RETHINKING WORKSPACE CONTROL FOR OPEN-ENDED DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Masked diffusion language models (dLLMs) generate text by resolving masked positions in parallel, but open-ended generation remains challenging because the final output length is unknown in advance. Existing methods typically rely on model-derived signals such as EOS statistics, confidence, or prediction density to dynamically adjust the workspace, introducing additional estimation and control complexity. We find that, with the decoding state fixed and these control signals bypassed, directly changing future `[MASK]` exposure can substantially alter current decoding decisions. This suggests a simpler principle: rather than repeatedly estimating workspace demand from model outputs, future workspace can be organized directly around decoding progress. We therefore introduce **MaskHorizon**, a framework for open-ended dLLM decoding. MaskHorizon maintains a local horizon of future `[MASK]` positions that advances with block-level decoding progress. Each forward pass predicts over the workspace up to this horizon, while commitment remains restricted to the active block. Experiments on LLaDA and Dream across reasoning and code-generation tasks show that MaskHorizon substantially reduces decoding computation while retaining competitive generation quality. The code for MaskHorizon is available at `https://anonymous.4open.science/r/MaskHorizon/`.

1 INTRODUCTION

Masked diffusion language models (dLLMs) generate text by iteratively resolving masked positions, allowing multiple tokens to be predicted and committed within a single model evaluation (Nie et al., 2025; Ye et al., 2025). Recent advances in parallel decoding, cache reuse, and adaptive decoding have substantially improved the efficiency of this generation paradigm (Wu et al., 2026b; Lu et al., 2026), making dLLMs increasingly capable on complex reasoning and code-generation tasks. Open-ended generation, however, still poses a basic challenge: the final output length is generally unknown in advance. A small workspace may constrain long responses, whereas allocating a large workspace from the beginning repeatedly processes positions that are not yet needed. This raises a central question for open-ended dLLM decoding: **how should the workspace evolve as decoding proceeds?**

Existing training-free variable-length methods typically adjust the workspace using model-derived signals such as EOS statistics, confidence, or prediction density (Li et al., 2026a; Yang et al., 2026). Translating these signals into workspace changes still requires additional thresholds, planning, or insertion rules, increasing algorithmic and computational complexity. Despite their different designs, these methods ultimately act through the same object: the unresolved workspace exposed to the model. This raises a basic question: **do their benefits come primarily from sophisticated signal-to-workspace decisions, or from the workspace changes themselves?**

Changing the workspace necessarily changes subsequent model inputs; the more relevant question is whether workspace exposure itself has a sufficiently strong effect on actual decoding decisions, even without the original signal-based control process. To isolate this effect, we conduct a matched-state intervention. Starting from the same intermediate decoding state, we construct two continuations with identical generated content, decoding frontier, and active block, while varying only the number of exposed future `[MASK]` positions. As shown in Fig. 1(a), this intervention changes 47.5%–60.9% of active-block top-1 predictions across four tasks and alters 94.7%–100% of commitment

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

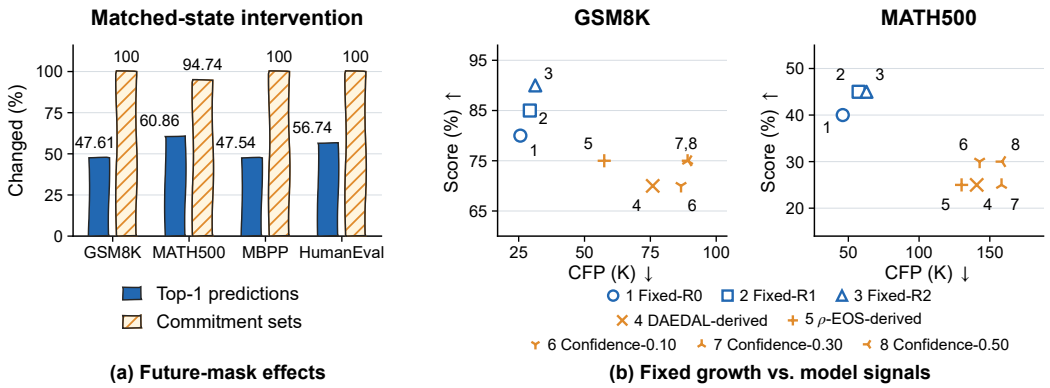


Figure 1: **Motivating workspace control.** (a) Future masks alter current predictions. (b) Fixed versus signal-derived growth under the same append-only interface. Score (%) and CFP (K); descriptive pilot protocols in Appendix B.2.

sets. **Workspace exposure itself therefore acts as a strong control variable**, motivating a further question: is sophisticated model-derived control actually necessary to obtain an effective workspace schedule?

