CARVE: Verified Expansion for Variable-Length Generation in Diffusion Language Models

Wail Bouhedja[1,2], Amr Mohamed[1,3], Guokan Shang[1]

[1]MBZUAI, [2]Sorbonne Université, [3]Ecole Polytechnique

Abstract

Masked diffusion language models predict tokens from a partially observed response canvas, enabling bidirectional conditioning and parallel token refinement. Yet standard masked-diffusion decoders use a rigid inference interface: the number of masked positions allocated to the answer is fixed before generation begins. Choosing this length is difficult. A short canvas can truncate reasoning or code, while a long canvas wastes computation and can perturb denoising. We introduce CARVE (**C**ounterfactual-**A**ware **R**eveal with **V**erified **E**xpansion), a training-free variable-length algorithm for masked diffusion LMs. Starting from a shorter canvas, CARVE can grow the response during decoding by inserting additional `[MASK]` positions. Rather than keeping every insertion, CARVE tests a candidate expanded canvas and asks a counterfactual question: would the model make similar predictions for the unresolved positions in the original canvas if the extra masked space were present? The inserted masks are kept only when they induce low Jensen–Shannon (JS) divergence on aligned unresolved positions. This makes length growth a verified stability decision rather than a pure confidence heuristic. CARVE applies without retraining to both full-canvas and blockwise diffusion decoders. Across code generation and mathematical reasoning benchmarks, CARVE consistently improves average performance over fixed-length baselines across all evaluated model families. Crucially, CARVE achieves these accuracy gains while reducing inference cost, reaching half the FLOPs of fixed-length decoding in some settings.

1 Introduction

Autoregressive language models (Brown et al., 2020) have a simple interface for open-ended generation: they emit one token at a time and stop when an end-of-sequence token is produced. Masked diffusion language models (Austin et al., 2021a; Campbell et al., 2022; Zheng et al., 2023; Lou et al., 2023; Sahoo et al., 2024; Shi et al., 2024), by contrast, generate by iteratively denoising a canvas of masked tokens. This paradigm enables attractive properties such as parallel token updates, bidirectional conditioning, and arbitrary-order refinement, and has recently scaled to competitive instruction-following models such as LLaDA (Nie et al., 2026) and Dream (Ye et al., 2025). However, it exposes a basic inference problem: the generation length must typically be fixed before decoding begins.

This fixed-canvas assumption is poorly matched to realistic generation. If the canvas is too short, the model may truncate code, reasoning, or explanations. If the canvas is too long, inference wastes computation on masked positions that may not be needed for the final answer, and can even harm quality by forcing the model to denoise beyond the useful response (Rossi et al., 2026). The appropriate length is also instance-dependent: two prompts from the same task may require very different response lengths, even when they use the same decoding setup. As a result, variable-length decoding is not a minor engineering detail, but a central obstacle to making diffusion LMs practical for open-ended generation.

A natural solution is to adapt the canvas during denoising. Prior work such as DAEDAL (Li et al., 2025) proposes training-free length expansion for diffusion language models by using internal confidence signals to decide when to allocate additional masked tokens. This shows that response length need not be fixed before decoding begins and can instead be adjusted during inference. Yet expansion introduces a second question that is easy to overlook: adding masks changes the denoising problem itself. In a bidirectional masked diffusion model, inserting new masked positions may perturb the model's predictions at other still-unresolved positions. An appropriate length expansion rule should therefore ask both whether additional space might

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 51779–51795
October 24-29, 2026 ©2026 Association for Computational Linguistics

be useful and whether inserting that space leaves the existing predictions sufficiently stable.

We introduce CARVE (**C**ounterfactual-**A**ware **R**eveal with **V**erified **E**xpansion), a training-free variable-length decoding algorithm for masked diffusion language models. CARVE augments a standard diffusion sampler with a verified expansion step. Before certain reveal steps, CARVE proposes inserting a number of additional masked tokens into the current canvas. It then runs the model once on the expanded canvas and compares the predictive distributions of the original and expanded canvases at aligned still-masked positions. The insertion is accepted only when the mean Jensen–Shannon divergence between these distributions remains below a fixed threshold. Otherwise, the expanded branch is rejected and we continue with the original canvas. In this way, the canvas grows only when the proposed expansion leaves the predictive distributions at aligned unresolved positions sufficiently stable under this criterion.

This turns length expansion into a stability check rather than a pure confidence heuristic. Each proposed insertion asks a local counterfactual question: would the model make essentially the same predictions for the unresolved tokens if the canvas contained additional space? When the answer is yes, the added masks can be committed without substantially changing the denoising state. When the answer is no, the proposal is treated as a destabilizing edit and discarded. This criterion differs from expansion rules based only on absolute confidence or end-of-sequence (EOS) token behavior (Yang et al., 2026): it measures the effect of the insertion on the model's remaining predictive state. The same principle applies across different masked diffusion backends; in this work, we instantiate CARVE for both full-canvas Dream decoding and blockwise LLaDA decoding.

Our experiments evaluate CARVE on code and mathematical reasoning benchmarks across different dLLMs. We compare against fixed-length baseline decoding and DAEDAL (Li et al., 2025), the closest training-free variable-length baseline. Across models, CARVE improves average performance over fixed-length baseline decoding. Our analysis of model-forward FLOPs further shows that verified expansion does not simply trade additional computation for accuracy: adaptive reveal and EOS cropping often offset the cost of branching. Consequently, CARVE provides a dual advantage, improving average task accuracy while yielding up to roughly half as many FLOPs as fixed-length decoding in some settings. Code is publicly available[1]. Our contributions are:

- We propose CARVE, a training-free variable-length algorithm that verifies each proposed canvas expansion by measuring Jensen–Shannon divergence on aligned unresolved positions.
- We show that verified expansion provides a model-agnostic mechanism for masked diffusion LMs, applying the same algorithm to both full-canvas Dream decoding and blockwise LLaDA decoding.
- We demonstrate that CARVE improves average performance over fixed-length baselines across code and mathematical reasoning benchmarks on three diffusion LMs, while often reducing inference FLOPs relative to fixed-length decoding.

2 Related Work

Discrete diffusion language models. Masked diffusion large language models (dLLMs) generate sequences by iteratively reversing categorical corruption processes, typically relying on absorbing-state masking to denoise positions in parallel (Austin et al., 2021a; Campbell et al., 2022; Zheng et al., 2023; Lou et al., 2023; Sahoo et al., 2024; Shi et al., 2024; Mohamed et al., 2026). Recent large-scale models, such as LLaDA and Dream, demonstrate that this paradigm scales effectively to complex instruction-following and reasoning tasks (Nie et al., 2026; Ye et al., 2025). However, because these models predict and reveal subsets of tokens across a predefined masked canvas, they inherently impose a strict length constraint before decoding begins.

Training-time variable-length generation in dLLMs. To overcome the fixed-length bottleneck, several approaches modify the underlying generative formulation or state space. Existing methods implement dynamic expansion and contraction for code infilling (Wu et al., 2026), formulate generation via explicit insertion and deletion edits (Havasi et al., 2025), or jointly denoise token identities and positional configurations (Zhang

[1] https://github.com/wailji/CARVE

et al., 2025; Kim et al., 2025). While these strategies successfully enable dynamic length adjustment, they require specialized architectural modifications or costly retraining. CARVE, by contrast, applies directly to pretrained masked dLLMs.

