# AdaSPEED: Adaptive Self-Speculative Early-Exit Decoding

Alice Liu
*Harvard University*
Cambridge, MA
aliceliu@c*

Valerio Pepe
*Harvard University*
Cambridge, MA
valeriopepe@c*

Julia Shephard
*Harvard University*
Cambridge, MA
jshephard@c*

Gabriel Wu
*Harvard University*
Cambridge, MA
gabrielwu@c*

***Abstract*—We propose *Adaptive Self-Speculative Early-Exit Decoding* (AdaSPEED), a method for speeding up transformer inference. Previous work on speculative decoding has shown that inference speeds can be increased by generating $D > 1$ tokens at a time with a subset of model layers, where $D$ is chosen to be constant ('Constant-D', [2]) or adaptively chosen to stop at the first token with decoding probability less than some threshold $\gamma$ ('Draft & Verify', [3]). In this work, we propose two improvements to this method: generating tokens based on the *product* of either (1) the token probabilities or (2) a linear probe's prediction of the token acceptance probabilities, and continuing until this product falls below $\gamma$. We test our methods on LayerSkip [1] and find that they speed up text generation by $17\%$ with greedy decoding, and $38\%$ with top-p decoding.**

***Index Terms*—Large Language Models, Speculative Decoding**

## I. Introduction

### A. Motivation

The success of transformer-based large language models has made them promising candidates for applications in latency-sensitive settings. However, transformer inference can be slow, as tokens must be decoded serially, and models are large. In standard transformer inference, decoding $k$ tokens requires $k$ consecutive runs of the full model, meaning a forward pass takes a significant amount of time.

Speculative and self-speculative decoding schemes tackle the problem of slow inference by ensuring that almost all serial generation is done by models smaller than the full model. Speculative decoding uses a small model to serially generate sequences of tokens, then verifies these sequences using the larger model. Although it speeds up inference, this technique has the drawback of requiring two models. Self-speculative decoding, in contrast, generates draft sequences of tokens using either a prefix or subset of the larger model's layers and verifies with the full model, requiring only one.

Two methods to speed up self-speculative decoding have been developed in the past year. LayerSkip, a variant of the LLaMa architecture that is trained for early-exiting of tokens, performs self-speculative decoding by generating $D > 1$ tokens and having them early-exit at a fixed layer $E$ [1]. Draft & Verify identifies an optimal subset of early layers to generate draft token sequences [3]. In contrast to LayerSkip, Draft & Verify generates tokens using these layers until its predicted probability that the next token it has generated will be accepted by the larger model falls below a certain threshold, $\gamma$. In both methods, the generated tokens of sequences are checked against the full model in parallel.

* Equal contribution. @c = @college.harvard.edu.

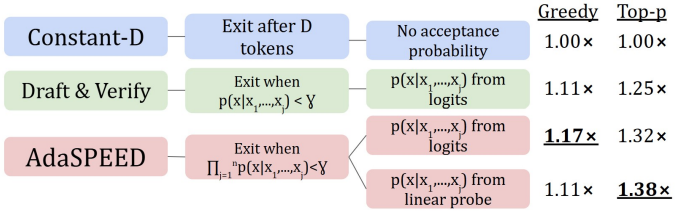


Fig. 1. The self-speculative decoding techniques used in this paper: AdaSPEED outperforms previous techniques in both decoding settings. The multipliers correspond to the average speedup across the 1B and 7B models.

### B. Our Work

While LayerSkip and Draft & Verify both provide substantial speedups over baseline autoregression, each has room for improvement. By always generating a constant number of tokens, LayerSkip preemptively cuts off long sequences of tokens that will likely be accepted by the larger model and wastes time extending short sequences of tokens that will likely not. Draft & Verify, on the other hand, cuts off token sequences solely on the basis of the next token's acceptance probability, rather than any measure of whether the full sequence will be accepted.