To examine this question, we standardize the basic growth operation and vary only the growth rule. All methods use the same initial workspace, commitment rule, stopping condition, and allocation cap, with every growth action appending new `[MASK]` positions at the right boundary. Under this shared interface, we compare DAEDAL-derived, ρ-EOS-derived, and confidence-based controllers with several simple rules that read no online model signal and maintain different local reserves. As shown in Fig. 1(b), on the two fixed evaluation cohorts, these simple rules achieve higher observed accuracy while processing fewer cumulative positions than the tested signal-based controllers. Different reserve sizes still exhibit a clear quality–computation trade-off and are not uniformly better when enlarged. **An effective workspace trajectory therefore need not rely on repeated model-state estimation.**

Based on this analysis, we introduce **MaskHorizon**, a training-free open-ended decoder based on progress-driven local workspace control. MaskHorizon uses block completion as a synchronization point and maintains a local reserve of future `[MASK]` positions beyond the active block. As the decoding frontier advances, the required workspace boundary advances with it. When the current allocation no longer satisfies the reserve requirement, MaskHorizon appends additional `[MASK]` positions at the right boundary and continues decoding over the enlarged workspace. These newly exposed positions participate in subsequent predictions but remain masked until their block becomes active. The moving reserve boundary thus forms a dynamic *horizon* that advances with the decoding process.

This exposure–commitment separation induces a useful structural invariant: at every growth boundary, a contiguous resolved prefix is followed by an all-`[MASK]` suffix. Because future positions remain unresolved until their block becomes active, this structure is preserved throughout decoding. Under this invariant, for any fixed net growth amount, inserting the same number of `[MASK]` positions anywhere within the masked suffix produces the same token sequence as appending them at the right boundary. There is therefore no need to explicitly select an insertion position. Once the reserve rule determines that additional workspace is required, the update can be implemented directly by right-boundary append. A formal proof of this equivalence is provided in Appendix A.2. Our contributions are threefold:

- **Workspace-control analysis.** We isolate workspace exposure from model-derived growth signals through matched-state interventions and shared-interface comparisons, showing that, on the tested cohorts, simple signal-free reserve rules can achieve higher observed accuracy with fewer cumulative forward positions.

- **MaskHorizon.** We introduce a training-free decoder that advances a local horizon of future [MASK] positions with block-level decoding progress. Its active-block commitment preserves an all-[MASK] suffix, making insertion-location search unnecessary.
- **Systematic evaluation.** Across LLaDA and Dream on reasoning and code-generation tasks, MaskHorizon consistently reduces decoding computation. Controlled studies further characterize the effects of workspace exposure, reserve size, commitment scope, and growth granularity.

2 Related Work

Efficient dLLM Decoding. Recent work has substantially improved diffusion language model inference by reducing either the number or the cost of iterative denoising steps. One line accelerates token resolution through confidence-aware parallel decoding, adaptive block sizes, and redundancy-aware commitment (Wu et al., 2026b; Lu et al., 2026; Wu et al., 2026c; Du et al., 2026). Another line reduces per-step computation through cache reuse, sparse attention, selective token computation, or early refinement termination (Wu et al., 2026a; Song et al., 2026; Long et al., 2026; Liang et al., 2026; Zou et al., 2026; Li et al., 2026b). These methods mainly optimize how efficiently an existing generation space is processed or resolved, while leaving the space itself largely fixed. Open-ended generation raises a complementary question: how should that space evolve when the required output length is unknown?

Variable-Length dLLMs. Recent work relaxes the fixed-space assumption through length estimation, EOS-driven expansion or contraction, confidence regularization, or structural inference (Li et al., 2026a; Yang et al., 2026; Cheng et al., 2026; Wu et al., 2026d; Sun et al., 2026). Other approaches incorporate flexible length directly into generation through mask insertion or deletion–insertion mechanisms (Kim et al., 2026; Zhang et al., 2025; Ding et al., 2026; Nie et al., 2026). Despite their different signals and operations, these methods generally infer workspace adjustments from the current model state. MaskHorizon instead organizes future workspace directly with block-level decoding progress and a local reserve, without repeatedly estimating workspace insufficiency from model outputs. This shifts the focus from predicting when space is inadequate to controlling how unresolved space is exposed as decoding proceeds.

3 MaskHorizon

MaskHorizon organizes open-ended decoding by controlling how future unresolved space is exposed as generation proceeds. At each block boundary, it maintains a local reserve beyond the active block; the full exposed workspace participates in prediction, while token commitment remains restricted to the active block. Block completion then advances the decoding frontier and triggers the next workspace update. Figure 2 illustrates this cycle: local-reserve growth, exposed-workspace prediction, and active-block parallel resolution. Algorithmic details and formal properties are provided in Appendix A.

Workspace state. Let $\mathbf{x} = (x_0, \ldots, x_{P-1})$ be a fixed prompt and $\mathbf{z}_t$ the generation workspace at decoding iteration t, containing resolved tokens and [MASK] positions. Here t indexes model evaluations rather than continuous diffusion time. Let N denote the current exposed workspace length, capped by $N_{\max}$, and let decoding start from N_0 [MASK] positions, where $0 < N_0 \leq N_{\max}$.