Training-free variable-length decoding in dLLMs. Inference-time methods bypass retraining by dynamically adjusting the canvas length during decoding. Prior approaches trigger length changes using scalar internal confidence cues (Li et al., 2025), length-regularized candidate scoring (Cheng et al., 2026), or implicit end-of-sequence (EOS) token densities (Yang et al., 2026). CARVE instead frames length expansion as a counterfactual stability test. It computes the mean Jensen–Shannon divergence between the base and expanded predictive distributions at aligned unresolved positions and accepts the expansion when this mean falls below τ_{JS}. This criterion measures local predictive stability; it does not guarantee positionwise preservation, semantic correctness, or output safety.

3 Methods

In this section, we introduce CARVE, a training-free variable-length decoding algorithm for masked diffusion language models. Figure 1 provides an overview, and Algorithm 1 summarizes the complete decoding procedure.

3.1 Preliminaries: Masked Discrete Diffusion

Masked diffusion language models generate text by denoising discrete token sequences. Given a clean response $x_0 = (x_{0,1}, \ldots, x_{0,L}) \in \mathcal{V}^L$, the forward process gradually replaces tokens with a special mask token [MASK]. Since masked tokens remain masked at all later timesteps, this is an *absorbing* process. At inference time, decoding starts from a masked canvas and progressively commits predicted tokens.

Forward absorbing process. The forward process is a Markov chain

$$q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1}), \tag{1}$$

with transitions that factorize over token positions. For each position i, [MASK] is absorbing:

$$q(x_{t,i} = [\mathrm{MASK}] \mid x_{t-1,i} = [\mathrm{MASK}]) = 1. \tag{2}$$

If $x_{t-1,i} \neq [\mathrm{MASK}]$, then

$$x_{t,i} = \begin{cases} x_{t-1,i}, & \text{with probability } 1-\beta_t, \\ [\mathrm{MASK}], & \text{with probability } \beta_t. \end{cases} \tag{3}$$

We assume that the corruption schedule satisfies $\bar{\alpha}_T = 0$, so the terminal state is fully masked: $x_T = [\mathrm{MASK}]^L$. Let $\bar{\alpha}_t = \prod_{r=1}^{t}(1-\beta_r)$ denote the token survival probability. The marginal corruption process is

$$x_{t,i} = \begin{cases} x_{0,i}, & \text{with probability } \bar{\alpha}_t, \\ [\mathrm{MASK}], & \text{with probability } 1-\bar{\alpha}_t. \end{cases} \tag{4}$$

Denoising model. Given a prompt x_{p} and a partially masked response x_t, the model predicts logits over the vocabulary at every response position:

$$F_t = f_\theta(x_{\mathrm{p}}, x_t, t). \tag{5}$$

The token distribution at position i is

$$p_{t,i} = \mathrm{softmax}(F_{t,i}), \tag{6}$$

where $F_{t,i}$ denotes the logits at position i. We write the resulting clean-token predictor in factorized form:

$$p_\theta(x_0 \mid x_{\mathrm{p}}, x_t, t) = \prod_{i=1}^{L} p_\theta(x_{0,i} \mid x_{\mathrm{p}}, x_t, t). \tag{7}$$

Although the output distribution factorizes over response positions, each factor is computed from the full partially masked canvas, allowing bidirectional conditioning on all visible tokens.

Training objective. The model is trained to reconstruct the original tokens at masked positions. Let $M_t = \{i : x_{t,i} = [\mathrm{MASK}]\}$. The masked denoising loss is

$$\mathcal{L}(\theta) = \mathbb{E}_{(x_{\mathrm{p}},x_0)\sim\mathcal{D}} \mathbb{E}_{t\sim\mathcal{U}\{1,\ldots,T\}} \mathbb{E}_{x_t\sim q(\cdot\mid x_0,t)} \left[-\sum_{i\in M_t} \log p_\theta\left(x_{0,i} \mid x_{\mathrm{p}}, x_t, t\right) \right]. \tag{8}$$

Decoding with partial reveal. At inference time, the response length L is chosen in advance and the decoder initializes a fully masked response canvas,

$$x^{(0)} = [\mathrm{MASK}]^L. \tag{9}$$

Let $\{\tau_s\}_{s=0}^{T}$ denote the reverse denoising schedule, with $\tau_0 = T$ corresponding to the maximally

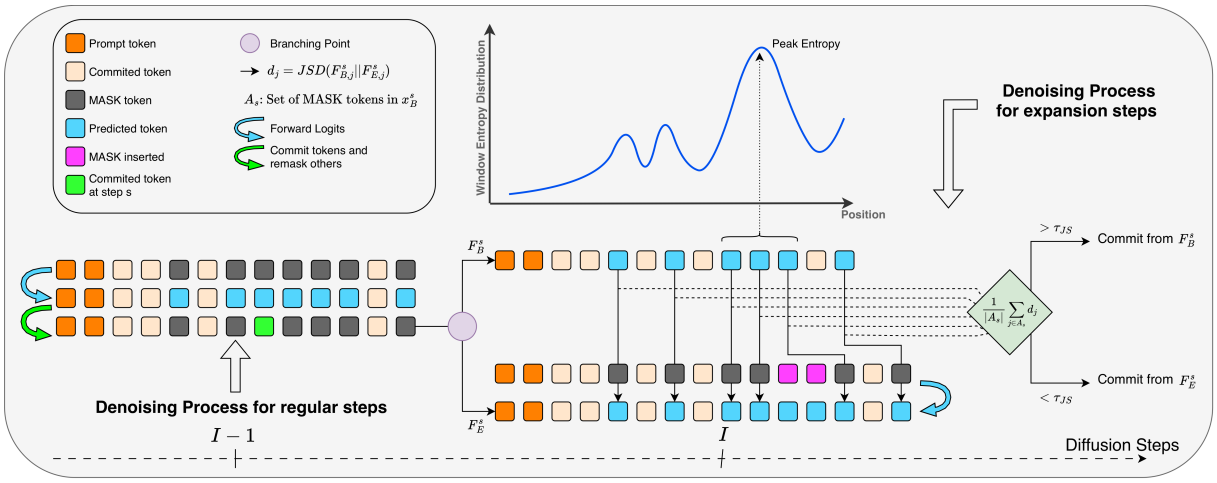

Figure 1: Overview of CARVE. Regular steps predict all masked positions and commit a subset of tokens. Expansion steps insert additional [MASK] tokens near a high-uncertainty region and compare the base and expanded canvases using JS divergence. The expanded canvas is kept only when the mean JS divergence on aligned unresolved positions falls below the acceptance threshold.

corrupted state and $\tau_T = 0$ to the clean state. At decoding step s, the model predicts token distributions for all response positions at diffusion time τ_s. A reveal rule then selects a subset of the currently masked positions to commit. Let

$$m_s = \left|\left\{i : x^s_{\mathrm{C},i} = [\mathrm{MASK}]\right\}\right| \tag{10}$$

denote the number of unresolved positions in the chosen branch. The canvas is updated as

$$x_i^{(s+1)} = \begin{cases} \hat{x}_i^{(s)}, & i \in R^{(s)}, \\ x_i^{(s)}, & i \notin R^{(s)}, \end{cases} \tag{11}$$

where $\hat{x}_i^{(s)}$ is obtained from the model distribution, either greedily or by sampling. Different diffusion models define different reveal rules. In this work, we keep the trained denoising model fixed and modify only the decoding procedure.

