DEX: Parallel Depth Exploration for Speculative Decoding

Weisi Yang
Northwestern University
weisi@u.northwestern.edu

Zipeng Sun
MILA – Quebec AI Institute
McGill University
zipeng.sun@mila.quebec

Stephen Xia
Northwestern University
stephen.xia@northwestern.edu

Project page: https://github.com/imec-nu/DEX

Abstract

Autoregressive LLM decoding processes every generated token through the full layer stack, even though many tokens become available at intermediate depths. Existing lossless depth-adaptive methods exploit this redundancy by choosing a single non-final exit depth and verifying its drafts with the final-depth model. However, our measurements show that this selection-based strategy leaves substantial headroom: choosing an exit too late wastes computation, while choosing one too early triggers fallback and discards dependent drafts. We propose Depth Exploration Decoding (DEX), a lossless speculative decoding algorithm that replaces single-depth selection with parallel exploration over multiple candidate depths. At each commit position, DEX validates candidates against the final-depth reference, commits the final-depth token, and collapses the exploration lattice to retain only reusable branch states. This expand–commit–collapse procedure preserves equivalence to standard autoregressive decoding while reducing the latency of generating each token. Across early-exit-trained and standard LLMs, DEX outperforms representative depth-selection baselines and achieves up to 61% higher decoding speed than the strongest depth-selection baseline. Moreover, DEX improves as the explored depths become finer, showing that parallel depth exploration provides a scalable way to exploit the underused depth axis of LLM decoding.

1 Introduction

The large parameter size of modern Large Language Models (LLMs) gives them strong capabilities across many domains, but also makes decoding expensive. A major source of this latency is autoregressive (AR) decoding: LLMs generate responses token by token, and each generated token must pass through the full layer stack before the next token can be produced. Recent studies [1, 2] identify redundancy along the depth dimension, suggesting that not all tokens require the full model depth to become predictable. Leveraging this observation, prior work [3–5] has proposed depth-adaptive decoding methods that allow tokens to exit from intermediate layers, reducing the amount of layer computation and improving decoding latency.

More recently, the draft-and-verify paradigm in speculative decoding [6] has been adopted by lossless depth-adaptive methods [7–9]. These methods draft candidates from non-final depths and verify them with the final-depth model, preserving equivalence with standard AR decoding while reducing layer computation. However, existing methods share a *selection bottleneck* that

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

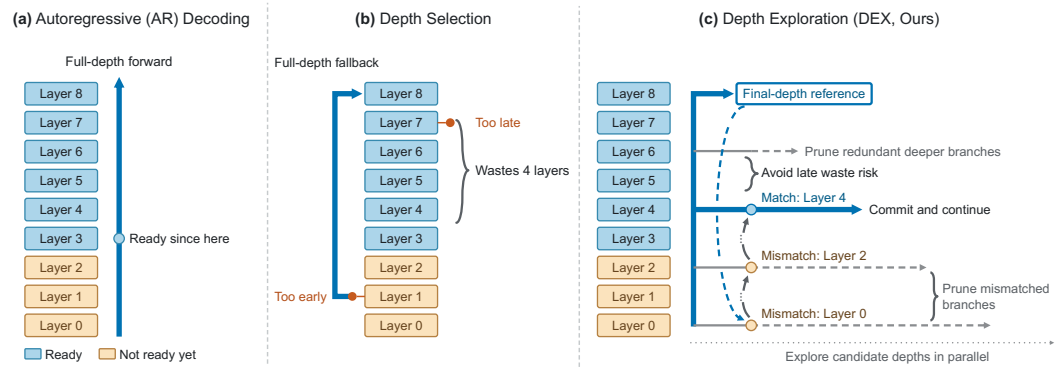

Figure 1: Comparison of different depth exploitation methods. (a) Autoregressive decoding always performs a full-depth forward pass. (b) Depth selection relies on a single exit-depth decision, risking either full-depth fallback when exiting too early or wasted computation when exiting too late. (c) Parallel depth exploration over candidate depths resolves branches with the final-depth reference and preserves only reusable branches.

limits their potential speedup: they choose only a single exit depth for each token, either fixed or through a heuristic or learned policy. Choosing too late wastes layers even when the token was already predictable earlier, while choosing too early can trigger final-depth replacement and discard descendant drafts. Our empirical studies confirm this gap between existing methods and an oracle that leverages the earliest intermediate depth whose prediction matches the full-depth output (Figure 3). This raises the key question:

Can we exploit depth redundancy more reliably?

We propose depth exploration (DEX), whose key idea is to replace single-depth prediction with multi-depth coverage. Instead of betting on one exit depth, DEX explores several candidate depths in parallel and uses the full-depth output as the reference to identify the earliest explored branch that matches it. A missed shallow candidate therefore does not immediately force a full-depth fallback: a deeper explored branch may still match and be reused. As a result, the error of depth adaptation is controlled by the spacing between explored depths rather than by the accuracy of a single exit-depth predictor. As the exploration set becomes finer, the first reusable branch moves closer to the earliest depth at which the token would have matched the full-depth output.

The main challenge is not merely exposing more exits, but turning parallel depth coverage into an exact decoding procedure. Exploration creates multiple possible futures, yet AR decoding has only one official prefix. After each commit, the decoder must identify which branch computations remain valid under the new prefix and which have become stale. DEX addresses this with an expand–commit–collapse procedure: it **expands** candidate branches across depth stages, **commits** only the final-depth-validated token, and **collapses** the exploration lattice onto the branch consistent with that commit (Section 3.2). This allows DEX to reuse parallel depth exploration when it agrees with the final model, while preserving lossless equivalence to standard AR decoding.

To conclude, our main contributions are:

- We define Earliest Available Depth (EAD) to quantify depth-side token readiness. Building on EAD, we formulate depth selection and depth exploration, and identify the selection bottleneck in existing lossless depth-adaptive decoding methods.
- We present DEX, a lossless speculative decoding algorithm that realizes depth exploration through parallel expansion, final-depth validation and commitment, and lattice collapse.
- We show that DEX achieves up to a 61% speedup over the strongest depth-selection baselines, with stronger scaling as more depth explorers are available.

2 Preliminaries and problem formulation

This section gives the compact formulation used by the method and experiments. Appendix A provides additional EAD measurements, illustrative examples, and full derivations.

2.1 Autoregressive decoding and token readiness

Consider a decoder-only language model with L layers that generates T new tokens. Standard autoregressive (AR) decoding will have LT layer computations. However, for many tokens, an intermediate layer may already produce a candidate that agrees with the final-depth decoding outcome. We use this agreement to define when a token becomes *ready* along the depth axis.

Definition 1 (Acceptance and stable earliest available depth)**.** Let $[L] = \{1, \ldots, L\}$ and $[T] = \{1, \ldots, T\}$. For token position $t \in [T]$ and depth $\ell \in [L]$, let $\Omega_{\mathcal{R}}(t, \ell) \in \{0, 1\}$ denote whether the candidate token produced at depth ℓ is accepted by the final-depth decoding rule $\mathcal{R}$. We set $\Omega_{\mathcal{R}}(t, L) = 1$ by construction. The earliest available depth is

$$\mathrm{EAD}_{\mathcal{R}}(t) = \min\{\ell \in [L] : \Omega_{\mathcal{R}}(t, \ell) = 1\}. \tag{1}$$

Because intermediate predictions can occasionally match and then diverge at later layers, we use a stable version as the default quantity. Define stable acceptance by

$$\Omega^{\mathrm{stab}}_{\mathcal{R}}(t, \ell) = 1 \quad \Longleftrightarrow \quad \Omega_{\mathcal{R}}(t, n) = 1, \ \forall n \in \{\ell, \ell + 1, \ldots, L\}. \tag{2}$$

The stable earliest available depth is

$$\mathrm{EAD}^{\mathrm{stab}}_{\mathcal{R}}(t) = \min\{\ell \in [L] : \Omega^{\mathrm{stab}}_{\mathcal{R}}(t, \ell) = 1\}. \tag{3}$$

Unless otherwise stated, EAD refers to this stable EAD and we omit the subscript $\mathcal{R}$ when the decoding rule is fixed.

2.2 Depth selection and its bottleneck

Existing lossless depth-adaptive methods typically instantiate depth adaptation as a selection problem: they choose one non-final depth, draft a token from that depth, and rely on final-depth verification to preserve equivalence with AR decoding.

Definition 2 (Depth selection)**.** A depth-selection policy assigns a selected non-final exit depth $s_t \in [L - 1]$ to each drafted token position t. Under the stable-readiness convention used in this analysis, the selected exit can exploit token readiness only when $\Omega^{\mathrm{stab}}(t, s_t) = 1$, equivalently $\mathrm{EAD}(t) \le s_t$.

Let p^{acc}_t denote the probability that the selected exit at s_t is accepted under this workload model. If this exit fails, final-depth verification must replace the token, and any drafted descendants that depend on the rejected token must be discarded. Let $v_t \ge 1$ be the number of discarded selected-exit tokens from position t onward, including token t. The expected layer cost charged to position t is

$$W^{\mathrm{sel}}_t = p^{\mathrm{acc}}_t s_t + (1 - p^{\mathrm{acc}}_t) \left(L + \sum_{i=t}^{t+v_t-1} s_i \right). \tag{4}$$

Proposition 3 (Selection overhead decomposition)**.** *Relative to an oracle that directly uses* $\mathrm{EAD}(t)$*, the expected overhead of depth selection is*

$$c^{\mathrm{sel}}_t = W^{\mathrm{sel}}_t - \mathrm{EAD}(t) = p^{\mathrm{acc}}_t (s_t - \mathrm{EAD}(t)) + (1 - p^{\mathrm{acc}}_t) \left(L + \sum_{i=t}^{t+v_t-1} s_i - \mathrm{EAD}(t) \right). \tag{5}$$

Proposition 3 exposes the selection bottleneck. The derivation and a numerical example are provided in Appendix A.2.

2.3 Depth exploration

We next formulate depth exploration as exposing a finite set of candidate depths and charging each token to the earliest explored depth that reaches stable readiness.