The frontier f marks the end of all fully resolved blocks. For block size B, the active block is

$$\mathcal{A}(f) = \{f, \ldots, e-1\}, \qquad e = \min(f + B, N_{\max}). \tag{1}$$

Before this block is resolved, MaskHorizon maintains $N \geq N_{\text{req}}(f) \geq e$, where N_{req} is defined in Section 3.1. The workspace is partitioned as

$$\underbrace{[0, f)}_{\text{resolved}} \mid \underbrace{[f, e)}_{\text{active}} \mid \underbrace{[e, N)}_{\text{future space}} \mid \underbrace{[N, N_{\max})}_{\text{unallocated}}.$$

Positions in $[0, N)$ are exposed to the model, whereas positions in $[N, N_{\max})$ are not model inputs. Future space $[e, N)$ remains masked until it becomes active. Exposure therefore determines which

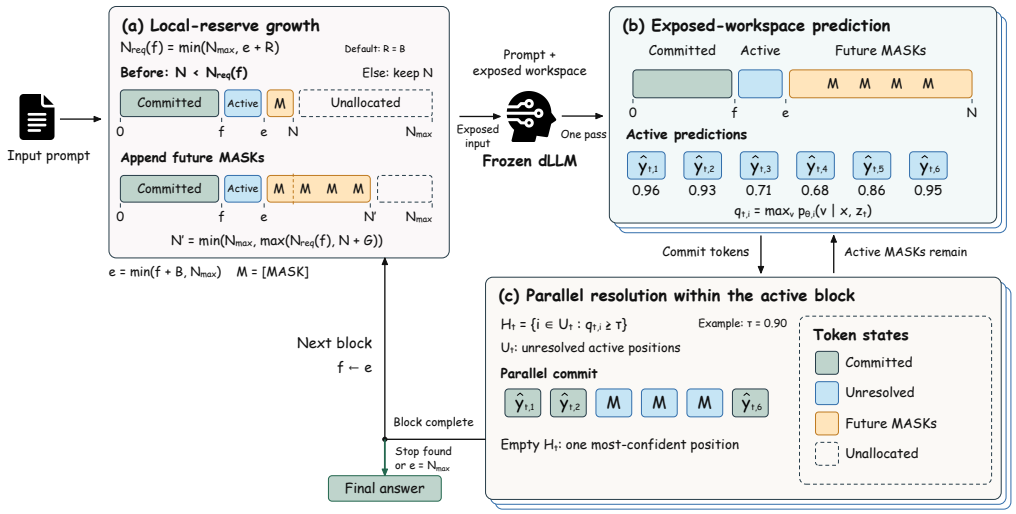


Figure 2: **MaskHorizon framework.** (a) Local-reserve growth exposes additional workspace when the current allocation falls behind the block-level requirement. (b) Exposed-workspace prediction uses the full exposed workspace. (c) Parallel resolution within the active block. Block completion advances the frontier and starts the next workspace update.

positions enter a forward pass, while commitment determines which predictions are written into the workspace.

3.1 Local-Reserve Workspace Growth

MaskHorizon ties workspace demand directly to the block frontier. Let R denote the minimum reserve maintained beyond the active block. At each block entry, the required workspace boundary is

$$N_{\mathrm{req}}(f) = \min(N_{\max},\, e + R)\,. \tag{2}$$

If $N < N_{\mathrm{req}}(f)$, the workspace grows to

$$N' = \min(N_{\max}, \max(N_{\mathrm{req}}(f),\, N + G))\,, \tag{3}$$

where G is the growth increment. The newly exposed interval $[N, N')$ is initialized with `[MASK]` positions; otherwise, the allocation remains unchanged.

The two quantities play different roles: R specifies the minimum future space required at the current frontier, whereas G controls the granularity of allocation. We use $R = B$ and $G = 2B$ by default, while B, R, G, and N_0 remain general configuration variables. Growth depends only on the frontier and current allocation, without EOS probabilities, confidence, prediction stability, or verification passes. Confidence can affect when the active block finishes, but is not an input to the growth rule.

The reserve is a lower bound rather than an exact exposed width: initialization and chunked growth may leave more than R future positions, while the allocation cap may leave fewer. Allocation is monotone and remains fixed within each block. For a fixed frontier sequence, the chunked rule stays within one growth increment of the minimum monotone allocation satisfying the same reserve constraint; Appendix A gives the formal bound.

3.2 Exposed-Workspace Prediction

Once the local reserve is exposed, it becomes part of the context for the current block. MaskHorizon therefore separates prediction scope from commitment scope: every model evaluation uses the full exposed workspace, while only unresolved positions in the active block can later be committed.