3.2 CARVE: Verified Canvas Expansion

CARVE is a training-free decoding method for masked diffusion language models. It keeps the pretrained denoising model and each backend's position-ranking criterion fixed while modifying inference through verified canvas expansion, an adaptive reveal-count schedule, and EOS-triggered cropping. CARVE branches into a candidate expanded canvas, measures the mean distribution shift at aligned unresolved positions, and reveals tokens from the branch selected by the verification criterion.

Let x^s be the response canvas at decoding step s, and let L_s be its current length. Standard masked diffusion decoding fixes this length before generation. In contrast, CARVE starts from an initial length L_0 and allows the canvas to grow up to a maximum length $L_{\max}$. At each step, we first define the current canvas as the *base branch*:

$$x^s_{\mathrm{B}} = x^s. \tag{12}$$

The model is evaluated on this branch to obtain base logits:

$$F^s_{\mathrm{B}} = f_\theta(x_{\mathrm{p}}, x^s_{\mathrm{B}}, \tau_s). \tag{13}$$

Uncertainty-guided branching. Before each potential canvas expansion, CARVE selects where an inserted span of masks may be most useful. It inserts new masks near the most uncertain region of the current canvas, using high predictive uncertainty as a heuristic for where additional capacity may be beneficial.

$$p^s_{\mathrm{B},j} = \mathrm{softmax}(F^s_{\mathrm{B},j}). \tag{14}$$

The uncertainty of position j is measured by entropy:

$$H^s_j = -\sum_{v \in \mathcal{V}} p^s_{\mathrm{B},j}(v) \log p^s_{\mathrm{B},j}(v). \tag{15}$$

Let W be an even window size and let $h = W/2$. We choose an anchor position c_s whose local window has the largest total entropy:

$$c_s = \arg\max_{c \in \mathcal{C}_s} \sum_{j=c-h+1}^{c+h} H^s_j, \tag{16}$$

where $\mathcal{C}_s$ is the set of valid anchor positions whose window lies inside the current response canvas. Intuitively, c_s marks the region where the model is least certain about how to complete the response.

Let

$$k_s = \min(k, L_{\max} - L_s) \tag{17}$$

be the number of mask tokens that can still be inserted. We then branch out by inserting these k_s masks immediately to the right of the anchor position c_s:

$$x_{\mathrm{E}}^s = \mathrm{Insert}(x_{\mathrm{B}}^s, c_s, k_s), \tag{18}$$

where Insert preserves all existing tokens and inserts k_s new mask tokens immediately after position c_s. This gives an *expanded branch*. The model is evaluated once on this expanded canvas:

$$F_{\mathrm{E}}^s = f_\theta(x_{\mathrm{p}}, x_{\mathrm{E}}^s, \tau_s). \tag{19}$$

Aligning base and expanded branches. The inserted masks shift every original position to the right of the anchor. To compare the base and expanded branches, we align each original response position with its corresponding position in the expanded branch. For compactness, response indices omit the prompt offset. The aligned index of an original position j is

$$a_s(j) = j + k_s \mathbf{1}[j > c_s]. \tag{20}$$

Thus, positions up to and including c_s keep their index, while positions after c_s shift by k_s slots. The newly inserted masks are excluded from the comparison because they have no counterpart in the base branch.

We verify the expansion only on unresolved positions that already existed in the base canvas:

$$\mathcal{A}_s = \{j : x_{\mathrm{B},j}^s = [\mathrm{MASK}]\}. \tag{21}$$

Committed tokens are excluded because their values are already fixed.

JS verification score. For every aligned unresolved position $j \in \mathcal{A}_s$, we compare the model's predictive distribution before and after insertion:

$$p_j = \mathrm{softmax}(F_{\mathrm{B},j}^s), \tag{22}$$

$$q_j = \mathrm{softmax}(F_{\mathrm{E},a_s(j)}^s). \tag{23}$$

Let

$$m_j = \frac{1}{2}(p_j + q_j). \tag{24}$$

Algorithm 1 CARVE DECODING WITH VERIFIED EXPANSION

Require: prompt x_{p}, initial length L_0, maximum length $L_{\max}$, steps T, reverse denoising schedule $\{\tau_s\}_{s=0}^{T}$, insertion size k, threshold τ_{JS}, interval I

$x^0 \leftarrow [\mathrm{MASK}]^{L_0}$
for $s = 0, \ldots, T-1$ **do**
  if x^s contains no mask tokens **then**
    return decoded response from x^s
  end if
  $L_s \leftarrow |x^s|$
  $x_{\mathrm{B}}^s \leftarrow x^s$
  $F_{\mathrm{B}}^s \leftarrow f_\theta(x_{\mathrm{p}}, x_{\mathrm{B}}^s, \tau_s)$
  $(x_{\mathrm{C}}^s, F_{\mathrm{C}}^s) \leftarrow (x_{\mathrm{B}}^s, F_{\mathrm{B}}^s)$
  if $s \bmod I = 0$ and $L_s < L_{\max}$ **then**
    choose uncertainty anchor c_s
    $k_s \leftarrow \min(k, L_{\max} - L_s)$
    $x_{\mathrm{E}}^s \leftarrow \mathrm{Insert}(x_{\mathrm{B}}^s, c_s, k_s)$
    $F_{\mathrm{E}}^s \leftarrow f_\theta(x_{\mathrm{p}}, x_{\mathrm{E}}^s, \tau_s)$
    compute D_{JS}^s on aligned unresolved positions
    if $D_{\mathrm{JS}}^s < \tau_{\mathrm{JS}}$ **then**
      $(x_{\mathrm{C}}^s, F_{\mathrm{C}}^s) \leftarrow (x_{\mathrm{E}}^s, F_{\mathrm{E}}^s)$
    end if
  end if
  reveal selected masks in x_{C}^s using F_{C}^s
  set x^{s+1} to the result
  if EOS is committed at position e_s before the canvas end **then**
    $\bar{x}^{s+1} \leftarrow x_{1:e_s}^{s+1}$
    $\bar{F}^{s+1} \leftarrow f_\theta(x_{\mathrm{p}}, \bar{x}^{s+1}, \tau_{s+1})$
    fill all remaining masks in $\bar{x}^{s+1}$ by greedily using argmax $\bar{F}^{s+1}$
    return decoded response from $\bar{x}^{s+1}$
  end if
end for
return decoded response from final canvas

The local Jensen–Shannon divergence is

$$d_j = \frac{1}{2}\mathrm{KL}(p_j \| m_j) + \frac{1}{2}\mathrm{KL}(q_j \| m_j). \tag{25}$$

The verification score is the mean divergence over aligned unresolved positions:

$$D_{\mathrm{JS}}^s = \frac{1}{|\mathcal{A}_s|} \sum_{j \in \mathcal{A}_s} d_j. \tag{26}$$

The expanded branch is accepted only if

$$D_{\mathrm{JS}}^s < \tau_{\mathrm{JS}}, \tag{27}$$

where τ_{JS} is a fixed threshold. A low score indicates that insertion changes the predictive distributions at aligned unresolved positions only slightly on average. When the score exceeds the threshold, the proposed expansion is rejected. Because the score is averaged across positions, acceptance does not guarantee that every individual prediction is preserved.