In this paper, we propose and implement two methods to speed up LayerSkip and self-speculative decoding more broadly. First, we implement a modification to the Draft & Verify acceptance schema. Instead of using the probability that the marginal token will be accepted, we generate tokens until the product that *all* token probabilities are accepted falls below a certain threshold $\gamma$. This approach, our "secret weapon," exploits the idea that we want to cut off our sequence once the model has likely made a mistake, even if each token individually has a high probability of acceptance, and the probability the model has made a mistake rises with each generation. Second, rather than using the probability of token acceptance as given by the model, we train a linear probe to predict whether a token generated by the model's early layers will be accepted by the full model.

We implement both of these methods on LayerSkip. We find that our approach, which we term "AdaSPEED", outperforms both standard LayerSkip and adaptive LayerSkip with Draft & Verify-inspired acceptance methods. We also find that adding the linear probe speeds up our model relative to other LayerSkip variants further when we use top-$p$ decoding. Our findings suggest that in self-speculative decoding, the joint probability of sequence acceptance may be more informative than the marginal probability. They also suggest that even probes trained on a small amount of data can provide a better proxy for token acceptance probability as it pertains to performance than using model logits. Our main contributions are additions to the theory underlying optimal token exiting as well as empirical results demonstrating a speedup induced by our method.

### C. Self-Speculative Decoding

Let $\mathcal{V}$ be a transformer model with $L$ layers, and let $x_1, \ldots, x_t$ be input tokens to the model. Self-speculative decoding is implemented in LayerSkip as follows. At each layer $\ell$, an LM head is fitted in order to output $p(x_{t+1}|\ell, x_1, \ldots, x_t)$, the probability distribution placed on possible tokens given that the algorithm early-exits at layer $\ell$. $p(x_{t+1}|L, x_1, \ldots, x_t)$ represents the logits of the full model. In the constant-$D$ algorithm, $D$ consecutive tokens are generated by assigning $x_{j+1} \leftarrow \arg\max p(x|\ell, x_1, \ldots, x_j)$. We describe the algorithm for Draft & Verify-inspired LayerSkip in Algorithm 1. This approach generates tokens until the predicted probability of the last token falls below an adaptive threshold $\gamma$. [1]

**Algorithm 1** Draft & Verify - Greedy Decoding

Initialize $i \leftarrow 0$; layer $\ell$ for early exiting; model $\mathcal{V}$; max tokens $T$ to generate
**while** $i < T$ **do**
  **for** $j \leftarrow i, \ldots, T$ **do**
    $x_{j+1} \leftarrow \arg\max p(x|\ell, x_1, \ldots, x_j)$
    **if** $p(x_{j+1}|\ell, x_1, \ldots, x_j) < \gamma$ **then**
      Break
  **for** $i \leftarrow i, \ldots, j$ **do**
    **if** $x_{i+1} \neq \arg\max p(x|L, x_1, \ldots, x_i)$ **then**
      $x_{i+1} \leftarrow \arg\max p(x|L, x_1, \ldots, x_i)$
      Break
  $i \leftarrow i + 1$
**return** $x_1, \ldots, x_T$

## II. Our Approach

### A. Product of acceptance probabilities

While the Draft & Verify approach uses the marginal probability, $p(x_{j+1}|\ell, x_1, \ldots, x_j)$, to decide whether to halt the drafting process, we instead choose to move to the verification stage when $\prod_{k=i}^{j} p(x_{k+1}|\ell, x_1, \ldots, x_k) < \gamma$. In considering this approach, it is important to remember that in the verification step, tokens are verified until one of them is rejected, at which point all subsequent tokens are rejected as well. Draft & Verify exploits this fact – if one token is likely to be rejected, subsequent tokens should not be added to the sequence. Implicitly, however, Draft & Verify disregards all previous tokens by choosing to move to the verification stage based on the marginal probability of token acceptance. By focusing instead on the joint probability – the product of the probabilities of token acceptance thus far, we ensure that we will stop drafting once it is likely that the draft sequence contains a mistake. This approach is shown in Algorithm 2.

