SEED: Self-Speculative Decoding via Implicit Encoder–Decoder

Hankun Lin[*], **Patrick Pynadath**[*], **Ruqi Zhang**
Department of Computer Science, Purdue University, USA
{lhankun, ppynadat, ruqiz}@purdue.edu

Abstract

Self-speculative decoding accelerates large language model (LLM) inference by drafting tokens from the target model itself, but faces a sharp tradeoff between the quality and cost of the draft. Early-exit methods produce drafts cheaply by terminating computation at intermediate layers, but forgo the deeper representations that later layers provide and thus suffer in draft quality. Multi-token prediction preserves draft quality by emitting from the model's final hidden states, but pays for a full forward pass to produce those states at every drafting step. We propose ***s**elf-sp**e**culative **e**ncoder-**d**ecoder* (**SEED**), a self-speculative method that obtains high-quality drafts cheaply by reusing the deep contextual representations already computed during verification. We reinterpret the standard decoder-only transformer as an implicit encoder–decoder: the first layers (encoder) build deep contextual representations, and the last few layers (decoder) emit tokens from them. Encoding and verification are merged into a single step: verification is performed by the full encoder–decoder, and the contextual representations of the verified prefix are cached for reuse during drafting. Drafting is therefore very fast: between verifications, the lightweight decoder drafts multiple tokens autoregressively, each conditioned on the cached representations and on preceding drafts. Experiments across multiple benchmarks show that SEED achieves up to 2.7× average speedup on 4B-scale models, outperforming both early-exit and MTP-style self-speculative baselines and running 28% faster than the state-of-the-art EAGLE-3, while preserving or even improving the generation quality of standard autoregressive fine-tuning. Code is available at https://github.com/lhk2004/SEED.

1 Introduction

Large language model (LLM) inference is memory-bound rather than compute-bound [31, 13]: generating each token requires a full forward pass through the model, yet that pass produces only a single output, leaving the GPU's parallel compute capacity underutilized. Speculative decoding [19, 7] amortizes this cost by using a cheap draft model to propose multiple candidate tokens that are verified in parallel by the target model in a single forward pass. However, maintaining a separate, well-aligned drafter complicates deployment. This has motivated self-speculative decoding, where drafting and verification are handled within a single unified model.

Self-speculative methods face a sharp tradeoff between the quality and cost of the draft. Early-exit or layer-skipping methods [39, 12, 35, 9] terminate the forward pass at an intermediate layer, making drafting cheap but forcing the drafter to predict from shallow representations, which limits draft quality. By contrast, multi-token prediction (MTP) methods [14, 33] emit drafts from the model's final hidden states, preserving representational depth but requiring a full forward pass at every drafting step. This raises a question: *can drafts be as deep as the full model yet as cheap as a few layers?*

[*]Equal contribution.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

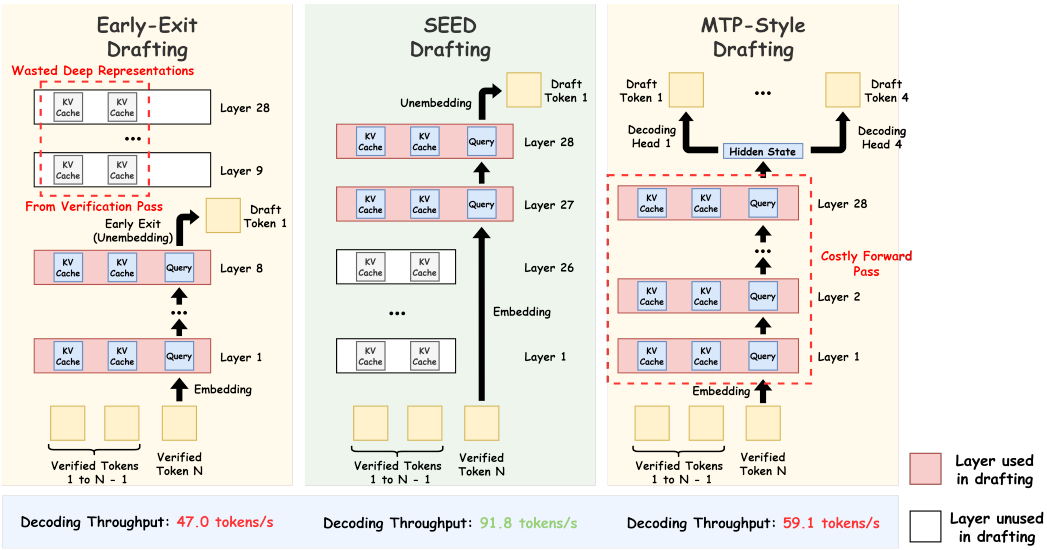

Figure 1: **Overview of SEED.** SEED partitions a 28-layer model into a 26-layer encoder and a 2-layer decoder. During drafting, the decoder generates drafts by cross-attending to the deep contextual KV cache produced during the previous verification step, avoiding repeatedly invoking the costly encoder. This design contrasts with two existing self-speculative decoding paradigms: early-exit drafting (instantiated using LayerSkip [12]) discards the deep representations from later layers, sacrificing draft quality, while multi-token prediction (instantiated using Apple MTP [33]) preserves draft quality by maintaining full representational depth but still requires a full forward pass at every drafting step. On GSM8K with draft length 4, SEED achieves substantially higher decoding throughput than early-exit drafting ($\sim 2.0\times$) and MTP ($\sim 1.6\times$).

Our answer adapts the encoder-decoder amortization principle developed in diffusion language models [3] to autoregressive speculative decoding. We view the standard decoder-only transformer as an implicit encoder-decoder: the first layers act as an encoder that builds deep contextual representations of the prefix, while the last few layers act as a decoder that autoregressively emits the next token from those representations. The crucial observation is that under this view, verification already does all the encoding work needed for subsequent drafting — the deep representations the drafter needs are already in the KV cache. Building on this, we propose ***self-speculative encoder-decoder*** (**SEED**), a self-speculative decoding method that trains the lightweight decoder to generate new draft tokens by conditioning on the verifier's deepest features, without invoking the encoder again until the next verification step. Each draft token therefore costs only a thin-decoder forward pass yet conditions on the verifier's full-depth representation of the prefix. Unlike prior methods that require careful architectural modifications or nuanced training procedures, SEED is remarkably simple: it adds only a single auxiliary loss on top of standard supervised fine-tuning. Figure 1 provides an overview of SEED.

Across math, coding, general knowledge, and summarization benchmarks, SEED achieves up to a **2.7$\times$ average speedup on 4B-scale models** while matching or improving the generation quality of standard autoregressive fine-tuning. It consistently outperforms both early-exit and MTP-based self-speculative baselines and runs **28% faster than the state-of-the-art EAGLE-3**. Moreover, our analysis reveals that, like MTP methods, SEED encourages the model to encode more predictive information about future tokens, which contributes to improving both draft quality and downstream task performance.

2 Related Work

There has been substantial interest in accelerating large language model (LLM) inference. While classical speculative decoding requires a small auxiliary model or multiple heads [34, 5, 2, 22], most relevant to our work are self-speculative decoding methods, which avoid using an additional draft

model by drafting from the target model itself. These methods fall broadly into two categories, multi-token prediction (MTP) and exiting early by skipping layers.

Multi-Token Prediction. Multi-token prediction (MTP) accelerates inference by predicting multiple tokens in a single forward pass, increasing the model's output per pass while keeping per-pass cost roughly fixed. Originally motivated by denser training signals, early work modified the pretraining objective to enable parallel token prediction [32, 14, 23], but reliance on pretraining from scratch makes these approaches impractical for accelerating deployed models. More recent work adapts the MTP pipeline for faster inference on existing models [4, 33, 28, 26, 6, 11, 16, 40]. A separate line replaces the standard MTP formulation with alternative decoding schemes such as Jacobi decoding [18, 15] or diffusion-based generation [3, 25, 8], but these either sacrifice quality by breaking causal dependencies or substantially complicate the training objective.

Early-Exit Methods. Early-exit methods reuse components of the verifier itself as the drafter, typically skipping later layers and using only the initial layers for drafting. This can be achieved through additional training [12, 24] or as a purely inference-time decision [39, 35, 9, 38]. SEED shares the layer-skipping spirit but inverts the design: rather than skipping later layers, we skip the initial layers and draft from only the final layers of the base model. This asymmetric choice lets the drafter condition on full-depth contextual representations, producing higher-quality draft tokens than early-exit alternatives.

For a more detailed comparison with existing LLM acceleration methods, see Appendix B.

3 Preliminaries

We represent a standard transformer as a function f that maps an input sequence $X = (X_1, X_2, \ldots, X_n)$ of tokens from a vocabulary $\mathcal{V}$ to a probability distribution over the next token, $f(X) \in \triangle(\mathcal{V})$. The transformer is composed of l layers $(f_1, f_2, \ldots, f_l)$. Each layer f_i takes as input the hidden states from the previous layer, $H^{i-1} = (H_1^{i-1}, H_2^{i-1}, \ldots, H_n^{i-1})$, and outputs updated representations $H^i = (H_1^i, H_2^i, \ldots, H_n^i)$. Each self-attention layer additionally computes keys and values that are cached to avoid redundant computation; we denote the KV cache available before processing token X_t at layer i as $KV_{<t}^i$. For brevity, we omit explicit notation for KV cache updates in the method description that follow.

The first layer f_1 operates on the token embeddings $H^0 = W_{\text{in}}X$, where W_{in} is the embedding matrix. The output of the final layer, H^l, is projected back to vocabulary space via a second embedding matrix W_{out} to obtain the next-token distribution,

$$P(\cdot \mid X_{\leq n}) = \text{softmax}(H_n^l W_{\text{out}}^\top). \tag{1}$$

Autoregressive transformer language models are trained by minimizing the standard next-token cross-entropy loss

$$\mathcal{L}_{\text{CE}} = \mathbb{E}_{X\sim\mathcal{D}}\left[\sum_{j=1}^{|X|} -\log f(X_j \mid X_{<j})\right], \tag{2}$$

where $\mathcal{D}$ denotes the training data distribution.

4 Self-Speculative Encoder-Decoder

We propose **s**elf-sp**e**culative **e**ncoder-**d**ecoder (**SEED**), a self-speculative decoding method that obtains high-quality drafts cheaply while keeping the model architecture unchanged. We first introduce the encoder-decoder reinterpretation of the standard decoder-only transformer, and explain why it enables cheap yet accurate drafting. We then present the training algorithm, which augments the standard next-token prediction objective with an additional speculative drafting objective. This trains the decoder to draft from raw token embeddings while cross-attending to cached deep representations. Finally, we describe the inference procedure, including a fixed-length drafting and a dynamic drafting with tree-based verification.

4.1 Method

A key observation underlying SEED is that a standard decoder-only transformer can be naturally interpreted as an implicit encoder-decoder architecture. This reinterpretation exposes a structural asymmetry that we exploit to make cheap yet accurate drafting.

Decoder-Only Transformers as Implicit Encoder-Decoders. Consider a transformer with l layers $(f_1, \ldots, f_l)$. We partition the model at some layer l', treating the first l' layers as an *encoder* that builds deep contextual representations of the prefix, and the remaining layers as a *decoder* that emits the next token from those representations. The encoder typically accounts for the majority of layers (e.g., $l' = 26$ out of $l = 28$ in our experiments).