Branch selection. After verification, decoding continues on exactly one branch. We define the *chosen branch* as

$$(x_{\mathrm{C}}^{s}, F_{\mathrm{C}}^{s}) = \begin{cases} (x_{\mathrm{E}}^{s}, F_{\mathrm{E}}^{s}), & D_{\mathrm{JS}}^{s} < \tau_{\mathrm{JS}}, \\ (x_{\mathrm{B}}^{s}, F_{\mathrm{B}}^{s}), & \text{otherwise.} \end{cases} \tag{28}$$

This branch-consistent design is important: CARVE never commits tokens using logits from a canvas different from the one being updated.

Expansion interval. The expansion is controlled by a hyperparameter I, called the expansion interval. Rather than branching out at every denoising step, CARVE attempts expansion once every I steps, as long as the active canvas has not reached $L_{\max}$. A standard decoding step requires one model forward pass. A step with an expansion attempt requires one additional forward pass for the expanded branch. Thus, before the canvas reaches $L_{\max}$, the average number of forward passes per decoding step is approximately $1 + 1/I$ times that of the underlying decoder; the corresponding FLOP overhead depends on the canvas length.

EOS-triggered cropping and greedy completion. CARVE uses EOS crop as its default stopping rule. After each reveal step, we check whether an end-of-sequence token has been committed before the end of the active response canvas. When this occurs, all positions to the right of the first EOS are discarded, since they would not contribute to the decoded answer. If the remaining prefix still contains mask tokens, the model performs one final forward pass on the cropped canvas and fills every unresolved position greedily by argmax. The resulting prefix is returned as the final response. This makes stopping consistent with answer extraction while avoiding additional denoising steps on tokens that would be discarded.

4 Experiments

We evaluate CARVE on code generation and mathematical reasoning benchmarks using three instruction-tuned masked diffusion language models. We also report DAEDAL (Li et al., 2025) as a training-free variable-length decoding baseline. In addition to task accuracy, we measure the inference cost of each method using forward-pass FLOPs.

4.1 Models

We evaluate Dream-v0-Instruct-7B, LLaDA-1.5, and LLaDA-8B-Instruct. Dream uses full-canvas masked diffusion decoding, while the LLaDA models use the LLaDA-family blockwise decoding setup, with block size 32 for LLaDA-1.5 and block size 64 for LLaDA-8B-Instruct.

4.2 Benchmarks

We evaluate CARVE on four benchmarks: HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021b), MATH-500 (Lightman et al., 2024), and GSM8K (Cobbe et al., 2021). Performance on HumanEval and MBPP is measured using pass@1, while performance on MATH-500 and GSM8K is measured using exact-match accuracy after answer extraction.

4.3 Compared Methods

We compare three decoding methods. **Baseline** denotes the standard fixed-length masked diffusion decoder for each model. **DAEDAL** is a training-free variable-length algorithm that expands the response canvas using internal confidence signals. Since DAEDAL is originally designed for LLaDA-style decoding, we also adapt it to Dream for a consistent comparison. **CARVE** is our training-free verified expansion algorithm. For all CARVE runs, we initialize the canvas at $L_0 = L_{\max}/2$ and allow it to grow up to $L_{\max}$.

4.4 Results

Table 1 shows that CARVE increases the unweighted four-benchmark average relative to fixed-length decoding by 0.92, 1.03, and 0.49 percentage points on Dream-v0-Instruct-7B, LLaDA-1.5, and LLaDA-8B-Instruct, respectively. The per-task effects are mixed: CARVE improves 9 of the 12 model–benchmark pairs, with its largest gain on HumanEval for Dream ($+4.27$ points). DAEDAL attains the higher average on LLaDA-8B-Instruct (53.42 versus 53.20).

This pattern of consistent improvements across diverse benchmarks for all evaluated model families suggests that verified expansion is both task-agnostic and model-agnostic. Rather than relying on a benchmark-specific length heuristic, CARVE uses the model's own predictive stability to decide when expansion is safe. The main exception is MATH-500 for LLaDA-8B-Instruct, where DAEDAL achieves the highest score; nevertheless, CARVE remains competitive on the aggregate and provides the most consistent gains across code and reasoning tasks.

Model	Method	HumanEval	MBPP	MATH-500	GSM8K	Average
Dream-v0-Instruct-7B	Baseline	55.49	**60.60**	39.60	**79.53**	58.80
	DAEDAL	**59.76** (+4.27)	56.40 (−4.20)	**40.80** (+1.20)	76.04 (−3.49)	58.25 (−0.55)
	CARVE	**59.76** (+4.27)	60.40 (−0.20)	40.20 (+0.60)	78.54 (−0.99)	**59.73** (+0.92)
LLaDA-1.5	Baseline	46.95	38.20	43.00	83.09	52.81
	DAEDAL	46.34 (−0.61)	**39.80** (+1.60)	42.00 (−1.00)	**83.85** (+0.76)	53.00 (+0.19)
	CARVE	**50.61** (+3.66)	37.80 (−0.40)	**43.40** (+0.40)	83.55 (+0.46)	**53.84** (+1.03)
LLaDA-8B-Instruct	Baseline	46.95	38.80	43.20	81.88	52.71
	DAEDAL	46.34 (−0.61)	**39.20** (+0.40)	**47.00** (+3.80)	81.12 (−0.76)	**53.42** (+0.71)
	CARVE	**48.17** (+1.22)	**39.20** (+0.40)	43.40 (+0.20)	**82.03** (+0.15)	53.20 (+0.49)

Table 1: Main results on code and mathematical reasoning benchmarks. Scores are reported as percentages. HumanEval and MBPP use pass@1; MATH-500 and GSM8K use exact-match accuracy after answer extraction. "Average" is the unweighted arithmetic mean of the four task scores. Deltas are percentage-point differences relative to the baseline for the same model. Averages and deltas are computed from unrounded scores. Boldface marks the best result, including ties, within each model block.

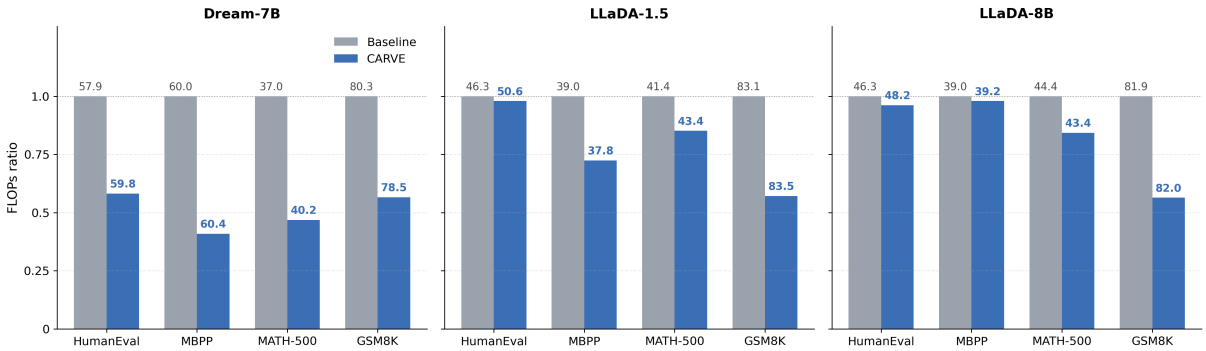

Figure 2: Per-task FLOPs ratio of CARVE relative to the fixed-length baseline. A value below 1.0 means that CARVE uses fewer FLOPs than the baseline. Although CARVE performs an additional forward pass when it branches into an expanded canvas, EOS cropping and adaptive reveal often offset this cost, making the final decoding trajectory more efficient.

