



# DEL: Context-Aware *Dynamic Exit Layer* for Efficient Self-Speculative Decoding

Hossein Entezari Zarch\*, Lei Gao\*, Chaoyi Jiang, Murali Annavaram

University of Southern California

{entezari, leig, chaoyij, annavara}@usc.edu

## Abstract

Speculative Decoding (SD) is a widely used approach to accelerate the inference of large language models (LLMs) without reducing generation quality. It operates by first using a compact model to draft multiple tokens efficiently, followed by parallel verification using the target LLM. This approach leads to faster inference compared to auto-regressive decoding. While there are multiple approaches to create a draft model, one promising approach is to use early-exit methods. These methods draft candidate tokens by using a subset of layers of the primary model and applying the remaining layers for verification, allowing a single model to handle both drafting and verification. While this technique reduces memory usage and computational cost, its performance relies on the choice of the exit layer for drafting and the number of tokens drafted (speculation length) in each SD round. Prior works use hyperparameter exploration to statically select these values. However, our evaluations show that these hyperparameter values are task-specific, and even within a task they are dependent on the current sequence context. We introduce DEL (Dynamic Exit Layer), a plug-and-play method that adaptively selects the exit layer and speculation length during inference. DEL dynamically tracks the token acceptance rate if the tokens are drafted at each layer of an LLM and uses that knowledge to heuristically select the optimal exit layer and speculation length. Our experiments across a broad range of models and downstream tasks show that DEL achieves overall speedups of  $2.16\times\sim 2.62\times$  over vanilla auto-regressive decoding and improves upon state-of-the-art SD methods, which peak at  $2.43\times$ , by up to  $0.19\times$ . The code is available at <https://github.com/hoenza/DEL>.

## 1 Introduction

Large language models (LLMs) such as Llama 2 (Touvron et al., 2023), Llama 3 (Dubey et al., 2024), GPT-4 (OpenAI et al., 2024), and Claude (Bai et al., 2022) have demonstrated strong performance across diverse tasks, driving adoption in search engines (Microsoft, 2023; Reid, 2023), chatbots (OpenAI, 2022), and virtual assistants (Wu et al., 2023a;b). However, LLM token generation remains slow due to its auto-regressive nature, where each token is generated sequentially based on all previous tokens. This sequential dependency results in low generation speed, as every new token requires access to the full set of model parameters. The resulting I/O bottleneck limits hardware utilization during inference, especially in small-batch or low-latency scenarios, such as edge deployments or real-time applications.

Speculative decoding (SD) (Leviathan et al., 2023; Chen et al., 2023) addresses this bottleneck by using a smaller model to draft a sequence of tokens auto-regressively, which are then verified in parallel by the target model. This reduces latency while preserving the output distribution of the main model. The performance of SD depends on two key factors: the speed and accuracy of the draft model, and the number of tokens drafted per SD round (speculation length  $\gamma$ ). Larger draft models increase acceptance rates but reduce draft speed.

---

\*Equal contribution.

A larger  $\gamma$  increases the number of proposed tokens per round, which may lead to more accepted tokens, but it also increases the likelihood of rejection, potentially reducing overall efficiency. Early versions of SD relied on separate draft models with fixed  $\gamma$ , tuned offline. Later work explored adaptive  $\gamma$  policies (Zhang et al., 2024).

LayerSkip (Elhoushi et al., 2024) is a recent self-speculative decoding method that improves efficiency by removing the need for a separate draft model. Instead, it reuses the first  $\mathcal{E}$  layers of the main model to draft tokens and the remaining layers for verification, reducing both memory and compute overhead. Prior works have relied on static configuration of  $\mathcal{E}$  and  $\gamma$ , selected via offline grid search. This introduces two key limitations. First, the optimal  $\mathcal{E}$  and  $\gamma$  vary significantly across tasks; configurations tuned for one task often underperform on others. For example, using a language modeling-optimized configuration on a summarization task reduces speedup from  $2.65\times$  to  $1.50\times$ . Second, even within a single task, optimal configurations can shift during generation. In code generation, for instance, a setting that performs well for early tokens may degrade performance later in the sequence. These findings indicate that fixed configurations are suboptimal and motivate the need for a dynamic approach that adapts  $\mathcal{E}$  and  $\gamma$  throughout the generation process.

Coincidentally, we observe that LayerSkip is particularly well-suited for dynamic selection of  $\mathcal{E}$  and  $\gamma$  with minimal overhead. First, since any prefix of the model layers can serve as a draft model, LayerSkip naturally exposes a spectrum of sub-models defined by the choice of exit layer  $\mathcal{E}$ , with no overhead when switching between them during generation. Second, for each draft token, LayerSkip computes hidden states of the first  $\mathcal{E}$  model layers during the drafting stage, and then computes the remaining layers hidden states during verification in each SD round. Importantly, these intermediate hidden states computed can be reused to estimate the token acceptance rate under different choices of  $\mathcal{E}$ . Together, these two properties enable real-time adaptation of  $\mathcal{E}$  and  $\gamma$  during inference, guided by layer-wise acceptance rate estimates. To fully exploit this flexibility, however, one needs an algorithmic approach to adaptively select appropriate values for  $\mathcal{E}$  and  $\gamma$  at each SD round.

To address this challenge, we introduce DEL (Dynamic Exit Layer), an algorithmic approach that is implemented as a plug-and-play module for LayerSkip that adaptively selects  $\mathcal{E}$  and  $\gamma$  at each round of speculative decoding. DEL tracks the acceptance rate of draft tokens for each layer using cached hidden states, and uses this information to estimate a metric called Token-per-Layer (TPL), which reflects decoding efficiency. By selecting the configuration that maximizes TPL, DEL improves speed without sacrificing output quality. DEL also adjusts  $\gamma$  dynamically within each round using a context-aware confidence threshold to decide when to stop drafting.

In summary, our key contributions are:

- We conducted an empirical study on the impact of  $\mathcal{E}$  and  $\gamma$  in LayerSkip, revealing that the optimal settings of these values are model- and input-dependent, motivating the need for dynamic selection of these parameters during inference.
- Based on these observations, we propose DEL, a plug-and-play module for LayerSkip that adaptively selects  $\mathcal{E}$  and  $\gamma$  by evaluating cached hidden states and applying a dynamic, context-aware thresholding mechanism.
- Through extensive evaluations across tasks and model sizes, we show that DEL achieves up to  $2.62\times$  speedup over vanilla decoding and outperforms dynamic LayerSkip baselines, which peak at  $2.43\times$ , while maintaining output quality and incurring negligible overhead.

## 2 Background

**Auto-regressive decoding.** Given an input sequence  $\{x_0, \dots, x_{t-1}\}$ , a transformer-based auto-regressive language model  $\mathcal{M}_p$  generates the next token  $x_t$  from the conditional probability distribution  $p(x_t \mid x_{<t})$ . Specifically, each transformer decoder layer processes the hidden states  $h_t^{\ell-1}$  from the previous layer as follows:

$$h_t^\ell = \text{Transformer}_\ell(h_t^{\ell-1}), \quad \ell \in [1, L],$$

where  $h_t^0$  is the embedding representation of  $x_{t-1}$ . After the  $L$ -th layer of the model, the predicted token  $x_t$  is determined by the probability output from a softmax classifier:  $p(x_t) = \text{LM-Head}(h_t^L) = \text{softmax}(W^\top h_t^L)$ . We use the notation  $p(x_t)$  to denote  $p(x_t | x_{<t})$ .

**Speculative decoding.** The SD algorithm generates tokens in two phases: drafting and verification. During the drafting phase, a small and efficient draft model  $\mathcal{M}_q$  is used to sample  $\gamma$  candidate token distributions  $\{q(x_t), \dots, q(x_{t+\gamma-1})\}$ . In the verification phase, the input sequence concatenated with these  $\gamma$  drafted tokens is passed to the target model  $\mathcal{M}_p$ , which computes the probabilities  $\{p(x_t), \dots, p(x_{t+\gamma})\}$  in parallel. The verification step compares  $p(x_{t+i})$  and  $q(x_{t+i})$ , and accepts the drafted token with probability  $\min(1, p(x_{t+i})/q(x_{t+i}))$ .

If a token  $x_{t+i}$  is rejected before all  $\gamma$  tokens are accepted, the remaining draft tokens are discarded, and  $x_{t+i}$  is resampled from the residual distribution  $\max(0, p(x_{t+i}) - q(x_{t+i}))$ . Otherwise, if all drafted tokens are accepted, SD samples one additional token from  $p(x_{t+\gamma})$  and appends it to the sequence. Each round of SD thus generates at least one and at most  $\gamma + 1$  tokens. Leviathan et al. (2023) prove that the output sequence from SD follows the same distribution as that of the target model. In the special case of greedy decoding, the verification phase simply compares the top-1 prediction from  $p(x_{t+i})$  and  $q(x_{t+i})$ . A key advantage of SD is that the target model verifies all draft tokens in parallel, offering higher efficiency than generating tokens sequentially.

**Early-exit self-speculative decoding.** LayerSkip (Elhoushi et al., 2024) removes the need for a separate draft model  $\mathcal{M}_q$  by leveraging a subset of the full model as the draft model. During decoding, only the first  $\mathcal{E}$  layers are executed, and the output is passed directly to the LM-Head to generate a draft token. Once  $\gamma$  tokens are drafted auto-regressively, they are passed through all  $L$  layers of the model for verification. LayerSkip’s verification phase reuses the KV cache from the draft phase, resulting in lower memory usage and reduced computation compared to other SD approaches that rely on independent draft models. Nevertheless, the performance of LayerSkip is sensitive to the choice of  $\mathcal{E}$  and  $\gamma$ .