Formally, we define $f_{enc} = f_{1:l'}$ and $f_{dec} = f_{l'+1:l}$, with $KV^{enc}_{<n}$ and $KV^{dec}_{<n}$ denoting the cached keys and values from all positions preceding n in the encoder and decoder respectively. After processing token X_n, the corresponding caches are updated to $KV^{enc}_{\leq n}$ and $KV^{dec}_{\leq n}$. For a context sequence $X_{\leq n}$, the encoder produces a deep contextual embedding of the current input token X_n:

$$H^{l'}_n = f_{enc}(\cdot \mid X_n, KV^{enc}_{<n}), \tag{3}$$

and the decoder predicts the next token from this representation while attending to the cached context:

$$X_{n+1} \sim \mathrm{softmax}\Big(f_{dec}(\cdot \mid H^{l'}_n, KV^{dec}_{<n})\Big). \tag{4}$$

This decomposition is purely conceptual as it is exactly equivalent to standard autoregressive decoding, but it makes a structural asymmetry explicit: the encoder accounts for most per-token computation while the decoder is a thin stack of only a few layers.

Standard autoregressive generation does not exploit this asymmetry, since the encoder is re-invoked for every new token. We propose to exploit it for self-speculative decoding: because context evolves slowly, encoding can be performed infrequently, and the decoder alone can emit tokens conditioned on a fixed contextual representation.

Verification Already Does the Encoder's Work for Drafting. Remarkably, we do not need to run the encoder separately to obtain the contextual representation. The verification step already does this work.

To verify a batch of proposed drafts, the full encoder-decoder runs on these drafts and produces $KV^{dec}_{<n}$, the deep contextual representations up to the last verified draft token X_{n-1}. Because the verifier has processed the entire prefix $X_{<n}$ during this forward pass, it should be able to predict another bonus token X_n through the same pass. Thus, the cache contains deep contextual representations only for tokens preceding X_n, which are precisely the representations needed for subsequent drafting. Verification and encoding therefore merge into a single step: one forward pass of the full model simultaneously verifies the drafts and prepares the deep contextual cache that the drafter consumes in the next round.

Cheap Drafting Between Verifications. With the deep representations already cached during verification, the decoder no longer requires a fresh encoder call to begin drafting. We instead feed the raw token embedding $H^0_n = W_{in}X_n$ of the latest token directly into the decoder, which conditions on the cached deep representations through cross-attention to predict the next token

$$X_{n+1} \sim \mathrm{softmax}\big(f_{dec}(\cdot \mid H^0_n, KV^{dec}_{<n})\big). \tag{5}$$

Between verifications, drafting requires only the lightweight decoder. The encoder is invoked again only when the next batch of drafts is verified, at which point it produces both verification and the refreshed cache.

Note that we use the term *cross-attention* as shorthand for attention whose queries come from raw token embeddings and whose keys and values come from the encoder-produced cache. This is implemented as standard self-attention over the KV cache and does not denote a separate encoder–decoder cross-attention module. We adopt the term "cross-attention" only to emphasize that in decoder, the cache carries full-depth, encoder-processed representations while the incoming query carries only a shallow raw embedding, playing a role analogous to conventional encoder–decoder cross-attention where decoder queries attend to encoder-derived keys and values.

4.2 Training Algorithm

We now describe how to train the decoder to draft accurately from raw embeddings while cross-attending to cached deep representations. We focus on the typical scenario in which SEED is integrated into a standard supervised fine-tuning pipeline: starting from a pre-trained base LLM, we fine-tune it on task data to improve task performance while simultaneously enabling self-speculative decoding. The same procedure also applies when starting from an already fine-tuned model and continuing training purely to enable inference acceleration, which we explore in Appendix E.2.

Standard Autoregressive Objective. To preserve the model's generation quality, we retain the standard next-token prediction objective, re-expressed under our encoder-decoder decomposition:

$$\mathcal{L}_{CE} = \mathbb{E}_{X\sim\mathcal{D}}\left[\sum_{j=1}^{|X|} -\log f_{dec}\left(X_j \mid H^{l'}_{j-1}, KV^{dec}_{<(j-1)}\right)\right]. \tag{6}$$

This loss exactly recovers standard autoregressive training, ensuring that the full encoder-decoder (the verifier) behaves identically to a standard autoregressive model.

Speculative Drafting Objective. At inference, the drafter must predict the next token conditioned on (i) deep contextual representations $KV^{dec}_{<n'}$ produced by the verifier for tokens up to the last verified position n', and (ii) only the raw token embeddings $H^0_{n'\leq\cdot\leq(j-1)} = W_{in}X_{n'\leq\cdot\leq(j-1)}$ for the last verified context token and any subsequent draft tokens, since invoking the encoder on draft tokens would defeat the purpose of cheap drafting. To train the decoder to predict the next token given this input, we introduce a speculative drafting objective:

$$\mathcal{L}_{spec} = \mathbb{E}_{X\sim\mathcal{D}}\left[\sum_{j=1}^{|X|} -\log f_{dec}\left(X_j \mid H^0_{n'\leq\cdot\leq(j-1)}, KV^{dec}_{<n'}\right)\right]. \tag{7}$$

Training with Block Attention. Optimizing $\mathcal{L}_{spec}$ requires choosing a boundary n' for each target token X_j that splits the conditioning context into cached deep representations $KV^{dec}_{<n'}$ and raw token embeddings $H^0_{n'\leq\cdot\leq(j-1)}$. This boundary directly mirrors the inference-time situation we want the decoder to handle: between two verification steps, the decoder must draft several new tokens on its own using only their raw embeddings, while cross-attending to the deep KV cache produced during the previous verification, and only after the next verification do these draft positions get committed to deep contextual representations. To expose the decoder to exactly this conditioning pattern during training, we partition each training sequence into contiguous, non-overlapping blocks of b tokens, where each block plays the role of a sequence of drafts that the decoder must produce alone between two verifications. For every target token X_j, we set n' to the index of the first token in the block containing X_{j-1} (i.e., $n' = \lfloor (j-2)/b \rfloor \times b + 1$). This means that within each block, the decoder sees only raw token embeddings for the current block's tokens (via causal self-attention) and cross-attends to the deep KV cache $KV^{dec}_{<n'}$ produced by all preceding blocks. Because n' is the same for every token within a block, all blocks can be processed in parallel.

Concretely, training proceeds in two stages within each gradient step. First, the full encoder–decoder processes the entire sequence in a single forward pass, computing both the standard autoregressive loss $\mathcal{L}_{CE}$ and caching the decoder's KV representations $KV^{dec}_{1:n}$ at every position. Second, the decoder alone processes all blocks in parallel using the block-attention mask shown in Figure 2: within each block, tokens attend causally to other tokens in the same block via self-attention over raw embeddings H^0, and additionally cross-attend to KV^{dec} entries from all preceding blocks (but not the current block). We adapt this block-wise training strategy from [3], but use *block-causal* masks throughout rather than the block-bidirectional masks: response tokens in the encoder-decoder attend causally, preserving the verifier as a valid autoregressive model, and tokens within each decoder block attend causally so the drafter can generate tokens autoregressively. This preserves the autoregressive structure on both sides of the encoder-decoder split, which their block-diffusion formulation does not require.

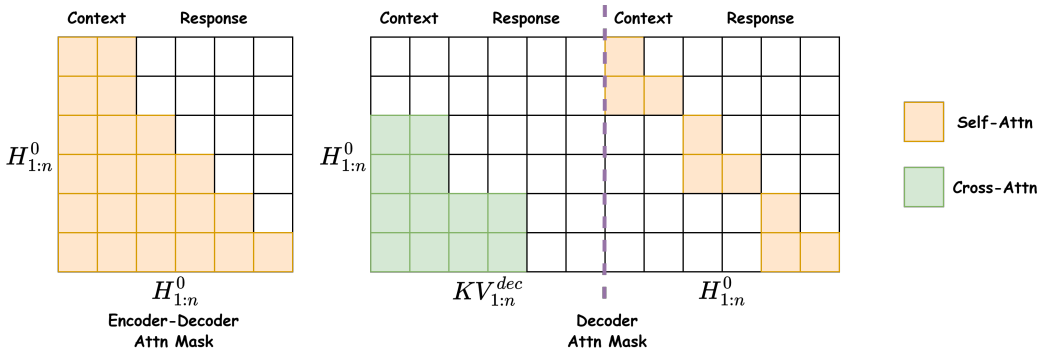

Figure 2: Example training-time attention masks for $n = 6$ tokens with block size $b = 2$. **Left:** In the joint encoder-decoder verifier, context tokens use bidirectional attention for richer encoding, while response tokens use causal attention for autoregressive drafting. **Right:** In the decoder, input token embeddings ($H^0_{1:n}$) attend causally to tokens within the same block via self-attention, and additionally cross-attend to the decoder's own KV cache ($KV^{dec}_{1:n}$) of all preceding blocks.

Final Training Objective. Our final training objective combines the standard autoregressive loss with the speculative drafting loss through a weighted average:

$$\mathcal{L} = \frac{\mathcal{L}_{CE} + \lambda \mathcal{L}_{spec}}{1 + \lambda}, \tag{8}$$

where λ controls the relative weight of the speculative drafting objective. In our experiments, we set $\lambda = 1.0$. Section 5.4 studies the impact of varying λ, and the complete training procedure is summarized in Algorithm 1.

4.3 Inference Algorithm

Fixed-Length Drafting. At inference time, the trained decoder f_{dec} acts as a lightweight draft model, while the full encoder–decoder $f_{dec} \circ f_{enc}$ serves as the verifier. Decoding alternates between drafting and verification: starting from a verified prefix, f_{dec} autoregressively generates d draft tokens by cross-attending to the deep contextual representations (KV^{dec}) cached from the previous verification step, after which the full encoder-decoder verifies these tokens in a single forward pass. We summarize the complete inference procedure in Algorithm 2.

Adaptive Drafting with Tree-Based Verification. Fixed-length drafting is sufficient to realize the core benefit of SEED, but it leaves additional speedup on the table because the decoder's confidence varies across generation steps: drafting too few tokens under-utilizes a confident decoder, while drafting too many wastes computation when the decoder is uncertain. To better exploit cheap drafting, we adopt two complementary techniques from prior work. First, following [24, 35, 38], we replace the fixed draft length with a confidence-based stopping rule, allowing the decoder to draft more tokens when confident and fewer when uncertain. Second, following [35] and [30], we expand the draft into multiple sibling candidates via a top-p rule rather than committing to a single token. Across drafting steps, this yields a tree of candidate tokens that the full model verifies in one forward pass using a tree-structured attention mask preserving the correct causal dependencies.

Both techniques are particularly well-suited to SEED. Because the decoder shares parameters with the verifier, its confidence reliably reflects acceptance rate, making the stopping rule effective. And because each drafting step is cheap (only a lightweight-decoder forward pass), the overhead of producing extra candidates is small relative to the gain in accepted tokens per verification. We use this adaptive tree-based drafting strategy in the main experiments unless otherwise specified. We defer further illustration and implementation details to Appendix D.1, with ablations in Appendix D.2.

Table 1: **Performance comparison on `Qwen3-1.7B-Base` and `Qwen3-4B-Base`.** 0-shot pass@1 accuracy (Acc.) is reported in %, and decoding throughput (Tput) in tokens/s. For CNN/Daily Mail, we report ROUGE-1 (R-1), ROUGE-2 (R-2), and ROUGE-L (R-L). For methods that keep the target model frozen, we report only their throughput. Given the size of its released drafter weights, PARD is evaluated only on `Qwen3-4B-Base`. The highest throughput and average speedup are shown in **bold**, and the second highest results are underlined.