Definition 4 (Exploration set and resolution)**.** Let $\mathcal{X} = \{d_0 < d_1 < \cdots < d_{N-1} = L\} \subseteq [L]$ be the explored depth set. The final depth is always included so that full-depth decoding remains available. For any $\ell \in [L]$, define the depth ceiling

$$\lceil \ell \rceil_{\mathcal{X}} = \min\{d \in \mathcal{X} : d \geq \ell\}. \tag{6}$$

The resolution of $\mathcal{X}$ is

$$\Delta(\mathcal{X}) = \max_{\ell \in [L]} \left(\lceil \ell \rceil_{\mathcal{X}} - \ell \right), \tag{7}$$

which measures the largest upward rounding error caused by restricting usable depths to $\mathcal{X}$.

Definition 5 (Depth exploration)**.** Given $\mathcal{X}$, the stable depth-exploration workload for token t is the earliest explored depth that reaches the stable-readiness threshold:

$$W_{\mathcal{X}}(t) = \min\{d \in \mathcal{X} : \Omega^{\text{stab}}(t, d) = 1\}. \tag{8}$$

Equivalently,

$$W_{\mathcal{X}}(t) = \lceil \text{EAD}(t) \rceil_{\mathcal{X}}. \tag{9}$$

Proposition 6 (Resolution-bounded exploration overhead)**.** *For any token t and exploration set $\mathcal{X}$, the overhead of depth exploration relative to the EAD oracle is*

$$c_t^{\text{exp}} = W_{\mathcal{X}}(t) - \text{EAD}(t) = \lceil \text{EAD}(t) \rceil_{\mathcal{X}} - \text{EAD}(t), \tag{10}$$

so

$$0 \leq c_t^{\text{exp}} \leq \Delta(\mathcal{X}). \tag{11}$$

A proof and an example for uniformly spaced exploration sets are provided in Appendix A.3.

Depth-side speedup. We use the same layer-cost accounting to relate these depth quantities to theoretical speedup. This accounting ignores non-layer costs and assumes uniform per-layer cost; the experiments in Section 4 report measured walltime speed.

Proposition 7 (Depth-side speedup and monotonicity)**.** *For any decoding algorithm with charged layer cost $W(t)$ for token t, its depth-side speedup over AR decoding is*

$$S = \frac{LT}{\sum_{t=1}^{T} W(t)}. \tag{12}$$

For depth exploration,

$$S_{\mathcal{X}} = \frac{LT}{\sum_{t=1}^{T} W_{\mathcal{X}}(t)} = \frac{LT}{\sum_{t=1}^{T} \lceil \text{EAD}(t) \rceil_{\mathcal{X}}}. \tag{13}$$

By Proposition 6,

$$1 \leq \frac{LT}{\sum_{t=1}^{T} \min\{\text{EAD}(t) + \Delta(\mathcal{X}), L\}} \leq S_{\mathcal{X}} \leq \frac{LT}{\sum_{t=1}^{T} \text{EAD}(t)} = S_{\text{EAD}}. \tag{14}$$

Here S_{EAD} is the oracle speedup obtained if every token's EAD were known and directly usable. Moreover, if $\mathcal{X} \subseteq \mathcal{X}'$, then $S_{\mathcal{X}} \leq S_{\mathcal{X}'} \leq S_{\text{EAD}}$; finer exploration monotonically approaches the EAD oracle under this depth-side accounting.

The derivation of Eq. (14) and the monotonicity statement is provided in Appendix A.4. While this analysis characterizes the ideal depth-side opportunity of exploration, Section 3.2 revisits this under a more realistic setting. The next section instantiates this idealized exploration model as an exact decoding algorithm and discusses the additional walltime costs introduced by branch execution.

Algorithm 1 Depth Exploration Decoding (DEX)

Input: Prompt tokens $\mathbf{x}$; exploration set $\mathcal{X} = \{d_0, \ldots, d_{N-1}\}$, where d_{N-1} is the final depth; stages $\Pi_k = \text{Layers}(d_{k-1}, d_k]$ with $d_{-1} = 0$; max token length T
Output: Generated tokens Y

$E_k \leftarrow (\Pi_k, \text{KV}_k)$, for $k = 0, \ldots, N-1$ ▷ depth explorer
$Y \leftarrow [\,]$, $B \leftarrow \emptyset$, $q \leftarrow 0$ ▷ B: explore buffer; q: FSM state
$S \leftarrow \text{init}(\mathbf{x}, \{E_k\})$ ▷ S: hidden states and per-stage KV views
while $|Y| < T$ **do**
  $H_q \leftarrow \text{expand}(\{E_k\}, S, q)$ ▷ parallel lattice expansion; Section 3.2
  $Z_q \leftarrow \text{DECS}(E_0, H_q, B, q)$ ▷ depth-coupled sampling; Section 3.3
  $B \leftarrow \text{insert}(B, Z_q)$
  if ReachFinalDepth(q) **then**
    $(b, y) \leftarrow \text{validate_and_commit}(B)$ ▷ b: accepted branch; y: committed token
    $Y \leftarrow Y \| y$
    if stop(Y) **then**
      return Y
    end if
    $(B, S, q) \leftarrow \text{collapse}(B, S, q, b)$ ▷ post-commit lattice collapse; Section 3.2
  else
    $q \leftarrow \text{advance}(q)$ ▷ advance FSM state; Section 3.2
  end if
end while
return Y

3 Method

3.1 Algorithm overview

We now describe DEX. Conceptually, DEX maps each depth stage $\Pi_k = \text{Layers}(d_{k-1}, d_k]$ to a depth explorer E_k, with each explorer deployed on a separate hardware computing unit. The output boundary of E_k corresponds to candidate depth $d_k \in \mathcal{X}$. Each explorer processes one depth stage and exposes candidates at its boundary. DEX commits the final-depth token, selects the earliest matching branch for reuse, and collapses stale branches. This resembles the principle of carry-lookahead and Kogge-Stone adders [10, 11], where parallel candidate computations expose a sequential dependency earlier than a purely serial execution. The whole execution procedure is demonstrated in Algorithm 1.

3.2 Lattice expansion and commit–collapse

Lattice expansion. To correctly manage candidate sequences generated by different depth explorers, we organize each exploration cycle around a *commit position* p_c, i.e., the next token position to be appended to the output sequence. In Figure 2 (left), this is the row marked as position 0. We represent depth exploration as a depth–position lattice unrolled over explorer rounds, using four explorers $E_0 \sim E_3$, each covering one quarter of the model depth. Each circle denotes an exploration token ETK_i, where i is its token id in the explore buffer.

At each round, an exploration token can expand in two directions. A depth expansion keeps the same token position and forwards its hidden state to the next explorer. A position expansion uses the current candidate token as the prefix for the next token position and starts that position from E_0. Each position expansion creates a new *sampling slot* for the next token position, i.e., a branch-conditioned next-token determined by its prefix history and target token position. The ETKs that instantiate the same sampling slot at different depths form a *slot chain*. In particular, the slot chain for the current commit position under the committed context is the *anchor path*. For example, E_0 first produces ETK_0 for p_c; in the next round, ETK_0's hidden state continues through E_1 to produce ETK_1 in the same slot chain, while ETK_0 is also used as a prefix to create a new slot whose first token is ETK_2. Subsequently, ETK_1 also expands in position to produce ETK_4. Although ETK_2 and ETK_4 share the same token position, they belong to different sampling slots because they are conditioned on different candidate prefixes.

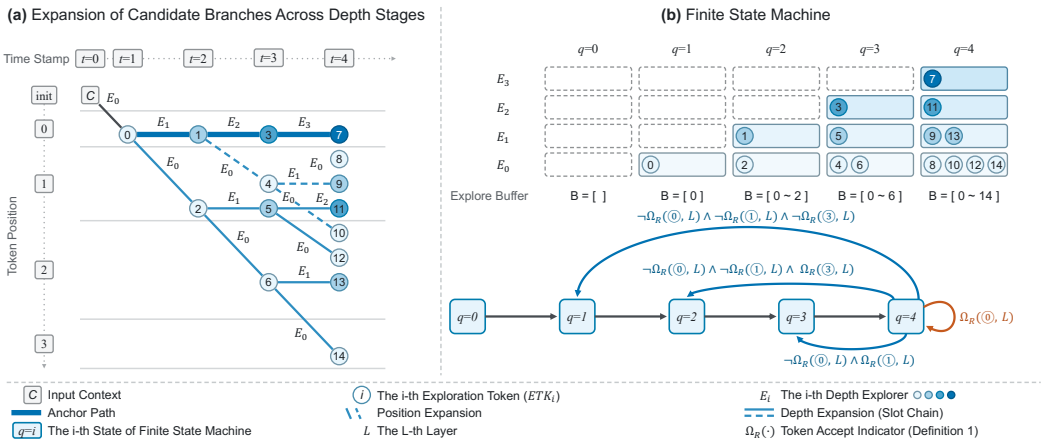

Figure 2: DEX execution with four depth explorers. (a) Candidate branches are unrolled on a depth–position lattice, where the anchor path reaches the final depth while descendant branches are precomputed for potential reuse. (b) The FSM records the explorer round, materialized buffer entries, and collapse transition after the final-depth reference determines the committed token.

After four explorer rounds, the anchor path reaches the final-depth node, which provides the reference token for the commit position. Other slot chains correspond to speculative descendants under branch-specific prefixes. Before collapse, the lattice follows a binary expansion structure, allowing ancestry, KV entries, and attention masks to be indexed consistently.

Commit process. Once the anchor path reaches the final depth, DEX uses its final-depth token as the commit reference. Let $(b_0, u_0), \ldots, (b_{N-1}, u_{N-1})$ denote the anchor-path branches and their candidate tokens at p_c, ordered by increasing depth, where u_{N-1} is the final-depth reference. DEX checks whether u_{N-1} appears among the non-final anchor candidates $\{u_0, \ldots, u_{N-2}\}$; if so, it selects the shallowest matching branch b_k, and otherwise falls back to the final-depth branch b_{N-1}. In both cases, the token appended to the official output is $y = u_{N-1}$, while the selected branch b determines which part of the lattice can be preserved during collapse.