Figure 2 shows that these accuracy gains do not come from simply spending more computation. Across models and tasks, CARVE often uses substantially fewer FLOPs than the fixed-length baseline. This may appear counterintuitive because the method branches out and performs an additional forward pass at any given expansion step. In practice, however, accepted expansions are controlled by the JS divergence, and EOS cropping removes suffix positions that would be discarded during answer extraction. As a result, the extra cost of branching is often balanced, and sometimes outweighed, by shorter effective decoding trajectories. Thus, CARVE improves accuracy while remaining computationally efficient.

5 Ablations

5.1 Adaptive reveal rule

We compare the adaptive reveal schedule used in CARVE against Dream's original commit schedule. Dream's original schedule reveals

$$n_s = \left\lfloor m_s \left(1 - \frac{\tau_{s+1}}{\tau_s}\right) \right\rfloor, \tag{29}$$

where m_s is the number of remaining masked positions and $\{\tau_s\}_{s=0}^{T}$ is the reverse denoising schedule. This schedule can spend late denoising steps without revealing any token when the reveal count rounds to zero. We instead use

$$n_s = \left\lceil \frac{m_s}{T - s} \right\rceil, \tag{30}$$

which distributes the remaining masks across the remaining step budget and guarantees progress at each step. We use the same adaptive rule for the LLaDA backend. Holding the rest of the Dream configuration fixed, the adaptive schedule improves average accuracy from 58.17 to 59.73 while reducing the average number of forward passes from 200.9 to 148.9.

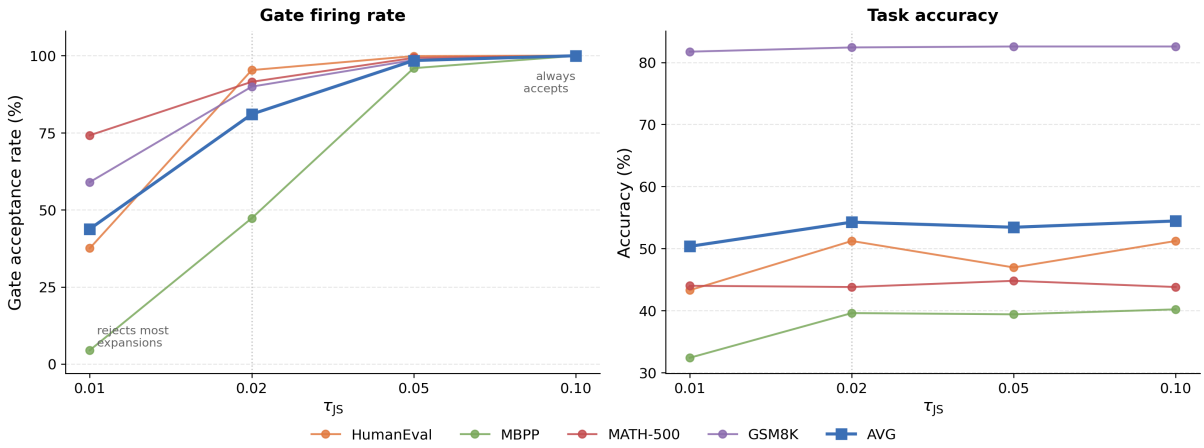

Figure 3: Effect of the JS acceptance threshold τ_{JS} on LLaDA-8B. Left: percentage of accepted expansions. Right: task accuracy under the same thresholds. Very small thresholds reject too many expansions, while large thresholds accept almost all proposals. We use $\tau_{\text{JS}} = 0.02$, which reaches the accuracy plateau while keeping CARVE selective.

Schedule	HE	MBPP	MATH	GSM8K
Original	54.27	**61.20**	38.80	78.39
Adaptive	**59.76**	60.40	**40.20**	**78.54**
Δ	(+5.49)	(−0.80)	(+1.40)	(+0.15)

Table 2: Accuracy ablation of the adaptive reveal schedule on Dream-v0-Instruct-7B. The Δ row reports percentage-point differences relative to the original schedule.

Schedule	HE	MBPP	MATH	GSM8K	Avg.
Original	100.7	151.0	344.7	207.3	200.9
Adaptive	**74.5**	**109.3**	**261.8**	**150.1**	**148.9**

Table 3: Forward-pass ablation of the adaptive reveal schedule on Dream-v0-Instruct-7B. Lower is better; values are averaged per sample.

5.2 Effect of the JS threshold

We ablate the expansion threshold τ_{JS} on the LLaDA-8B setting by sweeping $\tau_{\text{JS}} \in \{0.01, 0.02, 0.05, 0.10\}$. Figure 3 reports both the fraction of accepted expansions and the resulting task accuracy. The threshold controls a clear trade-off. When $\tau_{\text{JS}} = 0.01$, CARVE is overly conservative: many expansions are rejected, especially on MBPP, which coincides with lower accuracy, consistent with the canvas not growing enough. Conversely, larger thresholds such as 0.05 and 0.10 accept nearly all proposals, making the verification step almost vacuous and moving the method toward an always-expand setting.

We use $\tau_{\text{JS}} = 0.02$ as the default because it is the smallest tested threshold at which average accuracy reaches its plateau while CARVE remains selective. At this value, CARVE preserves the average accuracy of more permissive thresholds but still rejects a nontrivial fraction of proposed insertions. This supports the role of the JS divergence as a meaningful stability check rather than a constant expansion rule.

5.3 Insertion Mode

We compare the uncertainty-guided insertion rule used by CARVE with a simpler tail-insertion strategy. Table 4 reports the average score of the best configuration found for each mode and model. Mid-insert performs best on Dream and LLaDA-1.5, suggesting that placing new masks near uncertain regions can help the model allocate capacity where the current canvas is under-specified. Tail insertion, however, remains competitive and is strongest on LLaDA-8B, indicating that the optimal insertion location can depend on the backbone and decoding dynamics. We therefore use mid-insert as the default canvas-aware rule, while treating tail insertion as a strong, simpler alternative.

Model	Mid-insert	Tail
Dream-7B	**59.73**	59.00
LLaDA-1.5	**53.84**	52.88
LLaDA-8B	53.20	**54.29**

Table 4: Insertion-mode ablation. We compare average task accuracy for mid-insert and tail insertion using the best configuration found for each mode and model.

6 Discussion

CARVE expands the canvas only when the proposed insertion produces a small mean distribution shift at aligned unresolved positions. This stability criterion remains permissive enough to allow useful growth in the evaluated settings. Across models, CARVE consistently improves average accuracy over fixed-length baselines while often using less compute than fixed-length decoding.

The results suggest that effective length control for masked diffusion LMs is not only about adding more masks, but about adding them without destabilizing the denoising state. JS divergence provides this check by accepting an expanded branch only when the model's predictions remain stable after insertion. The ablations support this view: too small a threshold rejects useful growth, while too large a threshold makes CARVE nearly equivalent to always expanding.

Finally, the compute results show that the extra forward passes introduced by branching do not necessarily translate into higher total cost. In CARVE, branching is paired with EOS crop: once the model commits an end-of-sequence token, decoding stops on the useful prefix instead of continuing to refine suffix positions that would be discarded. Together with the adaptive reveal schedule, this offsets much of the cost of expansion attempts. As a result, CARVE improves the accuracy–compute trade-off using only inference-time changes, without retraining or modifying the underlying diffusion model.