### 3 Motivating Observation

**Exit layer  $\mathcal{E}$  and speculation length  $\gamma$  are task-dependent.** We begin by investigating whether a single configuration of exit layer  $\mathcal{E}$  and speculation length  $\gamma$  can generalize across tasks. To this end, we perform a grid search to identify the optimal  $(\mathcal{E}, \gamma)$  for each task, selecting the configuration that maximizes decoding speed (measured in tokens per second). We then apply each task-specific configuration to all other tasks to assess the generalizability of these settings. Figure 1(a) shows that applying a configuration tuned for one task to another often degrades performance. For instance, using the language modeling configuration for summarization reduces speedup from  $2.65\times$  to  $1.50\times$ , while applying the summarization configuration to language modeling drops speedup from  $2.01\times$  to  $1.54\times$ .

**Even within a single task, optimal settings vary by prompt and generation stage.** To investigate whether optimal  $(\mathcal{E}, \gamma)$  settings remain stable across prompts within a task, or even across the course of generating output from a single prompt, we conduct a finer-grained analysis. We randomly sample two prompts from the coding task and divide the generation process into fixed-length segments of 64 tokens. For each segment, we run a separate grid search to determine the optimal configuration for that specific generation window. Figure 1(b) visualizes the results as heatmaps, where the first row of three figures correspond to the first prompt, and the second row corresponds to the second prompt. The columns within each row correspond to the heat map generated for a particular segment. The heatmaps reveal that the best-performing configuration shifts not only between prompts but also across different stages of token generation within the same prompt. Notably, the globally optimal configuration for the coding task,  $(\mathcal{E}=7, \gamma=6)$ , often fails to achieve the best performance at finer granularity, as several segments exhibit higher decoding speed with alternative settings.

These findings highlight a critical insight: the optimal exit layer  $\mathcal{E}$  and speculation length  $\gamma$  are not solely functions of the underlying task, and they also depend heavily on the specific prompt and the dynamic context during generation. In practice, the optimal configuration

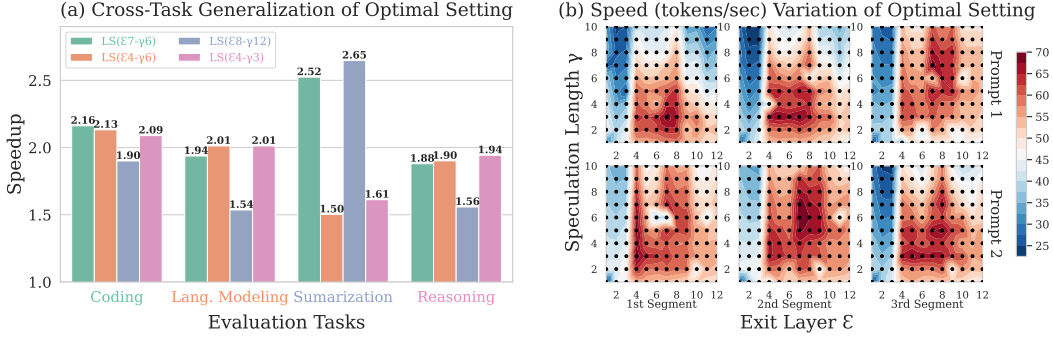


Figure 1: (a) Optimal  $(\mathcal{E}, \gamma)$  configurations of LayerSkip are task-specific; there is no universally best setting. The legend shows configurations tuned for each task, with colors matching the corresponding task label on the x-axis. Bars show the speedup achieved when applying each configuration to each task. Tasks perform best with their own configuration, while cross-task settings degrade performance. (b) Optimal  $(\mathcal{E}, \gamma)$  configurations vary across prompts and even within segments of a single prompt. Heatmaps show decoding speed over  $(\mathcal{E}, \gamma)$  grids. Shifting optimal regions motivate dynamic, context-aware adaptation.

often shifts throughout the generation process, even within a single prompt. This motivates our core contribution: a dynamic, context-aware approach that adaptively selects  $\mathcal{E}$  and  $\gamma$  at fine granularity for each stage of generation, in order to fully realize the potential efficiency of self-speculative decoding.

## 4 DEL

As shown in the prior section the token generation speed is a function of exit layer  $\mathcal{E}$  and speculation length  $\gamma$ . A draft model with a larger value of  $\mathcal{E}$  may increase the likelihood that a drafted token is accepted, but it also introduces higher latency for drafting each token. Similarly, a larger value of  $\gamma$  may reduce the total number of SD rounds but can also increase the number of redundant draft generations. Hence, one needs an appropriate token generation cost model to evaluate the trade-off introduced by  $\mathcal{E}$  and  $\gamma$ .

### 4.1 Defining TPL Metric

We introduce *Token-per-Layer (TPL)*, a metric that estimates the expected number of tokens generated per loaded layer in each SD round for the LayerSkip approach. TPL is defined as:

$$\text{TPL}(\ell, d) = \frac{\# \text{ generated tokens}}{\# \text{ loaded layers}} = \frac{1 - \alpha_\ell^{d+1}}{(1 - \alpha_\ell)(d\ell + L)} \quad (1)$$

where  $\alpha_\ell$  is the expected acceptance rate when using an exit layer  $\ell$  to draft the tokens. As described in prior work (Leviathan et al., 2023) if the probability that a token proposed by the draft model with  $\ell$  layers is accepted is  $\alpha_\ell$  then the expected number of newly generated tokens when using speculation length of  $d$  is given by  $\frac{1 - \alpha_\ell^{d+1}}{1 - \alpha_\ell}$ . Note that this equation is a simplification of  $\sum_{i=0}^d (\alpha_\ell)^i$ . We take this expected token generation length and divide that by the cost to generate these  $d$  tokens to create the TPL metric.

**Cost estimation.** While there are various approaches to measure the cost of generating tokens, we rely on an easily measurable approximation. In particular, we exploit a known observation that LLM token generation is heavily memory-bounded during inference, and the latency of generating a token is proportional to the number of transformer layers loaded into the GPU (Xia et al., 2024). Loading a model layer and forwarding multiple tokens for verification takes roughly the same amount of time as loading a model layer and forwarding a single token for drafting. By exploiting this observation, we estimate the cost of drafting

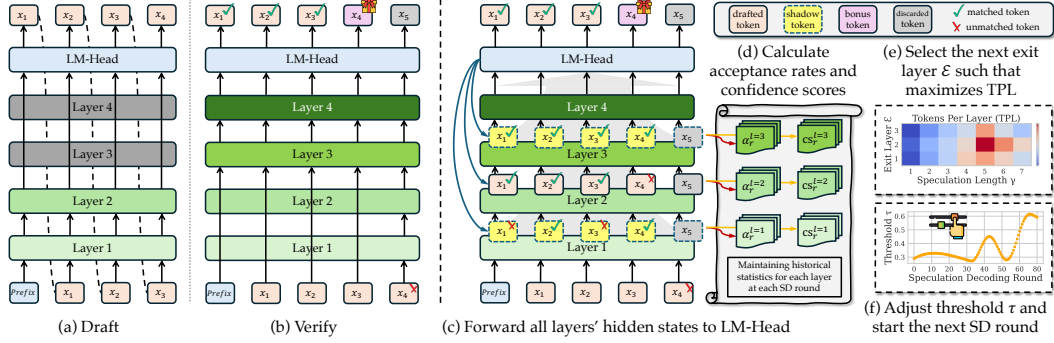


Figure 2: DEL Overview. (a–b) In this LayerSkip example, Layer 2 serves as the exit layer, drafting 4 tokens auto-regressively. During verification, 3 tokens are accepted, while  $x_4$  is rejected and becomes the bonus token. (c–d) The plug-and-play DEL module reuses the hidden states from all layers, forwarding them through the LM-Head to generate shadow tokens used to estimate layer-wise token acceptance rates and confidence scores without incurring additional forward passes. (e–f) DEL selects the next exit layer  $\mathcal{E}$  that maximizes Token-per-Layer (TPL) based on historical acceptance statistics, and updates the dynamic confidence threshold  $\tau$  to guide adaptive speculation length  $\gamma$  in the next SD round.

and verifying  $d$  tokens at each SD round with exit layer  $\ell$  as  $(d\ell + L)$ , where the draft model with  $\ell$  layers is loaded  $d$  times, and the target model with  $L$  layers is loaded once. This cost estimation is used to compute the TPL metric discussed above.

## 4.2 Modeling Expected Acceptance Rate

To accurately estimate TPL, as defined in Eq. (1), our methodology requires modeling the expected token acceptance rate  $\alpha_\ell$  for each layer. Different layers yield different values of  $\alpha_\ell$  and incur different drafting costs, leading to different TPL values. To enable efficient and adaptive estimation of  $\alpha_\ell$  in LayerSkip, we make the following key observation. One can directly utilize the hidden states produced in each SD round to efficiently compute the  $\alpha_\ell$  for each layer, as we discuss next.

**Shadow tokens generation.** Specifically, during the drafting and verification phases of LayerSkip (Figure 2(a) and (b)), we cache all generated hidden states across all layers, denoted as  $\{h_t^\ell, \dots, h_{t+\gamma}^\ell\}$ , for  $\ell \in [1, L]$ . As shown in Figure 2(c), we then forward these hidden states through the LM-head in a single pass and apply greedy decoding to obtain  $\{x_t^\ell, \dots, x_{t+\gamma}^\ell\}$ , producing  $L$  sets of tokens, one set for each layer. Each set represents the candidate tokens (referred to as *shadow tokens*) that would have been generated if the corresponding layer had been used as the draft model. These shadow tokens allow us to estimate the expected acceptance rate  $\alpha_\ell$  for each possible exit layer.

**$\alpha_\ell$  formulation.** Suppose layer  $E$  is used as the draft model during the drafting phase of SD round  $r$ . We compare the tokens  $\{x_t^E, \dots, x_{t+\gamma}^E\}$ , generated by the draft model, with  $\{x_t^L, \dots, x_{t+\gamma}^L\}$ , generated by the target model. We then identify the index of the first mismatched token:  $u_r = \min(\{i \in [0, \gamma] : x_{t+i}^E \neq x_{t+i}^L\})$ , if a mismatch exists; otherwise,  $u_r = \gamma$ . The index  $u_r$  indicates the point of divergence between the draft and target generations. Tokens beyond this point are discarded, as the context has diverged from the target model’s generation. For each layer  $\ell \in [1, L]$ , the number of matched shadow tokens within the valid context is computed as  $c_r^\ell = \sum_{i=0}^{u_r} \mathbb{I}(x_{t+i}^\ell = x_{t+i}^L)$ , where  $\mathbb{I}(\cdot)$  denotes the indicator function.