Finite-state control and collapse. DEX employs a finite-state machine (FSM) to govern branch lifecycles during depth exploration, as shown in Figure 2 (right). A state q records how many explorer rounds have been completed since the last committed token and determines whether the system should continue expansion or perform commitment. For each state, the figure shows the ETKs produced after forward execution, the resulting explore buffer B, and the per-explorer KV-cache views, where C denotes committed context tokens. Under the four-explorer setup, state q materializes the first $2^q - 1$ tokens in the binary expansion tree, e.g., $B = \{\mathrm{ETK}_0, \ldots, \mathrm{ETK}_{14}\}$ at state 4.

Lossless invariant. DEX is lossless with respect to standard autoregressive decoding because commitment is always defined by the final-depth reference. At every commit position, the committed prefix and the final-depth KV state maintained by DEX are identical to those of standard autoregressive decoding under the same decoding rule. The final-depth anchor node is computed from this committed prefix using the unmodified backbone model. DEX appends exactly the token produced by this node: if an earlier anchor candidate matches the final-depth token, the earlier branch only determines how much of the lattice can be reused; the committed token itself remains the final-depth token. During collapse, DEX preserves only nodes whose prefix equals the new committed prefix and whose per-stage hidden states and KV entries were computed under that prefix; all inconsistent nodes are discarded or recomputed. Therefore, by induction over commit positions, DEX produces the same token sequence and maintains the same final-depth state as standard autoregressive decoding. For sampling, the final-depth reference is sampled from unmasked final-depth logits using the target random source, so the committed sample distribution is unchanged.

Proposition 7 captures the ideal depth-side opportunity. In practice, DEX also pays branch-execution, communication, KV-management, and collapse overheads; Appendix B profiles these costs and shows that commit–collapse overhead is small relative to expansion.

3.3 Optimized depth-coupled sampling

If candidates in each slot chain are generated independently, different depths may repeatedly propose the same token. Such duplicates are unhelpful for DEX: if a shallower proposal v already matches the final-depth reference, the shallower branch would be selected; if it does not match, repeating v only reduces the opportunity to expose another candidate that may hit the reference.

To reduce duplicate proposals, DEX uses Depth-Coupled Sampling (DECS). Within each slot chain, DECS masks tokens already proposed by shallower depths when decoding later non-final candidates, encouraging deeper explorers to cover alternatives. The final-depth reference is never masked; in sampling mode, it is decoded from unmasked logits with the slot-chain Gumbel source, so the Gumbel-Max trick [12] preserves the exact target sample. DECS therefore only diversifies intermediate proposals while preserving lossless commitment. The full algorithm is given in Appendix C.1.

3.4 Adapt to standard LLMs with inducing adapters

By Proposition 7, the speedup of DEX is governed by the EAD distribution. Therefore, DEX can be directly applied to EAD-favorable LLMs, where most tokens already have shallow EADs. In standard LLMs, however, shallow and middle-layer hidden states are often not well aligned with the shared LM head, causing stable EADs to concentrate near the final depth. Motivated by prior work showing that early-exit behavior can be induced through pretraining or parameter-efficient tuning [5, 7, 8], we attach small residual adapters to these middle candidate depths to align intermediate hidden states with the shared LM head. These adapters are trained by self-distillation from the frozen final-depth model and affect only intermediate proposals; final-depth commitment remains unchanged. Details are in Appendix C.2.

4 Experiment

LLMs and datasets. We evaluate DEX on both early-exit-trained and standard LLMs. For depth-axis analysis, we use two LayerSkip-trained models, CodeLlama-34B and Llama-2-70B, denoted as LS-CodeLlama-34B and LS-Llama-2-70B. These models expose intermediate-layer predictions and are therefore suitable for measuring EAD, oracle headroom, and the gap between DEX and depth-selection baselines. We also evaluate standard CodeLlama-34B-Instruct [13], Llama-2-70B-Instruct [14], and Qwen3-32B [15] to study DEX's speedup under regular model settings and compare it with broader decoding accelerators. We use GSM8K for mathematical reasoning [16], HumanEval for code generation [17], and XSum for summarization [18].

Baselines and setup. We organize baselines into two groups. First, for depth-axis comparison, we compare DEX with representative depth-selection methods: LayerSkip self-speculative decoding (LSSD) [7], AdaDecode [8], and DEL [9]. Second, for broader comparison with other acceleration paradigms, we include autoregressive decoding (AR), tensor-parallel decoding (TP), Lookahead [19], AdaDecode [8], PEARL [20], and the EAGLE series [21, 22]. All methods are evaluated on NVIDIA H100 GPUs. TP is tested with tensor-parallel sizes 2, 4, and 8, and we report its best speed. LSSD/AdaDecode/DEL/Lookahead/PEARL/EAGLE use 3 GPUs for 32B/34B models and 4 GPUs for 70B models. DEX is evaluated under multiple uniformly partitioned exploration resolutions, denoted as DEX-$1/K$, where K is the number of depth explorers, each running on one GPU. For example, DEX-$1/3$ uses three explorers and adopts a granularity of one third of the full model depth. Unless specified, we use greedy decoding with batch size 1 and maximum generation length 512 for Llama-family models and 1024 for Qwen3-32B. Additional details and numeric results are provided in Appendix E and Appendix D.

4.1 Main results

Speed comparison. Figure 3 compares DEX with representative depth-selection methods on LayerSkip-trained models, where non-final-layer predictions are explicitly exposed. DEX achieves higher measured speed than LayerSkip, AdaDecode, and DEL across both models and all datasets. DEX achieves 20% and 38% higher overall speed than the strongest depth-selection baselines on LS-CodeLlama-34B and LS-Llama-2-70B, respectively, with up to a 61% speedup on GSM8K. The dashed oracle line denotes the speed implied by stable EAD measurements, serving as a depth-side

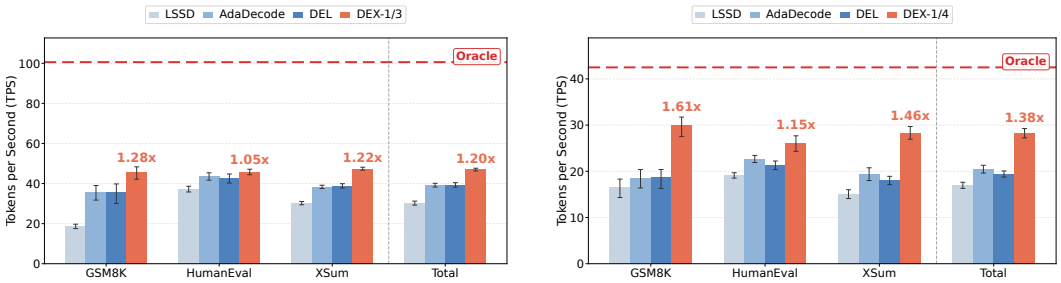

Figure 3: Depth-axis speed comparison on LS-CodeLlama-34B (left) and LS-Llama-2-70B (right). DEX outperforms depth-selection baselines across datasets, and the dashed line marks the oracle speed implied by EAD headroom.

headroom reference rather than an implemented baseline. The remaining oracle gap indicates further depth-side headroom, but DEX realizes a larger fraction of it than the selection baselines.

Scaling with exploration resolution. Figure 4 (left) shows that finer exploration generally improves speed. This trend is consistent with the formulation in Section 2: increasing the number of explored depths lets DEX track token readiness with a smaller upward rounding error. The gain is especially clear on LS-Llama-2-70B, where increasing the exploration resolution substantially narrows the gap to the EAD-implied oracle. This indicates that exploration resolution acts as an effective control knob for converting additional depth explorers into realized speed.

Empirical losslessness check. As established in Section 3.2, DEX is lossless with respect to standard autoregressive (AR) decoding. We further verify this property empirically on LS-CodeLlama-34B and standard CodeLlama-34B across all three benchmarks. Under greedy decoding, DEX achieves a 100% per-token match with AR decoding on all evaluated outputs.

4.2 Ablation study

Effectiveness of depth exploration. Figure 4 (right-top) isolates the effect of exploration by comparing DEX with a single-exit variant that disables intermediate depth branches except for the first and final depth candidates. At finer exploration resolutions, full exploration achieves up to $1.71\times$ the speed of the single-exit variant. At coarse resolutions, the two variants are close; at finer resolutions, full exploration gives a much larger gain. This suggests that the benefit comes not merely from running depth stages in parallel, but from exposing multiple candidate depths and committing the earliest branch accepted by the final-depth reference.

Effectiveness of Depth-Coupled Sampling. Figure 4 (right-bottom) evaluates the speedup effect of DECS. Compared with vanilla independent decoding, DECS yields a consistent speedup under both greedy decoding and temperature sampling, showing that this by-construction diversification translates into practical decoding speedups.

4.3 Generalization and scaling

To evaluate the applicability of depth exploration beyond early-exit-trained models, we apply DEX to standard LLMs with inducing adapters and compare its decoding speed against representative speculative and distributed decoding baselines. We examine both its competitiveness under matched GPU budgets and its scaling behavior as additional depth explorers become available.

Figure 5 shows that, under matched GPU budgets, DEX achieves overall speed within 5% of PEARL across CodeLlama-34B, Llama-2-70B, and Qwen3-32B. EAGLE achieves higher speed in the matched Llama-2-70B and Qwen3-32B settings. Appendix F further compares the auxiliary training costs of DEX and EAGLE-3, highlighting the training-efficiency advantages of DEX.