7 Conclusion

We introduced CARVE (**C**ounterfactual-**A**ware **R**eveal with **V**erified **E**xpansion), a training-free variable-length decoding algorithm for masked diffusion language models. CARVE addresses the fixed-canvas limitation of dLLMs by branching into candidate expanded canvases and accepting an expansion only when it preserves the model's predictions on aligned unresolved positions. This turns length control into a counterfactual stability test rather than a pure confidence or EOS heuristic.

Across code-generation and mathematical-reasoning benchmarks, CARVE improves average performance over fixed-length baselines for each evaluated diffusion LM. Although branching adds forward passes, adaptive reveal and EOS cropping often offset this cost, yielding a better accuracy–compute trade-off than fixed-length decoding. Overall, our results show that pretrained masked diffusion LMs already contain useful signals for safe length adaptation, which can be exploited directly at inference time without retraining or architectural changes.

Limitations

CARVE currently inserts a fixed number of mask tokens at each accepted expansion. In our main configurations, this value is set to $k = 16$. While this works well empirically, it does not adapt to the uncertainty or length requirements of each prompt. A natural direction for future work is to make insertion size adaptive, deciding not only where to expand the canvas but also how many new mask tokens should be added.

A second limitation is the alignment used by the JS divergence computation. CARVE compares predictions only on unresolved positions that already existed before insertion, while the newly inserted mask positions are excluded because they have no direct counterpart in the base canvas. This makes the verification step simple and well-defined, but it may be overly rigid. Future work could explore softer alignment or alternative divergence criteria that also account for the behavior of the newly inserted positions.

References

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. 2021a. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021b. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. 2022. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, 35:28266–28279.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg

Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Zicong Cheng, Ruixuan Jia, Jia Li, Guo-Wei Yang, Meng-Hao Guo, and Shi-Min Hu. 2026. Improving variable-length generation in diffusion language models via length regularization. *arXiv preprint arXiv:2602.07546*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Marton Havasi, Brian Karrer, Itai Gat, and Ricky TQ Chen. 2025. Edit flows: Flow matching with edit operations. *arXiv preprint arXiv:2506.09018*.

Jaeyeon Kim, Lee Cheuk-Kit, Carles Domingo-Enrich, Yilun Du, Sham Kakade, Timothy Ngotiaoco, Sitan Chen, and Michael Albergo. 2025. Any-order flexible length masked diffusion. *arXiv preprint arXiv:2509.01025*.

Jinsong Li, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Jiaqi Wang, and Dahua Lin. 2025. Beyond fixed: Training-free variable-length denoising for diffusion large language models. *arXiv preprint arXiv:2508.00819*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601.

Aaron Lou, Chenlin Meng, and Stefano Ermon. 2023. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*.

Amr Mohamed, Yang Zhang, Michalis Vazirgiannis, and Guokan Shang. 2026. Fast-decoding diffusion language models via progress-aware confidence schedules. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 35793–35807, San Diego, California, United States. Association for Computational Linguistics.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2026. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646.

Vittorio Rossi, Giacomo Ciro, Davide Beltrame, Luca Gandolfi, Paul Rottger, and Dirk Hovy. 2026. Diffusion language models are natively length-aware. *arXiv preprint arXiv:2603.06123*.

Model	Canvas ($L_{\max}$)	Steps	Temp.	Block	k	W	I
Dream-7B	128 / 256 / 512 / 256	$L_{\max}$	0.04	full	16	12	16
LLaDA-1.5	512 / 512 / 512 / 512	$L_{\max}$	0.05	32	16	4	1
LLaDA-8B	512 / 256 / 512 / 512	$L_{\max}$	0.03	64	16	8	1

Table 5: Main hyperparameters used for the evaluations. The canvas & $L_{\max}$ column reports HE / MBPP / MATH / GSM8K. The number of denoising steps is set equal to the task-specific maximum canvas length.

Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. 2024. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. 2024. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167.

Zirui Wu, Lin Zheng, Zhihui Xie, Jiacheng Ye, Jiahui Gao, Shansan Gong, Yansong Feng, Zhenguo Li, Wei Bi, Guorui Zhou, and 1 others. 2026. Dreamon: Diffusion language models for code infilling beyond fixed-size canvas. *arXiv preprint arXiv:2602.01326*.

Jingyi Yang, Yuxian Jiang, and Jing Shao. 2026. ρ-EOS: Training-free bidirectional variable-length control for masked diffusion llms. *arXiv preprint arXiv:2601.22527*.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. 2025. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*.

Andrew Zhang, Anushka Sivakumar, Chia-Wei Tang, and Chris Thomas. 2025. Flexible-length text infilling for discrete diffusion models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 31332–31347.

Lin Zheng, Jianbo Yuan, Lei Yu, and Lingpeng Kong. 2023. A reparameterized discrete diffusion model for text generation. *arXiv preprint arXiv:2302.05737*.

A Experimental Settings

Table 5 summarizes the task-specific decoding configurations for each model. For every model–benchmark pair, we initialize the canvas with $L_0 = L_{\max}/2$ masked positions and set the denoising budget to $T = L_{\max}$. The table also reports the sampling temperature, block size, insertion size k, uncertainty-window size W, and expansion interval I.

B Hardware

All experiments used AMD MI210 GPUs and consumed approximately 42 aggregate GPU-days. Runs used at most eight GPUs concurrently.

C Isolating the Contribution of Each Component

We decompose CARVE into its main components and evaluate controlled variants of the decoder. For each model and benchmark, all rows use the same prompts, maximum canvas length, reveal rule, and evaluation metric. The rows differ only in the decoding components enabled.

Configurations. *Fixed baseline* denotes standard fixed-length decoding at $L_{\max}$. + *adaptive reveal* uses the full canvas at $L_{\max}$ with the adaptive reveal rule, but without expansion or EOS cropping. *Full canvas* + *EOS-crop* adds EOS cropping to the previous setting. *Fixed-L_0* + *EOS-crop* fixes the canvas at $L_0 = L_{\max}/2$, disables expansion, and uses EOS cropping. *Always-expand* inserts new masks every I steps unconditionally, with the verification forward removed; reveal decisions are therefore based on the pre-insertion forward. *CARVE* is the full method.

Discussion. These controlled ablations show that the gains of CARVE do not come from a single independent shortcut. Adding adaptive reveal on top of vanilla decoding leaves performance unchanged, as expected: with a fixed canvas, both use the same position-ranking criterion but different reveal-count schedules. Its main role is to make a growing canvas usable: once new masks are inserted, the decoder needs a reveal schedule that can keep pace with the changing number of unresolved positions.

EOS cropping behaves differently. It is primarily an efficiency mechanism: on a full canvas, it approximately preserves accuracy while reducing computation spent on suffix positions that would be discarded after EOS. However, cropping alone is not enough. When the canvas is fixed at $L_0 = L_{\max}/2$ and cannot grow, the decoder becomes much cheaper but loses substantial accuracy, showing that a small initial canvas must be paired with a mechanism for allocating additional space.

The always-expand variant further clarifies the role of verification. On the LLaDA backends, removing the verification forward pass reduces accuracy relative to CARVE while saving only a small amount of compute. On Dream, the two variants are close. This suggests that unconditional growth can sometimes be sufficient, but is not a reliable replacement for verified expansion across backends. Moreover, this control removes not only the accept/reject decision, but also the forward pass from which an accepted expanded branch reveals its tokens. Without that pass, tokens are revealed from predictions computed before the canvas was enlarged.