The above procedure provides  $c_r^\ell$  and  $u_r$  for the current SD round  $r$ . Our goal is to estimate  $\alpha_\ell$  by tracking this information over multiple past rounds. To achieve this, we define a weighted sum function  $S(a) = \sum_{i=0}^{r-1} \omega^{r-i-1} a[i]$ , where  $a \in \mathbb{R}^r$  is the input vector and



$\omega \in [0, 1]$  is a decay control factor. The expected acceptance rate is then defined as:

$$\alpha_\ell = \frac{S(\mathbf{c}^\ell)}{S(\mathbf{u})}, \quad \ell \in [1, L), \quad (2)$$

where  $\mathbf{c}^\ell, \mathbf{u} \in \mathbb{R}^r$  represents the number of matched shadow tokens and length of the valid context for layer  $\ell$  across SD rounds up to  $r$ , respectively. The expected acceptance rate  $\alpha_\ell$  is computed as a weighted ratio of matched tokens to total tokens, with recent rounds weighted more heavily. As a result,  $\alpha_\ell$  reflects how well the predictions from a given exit layer align with the target model over time.

### 4.3 Selecting Exit Layer Dynamically

With the estimated  $\alpha_\ell$ , we evaluate  $\text{TPL}(\ell, d)$  for all possible values of  $\ell \in [1, L)$  and  $d \in [0, d_{\max}]$ . The optimal setting for the next SD round is selected such that  $\text{TPL}(\ell=\mathcal{E}, d=\gamma)$  is maximized. To determine the initial values of  $\mathcal{E}$  and  $\gamma$  for the first SD round, we begin inference with the pre-filling step, where the target model generates initial hidden states for all layers over the prompt context to evaluate TPL.

While  $\mathcal{E}$  and  $\gamma$  are selected at the SD round granularity based on contextual information and historical statistics, the speculation length  $\gamma$  can still be suboptimal for individual tokens due to variation in token-level predictability within an SD round. To address this, we introduce a dynamic draft-exiting mechanism that allows the draft process to be shortened or extended based on the confidence of individual draft tokens, enabling finer-grained and more accurate control within each SD round.

### 4.4 Dynamic Draft-Exiting

The probability of the top-1 draft token prediction is used as a confidence score, representing the likelihood that the token will be accepted during verification (Du et al., 2024; Xia et al., 2025). In DEL, drafting proceeds for up to  $d_{\max}$  steps but may stop earlier if the confidence score falls below a threshold. While prior work uses a fixed threshold, this approach may not generalize well across prompts. Challenging prompts might require higher thresholds, while simpler ones may tolerate lower values, and the optimal threshold can even vary within a single prompt. To address this, we adopt a dynamic threshold that adjusts based on a context-aware update rule.

Let  $\{cs_t^\ell, \dots, cs_{t+\gamma}^\ell\}$  denote the confidence scores for the shadow tokens  $\{x_t^\ell, \dots, x_{t+\gamma}^\ell\}$  at layer  $\ell$ . During SD round  $r$ , the sum of confidence scores over matched tokens within the valid context from layer  $\ell$  is defined as  $tcs_r^\ell = \sum_{i=0}^{u_r} cs_{t+i}^\ell \cdot \mathbb{I}(x_{t+i}^\ell = x_{t+i}^L)$ . Similarly, the sum of confidence scores over mismatched tokens from layer  $\ell$  is  $fcs_r^\ell = \sum_{i=0}^{u_r} cs_{t+i}^\ell \cdot \mathbb{I}(x_{t+i}^\ell \neq x_{t+i}^L)$ . We then define the dynamic threshold as follows:

$$\tau_\ell = \frac{1}{2} \left( \frac{S(tcs^\ell)}{S(\mathbf{c}^\ell)} + \frac{S(fcs^\ell)}{S(\mathbf{u} - \mathbf{c}^\ell)} \right), \quad \ell \in [1, L), \quad (3)$$

where  $tcs^\ell, fcs^\ell \in \mathbb{R}^r$  denote the historical confidence score sums for matched and mismatched tokens across SD rounds up to  $r$ , and  $\mathbf{c}^\ell, \mathbf{u} - \mathbf{c}^\ell \in \mathbb{R}^r$  represent number of matched and mismatched tokens over the same rounds. Therefore, the dynamic threshold  $\tau_\ell$  is computed as the midpoint between the weighted average confidence scores of matched and mismatched tokens, with greater emphasis on recent rounds. We provide the summarization of the DEL method in Appendix A.

## 5 Experiments

### 5.1 Experimental Setup

**Implementation details.** We evaluate DEL on top of LayerSkip (Elhoushi et al., 2024) using LLaMA-2, LLaMA-3, and CodeLLaMA models across a range of tasks: CNN/DailyMail

Models	Methods	AQuA-RAT (Reasoning)		CNN/DM (Lang. Mod.)		CNN/DM (Abs. Sum.)		XSUM (Abs. Sum.)		Speed (tokens/s)	Overall Speedup
		eTPL	Speedup	eTPL	Speedup	eTPL	Speedup	eTPL	Speedup		
LLaMA -3.2-1B	Vanilla	0.063	1.00×	0.063	1.00×	0.063	1.00×	0.063	1.00×	64.34	1.00×
	LS( $\mathcal{E}3\text{-}\gamma6$ )	0.146	2.01×	0.154	2.11×	0.144	1.93×	0.139	1.89×	127.87	1.99×
	FS( $\mathcal{E}3\text{-}\gamma6$ )	0.158	2.16×	0.184	2.48×	0.159	2.13×	0.159	2.18×	144.21	2.24×
	DV( $\mathcal{E}3$ )	0.122	1.72×	0.138	1.93×	0.132	1.83×	0.099	1.43×	111.14	1.73×
	DEL	0.170	<b>2.24×</b>	0.198	<b>2.56×</b>	0.173	<b>2.25×</b>	0.183	<b>2.37×</b>	151.73	<b>2.36×</b>
LLaMA -3-8B	Vanilla	0.031	1.00×	0.031	1.00×	0.031	1.00×	0.031	1.00×	32.57	1.00×
	LS( $\mathcal{E}3\text{-}\gamma6$ )	0.084	2.17×	0.087	2.29×	0.089	2.30×	0.077	2.08×	72.00	2.21×
	FS( $\mathcal{E}3\text{-}\gamma6$ )	0.094	2.47×	0.102	2.76×	0.087	2.30×	0.080	2.18×	79.13	2.43×
	DV( $\mathcal{E}3$ )	0.052	1.52×	0.052	1.50×	0.085	2.32×	0.051	1.46×	55.40	1.70×
	DEL	0.098	<b>2.53×</b>	0.109	<b>2.81×</b>	0.105	<b>2.72×</b>	0.092	<b>2.40×</b>	85.20	<b>2.62×</b>
LLaMA -2-7B	Vanilla	0.031	1.00×	0.031	1.00×	0.031	1.00×	0.031	1.00×	36.08	1.00×
	LS( $\mathcal{E}7\text{-}\gamma6$ )	0.064	1.83×	0.064	1.91×	0.074	2.16×	0.070	2.09×	71.93	1.99×
	FS( $\mathcal{E}7\text{-}\gamma6$ )	0.074	2.17×	0.073	2.17×	0.080	2.33×	0.074	2.25×	80.48	2.23×
	DV( $\mathcal{E}7$ )	0.048	1.39×	0.049	1.44×	0.088	2.52×	0.075	2.20×	67.86	1.88×
	DEL	0.087	<b>2.41×</b>	0.083	<b>2.41×</b>	0.097	<b>2.75×</b>	0.084	<b>2.44×</b>	90.29	<b>2.50×</b>
LLaMA -2-13B	Vanilla	0.025	1.00×	0.025	1.00×	0.025	1.00×	0.025	1.00×	27.08	1.00×
	LS( $\mathcal{E}7\text{-}\gamma4$ )	0.053	1.99×	0.049	1.89×	0.048	1.72×	0.050	1.87×	50.68	1.87×
	FS( $\mathcal{E}7\text{-}\gamma4$ )	0.062	2.24×	0.054	2.06×	0.049	1.77×	0.051	1.89×	53.99	1.99×
	DV( $\mathcal{E}7$ )	0.040	1.48×	0.038	1.45×	0.038	1.36×	0.038	1.41×	38.68	1.43×
	DEL	0.063	<b>2.25×</b>	0.056	<b>2.11×</b>	0.067	<b>2.29×</b>	0.054	<b>1.98×</b>	58.39	<b>2.16×</b>
LLaMA -2-70B	Vanilla	0.013	1.00×	0.013	1.00×	0.013	1.00×	0.013	1.00×	9.75	1.00×
	LS( $\mathcal{E}9\text{-}\gamma6$ )	0.033	2.51×	0.026	1.98×	0.028	2.11×	0.026	1.99×	20.90	2.14×
	FS( $\mathcal{E}9\text{-}\gamma6$ )	0.036	2.76×	0.027	2.09×	0.029	2.22×	0.026	2.00×	22.08	2.26×
	DV( $\mathcal{E}9$ )	0.021	1.59×	0.019	1.50×	0.020	1.51×	0.019	1.51×	14.86	1.52×
	DEL	0.038	<b>2.84×</b>	0.028	<b>2.15×</b>	0.035	<b>2.57×</b>	0.028	<b>2.12×</b>	23.49	<b>2.41×</b>

Table 1: Performance comparison of DEL against static and dynamic speculative decoding baselines on text generation tasks. We report empirically observed Token-per-Layer (eTPL), per-task speedup, and overall decoding speed. The best results are bolded and the second-best are underlined. DEL consistently achieves the highest speedups by up to 2.84× across different models and tasks.