Method	GSM8K		KodCode		ScienceQA		CNN/Daily Mail				Avg.
	Acc.	Tput	Acc.	Tput	Acc.	Tput	R-1	R-2	R-L	Tput	Speedup
`Qwen3-1.7B-Base`											
AR	56.3	37.9	66.5	36.0	93.2	36.6	38.6	17.7	28.6	34.2	1.0 ×
Common Acceleration Methods											
E2D2	26.6$^{\downarrow 29.7}$	67.6	40.6$^{\downarrow 25.9}$	65.1	88.7$^{\downarrow 4.5}$	68.1	32.8$^{\downarrow 5.8}$	12.1$^{\downarrow 5.6}$	23.3$^{\downarrow 5.3}$	56.8	1.8 ×
Apple MTP	-	59.1	-	54.6	-	56.6	-	-	-	37.5	1.4 ×
EAGLE-3	-	79.4	-	70.4	-	75.4	-	-	-	60.7	2.0 ×
Early-Exit Self-Speculative Methods											
LayerSkip	42.2$^{\downarrow 14.1}$	47.0	57.5$^{\downarrow 9.0}$	57.0	91.7$^{\downarrow 1.5}$	60.0	37.8$^{\downarrow 0.8}$	16.9$^{\downarrow 0.8}$	27.6$^{\downarrow 1.0}$	42.9	1.4 ×
SWIFT	-	43.0	-	41.2	-	40.2	-	-	-	36.2	1.1 ×
DEL	-	50.1	-	60.0	-	62.2	-	-	-	38.6	1.5 ×
SEED (Ours)	57.1$^{\uparrow 0.8}$	**91.8**	68.8$^{\uparrow 2.3}$	**105.3**	95.5$^{\uparrow 2.3}$	**111.7**	39.2$^{\uparrow 0.6}$	18.0$^{\uparrow 0.3}$	29.1$^{\uparrow 0.5}$	**63.2**	**2.6**×
`Qwen3-4B-Base`											
AR	75.1	28.3	76.2	27.6	95.4	26.0	39.0	18.2	29.1	24.0	1.0 ×
Common Acceleration Methods											
E2D2	35.5$^{\downarrow 39.6}$	55.8	48.0$^{\downarrow 28.2}$	53.5	93.5$^{\downarrow 1.9}$	54.6	32.7$^{\downarrow 6.3}$	11.5$^{\downarrow 6.7}$	23.5$^{\downarrow 5.6}$	40.3	1.9 ×
Apple MTP	-	45.1	-	49.7	-	41.3	-	-	-	29.4	1.6 ×
EAGLE-3	-	64.4	-	54.1	-	56.0	-	-	-	**49.9**	2.1 ×
PARD	-	57.7	-	65.4	-	62.7	-	-	-	41.0	2.1 ×
Early-Exit Self-Speculative Methods											
LayerSkip	62.2$^{\downarrow 12.9}$	34.2	71.1$^{\downarrow 5.1}$	43.7	92.4$^{\downarrow 3.0}$	39.8	38.7$^{\downarrow 0.3}$	17.6$^{\downarrow 0.6}$	28.5$^{\downarrow 0.6}$	28.7	1.4 ×
SWIFT	-	31.7	-	31.3	-	25.8	-	-	-	26.1	1.1 ×
DEL	-	34.8	-	41.0	-	39.4	-	-	-	22.1	1.3 ×
SEED (Ours)	76.0$^{\uparrow 0.9}$	**74.9**	81.0$^{\uparrow 4.8}$	**80.5**	96.8$^{\uparrow 1.4}$	**89.3**	39.8$^{\uparrow 0.8}$	18.4$^{\uparrow 0.2}$	29.3$^{\uparrow 0.2}$	43.3	**2.7**×

5 Experiments

5.1 Experimental Setup

Datasets & Metrics. We demonstrate the effectiveness of SEED by training task-specific models across a diverse set of benchmarks, covering mathematical reasoning, code generation, general knowledge question answering, and summarization: GSM8K [10], KodCode [36], ScienceQA [27], and CNN/Daily Mail [29]. For efficiency, we report the decoding throughput achieved by each method on every dataset with a single 48 GB NVIDIA A6000 GPU. For task quality, we report 0-shot pass@1 accuracy on GSM8K, KodCode, and ScienceQA, and ROUGE scores for CNN/Daily Mail. More details regarding dataset construction can be found in Appendix C.2.

Baselines. We compare our method against the standard autoregressive (AR) decoding baseline with SFT. We further include three groups of accelerated decoding approaches: (1) **Early-exit self-speculative decoding** methods, including LayerSkip [12], SWIFT [35], and DEL [38]; (2) **Speculative decoding** with a separate draft model, where we adopt the state-of-the-art EAGLE-3 [22] and PARD [1] (4B target model only); (3) **Diffusion language models** and **multi-token prediction (MTP)** methods, including E2D2 [3], which adopts an encoder–decoder division similar to ours to freeze context understanding but applies it within a blocked diffusion framework (Appendix B.3 further clarifies our novelty over E2D2), and Apple MTP [33], a strong MTP baseline that mitigates the loss of local dependency modeling in standard MTP. All methods use greedy decoding for generation.

Models. For methods that require training (AR, E2D2, LayerSkip, Apple MTP, and our method), we fine-tune the pre-trained `Qwen3-1.7B-Base` and `Qwen3-4B-Base` models [37] using the training

split of each task dataset and evaluate on the corresponding test split. SEED uses only the last 2 transformer layers of the model as the decoder and is trained with a block size of 10, i.e., each sequence is partitioned into 10-token blocks when computing the speculative drafting objective. We perform ablations on these choices in Section 5.4 and provide setup details for other methods in Appendix C.3.

5.2 Results

We present the main evaluation results in Table 1. Overall, SEED delivers substantial inference acceleration. Across all benchmarks, SEED achieves an average decoding speedup of **2.6**× on `Qwen3-1.7B-Base` and **2.7**× on `Qwen3-4B-Base`, outperforming all self-speculative decoding baselines. It also matches or surpasses the state-of-the-art EAGLE-3, despite EAGLE-3 relying on a separately trained drafter network. Beyond faster decoding, SEED improves task performance over standard autoregressive (AR) fine-tuning, whereas training-based acceleration methods such as E2D2 and LayerSkip suffer severe quality degradation.

5.3 Analysis

Here we explain why SEED outperforms early-exit and MTP-based self-speculative decoding methods as well as speculative decoding methods with a separate drafter like EAGLE-3. The advantages stem from both encoder and decoder: the encoder produces deeper, more future-predictive contextual representations, while the decoder, which shares parameters with the verifier, drafts in a way that remains closely aligned with the verifier's generation distribution. All experiments in this analysis, as well as the following ablation studies, are conducted by evaluating `Qwen3-1.7B-Base` models fine-tuned on GSM8K with the same setup as in our main experiments.

Better Drafting from Deeper Representations and Parameter Sharing with the Verifier. We first show that conditioning on deep representations enables SEED to outperform early-exit methods. As shown in Figure 3, SEED consistently achieves higher acceptance rates than LayerSkip despite using a much smaller 2-layer drafter. Early-exit methods like LayerSkip terminate computation at intermediate layers and thus discard the deeper representations from later layers that are crucial for accurate token prediction. SEED avoids this limitation by drafting from the very last layers of the model, giving the drafter access to the verifier's full-depth contextual features. Moreover, SEED outperforms EAGLE-3 on draft acceptance even though EAGLE-3 also produces drafts conditioned

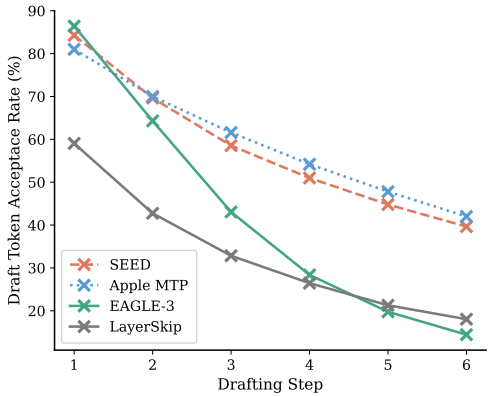

Figure 3: Draft token acceptance rates across drafting steps. For a fair comparison, all methods are evaluated under a fixed draft length of 6, with only one candidate draft token kept at each step. Compared to LayerSkip and EAGLE-3, SEED produces higher-quality drafts, both from deeper representations and from parameter sharing with the verifier. Compared to MTP, SEED achieves comparable draft quality at much lower cost.

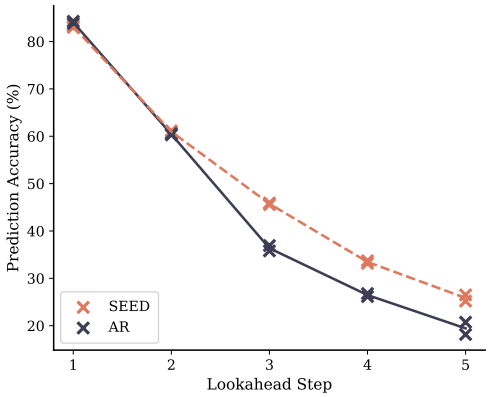

Figure 4: Prediction accuracy of trained linear probes in relation to lookahead step for two runs with different seeds. Compared to standard AR training, SEED forces the encoder to produce representations that are more predictive of distant future tokens.

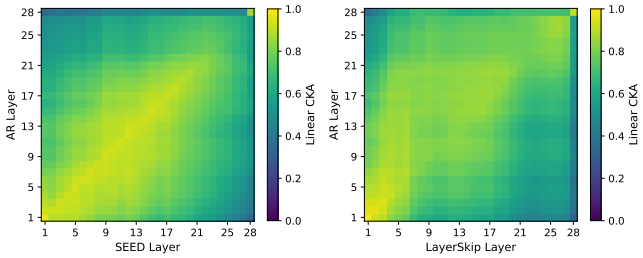

Figure 5: Cross-layer linear CKA matrices on GSM8K. **Left:** AR vs SEED. **Right:** AR vs LayerSkip.

Table 2: Ablation on the choice of λ. l_{draft} and r_{draft} denote the average draft token acceptance length and acceptance rate respectively.

λ	**Acc.**	**Tput**	l_{draft} (r_{draft})
0.1	58.5	81.5	3.0 (87.4%)
0.5	59.1	87.6	3.7 (88.3%)
1.0	57.1	91.8	4.0 (88.6%)
1.5	55.5	92.1	4.0 (88.0%)
2.0	55.1	91.4	4.0 (88.4%)

on deep representations. We attribute this gain to parameter sharing: SEED's drafter is the verifier's own last two layers, so each draft token is sampled from a distribution closely matching the verifier's. In contrast, EAGLE-3 relies on a separate draft model, whose output distribution can still deviate from the verifier's despite being trained to approximate it.

Cheaper Drafting with A Lightweight Decoder. Figure 3 shows that SEED and Apple MTP achieve comparable draft acceptance, since both condition drafting on deep contextual representations. The throughput gap therefore comes from drafting cost: MTP requires a full forward pass at every drafting step to produce the final hidden states, while SEED reuses the deep contextual representations cached during verification, requiring only a 2-layer decoder pass per draft token. Appendix F quantifies the dominant computation and gives an approximate latency model for fixed-length drafting.