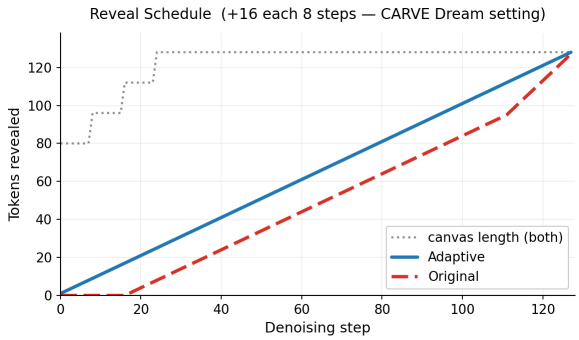

Figure 4: Reveal trace under a growing canvas in the Dream setting. The canvas grows by inserting 16 masks every 8 denoising steps until reaching $L_{\max} = 128$. The original schedule, designed for a fixed canvas, initially reveals no tokens, then reveals many tokens late in the trajectory. The adaptive rule keeps reveal progress synchronized with the growing canvas.

D Adaptive Reveal Under a Growing Canvas

Appendix C shows that the adaptive reveal rule is not a major source of fixed-canvas accuracy gains on a fixed canvas. Its role in CARVE is operational: when the canvas grows, newly inserted masks must also be revealed within the remaining denoising budget.

Dream's original schedule reveals

$$n_s = \left\lfloor M_s \left(1 - \frac{\tau_{s+1}}{\tau_s}\right) \right\rfloor,$$

where M_s is the number of remaining masked positions and $\{t_s\}_{s=0}^{T}$ is the denoising time grid. This value can round to zero, causing a model forward pass to reveal no tokens. CARVE instead uses

$$n_s = \left\lceil \frac{M_s}{T - s} \right\rceil,$$

which distributes the remaining masked positions across the remaining denoising steps and guarantees at least one reveal per step while masks remain.

Model	Configuration	HumanEval	MBPP	GSM8K	MATH-500	Avg.	FLOPs×
Dream	Fixed baseline	55.49	60.60	79.53	39.60	58.80	1.00
	+ adaptive reveal	55.49	60.60	79.53	39.60	58.80	1.00
	Full canvas + EOS-crop	55.49	60.60	79.53	39.60	58.80	0.49
	Fixed-L_0 + EOS-crop	46.34	59.00	68.39	38.40	53.03	0.32
	Always-expand	59.15	61.40	78.39	40.40	59.84	0.48
	CARVE	59.76	60.40	78.54	40.20	59.73	0.51
LLaDA-8B	Fixed baseline	46.95	38.80	81.88	43.20	52.71	1.00
	+ adaptive reveal	46.95	38.80	81.88	43.20	52.71	1.00
	Full canvas + EOS-crop	46.95	38.80	81.88	43.20	52.71	0.83
	Fixed-L_0 + EOS-crop	35.98	30.20	81.58	39.60	46.84	0.31
	Always-expand	45.73	39.00	82.56	42.20	52.37	0.82
	CARVE	48.17	39.20	82.03	43.40	53.20	0.85
LLaDA-1.5	Fixed baseline	46.95	38.20	83.09	43.00	52.81	1.00
	+ adaptive reveal	46.95	38.20	83.09	43.00	52.81	1.00
	Full canvas + EOS-crop	46.95	38.20	83.09	43.20	52.86	0.77
	Fixed-L_0 + EOS-crop	38.41	39.40	81.80	39.00	49.65	0.30
	Always-expand	48.17	37.60	83.93	43.80	53.38	0.75
	CARVE	50.61	37.80	83.55	43.40	53.84	0.78

Table 6: Component-wise ablation across the three evaluated backends. FLOPs are normalized by the fixed-length baseline for the same model and benchmark. The "+ adaptive reveal", "Full canvas + EOS-crop", and "Fixed-L_0 + EOS-crop" rows use no expansion. The "Always-expand" row removes the verification forward pass and inserts unconditionally.

Figure 4 illustrates the mismatch between a fixed-canvas reveal schedule and a growing canvas. Under the original rule, the decoder spends early steps expanding the canvas without committing tokens, then has to reveal the remaining masks late in the trajectory. The adaptive rule avoids this stall by keeping the number of remaining masks aligned with the remaining step budget.

This same effect is reflected in Table 3: the adaptive rule uses substantially fewer forward passes on Dream because it prevents reveal stalls under a growing canvas, reducing the average number of forward passes per sample from 200.9 to 148.9. Given this efficiency gain, we use the adaptive reveal rule by default for the LLaDA backends as well.

E Cost When the Response Fills the Canvas

The average FLOPs reduction of CARVE partly comes from EOS cropping, which shortens the effective decoding trajectory. The least favorable case is therefore a response that fills the canvas, where cropping provides little or no benefit and the expansion overhead remains. To isolate this setting, we select the 5% longest responses for each benchmark, i.e., examples that fill 99–100% of the canvas, and compare CARVE against the fixed-length baseline on the same examples.

Benchmark	Dream	LLaDA-8B	LLaDA-1.5
HumanEval	1.01× (99%)	1.02× (100%)	1.02× (100%)
MBPP	1.01× (100%)	1.04× (100%)	1.02× (100%)
GSM8K	1.01× (100%)	1.02× (100%)	1.01× (99%)
MATH-500	0.98× (100%)	1.02× (100%)	1.02× (100%)

Table 7: FLOPs ratio of CARVE relative to the fixed-length baseline on the 5% longest responses per benchmark. Parentheses report the percentage of the canvas filled by the decoded response.

As shown in Table 7, even in this adverse regime the overhead remains small. The largest observed cost is $1.04\times$ the fixed baseline, and most settings are within two percent of the baseline. This overhead is bounded by construction: expansion is attempted at most once every I decoding steps and can therefore add at most $\lceil T/I \rceil$ expanded-branch forward passes over a T-step trajectory. Rejected proposals still incur this additional forward pass. EOS-triggered completion can add at most one further forward pass.

F Wall-Clock Throughput and Peak Memory

FLOPs provide a hardware-independent proxy for inference cost. We additionally report wall-clock throughput and peak device memory on 8×AMD MI210 GPUs. Both quantities are measured as ratios of CARVE to the fixed-length baseline.

The throughput results broadly follow the

Benchmark	Dream	LLaDA-8B	LLaDA-1.5
HumanEval	1.52	0.93	0.97
MBPP	3.10	0.97	1.32
GSM8K	1.75	1.63	1.60
MATH-500	2.29	1.12	1.09

Table 8: Throughput ratio of CARVE relative to the fixed-length baseline, measured in tokens per second. Higher is better.

Benchmark	Dream	LLaDA-8B	LLaDA-1.5
HumanEval	0.99	1.01	1.01
MBPP	1.00	1.01	1.01
GSM8K	1.01	1.01	1.01
MATH-500	0.96	1.01	1.01

Table 9: Peak memory remains essentially unchanged empirically, with ratios between 0.96 and 1.01 in the evaluated settings.

FLOPs trends. CARVE is consistently faster on Dream and improves throughput on most LLaDA settings, with a few cases slightly below the fixed baseline when responses are long and cropping cannot offset the expansion forward passes. Peak memory remains essentially unchanged: CARVE never exceeds the same maximum canvas length used by the fixed-length decoder, and differs only in how that canvas is reached during decoding.

G Qualitative Examples

Figures 5–8 show qualitative examples from HumanEval and GSM8K using Dream-7B.