(CNN/DM) (Nallapati et al., 2016) for language modeling and summarization, XSUM (Narayan et al., 2018) for abstractive summarization, HumanEval (Chen et al., 2023) for code generation, and AQuA-RAT-CoT (Ling et al., 2017) for arithmetic reasoning.

Following prior work (Elhoushi et al., 2024; Xia et al., 2025; Zhou et al., 2024), we use 1-shot summarization for CNN/DM and 0-shot for XSUM. AQuA-RAT-CoT is evaluated in a 5-shot setting. The maximum generation length is set to 512 tokens. We randomly sample 1,000 test examples per dataset, except for HumanEval and AQuA-RAT, where full test sets are used. For HumanEval, we report both pass@1 and pass@10. Speculative sampling (Leviathan et al., 2023) is used with a batch size of 1. Our implementation builds on the codebase from Elhoushi et al. (2024).

**Baselines.** *Vanilla* refers to auto-regressive decoding without acceleration. *LS( $\mathcal{E}\text{-}\gamma$ )* uses LayerSkip with a fixed exit layer  $\mathcal{E}$  and static speculation length  $\gamma$ . *FS( $\mathcal{E}\text{-}\gamma$ )* is a LayerSkip variant with dynamic speculation. It initializes the speculation length to  $\gamma$ , then adjusts it using a finite-state controller: it increases by 1 if all draft tokens are accepted in the previous round, and decreases by 1 if any are rejected (Liu et al., 2025). *DV( $\mathcal{E}$ )* is another LayerSkip variant that applies the Draft & Verify method (Zhang et al., 2024), adjusting speculation length dynamically based on a confidence threshold to maintain a target acceptance rate. For all baselines,  $\mathcal{E}$  and  $\gamma$  are selected via grid search to maximize overall speedup. Detailed experimental setups and results are provided in Appendix B and C.

**Evaluation metrics.** We report the empirically observed Token-per-Layer (eTPL), defined as the total number of generated tokens divided by the number of loaded layers, averaged across all prompts. We also report decoding speed (tokens/s) and wall-time speedup, computed as the relative increase in average inference throughput compared to the auto-regressive baseline under the same settings (Zhang et al., 2024). For output quality, we use ROUGE-2 (Ganesan, 2018), implemented via the TorchMetrics library (Detlefsen et al., 2022)

Models	CodeLLaMA-7B					Methods	CodeLLaMA-34B				
	HumanEval (pass@1)		HumanEval (pass@10)				HumanEval (pass@1)		HumanEval (pass@10)		
	eTPL	Speedup	eTPL	Speedup	ROUGE-2		eTPL	Speedup	eTPL	Speedup	ROUGE-2
Vanilla	0.031	1.00×	0.031	1.00×	0.093	Vanilla	0.021	1.00×	0.021	1.00×	0.096
LS( $\mathcal{E}7\text{-}\gamma6$ )	0.063	1.95×	0.054	1.60×	0.094	LS( $\mathcal{E}9\text{-}\gamma6$ )	0.042	1.86×	0.038	1.64×	0.095
FS( $\mathcal{E}7\text{-}\gamma6$ )	0.066	2.05×	0.057	1.70×	0.093	FS( $\mathcal{E}9\text{-}\gamma6$ )	0.045	1.99×	0.039	1.70×	0.093
DV( $\mathcal{E}7$ )	0.069	2.08×	0.048	1.43×	0.092	DV( $\mathcal{E}9$ )	0.042	1.86×	0.032	1.43×	0.099
DEL	0.077	2.26×	0.062	1.73×	0.093	DEL	0.051	2.12×	0.043	1.79×	0.101

Table 2: Comparison of DEL and baselines on code generation tasks using CodeLLaMA-7B and 34B. Pass@1 uses greedy decoding; pass@10 uses sampling with temperature 0.6. We report empirical Token-per-Layer (eTPL), speedup, and ROUGE-2. Best results are bolded; second-best underlined. DEL reliably achieves the highest speedups across both decoding methods. The results indicate consistent output fidelity across methods, including DEL.

## 5.2 Overall Speedup Results

Table 1 shows that DEL consistently outperforms both static (LS) and dynamic (FS, DV) baselines across LLaMA model sizes and text generation tasks. DEL achieves a speedup of up to  $2.84\times$  on the AQuA-RAT task with the LLaMA-2-70B model and delivers overall speedups of  $2.16\times$  to  $2.62\times$  across all models and tasks over vanilla auto-regressive decoding. On LLaMA-2-7B, DEL reaches an overall speedup of  $2.50\times$ , exceeding the best baseline (FS) at  $2.23\times$  by  $0.27\times$  (a 12% speed gain). These gains are driven by higher eTPL, reflecting more efficient layer utilization. DEL maintains strong performance across diverse tasks, including reasoning, summarization, and language modeling, and generalizes well from 1B to 70B models, demonstrating scalability across different tasks and model sizes.

Table 2 reports results on code generation with CodeLLaMA-7B and 34B. DEL achieves the highest speedups under both greedy (pass@1) and sampling-based (pass@10) decoding, reaching up to  $2.26\times$  on 7B and  $2.12\times$  on 34B. In all cases, it obtains the highest eTPL, confirming its token-level efficiency. For greedy decoding, DEL produces exact token matches with the target model, ensuring identical outputs. In the sampling setting, speculative decoding theoretically preserves the target model’s output distribution, which is supported by the ROUGE-2 scores in Table 2. These results confirm that DEL maintains output fidelity, scales effectively with model size, and remains robust across different decoding strategies.

## 5.3 In-depth Analysis

**Ablation study of DEL.** DEL includes two key components: *dynamic exit layer selection* and *dynamic draft-exiting*. To assess their individual contributions, we conduct an ablation study on LLaMA2-7B across all tasks (Figure 3(a)). We evaluate three variants: **DEL- $\mathcal{E}$**  disables dynamic draft-exiting and selects both the exit layer and speculation length solely based on the dynamic exit layer selection mechanism (i.e., optimized for TPL). **DEL- $\gamma$**  fixes the exit layer and applies dynamic draft-exiting to adapt the speculation length  $\gamma$  at runtime. The fixed exit layer in DEL- $\gamma$  is chosen based on the best-performing baseline setting from Table 1. Full DEL, which combines both components, achieves higher speedup than either variant alone, confirming that both contribute to performance gains. **DEL- $\mathcal{E}$ -lim** enables both components, but restricts exit layers to  $3 < \ell < 32$ . The rationale for this study is to explore potential missed opportunities when exit layers are constrained. Even early layers (i.e., layers less than 3) can sometimes yield efficient drafts, and restricting their use reduces potential gains. Thus full flexibility in dynamically selecting exit layers is beneficial.

**Runtime and memory breakdown.** Recall that DEL forwards cached hidden states from all layers through the LM-Head to compute TPL. We quantify the runtime and memory overheads of this additional computation. Figure 3(b) shows the runtime breakdown for different speculation lengths  $\gamma$  on LLaMA-2-7B with  $\mathcal{E}=8$ . The reported runtime is measured after the drafting, verification, and the DEL module has completed processing the corresponding  $\gamma$  tokens. In terms of memory, DEL adds only  $0.52\%\sim 2.26\%$  over vanilla decoding, with minimal overhead compared to LayerSkip’s  $0.50\%\sim 1.42\%$ . These results show DEL is lightweight and easy to integrate. More details are provided in Appendix D.1.



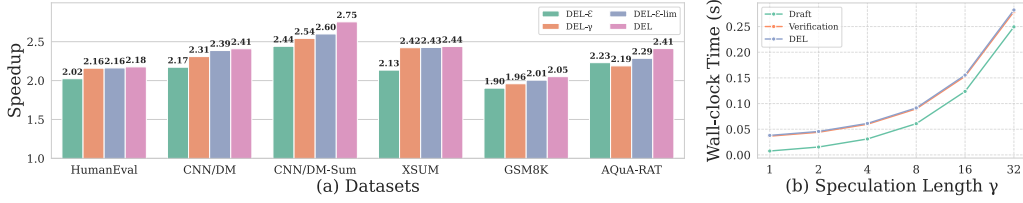


Figure 3: (a) Ablation on LLaMA2-7B shows both dynamic exit layer selection and dynamic draft-exiting components improve performance. (b) Runtime breakdown across  $\gamma$  shows DEL adds minimal overhead.

**Evolution of confidence threshold and exit layer.** Figure 4(a) shows how the confidence threshold  $\tau$  evolves over 200 SD rounds across different models and datasets. DEL adapts  $\tau$  over time, with noticeable variation, showing the importance of using a dynamic rather than fixed threshold. Figure 4(b) visualizes  $\tau$  for a sample prompt. The light blue cloud shows the range between confidence scores of matched (top) and mismatched (bottom) shadow tokens. The wide gap makes the midpoint a meaningful threshold for dynamic draft exiting. Figure 4(c) shows how DEL selects the exit layer  $\mathcal{E}$  and tracks acceptance rate  $\alpha$  during inference. DEL updates  $\mathcal{E}$  across rounds to keep  $\alpha$  high and maintain efficiency.

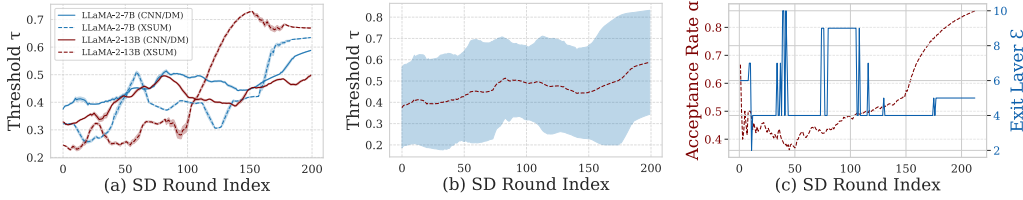


Figure 4: (a) Confidence threshold  $\tau$  evolves over 200 SD rounds across models and datasets. (b)  $\tau$  for a sample prompt, with accepted (top) and rejected (bottom) token scores shown as a shaded range. (c) Evolution of selected exit layer  $\mathcal{E}$  and acceptance rate  $\alpha$  during inference.