Improved Task Performance from Planning Ahead. We next explain the task performance gains of SEED over standard AR fine-tuning. We hypothesize that our speculative drafting objective implicitly trains the encoder to produce more future-predictive contextual representations: because the decoder must accurately predict tokens many steps ahead from these representations, the encoder is pushed to encode information about future tokens. This is similar to the effect observed in MTP methods [14, 23]. Our linear-probe experiment in Figure 4 directly tests this hypothesis: SEED's encoder produces hidden representations that remain more predictive of distant future tokens than those of a standard AR model, consistent with a "planning-ahead" effect induced by the speculative drafting objective. Additional details on the experimental design are provided in Appendix E.1.

What SEED's Learned Representations Look Like. Finally, we examine the layer-wise structure of the representations learned by SEED using Centered Kernel Alignment (CKA) [17]. With the hidden states collected from the linear-probe experiment, we compute cross-layer CKA matrices between the AR fine-tuned model and both the SEED model and the LayerSkip model. Figure 5 shows that SEED's representations remain very similar to the AR model's across the early and middle layers, indicating that the training dynamics of the encoder are largely unchanged relative to standard AR fine-tuning. The strongest divergence appears near the encoder-decoder interface (layer $l' = 26$), where SEED reshapes representations most substantially, likely by encouraging the encoder to output more future-predictive hidden states and by adapting the decoder to operate on a novel mixture of raw token embeddings and cached deep representations.

Together, these patterns suggest that SEED's training acts as a *localized* modification of the standard AR fine-tuning, targeting the layers responsible for drafting while preserving the deep contextual processing already performed by the bulk of the model. By contrast, LayerSkip alters the model's internal representations more substantially and shows broader representational drift across layers, explaining its degradation in downstream task performance.

5.4 Ablation Study

Figure 6 studies the effect of **decoder size**, where we keep the total number of transformer layers fixed and vary the encoder-decoder split (i.e., a larger decoder corresponds to a smaller encoder, and vice versa). Increasing the number of decoder layers improves draft quality, but it also increases the cost of each drafting step and reduces overall decoding throughput. For a better trade-off, we use only the last 2 layers as the decoder in our main experiments. Figure 7 studies the effect of the **training block size** used in the speculative drafting objective $\mathcal{L}_{spec}$. Increasing the block size generally improves acceptance length and decoding throughput, suggesting that training the decoder

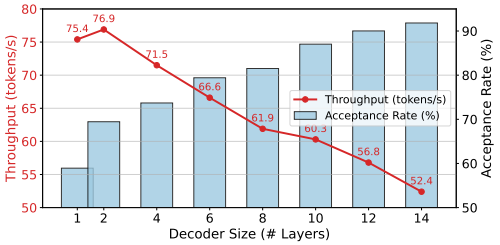

Figure 6: Ablation on decoder size. Models are evaluated with a simple fixed draft length of 4, so that the draft token acceptance rate directly reflects draft quality and isolates how decoder size influences drafting accuracy. Larger decoders improve draft acceptance but reduce throughput due to higher drafting cost.

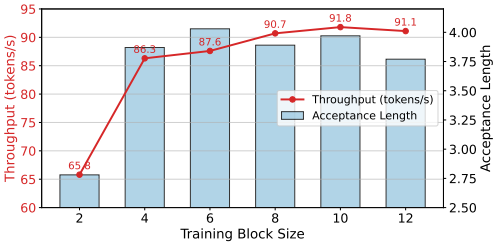

Figure 7: Ablation on training block size. Models are evaluated with dynamic drafting and tree-based verification. Larger block sizes improve draft acceptance and decoding throughput.

to operate over longer draft spans better prepares it for speculative generation at inference time. We use a block size of 10 in our main experiments as it provides the best decoding speedup. Table 2 studies the effect of the **loss weight** λ in Eq. (8). A larger λ places more weight on $\mathcal{L}_{spec}$, generally improving draft quality, while a smaller λ emphasizes the standard next-token prediction loss $\mathcal{L}_{CE}$, which favors task accuracy. We choose $\lambda = 1.0$ in our main experiments as it provides a favorable balance. Additional ablations are deferred to Appendix D.2.

6 Conclusion

SEED mitigates the central tradeoff in self-speculative decoding between draft quality and drafting cost by reframing decoder-only transformers as implicit encoder-decoder models. By reusing only the last layers as a lightweight drafter conditioned on cached deep representations of the verified prefix, SEED achieves cheap drafting without discarding the rich contextual features produced by the preceding encoder. Across diverse benchmarks, SEED consistently outperforms prior self-speculative baselines in decoding speed while preserving, and often improving, task performance relative to standard autoregressive fine-tuning. Our analysis further suggests that these gains arise because SEED reshapes the training dynamics near the encoder-decoder interface, likely by encouraging the model to learn more future-predictive internal representations.

Acknowledgement

This research is supported in part by NSF IIS-2508145, Amazon Research Award, and Lambda's Research Grant Program. We thank Ziteng Sun for his thoughtful comments on the manuscript.

References

[1] Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. Pard: Accelerating llm inference with low-cost parallel draft model adaptation. In *International Conference on Learning Representations*, volume 2026, pages 144562–144579, 2026.

[2] Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for medusa decoding. In *First Conference on Language Modeling*, 2024.

[3] Marianne Arriola, Yair Schiff, Hao Phung, Aaron Gokaslan, and Volodymyr Kuleshov. Encoder-decoder diffusion language models for efficient training and inference. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[4] Nikhil Bhendawade, Irina Belousova, Qichen Fu, Henry Mason, Mohammad Rastegari, and Mahyar Najibi. Speculative streaming: Fast llm inference without auxiliary models. *arXiv preprint arXiv:2402.11131*, 2024.

[5] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, pages 5209–5235, 2024.

[6] Yuxuan Cai, Xiaozhuan Liang, Xinghua Wang, Jin Ma, Haijin Liang, Jinwen Luo, Xinyu Zuo, Lisheng Duan, Yuyang Yin, and Xi Chen. Fastmtp: Accelerating llm inference with enhanced multi-token prediction. *arXiv preprint arXiv:2509.18362*, 2025.

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

[8] Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026.

[9] Longze Chen, Renke Shan, Huiming Wang, Lu Wang, Ziqiang Liu, Run Luo, Jiawei Wang, Hamid Alinejad-Rokny, and Min Yang. 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)*, pages 31608–31618, 2025.

[10] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[11] Felix Draxler, Justus Will, Farrin Marouf Sofian, Theofanis Karaletsos, Sameer Singh, and Stephan Mandt. Parallel token prediction for language models. *arXiv preprint arXiv:2512.21323*, 2025.

[12] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, 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)*, pages 12622–12642, 2024.

[13] Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, pages 14060–14079, 2024.

[14] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, pages 15706–15734, 2024.

[15] Lanxiang Hu, Siqi Kou, Yichao Fu, Samyam Rajbhandari, Tajana Rosing, Yuxiong He, Zhijie Deng, and Hao Zhang. Fast and accurate causal parallel decoding using jacobi forcing. *arXiv preprint arXiv:2512.14681*, 2025.

[16] John Kirchenbauer, Abhimanyu Hans, Brian Bartoldson, Micah Goldblum, Ashwinee Panda, and Tom Goldstein. Multi-token prediction via self-distillation. *arXiv preprint arXiv:2602.06019*, 2026.

[17] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pages 3519–3529. PMlR, 2019.

[18] Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. Cllms: consistency large language models. In *Proceedings of the 41st International Conference on Machine Learning*, pages 25426–25440, 2024.

[19] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[20] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7421–7432, 2024.

[21] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, pages 28935–28948, 2024.

[22] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[23] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[24] Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding for accelerating llms via double early exiting. *Advances in Neural Information Processing Systems*, 37:11946–11965, 2024.

[25] Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Jan Kautz, Ce Zhang, and Pavlo Molchanov. Tidar: Think in diffusion, talk in autoregression. *arXiv preprint arXiv:2511.08923*, 2025.

[26] Xiaohao Liu, Xiaobo Xia, Weixiang Zhao, Manyi Zhang, Xianzhi Yu, Xiu Su, Shuo Yang, See-Kiong Ng, and Tat-Seng Chua. L-mtp: Leap multi-token prediction beyond adjacent context for large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[27] Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in neural information processing systems*, 35:2507–2521, 2022.

[28] Somesh Mehra, Javier Alonso Garcia, and Lukas Mauch. On multi-token prediction for efficient llm inference. *arXiv preprint arXiv:2502.09419*, 2025.

[29] Ramesh Nallapati, Bowen Zhou, Cicero Dos Santos, Çağlar Gulçehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of the 20th SIGNLL conference on computational natural language learning*, pages 280–290, 2016.

[30] Zhiyuan Ning, Jiawei Shao, Ruge Xu, Xinfei Guo, Jun Zhang, Chi Zhang, and Xuelong Li. Caspec: Cascade adaptive self-speculative decoding for on-the-fly lossless inference acceleration of llms. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[31] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of machine learning and systems*, 5:606–624, 2023.

[32] Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2401–2410, 2020.

[33] Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your llm knows the future: Uncovering its multi-token prediction potential. *arXiv preprint arXiv:2507.11851*, 2025.

[34] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

[35] Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. Swift: On-the-fly self-speculative decoding for llm inference acceleration. In *The Thirteenth International Conference on Learning Representations*, 2025.

[36] Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6980–7008, 2025.

[37] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[38] Hossein Entezari Zarch, Lei Gao, Chaoyi Jiang, and Murali Annavaram. Del: Context-aware dynamic exit layer for efficient self-speculative decoding. *arXiv preprint arXiv:2504.05598*, 2025.

[39] Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft& verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11263–11282, 2024.

[40] Guoliang Zhao, Ruobing Xie, An Wang, Shuaipeng Li, Huaibing Xie, and Xingwu Sun. Self-distillation for multi-token prediction. *arXiv preprint arXiv:2603.23911*, 2026.

[41] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

A Limitations and Future Directions

While SEED achieves strong speedups and preserves task quality across the settings studied in this work, several limitations remain.

Scaling to Larger Models. Our experiments focus on `Qwen3-1.7B-Base` and `Qwen3-4B-Base` due to computational constraints. Although these models cover multiple tasks and already demonstrate consistent gains, we have not yet validated SEED on larger models like 8B, 14B, or 32B-scale LLMs. The efficiency-quality tradeoff may change with model size: larger models have deeper stacks and potentially richer intermediate representations, which could make the encoder-decoder partition more effective. Evaluating SEED at larger scales is therefore an important next step.

Beyond Task-Specific Fine-Tuning. In this paper, SEED is mainly studied in task-specific supervised fine-tuning settings. An important direction for future work is extending SEED beyond task-specific fine-tuning to larger-scale post-training or even pretraining settings. Applying the speculative drafting objective during general instruction tuning, preference optimization, continual post-training, or pretraining could produce models that are natively compatible with SEED across a wider range of prompts and domains. Such a setting would also allow us to study whether the "planning-ahead" behavior induced by SEED improves general model capabilities and robustness beyond the task-specific benchmarks considered here.

Adaptive Partitioning and Hardware-Aware Implementation. Our main experiments use a simple fixed partition, where the top two transformer layers serve as the decoder. This design is intentionally simple, but it is unlikely to be optimal for all model sizes, tasks, sequence lengths, or deployment environments. Future work could explore adaptive partitioning strategies that choose the encoder-decoder split based on model depth, layerwise representational quality, draft acceptance statistics, or runtime constraints. In addition, practical speedup depends heavily on implementation details such as KV-cache layout, tree-attention kernels, batching behavior, and GPU memory bandwidth. Hardware-aware implementations, including custom kernels for decoder-only drafting and tree-based verification, could further improve the realized speedup of SEED in production serving systems.