At forward pass t, the frozen dLLM processes $[\mathbf{x}; \mathbf{z}_t]$, including all N_t exposed generation positions. Let

$$\mathcal{U}_t = \{i \in \mathcal{A}(f) : z_{t,i} = \texttt{[MASK]}\}$$

denote the unresolved active positions. For each $i \in \mathcal{U}_t$, define

$$\hat{y}_{t,i} = \arg\max_v p_{\theta,i}(v \mid \mathbf{x}, \mathbf{z}_t), \qquad q_{t,i} = \max_v p_{\theta,i}(v \mid \mathbf{x}, \mathbf{z}_t). \tag{4}$$

Predictions outside $\mathcal{U}_t$ are not written into the workspace: the resolved prefix remains unchanged and future positions remain masked. Thus, the block boundary restricts commitment rather than prediction. MaskHorizon retains the native parallel masked-completion behavior of the dLLM while using the active block only as a local resolution boundary. Model-specific logit alignment is described in Appendix B.1.

3.3 Parallel Resolution within the Active Block

The active block specifies which positions may be resolved, but introduces no ordering among them. Since a single dLLM forward pass already predicts all unresolved active positions, MaskHorizon commits sufficiently confident positions together rather than imposing an artificial left-to-right order. Using the standard confidence criterion for parallel dLLM decoding, we define

$$\mathcal{H}_t = \{i \in \mathcal{U}_t : q_{t,i} \geq \tau\}. \tag{5}$$

If $\mathcal{H}_t$ is nonempty, all positions in $\mathcal{H}_t$ are committed simultaneously. Otherwise, the most confident unresolved position is selected to ensure progress:

$$\mathcal{S}_t = \begin{cases} \mathcal{H}_t, & \mathcal{H}_t \neq \varnothing, \\ \{i_t^\star\}, & \mathcal{H}_t = \varnothing, \end{cases} \qquad i_t^\star \in \arg\max_{i \in \mathcal{U}_t} q_{t,i}. \tag{6}$$

Selected positions are updated simultaneously, $z_{t+1,i} = \hat{y}_{t,i}$ for $i \in \mathcal{S}_t$, while all other positions remain unchanged. Prediction and commitment repeat until the active block is fully resolved.

Block completion closes one decoding cycle and provides the synchronization point for workspace growth. The decoder first checks for a recognized stop; otherwise, MaskHorizon advances the frontier, $f \leftarrow e$, and updates the reserve requirement for the next block.

Restricting commitment to the active block also induces an important structural property. At every growth boundary, the exposed sequence consists of a contiguous resolved prefix followed by an all-`[MASK]` suffix. For a fixed net growth amount, inserting new masks anywhere inside this suffix therefore produces the same token sequence as appending them at the right boundary. Insertion location is consequently not an independent control variable: MaskHorizon only needs to decide **when** and **how much** to grow. The formal equivalence is proved in Appendix A.2.

4 Experiments

We evaluate MaskHorizon from both overall performance and design behavior. We first compare its quality and decoding cost with existing fixed- and variable-length decoders across two dLLM backbones and four reasoning and code-generation tasks (RQ1). We then isolate the two central design choices: how much future workspace should be exposed during decoding (RQ2), and why commitment should remain local to the active block (RQ3). Additional controlled comparisons examine workspace-growth policies and block granularity, while the final analysis relates reductions in model evaluations and exposed workspace to practical decoding efficiency (RQ4).

4.1 Experimental Setup

Models and Data. We evaluate LLaDA-8B-Instruct (Nie et al., 2025) and Dream-v0-Instruct-7B (Ye et al., 2025) on 100 test examples each from GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021; Lightman et al., 2023), and MBPP (Austin et al., 2021), and all 164 HumanEval examples (Chen et al., 2021). All evaluations are test-time decoding with frozen models. Baselines are Vanilla, Fast-dLLM (Wu et al., 2026b), AdaBlock-dLLM (Lu et al., 2026), and DAEDAL (Li et al., 2026a); DAEDAL-port* denotes our Dream adaptation.

Table 1: **Main results on LLaDA-8B-Instruct.** Blue parentheses show NFE reduction factors relative to Vanilla. Best and second-best Score, NFE, and CFP are bold and underlined. Highest utilization within each dataset is also bold.