With additional depth explorers, DEX achieves higher speed gains, reaching $2.90\times$, $3.68\times$, and $1.80\times$ speedups over AR on CodeLlama-34B, Llama-2-70B, and Qwen3-32B, respectively, using 6, 8, and 8 GPUs. These results demonstrate the scaling potential of depth exploration and its ability to translate

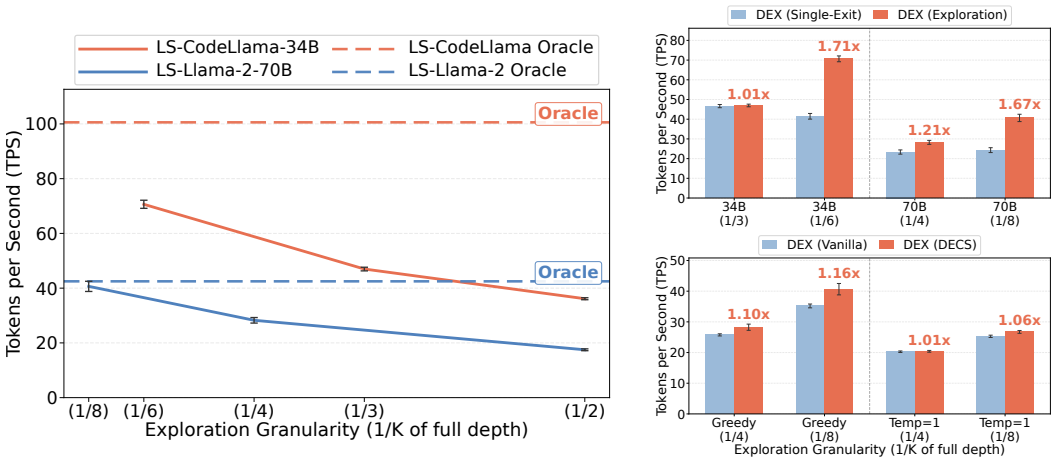

Figure 4: Scaling behavior and architectural analysis of DEX. Left: speed under different exploration resolutions. Right-top: comparison between single-exit decoding and depth exploration. Right-bottom: comparison between vanilla decoding and DECS-enhanced exploration in both greedy and sampling (Temperature=1.0) cases.

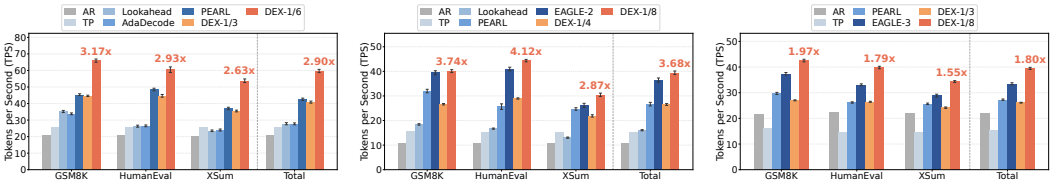

Figure 5: Broader speed comparison across CodeLlama-34B, Llama-2-70B, and Qwen3-32B. With additional GPUs enabling finer depth exploration, scaled DEX achieves the highest measured speed among the evaluated configurations.

additional computing resources into higher single-stream decoding speed. Section 6 provides a closer look at the resource-to-resolution tradeoff.

5 Related work

Depth-axis LLM acceleration. The depth axis asks how early the next token becomes determined before the final layer. Mixture-of-Depths exploits this redundancy through learned layer routing, but changes the model's conditional-compute architecture rather than providing a lossless wrapper for an existing full-depth decoder [23]. For decoding, LayerSkip trains intermediate layers for early prediction, while Draft & Verify, SWIFT, and PPSD-style methods use skipped, partial-depth, or pipelined executions as self-drafters followed by final-depth verification [7, 24–26]. AdaDecode, DEL, and PELM further make this path adaptive, by overlapping intermediate-layer prediction with deferred deeper computation or by adjusting the exit layer and speculation length [8, 9, 27]. These methods still try one chosen early-exit, skip, or pipeline path in a speculative round. However, our proposed DEX instead exposes multiple candidate depths for the same commit position and reuses the shallowest branch certified by the final-depth reference.

Token-axis LLM acceleration. The token axis asks how many future token positions can be resolved from one target-model verification. Speculative decoding uses a draft model to propose a block of tokens and the target model to verify it [6, 28, 29]. Medusa and the EAGLE family improve the draft side with extra prediction heads or feature-level drafts [30, 31, 21, 22]. Lookahead Decoding avoids a separate draft model by using the target model itself to find and verify short candidate continuations [19]. PEARL and Speculative Speculative Decoding focus on the scheduling of the draft–verify loop, reducing idle time through adaptive draft lengths or predicted verification outcomes [20, 32]. These methods speed up decoding by accepting more future tokens per target-model step. In contrast, DEX

exploits depth-side redundancy by exploring multiple intermediate depths for each commit position and reusing states from branches validated against the final-depth reference.

Parallel execution for LLM decoding. Another direction improves decoding speed by better using parallel hardware. Tensor and pipeline parallelism partition full-depth model execution across devices, while decoding-specific systems such as EasySpec and PPSD-style methods overlap parts of the draft–verify pipeline or parallelize layer execution [33, 34, 29, 26]. DEX also uses multiple devices, but the devices are not merely shards of one full-depth pass or one draft–verify pipeline. Instead, they act as depth explorers that expose competing candidate depths, after which the final-depth reference commits the token and collapses the exploration lattice.

6 Discussion and limitations

Hardware-agnostic formulation and resource-dependent realization. Depth exploration separates the algorithmic question of *which depths to expose* from the systems question of *how to execute them*. The exploration set $\mathcal{X}$, the rounded workload $W_{\mathcal{X}}(t) = \lceil \mathrm{EAD}(t) \rceil_{\mathcal{X}}$, and the resolution bound in Proposition 7 characterize, under idealized layer-work accounting, how a finite set of candidate depths approximates the EAD oracle. This formulation does not prescribe a hardware schedule: the same lattice could be evaluated sequentially, or with limited opportunistic overlap, on a single device, but such schedules mainly expose additional candidate states and are not expected to provide wall-clock speedup without sufficient parallel execution resources.

DEX is one distributed realization of this formulation. To convert depth exploration into wall-clock speedup, our current implementation assigns each explored depth boundary to a depth explorer and places each explorer on a separate GPU. Thus, a DEX-1/K configuration uses K depth explorers and K GPUs. Additional GPUs instantiate finer exploration resolution rather than serving as hidden resources. We therefore report GPU counts explicitly and separate matched-resource comparisons from scaling results. For resource-aware comparison, we evaluate tensor parallelism with 2, 4, and 8 GPUs and PEARL under its distributed configuration, using 3 GPUs for 32B/34B models and 4 GPUs for the 70B model. Under matched GPU counts, DEX is competitive with PEARL on 34B and 70B models, while higher-K configurations evaluate how additional devices translate into finer exploration resolution and higher wall-clock speed.

Expansion growth and overhead. As described in Section 3.2, DEX expands a depth–position lattice during each exploration cycle. With K depth explorers, a full expansion cycle can materialize up to $2^K - 1$ exploration tokens, and the largest token batch processed by one explorer in our 8-explorer configuration reaches $2^7 = 128$ tokens. This growth is the main practical cost of finer exploration. While Proposition 7 shows that finer exploration reduces the idealized depth-side rounding error, the realized wall-clock gain also depends on explorer forward cost, memory traffic, attention-mask construction, communication, and branch-state management. Our current scaling study evaluates up to 8 explorers, where DEX continues to improve in the tested settings. We do not extrapolate this monotonic wall-clock behavior beyond the evaluated range: at larger K, marginal gains may saturate as lattice expansion and compute-bound overheads become more prominent. A promising direction is to study how to prune low-value branches or construct sparse exploration lattices while preserving the final-depth commit invariant.

Additional discussions are provided in Appendix F.

7 Conclusion

By formalizing token readiness through EAD, we showed that selection-based methods are limited by the need to choose one exit depth under uncertainty, whereas exploration converts this uncertainty into a resolution-controlled upward rounding problem over candidate depths. Building upon that, we presented DEX, a lossless decoding algorithm that realizes this formulation through an expand–commit–collapse procedure. On early-exit-trained LLMs, DEX consistently outperforms representative depth-selection methods under matched GPU budgets, realizing more of the EAD-implied depth-side headroom. On standard LLMs, DEX demonstrates broader applicability and achieves substantial end-to-end decoding speedup as more depth explorers become available. Together, these results establish parallel depth exploration as a practical, lossless, and scalable approach to exploiting depth-side redundancy in autoregressive LLM decoding.

8 Acknowledgements

This research was supported in part through the computational resources and staff contributions provided for the Quest high performance computing facility at Northwestern University which is jointly supported by the Office of the Provost, the Office for Research, and Northwestern University Information Technology. We also thank Lambda for providing computational resources that supported this research.

References

[1] Daria Lioubashevski, Tomer M. Schlank, Gabriel Stanovsky, and Ariel Goldstein. Looking beyond the top-1: Transformers determine top tokens in order. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=2B11W1Z6ID.

[2] Róbert Csordás, Christopher D Manning, and Christopher Potts. Do language models use their depth efficiently? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Kz6eUL86XP.

[3] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJg7KhVKPH.

[4] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=uLYc4L3C81A.

[5] Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. EE-LLM: Large-scale training and inference of early-exit large language models with 3d parallelism. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=xFk0w9zoV3.

[6] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[7] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12622–12642, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.681. URL https://aclanthology.org/2024.acl-long.681/.

[8] Zhepei Wei, Wei-Lin Chen, Xinyu Zhu, and Yu Meng. Adadecode: Accelerating llm decoding with adaptive layer parallelism. *arXiv preprint arXiv:2506.03700*, 2025.

[9] Hossein Entezari Zarch, Lei Gao, Chaoyi Jiang, and Murali Annavaram. DEL: Context-aware dynamic exit layer for efficient self-speculative decoding. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=cAFxSuXQvT.

[10] A. Weinberger and J. L. Smith. A logic for high-speed addition. *National Bureau of Standards Circular*, 591:3–12, 1958.

[11] Peter M Kogge and Harold S Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE transactions on computers*, 100(8):786–793, 1973.

[12] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

[13] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024. URL https://arxiv.org/abs/2308.12950.

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

[15] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

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

[17] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

[18] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *ArXiv*, abs/1808.08745, 2018.

[19] 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*, ICML'24. JMLR.org, 2024.

[20] Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. PEARL: Parallel speculative decoding with adaptive draft length. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=QOXrVMiHGK.

[21] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Empirical Methods in Natural Language Processing*, 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 *Annual Conference on Neural Information Processing Systems*, 2025.

[23] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models, 2024. URL https://arxiv.org/abs/2404.02258.

[24] Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft& verify: Lossless large language model acceleration via self-speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 11263–11282. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.607. URL https://doi.org/10.18653/v1/2024.acl-long.607.

[25] Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. Swift: On-the-fly self-speculative decoding for llm inference acceleration, 2025. URL https://arxiv.org/abs/2410.06916.

[26] Ruanjun Li, Ziheng Liu, Yuanming Shi, Jiawei Shao, Chi Zhang, and Xuelong Li. Optimized early-exit based speculative decoding via pipeline parallelism, 2026. URL https://openreview.net/forum?id=6ezbdRe90k.