B Additional Discussions on Related Work

A high-level comparison of representative LLM inference acceleration methods, including our approach, is summarized in Table 3. Unlike many prior methods, SEED does not bring additional system complexity by introducing external modules, while achieving considerable inference speedups. More importantly, because SEED preserves the standard next-token prediction objective and achieves generation quality on par with autoregression, it can be seamlessly integrated into existing supervised fine-tuning pipelines. This allows training data to simultaneously improve both inference efficiency and generation quality, an advantage not shared by most existing LLM acceleration methods, which typically leverage training data solely for inference speedup, often at the expense of generation quality.

B.1 Connection to Self-Speculative Decoding with Layer Skipping

A closely related approach to SEED is self-speculative decoding with *layer skipping*, which accelerates drafting by omitting a subset of transformer layers during the draft forward pass [24, 35, 39]. Using the notation from Section 4, this strategy can also be interpreted within our encoder–decoder framework. Specifically, the skipped layers $\mathcal{E} = \{e_1, e_2, \ldots, e_s\}$ can be viewed as the encoder, while the remaining layers $\mathcal{D} = \{d_1, \ldots, d_{l-s}\}$ act as the decoder that generates draft tokens.

However, this formulation reveals an inherent limitation of layer-skipping approaches. Suppose we have some verified context tokens $X_1, X_2, \ldots, X_n$ and are about to propose a new draft token. At any decoder layer $\ell \in \mathcal{D}$, the attention mechanism

$$\text{Attn}(H_n^\ell, KV_{<n}^\ell)$$

operates on representations at the same layer, where verified tokens ($i < n$) are represented by their cached keys and values $KV_{<n}^\ell$ obtained during the previous verification step. In typical layer-skipping

Table 3: **Comparison of SEED against existing LLM acceleration methods.** "Lossless" denotes no degradation in generation quality compared to standard autoregressive models.

Method	No Ext. Module	Lossless	Training Cost	Speedup
General MTP				
Meta MTP [14] & DeepSeek MTP [23]	✗	✓	High	Mid
FastMTP [6]	✗	✓	Low	Mid
Apple MTP [33]	✗	✓	Low	High
Jacobi Forcing [15]	✓	✗	Low	High
Diffusion Language Models				
E2D2 [3]	✓	✗	Low	Mid
TiDAR [25]	✓	✗	High	High
General Speculative Decoding				
MEDUSA [5]	✗	✓	Low	High
EAGLE-3 [22]	✗	✓	High	High
Self-Speculative Decoding				
LayerSkip [12]	✓	✗	Low	Low
SWIFT [35] & CLaSp [9]	✓	✓	None	Low
CAS-Spec [30]	✓	✓	None	Mid
SEED (Ours)	✓	✓	Low	High

schemes, a large portion of the retained layers $\mathcal{D}$ include early layers of the transformer. Consequently, the KV caches available to these layers correspond to relatively shallow representations of the verified prefix. The draft model therefore attends only to low-level contextual features when generating candidate tokens, which can significantly degrade draft quality.

In contrast, SEED places the drafting component strictly in the *last few* layers of the transformer ($\ell \in [l' + 1, l]$). Rather than conditioning on a single encoder-produced hidden state, the decoder attends directly to the verified KV caches $KV^{dec}_{\leq n}$, which were computed after the verified tokens passed through all l' encoder layers. These cached keys and values therefore encode the deepest contextual representations available in the model and effectively serve as the shared encoder state.

As a result, the draft model in SEED always conditions on deep, highly contextualized representations of the verified prefix. This asymmetric design preserves the representational strength of the full model while keeping the draft computation inexpensive, enabling higher draft acceptance rates and improved overall decoding throughput.

B.2 Comparison to the EAGLE Series

Here we provide a deeper theoretical and architectural comparison between SEED and the EAGLE series of speculative decoding algorithms [21, 20, 22]. Both approaches share a fundamental motivation: draft models can achieve significantly higher acceptance rates if they are conditioned on deep, highly structured contextual representations of the verified prefix, rather than shallow representations or raw tokens alone. However, the methods diverge in how they formulate the drafting objective and system architecture.

The original EAGLE [21] formulates the draft model as performing feature-level autoregression. Given a sequence of hidden representations $H_0, H_1, \ldots, H_{n-1}$, the draft model learns to predict the next hidden representation H_n. Because continuous hidden representations are highly structured, this regression task becomes easier to learn than directly predicting raw discrete tokens. However, conditional dependencies in language are based on *discrete identity*. To resolve this, EAGLE samples the discrete token X_{n+1} from H_n and then fuses the embedding of X_{n+1} with H_n to predict the next step. Yet one problem remains: during training, EAGLE's draft model receives ground-truth features from the target model as input at every position, but at inference time, the draft model must

condition on its own predicted features for speculative positions. This creates an inherent exposure bias and distribution shift, because if the draft model produces poor feature predictions early on, subsequent inputs will be severely out-of-distribution. This motivates the authors in EAGLE-3 [22] to drop feature regression entirely and instead focus purely on token prediction, handling the exposure bias through a training-time drafting mechanism.

Meanwhile, SEED operates under the simplified hypothesis that hidden representations are valuable not because they are easier to predict, but because they make predicting the next token easier. By formulating drafting as an implicit encoder-decoder process, SEED avoids the feature-level distribution shift entirely. During both training and inference, the decoder processes draft tokens using only their raw token embeddings (H^0), while cross-attending to the deep contextual representations (KV^{dec}) of the verified prefix. Because the decoder is explicitly trained to handle this specific mixture of representation types, it never has to condition on its own continuous feature predictions. Consequently, SEED achieves high draft quality through direct token prediction, without the need for auxiliary feature regression objectives or feature fusion mechanisms. Moreover, because the drafter shares parameters with the verifier, it remains better aligned with the target model's distribution. In contrast to EAGLE-3, which conditions drafting on multiple levels of intermediate features and requires a complex training-time test simulation to mitigate exposure bias, SEED proves that it is possible to achieve higher draft quality by simply reusing the verifier's high-level contextual representations through the KV cache.

B.3 Comparison to E2D2

E2D2 [3] pioneered the amortization principle of separating expensive clean-context encoding from repeated lightweight generation. SEED can be viewed as bringing this principle into autoregressive decoding with exact verifier-based acceptance. Concretely, the new components are: (i) the speculative drafting objective that trains the last layers to draft from raw token embeddings while conditioning on verifier-cached deep KV representations, (ii) the fully block-causal training masks that preserve the verifier as a valid autoregressive model, and (iii) the resulting draft-then-verify inference loop in which verification and encoding merge into a single forward pass. Crucially, this AR speculative formulation changes the nature of the result: instead of merely extending the quality–efficiency frontier, exact verification enables acceleration without sacrificing model performance.

To disentangle whether this advantage comes from the formulation or from implementation / training choices, we run a controlled comparison against E2D2 on GSM8K with `Qwen3-1.7B-Base`, matching the training budget, training block size $b = 4$, and encoder-decoder split, and disabling SEED's adaptive drafting and tree verification with a fixed draft length of 4. This matches the split and generation span, while the inference procedures retain their respective diffusion and speculative computations.

Table 4: Controlled comparison with E2D2. We compare E2D2 and SEED across different decoder sizes by training and evaluating corresponding `Qwen3-1.7B-Base` models on GSM8K. SEED achieves much better task performance while using a smaller decoder, proving that exact verification enables acceleration without sacrificing model performance.

Decoder Size (# layers)	E2D2 Acc. (%)	E2D2 Tput (tokens/s)	SEED Acc. (%)	SEED Tput (tokens/s)
2	23.7	75.9	57.5	74.3
4	26.6	67.6	57.9	66.1
6	32.3	63.2	59.4	60.8
8	32.4	57.3	58.8	56.3
10	36.3	51.2	57.4	53.3
12	41.3	46.4	57.0	48.8
14	43.4	42.8	57.6	46.4

The two methods achieve nearly identical throughput at every split, confirming that they share the same amortization mechanism. The difference is in task quality: E2D2 must trade accuracy for speed, whereas SEED's accuracy is essentially flat across all decoder sizes because the exact verifier guarantees that the output distribution matches autoregressive decoding regardless of drafter capacity. This lets us use the smallest, fastest decoder while securing both speed and accuracy.

C Further Details on Experimental Setup

C.1 Hardware

All 1.7B models are fine-tuned on a single 48 GB NVIDIA A6000 GPU, and all 4B models are fine-tuned on either NVIDIA 80 GB H100 GPUs or 180 GB B200 GPUs. Some 4B models are trained on two H100 GPUs using PyTorch Fully Sharded Data Parallel (FSDP) [41], while others are trained on a single B200 GPU. All trained models are evaluated on a single NVIDIA A6000 GPU to measure task performance and throughput.

C.2 Datasets

Following E2D2 [3], we train and evaluate task-specific models on four datasets covering math reasoning, code generation, general knowledge, and summarization: GSM8K [10], KodCode [36], ScienceQA [27], and CNN/Daily Mail [29].

For GSM8K, we use the Hugging Face dataset `openai/gsm8k`, with the official train split for training and test split for evaluation. We pre-process the data by adding a prefix `"Please reason step by step, and put your final answer within $\boxed{}$."` to the inputs, prefacing the answers with `"Answer:  "`, and wrapping the solutions in `"$\boxed{}$"`. Inputs and targets are truncated to a maximum length of 384 tokens each, ensuring a maximum sequence length of 768 during training. Final evaluation is performed in 0-shot mode using the `lm-eval harness` library with the "flexible match" criteria, and similar pre-processing is applied to question texts. Post-processing is done to truncate text at `<|endoftext|>` tokens, and solutions are reverted to their original form of `"### <Answer>"`. Inputs are pre-processed as above.

For KodCode, we use the Hugging Face dataset `KodCode/KodCode-V1-SFT-R1` with a deterministic reconstruction of train/test splits. We first keep only samples with online judge style questions, then randomly subsample 50% of these examples, and finally define the test set as the last 1000 samples of this subsample while using the remainder as the training set. Each problem is prompted as a Python programming task ending with a `[BEGIN]` marker. Inputs and targets are truncated to a maximum length of 512 tokens each, ensuring a maximum sequence length of 1024 during training. During evaluation, we restrict the test set to problems marked with difficulty level "easy". For each problem, the model generates a program which is then executed against the provided test cases, and we report exact problem-solving accuracy based on whether all tests pass.

For ScienceQA, we use the Hugging Face dataset `derek-thomas/ScienceQA` and restrict it to text-only examples. The training set is constructed by merging the original train and validation splits, while the evaluation set is created by sampling 1000 examples from the original test split. Inputs are formatted as multiple-choice questions with answer options labeled (A) to (H), and targets consist of the reasoning followed by a canonical final statement of the form `"The answer is (X)"`. Inputs and targets are truncated to a maximum length of 384 tokens each, ensuring a maximum sequence length of 768 during training. For evaluation, the predicted option letter is extracted from the generated text, and exact-answer accuracy is reported.

For CNN/Daily Mail, we use the Hugging Face dataset `abisee/cnn_dailymail` (version 3.0.0), with the official train split for training and the test split for evaluation. Inputs consist of the news article prefixed with a summarization instruction, while targets are the corresponding summaries prefixed with `"Summary:  "`. Unlike the previous datasets, we allocate the token budget asymmetrically, assigning 90% of the sequence length to the article and 10% to the summary, for a total sequence length of 1024 tokens. During evaluation, we apply the same length constraints and evaluate on up to 1000 samples from the test split. Summarization quality is measured using ROUGE-1, ROUGE-2, and ROUGE-L.

C.3 Baselines

Our overall implementation builds on the official E2D2 codebase[2] [3]. Accordingly, our AR and E2D2 baselines follow their implementation. For E2D2, we use the last 4 layers as the decoder for a better trade-off between performance and speedup. Consistent with the original setup, E2D2 is trained and evaluated using a block size of 4.

[2] https://github.com/kuleshov-group/e2d2

For Apple MTP [33], the original paper does not provide an official code release. We therefore adopt the open-source unofficial implementation from https://github.com/siihwanpark/MTP-GLoRA and make the necessary modifications to better match the algorithmic details described in the paper. We fine-tune the model to handle 8 mask tokens as input by training gated-LoRA weights and a 2-layer MLP sampler module, using a LoRA rank of 16 and a LoRA alpha of 32. For evaluation, we set the draft length to 8 across all datasets, with the exception of CNN/Daily Mail, where a draft length of 4 is used.

For LayerSkip [12], the released repository provides inference code but not training code. We therefore use the implementation in Hugging Face TRL[3] and make the necessary adjustments to align it with the training procedure described in the original paper. The early-exit layer is set to layer 8 for `Qwen3-1.7B-Base` and layer 12 for `Qwen3-4B-Base`. We fine-tune the base models using a composite objective $L = L_{\text{final}} + \lambda \cdot L_{\text{early-exit}}$ with $\lambda = 1.0$. Training follows a rotational early-exit curriculum with stride 8, and layer dropout is applied with a constant schedule and a maximum dropout rate of 0.1.

For EAGLE-3 [22], we use the publicly available AngelSlim checkpoints `AngelSlim/Qwen3-1.7B_eagle3`[4] and `AngelSlim/Qwen3-4B_eagle3`[5] paired with `Qwen3-1.7B` and `Qwen3-4B` instruct models due to the lack of corresponding EAGLE-3 weights for the base models. Evaluation is conducted using the official EAGLE GitHub repository[6]. Note that although EAGLE-3 reports more than $5\times$ average speedup on Vicuna-, LLaMA-, and DeepSeek-based models, our experiments on `Qwen3-1.7B` and `Qwen3-4B` show only around $2\times$ speedup on a single NVIDIA A6000. This observation is consistent with the results reported by AngelSlim in https://huggingface.co/AngelSlim/Qwen3-1.7B_eagle3.

For PARD [1], we evaluate the released target-independent Qwen3 drafter[7], paired with the corresponding task-specific, fine-tuned `Qwen3-4B-Base` AR checkpoint under the same evaluation setting as SEED. We include PARD only in the 4B portion of Table 1, as its 0.6B-parameter drafter is relatively large for a 1.7B target model.

For SWIFT [35] and DEL [38], we use the authors' released implementations directly. SWIFT runs on top of our fine-tuned AR checkpoints, with Bayesian optimization interval 1, update interval 25, layer skip ratio 0.4, context window 50, maximum optimization iterations 1000, tolerance limit 300, and early-stop score threshold 0.95. DEL runs on top of our fine-tuned LayerSkip checkpoints by maintaining an exponential moving average on acceptance/confidence statistics with retention factor $\omega = 0.95$.

C.4 Training

We train all models using the MosaicML Composer framework[8]. Unless otherwise noted, we use the same optimization setup across methods to ensure a fair comparison. Specifically, we apply a linear learning-rate warm-up for the first 100 steps, followed by cosine decay. Model checkpoints are selected based on validation loss with early stopping. We set the random seed to 1 during training.

Training hyperparameters vary by model scale and dataset, primarily in learning rate, batch size, and maximum sequence length. Table 5 summarizes the final training setup.

C.5 Task Prompts

We now describe the task prompts used to train and evaluate the baseline methods in our experiments. The prompts are designed to be simple and straightforward, as detailed below:

[3] https://github.com/huggingface/trl/pull/3111
[4] https://huggingface.co/AngelSlim/Qwen3-1.7B_eagle3
[5] https://huggingface.co/AngelSlim/Qwen3-4B_eagle3
[6] https://github.com/SafeAILab/EAGLE
[7] https://huggingface.co/amd/PARD-Qwen3-0.6B
[8] https://docs.mosaicml.com/projects/composer

Table 5: **Training configurations used for task-specific fine-tuning.** For each dataset and model scale, all methods share the same optimizer schedule and training budget to maintain comparability.

Dataset	Base Model	Batch Size	LR	Steps	Max Length
GSM8K	`Qwen3-1.7B-Base`	1	$1e^{-5}$	30,000	768
KodCode	`Qwen3-1.7B-Base`	1	$1e^{-5}$	30,000	1024
ScienceQA	`Qwen3-1.7B-Base`	1	$1e^{-5}$	30,000	768
CNN/Daily Mail	`Qwen3-1.7B-Base`	32	$1e^{-5}$	30,000	1024
GSM8K	`Qwen3-4B-Base`	1	$5e^{-6}$	30,000	768
KodCode	`Qwen3-4B-Base`	1	$5e^{-6}$	30,000	1024
ScienceQA	`Qwen3-4B-Base`	1	$5e^{-6}$	30,000	768
CNN/Daily Mail	`Qwen3-4B-Base`	32	$5e^{-6}$	30,000	1024

Task Prompts

GSM8K

Please reason step by step, and put your final answer within $\boxed{}$. {question}

KodCode

You are an expert Python programmer. Solve the following problem.

{question}

ScienceQA

The following is a multiple choice question. Think step by step and then give your final answer.

{question}

(A) {choice A}
(B) {choice B}
...

CNN/Daily Mail

Summarize the following article: {article}

C.6 Evaluation

We evaluate all trained models using greedy decoding. To measure decoding throughput, we adopt a different protocol from prior work, which typically continues generation until the maximum sequence length is reached. We find that if we force models to continue generating after the `<|endoftext|>` token, draft models tend to adapt to repetitive output patterns, which can artificially increase drafting accuracy and decoding throughput. However, in real-world applications, we only care about the tokens generated before the `<|endoftext|>` token. Therefore, in all experiments, we measure decoding throughput by terminating generation as soon as the `<|endoftext|>` token is generated.

D SEED Implementation Details

D.1 Algorithms

We include SEED training and inference with fixed draft length in Algorithm 1 and Algorithm 2. Since our main experiments use adaptive drafting with tree-based verification, we further describe that procedure here and illustrate it in Figure 8.

Confidence-Aware Dynamic Drafting. A fixed draft length is often suboptimal because the decoder's confidence varies across generation steps. Instead of always drafting exactly d tokens, we let the decoder continue drafting as long as its confidence remains above a predefined threshold.

Algorithm 1 SEED Training

Require: Transformer $f = f_{dec} \circ f_{enc}$, encoder-decoder divider l', training data D, block-size b, speculative drafting objective weight λ

for $X \in D$ **do**
  $H^0 \leftarrow W_{in}X$
  $H^{l'} \leftarrow f_{enc}(X)$
  ▷ Standard next-token prediction loss
  $\mathcal{L}_{CE} \leftarrow \sum_{j=2}^{|X|} -\log f_{dec}(X_j \mid H^{l'}_{j-1}, KV^{dec}_{<(j-1)})$
  } Computed in a single forward pass of f
  ▷ In practice, this for loop can be computed in parallel for all blocks using the block causal attention mask illustrated in Figure 2
  $L_{spec} \leftarrow 0$
  for $j = 2$ to $|X|$ **do**
    ▷ Compute the start index of the block to which the last input token X_{j-1} belongs
    $n' \leftarrow \lfloor \frac{j-2}{b} \rfloor \times b + 1$
    ▷ Speculative drafting objective
    $\mathcal{L}_{spec} \leftarrow \mathcal{L}_{spec} - \log f_{dec}\left(X_j \mid H^0_{n' \le \cdot \le (j-1)}, KV^{dec}_{<n'}\right)$
  end for
  $L = \frac{\mathcal{L}_{CE} + \lambda \mathcal{L}_{spec}}{1+\lambda}$
  $\nabla\theta \leftarrow \text{Backprop}(\mathcal{L})$
  $f \leftarrow \text{OptimizerStep}(f, \nabla\theta)$
end for
return f

Algorithm 2 SEED Inference with Fixed Draft Length

Require: Transformer with encoder $f_{enc} = f_{1:l'}$, decoder $f_{dec} = f_{l'+1:l}$, prompt $X_{1:t}$, draft length d, target length T

Initialize $KV_{1:t}$ by calling $(f_{dec} \circ f_{enc})(X_{1:t})$
while $t < T$ **do**
  ▷ Drafting Phase: Autoregressively draft d tokens using only the decoder
  $X_{draft}[0] \leftarrow X_t$
  for $i = 1$ to d **do**
    $q_i(x) \leftarrow f_{dec}(W_{in}X_{draft}[i-1], KV^{dec}_{1:(t+i-2)})$
    $X_{draft}[i] \leftarrow \arg\max q_i(x)$
  end for
  $KV_{1:(t-1)} \leftarrow \text{Truncate}(KV_{1:(t+d-1)}, t-1)$
  ▷ Verification Phase: Forward the full model in parallel
  $p_1(x), \ldots, p_{d+1}(x) \leftarrow (f_{dec} \circ f_{enc})(X_{draft}[0:d], KV_{1:(t-1)})$
  $n_{accept} \leftarrow 0$
  for $i = 1$ to d **do**
    if $\arg\max_x p_i(x) = X_{draft}[i]$ **then**
      $n_{accept} \leftarrow n_{accept} + 1$
    else
      break
    end if
  end for
  ▷ Append verified tokens and a bonus correction token
  $x_{new} \leftarrow \arg\max_x p_{n_{accept}+1}(x)$
  $X_{(t+1):(t+n_{accept}+1)} \leftarrow [X_{draft}[1:n_{accept}], x_{new}]$
  $KV_{1:(t+n_{accept})} \leftarrow \text{Truncate}(KV_{1:(t+d)}, t+n_{accept})$
  $t \leftarrow t + n_{accept} + 1$
end while
return $X_{1:T}$

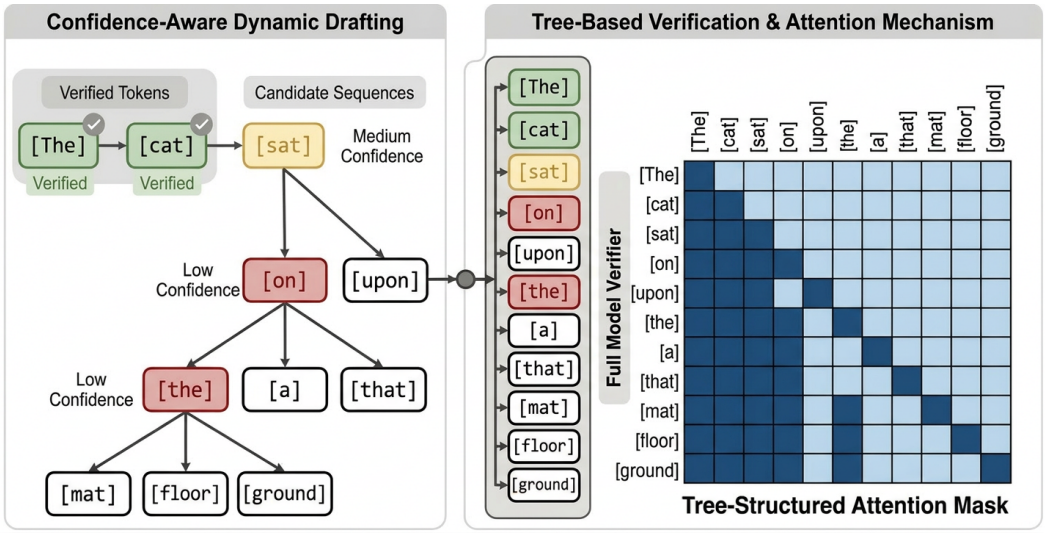

Figure 8: Illustration of confidence-aware dynamic drafting and tree-based verification in SEED. **Left:** starting from a verified prefix, the lightweight decoder drafts autoregressively, with the current node expanded into multiple candidates using a top-p rule, thus producing a small tree of candidate continuations. **Right:** all candidate tokens are packed and verified in one full-model forward pass using a tree-structured attention mask. Each candidate token attends to the entire verified prefix and to the candidate tokens on its own ancestral path, but not to tokens from unrelated branches. This allows parallel verification of multiple draft branches while preserving the causal dependencies of autoregressive decoding.