		Quality	Decoding cost		Workspace		
Dataset	**Method**	**Score↑**	**NFE↓**	**CFP (K)↓**	E	N	Util. (%)↑
GSM8K	Vanilla	81.00	768.00 (1.00×)	701.94	273.64	768.00	35.63
	Fast-dLLM	82.00	<u>114.75</u> (6.69×)	105.14	274.22	768.00	35.71
	AdaBlock-dLLM	81.00	116.76 (6.58×)	106.97	268.30	768.00	34.93
	DAEDAL	<u>85.00</u>	119.59 (6.42×)	<u>53.91</u>	231.11	322.59	71.64
	MaskHorizon	**87.00**	**77.51** (9.91×)	**27.66**	192.39	255.36	**75.34**
MATH500	Vanilla	44.00	768.00 (1.00×)	703.15	483.76	768.00	62.99
	Fast-dLLM	<u>46.00</u>	<u>169.77</u> (4.52×)	156.58	481.78	768.00	62.73
	AdaBlock-dLLM	**49.00**	178.36 (4.31×)	164.32	484.54	768.00	63.09
	DAEDAL	43.00	207.58 (3.70×)	<u>144.65</u>	426.97	546.96	78.06
	MaskHorizon	42.00	**121.63** (6.31×)	**52.78**	290.35	358.40	**81.01**
MBPP	Vanilla	38.00	768.00 (1.00×)	734.12	333.15	768.00	43.38
	Fast-dLLM	38.00	<u>136.32</u> (5.63×)	130.70	336.04	768.00	43.76
	AdaBlock-dLLM	38.00	141.45 (5.43×)	135.76	322.07	768.00	41.94
	DAEDAL	<u>42.00</u>	174.05 (4.41×)	<u>117.89</u>	300.86	538.80	55.84
	MaskHorizon	**43.00**	**97.34** (7.89×)	**39.92**	209.90	273.28	**76.81**
HumanEval	Vanilla	45.12	768.00 (1.00×)	776.83	604.62	768.00	78.73
	Fast-dLLM	44.51	<u>209.76</u> (3.66×)	<u>213.52</u>	610.46	768.00	79.49
	AdaBlock-dLLM	**46.34**	221.60 (3.47×)	225.75	601.48	768.00	78.32
	DAEDAL	<u>45.73</u>	281.82 (2.73×)	251.16	569.38	728.29	78.18
	MaskHorizon	43.29	**144.90** (5.30×)	**78.07**	333.62	396.10	**84.23**

Decoding Configuration. We use greedy token selection, BF16, batch size one, no KV cache, and $N_{\max} = 768$. MaskHorizon uses $B = R = 32$, $G = 64$, and $\tau = 0.90$, with $N_0 = 192/256/192/224$ in task order. The granularity sweep preserves $R = B$ and $G = 2B$. Full experimental protocols and reproduction details are provided in Appendix B.

Metrics. Score is accuracy for mathematics and pass@1 for code, both in percent. We report mean forward evaluations (NFE $= T$), cumulative forward positions CFP $= \sum_{t=1}^{T}(P + N_t)$ in thousands, effective output length E, final workspace N, and utilization. Here P is prompt length and N_t is exposed workspace at forward t. CFP measures processed positions, not FLOPs.

4.2 Overall Effectiveness (RQ1)

Tables 1 and 2 compare the quality–cost trade-off across both backbones and all four tasks. On LLaDA, MaskHorizon reduces NFE by 35.19–48.59% and CFP by 48.69–68.91% relative to DAEDAL, while also using smaller final workspaces on every task. On Dream, it reduces NFE by 13.16–23.34% and CFP by 27.47–33.16% relative to DAEDAL-port, again with smaller final workspaces throughout. Across all eight model–task pairs, MaskHorizon consistently lowers both NFE and CFP, with higher scores in five cases and lower scores in three. The larger reduction in CFP than in NFE shows that the efficiency gain comes not only from fewer model evaluations, but also from processing a more compact exposed workspace at each stage of decoding. This pattern is consistent with the intended role of MaskHorizon: exposing future capacity progressively rather than carrying a large workspace throughout generation, while keeping the resulting quality changes task-dependent.

4.3 Future Workspace Exposure (RQ2)

We first examine whether future workspace affects current decoding decisions, and how much future space needs to be exposed. At identical intermediate states, appending only future `[MASK]` positions changes the parallel commitment set in 99.22% of eligible LLaDA states and 95.41% of eligible Dream states (Appendix C.1), extending the pilot in Figure 1(a). This intervention isolates

Table 2: **Main results on Dream-v0-Instruct-7B.** Notation follows Table 1. The anomalous Fast-dLLM reproduction is reported separately in Appendix B.3. *DAEDAL adapted to Dream.