**Algorithm 2** AdaSPEED with Greedy Decoding

Initialize $i \leftarrow 0$; exit layer $\ell$; model $\mathcal{V}$; max tokens $T$
**while** $i < T$ **do**
  **for** $j \leftarrow i, \ldots, T$ **do**
    $x_{j+1} \leftarrow \arg\max p(x|\ell, x_1, \ldots, x_j)$
    **if** $\prod_{k=i}^{j} p(x_{k+1}|\ell, x_1, \ldots, x_k) < \gamma$ **then**
      Break
  **for** $i \leftarrow i, \ldots, j$ **do**
    **if** $x_{i+1} \neq \arg\max p(x|L, x_1, \ldots, x_i)$ **then**
      $x_{i+1} \leftarrow \arg\max p(x|L, x_1, \ldots, x_i)$
      Break
  $i \leftarrow i + 1$
**return** $x_1, \ldots, x_T$

The motivation for using the product of probabilities instead of the marginal probability becomes clear in the following example: if the model is generating text in which the maximum token probability at layer $\ell$ always happens to be greater than $\gamma$, then the D&V algorithm will never break out of its speculation loop. However, after generating dozens of tokens, it is very likely that one of these tokens was a mistake. Considering the product of probabilities circumvents this issue, as the speculation loop is guaranteed to break eventually (as long as the probabilities are bounded away from 1).

### B. Linear probes

The standard Draft & Verify approach uses the probability predicted by the intermediate layers as the metric of acceptance. However, confidence at an early layer may not be an ideal proxy for the probability that the token is actually accepted. To more realistically estimate this value, we train a linear probe to predict the probability of a match between outputs at intermediate layer $\ell$ and outputs at the final (verification) layer $L$. The probe is a one-layer neural network that uses binary classification loss. With a batch size of $N$, for $n = 1, \ldots, N$, let $x_n$ be the speculative token predicted at layer $\ell$ and let $y_n$ the generated token at the final layer $L$, both at token position $n$. Let $h_n \in \mathbb{R}^d$ be the hidden state at layer $n$. Then our loss function for the linear probe (represented by a direction $v \in \mathbb{R}^d$ and bias $b \in \mathbb{R}$) is

$$\mathcal{L}_{\text{probe}} = \frac{1}{N} \sum_{n=1}^{N} \left( -\log \begin{cases} \sigma(v \cdot h_n + b) & y_n = x_n \\ 1 - \sigma(v \cdot h_n + b) & y_n \neq x_n \end{cases} \right).$$

[1] The algorithm to adaptively change $\gamma$, taken from Draft & Verify [3] is discussed in the Appendix.

TABLE I
CNN DAILY MAIL SUMMARIZATION RESULTS

| Methods | Greedy decoding (1B) | Top-p decoding (1B) | Greedy decoding (7B) | Top-p decoding (7B) |
|---|---|---|---|---|
| **LayerSkip** | 68tps (1.00x) | 43tps (1.00x) | 34tps (1.00x) | 29tps (1.00x) |
| **Draft & Verify** | 75tps (1.10x) | 58tps (1.35x) | 38tps (1.12x) | 33tps (1.14x) |
| **AdaSPEED (no probe)** | **79tps (1.16x)** | 61tps (1.42x) | **40tps (1.18x)** | 35tps (1.21x) |
| **AdaSPEED (probe)** | 75tps (1.10x) | **65tps (1.51x)** | 38tps (1.12x) | **36tps (1.24x)** |

In the experiments that follow, we train linear probes that are specific to each model and exit layer.

## III. IMPLEMENTATION

We implement our algorithm using Meta's LayerSkip library, using Meta's LayerSkip-Llama models with 1 billion and 7 billion parameters. We test our method on the CNN/Daily Mail Summarization dataset, which contains slightly over 300,000 articles and their summaries from both CNN and the Daily Mail. We select this dataset because it is used to test LayerSkip and heavily emphasized by their authors, as well as being a popular summarization dataset. Since self-speculative decoding requires that outputs are accepted by the full transformer model, ensuring lossless generation, we can measure the success of our model solely through its tokens/sec. This is a standard metric in self-speculative decoding research, used in both LayerSkip and Draft & Verify.

We train our probes for 500 batches of 4 CNN/DM summarization tasks each, yielding classification accuracies of $\sim 0.75$. Even with this accuracy (which does not improve if trained for longer), AdaSPEED with probes outperforms AdaSPEED without probes in terms of token acceptance rate, meaning the model is picking up meaningful signal.