Starting from the current verified prefix, the decoder autoregressively proposes the next token while conditioning on the verified decoder KV cache and the raw embeddings of previously drafted tokens, exactly as in the fixed-length setting. Also, at each drafting step, we expand the current node into multiple sibling candidates using a top-p rule, following prior adaptive self-speculative decoding methods. Repeating this procedure over several steps produces a small tree of candidate continuations rather than a single draft sequence. This confidence-aware strategy allocates computation more effectively. When the decoder is confident, generation proceeds with a narrow branch and low drafting overhead. When uncertainty increases, the procedure broadens the candidate set so that the verifier can evaluate several plausible continuations in parallel. In practice, we set $\tau = 0.7$ for GSM8K, KodCode, and ScienceQA, and $\tau = 0.5$ for CNN/Daily Mail. For candidate expansion under the top-p rule, let p_0 denote the probability of the current top-1 draft token. We then choose the number of candidate draft tokens k at each step according to

$$k = \begin{cases} 1 & \text{if } p_0 > 0.95, \\ 3 & \text{if } 0.8 < p_0 \leq 0.95, \\ 5 & \text{if } 0.5 < p_0 \leq 0.8, \\ 10 & \text{otherwise.} \end{cases}$$

Tree-Based Verification. Once the candidate tree is constructed, we verify all drafted tokens with a single full-model forward pass. To do so, we linearize the tree nodes into a packed sequence and apply a tree-structured attention mask that preserves the correct autoregressive dependencies. Concretely, every candidate token is allowed to attend to: (i) all verified prefix tokens, and (ii) the candidate tokens on its own ancestral path in the draft tree. Tokens from different branches are not allowed to attend to one another unless they share the same ancestors. This reproduces exactly the causal structure each branch would have under independent autoregressive evaluation, while enabling all candidates to be processed in parallel in one verifier call. After the verifier produces logits for all candidate nodes, we traverse the tree from the root and greedily accept the longest prefix whose verified predictions match the drafted tokens. As in standard speculative decoding, after the accepted draft prefix is determined, we additionally append one bonus token predicted by the verifier at the

first non-accepted position. The KV cache is then truncated to the accepted path, the verified prefix is updated, and the next round of drafting begins.

D.2 Further Ablations

Effect of the Confidence Threshold τ. In our main experiments, SEED employs a dynamic drafting strategy that terminates drafting when the top-1 probability of the draft token falls below a confidence threshold τ. To study the impact of τ on decoding efficiency, we vary the threshold and measure the resulting decoding throughput, draft token acceptance rate, and average acceptance length. Specifically, we evaluate a GSM8K-finetuned `Qwen3-1.7B-Base` SEED model on GSM8K, and the results are shown in Figure 9. As τ increases, the acceptance rate of draft tokens consistently improves, indicating that the internal confidence of the draft model correlates positively with draft quality. However, a higher τ also shortens the average acceptance length because drafting terminates earlier. Despite this trade-off, decoding throughput remains relatively stable within a wide range of thresholds ($\tau \in [0.5, 0.9]$), demonstrating that SEED is robust to the choice of τ. In our experiments, we select $\tau = 0.7$ as it provides a favorable balance between acceptance rate and drafting length. For more challenging generation tasks such as CNN/Daily Mail summarization, where draft token acceptance lengths are typically shorter, we adopt a slightly lower threshold ($\tau = 0.5$) to maintain longer drafting segments and better efficiency.

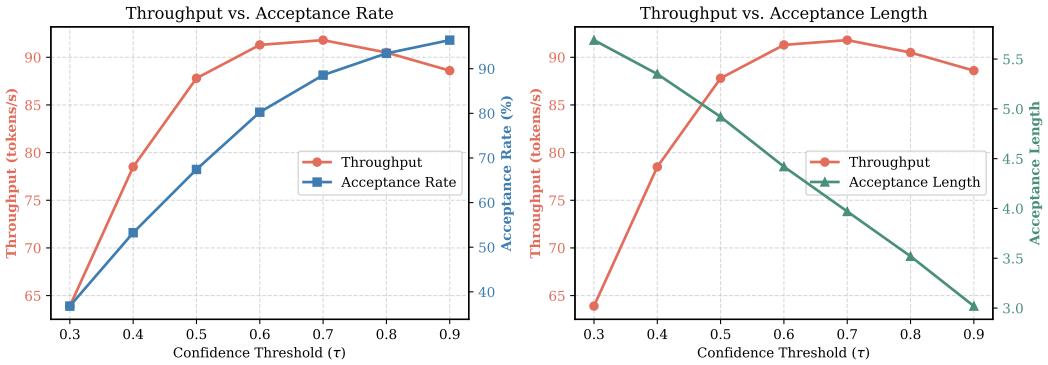

Figure 9: Effect of the confidence threshold τ used in dynamic drafting. We evaluate a GSM8K-finetuned `Qwen3-1.7B-Base` SEED model on GSM8K and report average decoding throughput, draft token acceptance rate, and draft token acceptance length. Increasing τ improves acceptance rate but shortens acceptance length due to earlier termination of drafting. Overall decoding throughput remains stable for τ between 0.5 and 0.9, indicating that SEED is relatively insensitive to the precise choice of τ.

Table 6: Ablation study on tree-based verification. We evaluate a GSM8K-finetuned `Qwen3-1.7B-Base` SEED model on GSM8K. We report decoding throughput (Tput) in tokens/s, average draft token acceptance length (Acc. Len), and draft token acceptance rate (Acc. Rate) under fixed-length and dynamic drafting strategies. Tree-based verification consistently improves acceptance statistics and overall throughput.

Method	**Fixed-Length Drafting** ($d = 4$)			**Fixed-Length Drafting** ($d = 8$)			**Dynamic Drafting** ($\tau = 0.7$)		
	Tput	Acc. Len	Acc. Rate	Tput	Acc. Len	Acc. Rate	Tput	Acc. Len	Acc. Rate
w/ Tree Attn	82.3	3.16	79.80%	82.5	4.65	59.10%	91.8	3.97	88.57%
w/o Tree Attn	76.9$^{\downarrow 5.4}$	2.67	69.45%	75.2$^{\downarrow 7.3}$	3.67	46.64%	85.1$^{\downarrow 6.7}$	3.28	77.59%

Effect of Tree-Based Verification. In our main experiments, we also keep multiple candidate draft tokens at each drafting step, forming a draft tree that is subsequently verified in parallel by the full model using a tree-structured attention mask. To evaluate the contribution of this design to decoding efficiency, we ablate the tree-based verification mechanism by disabling tree attention and restricting drafting to a single candidate per step. We conduct this study on a GSM8K-finetuned

`Qwen3-1.7B-Base` SEED model and report results on GSM8K. As shown in Table 6, enabling tree-based verification consistently improves throughput, acceptance length, and acceptance rate across both fixed and dynamic drafting settings. These results demonstrate that expanding the candidate set and verifying branches in parallel substantially improves draft token acceptance, leading to higher overall decoding speedup.

Ablating Parameter Sharing. To isolate the contribution of parameter sharing, we train an additional SEED control on GSM8K using `Qwen3-1.7B-Base` in which the drafter and verifier are fully parameter-disjoint, aiming to directly test whether enforced parameter sharing improves drafter-verifier alignment relative to an identically initialized but independently parameterized drafter. The two-layer drafter is initialized by copying the verifier's token embedding and tied output head, final normalization layer, and top two transformer layers. The drafter continues to consume the verifier-produced KV cache from layers 27 and 28, preserving the deep-feature conditioning mechanism. Both variants in this ablation use training block size $b = 4$. All other training settings and objectives are unchanged. We evaluate both variants with a fixed draft length of 4 and without tree-based verification.

Table 7 shows that removing parameter sharing substantially degrades both draft quality and task performance, despite the fact that this variant has more parameters and, in principle, greater expressive capacity. Although the independently parameterized drafter retains the same initialization and deep-feature conditioning mechanism, it still suffers from poorer draft token acceptance, resulting in reduced throughput. The degraded task performance further suggests that parameter sharing provides an important alignment benefit between the drafter and the verifier, making the joint optimization process more effective.

Table 7: Parameter-sharing ablation on GSM8K with `Qwen3-1.7B-Base`. Removing parameter sharing substantially degrades both draft quality and task performance.

	GSM8K Acc. (%)	Throughput (tokens/s)	Draft Token Acc. Rate (%)	Draft Token Acc. Length
SEED	57.5	74.3	73.0	2.9
w/o Parameter Sharing	44.1$^{\downarrow 13.4}$	68.8$^{\downarrow 5.5}$	68.5$^{\downarrow 4.5}$	2.7$^{\downarrow 0.2}$

E Additional Experimental Results

E.1 Why SEED Improves Task Performance

Here we provide some further theoretical intuition as to why multi-token prediction (MTP) and SEED achieve stronger task performance than standard autoregressive (AR) fine-tuning. We also include additional results of training linear probes to predict future tokens on two more datasets in Figures 10 and 11, using the intermediate representations of `Qwen3-1.7B-Base` AR and SEED models.

In MTP-style training, the model learns to predict multiple future tokens from a single contextual representation. Concretely, given the hidden states of the final context token $h(t_0)$, the model learns to approximate $P(t_1, t_2, t_3 \mid h(t_0))$, forcing $h(t_0)$ to encode information that is predictive not only of the immediate next token but also of tokens further into the future, which encourages the model to implicitly plan ahead.

Similarly, during SEED drafting, the decoder learns to model $P(t_1 \mid H^0_{t_0}, KV^{dec}_{<t_0})$, $P(t_2 \mid H^0_{t_1}, H^0_{t_0}, KV^{dec}_{<t_0})$, $P(t_3 \mid H^0_{t_2}, H^0_{t_1}, H^0_{t_0}, KV^{dec}_{<t_0})$ and so on, where $H^0_{t_i}$ denotes the raw token embedding of token t_i. Importantly, these embeddings are fixed lookup vectors that cannot adapt to absorb predictive information during training. As a result, the gradients produced by the speculative loss $\mathcal{L}_{spec}$ primarily propagate through the cached contextual representations $KV^{dec}_{<t_0}$, which originate from the encoder's hidden states. Consequently, the encoder is encouraged to compress information about upcoming tokens into its contextual representations so that the decoder can reliably predict future tokens during drafting.

For the linear probe experiments, we train both a SEED model and an AR model with a block size of 4 to examine the effect of planning ahead under a relatively short horizon. We then extract the

representations from the final encoder layer l' of the SEED model and the corresponding layer l' of its AR fine-tuned counterpart, run both models on the GSM8K test split, collect the hidden states produced at layer l', and train five linear probes to independently predict future tokens at lookahead steps from 1 to 5. Specifically, we collect 20,000 target-only token positions from 1,000 samples in the test split and compute prediction accuracy against the ground-truth labels from the original datasets.

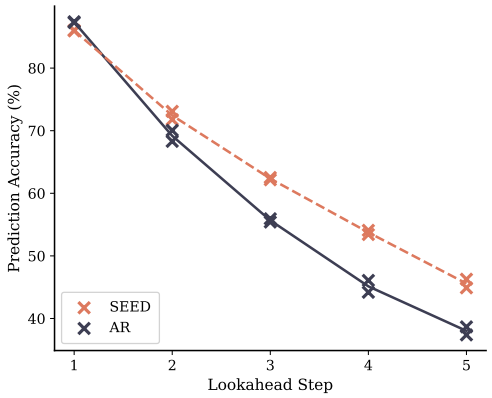

Figure 10: Training linear probes to predict future tokens from intermediate representations on Kodcode.

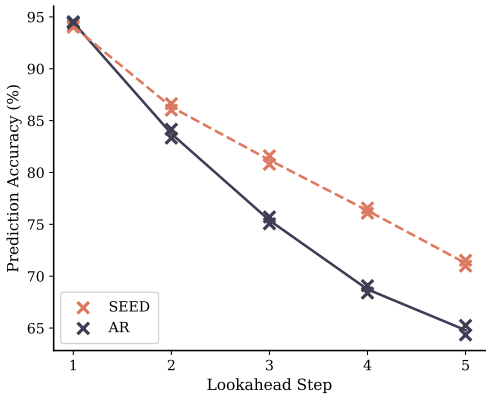

Figure 11: Training linear probes to predict future tokens from intermediate representations on ScienceQA.

E.2 Accelerating an Already Well-Tuned Model

In our main experiments, SEED is applied during task-specific fine-tuning, yielding both inference acceleration and improved task performance. A natural follow-up question is whether SEED can be used purely to accelerate an already strong task model, without sacrificing its existing quality. To study this practical setting, we start from a `Qwen3-1.7B-Base` model that has already been fine-tuned with standard autoregressive (AR) training on GSM8K. We then perform an additional round of training to convert this AR model into a SEED model. The training setup is identical to that described in Appendix C.4, except that initialization comes from the task-tuned AR checkpoint rather than the untuned base model.

Table 8 summarizes the results. Retrofitting SEED onto an already fine-tuned AR model preserves task accuracy while achieving essentially the same decoding throughput as a SEED model trained directly from the base model. The small accuracy gain is consistent with the MTP-like "planning ahead" effect induced by the speculative objective, as discussed in the main body.

These results are also aligned with our CKA analysis in Figure 5, which shows that SEED primarily reshapes internal model representations near the encoder–decoder interface while leaving earlier layers largely unchanged. This suggests that standard AR fine-tuning and SEED training are structurally compatible: the latter can be placed on top of the former without disrupting previously learned task knowledge.

Table 8: SEED applied as a second-stage fine-tuning procedure on top of an AR-fine-tuned model. Dynamic drafting ($\tau = 0.7$) and tree-based verification are used for SEED models. Continued fine-tuning preserves accuracy on GSM8K while achieving comparable speedup.

Model	Accuracy (%)	Tput (tokens/s)	Acc. Len	Acc. Rate (%)
AR (from base)	56.3	37.9	-	-
SEED (continued from AR)	56.5$^{\uparrow 0.2}$	90.6$^{\uparrow 52.7}$	3.91	86.82
SEED (from base)	57.1	91.8	3.97	88.57

F Computational Cost Analysis

Here we provide a more comprehensive analysis of the theoretical computational cost of each method. For concreteness, we use `Qwen3-1.7B-Base` as an example. The model has $l = 28$ transformer layers with $d_{\text{model}} = 2048$, $d_{\text{ffn}} = 6144$, 16 query heads, 8 KV heads, $d_{\text{head}} = 128$, a vocabulary size of $|V| = 151{,}936$, and tied input / output embeddings. Each attention block contains $W_Q\,(2048 \times 2048) + W_K\,(2048 \times 1024) + W_V\,(2048 \times 1024) + W_O\,(2048 \times 2048)$, for a total of 12.58M parameters. The MLP block contributes $3 \times (2048 \times 6144) = 37.75$M parameters. Therefore, each transformer layer contains $P_{\text{layer}} = 50.33$M parameters, and all 28 layers together contain 1.409B parameters. The tied embedding / LM head contributes an additional $151936 \times 2048 = 311.2$M parameters, resulting in a total of $P_{\text{full}} = 1.721$B parameters. Throughout the following analysis, we normalize the computational cost of a full forward pass through the entire model as *1 unit*, and express the cost of all compared methods relative to this baseline. Note that this parameter-based analysis is intended as a simple estimate of relative computational cost. It assumes that the amount of computation roughly scales with the number of parameters touched while ignoring operation-specific factors such as differences in FLOPs, memory access patterns, and hardware utilization across components.

Table 9: Approximate drafting cost for a 1.7B model. Note that Apple MTP amortizes a full-model call over a draft block.

Method	Weights Touched per Draft Token	Weights	Cost (units)
AR	28 layers + head	1721M	1.000
SEED	2 decoder layers + head	411.9M	0.239
LayerSkip	8 layers + head	713.8M	0.415
Apple MTP	(28 layers + head) per block + sequential sampler head	$\geq$ 1721M	≥ 1.000
EAGLE-3	1 fused draft layer + reduced-vocab head	$\approx$ 137M	≈ 0.080

As shown in Table 9, SEED's drafting step is $4.2\times$ cheaper than a full pass and $1.7\times$ cheaper than LayerSkip's early-exit pass, despite conditioning on strictly deeper representations.

Verification over $(d{+}1)$ tokens requires one full-model forward pass for all methods except LayerSkip. Since LayerSkip reuses the KV cache of the first 8 layers computed during drafting, verification only recomputes layers 9 - 28 together with the LM head. The corresponding cost is therefore $(20 \times 50.33 + 311.2)/1721 = 0.766$ relative to a full forward pass.

Now consider a fixed draft length d and per-token acceptance rate r. Each speculative decoding round incurs a computational cost of

$$T_{\text{unit}} = d \cdot c_{\text{draft}} + c_{\text{verify}}$$

where c_{draft} and c_{verify} denote the per-token drafting cost and the verification cost respectively. In return, each round commits an average of

$$n = d \cdot r + 1$$

tokens, consisting of the accepted draft tokens plus the bonus token generated during verification. Using the measured acceptance rates at $d = 4$ (SEED: 73.0%, Apple MTP: 69.8%, LayerSkip: 43.1%) and taking one computational unit to correspond to a full autoregressive forward pass ($1/37.9$ s $= 26.4$ ms, based on the fine-tuned `Qwen3-1.7B-Base` achieving 37.9 tokens/s on GSM8K with a single A6000), we obtain the following throughput predictions by computing $1000/T_{ms} \times n$:

Despite its simplicity, this computational model closely predicts SEED's measured throughput (w/ fixed draft length and w/o tree-based verification) and captures the relative performance trends across methods. The remaining discrepancies particularly for LayerSkip and Apple MTP are expected because the model omits certain operation-specific costs and implementation overheads.

For completeness, EAGLE-3 achieves a substantially lower per-token drafting cost (0.08 units), owing to its lightweight single-layer drafter and reduced-vocabulary draft head. Therefore, SEED's

Table 10: Approximate throughput predictions for fixed draft length $d = 4$ without tree-based verification. Apple MTP generates all four draft tokens in a single forward pass. However, its sampler applies the 311M-parameter LM head sequentially once per draft token, contributing an additional cost of $4 \times 311.2/1721 = 0.72$ units.

Method	c_{draft}	c_{verify}	T_{unit}	T (ms)	n	Predicted (tokens/s)	Measured (tokens/s)
SEED	0.239	1.000	1.96	51.7	3.92	**75.8**	**74.3**
LayerSkip	0.415	0.766	2.43	64.1	2.72	42.4	47.0
Apple MTP	$1.72/4$	1.000	2.72	71.8	3.79	52.8	44.6

advantage does not primarily come from reducing the absolute drafting cost. Instead, it arises from the improved draft quality and higher acceptance rates.

The relative drafting cost of SEED also becomes more favorable as the backbone model scales. For `Qwen3-4B-Base` (36 layers, $d_{\text{model}} = 2560$, $P_{\text{layer}} = 101$M, and a tied embedding / LM head of 389M parameters), the drafting cost is $(2 \times 101 + 389)/4020 = 0.147$ units, compared with 0.239 for `Qwen3-1.7B-Base`. This scaling behavior is consistent with the larger average speedup observed on the 4B model.

G Broader Impacts

This work develops SEED, a method for accelerating large language model (LLM) inference by improving self-speculative decoding. The primary positive societal impact is that faster inference can reduce the computational cost, latency, and energy consumption of deploying LLMs. More efficient inference may make advanced language models more accessible to researchers, developers, and users with limited computational resources, and may enable lower-cost applications in domains such as education, programming assistance, summarization, and scientific support.

At the same time, improving LLM inference efficiency can also amplify existing risks associated with language models. Lowering the cost and latency of generation could make it easier to produce large volumes of harmful or misleading content, including spam, phishing messages, disinformation, synthetic reviews, or other forms of automated manipulation. Faster inference may also facilitate misuse in surveillance, impersonation, or automated social engineering systems when combined with models capable of generating persuasive or personalized text.

Nevertheless, these risks are not unique to SEED, but they are relevant because the method improves the efficiency of LLM generation. Future work on efficient inference should continue to evaluate not only speed and quality, but also whether acceleration changes the scale or detectability of potential misuse.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We include a clear statement of the main claims made in the abstract and introduction. The scope of this paper is stated as well. The claims made in the abstract and introduction are empirically verified in Section 5, where the experimental results, analysis, and implications all support the claims made and our contributions.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We comprehensively discuss the limitations of our work in Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not include formal theoretical results such as theorems, lemmas, propositions, or corollaries that require assumptions and proofs. It contains numbered equations defining the model decomposition, training objectives, and inference procedures, as well as numbered algorithms, but these are methodological specifications rather than claimed theoretical results. The theoretical discussion is presented as intuition and is supported empirically through experiments and ablations, so no formal proofs are required.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We disclose all information needed for reproduction in Section 5.1 and Appendix C.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility.

In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We will provide data, code, and instructions for the final paper.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide all the experimental details in Section 5.1 and Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We do conduct extensive experiments on multiple benchmarks across different model sizes to support the main claims of the paper, and we report all the results in Section 5 and Appendix E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide information about the compute resources needed to reproduce the experiments in Appendix C.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: We have read and understood the code of ethics and have made every effort to adhere to it.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss both positive and negative societal impacts of our work in Appendix G.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper does not pose such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use publicly available datasets and pretrained models, all of which are properly cited in the paper. We adhere to the licenses and terms of use as specified by their original creators.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.

- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We will release our codebase along with detailed README files.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: We use LLMs solely for editing assistance.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.