## 6 Related Works

Early studies on speculative decoding (Leviathan et al., 2023; Chen et al., 2023) introduced a rejection sampling method that preserves the target model’s distribution and maintains output quality. Recent work has focused on identifying effective stopping criteria by monitoring confidence scores to trigger verification. Kangaroo (Liu et al., 2024a) halts speculation when the draft model’s confidence falls below a fixed threshold, while AdaEDL (Agrawal et al., 2024) estimates a lower bound on token acceptance rate using draft logit entropy. SpecDec++ (Huang et al., 2024) augments the draft model with a trained prediction head to estimate token acceptance rate and dynamically adjust speculation length. CaPE (Du et al., 2024) and SWIFT (Xia et al., 2025) improve speculation accuracy by using confidence scores to expand draft sequences with informative tokens. However, these methods rely on fixed thresholds that do not adapt to varying prompts or tasks. Draft & Verify (Zhang et al., 2024) addresses this limitation by proposing an adaptive, feedback-based thresholding method to enforce a fixed acceptance rate, but it requires extensive hyperparameter tuning.

Self-speculative decoding methods eliminate the overhead of serving two models by using a subset or modified version of the target model to generate draft tokens. Draft & Verify (Zhang et al., 2024), SWIFT (Xia et al., 2025), and Draft on the Fly (Metel et al., 2024) reduce draft generation time by selecting intermediate layers based on Bayesian optimization. Kangaroo (Liu et al., 2024a) uses a shallow sub-network as the draft model, and a lightweight adapter module is trained on it to align its representations with the full model. LayerSkip (Elhoushi et al., 2024) increases early exit accuracy without auxiliary modules through a

specialized training strategy. By performing verification with the remaining layers, it reduces memory usage and benefits from shared computation between drafting and verification. The most relevant approach to ours is S3D (Zhong & Bharadwaj, 2024), which reduces speculation cost by selecting how many mid-layers to skip, but its time-consuming offline training makes it neither plug-and-play nor easily adaptable to different models and tasks. Like prior speculative decoding methods, DEL assumes sufficient alignment between the draft and target distributions; recent works have shown that under out-of-distribution prompts, performance may degrade due to reduced acceptance rates (Hong et al., 2025; Goel et al., 2024; Liu et al., 2024b).

## 7 Conclusion

We introduced DEL, a plug-and-play module for LayerSkip that dynamically selects the exit layer and speculation length to maximize decoding efficiency. DEL tracks token acceptance rates across layers and estimates a Token-per-Layer (TPL) metric, leveraging context-aware feedback to identify the optimal configuration for each speculative decoding round. Extensive evaluations across models and tasks show DEL achieves overall speedups of  $2.16\times\sim 2.62\times$  over auto-regressive decoding. DEL’s lightweight design enables seamless integration with existing SD pipelines, enhancing performance without sacrificing quality.

## Acknowledgment

We sincerely thank all the reviewers for their time and constructive comments. This material is based upon work supported by NSF award number 2224319, REAL@USC-Meta center, and VMware gift. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

## References

- Sudhanshu Agrawal, Wonseok Jeon, and Mingu Lee. Adaedl: Early draft stopping for speculative decoding of large language models via an entropy-based lower bound on token acceptance probability. *arXiv preprint arXiv:2410.18351*, 2024.
- Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.
- Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL <https://arxiv.org/abs/2302.01318>.
- 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. Training verifiers to solve math word problems, 2021. URL <https://arxiv.org/abs/2110.14168>.
- Nicki Skafted Detlefsen, Jiri Borovec, Justus Schock, Ananya Harsh Jha, Teddy Koker, Luca Di Liello, Daniel Stancl, Changsheng Quan, Maxim Grechkin, and William Falcon. Torchmetrics-measuring reproducibility in pytorch. *Journal of Open Source Software*, 7(70):4101, 2022.
- Cunxiao Du, Jing Jiang, Xu Yuanchen, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, et al. Glide with a cape: A low-hassle method to accelerate speculative decoding. *arXiv preprint arXiv:2402.02082*, 2024.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

- Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.
- Kavita Ganesan. Rouge 2.0: Updated and improved measures for evaluation of summarization tasks. *arXiv preprint arXiv:1803.01937*, 2018.
- Raghavv Goel, Mukul Gagrani, Wonseok Jeon, Junyoung Park, Mingu Lee, and Christopher Lott. Direct alignment of draft model for speculative decoding with chat-fine-tuned llms. *arXiv preprint arXiv:2403.00858*, 2024.
- Fenglu Hong, Ravi Raju, Jonathan Lingjie Li, Bo Li, Urmish Thakker, Avinash Ravichandran, Swayambhoo Jain, and Changran Hu. Training domain draft models for speculative decoding: Best practices and insights. *arXiv preprint arXiv:2503.07807*, 2025.
- Kaixuan Huang, Xudong Guo, and Mengdi Wang. Specdec++: Boosting speculative decoding via adaptive candidate lengths. *arXiv preprint arXiv:2405.19715*, 2024.
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.
- Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 158–167, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1015. URL <https://aclanthology.org/P17-1015/>.
- Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding via double early exiting. *arXiv preprint arXiv:2404.18911*, 2024a.
- Jiesong Liu, Brian Park, and Xipeng Shen. A drop-in solution for on-the-fly adaptation of speculative decoding in large language models, 2025. URL <https://openreview.net/forum?id=x0t0fdbBqK>.
- Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. In *Proceedings of the 41st International Conference on Machine Learning, ICML’24*. JMLR.org, 2024b.
- Michael R. Metel, Peng Lu, Boxing Chen, Mehdi Rezagholizadeh, and Ivan Kobyzev. Draft on the fly: Adaptive self-speculative decoding using cosine similarity. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 2267–2272, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.124. URL <https://aclanthology.org/2024.findings-emnlp.124/>.
- Microsoft. Copilot. <https://copilot.microsoft.com/>, 2023. Accessed: 2025-02-05.
- Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In Stefan Riezler and Yoav Goldberg (eds.), *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 280–290, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1028. URL <https://aclanthology.org/K16-1028/>.
- Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don’t give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun’ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1797–1807, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1206. URL <https://aclanthology.org/D18-1206/>.

- OpenAI. Chatgpt. <https://chat.openai.com/>, 2022. Accessed: 2025-02-05.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, and et al. Gpt-4 technical report, 2024. URL <https://arxiv.org/abs/2303.08774>.
- Elizabeth Reid. Supercharging search with generative ai. <https://blog.google/products/search/generative-ai-search/>, 2023. Accessed: 2025-02-05.
- Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024. URL <https://arxiv.org/abs/2308.12950>.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023a.
- Yiran Wu, Feiran Jia, Shaokun Zhang, Hangyu Li, Erkang Zhu, Yue Wang, Yin Tat Lee, Richard Peng, Qingyun Wu, and Chi Wang. An empirical study on challenging math problem solving with gpt-4. *arXiv preprint arXiv:2306.01337*, 2023b.
- Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7655–7671, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.456. URL <https://aclanthology.org/2024.findings-acl.456/>.
- Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. SWIFT: On-the-fly self-speculative decoding for LLM inference acceleration. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=EKJhH5D5wA>.
- Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.607. URL <https://aclanthology.org/2024.acl-long.607/>.
- Wei Zhong and Manasa Bharadwaj. S3d: A simple and cost-effective self-speculative decoding scheme for low-memory gpus, 2024. URL <https://arxiv.org/abs/2405.20314>.
- Yang Zhou, Zhuoming Chen, Zhaozhuo Xu, Xi Victoria Lin, and Beidi Chen. SIRIUS: Contextual sparsity with correction for efficient LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://openreview.net/forum?id=5bR211b2eh>.

## A Algorithm

Algorithm 1 summarizes self-speculative decoding with DEL, where  $\mathcal{E}$  and  $\gamma$  are dynamically adjusted to maximize efficiency and optimize token generation.

---

### Algorithm 1 Self-Speculative Decoding with DEL

---

**Require:**

- 1: Target model  $\mathcal{M}_p$  with  $L$  layers
  - 2: Initialize hidden states, exit layer  $\mathcal{E}$ , and threshold  $\tau$  by  $\mathcal{M}_p$  pre-filling
  - 3:  $r \leftarrow 0$  ▷ Initialize SD round counter
  - 4:  $c, u, tcs, fcs \leftarrow []$  ▷ Lists to store DEL statistics
  
  - 5: **while** tokens remain to be generated **do**
  - 6:  $\mathcal{D} \leftarrow []$  ▷ Initialize token list
  - 7: **for**  $i = 0$  to  $d_{\max} - 1$  **do** ▷ Draft tokens
  - 8: Draft token  $x_{t+i}$  using the first  $\mathcal{E}$  layers with confidence score  $s$
  - 9: **if**  $s < \tau$  **then**
  - 10: **Break** ▷ Early exit if low confidence
  - 11: **end if**
  - 12: Append  $x_{t+i}$  to  $\mathcal{D}$
  - 13: **end for**
  - 14: Verify  $\mathcal{D}$  by  $\mathcal{M}_p$
  - 15: DEL\_UPDATE( $r$ )
  - 16:  $r \leftarrow r + 1$
  - 17: **end while**
  
  - 18: **function** DEL\_UPDATE( $r$ )
  - 19:  $\{(x_t^\ell, cs_t^\ell), \dots, (x_{t+\gamma}^\ell, cs_{t+\gamma}^\ell)\} \leftarrow \text{LM-Head}(\{h_t^\ell, \dots, h_{t+\gamma}^\ell\}), \ell \in [1, L]$
  - 20:  $u_r = \min(\{i \in [0, \gamma] : x_{t+i}^\ell \neq x_{t+i}^L\})$  if mismatch, else  $u_r = \gamma$
  - 21:  $c_r^\ell = \sum_{i=0}^{u_r} \mathbb{I}(x_{t+i}^\ell = x_{t+i}^L)$  ▷ Count of matched tokens in valid context
  - 22: Append  $c_r^\ell$  to  $c$  and  $u_r$  to  $u$
  - 23:  $\alpha_\ell = \frac{S(c^\ell)}{S(u)}$  ▷ Calculate acceptance rate
  - 24:  $(\mathcal{E}, \gamma) = \arg \max_{\ell \in [1, L], d \in [0, d_{\max}]} \text{TPL}(\ell, d)$
  - 25:  $tcs_r^\ell = \sum_{i=0}^{u_r} cs_{t+i}^\ell \mathbb{I}(x_{t+i}^\ell = x_{t+i}^L)$  ▷ Compute true confidence scores
  - 26:  $fcs_r^\ell = \sum_{i=0}^{u_r} cs_{t+i}^\ell \mathbb{I}(x_{t+i}^\ell \neq x_{t+i}^L)$  ▷ Compute false confidence scores
  - 27: Append  $tcs_r^\ell$  to  $tcs$  and  $fcs_r^\ell$  to  $fcs$
  - 28:  $\tau_\ell = \frac{1}{2} \left( \frac{S(tcs^\ell)}{S(c^\ell)} + \frac{S(fcs^\ell)}{S(u - c^\ell)} \right)$  ▷ Dynamic threshold update
  - 29: Set updated exit layer  $\mathcal{E}$  and threshold  $\tau$  for the next SD round
  - 30: **end function**
- 

## B Detailed Experimental Setups

### B.1 Models and Datasets

We evaluate DEL on LLaMA-2, LLaMA-3, and CodeLLaMA models (Touvron et al., 2023; Dubey et al., 2024; Rozière et al., 2024), finetuned for LayerSkip by Elhoushi et al. (2024). Our experiments cover various generation tasks, including language modeling, summarization, arithmetic reasoning, code generation, and translation. For language modeling and summarization, we use CNN/DailyMail (CNN/DM) (Nallapati et al., 2016), while XSUM (Narayan et al., 2018) is used for abstractive summarization. HumanEval (Chen et al., 2023) is used for code generation, and GSM8K (Cobbe et al., 2021) and AQuA-RAT-CoT (Ling et al., 2017) are used for arithmetic reasoning. To cover multilingual generation, we evaluate on the WMT14 De-En dataset (German to English translation). Following prior work (Elhoushi et al., 2024;



Xia et al., 2025; Zhou et al., 2024), we use 1-shot summarization for CNN/DM and 0-shot for XSUM. GSM8K and AQuA-RAT-CoT are evaluated in a 5-shot setting. HumanEval is evaluated on CodeLLaMA-7B and 34B using pass@1 (greedy decoding) and pass@10 (random sampling), with pass@10 results averaged over 10 runs. The maximum generation length is set to 512 tokens. We randomly sample 1,000 test examples per dataset, except for HumanEval and AQuA-RAT, where the full test sets are used. Prompt shots for CNN/DM and GSM8K are randomly sampled from their respective training sets, while AQuA-RAT uses the specific CoT prompt shots provided by Zhou et al. (2024).

## B.2 Inference Setup

To evaluate DEL, we used specific hyperparameter settings. For a given input prompt, DEL initializes  $\alpha_\ell$  by pre-filling and considers the last 32 tokens of the context to determine the optimal exit layer  $\mathcal{E}$  and speculation length  $\gamma$  for the first initial SD round. We apply a decay factor of  $\omega = 0.95$  in the weighted sum function in Section 4.2, to gradually forget information from previous SD rounds. For instance,  $0.95^{13} \simeq 0.5$  indicates that information from 13 rounds prior is given half the importance of the current round. DEL restricts the maximum speculation length to  $d_{\max} = 18$  tokens. For code generation tasks using random sampling, we set the temperature to 0.6 and *top-p* to 0.95. All experiments were conducted using PyTorch 2.2.1. Models LLaMA-3.2-1B, LLaMA-2-7B, LLaMA-2-13B, and CodeLLaMA-7B were run on a single NVIDIA A100 (SXM4-40GB) GPU with CUDA 12.8 and 16 CPU cores from an AMD EPYC 7H12 64-Core processor. CodeLLaMA-34B used two instances of the same GPU, while LLaMA-70B ran on two NVIDIA H100 (PCIe-80GB) GPUs with the same CUDA version and 16 CPU cores from an Intel Xeon 6548Y processor. To eliminate hardware variability, we ensured that the same GPU/CPU instances were used consistently for each model-dataset pair. We adopted speculative sampling (Leviathan et al., 2023) as the acceptance strategy with a batch size of 1. Our implementation is based on the codebase from Elhoushi et al. (2024).

## B.3 Baselines

Vanilla refers to standard auto-regressive decoding without acceleration. LS( $\mathcal{E}$ - $\gamma$ ) applies LayerSkip with a fixed exit layer  $\mathcal{E}$  and a static speculation length  $\gamma$ . This baseline has two kinds of configurations. LS( $\mathcal{E}$ - $\gamma$ )<sup>†</sup> uses configurations that are proposed by the original LayerSkip paper, if available. Specifically, ( $\mathcal{E}=8$ ,  $\gamma=6$ ) is applied for HumanEval generation on LLaMA-2-7B, ( $\mathcal{E}=8$ ,  $\gamma=12$ ) for CNN/DM 1-shot and XSUM summarization on LLaMA-2-7B, ( $\mathcal{E}=7$ ,  $\gamma=4$ ) for HumanEval on LLaMA-2-13B, and ( $\mathcal{E}=15$ ,  $\gamma=4$ ) for XSUM summarization. The second configuration kind, LS( $\mathcal{E}$ - $\gamma$ ), determines the exit layer and speculation length by conducting a grid search over 10 randomly sampled inputs from a calibration dataset, with a maximum generation length of 256 tokens. Each configuration in the grid search space, consisting of potentially effective combinations of exit layers and speculation lengths for the underlying model, is evaluated on these 10 samples. This process identifies the most effective configurations for the given model and task. FS( $\mathcal{E}$ - $\gamma$ ) is a LayerSkip variant with a fixed exit layer  $\mathcal{E}$  and dynamic speculation length. It starts with an initial  $\gamma$  and adjusts it using a finite-state controller: increasing  $\gamma$  by 1 if all draft tokens are accepted in the previous round and decreasing it by 1 if any are rejected (Liu et al., 2025). DV( $\mathcal{E}$ ) is another LayerSkip variant that maintains a fixed exit layer  $\mathcal{E}$  but dynamically adjusts the speculation length using the Draft & Verify strategy (Zhang et al., 2024), where the speculation length is adapted based on a confidence threshold to maintain a target acceptance rate.

## B.4 Evaluation Metrics

We use the following metrics to evaluate the performance of DEL. eTPL: empirically observed Token-per-Layer is defined as the total number of generated (verified) tokens divided by the number of loaded layers, averaged across all prompts. For each prompt, we count the total number of loaded layers during generation and compute eTPL by dividing the number of generated tokens by this value. The final reported eTPL is the average across all prompts in the test set. Speed (tokens/s): Speed is defined as the total number of generated

(verified) tokens divided by the total time (seconds) taken to generate the output for a given prompt. This value is computed for each prompt, and the reported speed is the average across all test samples. Speedup: This refers to the ratio of the decoding speed achieved by the given method to that of standard auto-regressive (Vanilla) decoding.

## C Extended Exeprienmental Results

We present the detailed statistics of our main experimental results in Table 3. DEL consistently outperforms both static (LS) and dynamic (FS, DV) baselines across a range of LLaMA model sizes and text generation tasks. On LLaMA-2-70B, DEL achieves a remarkable speedup of  $2.84\times$  on AQuA-RAT (Reasoning), outperforming the best FS configuration at  $2.78\times$ . For CNN/DM (Abstractive Summarization) on the same model, DEL attains a speedup of  $2.57\times$ , while the strongest baseline, FS( $\mathcal{E}9\text{-}\gamma6$ ), achieves only  $2.22\times$ .