Prompt	`from typing import List`
	
	`def all_prefixes(string: str) -> List[str]:`
	`  """ Return list of all prefixes from shortest to longest of the input string`
	`  >>> all_prefixes('abc')`
	`  ['a', 'ab', 'abc']`
	`  """`

Vanilla ✗
128 fwds
11.95 s
len = 128
	`  prefixes = []`
	`  for i in range(len(string) + 1):`
	`  prefixes.append(string[:i])`
	`  return prefixes`

DAEDAL ✓
46 fwds
4.91 s
len = 256
	`prefixes = []`
	
	`  for i in range(len(string)):`
	`  prefixes.append(string[:i+1])`
	
	`  return prefixes`

CARVE ✓
30 fwds
3.44 s
len = 96
	`prefixes = []`
	`  for i in range(len(string)):`
	`  prefixes.append(string[:i + 1])`
	`  return prefixes`

Figure 5: Example HumanEval sample for Dream-7B methods (`HumanEval_14`). *fwds* is the number of model forward passes, *len* the peak canvas length reached during decoding.

Prompt

```
def anti_shuffle(s):
    """
    Write a function that takes a string and returns an ordered version of it.
    Ordered version of string, is a string where all words (separated by space)
    are replaced by a new word where all the characters arranged in
    ascending order based on ascii value.
    Note: You should keep the order of words and blank spaces in the sentence.

    For example:
    anti_shuffle('Hi') returns 'Hi'
    anti_shuffle('hello') returns 'ehllo'
    anti_shuffle('Hello World!!!') returns 'Hello !!!Wdlor'
    """
```

Vanilla ×
128 fwds
11.51 s
len = 128

```
    words = s.split()
    ordered_words = []
    for word in words:
        ordered_word = ''.join(sorted(word))
        ordered_words.append(ordered_word)
    return ' '.join(ordered_words)
```

```

You can test this function with the provided examples:
```python
assert anti_shuffle('Hi') == 'Hi'
assert anti_shuffle('hello') == 'ehllo'
assert anti_shuffle('Hello World!!!') == 'Hello !!!Wdlor'
```

DAEDAL ✓
137 fwds
15.27 s
len = 280

```
words = s.split()
    sorted_words = []

    for word in words:
        sorted_word = ''.join(sorted(word))
        sorted_words.append(sorted_word)

    return ' '.join(sorted_words)
```

```

The function ‘anti_shuffle‘ takes a string ‘s‘ as input and returns an ordered version
of it.
    It splits the string into words using the ‘split‘ method, and then iterates over
each word
    to sort its characters in ascending order using the ‘sorted‘ function. The sorted
words are
    then joined back together with spaces to form the final string.
```

CARVE ✓
48 fwds
4.40 s
len = 112

```
words = s.split()
    ordered_words = []
    for word in words:
        ordered_word = ''.join(sorted(word))
        ordered_words.append(ordered_word)
    return ' '.join(ordered_words)
```

Figure 6: Example HumanEval sample for Dream-7B methods (HumanEval_86). *fwds* is the number of model forward passes, *len* the peak canvas length reached during decoding.
```

Question	*John decides to do several activities while out on vacation. He spends 6 hours boating and half that time swimming. He also watched 3 different shows which were 2 hours each. This was 30% of the time he spent.* *He spent 40% of his time sightseeing. How much time did he spend sightseeing?*
Answer	20
Vanilla × 256 fwds 69.01 s len = 256	He spent 6 hours boating so he spent 6 / 2 = <<6/2=3>>3 hours swimming. So he spent 6 + 3 = <<6+3=9>>9 hours boating and swimming. So he spent 9 + 6 = <<9+6=15>>15 hours on those activities. So he spent 15 / 30 = <<15/30=0.5>>0.5 of his time on those activities. So he spent 1 / 0.5 = <<1/0.5=2>>2 hours on sightseeing. So he spent 2 / 40 = <<2/40=0.05>>0.05 of his time sightseeing. So he spent 1 / 0.05 = <<1/0.05=20>>20 hours on sightseeing. So he spent 20 * 40 = <<20*40=800>>800 minutes sightseeing. So he spent 800 / 60 = <<800/60=13>>13 hours sightseeing. #### 13
DAEDAL × 158 fwds 46.00 s len = 296	He spent 6 hours boating so he spent 6 / 2 = <<6/2=3>>3 hours swimming. So he spent 6 + 3 = <<6+3=9>>9 hours boating and swimming. So he spent 3 * 2 = <<3*2=6>>6 hours watching shows. So he spent 9 + 6 = <<9+6=15>>15 hours on those activities. So he spent 100 - 30 = <<100-30=70>>70% of his time on those activities. So he spent 15 / .7 = <<15/.7=21.428571428571428>>21.428571428571428 hours on his vacation. So he spent 21.428571428571428 * .4 = <<21.428571428571428*.4=8.571428571428571>>8.571428571428571 hours sightseeing. #### 8.57
CARVE ✓ 146 fwds 38.41 s len = 256	He spent 6 * 0.5 = <<6*0.5=3>>3 hours swimming. So he spent 6 + 3 = <<6+3=9>>9 hours boating and swimming. So he spent 9 + 2 * 3 = <<9+2*3=15>>15 hours on those activities. So he spent 15 / 0.3 = <<15/0.3=50>>50 hours on vacation. So he spent 50 * 0.4 = <<50*0.4=20>>20 hours sightseeing. #### 20

Figure 7: Example GSM8K sample for Dream-7B methods (GSM8K_0353). *fwds* is the number of model forward passes, *len* the peak canvas length reached during decoding.

Question	*Mike was a pen pal with 5 people. He stopped being penpals with 2 of them. They each send 2 letters a week that are 5 pages long. He responds in kind. He can write a page every 6 minutes. How many hours does he spend writing a week?*
Answer	3
Vanilla × 256 fwds 68.80 s len = 256	He stopped being penpals with 2 people so he is now penpals with 5 - 2 = <<5-2=3>>3 people. They each send 2 letters a week that are 5 pages long so he gets 2 * 5 = <<2*5=10>>10 pages a week. He responds in kind so he writes 10 pages a week. He can write a page every 6 minutes so he writes 10 * 6 = <<10*6=60>>60 minutes a week. There are 60 minutes in an hour so he writes 60 / 60 = <<60/60=1>>1 hour a week. #### 1
DAEDAL × 119 fwds 34.59 s len = 320	He stopped being penpals with 2 people so he is now penpals with 5 - 2 = <<5-2=3>>3 people. They each send 2 letters a week that are 5 pages long so that is 2 * 5 = <<2*5=10>>10 pages per person. He responds in kind so he writes 10 * 3 = <<10*3=30>>30 pages a week. He can write a page every 6 minutes so he spends 30 * 6 = <<30*6=180>>180 minutes a week. #### 180
CARVE ✓ 129 fwds 33.72 s len = 256	He stopped being penpals with 5 - 2 = <<5-2=3>>3 people. He writes 3 * 2 = <<3*2=6>>6 letters a week. He writes 6 * 5 = <<6*5=30>>30 pages a week. He writes 30 * 6 = <<30*6=180>>180 minutes a week. He writes 180 / 60 = <<180/60=3>>3 hours a week. #### 3

Figure 8: Example GSM8K sample for Dream-7B methods (GSM8K_0242). *fwds* is the number of model forward passes, *len* the peak canvas length reached during decoding.