		Quality	Decoding cost		Workspace		
Dataset	**Method**	**Score↑**	**NFE↓**	**CFP (K)↓**	E	N	Util. (%)↑
GSM8K	Vanilla	**86.00**	768.00 (1.00×)	695.98	227.15	768.00	29.58
	AdaBlock-dLLM	85.00	122.07 (6.29×)	110.79	228.13	768.00	29.70
	DAEDAL-port*	81.00	110.93 (6.92×)	42.77	187.75	260.52	72.07
	MaskHorizon	84.00	**85.88** (8.94×)	**29.14**	177.74	244.48	**72.70**
MATH500	Vanilla	53.00	768.00 (1.00×)	697.74	406.03	768.00	52.87
	AdaBlock-dLLM	52.00	163.07 (4.71×)	149.11	394.13	768.00	51.32
	DAEDAL-port*	**56.00**	152.75 (5.03×)	71.84	283.87	361.89	78.44
	MaskHorizon	52.00	**117.10** (6.56×)	**49.49**	265.91	338.56	**78.54**
MBPP	Vanilla	53.00	768.00 (1.00×)	725.90	187.38	768.00	24.40
	AdaBlock-dLLM	54.00	122.55 (6.27×)	116.18	183.03	768.00	23.83
	DAEDAL-port*	53.00	135.28 (5.68×)	62.21	165.36	301.93	54.77
	MaskHorizon	**55.00**	**117.48** (6.54×)	**45.12**	185.00	257.92	**71.73**
HumanEval	Vanilla	51.83	768.00 (1.00×)	762.40	257.42	768.00	33.52
	AdaBlock-dLLM	51.83	144.15 (5.33×)	144.10	262.47	768.00	34.18
	DAEDAL-port*	52.44	145.55 (5.28×)	78.96	204.98	336.88	60.85
	MaskHorizon	**54.88**	**112.14** (6.85×)	**52.78**	204.68	274.54	**74.56**

exposure itself: the observed changes reflect different decoding behavior, not necessarily improved correctness or larger commitment sets.

We then vary the amount of future workspace exposed during decoding. On LLaDA, one reserve block (R_1) improves score over no-reserve growth (R_0) by 2.00–5.49 percentage points, at additional CFP (Table 3). Increasing the reserve further is not consistently beneficial. Compared with full exposure under the same parallel resolver, R_1 reduces CFP by 62.64–69.78%, although MATH500 and HumanEval score 4.00 and 1.22 points lower, respectively. Full-cohort continuations show the same mixed pattern: reserve growth lowers suffix CFP in seven of eight groups, while scores improve in three, tie in two, and decline in three. A limited future reserve therefore captures much of the benefit of advance exposure without carrying the cost of a fully exposed workspace, although the preferred reserve remains task-dependent.

Table 3: **Workspace reserve on LLaDA.** Score (%); CFP (K). R_k: k reserve blocks; R_1: default; Full: full exposure. Bold: best Score or lowest CFP.

Task		R_0	R_1	R_2	R_3	Full
GSM8K	Score	85.00	87.00	**88.00**	**88.00**	82.00
	CFP	**23.55**	27.66	31.26	31.28	91.54
MATH500	Score	37.00	42.00	45.00	40.00	**46.00**
	CFP	**41.54**	52.78	60.96	71.14	148.86
MBPP	Score	41.00	**43.00**	**43.00**	42.00	38.00
	CFP	**31.33**	39.92	41.17	40.89	118.45
HumanEval	Score	37.80	43.29	42.68	43.29	**44.51**
	CFP	**43.54**	78.07	100.45	113.55	208.95

4.4 Why Expose Ahead but Commit Locally? (RQ3)

We next ask whether future workspace should only inform prediction or also be committed as soon as it is exposed. To isolate this choice, we expand the eligible set from unresolved active-block positions to all unresolved exposed positions, while keeping the workspace-growth policy, confidence-based selector, and block-completion stopping rule unchanged. Allocation follows the same function of the frontier in both variants, although their realized decoding trajectories may differ.

Table 4: **Commitment scope on LLaDA.** Local: active block; All: all exposed positions. Score (%); CFP (K).

	Score↑		**NFE↓**		**CFP↓**	
Task	Local	All	Local	All	Local	All
GSM8K	87.00	52.00	77.51	82.26	27.66	27.79
MATH500	42.00	24.00	121.63	108.94	52.78	44.26
MBPP	43.00	42.00	97.34	85.58	39.92	33.20
HumanEval	43.29	38.41	144.90	94.91	78.07	44.85

Allowing commitment over all exposed positions lowers score in all eight model–task groups, but the magnitude varies substantially (Table 4; Appendix C.2). LLaDA–MBPP loses only one percentage point, whereas Dream shows larger losses on both code tasks. CFP decreases on LLaDA's MATH500, MBPP, and HumanEval tasks, but increases in the other groups. These results support the separation used by MaskHorizon: future positions can participate in prediction, while commitment is deferred until their block becomes active. The comparison does not isolate the specific cause of the observed score reductions.

4.4.1 Parallel Active-Block Resolution

Future reserve and parallel commitment play different roles. We pair growth with or without reserve with either single-position or parallel commitment to separate their contributions (Table 5; additional costs in Appendix C.2). The four settings pair each exposure policy with each commitment rule, so each effect is compared while the other choice is held fixed. Single-position commitment selects the most confident unresolved position, not the leftmost one. Parallel commitment reduces NFE and CFP in every model–task group under either exposure policy, although its quality changes are mixed. Adding reserve improves score under both commitment rules in all eight groups, but also increases CFP: it supplies future context rather than reducing the number of repeated model evaluations required to complete generation.