### A. Results

Table I shows our results on the CNN/Daily Mail Summarization dataset. Regardless of sampling methodology and model size, AdaSPEED is faster than Draft & Verify. While these results are consistent across model sizes, AdaSPEED offers more of a speedup over baseline LayerSkip for the 1 billion parameter model. A potential explanation for this is that in a smaller model, each layer holds proportionally predictive power about the generated token, so the signal given by the joint probability of an earlier layer is more meaningful than it is in a bigger model. This suggests AdaSPEED could be particularly valuable for deployment in scenarios where smaller models are preferable.

Within each model, with greedy decoding our probe-less approach generates more tokens/sec than when we deploy AdaSPEED with the probes. In contrast, when we employ top-$p$ decoding, AdaSPEED with probes dominates AdaSPEED without probes. One explanation for the relative dominance of the probe method in top-$p$ decoding is that probes trained to require both the early exit and final layer to agree on the top $p$ tokens may have richer prediction signals than probes trained to have layers agree on the argmaxed token. Top-$p$ probes will thus be more robust predictors of acceptance after being trained than the probes we train for the greedy models.

In general, we observe that the optimal exit layer for AdaSPEED when implemented without probes is lower than the optimal exit layer for baseline LayerSkip, Draft & Verify, and AdaSPEED with probes (Layer 4 vs. 6). We anticipate that this may give AdaSPEED an edge in distributed computing.

## IV. RELATED WORK

Speculative decoding was introduced by Leviathan et al. as a technique to improve transformer inference speed [2], proposing the use of a separate model as the speculator. Elhoushi et al. (LayerSkip) build off this work by demonstrating that a single model can be trained such that a prefix of its layers serves as a good speculator [1], with the benefit that only the remaining $L - \ell$ model layers have to be run during the verification step. However, they still focus on a decoding algorithm in which the exit layer and number of speculated tokens are constant.

Finally, Zhang et. al (Draft & Verify) propose an adaptive speculative decoding algorithm in which tokens are speculatively generated until the marginal probability of the next token to be decoded falls below $\gamma$ [3], with an adaptive procedure for tuning $\gamma$ during generation.

## V. CONCLUSION

AdaSPEED outperforms both LayerSkip and Draft & Verify-implemented LayerSkip by exploiting the use of both linear probes and the joint probability distribution of a full token sequence being accepted. We anticipate that some insights developed in self-speculative decoding might speed the strategy up further. In particular, work to adaptively pick the exit layer during token generation, rather than in advance, might improve throughput. Further, while our product-based approach appears to work well, more complex criteria for leveraging the joint distribution and exiting the draft stage might work even better, e.g. penalizing tokens with a low acceptance probability earlier in the generation sequence. Future work could also focus on better probe training methods, or other types of sampling (e.g. top-k) and more datasets.

## VI. GROUP CONTRIBUTION STATEMENT

All team members contributed to the development of the method. GW, VP, and JS implemented the method and experiments. VP conducted the hyperparameter sweep for the optimal exit layer. JS, VP, and AL ran many of the experiments. JS led the writing of the report.

## References


[1] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. 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)*, page 12622–12642. Association for Computational Linguistics, 2024.

[2] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023.

[3] 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, 2024.


## VII. Appendix

### A. Adaptive Acceptance Probability Threshhold

As in Draft & Verify, we use an acceptance probability $\gamma$, which we update using the adaptive rule defined in [3]:

$$AR \leftarrow \beta_1 AR + (1-\beta_1) AR_e,$$

$$\tilde{\gamma} = \begin{cases} \gamma + \epsilon \text{ if } AR \leq \alpha \\ \gamma - \epsilon, \text{ otherwise} \end{cases}$$

$$\gamma \leftarrow \beta_2 \gamma + (1-\beta_2)\tilde{\gamma}.$$

We implement this in our Draft & Verify method, as well as in AdaSPEED. Constant-$D$ LayerSkip does not require $\gamma$-updating by definition. Values of $\beta_1, \beta_2, \epsilon$ are taken from Draft & Verify, which has them optimized. Conducting a hyperparameter search, we choose to initialize $\gamma = 0.8$