K on Llama13b

Figure 6: α *vs.* $Start$ accross model scales. The consistent HP sensitivity trends between Validation and Test sets suggest that optimal hyperparameters can be identified on a small validation set and transferred to larger models, saving extensive search efforts.

reduction in Table 6. It can be observed that the FLOPs reduction varies for different types of tasks, ranging from $1.07x$ to $1.94x$. This variation is because of different output lengths of each task. We argue that, *the computation required decreases during the generation process*, and allocating fewer computational resources for later tokens can improve computational efficiency.

Recall that the universality decoder layers, our algorithmic improvements D^3 are fully compatible with traditional model acceleration techniques. This compatibility is noteworthy, especially considering that many earlier instance-wise dynamic depth methods (Schuster et al., 2022) did not support batching and KV cache. These improvements are applicable to various LLMs and can be easily transferred despite minor architectural and activation function differences.

Wall-clock time compared with HuggingFace implementation. It's worth mentioning that the actual speedup may slightly vary during generation, given the impossibility of predicting the number of tokens the model will generate in advance (Corro et al., 2024). Additionally, the introduction of certain computations, such as conditional statements and array creation, also adds to the computational

color mapping:	80 layers	79 layers	78 layers	77 layers	76 layers	75 layers

X	Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?
Standard Y_{full_80}	\<s\>_Jan_et_e_ats__3_eggs_for_breakfast_and_b_akes_m_uff_ins_with__4_eggs_._\n_So_,_she_s_ells__1_6_-__3_-__4_=_<<_1_6_-_3_-_4_=_9_>>_9_eggs_at_the_far_mers_'_market_._\n_She_s_ells__9_eggs_at_$_2_per_egg_,_so_she_makes_$_2_x__9_=_$_<<_2_*_9_=_1_8_>>_1_8_per_day_at_the_far_mers_'_market_._\n_####__1_8_\n_\n
$\alpha = 0.999$ Y_1	\<s\>_Jan_et_s_ells__1_6_-__3_-__4_=_<<_1_6_-_3_-_4_=_9_>>_9_du_ck_eggs_a_day_._\n_She_makes__9_*__2_=_$_<<_9_*_2_=_1_8_>>_1_8_every_day_at_the_far_mer_'_s_market_._\n_####__1_8
$\alpha = 0.9999$ Y_2	\<s\>_Jan_et_e_ats__3_eggs_and_b_akes__4_eggs_,_so_she_s_ells__1_6_-__3_-__4_=_<<_1_6_-_3_-_4_=_9_>>_9_eggs_._\n_She_s_ells__9_eggs_for_$_2_each_,_so_she_makes_$_2_*__9_=_<<_2_*_9_=_1_8_>>_1_8_dollars_._\n_####__1_8_\n_\n

Figure 7: Example output comparison in GSM8K: Hugging Face implementation vs. D^3 outputs under different decay rate α by our design power law decay function $\lfloor L \times (\alpha^i) \rfloor$. Tokens of different colors correspond to the number of computational layers used for each token T_i. When the decay rate is relatively small, the computation decreases rapidly, as seen in the output of Y_1. Conversely, for Y_2, the computation decreases more slowly.

load. Therefore, to accurately assess the practical acceleration benefits of our approach, we integrated D^3 into the lm-harness evaluation framework and conducted end-to-end speed measurements for the entire GSM8K task within the framework, comparing it with the HF implementation. For the LLama 7B and 13B versions, we utilized 2xV100 GPUs with 32GB memory, setting the batch size to 4. As for the 70B model, we employed 8xA100 GPUs with 40GB memory, setting the batch size to 8. All three models were subjected to data parallelism during inference. The results are presented in Table 7, indicating that our method achieves significant acceleration in real-world scenarios, with this advantage becoming more pronounced as the batch size increases.

4.3 Factor Study

We investigate the impact of Flex Layer start position and decay rate on performance and conduct case studies on output samples for different decay rates.

Exploration of decay rate and flex Layer impact. Firstly, we examine the effect of decay rate α and Flex Layer $start$ depth on performance on GSM8K, as shown in Figure 6. We draw the following conclusions: (i) At the same decay rate, it is more appropriate to start the decay from the middle of the model for models of various sizes. This experimental conclusion confirms our hypothesis in Section 3.3 that LLMs have Core and Flex Layers. The Core Layer generally represents the layers at the beginning and end, while the Flex Layer represents the middle layers, where redundant layers exist and can be skipped to improve inference efficiency. (ii) Additionally, the results presented in

the Appendix Table 9 for BBH exhibit a similar trend, with optimal hyperparameters differing from those for GSM8K. In this appendix, we provide the detailed numerical results for the BBH benchmark, as shown in Table 8. Table 9 presents the description and search range for each parameter during the hyperparameter tuning process. Table 10 demonstrates the successful transfer of optimal parameters, identified using 10% of the validation set, from smaller models to larger-scale models.

This observation suggests that α and $start$ are task-specific parameters. We hypothesize that this is due to the BBH task having a much shorter output length compared to GSM8K, which necessitates a faster decay rate (i.e., a smaller decay coefficient) for optimal performance.

Example Output: optimizing model capacity allocation by token position. Figure 7 illustrates two example outputs from D^3 for mathematical reasoning responses on the GSM8K dataset, compared to the outputs of the full model (Llama70B with 80 layers) implemented by Huggingface. Black tokens indicate passage through all layers, while other colors represent the number of layers each token has traversed, as computed under different decay coefficients.

5 Conclusion

This work introduces a token-position wise layer skipping framework called D^3 which saves 1.5x times operations efficiently while maintaining competitive performance. Through analysis of the missing states and input-output flow, we design a training-free algorithm using the power law decay function to decide how many layers to keep per generated token. Experimental results on main-

stream LLMs, demonstrate D^3 significantly improves speed (average 1.5x) compared to Hugging-Face implementation on GSM8K and BBH benchmarks, while keeping performance barely no drop. Additionally, D^3 can complement other model acceleration techniques, such as batch processing and KV caching, potentially enhancing inference efficiency. We argue that D^3 establishes a new paradigm for efficient inference alongside existing effective methods.

6 Limitation

This work approaches LLM behavior from a global perspective, analyzing per-token perplexity (PPL) dynamics. However, it may overlook the impact of certain special tokens, which could introduce nuances that are not fully captured in the proposed framework.

Acknowledgments

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. 2017. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning*, pages 527–536. PMLR.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *CoRR*, abs/2302.01318.

Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James R. Glass, and Pengcheng He. 2024. Dola: Decoding by contrasting layers improves factuality in large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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 preprint arXiv:2110.14168*.

Luciano Del Corro, Allison Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Hassan Awadallah, and Subhabrata Mukherjee. 2024. Skipdecode: Autoregressive skip decoding with batching and caching for efficient LLM inference.

Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. 2021. The efficiency misnomer. *arXiv preprint arXiv:2110.12894*.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2019. Depth-adaptive transformer. *arXiv preprint arXiv:1910.10073*.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. 2024. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*.

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.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. 2024. Shortened llama: A simple depth pruning for large language models. *arXiv preprint arXiv:2402.02834*.

Jun Kong, Jin Wang, Liang-Chih Yu, and Xuejie Zhang. 2022. Accelerating inference for pretrained language models by unified multi-perspective early exiting. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 4677–4686.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Lei Li, Yankai Lin, Deli Chen, Shuhuai Ren, Peng Li, Jie Zhou, and Xu Sun. 2020. Cascadebert: Accelerating inference of pre-trained language models via calibrated complete models cascade. *arXiv preprint arXiv:2012.14682*.

Xiaonan Li, Yunfan Shao, Tianxiang Sun, Hang Yan, Xipeng Qiu, and Xuanjing Huang. 2021. Accelerating bert inference for sequence labeling via early-exit. *arXiv preprint arXiv:2105.13878*.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. 2020. Fastbert: a self-distilling bert with adaptive inference time. *arXiv preprint arXiv:2004.02178*.

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.

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. *CoRR*, abs/2403.03853.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient large-scale language model training on GPU clusters using megatron-lm. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021*, page 58. ACM.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472.

Jiwon Song, Kyungseok Oh, Taesu Kim, Hyungjun Kim, Yulhwa Kim, and Jae-Joon Kim. 2024. Sleb: Streamlining llms through redundancy verification and elimination of transformer blocks. *arXiv preprint arXiv:2402.09025*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pages 2464–2469. IEEE.

Eric Todd, Millicent L. Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. 2024. Function vectors in large language models. In *Proceedings of the 2024 International Conference on Learning Representations*.

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

Neeraj Varshney, Agneet Chatterjee, Mihir Parmar, and Chitta Baral. 2023. Accelerating llama inference by enabling intermediate layer decoding via instruction tuning with lite. *arXiv e-prints*, pages arXiv–2310.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. 2018. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424.

Yifei Yang, Zouying Cao, and Hai Zhao. 2024. Laco: Large language model pruning via layer collapse. *arXiv preprint arXiv:2402.11187*.

Dewen Zeng, Nan Du, Tao Wang, Yuanzhong Xu, Tao Lei, Zhifeng Chen, and Claire Cui. 2023. Learning to skip for language modeling. *arXiv preprint arXiv:2311.15436*.

Jianyi Zhang, Da-Cheng Juan, Cyrus Rashtchian, Chun-Sung Ferng, Heinrich Jiang, and Yiran Chen. 2024. Sled: Self logits evolution decoding for improving factuality in large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2024*.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2023. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *arXiv preprint arXiv:2309.08168*.

Tasks	7B	D^3-7B	13B	D^3-13B	70B	D^3-70B
boolean expressions	68.40	68.40	72.80	72.80	82.00	82.00
causal judgement	50.27	50.27	54.55	54.55	62.57	62.57
date understanding	36.80	37.60 ↑ (0.8)	51.20	51.60 ↑ (0.4)	61.60	61.60
disambiguation qa	53.60	49.20 ↓ (4.4)	32.80	36.40 ↑ (3.6)	57.20	58.00 ↑ (0.8)
dyck languages	8.00	19.60 ↑ (11.6)	6.00	7.60 ↓ (1.6)	20.00	22.80 ↑ (2.8)
formal fallacies	43.60	43.60	52.40	52.40	52.00	51.60 ↓ (0.4)
geometric shapes	9.20	9.60 ↑ (0.4)	31.60	34.00 ↑ (2.4)	47.60	46.80 ↓ (0.8)
hyperbaton	48.40	48.40	61.20	61.20	74.00	72.80 ↓ (1.2)
logical deduction (5 objects)	25.60	24.00 ↓ (1.6)	21.20	25.60 ↑ (4.4)	35.20	36.00 ↑ (0.8)
logical deduction (7 objects)	14.80	15.20 ↑ (0.4)	18.40	24.00 ↑ (5.6)	42.00	41.20 ↓ (0.8)
logical deduction (2 objects)	32.40	36.00 ↑ (3.6)	39.60	41.60 ↑ (2)	59.20	63.60 ↑ (4.4)
movie recommendation	38.40	41.60 ↑ (3.2)	74.40	76.00 ↑ (1.6)	92.00	92.80 ↑ (0.8)
multistep arithmetic two	0.40	0.80 ↑ (0.4)	1.20	1.20	1.20	1.60 ↑ (0.4)
navigate	41.60	41.60	59.20	59.20	59.60	58.80 ↓ (0.8)
object counting	35.60	28.80 ↓ (6.8)	50.00	48.40 ↓ (1.6)	51.60	53.20 ↑ (1.6)
penguins in a table	24.66	27.40 ↑ (2.74)	29.45	30.14 ↑ (0.69)	39.04	41.10 ↑ (2.06)
reasoning about colored objects	21.60	21.20 ↓ (0.4)	27.20	26.80 ↓ (0.4)	45.60	49.20 ↑ (3.6)
ruin names	26.40	27.60 ↑ (1.2)	40.00	43.60 ↑ (3.6)	82.80	84.80 ↑ (2)
salient translation error detection	23.60	22.00 ↓ (1.6)	35.20	34.00 ↓ (1.2)	50.40	50.40
snarks	46.63	48.88 ↑ (2.25)	52.81	53.37 ↑ (0.56)	80.34	80.90 ↑ (0.56)
word sorting	12.00	20.00 ↑ (10)	16.00	31.60 ↑ (15.6)	32.40	27.20 ↓ (5.2)
sports understanding	66.40	66.40	64.80	64.80	80.40	80.40
temporal sequences	7.60	10.00 ↑ (2.4)	18.80	12.00 ↓ (6.8)	62.40	66.80 ↑ (4.4)
tracking shuffled objects (5 objects)	15.60	16.8 ↑ (1.2)	17.20	16.40 ↓ (0.8)	16.40	15.60 ↓ (0.8)
tracking shuffled objects (7 objects)	16.80	11.60 ↓ (5.2)	12.80	14.80 ↑ (2)	14.00	15.60 ↑ (1.6)
tracking shuffled objects (2 objects)	33.20	33.20	32.00	33.20 ↑ (1.2)	30.40	29.20 ↓ (1.2)
web of lies	48.80	48.80	52.00	52.00	48.80	48.80
Avg. 27 Tasks	31.22	31.88 ↑ (0.66)	37.77	39.07 ↑ (1.3)	50.90	51.42 ↑ (0.52)

Table 8: Detailed Performance comparison for BBH 27 tasks between HuggingFace Implementation and D^3

T_i Drop Layer set	HP	Grid Search Space	Description
$\{l \mid \text{start} < l < \text{start} + L - \lfloor L \times (\alpha^i) \rfloor\}$	$start$	[0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]	Drop layer start position
	α	[0.999, 0.9999]	Decay rate

Table 9: Details of the computation layer ID set used for generating token T_i at time step i, along with the search range and descriptions for the formula's hyperparameters.

α	start													
	0.2		0.3		0.4		0.5		0.6		0.7		0.8	
	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test
Llama 7b	Dense Implementation of Validation and Test Results: 11.76, 10.61													
0.999	6.68	7.42	5.61	4.85	3.34	3.63	4.41	5.00	5.35	5.46	6.68	6.22	6.68	8.34
0.9999	11.63	10.46	9.36	8.49	8.16	6.44	8.02	7.66	7.62	8.42	10.69	9.70	11.36	12.05
Llama 13b	Dense Implementation of Validation and Test Results: 23.32, 22.67													
0.999	12.70	13.19	8.42	9.02	3.88	4.70	9.36	9.40	14.71	13.04	16.58	16.30	16.44	16.30
0.9999	21.52	20.55	18.72	19.94	10.30	9.33	19.65	18.04	22.72	21.53	24.60	23.43	23.54	22.44

Table 10: Hyperparameter Search Results: Performance for Different $start$ and α Values, Including Validation (10% of train set) and Test Results. The results indicate that **minor searches in validation parameters are effective on test performance, showing consistent sensitivity trends**. Due to space constraints, only the GSM8k results are shown; similar results apply to the BBH task. We highlighted the best parameter combinations in Table 1, noting that a broader search could yield even better results.