Table 5: **Exposure–commitment ablation.** Reserve gains over capacity-only growth, and parallel NFE savings over single-position commitment.

	ΔScore (pp)		**NFE saved (%)**	
Task	Single	Parallel	Capacity	Reserve
LLaDA				
GSM8K	+1.00	+2.00	61.95	62.62
MATH500	+2.00	+5.00	58.12	60.41
MBPP	+1.00	+2.00	55.67	57.40
HumanEval	+4.27	+5.49	57.91	58.72
Dream				
GSM8K	+3.00	+3.00	54.33	56.00
MATH500	+7.00	+5.00	53.84	58.23
MBPP	+2.00	+2.00	39.59	42.73
HumanEval	+6.71	+6.71	45.79	48.78

4.4.2 Workspace Policy under Shared Commitment

We next ask whether the computational savings come from workspace policy rather than differences in token commitment. Replacing only DAEDAL's commitment selector leaves its token outputs, NFE, CFP, and scores unchanged, so we further align its initial workspace, `[MASK]` padding, block-completion stopping, and parallel commitment with MaskHorizon, while retaining DAEDAL's EOS probing and insertion policy. This gives a controlled comparison in which the main remaining difference lies in how the workspace is managed.

Table 6: **Workspace policy under shared commitment.** Ctrl.: adapted DAEDAL; Ours: MaskHorizon.

	Score (%)		**NFE**		**CFP (K)**	
Task	Ctrl.	Ours	Ctrl.	Ours	Ctrl.	Ours
LLaDA						
GSM8K	83.00	87.00	117.79	77.51	66.78	27.66
MATH500	44.00	42.00	198.88	121.63	156.48	52.78
MBPP	36.00	43.00	168.31	97.34	133.60	39.92
HumanEval	45.12	43.29	269.34	144.90	254.87	78.07
Dream						
GSM8K	80.00	84.00	113.44	85.88	56.91	29.14
MATH500	57.00	52.00	151.80	117.10	97.99	49.49
MBPP	53.00	55.00	118.62	117.48	72.78	45.12
HumanEval	51.22	54.88	140.40	112.14	103.31	52.78

MaskHorizon reduces NFE and CFP in all eight model–task groups under this shared commitment setting (Table 6), with higher scores in five groups and lower scores in three. The adapted control is not native DAEDAL, so the comparison isolates workspace policy rather than ranking the original methods. The fixed-cohort growth controls provide a further check: smaller DAEDAL-derived increments substantially reduce CFP (Table B1), showing that growth magnitude and the resulting workspace trajectory explain part of the gap, rather than the model-derived signal alone.

4.5 Block-Granularity Sensitivity

We examine how MaskHorizon changes with the granularity of block-wise decoding and workspace growth. Table 7 varies $B \in \{16, 32, 64\}$ while preserving $R = B$ and $G = 2B$, so block size, reserve, and growth increment change together rather than isolating block width alone.

Finer granularity reduces CFP on all four tasks, but matches the default score only on GSM8K. Coarser granularity improves both score and CFP on MBPP, while increasing CFP on the other tasks. Overall, $B = 32$ provides a stable default across tasks, although no single granularity performs best in every case. Additional controlled analyses are reported in Appendix C.

Table 7: **LLaDA granularity sensitivity.** $R = B, G = 2B$. Score (%)↑; CFP (K)↓. NFE and N: Appendix C.3.

	$B = 16$		$B = 32$ (default)		$B = 64$	
Task	Score	CFP	Score	CFP	Score	CFP
GSM8K	87.00	26.362	87.00	27.664	83.00	30.706
MATH500	39.00	48.801	42.00	52.775	41.00	63.181
MBPP	42.00	37.270	43.00	39.919	45.00	38.231
HumanEval	38.41	62.430	43.29	78.071	43.90	107.830

4.6 Computational and Practical Efficiency (RQ4)

We next examine where the reduction in decoding cost comes from and whether it translates into lower wall-clock latency. On LLaDA–GSM8K, MaskHorizon uses 0.65× as many forward evaluations and a 0.79× call-weighted input width as DAEDAL, corresponding to a CFP ratio of 0.51. This shows that the reduction comes jointly from fewer model evaluations and a smaller exposed workspace, rather than from either factor alone.

Figure 3 reports the corresponding wall-clock measurements. MaskHorizon has the lowest mean latency among the profiled methods in every model–task group. Relative to DAEDAL (DAEDAL-port on Dream), the measured speedups are 2.26–4.29× on LLaDA and 1.39–1.69× on Dream. These results show that the reduction in NFE and CFP also translates into lower same-session runtime, while not implying hardware-independent speedup.

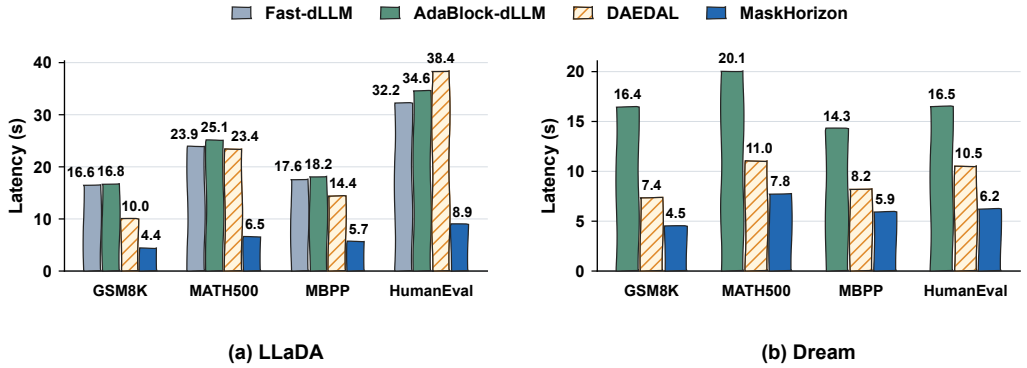


Figure 3: **Historical same-session generation latency.** Mean wall-clock generation time for the profiled decoding methods on (a) LLaDA and (b) Dream across GSM8K, MATH500, MBPP, and HumanEval. All methods within each backbone are measured under the same profiling environment. MaskHorizon achieves the lowest mean latency across all profiled model–task groups. Lower is better.

5 Conclusion

This work studies open-ended dLLM decoding through the control of future workspace. Our analysis shows that workspace exposure itself can substantially affect decoding decisions, while effective growth schedules need not rely on repeated model-derived signals. Based on this observation, MaskHorizon maintains a moving local horizon of future `[MASK]` positions that advances with block-level decoding progress, while restricting commitment to the active block. This structure also removes the need to search for insertion locations, since equivalent growth can be implemented by right-boundary append. Across LLaDA and Dream on reasoning and code-generation tasks, MaskHorizon reduces model evaluations and cumulative forward processing in all evaluated model–task pairs, while achieving better generation quality in most of them. These results suggest that open-ended dLLM generation can be controlled through a progressively advancing mask horizon rather than repeated estimation of workspace demand.

AI Use Statement. Generative AI tools, including OpenAI Codex, assisted with literature discovery, translation, manuscript drafting and editing, figure refinement, experimental design and feedback, evaluation code, and result interpretation. They also assisted with formulating and presenting the allocation bound and its proof. Reported measurements are based on model executions and recorded evaluation outputs, not generated estimates. The authors are responsible for the final text, code, claims, and results.

REFERENCES

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

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.

Fangyu Ding, Ding Ding, Sijin Chen, Kaibo Wang, Peng Xu, Zijin Feng, Haoli Bai, Kai Han, Youliang Yan, Binhang Yuan, and Jiacheng Sun. Beyond masks: Efficient, flexible diffusion language models via deletion-insertion processes. In *The Fourteenth International Conference on Learning Representations*, 2026.

Zhenbang Du, Kejing Xia, Xinrui Zhong, Yonggan Fu, Nicolai Oswald, Binfei Ji, Brucek Khailany, Pavlo Molchanov, and Yingyan Lin. R^2-dLLM: Accelerating Diffusion Large Language Models via Spatio-Temporal Redundancy Reduction. *arXiv preprint arXiv:2604.18995*, 2026.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks*, 2021.

Jaeyeon Kim, Cheuk-Kit Lee, Carles Domingo-Enrich, Yilun Du, Sham Kakade, Timothy Ngotiaoco, Sitan Chen, and Michael Albergo. Any-order flexible length masked diffusion. In *The Fourteenth International Conference on Learning Representations*, 2026.

Jinsong Li, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Jiaqi Wang, and Dahua Lin. Beyond fixed: Training-free variable-length denoising for diffusion large language models. In *The Fourteenth International Conference on Learning Representations*, 2026a.

Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Yi Liang, Soroush Vosoughi, and Shiwei Liu. Diffusion language model knows the answer before it decodes. In *The Fourteenth International Conference on Learning Representations*, 2026b.

Kaihua Liang, Xin Tan, An Zhong, Hong Xu, and Marco Canini. Focus: Dllms know how to tame their compute bound. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Lingkun Long, Yushi Huang, Shihao Bai, Ruihao Gong, Jun Zhang, Ao Zhou, and Jianlei Yang. Focus-dllm: Accelerating long-context diffusion llm inference via confidence-guided context focusing. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12146–12160. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.556.

Guanxi Lu, Hao Mark Chen, Yuto Karashima, Zhican Wang, Daichi Fujiki, and Hongxiang Fan. Adablock-dllm: Semantic-aware diffusion llm inference via adaptive block size. In *The Fourteenth International Conference on Learning Representations*, 2026.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Shen Nie, Qiyang Min, Shaoxuan Xu, