Characterizing self-speculative decoding approaches for accelerating LLMs

Jungmin Ha [1] **Karthik Ganesan** [1] **Anh Nguyen** [2] **Tanvir Ahmed** [2] **Andreas Moshovos** [1]

Abstract

Large language model (LLM) inference is bottlenecked by autoregressive token generation, leading to poor GPU utilization and high LLM inference latency. Speculative decoding is a promising technique to speed up inference where 'speculative' tokens are generated with a cheaper model, and then validated *en masse* by the full model. While effective, speculative decoding requires an additional "draft" model. Self-speculative decoding obviates this need by reusing parts of the full model itself to generate candidate tokens. Existing methods implement this approach in different ways: some exit at an early decoder block, while others selectively skip layers within each block. Such a complex design space entails different tradeoffs in draft quality, draft & verification cost, and overall speedup. To better understand these tradeoffs, we characterize five self-speculative decoding techniques across three model sizes and three datasets and provide recommendations for future research in this area.

1. Introduction

Large language models (LLMs) have grown increasingly popular in recent years, with LLMs such as OpenAI's GPT (Brown et al., 2020), Google's Gemini (Team et al., 2023), and Anthropic's Claude (Anthropic, 2024), being widely used in everyday tasks. However, this growth in LLMs has severely taxed the compute capabilities of even high-end graphics processing units (GPUs). A major bottleneck for LLM inference is that the decode phase, which dominates inference time, uses matrix-vector operations. This results in poor utilization of GPUs, which are optimized for matrix-matrix operations. One promising approach to speed-up LLMs is *speculative decoding*, which uses a second, smaller model to generate several 'candidate' tokens. These tokens are then verified in parallel by the full model, using more GPU-friendly matrix-matrix operations.

This work studies *self-speculative decoding*, which obviates the need for a separate model and instead uses a smaller portion of the full model to generate candidate tokens. Self-speculative decoding techniques broadly fall into two categories: 1) *layer skipping*, and 2) *early exit*. While these techniques have much in common, they are not identical and allow for different trade-offs between latency, model cost and output quality. However, no existing benchmark isolates self-speculative methods or decomposes the throughput factors that separate layer skipping from early exit (Xia et al., 2024; Sun et al., 2025). By characterizing prior works that implement these techniques, this work aims to make it easier for others to understand the underlying trade-offs.

We characterize five methods: 1) SWIFT (Xia et al., 2025), 2) Draft & Verify (Zhang et al., 2024), 3) LayerSkip (Elhoushi et al., 2024), 4) DEL (Zarch et al., 2025) and 5) Kangaroo (Liu et al., 2024a). We primarily evaluate using three *LayerSkip-Llama-2* models, namely 7B, 13B and 70B, using three datasets (i.e., CNN/DailyMail, GSM8K and HumanEval). Results for the standard *Llama-2*, *Vicuna-v1.3* and the more recent *LayerSkip-Llama-3* 8B models are included in Appendix A. Based on the analysis results, we conclude with takeaways for developing better self-speculative decoding techniques in the future.

2. Background

Modern LLMs are typically based on the decoder-only Transformer architecture (Vaswani et al., 2017). Without loss of generality, we can assume that LLMs consist of L identical decoder blocks where each block contains masked multi-head self-attention and a feedforward network (FFN), connected via residual connections and normalization layers, as shown in Figure 2 (left). During inference, the model processes a sequence of input tokens by propagating the token embedding through all the decoder blocks. The final hidden state is then projected by a language modeling head (LM head) to a logit over the vocabulary.

LLM inference occurs in two phases: *pre-fill* and *decode* (Patel et al., 2024), shown in Figure 1. During pre-fill, all tokens

[1]The Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada [2]Fujitsu, Toronto, Canada. Correspondence to: Jungmin Ha <jay.ha@mail.utoronto.ca>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

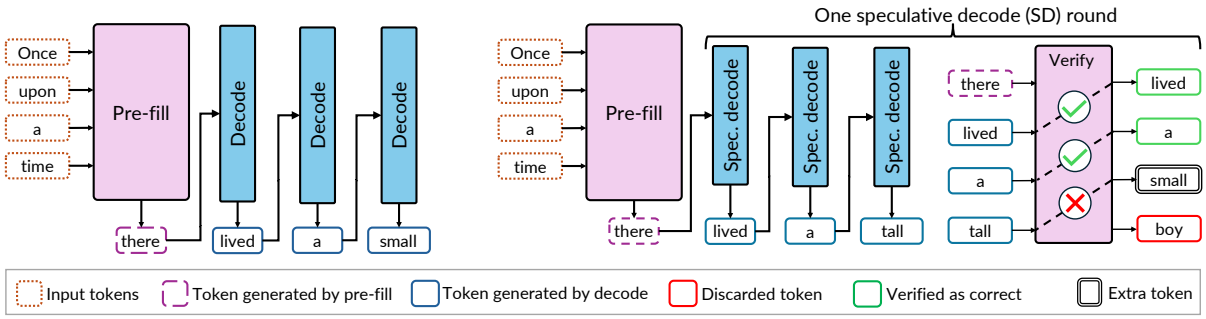


Figure 1. Comparison of the phases for regular LLM inference (left) and inference using speculative decode (right).

in the prompt are input in parallel to the model. At the end of this phase, a single new token is produced. This phase uses **matrix-matrix** multiplications (GEMM), which make the best use of GPUs. This is because GEMM operations exhibit good data reuse and modern GPUs often implement specialized processing units (e.g., tensor cores) specifically targeting GEMM operations. In contrast, the decode phase generates tokens auto-regressively, one token at a time, using cached key–value (KV) pairs from prior steps. This phase is inherently sequential and dominated by **matrix–vector** multiplications (GEMV). As a result, decode latency is the primary bottleneck in LLM inference. A number of approaches have been proposed to improve LLM inference efficiency. One family of such techniques is *speculative decoding*, which we describe next.

2.1. Speculative decoding

Speculative decoding involves generating d candidate tokens using a small, fast *draft model* and then verifying all candidate tokens in parallel with the *full model*,[1] as shown in Figure 1 (right) (Leviathan et al., 2023; Chen et al., 2023). We refer to one such draft-verify cycle as a speculative decode (SD) *round*, and to d as the *draft length*. During verification, the full model processes $d+1$ tokens (the token produced from pre-fill or the previous verification and d draft tokens) in parallel, predicting the next token at each position. The longest prefix of draft tokens that match the predictions of the full model are taken as correct, while all draft tokens starting from the earliest 'incorrect' token are rejected by the full model. In addition, the full model produces an *extra token* at the last accepted draft position.

In Figure 1 (right), the first 2 out of the 3 draft tokens match the full model's predictions and are accepted. The third ('tall') is rejected and replaced by the extra token ('small'). The full model's prediction at the rejected draft position ('boy') is discarded because it was conditioned on the rejected draft token 'tall'. One SD round therefore advances decoding by between 1 token (no draft tokens accepted, leaving only the extra token) and $d+1$ tokens (all draft tokens are accepted plus the extra token). We define the *acceptance rate* as the fraction of the d draft tokens accepted per round.

The benefit of speculative decode depends on two factors: 1) the cost of producing the draft tokens, and 2) the quality of the draft tokens. In the best case, the draft model is always correct and every full-model GEMV is replaced by a cheaper draft-model GEMV, and the only full-model work per round is a single GEMM over all d draft tokens. In the worst case, a draft model whose tokens are consistently rejected adds drafting overhead without progress and slows decoding overall.

Despite its potential effectiveness, speculative decoding suffers from two drawbacks: 1) The draft model must be trained to closely match the full model's output distribution. This requires either training the small model from scratch or distilling the full model - both of which are expensive and must be repeated for every new full model or when the full model is fine-tuned for a different task. 2) The draft model also occupies additional GPU memory at deployment alongside the full model. One approach that overcomes these drawbacks is *self-speculative decoding*, which we talk about next.

2.2. Self-Speculative Decoding

Self-speculative decoding creates the draft model using a subset of the full model's own layers, obviating the need for an entirely separate model. We explore two variants that have been explored in the literature: *early exit* and *layer skipping*. *Parallel decoding* is another approach that generates tokens *in parallel* using auxiliary decoding heads (Cai et al., 2024) or look-ahead embeddings (Monea et al., 2023; Lin et al., 2025; Chen et al., 2025a). As parallel decoding does not use the 'draft-then-verify' approach of early exit and layer skipping, we leave the analysis of parallel decoding to future work.

2.2.1. Early Exit

Early-exit methods generate draft tokens by terminating the full model's forward pass at an intermediate decoder block

[1]The full model is sometimes called the target model in prior work; we use "full" throughout this work to contrast with the partial-model used by self-speculative methods in Section 2.2.

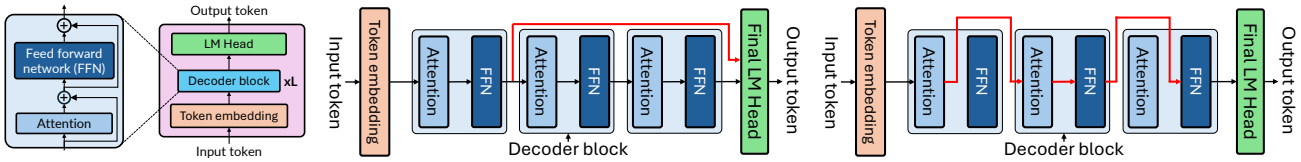


Figure 2. Architecture of a typical LLM (left), early exit model showing decoder blocks being skipped (middle), and layer skipping model, showing internal layers being skipped (right).

$E < L$, where L is the total number of decoder blocks, and sending the output of block E directly to the LM head. For example, $E = 1$ and $L = 3$ in Figure 2 (middle). The blocks above E run only during verification.

We now list some early-exit approaches proposed in prior work. LayerSkip augments training with layer-dropout and early-exit losses (Elhoushi et al., 2024). In this way, intermediate decoder blocks produce representations that the LM head can decode directly, enabling early exit at a fixed block during inference. DEL builds on LayerSkip but selects the exit block dynamically based on hidden states across blocks (Zarch et al., 2025). Kangaroo trains a small 'adapter network' to bridge the output of an early decoder block directly to the LM head of the full model (Liu et al., 2024a). EESD adds a learnable transformer layer above the first E decoder blocks and trains it via self-distillation (Liu et al., 2024b). SpecEE reduces each block's exit-decision search space by leveraging tokens from a draft model and trains lightweight per-block MLP predictors to decide when to exit (Xu et al., 2025).

Because early layers were not designed to function as standalone models, all early-exit methods require either retraining the full model (e.g., LayerSkip, DEL) or training an auxiliary adapter that bridges the exit block to the LM head (e.g., Kangaroo). In contrast, another approach is layer skipping which avoids this training requirement.

2.2.2. Layer Skipping

Layer-skipping methods generate draft tokens by executing a *non-contiguous* subset of the full model's sub-block components (e.g., attention and FFN layers) within each decoder block, as illustrated in Figure 2 (right). Layer skipping is supported by work that shows that only a small input-dependent subset of attention heads and FFN neurons is needed to recover the full model's output (Liu et al., 2023). Since residual connections preserve input/output dimensions throughout the model, bypassing layers requires no architectural changes. The 'skip pattern' of layers must be aggressive enough to reduce drafting cost, yet selective enough to retain draft quality.

Draft & Verify uses a one-time offline Bayesian optimization on task-specific samples to select a fixed skip pattern that is reused at inference (Zhang et al., 2024). SWIFT instead discovers its pattern on-the-fly via random search and Bayesian optimization during a brief warm-up, after which the pattern is frozen for the remainder of generation (Xia et al., 2025). Other layer-skipping methods—CLaSp (Chen et al., 2025b), KNN-SSD (Song et al., 2026), and KnapSpec (Cha et al., 2026)—re-select skip patterns per round via k-nearest-neighbor retrieval or knapsack-style optimization

As Section 4 will show, layer skipping sees high acceptance rates. However, this irregular pattern of skipping layers comes at a cost. To preserve draft quality, the draft model uses more of the full model's layers compared to early-exit methods, increasing drafting latency. Another disadvantage is that by bypassing some layers, the activations and KV cache created during drafting cannot be used during verification. The full model must be run from scratch over all candidate tokens for verification.

In contrast, early-exit methods can reuse the draft models' computation directly, since the subset of decoder blocks run during drafting are the same as the full models. Therefore, these two approaches offer complementary trade-offs. Layer skipping opts for a flexible, training-free draft model, but at the cost of a higher-latency draft model and no reuse during verification. On the other hand, early exit incurs the burden of training and a more constrained draft in exchange for a cheaper draft and cheaper verification. We explore these trade-offs in Section 4.

3. Methodology

All experiments are run on NVIDIA H100 SXM GPUs with 80GB of HBM and using PyTorch 2.11.0. The GPU wall-clock time was measured via CUDA events. Appendix B provides the full testbed configuration.

Techniques: Table 1 summarizes the evaluated techniques. From the list of all prior self-speculative decoding techniques, we choose ones with publicly available code repositories. We use author-provided code and the best recommended hyper-parameters for all techniques. We compare every technique against the model running regular inference (as shown in Figure 1 (left)). We refer to this model throughout as the **baseline**.