[27] Weisi Yang and Stephen Xia. Pelm: Power efficient on-device llm inference with speculative decoding and dynamic voltage frequency scaling. In *Proceedings of the 2026 ACM/IEEE International Conference on Embedded Artificial Intelligence and Sensing Systems*, SenSys '26, page 438–451, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400723094. doi: 10.1145/3774906.3802783. URL https://doi.org/10.1145/3774906.3802783.

[28] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

[29] Yize Wu, Ke Gao, Ling Li, and Yanjun Wu. Easyspec: Layer-parallel speculative decoding for efficient multi-gpu utilization, 2025. URL https://arxiv.org/abs/2502.02493.

[30] 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, 2024. URL https://arxiv.org/abs/2401.10774.

[31] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty, 2025. URL https://arxiv.org/abs/2401.15077.

[32] Tanishq Kumar, Tri Dao, and Avner May. Speculative speculative decoding, 2026. URL https://arxiv.org/abs/2603.03251.

[33] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020. URL https://arxiv.org/abs/1909.08053.

[34] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism, 2019. URL https://arxiv.org/abs/1811.06965.

[35] Aeala. Sharegpt_vicuna_unfiltered, 2023. URL https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered.

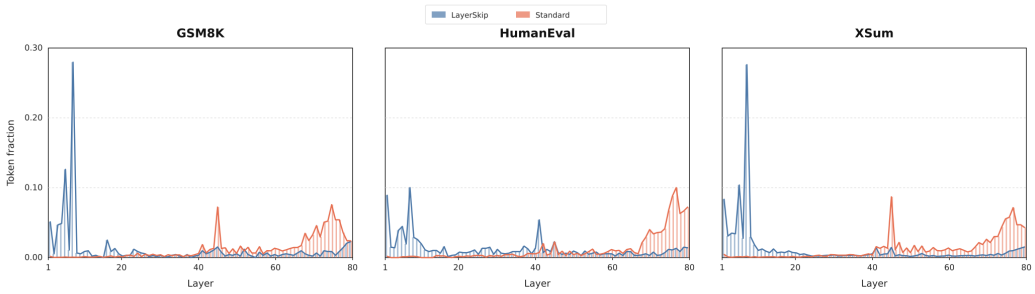

Figure 6: Distribution of stable EAD across different datasets on Llama2-70B.

A Additional formulation details and proofs

Section 2 presents the compact formulation used in the main paper. This appendix provides additional empirical measurements, illustrative examples, and full derivations for the selection and exploration results stated in the main text. We use the same notation, stable-EAD convention, and idealized layer-work accounting as in Section 2.

A.1 Additional EAD measurements

Using the stable EAD definition from Section 2, we measure token readiness on three datasets using two Llama2-70B variants: the standard model [14] and a LayerSkip model trained with early-exit supervision [7]. For visualization, we randomly sample 36 prompts per dataset and plot the EAD distributions in Figure 6.

Observation. The standard model has EADs concentrated near the final layer, while LayerSkip shows many tokens becoming stable at substantially earlier layers. This suggests that depth-side token readiness can be induced by early-exit-oriented training. The measurement motivates the adapter design in Section 3.4: adapters improve intermediate proposals by making non-final depths more likely to match the final-depth outcome, thereby shifting observed EADs earlier. The same measurement also motivates the decoding question addressed by DEX: given early-ready tokens, how should a lossless decoder exploit them without changing the final-depth decoding rule?

A.2 Proof and example for selection overhead

Proof of Proposition 3. By the workload model in Section 2, the expected layer work charged to position t under depth selection is

$$W_t^{\mathrm{sel}} = p_t^{\mathrm{acc}} s_t + (1 - p_t^{\mathrm{acc}}) \left(L + \sum_{i=t}^{t+v_t-1} s_i \right). \tag{15}$$

The EAD oracle would charge $\mathrm{EAD}(t)$ layers to the same token. Subtracting this oracle workload gives

$$\begin{aligned} c_t^{\mathrm{sel}} &= W_t^{\mathrm{sel}} - \mathrm{EAD}(t) && (16)\\ &= p_t^{\mathrm{acc}} s_t + (1 - p_t^{\mathrm{acc}}) \left(L + \sum_{i=t}^{t+v_t-1} s_i \right) - \mathrm{EAD}(t) && (17)\\ &= p_t^{\mathrm{acc}} (s_t - \mathrm{EAD}(t)) + (1 - p_t^{\mathrm{acc}}) \left(L + \sum_{i=t}^{t+v_t-1} s_i - \mathrm{EAD}(t) \right), && (18) \end{aligned}$$

where the last step uses $p_t^{\mathrm{acc}} + (1 - p_t^{\mathrm{acc}}) = 1$. This is exactly the claimed decomposition. The first term is the overhead incurred when the selected exit succeeds, and the second term is the overhead incurred when the selected exit fails and descendant drafts are discarded.

Example. Consider an $L = 20$ layer model and a token with $\mathrm{EAD}(t) = 8$. A conservative selection $s_t = 15$ reaches the token's stable-readiness depth, but it still pays $15 - 8 = 7$ extra layers beyond the EAD oracle. In contrast, an aggressive selection $s_t = 4$ is earlier than the stable-readiness depth. If this exit fails and two descendant selected-exit tokens s_{t+1} and s_{t+2} are discarded, the failed branch charges

$$20 + 4 + s_{t+1} + s_{t+2} = 24 + s_{t+1} + s_{t+2} \tag{19}$$

layer computations to position t under the same accounting. This can exceed the 20-layer cost of standard AR decoding for that position. The example illustrates why a single underestimated exit depth can turn an otherwise early-ready token into a full-depth fallback with additional rollback cost.

A.3 Proof and example for exploration overhead

Proof of Proposition 6. By Definition 5, depth exploration charges token t to the first explored depth that reaches the stable-readiness threshold:

$$W_{\mathcal{X}}(t) = \lceil \mathrm{EAD}(t) \rceil_{\mathcal{X}} . \tag{20}$$

Therefore

$$c_t^{\mathrm{exp}} = W_{\mathcal{X}}(t) - \mathrm{EAD}(t) = \lceil \mathrm{EAD}(t) \rceil_{\mathcal{X}} - \mathrm{EAD}(t). \tag{21}$$

The depth ceiling always returns an explored depth no earlier than its input, so $c_t^{\mathrm{exp}} \geq 0$. By Definition 4, $\Delta(\mathcal{X})$ is the largest possible upward rounding error over all depths in $[L]$. Since $\mathrm{EAD}(t) \in [L]$, we have

$$\lceil \mathrm{EAD}(t) \rceil_{\mathcal{X}} - \mathrm{EAD}(t) \leq \Delta(\mathcal{X}). \tag{22}$$

Thus $0 \leq c_t^{\mathrm{exp}} \leq \Delta(\mathcal{X})$.

Example. Using the same $L = 20$ model and a token with $\mathrm{EAD}(t) = 8$, consider the exploration set $\mathcal{X} = \{5, 10, 15, 20\}$. Then

$$\lceil \mathrm{EAD}(t) \rceil_{\mathcal{X}} = \lceil 8 \rceil_{\mathcal{X}} = 10. \tag{23}$$

The earliest explored stable-ready branch is therefore the branch at depth 10, which incurs only $10 - 8 = 2$ additional layers beyond the EAD oracle. The official output token remains the final-depth reference token; the selected explored branch only determines how much branch state can be preserved during lattice collapse. For this exploration set, the largest upward rounding error is $\Delta(\mathcal{X}) = 4$, i.e., less than the adjacent spacing of 5 layers. This deterministic rounding behavior contrasts with depth selection, where an underestimated exit depth can trigger full-depth fallback and rollback.

A.4 Depth-side speedup derivation and monotonicity

Proof of Proposition 7. Standard AR decoding executes L layers for each of T generated tokens, so its total layer workload is LT. If a compared algorithm charges $W(t)$ layers to token t, its total charged layer workload is $\sum_{t=1}^{T} W(t)$. The corresponding depth-side speedup is therefore

$$S = \frac{LT}{\sum_{t=1}^{T} W(t)}, \tag{24}$$

which gives Eq. (12).

For depth exploration, substituting $W_{\mathcal{X}}(t) = \lceil \mathrm{EAD}(t) \rceil_{\mathcal{X}}$ gives

$$S_{\mathcal{X}} = \frac{LT}{\sum_{t=1}^{T} W_{\mathcal{X}}(t)} = \frac{LT}{\sum_{t=1}^{T} \lceil \mathrm{EAD}(t) \rceil_{\mathcal{X}}}, \tag{25}$$

which gives Eq. (13). Proposition 6 implies

$$\mathrm{EAD}(t) \leq \lceil \mathrm{EAD}(t) \rceil_{\mathcal{X}} \leq \min\{\mathrm{EAD}(t) + \Delta(\mathcal{X}), L\}. \tag{26}$$

Summing over tokens gives

$$\sum_{t=1}^{T} \mathrm{EAD}(t) \leq \sum_{t=1}^{T} \lceil \mathrm{EAD}(t) \rceil_{\mathcal{X}} \leq \sum_{t=1}^{T} \min\{\mathrm{EAD}(t) + \Delta(\mathcal{X}), L\}. \tag{27}$$

Table 1: Overall runtime profile of DEX on the 8-explorer 70B run. The table aggregates 160 DEX steps across 8 explorers/GPUs, giving 1280 rank-step samples. Shares are normalized over the listed profiled components.

Component	Avg. time / rank-step (ms)	Share of profiled time
Expansion	13.830	89.13%
Communication	1.222	7.88%
Collapse	0.309	1.99%
Decode/commit	0.156	1.01%
Total profiled	15.517	100.00%

All terms are positive, so taking reciprocals reverses the inequalities. Multiplying by LT yields

$$\frac{LT}{\sum_{t=1}^{T} \min\{\mathrm{EAD}(t) + \Delta(\mathcal{X}), L\}} \le S_{\mathcal{X}} \le \frac{LT}{\sum_{t=1}^{T} \mathrm{EAD}(t)} = S_{\mathrm{EAD}}. \tag{28}$$

Since $\min\{\mathrm{EAD}(t) + \Delta(\mathcal{X}), L\} \le L$ for every t, the leftmost denominator is at most LT, so

$$1 \le \frac{LT}{\sum_{t=1}^{T} \min\{\mathrm{EAD}(t) + \Delta(\mathcal{X}), L\}}. \tag{29}$$

Combining these inequalities gives Eq. (14).

It remains to prove monotonicity. If $\mathcal{X} \subseteq \mathcal{X}'$, then for every $\ell \in [L]$,

$$\lceil \ell \rceil_{\mathcal{X}'} \le \lceil \ell \rceil_{\mathcal{X}}, \tag{30}$$

because the finer set $\mathcal{X}'$ has all depths available in $\mathcal{X}$ and possibly more. Therefore,

$$\sum_{t=1}^{T} \lceil \mathrm{EAD}(t) \rceil_{\mathcal{X}'} \le \sum_{t=1}^{T} \lceil \mathrm{EAD}(t) \rceil_{\mathcal{X}}. \tag{31}$$

Taking reciprocals and multiplying by LT gives $S_{\mathcal{X}} \le S_{\mathcal{X}'}$. Since $\lceil \mathrm{EAD}(t) \rceil_{\mathcal{X}'} \ge \mathrm{EAD}(t)$ for every token, $S_{\mathcal{X}'} \le S_{\mathrm{EAD}}$. Hence $S_{\mathcal{X}} \le S_{\mathcal{X}'} \le S_{\mathrm{EAD}}$. Equality with the EAD oracle holds when $\mathcal{X} = [L]$.

Selection speedup for completeness. Substituting the selection workload from Proposition 3 into Eq. (12) gives

$$S_{\mathrm{sel}} = \frac{LT}{\sum_{t=1}^{T} \left[p_t^{\mathrm{acc}} s_t + (1 - p_t^{\mathrm{acc}}) \left(L + \sum_{i=t}^{t+v_t-1} s_i \right) \right]}. \tag{32}$$

This expression is not guaranteed to exceed one: failed selections increase the denominator through full-depth replacement and discarded descendant work. This is the speedup-side view of the selection bottleneck discussed in Section 2.

Together, these details support the main formulation. Depth selection compresses token-readiness uncertainty into a single selected exit depth, so its realized speedup depends on selector quality and rollback behavior. Depth exploration instead turns the unknown EAD into a deterministic upward rounding problem over $\mathcal{X}$, making the depth-side overhead explicitly controlled by exploration resolution. DEX instantiates this formulation by expanding multiple candidate depth branches, committing according to the final-depth reference, and preserving only branch states consistent with the committed prefix.

B DEX overhead analysis

From theoretical speed analysis to walltime speedup. Proposition 7 characterizes the idealized depth-side critical-path opportunity of exploration. In the actual implementation, the final-depth token is retained as the exact commit reference, while parallel explorers compute reusable candidate branches before the commit is resolved. Thus, DEX accelerates decoding not by changing the target

final-depth distribution, but by shortening the amount of post-commit computation that must be performed sequentially.

We use the following calibrated walltime model:

$$S_{\mathcal{X}}^{\text{wall}} = \frac{T(L + \delta_{\text{AR}})}{\sum_{t=1}^{T} \left(\lceil \text{EAD}(t) \rceil_{\mathcal{X}} + \delta_{\mathcal{X}}(t) \right)}.$$

Here δ_{AR} denotes the average per-token overhead of AR beyond layer forward, and $\delta_{\mathcal{X}}(t)$ denotes the extra cost paid by DEX at token t under exploration set $\mathcal{X}$, including lattice expansion, batched branch execution, attention masking, KV-cache management, communication, commit, and collapse. Since DEX introduces branch and distributed execution costs that are not present in AR, the realized walltime speedup is lower than the idealized depth-side speedup. Our profile in Table 1 reports an implementation profile. Expansion remains the dominant cost, accounting for 89.13% of the profiled time. The main non-expansion cost is communication, while collapse and decode/commit are small. This suggests that the practical runtime cost of DEX mainly comes from distributed synchronization and branch expansion, not from the commit–collapse bookkeeping itself.

C DEX additional details

C.1 DECS

Depth-Coupled Sampling (DECS) is used to diversify the intermediate proposals exposed by different depths within the same sampling slot. In the depth–position lattice, a sampling slot may be materialized at multiple candidate depths, forming a slot chain. If each depth decodes independently, shallower and deeper candidates in the same slot often repeat the same token. Such duplicates provide little benefit: if the repeated token matches the final-depth reference, the shallower occurrence would already be selected; otherwise, repeating it only reduces the chance that another depth proposes a token matching the reference.

For a lattice node i, DECS collects the tokens that have already been proposed by shallower nodes in the same slot chain, denoted by $\text{Prior}(i)$, and masks these tokens before decoding the current non-final candidate. This encourages deeper explorers to cover alternative tokens while leaving the final-depth reference unchanged. In particular, when the current node is the final-depth reference node i_{ref} at the final FSM state $q_{\max}$, DECS disables masking and decodes from the original logits. Therefore, the committed token remains exactly the token produced by the unmodified final-depth model, and DECS only affects which intermediate branch may be reused after commitment.

For sampling, DECS uses a shared Gumbel buffer indexed by the slot root $r(i)$. All depths in the same slot chain use the same Gumbel row, so the final-depth reference is sampled by the standard Gumbel-Max rule from the unmasked final-depth logits. The masking operation is applied only to non-final proposals and therefore does not change the target sampling distribution of the committed token.

C.2 Inducing adapter

For standard LLMs, we train inducing adapters and keep the backbone fixed. Adapters are attached only to the middle candidate depths used by DEX, i.e., $\ell \in \mathcal{X} \setminus \{L\}$; the final-depth branch uses the original hidden state and LM head without an adapter. In a DEX-$1/K$ configuration, this corresponds to the $K - 1$ non-final stage boundaries. For a hidden state $h_\ell \in \mathbb{R}^h$, the adapter computes

$$\Delta h_\ell = \alpha_\ell W_{2,\ell} \sigma(W_{1,\ell} \text{RMSNorm}(h_\ell)), \qquad \tilde{h}_\ell = h_\ell + \Delta h_\ell,$$

where $W_{1,\ell} \in \mathbb{R}^{m \times h}$, $W_{2,\ell} \in \mathbb{R}^{h \times m}$, $m = 2h$ by default, σ is the backbone activation, and α_ℓ is a learnable scalar initialized to 10^{-3}. The induced representation $\tilde{h}_\ell$ is decoded by the same final normalization and LM head as the backbone.

The parameter count of one adapter is

$$P_{\text{ad}} = hm + mh + m + 2h + 1,$$

including two linear layers with biases, the RMSNorm scale, and the scalar gate. With $m = 2h$, this becomes $P_{\text{ad}} = 4h^2 + 4h + 1$. For $n = |\mathcal{X}| - 1$ non-final depths, the total adapter size is nP_{ad}, and

Algorithm 2 Depth-Coupled Sampling (DECS)

Input: Logits $\ell \in \mathbb{R}^V$; lattice node i; explore buffer $\mathcal{B}$; prior-depth map $\mathrm{Prior}(\cdot)$; slot-root map $r(\cdot)$; mode $m \in \{greedy, sample\}$; temperature τ; shared Gumbel buffer $\mathcal{G}$; current state q; final state $q_{\max}$; reference node i_{ref}
Output: Decoded token y

$\mathcal{A} \leftarrow \{\mathcal{B}[a] \mid a \in \mathrm{Prior}(i),\ \mathcal{B}[a] \neq \perp\}$ ▷ prior proposals in the same slot chain
if $q = q_{\max}$ and $i = i_{\mathrm{ref}}$ **then**
  $\mathcal{A} \leftarrow \emptyset$ ▷ preserve reference token
end if
for all $v \in \mathcal{A}$ **do**
  $\ell_v \leftarrow -\infty$ ▷ mask prior same-slot proposals
end for
if $m = greedy$ **then**
  $y \leftarrow \arg\max_v \ell_v$
else
  if $\mathcal{G}_{r(i),:}$ is uninitialized **then**
    Draw $\mathcal{G}_{r(i),v} \sim \mathrm{Gumbel}(0, 1)$ for all v
  end if
  $y \leftarrow \arg\max_v \left(\ell_v/\tau + \mathcal{G}_{r(i),v}\right)$
end if
$\mathcal{B}[i] \leftarrow y$
return y

its relative size is $nP_{\mathrm{ad}}/P_{\mathrm{backbone}}$. For example, this gives 536.9M parameters for CodeLlama-34B under DEX-$1/3$ ($\approx 1.58\%$ of the backbone), 805.4M for Llama2-70B under DEX-$1/4$ ($\approx 1.15\%$), and 1.88B for Llama2-70B under DEX-$1/8$ ($\approx 2.68\%$).

We train the adapters by self-distillation on ShareGPT [35], which contains around 70k samples for training. We use 70K samples, truncate or pad each sequence to at most 1024 tokens, and mask padding and unsupervised positions from the loss. For each batch, the frozen full-depth model produces teacher logits z_L. Each selected non-final depth produces student logits z_ℓ through its adapter and the shared LM head. The objective is masked full-vocabulary KL distillation:

$$\mathcal{L} = \frac{1}{|\mathcal{X}| - 1} \sum_{\ell \in \mathcal{X} \setminus \{L\}} \tau^2 \mathrm{KL}\left(\mathrm{softmax}(z_L/\tau) \parallel \mathrm{softmax}(z_\ell/\tau)\right),$$

with $\tau = 1$.

We optimize only adapter parameters with AdamW. The reported runs use learning rate 10^{-3}, weight decay 0, two epochs, 6% warmup steps followed by linear decay, micro-batch size 4, and gradient accumulation 4, giving an effective batch size of 16. The backbone, final normalization, and LM head are frozen throughout training. Adapter training is performed on 4 NVIDIA H100 GPUs and takes about 8 hours for 32B/34B models and 12.6 hours for 70B models.

D Experiment details

Implementations for each method. `DEX` is implemented with Hugging Face Transformers, and cross-GPU communication is performed through the NCCL backend. Unless otherwise stated, all baselines are evaluated using their official implementations; Tensor Parallelism is launched with DeepSpeed. For LayerSkip-trained models, `DEX` uses no inducing adapter. For standard Llama/CodeLlama/Qwen models, `DEX` attaches inducing adapters only to non-final candidate depths, while the final-depth branch, final normalization, and LM head remain unchanged. Thus, adapters only affect which intermediate branches may become reusable; the committed output is still resolved against the final-depth branch, preserving equivalence to the target decoding distribution. PEARL uses Llama-2-7B as the draft model for Llama-2-70B, CodeLlama-7B for CodeLlama-34B, and Qwen3-1.7B for Qwen3-32B. EAGLE-2 on Llama-2-70B uses `yuhuili/EAGLE-llama2-chat-70B` as the draft model, and EAGLE-3 on Qwen3-32B uses `AngelSlim/Qwen3-32B_eagle3`.

Table 2: Depth-axis speed comparison results. Speed (TPS) is reported in tokens/s, with 95% bootstrap confidence intervals below. Speedup is the speed ratio to the strongest of LSSD, AdaDecode, and DEL for the same model and dataset. Total pools all three datasets.

		GSM8K		**HumanEval**		**XSum**		**Total**	
Method	GPUs	TPS	Speedup	TPS	Speedup	TPS	Speedup	TPS	Speedup
				LS-CodeLlama-34B					
LSSD	3	18.82 [17.51, 19.72]	0.53×	37.31 [35.79, 38.65]	0.86×	30.22 [29.38, 31.05]	0.78×	30.23 [29.24, 31.23]	0.77×
AdaDecode	3	35.74 [31.73, 39.04]	1.00×	43.62 [41.67, 45.36]	1.00×	38.33 [37.50, 39.17]	0.99×	39.19 [38.25, 40.13]	1.00×
DEL	3	35.47 [30.07, 39.81]	0.99×	42.64 [40.24, 44.77]	0.98×	38.82 [37.76, 39.92]	1.00×	39.32 [38.22, 40.44]	1.00×
DEX-1/3	3	45.88 [42.16, 48.34]	1.28×	45.79 [44.39, 47.15]	1.05×	47.39 [46.66, 48.15]	1.22×	46.99 [46.31, 47.64]	1.20×
				LS-Llama-2-70B					
LSSD	4	16.64 [14.32, 18.31]	0.89×	19.13 [18.53, 19.69]	0.84×	15.03 [14.10, 16.01]	0.78×	16.97 [16.31, 17.63]	0.83×
AdaDecode	4	18.55 [16.39, 20.38]	1.00×	22.71 [21.90, 23.43]	1.00×	19.34 [18.00, 20.76]	1.00×	20.47 [19.62, 21.31]	1.00×
DEL	4	18.62 [16.31, 20.39]	1.00×	21.29 [20.36, 22.23]	0.94×	18.00 [17.11, 18.91]	0.93×	19.41 [18.75, 20.08]	0.95×
DEX-1/4	4	29.90 [27.51, 31.73]	1.61×	26.07 [24.32, 27.68]	1.15×	28.31 [26.94, 29.69]	1.46×	28.26 [27.24, 29.26]	1.38×

Hyperparameters. We tune LayerSkip self-speculation and AdaDecode on LayerSkip CodeLlama-34B and Llama-2-70B using five prompts from each of HumanEval, XSum, and GSM8K. For self-speculation, the verification layer is fixed to the full model depth, i.e., 48 for CodeLlama-34B and 80 for Llama-2-70B. For CodeLlama-34B, we search exit layers $\{2, 4, 6, 8, 10\}$ and speculation lengths $\{2, 4, 6, 8\}$. For Llama-2-70B, we search exit layers $\{4, 8, 12, 16, 20\}$ with the same speculation lengths. For AdaDecode, intermediate hidden states are projected using the shared LayerSkip LM head, with shortcut layers uniformly distributed across depth: layers $8, 16, 24, 32, 40$ for CodeLlama-34B and layers $10, 20, \ldots, 70$ for Llama-2-70B. We search speculation lengths $\{2, 4, 5, 6, 8\}$ and confidence thresholds $\{0.2, 0.4, 0.6, 0.8\}$. Speed is measured as the total number of generated tokens divided by the total end-to-end response time.

The final self-speculation settings are as follows. For CodeLlama-34B, we use HumanEval $(8, 4)$, GSM8K $(2, 2)$, and XSum $(10, 4)$. For Llama-2-70B, we use HumanEval $(8, 4)$, GSM8K $(8, 4)$, and XSum $(8, 6)$. Each tuple denotes $(\texttt{exit_layer}, \texttt{num_speculations})$.

The final AdaDecode settings are as follows. For CodeLlama-34B, we use HumanEval $(8, 0.6)$, GSM8K $(8, 0.6)$, and XSum $(8, 0.8)$. For Llama-2-70B, we use HumanEval $(8, 0.8)$, GSM8K $(8, 0.8)$, and XSum $(8, 0.6)$. Each tuple denotes $(\texttt{num_speculations}, \texttt{threshold})$.

For DEL, we use its official dynamic decoding configuration and do not perform an additional grid search.

For Lookahead, PEARL, and EAGLE-series baselines, we follow their official evaluation configurations.

Datasets. We randomly sample 128 prompts from each of GSM8K, HumanEval, and XSum. For Llama and CodeLlama experiments, the maximum number of new tokens is set to 512. For Qwen models, we set the maximum number of new tokens to 1024 to avoid truncating longer generated responses. By default, evaluations use greedy decoding with batch size 1.

E Numeric experiment results

We provide numeric results of figures in Section 4 with bootstrap confidence intervals, as shown in Tables 2–6.

Table 3: Throughput scaling with exploration granularity. DEX-$1/K$ denotes an exploration granularity of $1/K$ of the full model depth. Total TPS is computed by pooling the generated tokens and decoding time across GSM8K, HumanEval, and XSum. Values in brackets are 95% bootstrap confidence intervals. Oracle rows give the EAD-implied throughput reference.

Configuration	Total TPS
LS-CodeLlama-34B	
DEX-1/2	36.14 [35.77, 36.50]
DEX-1/3	46.99 [46.31, 47.64]
DEX-1/6	70.63 [69.16, 72.12]
EAD-implied Oracle	100.58
LS-Llama-2-70B	
DEX-1/2	17.49 [17.15, 17.82]
DEX-1/4	28.26 [27.24, 29.26]
DEX-1/8	40.66 [38.79, 42.51]
EAD-implied Oracle	42.49

Table 4: Ablation of depth exploration. The single-exit variant and full exploration are compared at the same exploration granularity. TPS is aggregated across GSM8K, HumanEval, and XSum, with 95% bootstrap confidence intervals below. Speedup is the ratio of exploration TPS to single-exit TPS.

Granularity	Single-Exit TPS	Exploration TPS	Speedup
LS-CodeLlama-34B			
$1/3$	46.67 [45.84, 47.46]	46.99 [46.31, 47.64]	$1.01\times$
$1/6$	41.40 [40.04, 42.92]	70.63 [69.16, 72.12]	$1.71\times$
LS-Llama-2-70B			
$1/4$	23.35 [22.27, 24.41]	28.26 [27.24, 29.26]	$1.21\times$
$1/8$	24.29 [23.07, 25.54]	40.66 [38.79, 42.51]	$1.67\times$

Table 5: Effect of DECS under greedy decoding and temperature sampling. TPS is aggregated across GSM8K, HumanEval, and XSum, with 95% bootstrap confidence intervals below. Speedup is the ratio of DECS TPS to vanilla TPS under the same decoding setting and exploration granularity.

Decoding	Granularity	Vanilla TPS	DECS TPS	Speedup
LS-Llama-2-70B				
Greedy	$1/4$	25.77 [25.41, 26.11]	28.26 [27.24, 29.26]	$1.10\times$
Greedy	$1/8$	35.20 [34.57, 35.81]	40.66 [38.79, 42.51]	$1.16\times$
Temperature $= 1$	$1/4$	20.27 [20.02, 20.52]	20.41 [20.11, 20.69]	$1.01\times$
Temperature $= 1$	$1/8$	25.29 [24.91, 25.66]	26.73 [26.27, 27.19]	$1.06\times$

Table 6: Broader comparison results on standard LLMs. Speed (TPS) is reported in tokens/s, with 95% bootstrap confidence intervals below. Speedup is the speed ratio to AR for the same model and dataset. TP reports the best speed over 2, 4, and 8 GPUs. AR and TP retain the original point estimates without confidence intervals. Total pools all three datasets.