Methods	AQuA-RAT (Reasoning)		CNN/DM (Lang. Mod.)		CNN/DM (Abs. Sum.)		GSM8K (Reasoning)		HumanEval (Coding)		XSUM (Abs. Sum.)		WMT14 De-En (Translation)	
	Speed	Speedup	Speed	Speedup	Speed	Speedup	Speed	Speedup	Speed	Speedup	Speed	Speedup	Speed	Speedup
LLaMA-3.2-1B														
Vanilla	63.17	1.00 $\times$	65.15	1.00 $\times$	64.23	1.00 $\times$	63.91	1.00 $\times$	64.84	1.00 $\times$	64.83	1.00 $\times$	63.92	1.00 $\times$
LS( $\mathcal{E}3\text{-}\gamma3$ )	118.30	1.87 $\times$	124.01	1.90 $\times$	116.75	1.82 $\times$	115.52	1.81 $\times$	120.09	1.85 $\times$	116.63	1.80 $\times$	119.23	1.87 $\times$
LS( $\mathcal{E}3\text{-}\gamma6$ )	127.09	2.01 $\times$	137.65	2.11 $\times$	124.23	1.93 $\times$	117.03	1.83 $\times$	126.16	1.95 $\times$	122.49	1.89 $\times$	127.63	2.00 $\times$
LS( $\mathcal{E}4\text{-}\gamma3$ )	108.31	1.71 $\times$	115.48	1.77 $\times$	107.13	1.67 $\times$	103.97	1.63 $\times$	109.44	1.69 $\times$	108.26	1.67 $\times$	108.74	1.70 $\times$
FS( $\mathcal{E}3\text{-}\gamma3$ )	138.71	2.20 $\times$	163.12	2.50 $\times$	137.26	2.14 $\times$	132.43	2.07 $\times$	134.68	2.08 $\times$	141.89	2.19 $\times$	150.91	2.36 $\times$
FS( $\mathcal{E}3\text{-}\gamma6$ )	136.59	2.16 $\times$	161.86	2.48 $\times$	137.05	2.13 $\times$	130.98	2.05 $\times$	135.05	2.08 $\times$	141.34	2.18 $\times$	151.97	2.38 $\times$
FS( $\mathcal{E}4\text{-}\gamma3$ )	121.20	1.92 $\times$	142.35	2.18 $\times$	121.66	1.89 $\times$	117.15	1.83 $\times$	119.74	1.85 $\times$	124.56	1.92 $\times$	132.81	2.08 $\times$
DV( $\mathcal{E}3$ )	108.88	1.72 $\times$	125.72	1.93 $\times$	117.51	1.83 $\times$	90.10	1.41 $\times$	107.83	1.66 $\times$	92.43	1.43 $\times$	94.27	1.47 $\times$
DV( $\mathcal{E}4$ )	104.71	1.66 $\times$	118.65	1.82 $\times$	113.27	1.76 $\times$	85.05	1.33 $\times$	104.49	1.61 $\times$	88.85	1.37 $\times$	91.62	1.43 $\times$
DEL	141.49	<b>2.24<math>\times</math></b>	167.00	<b>2.56<math>\times</math></b>	144.78	<b>2.25<math>\times</math></b>	135.74	<b>2.12<math>\times</math></b>	140.70	<b>2.17<math>\times</math></b>	153.63	<b>2.37<math>\times</math></b>	151.11	<b>2.36<math>\times</math></b>
LLaMA-3-8B														
Vanilla	32.85	1.00 $\times$	32.57	1.00 $\times$	32.68	1.00 $\times$	31.90	1.00 $\times$	31.65	1.00 $\times$	32.16	1.00 $\times$	32.29	1.00 $\times$
LS( $\mathcal{E}3\text{-}\gamma3$ )	65.48	1.99 $\times$	67.77	2.08 $\times$	67.81	2.07 $\times$	56.51	1.77 $\times$	63.62	2.01 $\times$	62.13	1.93 $\times$	66.71	2.07 $\times$
LS( $\mathcal{E}3\text{-}\gamma6$ )	71.45	2.17 $\times$	74.57	2.29 $\times$	75.22	2.30 $\times$	54.42	1.71 $\times$	71.23	2.25 $\times$	66.75	2.08 $\times$	74.74	2.31 $\times$
LS( $\mathcal{E}8\text{-}\gamma6$ )	57.11	1.74 $\times$	57.90	1.78 $\times$	68.54	2.10 $\times$	47.50	1.49 $\times$	56.91	1.80 $\times$	57.61	1.79 $\times$	58.57	1.81 $\times$
FS( $\mathcal{E}3\text{-}\gamma3$ )	81.95	2.49 $\times$	88.74	2.72 $\times$	74.73	2.29 $\times$	62.23	1.95 $\times$	76.16	2.41 $\times$	69.96	2.18 $\times$	85.13	2.64 $\times$
FS( $\mathcal{E}3\text{-}\gamma6$ )	81.22	2.47 $\times$	89.79	2.76 $\times$	75.33	2.30 $\times$	61.15	1.92 $\times$	75.87	2.40 $\times$	70.19	2.18 $\times$	84.49	2.62 $\times$
FS( $\mathcal{E}8\text{-}\gamma6$ )	66.51	2.02 $\times$	68.36	2.10 $\times$	73.89	2.26 $\times$	56.34	1.77 $\times$	62.58	1.98 $\times$	61.86	1.92 $\times$	67.85	2.10 $\times$
DV( $\mathcal{E}3$ )	49.80	1.52 $\times$	49.00	1.50 $\times$	75.73	2.32 $\times$	44.69	1.40 $\times$	47.97	1.52 $\times$	47.06	1.46 $\times$	49.22	1.52 $\times$
DV( $\mathcal{E}8$ )	45.16	1.37 $\times$	47.29	1.45 $\times$	79.79	2.44 $\times$	42.53	1.33 $\times$	62.90	1.99 $\times$	62.45	1.94 $\times$	48.83	1.51 $\times$
DEL	83.08	<b>2.53<math>\times</math></b>	91.56	<b>2.81<math>\times</math></b>	88.97	<b>2.72<math>\times</math></b>	64.43	<b>2.02<math>\times</math></b>	78.41	<b>2.48<math>\times</math></b>	77.21	<b>2.40<math>\times</math></b>	88.23	<b>2.73<math>\times</math></b>
LLaMA-2-7B														
Vanilla	37.08	1.00 $\times$	36.05	1.00 $\times$	35.61	1.00 $\times$	36.31	1.00 $\times$	35.98	1.00 $\times$	35.59	1.00 $\times$	35.37	1.00 $\times$
LS( $\mathcal{E}7\text{-}\gamma6$ )	67.99	1.83 $\times$	68.71	1.91 $\times$	76.77	2.16 $\times$	58.76	1.62 $\times$	68.00	1.89 $\times$	74.24	2.09 $\times$	71.66	2.01 $\times$
LS( $\mathcal{E}8\text{-}\gamma6$ ) <sup>†</sup>	65.65	1.77 $\times$	65.94	1.83 $\times$	77.96	2.19 $\times$	59.83	1.65 $\times$	66.81	1.86 $\times$	70.55	1.98 $\times$	68.27	1.92 $\times$
LS( $\mathcal{E}8\text{-}\gamma12$ ) <sup>†</sup>	57.27	1.54 $\times$	58.22	1.61 $\times$	78.67	2.21 $\times$	48.64	1.34 $\times$	59.12	1.64 $\times$	67.65	1.90 $\times$	62.72	1.76 $\times$
FS( $\mathcal{E}7\text{-}\gamma6$ )	80.50	2.17 $\times$	78.40	2.17 $\times$	82.98	2.33 $\times$	69.68	1.92 $\times$	72.27	2.01 $\times$	80.02	2.25 $\times$	82.25	2.31 $\times$
FS( $\mathcal{E}8\text{-}\gamma6$ )	76.44	2.06 $\times$	74.72	2.07 $\times$	84.24	2.37 $\times$	68.99	1.90 $\times$	69.24	1.92 $\times$	77.89	2.19 $\times$	78.23	2.20 $\times$
FS( $\mathcal{E}8\text{-}\gamma12$ )	74.58	2.01 $\times$	72.95	2.02 $\times$	82.71	2.32 $\times$	66.31	1.83 $\times$	66.47	1.85 $\times$	75.68	2.13 $\times$	74.72	2.10 $\times$
DV( $\mathcal{E}7$ )	51.57	1.39 $\times$	51.91	1.44 $\times$	89.65	2.52 $\times$	49.76	1.37 $\times$	72.05	2.00 $\times$	78.31	2.20 $\times$	61.61	1.73 $\times$
DV( $\mathcal{E}8$ )	51.27	1.38 $\times$	54.57	1.51 $\times$	90.78	2.55 $\times$	49.26	1.36 $\times$	76.06	2.11 $\times$	76.91	2.16 $\times$	65.61	1.84 $\times$
DEL	89.43	<b>2.41<math>\times</math></b>	86.86	<b>2.41<math>\times</math></b>	98.10	<b>2.75<math>\times</math></b>	72.31	<b>1.99<math>\times</math></b>	76.31	<b>2.12<math>\times</math></b>	86.77	<b>2.44<math>\times</math></b>	94.42	<b>2.65<math>\times</math></b>
LLaMA-2-13B														
Vanilla	27.69	1.00 $\times$	27.39	1.00 $\times$	25.97	1.00 $\times$	27.96	1.00 $\times$	27.60	1.00 $\times$	27.28	1.00 $\times$	27.84	1.00 $\times$
LS( $\mathcal{E}7\text{-}\gamma3$ )	52.95	1.91 $\times$	51.48	1.88 $\times$	43.81	1.69 $\times$	47.27	1.69 $\times$	50.01	1.81 $\times$	50.20	1.84 $\times$	51.20	1.84 $\times$
LS( $\mathcal{E}7\text{-}\gamma4$ ) <sup>†</sup>	55.19	1.99 $\times$	51.86	1.89 $\times$	44.61	1.72 $\times$	47.57	1.70 $\times$	50.74	1.84 $\times$	51.04	1.87 $\times$	52.19	1.87 $\times$
LS( $\mathcal{E}15\text{-}\gamma4$ ) <sup>†</sup>	43.13	1.56 $\times$	40.74	1.49 $\times$	44.36	1.71 $\times$	41.81	1.50 $\times$	40.91	1.48 $\times$	41.43	1.52 $\times$	43.23	1.55 $\times$
FS( $\mathcal{E}7\text{-}\gamma3$ )	63.54	2.29 $\times$	57.72	2.11 $\times$	45.57	1.75 $\times$	52.18	1.87 $\times$	52.29	1.89 $\times$	52.18	1.91 $\times$	54.94	1.97 $\times$
FS( $\mathcal{E}7\text{-}\gamma4$ )	62.04	2.24 $\times$	56.48	2.06 $\times$	45.94	1.77 $\times$	52.37	1.87 $\times$	51.96	1.88 $\times$	51.51	1.89 $\times$	54.87	1.97 $\times$
FS( $\mathcal{E}15\text{-}\gamma4$ )	48.69	1.76 $\times$	46.28	1.69 $\times$	46.90	1.81 $\times$	44.81	1.60 $\times$	41.70	1.51 $\times$	44.32	1.62 $\times$	45.36	1.63 $\times$
DV( $\mathcal{E}7$ )	41.03	1.48 $\times$	39.77	1.45 $\times$	35.39	1.36 $\times$	38.72	1.38 $\times$	38.99	1.41 $\times$	38.53	1.41 $\times$	40.75	1.46 $\times$
DV( $\mathcal{E}15$ )	40.02	1.45 $\times$	36.33	1.33 $\times$	50.97	1.96 $\times$	37.12	1.33 $\times$	47.62	1.73 $\times$	48.36	1.77 $\times$	46.04	1.65 $\times$
DEL	62.18	<b>2.25<math>\times</math></b>	57.89	<b>2.11<math>\times</math></b>	59.50	<b>2.29<math>\times</math></b>	52.96	<b>1.89<math>\times</math></b>	55.51	<b>2.01<math>\times</math></b>	53.97	<b>1.98<math>\times</math></b>	55.16	<b>1.98<math>\times</math></b>
LLaMA-2-70B														
Vanilla	9.71	1.00 $\times$	10.47	1.00 $\times$	9.03	1.00 $\times$	9.52	1.00 $\times$	10.51	1.00 $\times$	9.78	1.00 $\times$	9.83	1.00 $\times$
LS( $\mathcal{E}8\text{-}\gamma6$ )	24.57	2.53 $\times$	20.21	1.93 $\times$	18.96	2.10 $\times$	17.86	1.88 $\times$	20.50	1.95 $\times$	19.28	1.97 $\times$	19.54	2.00 $\times$
LS( $\mathcal{E}9\text{-}\gamma3$ )	21.55	2.22 $\times$	20.02	1.91 $\times$	17.92	1.99 $\times$	17.69	1.86 $\times$	20.07	1.91 $\times$	18.90	1.93 $\times$	20.55	2.10 $\times$
LS( $\mathcal{E}9\text{-}\gamma6$ )	24.35	2.51 $\times$	20.71	1.98 $\times$	19.04	2.11 $\times$	17.41	1.83 $\times$	20.53	1.95 $\times$	19.49	1.99 $\times$	19.32	1.97 $\times$
FS( $\mathcal{E}8\text{-}\gamma6$ )	26.11	2.69 $\times$	20.88	1.99 $\times$	19.81	2.20 $\times$	17.83	1.87 $\times$	20.24	1.93 $\times$	18.85	1.93 $\times$	19.54	2.00 $\times$
FS( $\mathcal{E}9\text{-}\gamma3$ )	27.00	2.78 $\times$	21.79	2.08 $\times$	19.87	2.20 $\times$	17.88	1.88 $\times$	20.47	1.95 $\times$	19.31	1.97 $\times$	19.98	2.04 $\times$
FS( $\mathcal{E}9\text{-}\gamma6$ )	26.81	2.76 $\times$	21.93	2.09 $\times$	20.05	2.22 $\times$	17.82	1.87 $\times$	20.47	1.95 $\times$	19.52	2.00 $\times$	20.15	2.06 $\times$
DV( $\mathcal{E}8$ )	15.37	1.58 $\times$	15.57	1.49 $\times$	13.66	1.51 $\times$	14.17	1.49 $\times$	15.65	1.49 $\times$	14.66	1.50 $\times$	17.52	1.79 $\times$
DV( $\mathcal{E}9$ )	15.41	1.59 $\times$	15.66	1.50 $\times$	13.65	1.51 $\times$	14.12	1.48 $\times$	15.57	1.48 $\times$	14.73	1.51 $\times$	17.38	1.78 $\times$
DEL	27.55	<b>2.84<math>\times</math></b>	22.48	<b>2.15<math>\times</math></b>	23.20	<b>2.57<math>\times</math></b>	19.05	<b>2.00<math>\times</math></b>	22.90	<b>2.18<math>\times</math></b>	20.73	<b>2.12<math>\times</math></b>	20.69	<b>2.12<math>\times</math></b>

Table 3: Detailed performance comparison of DEL against static (LS) and dynamic (FS, DV) speculative decoding baselines across multiple tasks and model sizes. We report speed (in tokens/sec) and speedup for each method. DEL achieves the highest speedup in most cases, demonstrating consistent gains across different task types and model scales. The best results are bolded, and the second-best results are underlined.

### C.1 Extended Baseline Comparisons

We compare DEL with SWIFT (Xia et al., 2025), a recent self-speculative decoding method, on the HumanEval (code generation) and CNN/DailyMail (language modeling) datasets using LLaMA-2-7B and 13B models. Like DEL, SWIFT is a plug-and-play method that requires no model retraining or architectural modifications. However, unlike DEL, it operates without early-exit capability. To ensure a fair comparison, we evaluate DEL on top of LayerSkip and compare its speedup gain over LayerSkip to the gain SWIFT achieves over vanilla decoding. As shown in Table 4, DEL consistently delivers larger relative improvements across all settings. These results demonstrate that DEL offers stronger acceleration benefits when built atop early-exit self-speculation, highlighting its complementarity with LayerSkip.

Models	LLaMA-2-7B				Methods	LLaMA-2-13B			
	HumanEval		CNN/DM			HumanEval		CNN/DM	
	Speedup	Gain	Speedup	Gain		Speedup	Gain	Speedup	Gain
Vanilla	1.00×	—	1.00×	—	Vanilla	1.00×	—	1.00×	—
SWIFT	1.01×	<b>0.01</b> ×	1.06×	<b>0.06</b> ×	SWIFT	1.13×	<b>0.13</b> ×	1.26×	<b>0.26</b> ×
LS( $\mathcal{E}7\text{-}\gamma 6$ )	1.92×	—	1.88×	—	LS( $\mathcal{E}9\text{-}\gamma 4$ )	1.68×	—	1.67×	—
DEL	2.07×	<b>0.15</b> ×	2.30×	<b>0.48</b> ×	DEL	1.85×	<b>0.17</b> ×	2.28×	<b>0.61</b> ×

Table 4: Speedup comparison of DEL and SWIFT on HumanEval and CNN/DM using LLaMA-2-7B and 13B. “Speedup” is measured relative to vanilla decoding. “Gain” denotes improvement over LayerSkip for DEL, and over Vanilla for SWIFT. DEL consistently achieves larger relative gains than SWIFT across tasks and model sizes.

## D Further Analysis

### D.1 Memory Breakdown

Table 5 shows the maximum allocated GPU memory during the inference of a randomly selected prompt from each dataset. The experiments were conducted on LLaMA-2-7B using three decoding methods: Vanilla auto-regressive decoding, LayerSkip with exit layer 7 and speculation length 6 (LS( $\mathcal{E}7\text{-}\gamma 6$ )), and DEL. For each task, the maximum GPU memory used during the entire inference process is reported, along with the memory overhead compared to Vanilla decoding. The results demonstrate that both LS and DEL introduce minimal memory overhead. LS( $\mathcal{E}7\text{-}\gamma 6$ ) incurs an overhead of 0.50%~1.42%, depending on the task, with an average overhead of 0.91% across all tasks. DEL introduces a slightly higher overhead, ranging from 0.52% to 2.26%, with an overall average of 1.54%. Overall, DEL maintains a minimal memory overhead relative to Vanilla decoding, ensuring scalability and making it easy to integrate into various scenarios without significant resource constraints.

Methods		HumanEval (Coding)	CNN/DM (Lang. Mod.)	CNN/DM (Abs. Sum.)	XSUM (Abs. Sum.)	GSM8K (Reasoning)	AQuA-RAT (Reasoning)	Overall
Vanilla	Memory Overhead	13.184	13.200	14.348	13.288	14.252	14.015	13.715
		—	—	—	—	—	—	—
LS( $\mathcal{E}7\text{-}\gamma 6$ )	Memory Overhead	13.250	13.270	14.551	13.363	14.435	14.173	13.840
		0.50%	0.53%	1.42%	0.56%	1.28%	1.13%	0.91%
DEL	Memory Overhead	13.253	13.393	14.672	13.487	14.473	14.275	13.926
		0.52%	1.46%	2.26%	1.50%	1.55%	1.86%	1.54%

Table 5: Maximum allocated GPU memory (in GB) during inference of a randomly selected prompt from each dataset using LLaMA-2-7B. Memory overhead relative to Vanilla auto-regressive decoding is shown as a percentage. DEL maintains a minimal memory overhead, ranging from 0.52% to 2.26%, with an overall average of 1.54%, ensuring scalability and ease of integration across different scenarios.

## D.2 Decay Control Factor $\omega$ Sensitivity Analysis

We analyze the impact of different values of  $\omega$  on the achieved speedup by running LLaMA-2-7B on the CNN/DM Summarization task.  $\omega$  is the decay factor used in the weighted sum function, defined in Section 4.2. This function gradually forgets information from previous SD rounds, and  $\omega$  controls the rate of decay. Table 6 presents the results of varying  $\omega \in \{0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 1.0\}$ . For reference, the first column (with  $\omega = -$ ) shows the baseline results for Vanilla decoding with no speculative acceleration. The table reports eTPL, Speed (tokens/sec), and Speedup for each  $\omega$  value. The results indicate that DEL is largely insensitive to  $\omega$ , with speedup improving from 2.54 $\times$  at  $\omega = 0.5$  to a peak of 2.60 $\times$  at  $\omega = 0.9$  and 1.0, and remaining stable (2.57 $\times$ ) even at  $\omega = 0.95$ .

Metrics	$\omega = -$	$\omega = 0.5$	$\omega = 0.6$	$\omega = 0.7$	$\omega = 0.8$	$\omega = 0.9$	$\omega = 0.95$	$\omega = 1$
eTPL	0.031	0.091	0.092	0.092	0.093	0.094	0.094	0.093
Speed	35.61	90.61	91.41	91.04	92.53	92.75	91.55	92.66
Speedup	1.00 $\times$	2.54 $\times$	2.57 $\times$	2.56 $\times$	2.60 $\times$	2.60 $\times$	2.57 $\times$	2.60 $\times$

Table 6: Effect of varying  $\omega$  on eTPL, speed (tokens/sec), and speedup during inference on the CNN/DM Summarization task using LLaMA-2-7B. DEL remains largely insensitive to  $\omega$ , achieving near-optimal speedups (up to 2.60 $\times$ ) across a wide range of values.