Models: We primarily use *Llama-2* (Touvron et al., 2023) 7B, 13B & 70B models for our evaluation, as these models

Table 1. List of self-speculative decoding techniques evaluated.

PAPER	TRAINING REQUIRED?	DRAFT LENGTH	DRAFT SIZE
LAYER SKIPPING			
SWIFT	NO	DYNAMIC	FIXED
DRAFT & VERIFY	NO	DYNAMIC	FIXED
EARLY EXIT			
LAYERSKIP	YES	STATIC	FIXED
DEL	YES	DYNAMIC	VARIABLE
KANGAROO	YES	DYNAMIC	FIXED

are supported by all the techniques we study. As *LayerSkip* requires retraining, we use the models they provide for our experiments, and refer to these as *LayerSkip-Llama-2* models. *LayerSkip-Llama-2* models closely match the output quality of the standard *Llama-2* models (Elhoushi et al., 2024). Appendix A also provides results for the standard *Llama-2* models, *Vicuna-v1.3* (Chiang et al., 2023) 7B & 13B models and the newer *LayerSkip-Llama-3* 8B model.

As *LayerSkip-Llama-2* models have a maximum context length of 2048 tokens, we limit the maximum generated tokens to 512 and truncate prompts to 1504 tokens. *SWIFT* is given an extra 32 context tokens, since, as discussed in Section 4.3, it uses a tree-based approach during drafting. We decode greedily (`temperature = 0`) with a fixed random seed.

Datasets: We use datasets commonly used in prior work on self-speculative decoding, such as: *HumanEval* (Chen et al., 2021), *CNN/DailyMail* (Hermann et al., 2015; See et al., 2017) and *GSM8K* (Cobbe et al., 2021). Due to space limitations, results are presented for *HumanEval* in Section 4, while results for the other two datasets are in Appendix A. *HumanEval* is a benchmark of hand-written programming problems with unit tests designed to assess the functional correctness of code generated by LLMs.

Metrics: The primary metric we evaluate is throughput, measured in tokens per second (tps). Throughput is calculated as the number of generated tokens, divided by the time to generate all tokens per sample, normalized to the baseline. As self-speculative decoding targets the decode phase, we do not include the time for pre-fill in our measurements. In our experiments, pre-fill accounts for $< 1\%$ of the total latency and is also fixed for all techniques.

To better understand where time is spent during self-speculative decode, we collect additional metrics, such as draft length, draft model cost and acceptance rate, to measure the quality and overheads of the underlying speculation. We define these metrics in Section 4 as we present our measurements and explain why each one is useful in understanding the behaviour of different techniques.

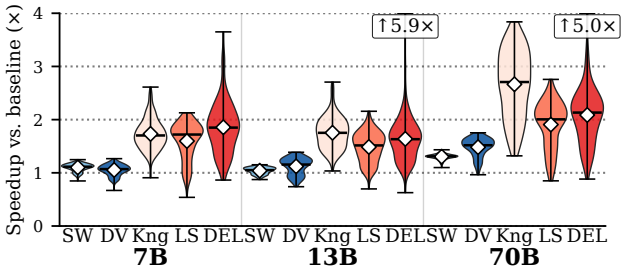


Figure 3. Throughput, normalized to the baseline for *LayerSkip-llama-2* models on *HumanEval*.

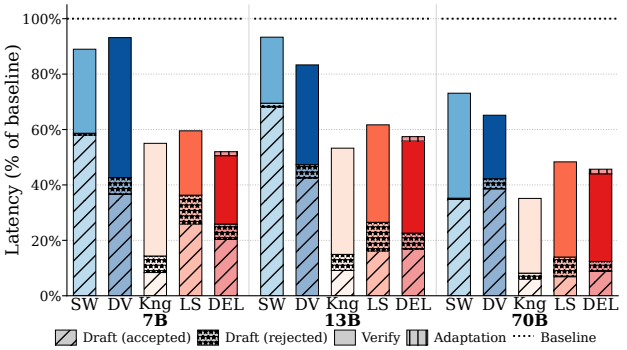


Figure 4. Latency breakdown, normalized to the baseline for *LayerSkip-llama-2* models on *HumanEval*. 'Adaptation' segment represents *DEL*'s per-round reconfiguration overhead.

4. Characterization

4.1. Throughput

We begin with throughput as it is the key motivation for self-speculative decoding techniques. Figure 3 shows the throughput for the 3 *LayerSkip-llama-2* models, for the *HumanEval* dataset. For each technique, we show throughput normalized to the baseline (i.e., the model with no technique applied). The two layer skipping approaches only see modest speed-ups of $1.04\times$ –$1.48\times$ over the baseline, while the early exit approaches achieve significantly higher speed-ups. Across early-exit methods, mean speed-ups range from $1.49\times$ –$2.67\times$, with a best case of $\sim 5.9\times$ for DEL. DEL's $\sim 5.9\times$ outliers stem from cases where LayerSkip-Llama-2 loops on a repeating token without emitting EOS; DEL exploits this by dynamically setting $E = 1$ and $d = d_{max}$.

Figure 4 aids in understanding these speedups by reporting the latency breakdown per technique, split into draft and verify stages. To preserve the relative speed-up of each technique, we normalize to the runtime of the baseline. One reason why layer skipping is slower is because these techniques spend more of their runtime drafting. To understand why, we start by analyzing *draft length*.

4.2. Draft Length

Draft length is the number of tokens generated by the draft model per round, before the full model is used to verify all

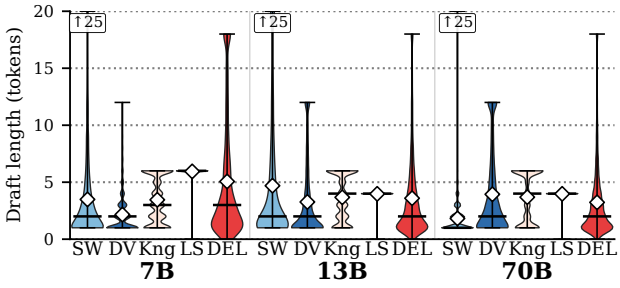


Figure 5. Number of draft tokens per SD round on HumanEval.

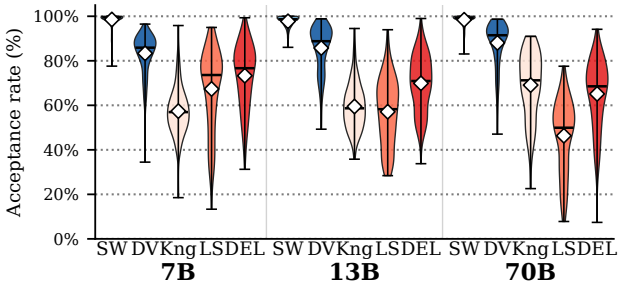


Figure 6. Acceptance rate of candidate tokens on HumanEval.

candidate tokens. The longer the draft length, the greater the potential speed-up using self-speculative decoding. Thus, all techniques aim to have the longest draft length possible, up to a maximum draft length ($d_{\max}$).

Figure 5 shows the draft length per speculative decode round. All techniques except *LayerSkip* achieve a median draft length of $\sim 2-3$ tokens, with a long upper tail that stops at $d_{\max}$. In contrast, *LayerSkip* sees a slightly higher median draft length of $\sim 4-6$ tokens. This is because it is the only method that uses a fixed draft length. However, as Section 4.3 will show, using a fixed draft length negatively impacts the acceptance rate for *LayerSkip*.

The other four methods continue drafting as long as the draft model's *confidence* (measured as top–1 probability) stays above a *draft-stop threshold* (ϵ) or until the maximum draft length ($d_{\max}$) is reached for this round. This dynamic stopping rule brings the median draft length down to $\sim 2-3$ tokens. If the draft model's confidence drops after a few tokens, the round ends well before $d_{\max}$. On the other hand, when the confidence holds for several tokens and the draft-stop condition is never triggered, drafting runs all the way to $d_{\max}$. Every technique reaches $d_{\max}$, as this corresponds to the longest draft length shown in Figure 5. This bimodal behaviour — short drafts on most rounds, occasional long drafts on easy token sequences — is the source of the long upper tail visible in Figure 5. This is most pronounced for *SWIFT*, where draft length reaches 25.

Among the four dynamic-length methods, *DEL* shows the highest median draft length. In *SWIFT* and *Kangaroo*, both $d_{\max}$ and ϵ are fixed. However, in *Draft & Verify*, $d_{\max}$ is fixed, while ϵ varies at runtime. *DEL* is the only method that adapts both; it adjusts ϵ and re-selects $d_{\max}$ (up to 18) every round to maximize the throughput. Thus, draft length varies considerably across both early exit and layer skipping techniques and is not solely indicative of their final performance.

4.3. Acceptance Rate

While a longer draft length can be beneficial, it must be coupled with a high acceptance rate to yield speed-ups. Figure 4 shows that verification time varies widely between techniques. One reason for this is low quality draft tokens are rejected by the full model. This causes additional SD rounds, which increases both draft and verification latency. Thus, acceptance rate is a key indicator of throughput.

As Figure 6 shows, the two layer skipping techniques achieve high acceptance rates. This is corroborated in Figure 4 as these techniques suffer very little overhead due to rejected draft tokens. *Draft & Verify* has a mean acceptance rate of $\sim 86\%$, but can be as low as $\sim 40\%$. *SWIFT* achieves a mean acceptance rate of $\sim 99\%$, dipping occasionally to $\sim 80\%$. This is because *SWIFT* maintains multiple high-probability tokens (top-k) during each step of drafting. Thus, in a single round, *SWIFT* generates a 'tree' of candidate tokens rather than a single list like the other techniques. At the end of the round, all of these tokens are verified, increasing the chances of some tokens matching the full model output. This is the source of *SWIFT*'s $\sim 99\%$ acceptance rate.

In contrast, the early–exit techniques achieve a far lower mean and show greater variability in acceptance rate, driven by two factors. First, early–exit methods draft from a much smaller sub–network — only 7%–25% of the full model's layers, compared to 47%–61% for the layer–skipping methods. With far less compute, the draft predictions are weaker on average, lowering the mean acceptance rate.

Second, the three early–exit methods differ in when they stop drafting. *LayerSkip* runs a fixed number of draft steps d without checking the draft model's confidence in between. When the next token is easy to predict, almost every drafted token is accepted; when it is hard, the draft model keeps proposing tokens that the full model rejects. This leads to the acceptance rate drop we see in Figure 6. *Kangaroo* and *DEL* avoid this by stopping as soon as a draft confidence falls below a draft–stop threshold. As a result, both have higher mean acceptance rates with less variability than *LayerSkip*.

Despite their higher acceptance rates, layer skipping methods perform worse than early exit methods. Draft model cost is the final factor affecting performance, which helps us understand why.

4.4. Draft Model cost

To quantify the cost of running the draft model, we measure the per-step decode latencies of both the draft model

Table 2. Per–step decode latency of draft model, relative to the per–step decode latency of the baseline *LayerSkip–llama–2* model.

TECHNIQUE \ MODEL	7B	13B	70B
SWIFT	74%	82%	53%
DV	56%	57%	49%
KANGAROO	12%	13%	8%
LAYERSKIP	32%	23%	10%

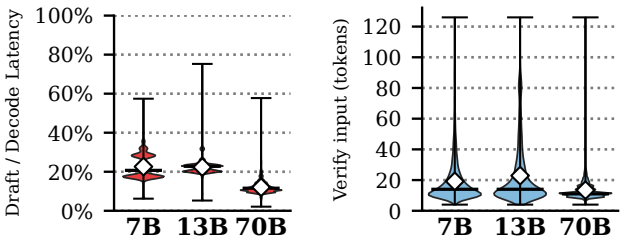


Figure 7. Draft model cost for *DEL* (left) and number of tokens during verification for *SWIFT* (right) on HumanEval.

and the full model. For the relatively short context lengths studied, there is little variance in the decode latencies based on context length. Therefore, draft model cost is calculated as the ratio of the average per-step decode latencies for the draft and the full model. Table 2 lists the costs for all techniques apart from *DEL*. *DEL* dynamically adjusts its exit point based on the current context. Figure 7 (left) shows the distribution of draft model costs for *DEL*.

Despite smaller draft lengths and lower acceptance rates, early-exit techniques achieve higher average speed-ups. *LayerSkip*'s static exit points are relatively deep ($E = 7$ or 8), giving a higher draft cost of 10%–32%. *Kangaroo* achieves a lower draft cost of 8%–13% by exiting much earlier ($E = 2$ for 7B, $E = 3$ for 13B, and $E = 6$ for 70B), using a small trained adapter to bridge the premature hidden state to the LM head. The adapter adds some overhead, but the savings from running only the first few decoder blocks dominate. While the draft model cost varies significantly for *DEL*, the average is in line with the other two early exit techniques. In addition, early exit techniques re-use computations from drafting to speed-up verification. This results in the shorter proportion of time they spend verifying (Figure 4).

In contrast, *SWIFT* and *Draft & Verify* have much higher draft model costs. Both techniques use a larger portion of the full model while drafting, as seen with the larger bars in Figure 4. Furthermore, as the draft models for these techniques use non-contiguous layers, verification cannot reuse any computations from drafting. This results in them taking longer for verification (Figure 4).

While the 'tree-like' drafting used by *SWIFT* leads to high acceptance rates, this greatly increases the number of tokens that the full model must verify each round, as shown in Figure 7 (right). In some cases, the full model in *SWIFT* verifies up to ~ 120 tokens per round. This extra verification work, on top of the already large layer-skipping draft, explains why *SWIFT*'s high acceptance rate and long draft length do not translate into a high speed-up.

5. Takeaways

The following are the key takeaways from our analysis:

Draft cost, not acceptance rate, dominates throughput: Throughput depends on the overhead to produce and validate draft tokens as well as the accuracy of those tokens. While acceptance rate is the metric most often reported by self-speculative decoding studies, it proves to be a poor standalone predictor of throughput. Instead, it is the draft model cost that impacts throughput the most. Specifically, the throughput of methods using the smallest draft models (i.e., *Kangaroo* and *DEL*) consistently outperforms that of methods with large draft models (i.e., *SWIFT* and *Draft & Verify*), even though the latter produce higher-quality drafts. We recommend future work should focus on and report both draft and verification costs, alongside acceptance rates.

Verification costs are very different across methods: Early-exit methods reuse the draft layers' computation during verification. Layer-skipping methods bypass *non-contiguous* layers so cannot reuse anything and consequently the full model must be run from scratch for verification. This asymmetry, not draft size alone, is what allows early-exit methods to win despite their weaker drafts.

Early exit requires re-training: Early exit methods require retraining the full model (e.g., LayerSkip) or an auxiliary component (e.g., Kangaroo). Where such retraining is infeasible, layer skipping techniques are still a viable option for self-speculative decoding.

Context-specific adaptation outperforms static configurations: *DEL* adapts its exit block, draft length, and draft-stop threshold each round, whereas *LayerSkip* fixes them. *LayerSkip*'s static approach has the widest spread of acceptance rates, since a fixed draft length cannot adapt to hard tokens. As a result, *DEL* outperforms *LayerSkip* in every case.

6. Conclusion

Self-speculative decoding is a promising technique for accelerating LLM inference. Differing approaches, such as layer skipping and early exit, make it difficult to reason about the relative pros and cons of existing techniques. To aid further research, we characterized five self-speculative decoding techniques across three *LayerSkip-Llama-2* sizes and three datasets, decomposing throughput into draft length, acceptance rate, and draft model cost. We hope our work will help researchers reason about the trade-offs each approach imposes and identify underexplored areas of the design space.

References

Anthropic. The Claude 3 model family: Opus, Sonnet, Haiku. Technical report, Anthropic, 2024. URL https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235. PMLR, 2024.

Cha, S., Kim, G., Han, D., Yang, T., and Han, I. Knapspec: Self-speculative decoding via adaptive layer selection as a knapsack problem. *arXiv preprint arXiv:2602.20217*, 2026.

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

Chen, H. M., Luk, W., Yiu, K. F. C., Li, R., Mishchenko, K., Venieris, S., and Fan, H. Hardware-aware parallel prompt decoding for memory-efficient acceleration of LLM inference. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 2221–2238, Suzhou, China, November 2025a. Association for Computational Linguistics.

Chen, L., Shan, R., Wang, H., Wang, L., Liu, Z., Luo, R., Wang, J., Alinejad-Rokny, H., and Yang, M. Clasp: In-context layer skip for self-speculative decoding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 31608–31618, 2025b.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6, 2023.

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

Elhoushi, M., Shrivastava, A., Liskovich, D., Hosmer, B., Wasti, B., Lai, L., Mahmoud, A., Acun, B., Agarwal, S., Roman, A., et al. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, 2024. doi: 10.18653/v1/2024.acl-long.681. URL https://aclanthology.org/2024.acl-long.681.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28, 2015.

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

Lin, F., Yi, H., Yang, Y., Li, H., Yu, X., Lu, G., and Xiao, R. BiTA: Bi-directional tuning for lossless acceleration in large language models. *Expert Systems with Applications*, 279:127305, 2025.

Liu, F., Tang, Y., Liu, Z., Ni, Y., Han, K., and Wang, Y. Kangaroo: Lossless self-speculative decoding via double early exiting. *arXiv preprint arXiv:2404.18911*, 2024a.

Liu, J., Wang, Q., Wang, J., and Cai, X. Speculative decoding via early-exiting for faster llm inference with thompson sampling control mechanism. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3027–3043, 2024b.

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

Monea, G., Joulin, A., and Grave, E. Pass: Parallel speculative sampling. *arXiv preprint arXiv:2311.13581*, 2023.

Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, Í., Maleki, S., and Bianchini, R. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132. IEEE, 2024.

See, A., Liu, P. J., and Manning, C. D. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association*

for Computational Linguistics (Volume 1: Long Papers), pp. 1073–1083, 2017.

Song, M., Xia, H., Zhang, J., Leong, C. T., Xu, Q., Li, W., and Li, S. Knn-ssd: Enabling dynamic self-speculative decoding via nearest neighbor layer set optimization. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 641–655, 2026.

Sun, S., Li, Y., Li, X., Lian, Y., Lin, W., Zhen, H.-L., Yang, Z., Chen, C., Yu, X., Yuan, M., et al. Scaling up, speeding up: A benchmark of speculative decoding for efficient llm test-time scaling. *arXiv preprint arXiv:2509.04474*, 2025.

Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

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

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

Xia, H., Yang, Z., Dong, Q., Wang, P., Li, Y., Ge, T., Liu, T., Li, W., and Sui, Z. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7655–7671, 2024.

Xia, H., Li, Y., Zhang, J., Du, C., and Li, W. 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.

Xu, J., Pan, J., Zhou, Y., Chen, S., Li, J., Lian, Y., Wu, J., and Dai, G. Specee: Accelerating large language model inference with speculative early exiting. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 467–481, 2025.

Zarch, H. E., Gao, L., Jiang, C., and Annavaram, M. Del: Context-aware dynamic exit layer for efficient self-speculative decoding. *arXiv preprint arXiv:2504.05598*, 2025.

Zhang, J., Wang, J., Li, H., Shou, L., Chen, K., Chen, G., and Mehrotra, S. Draft& verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, 2024.

A. Additional results

This section presents additional results that we did not have space for in the main paper body.

1. CNN/DailyMail results for LayerSkip-Llama-2 models (Section A.2).
2. GSM8k results for LayerSkip-Llama-2 models (Section A.3).
3. Standard Llama-2 family (Section A.4).
4. LayerSkip-Llama-3 8B model (Section A.5).
5. Vicuna-v1.3 family (Section A.6).

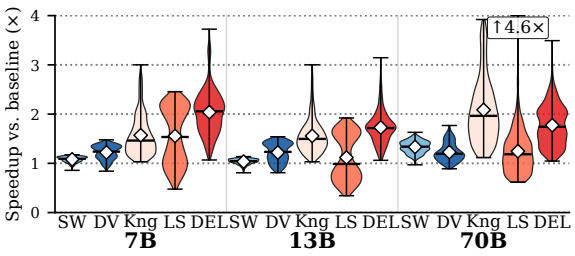
Draft model size (DRAFT SIZE). Throughout this appendix we report draft model size as the fraction of the full model executed during drafting, computed as the number of decoder blocks used during drafting divided by L, where L is the total number of decoder blocks. For layer-skipping methods (*SWIFT*, *Draft & Verify*), which operate at sub-block granularity by selectively skipping individual attention and FFN layers, a partially-executed decoder block contributes a fractional value: attention and FFN layers are weighted by their parameter counts so that a full decoder block sums to 1. For *Kangaroo*, the trained single-layer attention adapter is included in the draft size using the same parameter-weighted convention as the layer-skipping methods. For DEL, which adapts its exit layer per round, draft size is reported as mean ± std across rounds.

A.1. LayerSkip-Llama-2: HumanEval results

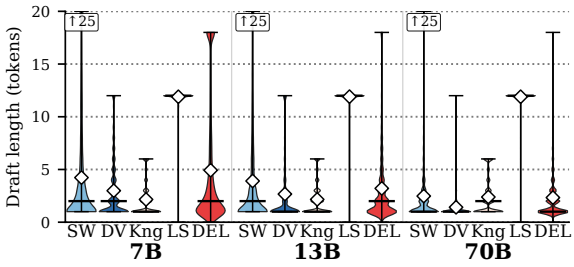
Table 3. Results for *layerskip-llama-2* 7B, 13B & 70B models for the HumanEval dataset.

METHOD	SIZE	THROUGHPUT (TOK/S)	DRAFT SIZE (FRAC)	DRAFT LENGTH (TOK/ROUND)	ACCEPTANCE
BASELINE	7B	63.0 ± 0.3	–	–	–
	13B	52.8 ± 0.4	–	–	–
	70B	15.9 ± 0.2	–	–	–
SWIFT	7B	69.3 ± 4.6	0.61	3.53 ± 3.95	0.99 ± 0.03
	13B	54.9 ± 3.3	0.57	4.90 ± 5.95	0.98 ± 0.03
	70B	20.7 ± 0.8	0.53	1.82 ± 1.26	0.98 ± 0.03
D&V	7B	66.3 ± 6.6	0.47	2.16 ± 1.73	0.83 ± 0.09
	13B	59.2 ± 8.4	0.50	3.27 ± 3.26	0.86 ± 0.10
	70B	23.6 ± 2.8	0.49	3.94 ± 3.55	0.88 ± 0.10
KANGAROO	7B	109.1 ± 17.7	0.07	3.46 ± 1.99	0.57 ± 0.10
	13B	92.5 ± 15.7	0.08	3.67 ± 2.03	0.59 ± 0.10
	70B	42.4 ± 10.0	0.08	3.69 ± 2.22	0.69 ± 0.15
LAYERSKIP	7B	100.6 ± 21.9	0.25	5.94 ± 0.45	0.67 ± 0.18
	13B	78.5 ± 14.7	0.17	3.97 ± 0.24	0.57 ± 0.15
	70B	30.4 ± 6.9	0.10	3.98 ± 0.24	0.46 ± 0.16
DEL	7B	117.0 ± 30.4	0.19 ± 0.06	5.06 ± 5.24	0.73 ± 0.14
	13B	87.1 ± 25.8	0.17 ± 0.04	3.59 ± 3.68	0.70 ± 0.13
	70B	33.3 ± 9.0	0.12 ± 0.04	3.23 ± 3.31	0.65 ± 0.15

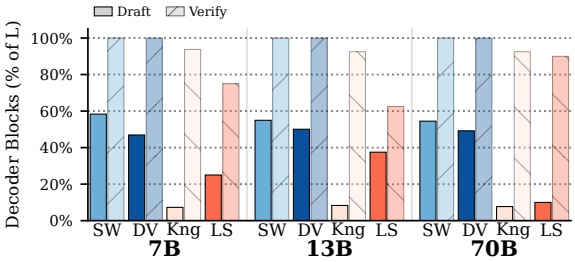
A.2. LayerSkip-Llama-2: CNN/DailyMail results



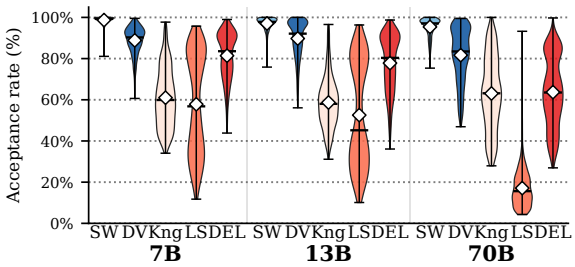
(a) Throughput (Normalized to baseline).



(b) Number of draft tokens per SD round.



(c) Draft and verify model size (Appendix A) for all techniques apart from DEL, normalized to the total number of decoder blocks. Verify size reflects only the blocks executed during verification.



(d) Acceptance rate per sample.

Figure 8. Results for *layerskip-llama-2* 7B, 13B & 70B models for the *CNN/DailyMail* dataset.

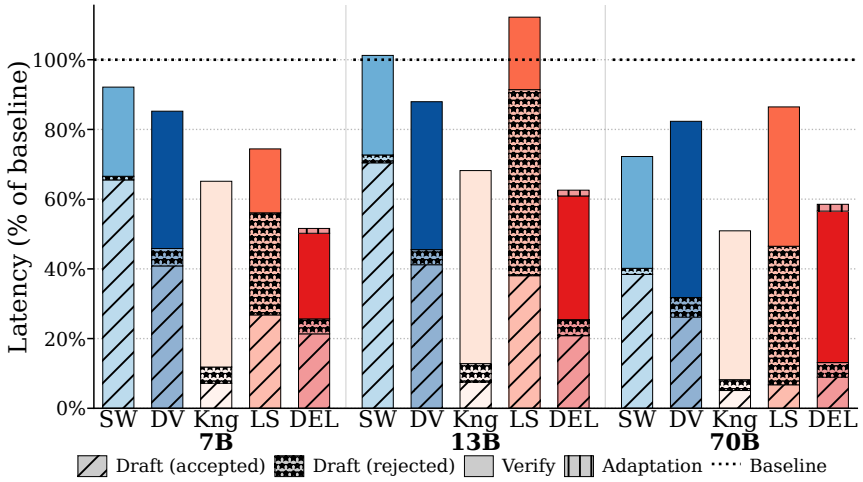


Figure 9. Latency breakdown, normalized to the baseline for *LayerSkip-llama-2* models running *CNN/DM*.

Figure 10. Draft model cost for *DEL* (left) and number of tokens during verification for *SWIFT* (right) on CNN/DM.

Table 4. Results for *layerskip-llama-2* 7B, 13B & 70B models for CNN/DailyMail.

		CNN/DM			
METHOD	SIZE	THROUGHPUT (TOK/S)	DRAFT SIZE (FRAC)	DRAFT LENGTH (TOK/ROUND)	ACCEPTANCE
BASELINE	7B	63.2 ± 0.2	–	–	–
	13B	50.4 ± 1.0	–	–	–
	70B	15.1 ± 0.4	–	–	–
SWIFT	7B	68.3 ± 3.7	0.58	4.37 ± 5.45	0.99 ± 0.02
	13B	52.1 ± 2.8	0.55	4.01 ± 5.07	0.97 ± 0.03
	70B	20.1 ± 1.8	0.54	2.50 ± 3.15	0.95 ± 0.04
D&V	7B	77.2 ± 9.0	0.47	2.99 ± 2.82	0.89 ± 0.07
	13B	61.4 ± 9.8	0.50	2.67 ± 3.11	0.90 ± 0.09
	70B	18.5 ± 2.8	0.49	1.41 ± 1.46	0.82 ± 0.13
KANGAROO	7B	99.3 ± 25.6	0.07	2.16 ± 1.64	0.61 ± 0.14
	13B	78.4 ± 16.9	0.08	2.16 ± 1.58	0.59 ± 0.12
	70B	31.4 ± 10.0	0.08	2.39 ± 1.81	0.63 ± 0.18
LAYERSKIP	7B	98.4 ± 35.8	0.25	11.90 ± 0.87	0.58 ± 0.24
	13B	56.3 ± 23.1	0.38	11.91 ± 0.83	0.53 ± 0.25
	70B	18.8 ± 6.7	0.10	11.90 ± 0.90	0.17 ± 0.09
DEL	7B	128.9 ± 27.6	0.20 ± 0.06	4.93 ± 5.70	0.81 ± 0.11
	13B	87.0 ± 16.1	0.21 ± 0.09	3.20 ± 3.58	0.78 ± 0.14
	70B	26.8 ± 6.1	0.13 ± 0.05	2.33 ± 2.38	0.64 ± 0.17

A.3. LayerSkip-Llama-2: GSM8k results

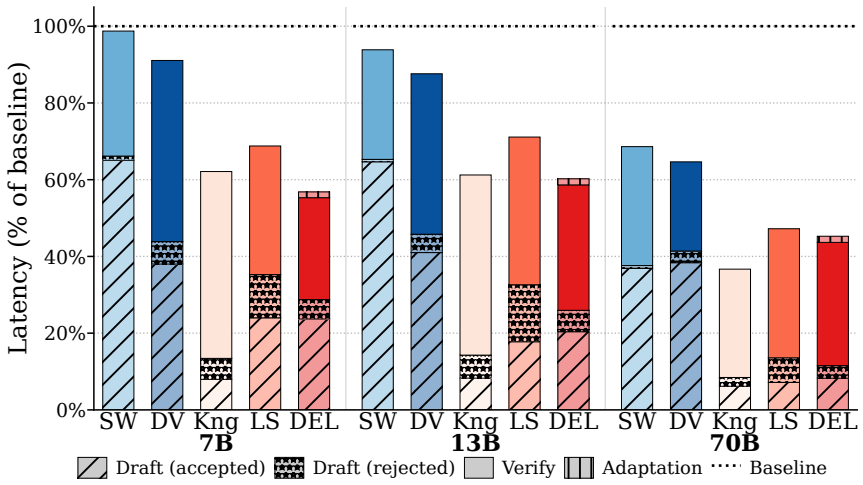
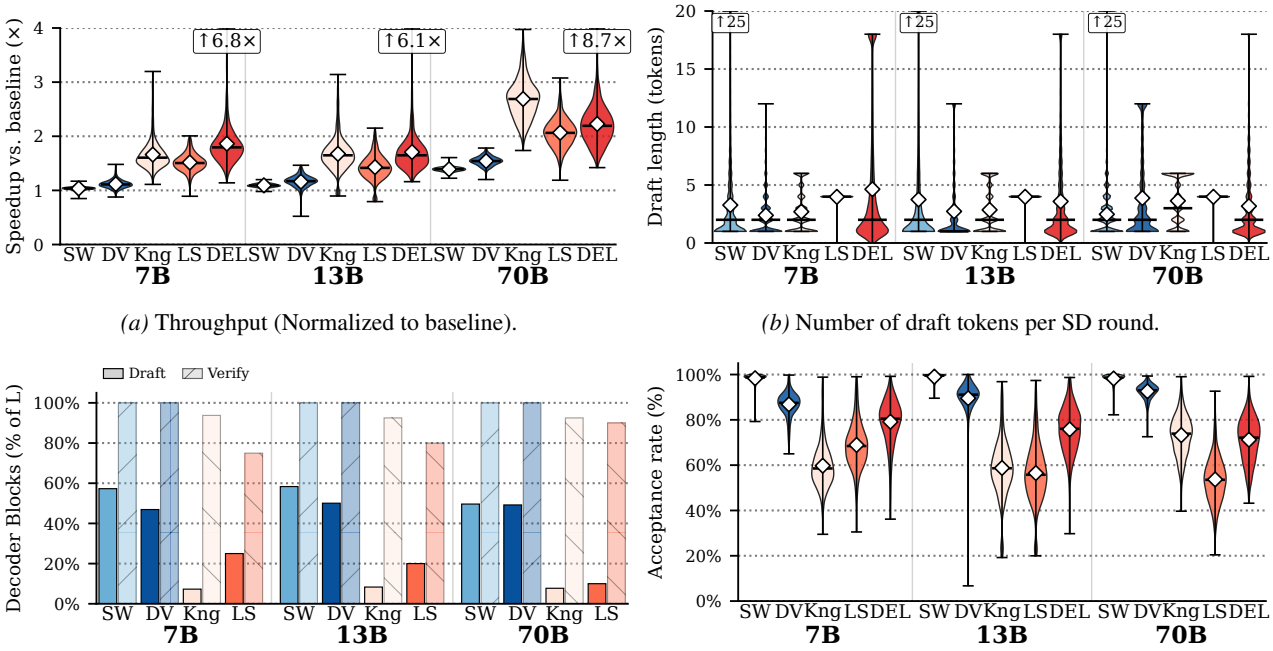


Figure 11. Latency breakdown, normalized to the baseline for *LayerSkip-llama-2* models running *GSM8k*.



(a) Throughput (Normalized to baseline).

(b) Number of draft tokens per SD round.

(c) Draft and verify model size for all techniques apart from DEL.

(d) Acceptance rate per sample.

Figure 12. Results for *layerskip-llama-2* 7B, 13B & 70B models for the *GSM 8k* dataset.

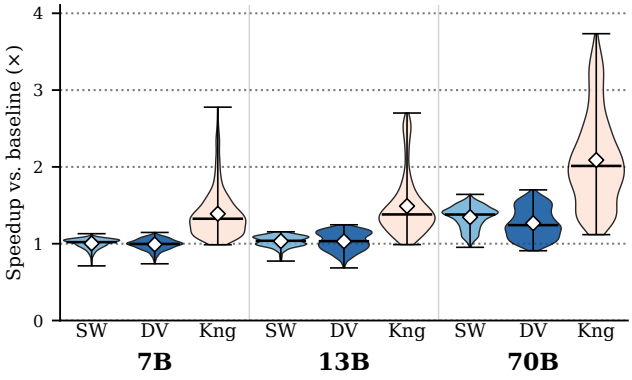
Figure 13. Draft model cost for *DEL* (left) and number of tokens during verification for *SWIFT* (right) on GSM8k.

Table 5. Results for *layerskip-llama-2* 7B, 13B & 70B models for the GSM8k dataset.

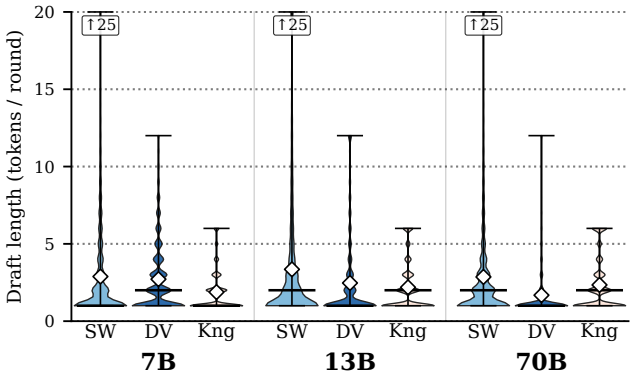
		GSM8K			
METHOD	SIZE	THROUGHPUT (TOK/S)	DRAFT SIZE (FRAC)	DRAFT LENGTH (TOK/ROUND)	ACCEPTANCE
BASELINE	7B	63.2 ± 0.2	–	–	–
	13B	52.2 ± 0.2	–	–	–
	70B	15.6 ± 0.0	–	–	–
SWIFT	7B	65.4 ± 1.9	0.57	3.32 ± 3.94	0.98 ± 0.02
	13B	57.1 ± 1.7	0.58	3.92 ± 5.44	0.99 ± 0.01
	70B	21.7 ± 0.6	0.50	2.48 ± 1.83	0.98 ± 0.02
D&V	7B	70.5 ± 4.8	0.47	2.38 ± 1.99	0.87 ± 0.05
	13B	60.6 ± 6.4	0.50	2.73 ± 2.97	0.90 ± 0.09
	70B	24.1 ± 1.3	0.49	3.87 ± 3.65	0.93 ± 0.03
KANGAROO	7B	105.0 ± 16.7	0.07	2.70 ± 1.82	0.60 ± 0.10
	13B	87.5 ± 16.3	0.08	2.85 ± 1.93	0.59 ± 0.12
	70B	42.0 ± 5.6	0.08	3.64 ± 2.10	0.73 ± 0.09
LAYERSKIP	7B	95.7 ± 10.1	0.25	3.98 ± 0.23	0.69 ± 0.10
	13B	74.6 ± 11.7	0.20	3.98 ± 0.22	0.56 ± 0.13
	70B	32.3 ± 4.0	0.10	3.98 ± 0.22	0.54 ± 0.10
DEL	7B	118.0 ± 26.7	0.22 ± 0.08	4.63 ± 5.43	0.79 ± 0.10
	13B	89.4 ± 18.4	0.21 ± 0.09	3.58 ± 3.96	0.76 ± 0.10
	70B	35.0 ± 7.2	0.11 ± 0.05	3.18 ± 3.24	0.71 ± 0.10

A.4. Standard Llama-2 results

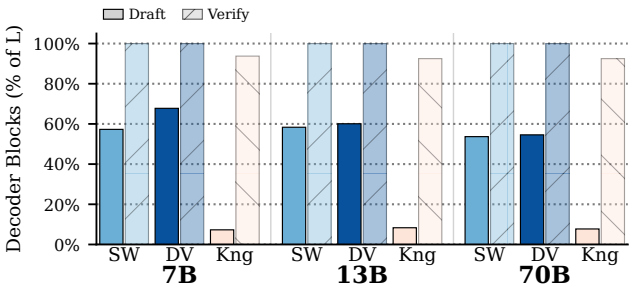
We report results for the standard (non-LayerSkip-trained) Llama-2 7B, 13B, and 70B models across all three datasets. LayerSkip and DEL are omitted for this family because their retraining recipe is specific to their released checkpoints.



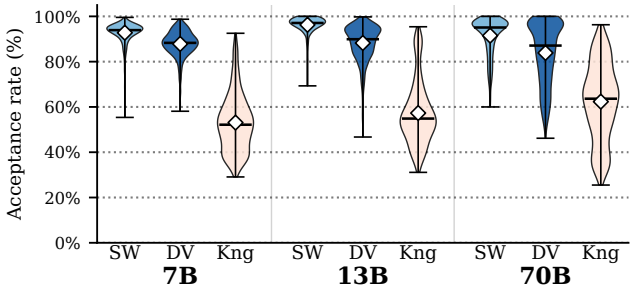
(a) Throughput (Normalized to baseline).



(b) Number of draft tokens per SD round



(c) Draft and verify model size.



(d) Acceptance rate per sample.

Figure 14. Results for standard *Llama-2* 7B, 13B & 70B models on *CNN/DailyMail*.

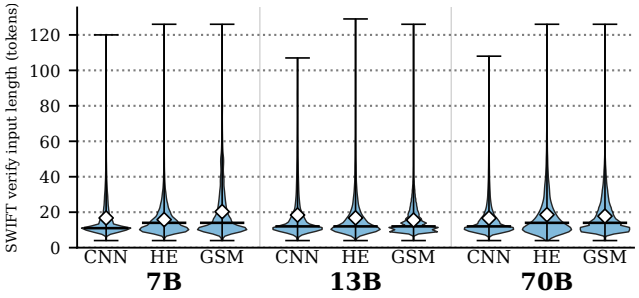
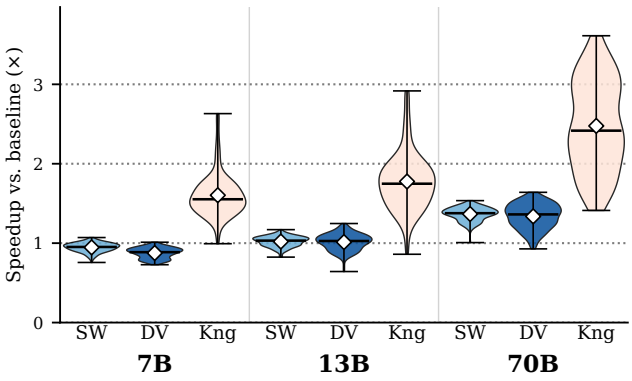
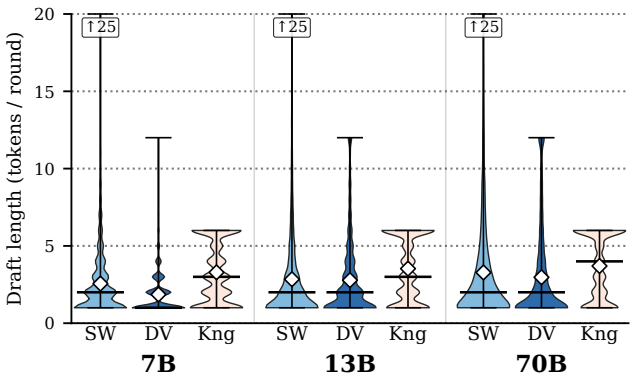


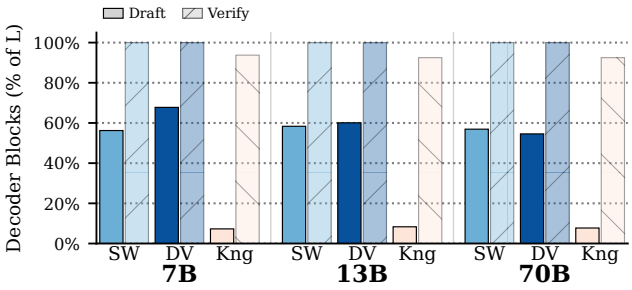
Figure 15. SWIFT's tree-drafted verify-input length for standard *Llama-2*.



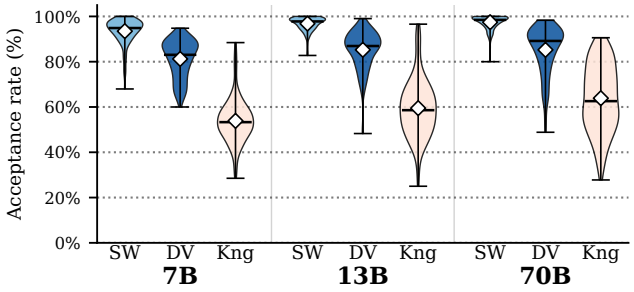
(a) Throughput (Normalized to baseline).



(b) Number of draft tokens per SD round.

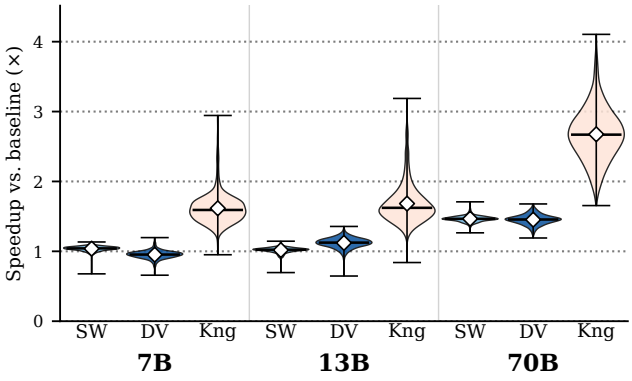


(c) Draft and verify model size.

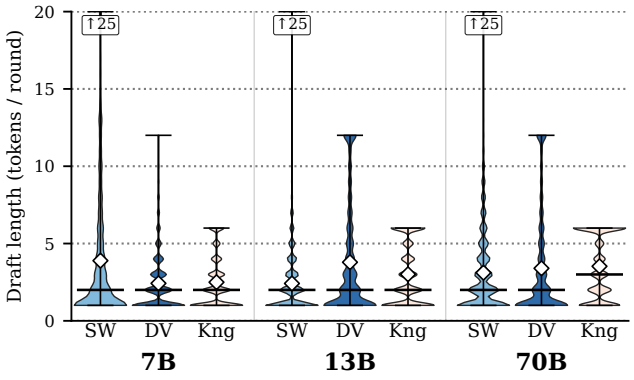


(d) Acceptance rate per sample.

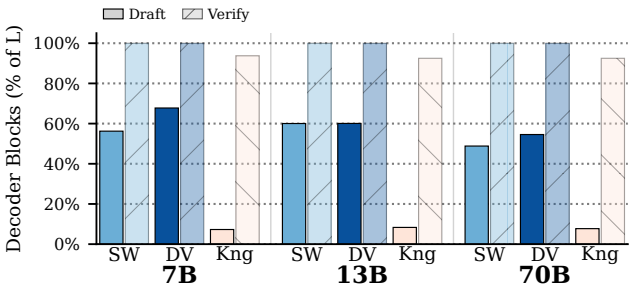
Figure 16. Results for standard *Llama-2* 7B, 13B & 70B models on *HumanEval*.



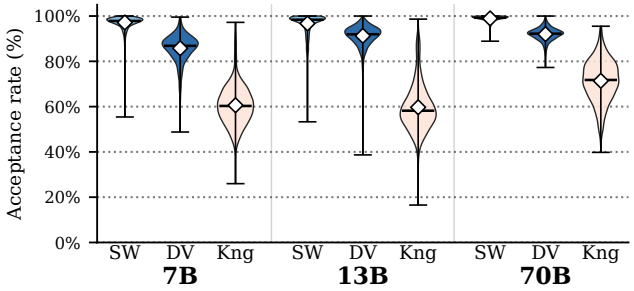
(a) Throughput (Normalized to baseline).



(b) Number of draft tokens per SD round.



(c) Draft and verify model size.



(d) Acceptance rate per sample.

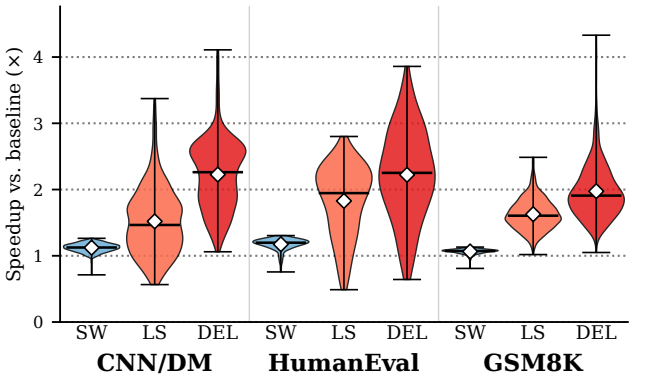
Figure 17. Results for standard *Llama-2* 7B, 13B & 70B models on *GSM8K*.

Table 6. Results for standard *Llama2* 7B, 13B & 70B models for all three datasets

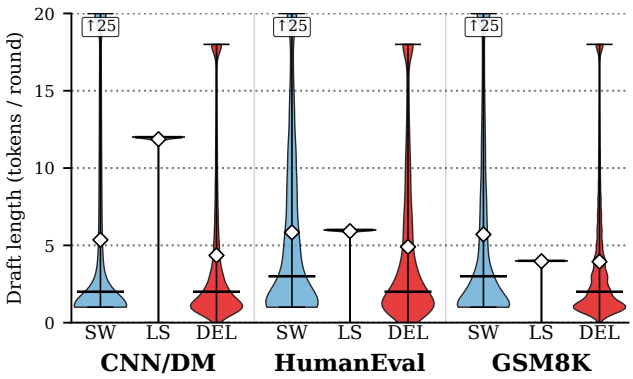
Method	Size	Throughput (tok/s)	Draft size (frac)	Draft length (tok/round)	Acceptance
CNN/DM					
Baseline	7B	62.3 ± 0.4	–	–	–
	13B	53.2 ± 0.2	–	–	–
	70B	15.1 ± 0.4	–	–	–
SWIFT	7B	62.6 ± 3.8	0.57	2.90 ± 3.35	0.93 ± 0.05
	13B	55.0 ± 3.7	0.58	3.41 ± 4.35	0.96 ± 0.04
	70B	20.3 ± 2.1	0.54	2.89 ± 3.47	0.91 ± 0.08
D&V	7B	61.7 ± 4.3	0.68	2.71 ± 2.23	0.88 ± 0.06
	13B	54.8 ± 6.4	0.60	2.47 ± 2.70	0.88 ± 0.08
	70B	19.1 ± 2.9	0.55	1.68 ± 1.96	0.84 ± 0.13
Kangaroo	7B	86.6 ± 19.0	0.07	1.88 ± 1.34	0.53 ± 0.13
	13B	79.2 ± 20.6	0.08	2.19 ± 1.60	0.57 ± 0.14
	70B	31.5 ± 9.0	0.08	2.34 ± 1.74	0.62 ± 0.17
Human Eval					
Baseline	7B	63.3 ± 0.2	–	–	–
	13B	52.3 ± 0.6	–	–	–
	70B	16.0 ± 0.1	–	–	–
SWIFT	7B	59.9 ± 3.6	0.56	2.56 ± 2.26	0.93 ± 0.06
	13B	53.4 ± 3.4	0.58	2.86 ± 3.21	0.97 ± 0.03
	70B	21.9 ± 1.3	0.57	3.32 ± 3.71	0.97 ± 0.03
D&V	7B	55.4 ± 4.2	0.68	1.86 ± 1.24	0.81 ± 0.08
	13B	53.0 ± 5.4	0.60	2.78 ± 2.55	0.85 ± 0.08
	70B	21.4 ± 2.5	0.55	2.97 ± 3.02	0.85 ± 0.10
Kangaroo	7B	101.5 ± 16.9	0.07	3.29 ± 1.97	0.54 ± 0.10
	13B	92.9 ± 18.8	0.08	3.53 ± 2.06	0.59 ± 0.13
	70B	39.6 ± 9.0	0.08	3.70 ± 2.13	0.64 ± 0.15
GSM8K					
Baseline	7B	62.9 ± 0.2	–	–	–
	13B	52.7 ± 0.2	–	–	–
	70B	15.5 ± 0.0	–	–	–
SWIFT	7B	65.2 ± 2.2	0.56	3.94 ± 4.38	0.97 ± 0.03
	13B	53.6 ± 2.0	0.60	2.42 ± 2.06	0.97 ± 0.04
	70B	22.7 ± 0.7	0.49	3.13 ± 2.77	0.99 ± 0.01
D&V	7B	59.8 ± 3.6	0.68	2.42 ± 1.95	0.86 ± 0.06
	13B	59.1 ± 3.8	0.60	3.79 ± 3.57	0.91 ± 0.04
	70B	22.5 ± 1.1	0.55	3.40 ± 3.39	0.92 ± 0.03
Kangaroo	7B	101.5 ± 13.6	0.07	2.50 ± 1.71	0.61 ± 0.09
	13B	88.7 ± 14.6	0.08	3.01 ± 1.96	0.60 ± 0.10
	70B	41.4 ± 5.7	0.08	3.53 ± 2.07	0.71 ± 0.09

A.5. LayerSkip-Llama-3 (8B) results

We report results for the LayerSkip-Llama-3 8B checkpoint released by Elhoushi et al. (2024) on all three datasets. As *Kangaroo* requires adapter training and *Draft & Verify* requires Bayesian Optimization, only *SWIFT*, *LayerSkip*, and *DEL* were available for this model.

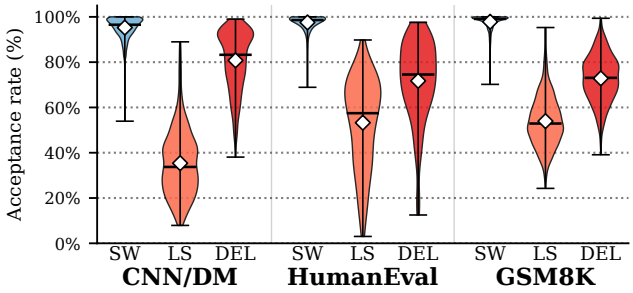


(a) Throughput (Normalized to baseline).



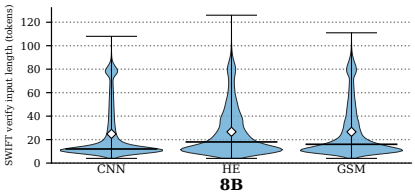
(b) Number of draft tokens per SD round.

(c) Draft and verify model size.

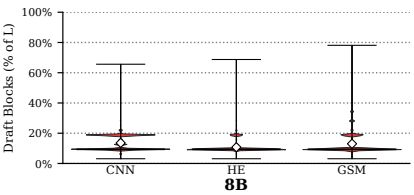


(d) Acceptance rate per sample.

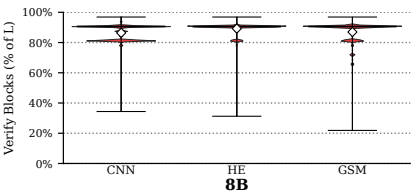
Figure 18. Results for *layerskip-llama-3-8B* on *CNN/DailyMail, HumanEval, and GSM8K*.



(a) SWIFT verify-input length.



(b) DEL draft-model size.



(c) DEL verify model size.

Figure 19. Per-method diagnostics for *layerskip-llama-3-8B* on *CNN/DailyMail, HumanEval, and GSM8K*.

Table 7. Results for *layerskip-llama-3* 8B for all three datasets

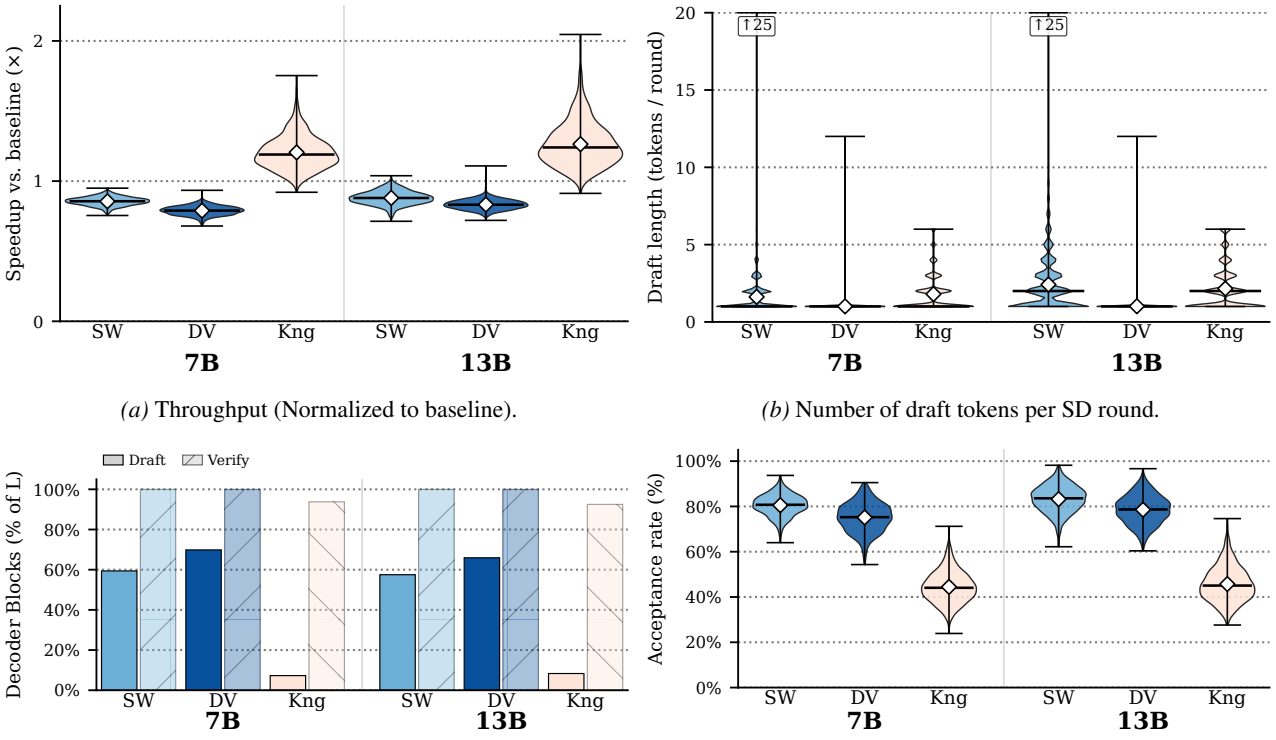
Method	Size	Throughput (tok/s)	Draft size (frac)	Draft length (tok/round)	Acceptance
CNN/DM					
Baseline	8B	58.4 ± 0.2	–	–	–
SWIFT	8B	65.5 ± 3.8	0.61	5.80 ± 7.62	0.95 ± 0.05
LayerSkip	8B	88.7 ± 29.3	0.12	11.87 ± 1.00	0.35 ± 0.15
DEL	8B	130.0 ± 30.9	0.14 ± 0.05	4.35 ± 5.25	0.81 ± 0.13
Human Eval					
Baseline	8B	57.6 ± 0.2	–	–	–
SWIFT	8B	67.6 ± 5.2	0.63	6.06 ± 6.35	0.98 ± 0.04
LayerSkip	8B	105.3 ± 30.4	0.12	5.92 ± 0.54	0.53 ± 0.20
DEL	8B	128.0 ± 39.0	0.11 ± 0.04	4.90 ± 5.25	0.72 ± 0.18
GSM8K					
Baseline	8B	58.2 ± 0.2	–	–	–
SWIFT	8B	62.0 ± 2.1	0.53	6.05 ± 6.96	0.98 ± 0.03
LayerSkip	8B	94.7 ± 13.4	0.12	3.97 ± 0.24	0.54 ± 0.11
DEL	8B	114.9 ± 24.1	0.13 ± 0.07	3.94 ± 4.52	0.73 ± 0.10

A.6. Vicuna-v1.3 results

We report results for the chat-tuned Vicuna-v1.3 7B and 13B models. Unlike base Llama-2, which typically generates close to the 512-token cap, Vicuna's chat fine-tuning causes the model to emit End-of-Sequence(EOS) token (</s>) before the cap (Table 8).

Table 8. Average number of output tokens for Vicuna-v1.3. Vicuna terminates early via </s> on all three datasets.

Size	CNN/DM	GSM8K	HumanEval
7B	184	171	311
13B	171	142	306



(a) Throughput (Normalized to baseline).

(b) Number of draft tokens per SD round.

(c) Draft and verify model size.

(d) Acceptance rate per sample.

Figure 20. Results for *Vicuna-v1.3* 7B & 13B on *CNN/DailyMail*.

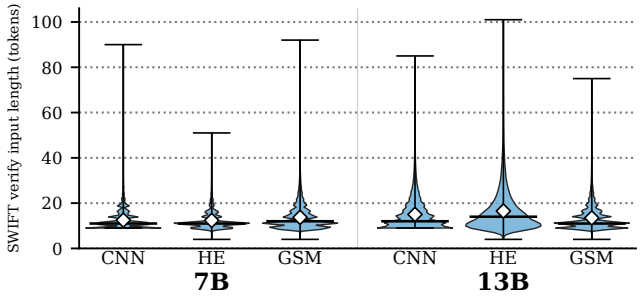
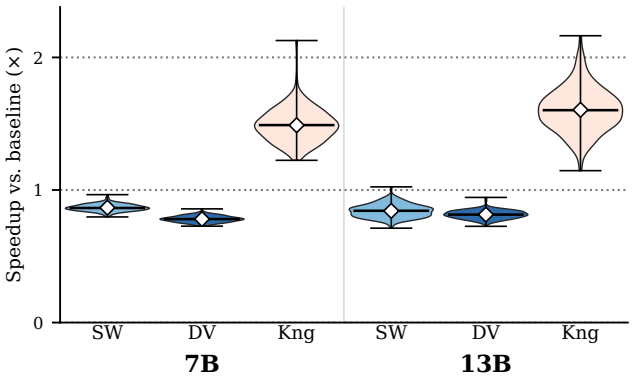
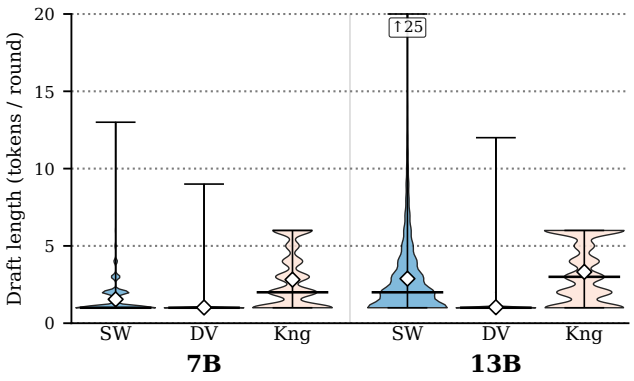


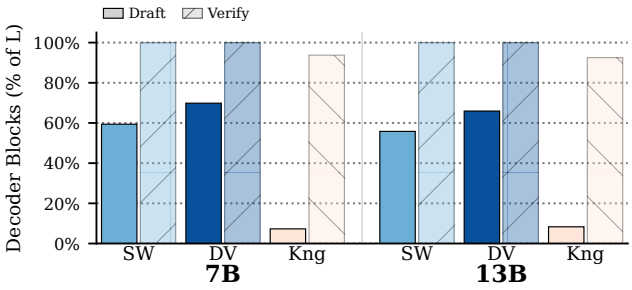
Figure 21. SWIFT's tree-drafted verify-input length for *Vicuna-v1.3*.



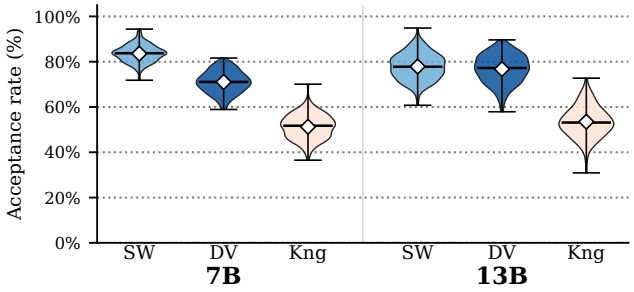
(a) Throughput (Normalized to baseline).



(b) Number of draft tokens per SD round.

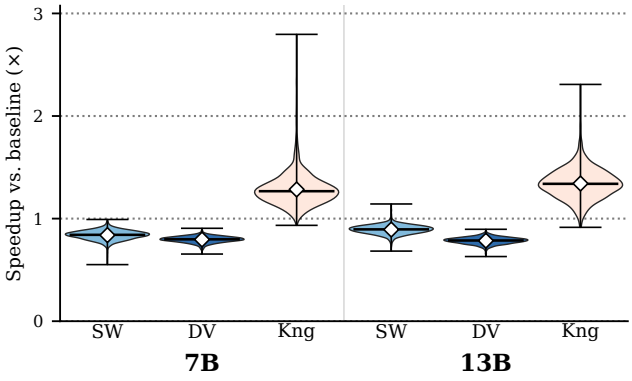


(c) Draft and verify model size.

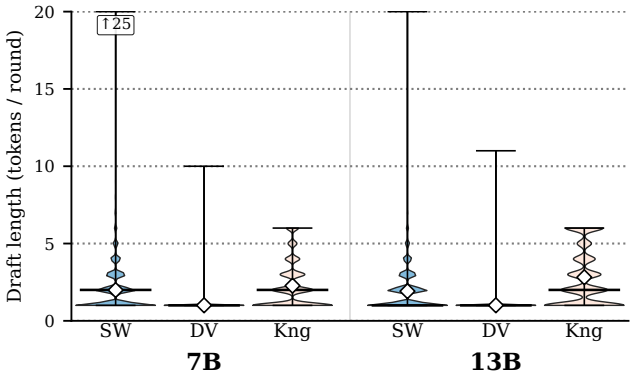


(d) Acceptance rate per sample.

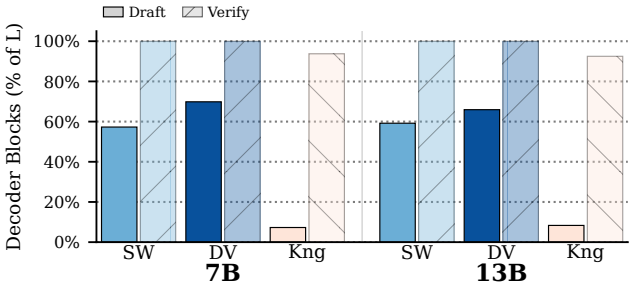
Figure 22. Results for *Vicuna-v1.3* 7B & 13B on *HumanEval*.



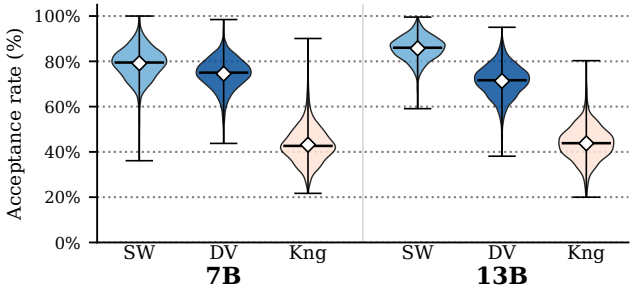
(a) Throughput (Normalized to baseline).



(b) Number of draft tokens per SD round.



(c) Draft and verify model size.



(d) Acceptance rate per sample.

Figure 23. Results for *Vicuna-v1.3* 7B & 13B on *GSM8K*.

Table 9. Results for *Vicuna-v1.3* 7B & 13B models for all three datasets

Method	Size	Throughput (tok/s)	Draft size (frac)	Draft length (tok/round)	Acceptance
		CNN/DM			
Baseline	7B	62.8 ± 0.3	–	–	–
	13B	52.2 ± 0.4	–	–	–
SWIFT	7B	53.7 ± 1.9	0.59	1.62 ± 0.96	0.81 ± 0.05
	13B	45.9 ± 2.5	0.57	2.42 ± 1.86	0.83 ± 0.06
D&V	7B	49.5 ± 2.0	0.70	1.01 ± 0.17	0.75 ± 0.06
	13B	43.5 ± 1.9	0.66	1.01 ± 0.28	0.79 ± 0.06
Kangaroo	7B	75.6 ± 7.4	0.07	1.80 ± 1.16	0.44 ± 0.07
	13B	66.0 ± 8.1	0.08	2.15 ± 1.45	0.46 ± 0.08
		Human Eval			
Baseline	7B	63.3 ± 0.2	–	–	–
	13B	52.6 ± 0.2	–	–	–
SWIFT	7B	54.9 ± 1.6	0.59	1.55 ± 0.97	0.84 ± 0.04
	13B	44.3 ± 2.6	0.56	2.89 ± 2.79	0.78 ± 0.06
D&V	7B	49.4 ± 1.5	0.70	1.02 ± 0.21	0.71 ± 0.05
	13B	42.8 ± 1.7	0.66	1.03 ± 0.37	0.77 ± 0.06
Kangaroo	7B	94.2 ± 7.8	0.07	2.82 ± 1.83	0.51 ± 0.05
	13B	84.4 ± 8.7	0.08	3.32 ± 1.96	0.53 ± 0.07
		GSM8K			
Baseline	7B	62.8 ± 0.2	–	–	–
	13B	52.5 ± 0.3	–	–	–
SWIFT	7B	52.7 ± 2.8	0.57	1.99 ± 1.38	0.79 ± 0.07
	13B	46.9 ± 2.2	0.59	1.92 ± 1.29	0.86 ± 0.05
D&V	7B	50.0 ± 1.9	0.70	1.00 ± 0.11	0.75 ± 0.07
	13B	41.2 ± 1.7	0.66	1.01 ± 0.15	0.71 ± 0.07
Kangaroo	7B	80.6 ± 9.8	0.07	2.27 ± 1.53	0.43 ± 0.08
	13B	70.4 ± 7.8	0.08	2.82 ± 1.82	0.44 ± 0.07

B. Full configuration

This section lists the full configuration used for our evaluation.

B.1. Hardware and Software environment

This subsection documents the GPU hardware, CUDA stack, and Python package versions used across all reported runs.

Hardware. Models are evaluated on a server with the following configuration:

1. $2\times$ AMD EPYC 9655 CPUs, running at $2.6GHz$, for a total of 192 cores.
2. $4\times$ NVidia H100 SXM GPUs with 80GB of VRAM, connected via NVLink.
3. 768GB of RAM.

The 7B and 13B models are evaluated on a single NVIDIA H100 80 GB (SXM5), while the 70B models are sharded across four H100s on a single system using HuggingFace's `device_map="auto"` placement. Each model is loaded in its native precision: FP16 for the standard Llama-2 (7B/13B/70B), Vicuna-1.3 (7B/13B), and LayerSkip-Llama-2 (7B/13B) models, and BF16 for LayerSkip-Llama-2-70B and LayerSkip-Llama-3-8B.

Software. We use PyTorch 2.11.0 and CUDA 12.9 for all methods. The HuggingFace `transformers` version is the only main dependency that varies across methods: *Baseline*, *SWIFT*, *Draft & Verify*, *LayerSkip*, and *DEL* use `transformers` 4.45.2, while *Kangaroo* requires `transformers` 4.34.0 because the released codebase is pinned to that version. The exact versions used in our evaluation is given in Table 10.

Determinism and decoding. Throughout the paper we run greedy decoding (`temperature` $= 0$) with a fixed seed (42) and a fixed `max_new_tokens` $= 512$. Self-speculative decoding is output-equivalent to standard autoregressive decoding under greedy sampling (Leviathan et al., 2023; Chen et al., 2023; Zhang et al., 2024). Therefore, all throughput differences reflect inference efficiency alone.

Table 10. Per-method software stack and source code. PyTorch 2.11.0 and CUDA 12.9 are shared across all methods.

Method	`transformers`	`accelerate`	`datasets`	`hf_hub`	GitHub Repository	Commit
Baseline	4.45.2	1.13.0	4.8.4	0.36.2	—	—
SWIFT	4.45.2	1.13.0	4.8.4	0.36.2	`hemingkx/SWIFT`	`88caed7`
Draft & Verify	4.45.2	1.13.0	4.8.4	0.36.2	`dilab-zju/self-speculative-decoding`	`688db8b`
LayerSkip	4.45.2	1.0.1	3.0.1	0.36.2	`facebookresearch/LayerSkip`	`f42c669`
DEL	4.45.2	1.0.1	3.0.1	0.36.2	`hoenza/DEL`	`93feac5`
Kangaroo	4.34.0	0.25.0	3.0.1	0.25.1	`Equationliu/Kangaroo`	`4f0949d`

B.2. Configurations used per technique

This subsection lists the hyperparameters used for each self-speculative decoding method. Each value cites its source: the original paper, the reference implementation (source code in Table 10), our own calibration, or training. Our baseline is autoregressive decoding using HuggingFace's `model.generate()`.

B.2.1. SWIFT

SWIFT requires no pre-inference training or calibration. It discovers its per-layer skip pattern via an online warm-up at the start of each run and exposes a single tunable hyperparameter.

Online warm-up. An online warm-up phase at the start of inference discovers a per-layer attention/FFN skip pattern via random search at every step and Bayesian optimization every $\beta = 25$ steps (paper default). The skip pattern is then frozen for the remainder of generation. Warm-up time is excluded from throughput measurements; the impact is negligible since SWIFT converges within 1–10 samples. The converged skip patterns for each (base, size, dataset) cell are listed in Table 11.

Inference-time hyperparameters. The only hyperparameter we set is `swift_skip_ratio` (r in the *SWIFT* paper), the target fraction of sub-layers to skip. We follow the values reported by Xia et al. (2025): $r = 0.45$ for 7B and 13B and

Table 11. Converged *SWIFT* skip patterns for every (base, size, dataset) configuration. Indices are 0-based decoder-block positions. Total layers: 7B & 8B = 32, 13B = 40, 70B = 80.

SIZE	DATASET	ATTN-SKIP LAYERS	FFN-SKIP LAYERS
		LAYERSKIP-LLAMA-2	
7B	CNN/DM	8, 10, 12, 14, 17, 19, 20, 22, 24 - 27, 29, 30	3, 9, 10, 12, 18, 19, 21, 23, 25, 27 - 30
7B	GSM8K	2, 14, 16 - 18, 20, 21, 23, 25, 27 - 30	8, 14, 16 - 26, 30
7B	HUMANEVAL	5, 9, 10, 15, 17 - 28, 30	8, 10, 15, 17, 19, 20, 25, 27, 29, 30
13B	CNN/DM	6, 7, 15, 17, 23, 26, 28, 29, 31 - 34, 36, 38	4, 7, 9 - 12, 15, 17, 18, 20, 22, 23, 25 - 27, 29, 30, 33, 37, 38
13B	GSM8K	11, 15, 16, 20, 22 - 26, 28, 29, 31 - 35, 37, 38	7, 10, 11, 16, 19, 20, 23, 24, 26 - 28, 31, 32, 34, 35, 37
13B	HUMANEVAL	6, 7, 8, 11, 14, 20, 22 - 26, 28, 32 - 35, 38	7, 9, 11, 12, 15, 20, 22 - 24, 28 - 31, 34, 35, 37, 38
70B	CNN/DM	9, 13, 16 - 21, 23, 24, 26, 28, 29, 32 - 34, 36, 38, 40, 43, 48 - 50, 52 - 54, 58 - 61, 63 - 65, 67, 68, 70 - 72, 74 - 78	6, 10, 12, 13, 16, 20 - 22, 24, 25, 29, 33, 34, 36, 37, 40, 41, 44, 47 - 51, 57, 58, 61, 63, 64, 67 - 69, 71, 72, 75, 76
70B	GSM8K	5 - 7, 9, 10, 13, 19, 21, 22, 24, 25, 28, 29, 34 - 37, 41 - 44, 46 - 50, 52, 54, 55, 59, 61, 62, 64, 65, 68, 69, 78	4, 7 - 9, 11, 15 - 17, 21, 26 - 29, 31, 36, 37, 39, 41 - 44, 46 - 48, 50 - 52, 57 - 59, 62, 63, 65 - 72, 75
70B	HUMANEVAL	4, 7, 11, 19, 20, 25 - 30, 32 - 34, 36, 37, 41 - 43, 45, 46, 49, 50, 53 - 59, 61, 63 - 65, 68 - 70, 72, 73, 75, 76	3, 5, 15, 17, 18, 20, 22, 25, 27 - 32, 34, 37, 38, 40, 44, 45, 48, 50, 52, 55, 56, 58, 62 - 65, 67, 71, 73 - 75, 77, 78
		LAYERSKIP-LLAMA-3	
8B	CNN/DM	3, 5 - 8, 10, 12, 13, 16, 17, 19, 22, 25, 29, 30	1, 3, 7, 13, 15, 18 - 21, 25, 27, 28
8B	GSM8K	7, 8, 14, 16, 19, 20, 22, 23, 26, 27, 30	1 - 3, 5, 6, 8 - 10, 13, 17, 19, 23, 25, 27, 29, 30
8B	HUMANEVAL	1, 3, 4, 7, 8, 11, 14, 15, 18, 19, 21 - 23, 25, 29, 30	4, 7, 8, 19, 21, 22, 24, 25, 27, 29, 30
		LLAMA-2	
7B	CNN/DM	9, 12, 14, 15, 18 - 21, 23 - 25, 27, 30	7, 11, 13 - 15, 19 - 23, 25 - 28
7B	GSM8K	8, 13 - 15, 20, 21, 23, 25 - 29	2, 8, 11 - 15, 19 - 24, 26, 27
7B	HUMANEVAL	7, 9 - 11, 15, 18 - 21, 25, 27, 28	7, 9, 11 - 13, 16, 17, 19 - 24, 26, 27
13B	CNN/DM	8, 11, 12, 14, 17, 19, 21 - 25, 27, 29 - 32, 34, 35	6, 9, 10, 13, 14, 17, 20, 22, 23, 25, 27, 29 - 31, 33, 37
13B	GSM8K	1, 5, 8, 12, 15, 16, 19, 22 - 24, 27 - 30, 32 - 37	8, 10, 11, 13, 17, 18, 20, 22, 26 - 28, 30, 31, 35
13B	HUMANEVAL	5, 12, 15, 16, 19, 21 - 23, 25 - 29, 32, 33, 35 - 37	9, 10, 12 - 14, 17, 20, 21, 23, 24, 27, 28, 31, 33, 35, 36
70B	CNN/DM	11, 14, 16 - 18, 20, 21, 23, 26 - 32, 35 - 37, 39 - 41, 43, 46, 50 - 52, 54, 56 - 59, 61 - 63, 65, 69 - 74, 78	7, 11, 13 - 16, 22, 23, 25, 26, 29, 30, 34, 36, 42 - 47, 49 - 57, 59, 61, 62, 64, 66, 69, 70
70B	GSM8K	9, 11, 12, 16 - 18, 20, 22, 23, 27, 28, 30 - 32, 35, 37, 39, 42, 45, 47, 50, 51, 53, 58, 59, 61 - 64, 66, 69, 71, 72, 75, 77, 78	6 - 10, 12, 14 - 17, 20, 22, 23, 27, 30, 31, 34, 35, 41, 42, 44 - 48, 50 - 52, 54 - 56, 58 - 60, 61, 64, 66 - 71
70B	HUMANEVAL	3, 5, 9 - 13, 16 - 18, 20, 22 - 25, 27, 29 - 32, 35, 36, 42, 45 - 48, 50, 51, 53, 54, 57, 59 - 61, 63, 64, 66, 68 - 71, 73, 75, 77, 78	11, 13, 14, 17, 20, 21, 23, 24, 28, 31 - 33, 36, 42, 43, 45, 50 - 52, 54, 58 - 66, 72, 75, 76
		VICUNA-V1.3	
7B	CNN/DM	4, 5, 8, 11, 12, 15, 16, 19, 22 - 24, 27 - 30	7, 9, 12, 17, 21, 22, 24, 25, 27 - 30
7B	GSM8K	5, 10, 11, 13, 15, 16, 20 - 23, 26, 28, 29	10 - 14, 16, 18, 20 - 23, 25 - 27
7B	HUMANEVAL	3, 6, 10, 11, 14, 15, 18, 19, 21, 24 - 26, 28 - 30	5, 8, 14 - 16, 19, 21, 24, 25, 27 - 29
13B	CNN/DM	4, 5, 9, 14, 17, 19, 20, 22, 24, 27, 29, 31 - 34, 36, 38	5, 8, 10, 14, 15, 17, 20, 22, 23, 26, 28, 29, 32 - 34, 37, 38
13B	GSM8K	7, 11, 14, 15, 17 - 19, 22 - 29, 33, 36 - 38	4, 7, 9 - 11, 13, 15, 16, 24, 26, 28, 30, 31, 34, 36
13B	HUMANEVAL	6, 7, 9, 17, 18, 20, 25 - 27, 29, 31 - 34, 37	7, 8, 12 - 14, 16, 18, 19, 24 - 26, 28, 29, 31 - 34, 36, 37

$r = 0.50$ for 70B. We keep the default for the remaining hyperparameters: a maximum draft length of $d_{max} = 25$ and the draft-stop threshold $\epsilon = 0.8$.

B.2.2. DRAFT & VERIFY

Draft & Verify requires no retraining, but fixes its skip pattern via a one-time offline Bayesian-optimization (BO) per model. We set two paper-default hyperparameters at inference.

Offline calibration. The optimizer acts at sub-layer granularity, bypassing attention and FFN layers. We follow the authors' protocol: BO is run for 1,000 iterations on a mixed CNN/DailyMail + XSum target task, once per model, and the resulting pattern is reused across all three evaluation datasets. The full skip patterns are reported in Table 12.

BO calibration takes ∼2.5 hours per 13B model and ∼6 hours for the 70B model on A100 GPUs (Zhang et al., 2024); in our runs on H100 we observe times of a similar order. This cost is paid once per model and excluded from throughput, but it is the main practical drawback compared to *SWIFT*'s online optimization.

Inference-time hyperparameters. The two hyperparameters we set at inference are `max_step_draft` $= 12$ (K in the paper, the maximum draft length) and `th_stop_draft` $= 0.6$ (the initial draft-stop threshold ϵ_0). Both are recommended from the paper (Zhang et al., 2024). We enable the adaptive draft-exit mechanism (`auto_th_stop_draft` = `true`), which updates ϵ online. D&V's skip pattern is fixed, but its draft length and draft-stop threshold are dynamic at inference.

Table 12. Draft & Verify skip patterns for every configuration. Indices are 0-based decoder-block positions.

SIZE	ATTN-SKIP LAYERS	FFN-SKIP LAYERS
	LAYERSKIP-LLAMA-2	
7B	2 - 4, 9, 10, 12, 15, 17 - 19, 21 - 31	3, 4, 9 - 11, 15, 18, 22, 24, 25, 27 - 31
13B	2, 6, 8, 10, 11, 14, 16, 18, 19, 22 - 38	5, 9, 10, 14, 16, 18, 19, 24 - 27, 29, 32, 35 - 38
70B	5, 9, 11, 12, 14, 16 - 18, 20 - 23, 25, 27 - 29, 31 - 35, 37, 41, 42, 47 - 59, 61 - 69, 71 - 75, 77, 78	8, 11, 13, 15 - 18, 21, 23 - 25, 28, 30, 33 - 36, 42, 48, 49, 51, 52, 55 - 57, 59 - 64, 66, 67, 69, 70, 72, 74, 76
	LLAMA-2	
7B	2, 4, 6, 9, 10, 12, 14, 17 - 19, 22 - 25, 28 - 30	2, 8, 9, 11, 12, 15, 23
13B	2, 6, 8 - 11, 14, 18, 20, 22 - 36	2, 8 - 11, 14, 17, 20, 23, 26 - 28
70B	7, 8, 11, 13 - 23, 26, 28 - 32, 35 - 37, 41, 42, 45 - 47, 50 - 54, 56 - 61, 63 - 66, 68 - 70, 72 - 75, 78, 79	8, 10, 11, 13, 14, 16, 20 - 23, 26 - 28, 30 - 32, 37, 42, 47 - 51, 55, 57, 59 - 62, 64, 70 - 72
	VICUNA-V1.3	
7B	3, 5, 6, 8, 9, 16, 20 - 23, 25 - 31	8 - 10, 13, 25, 29
13B	3 - 5, 7, 9, 11, 16, 18, 22, 23, 25, 27 - 38	4, 6, 8, 11, 18, 22, 23, 32, 37

B.2.3. LAYERSKIP

LayerSkip requires the authors' early-exit-trained checkpoints (`facebook/layerskip-llama-2-{7B,13B,70B}`) and exposes two hyperparameters: the exit decoder block E and the static draft length d (number of draft tokens per round).

Inference-time hyperparameters. We use the values reported by Elhoushi et al. (2024) (Table 1) whenever available. This covers the 7B and 13B models on CNN/DM and HumanEval. For the remaining (size, dataset) pairs (GSM8k across all sizes, and all 70B entries), we selected (E, d) via a 2D grid sweep on 20 samples per dataset, choosing the pair that maximized throughput. Per-(size, dataset) values are listed in Table 13.

Table 13. LayerSkip hyperparameters per model size and dataset. E = exit layer, d = draft length. *Paper*: value from Table 1 of Elhoushi et al. (2024). *Sweep*: tuned on our hardware via a 2D (E, d) grid.

SIZE	DATASET	E	d	SOURCE
7B	CNN/DM	8	12	PAPER
7B	HUMANEVAL	8	6	PAPER
7B	GSM8K	8	4	SWEEP
13B	CNN/DM	15	12	PAPER
13B	HUMANEVAL	7	4	PAPER
13B	GSM8K	8	4	SWEEP
70B	CNN/DM	8	12	SWEEP
70B	HUMANEVAL	8	4	SWEEP
70B	GSM8K	8	4	SWEEP

B.2.4. DEL

DEL uses the same LayerSkip-trained checkpoints as *LayerSkip*, but does not require per-(model, dataset) tuning, since it adapts its hyperparameters each round at runtime.

Inference-time hyperparameters. DEL adapts its three runtime knobs each round — the exit block E, draft length d, and draft-stop threshold ϵ — using cached intermediate hidden states and their statistics. We use the values reported by Zarch et al. (2025) for the algorithm's own meta-parameters: $d_{\max} = 18$ (maximum draft length), $\omega = 0.95$ (EMA decay over historical acceptance and confidence statistics).

B.2.5. KANGAROO

At inference time, *Kangaroo* constructs the draft model with the first E decoder blocks of the full model followed by a small *adapter* module (a single multi-head attention layer trained to bridge the early-exit hidden state and the LM head). The adapter must be trained per-model; the full model itself is frozen throughout.

Adapter training. Each adapter is trained in two phases.

- *Phase 1 (data generation):* we run the full model on the ShareGPT dataset used by the original paper to materialize hidden states at the chosen exit decoder block.
- *Phase 2 (adapter training):* the light-weight adapter is trained with the loss function from Liu et al. (2024a), using their default optimizer settings and a cosine schedule for 20 epochs.

We deploy the final epoch's checkpoint.

Inference-time hyperparameters. The configurable hyperparameters at inference are: exit decoder block E, draft-stop threshold ϵ, and maximum draft length d_{max}. We use the values recommended by Liu et al. (2024a): `threshold` $\epsilon = 0.6$ and max draft length $d_{max} = 6$ for every (base, size, dataset) cell. The exit block is per model size: $E = 2$ for 7B, $E = 3$ for 13B, and $E = 6$ for 70B.

Adapter sources. For `lmsys/vicuna-7b-v1.3` and `lmsys/vicuna-13b-v1.3` we use the adapters released by the authors directly. For the six other 7B/13B/70B models (`meta-llama/Llama-2-7b-hf`, `meta-llama/Llama-2-13b-hf`, `meta-llama/Llama-2-70b-hf`, `facebook/layerskip-llama-2-7B`, `facebook/layerskip-llama-2-13B`, and `facebook/layerskip-llama-2-70B`) we trained the adapters ourselves following the steps above.

B.3. Datasets

We evaluate on three HuggingFace Hub benchmarks (Table 14). Samples are used as released and inserted directly into the templates below.

Benchmark	HF identifier	Split	N	Selection
CNN/DailyMail (Hermann et al., 2015; See et al., 2017)	`abisee/cnn_dailymail` (v3.0.0)	test (11,490)	1,000	first 1,000
HumanEval (Chen et al., 2021)	`openai/openai_humaneval`	test (164)	164	full split
GSM8K (Cobbe et al., 2021)	`openai/gsm8k` (`main`)	test (1,319)	1,319	full split

Table 14. Evaluation slices. CNN/DailyMail is sub-sampled; HumanEval and GSM8K are evaluated in full.

- **HumanEval** (raw prompt). The HumanEval `prompt` field (docstring + function signature) is used as-is. No exemplar or instruction is added. Every prompt is under 300 tokens and fits inside our context budget.
- **CNN/DailyMail** (1-shot). The prompt is:

  `Article: ⟨exemplar_article⟩\n\nSummary: ⟨exemplar_summary⟩\n\nArticle: ⟨test_article⟩\n\nSummary:`

  The Llama-2 and LayerSkip-Llama-2 bases are not instruction-tuned, so an explicit instruction like "Summarize the following article." is ignored. We use a single in-context example instead. The exemplar is the shortest CNN/DM train pair by Llama-2 token count and is fixed across all runs. It uses ≈130 tokens (83 article + 31 summary), leaving ∼1,370 tokens for the test article under the 1,504-token input cap derived below.
- **GSM8K** (5-shot chain-of-thought). We use the chain-of-thought format from Wei et al. (2022) as implemented by *DEL* upstream: five "`Question: ...\nAnswer: ...\n`" shots followed by a blank line and the test "`Question: ...\nAnswer:`". Each exemplar answer ends with "`#### ⟨number⟩`", and the model imitates this format after its reasoning steps. We do not stop on this marker; generation runs to the full 512-token cap so throughput is measured over a uniform window. The five shots are the shortest train pairs by Llama-2 token count, totaling ≈360 tokens.
- **Vicuna chat-tuned bases.** Vicuna-v1.3 wraps the user content in FastChat's single-turn template (Chiang et al., 2023)[2]:

  `A chat between a curious user and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the user's questions. USER: ⟨instruction⟩\n\n⟨input⟩ ASSISTANT:`

  The instruction is task-specific (e.g., "`Summarize the following article:`", "`Complete the following Python function:`"). The input is the test article, code prefix, or full 5-shot GSM8K block. We drop the CNN/DM exemplar since chat-tuned models follow the instruction directly, but keep the GSM8K shots since chain-of-thought still helps.

[2]The system prompt and `USER:`/`ASSISTANT:` markers come from FastChat's `conversation.py` (`vicuna_v1.1` template, also used for v1.3): https://github.com/lm-sys/FastChat/blob/main/fastchat/conversation.py.

Generation budget and context cap. Output is capped at $\texttt{max\_new\_tokens} = 512$ across all runs. The context cap is 2,048 tokens, set by the smallest `max_position_embeddings` across our bases (Table 15).

Base	`max_position_embeddings`
Llama-2 (7B / 13B / 70B)	4,096
Vicuna-v1.3 (7B / 13B)	2,048
LayerSkip-Llama-2 (7B / 13B)	2,048
LayerSkip-Llama-2 (70B)	4,096
LayerSkip-Llama-3 (8B)	8,192

Table 15. Released context windows for every model.

The cap on the assembled prompt (exemplars + chat wrapper + test input + trailing cue) is

$$\texttt{max\_input\_len} = \min(\texttt{max\_position\_embeddings}, 2048) - \texttt{max\_new\_tokens} - 32,$$

where the 32-token margin reserves headroom for SWIFT's draft tree. This gives an input cap of 1,504 tokens. The CNN/DailyMail article slot is then $\text{slot\_budget} = \texttt{max\_input\_len} - \text{exemplar\_tokens} - 5$, where the 5 tokens are headroom for re-tokenization drift when different parts are joined. For CNN/DailyMail this is $1{,}504 - 130 - 5 = 1{,}369$ tokens.

Articles longer than 1,369 tokens are right-truncated; the exemplar and trailing "`Summary:`" cue are preserved. HumanEval prompts (<300 tokens) and GSM8K prompts (≈360-token scaffold + short question) never trigger truncation.

Worked examples. Figure 24 shows the fully-assembled CNN/DailyMail base prompt. Figures 25 and 26 show the corresponding GSM8K and HumanEval prompts.

```
Article: (CNN) -- Zlatan Ibrahimovic scored all four goals in Sweden's 4-2 win
over England -- but his final shot was something special. His audacious overhead
volley from 30 yards was labeled on social networking sites as the greatest ever
soccer goal. What do you think? Share your views on Ibrahimovic's wonder goal.
Summary: Zlatan Ibrahimovic scores a 30-yard overhead kick. Ibrahimovic scored
all four goals for Sweden.
Article: ⟨test article, truncated to budget⟩
Summary:
```

Figure 24. CNN/DailyMail base prompt (Llama-2 / LayerSkip-Llama-2).

```
Question: Tom and Tim both brought 4, six-sided dice to school. How many total
sides are there?
Answer: There are 8 total dice because 4 plus 4 equals 8
There are 48 sides because 8 times six equals <<8*6=48>>48
#### 48
Question: What is fifteen more than a quarter of 48?
Answer: A quarter of 48 is 48/4=<<48/4=12>>12.
The number is 12+15=<<12+15=27>>27.
#### 27
(...three more shots: pill schedule, dozen-cookies, group-of-students ...)
Question: ⟨test question⟩
Answer:
```

Figure 25. GSM8K 5-shot chain-of-thought prompt.

```
from typing import List

def has_close_elements(numbers:  List[float], threshold:  float) -> bool:
  """ Check if in given list of numbers, are any two numbers closer to each other
than
  given threshold.
  >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
  False
  >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
  True
  """
```

Figure 26. HumanEval prompt.