		GSM8K		**HumanEval**		**XSum**		**Total**	
Method	GPUs	TPS	Speedup	TPS	Speedup	TPS	Speedup	TPS	Speedup
					CodeLlama-34B				
AR	3	20.81	1.00×	20.64	1.00×	20.41	1.00×	20.61	1.00×
TP	2/4/8	25.91	1.25×	25.97	1.26×	25.63	1.26×	25.82	1.25×
Lookahead	3	35.22 [34.57, 35.84]	1.69×	26.29 [25.68, 26.91]	1.27×	23.52 [23.05, 24.01]	1.15×	27.78 [27.16, 28.43]	1.35×
AdaDecode	3	33.80 [33.25, 34.32]	1.62×	26.51 [25.97, 27.04]	1.28×	23.98 [23.39, 24.59]	1.17×	27.69 [27.12, 28.28]	1.34×
PEARL	3	45.34 [44.81, 45.89]	2.18×	48.53 [47.72, 49.25]	2.35×	37.02 [36.27, 37.76]	1.81×	42.58 [41.90, 43.26]	2.07×
DEX-1/3	3	44.64 [44.23, 45.04]	2.15×	44.66 [43.83, 45.46]	2.16×	35.46 [34.96, 35.99]	1.74×	40.84 [40.22, 41.47]	1.98×
DEX-1/6	6	65.96 [64.97, 66.94]	3.17×	60.47 [58.75, 62.14]	2.93×	53.77 [52.68, 54.87]	2.63×	59.68 [58.73, 60.65]	2.90×
					Llama-2-70B				
AR	4	10.73	1.00×	10.77	1.00×	10.63	1.00×	10.73	1.00×
TP	2/4/8	15.64	1.46×	15.18	1.41×	15.19	1.43×	15.29	1.42×
Lookahead	4	18.42 [18.13, 18.70]	1.72×	16.71 [16.48, 16.94]	1.55×	13.02 [12.77, 13.28]	1.22×	16.06 [15.81, 16.30]	1.50×
PEARL	4	31.98 [31.28, 32.71]	2.98×	25.64 [24.51, 26.79]	2.38×	24.69 [24.16, 25.21]	2.32×	26.65 [25.93, 27.39]	2.48×
EAGLE-2	4	39.46 [38.64, 40.27]	3.68×	40.95 [40.23, 41.68]	3.80×	26.19 [25.39, 27.00]	2.46×	36.48 [35.63, 37.31]	3.40×
DEX-1/4	4	26.63 [26.34, 26.93]	2.48×	28.97 [28.72, 29.22]	2.69×	21.90 [21.41, 22.35]	2.06×	26.52 [26.13, 26.89]	2.47×
DEX-1/8	8	40.08 [39.49, 40.70]	3.74×	44.38 [43.87, 44.86]	4.12×	30.47 [29.79, 31.15]	2.87×	39.44 [38.70, 40.15]	3.68×
					Qwen3-32B				
AR	3	21.56	1.00×	22.22	1.00×	22.17	1.00×	21.98	1.00×
TP	2/4/8	16.17	0.75×	14.66	0.66×	14.73	0.66×	15.24	0.69×
PEARL	3	29.71 [29.29, 30.13]	1.38×	26.19 [25.92, 26.45]	1.18×	25.67 [25.39, 25.96]	1.16×	27.27 [27.02, 27.54]	1.24×
EAGLE-3	3	37.38 [36.91, 37.87]	1.73×	33.07 [32.72, 33.44]	1.49×	28.87 [28.33, 29.41]	1.30×	33.46 [33.06, 33.88]	1.52×
DEX-1/3	3	27.01 [26.82, 27.21]	1.25×	26.45 [26.26, 26.65]	1.19×	24.16 [23.88, 24.44]	1.09×	26.12 [25.95, 26.28]	1.19×
DEX-1/8	8	42.51 [42.01, 43.03]	1.97×	39.82 [39.33, 40.29]	1.79×	34.34 [33.96, 34.72]	1.55×	39.46 [39.06, 39.87]	1.80×

Table 7: Auxiliary training cost comparison between DEX-1/3 and EAGLE-3 on Qwen3-32B. Parameter counts and training FLOPs are architecture-based estimates. EAGLE-3's response-regeneration cost is excluded.

Item	DEX-1/3	EAGLE-3
Trainable parameters	$\sim$209.8M	$\sim$727.5M
Training data	ShareGPT: 70K	ShareGPT: 68K + UltraChat-200K: 464K
Data regeneration	No	Yes; target-generated responses
Training method	Self-distillation	Training-Time Test (TTT)
Estimated compute/token	73.60 GFLOP	93.84 GFLOP
Estimated compute/epoch	5.28 EFLOP	51.12 EFLOP

F Extended discussion

Resource-dependent limit of depth exploration. Depth exploration also has a resource-dependent upper bound. Let d_0 be the shallowest explored boundary in $\mathcal{X}$. Even if every token becomes ready before d_0, DEX can only reuse the branch at d_0, so the idealized depth-side speedup is bounded by approximately L/d_0. Under a uniform K-way depth partition, $d_0 \approx L/K$, giving an approximate upper bound of K, which matches the number of depth explorers in the current implementation. This limit clarifies the role of exploration resolution: increasing K improves the depth-side approximation to the EAD oracle, but also requires more devices and increases lattice-expansion overhead.

Training cost of inducing adapters. DEX requires no additional training when applied to early-exit-trained LLMs such as LayerSkip models. For standard LLM checkpoints, DEX uses lightweight inducing adapters trained through self-distillation, while keeping the backbone frozen. To further examine the training-cost trade-offs, we compare DEX with EAGLE-3 on the common Qwen3-32B setting in terms of trainable parameters, training data, and estimated training compute. Following the reported model architectures, datasets, and training procedures, we assume 1,024 tokens per sample and exclude EAGLE-3's target-model response regeneration cost, which favors EAGLE-3 in this comparison. Table 7 summarizes the results.

DEX-1/3 requires approximately 209.8M trainable adapter parameters, compared with 727.5M for the EAGLE-3 draft model, representing a 3.5$\times$ reduction. With two training epochs, the reported DEX training recipe costs approximately 10.56 EFLOP, compared with 51.12 EFLOP for a single estimated EAGLE-3 epoch. Thus, DEX requires approximately 4.8$\times$ less auxiliary training compute, even before accounting for EAGLE-3's response-regeneration cost.

Under the matched 3-GPU inference setting, DEX-1/3 achieves 26.1 TPS (1.19$\times$ over AR), while EAGLE-3 achieves 33.5 TPS (1.52$\times$ over AR). Although EAGLE-3 achieves higher decoding speed under the same GPU budget, DEX offers substantially lower auxiliary training requirements. Normalizing the incremental speedup over AR by the estimated training FLOPs yields an approximately 1.8$\times$ advantage for DEX in incremental speedup per training FLOP.

This comparison highlights the favorable training-cost trade-off of DEX, particularly when auxiliary training resources are limited. The estimate should be interpreted with the caveat that the exact EAGLE-3 training recipe is not fully disclosed. With additional explorers, DEX-1/8 achieves higher absolute speed, reaching 39.5 TPS, at the cost of a larger 8-GPU inference budget.

Broader impacts. DEX improves the efficiency of autoregressive LLM decoding by increasing end-to-end speed while preserving the final-depth decoding rule. This can reduce inference latency per generated token, making large-model deployment more accessible. At the same time, faster and cheaper generation may also lower the cost of harmful large-scale uses, such as spam, misinformation, or automated abuse. Because DEX is a lossless decoding wrapper and does not modify the final-depth model distribution, these risks are primarily inherited from the underlying model and deployment context. Practical deployments should therefore combine decoding acceleration with the same safety filtering, monitoring, and rate-limiting mechanisms used for the base model.

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: The abstract and introduction state the paper's main contributions and scope, including the formulation of depth exploration, the lossless expand–commit–collapse decoding procedure, and the evaluated decoding speed improvements over depth-selection and distributed decoding baselines. These claims are supported by the method description and empirical results in Sections 2–4.

   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 discuss limitations in Appendix F, including the resource-dependent realization of depth exploration, the growth of the depth–position lattice with the number of explorers, possible wall-clock saturation at finer exploration resolutions, and the resource-dependent upper bound of pure depth exploration.

   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: [Yes]

   Justification: Section 2 states the stable-EAD convention, the idealized layer-cost accounting, and the assumptions behind the depth-side speedup model. Appendix A provides complete derivations and proofs for the selection overhead, exploration overhead, speedup bound, and monotonicity statements.

   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: The paper provides the algorithmic details of DEX in Section 3, Appendix C and Algorithms 1–2, and reports the models, datasets, baselines, hardware, GPU counts, decoding settings, and baseline hyperparameters in Section 4 and Appendix D.

   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 provide code through the link included in the submission, including the core DEX implementation, scripts, and instructions for running DEX and reproducing the main experiments. Appendix D describes the data access, decoding settings, baseline configurations, and evaluation protocol.

   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: Section 4 specifies the main experimental setup, including evaluated model variants, datasets, baseline methods, hardware, GPU counts, exploration resolutions, batch size, decoding mode, and maximum generation length. Appendices C.2 and D provide additional details on adapter training, dataset/workload selection, baseline configurations, hyperparameter choices, and the speed measurement protocol. For experiments that do not involve training, the relevant decoding and evaluation settings are specified instead of training splits or optimizer details.

   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 report numerical speed results with bootstrapped 95% confidence intervals, obtained by resampling evaluation samples, to quantify variability across workloads. The results cover multiple models, datasets, and decoding baselines, including comparisons under matched GPU budgets. These statistics provide uncertainty estimates for the main experimental results and support the assessment of DEX's speed improvements over depth-selection baselines.

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: Section 4 reports the compute workers and resource allocation used in the main experiments, including NVIDIA H100 GPUs, GPU counts for tensor parallelism and distributed baselines, and the DEX-1/K setting where K depth explorers run on K GPUs. Appendices C.2 and D further provide implementation details, adapter-training compute, speed measurement protocol, dataset sampling, and decoding settings needed to reproduce the reported experiments.

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 reviewed the NeurIPS Code of Ethics and believe that the research conforms to it. The paper focuses on lossless LLM decoding acceleration and does not involve human-subject research or private data collection.

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 broader impacts in Appendix F, including the potential benefits of lower-latency LLM inference, as well as the potential risks that faster generation may reduce the cost of large-scale misuse. Since DEX is a lossless decoding wrapper that preserves the final-depth model distribution, these risks primarily follow from the underlying deployed model and application setting rather than from changing model behavior.

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: The paper does not release new pretrained models, image generators, scraped datasets, or other high-risk assets that would require additional release safeguards. The released artifact is an implementation of the proposed lossless decoding method, while the experiments use publicly available model checkpoints and datasets.

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: The paper identifies the existing model checkpoints, datasets, and baseline methods used in the experiments, including CodeLlama, Llama2, Qwen3, GSM8K, HumanEval, XSum, LayerSkip, AdaDecode, DEL, Lookahead, PEARL, and EAGLE-series baselines. These assets are publicly available and are used only for research evaluation under their respective licenses and terms of use.

    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 release a code artifact through the link included in the submission, including the core DEX implementation, scripts, dependency specifications, and instructions for reproducing the main speed experiments. We do not introduce new datasets or new pretrained model checkpoints; the experiments use publicly available models and datasets.

    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: The paper does not involve crowdsourcing, human-subject experiments, user studies, or newly collected human annotations. The empirical evaluation uses publicly available model checkpoints, datasets, and benchmarks.

    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: The paper does not involve human-subject research, user studies, crowdsourcing, or newly collected human annotations, and therefore does not require institutional review board approval or an equivalent review process.

    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: [Yes]

    Justification: LLMs are central to the proposed method and experiments rather than being used only for writing assistance. Sections 3 and 4 describe the DEX decoding procedure and its application to evaluated LLMs, including the model variants, final-depth reference rule, explored intermediate depths, decoding settings, and speed evaluation protocol.

    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.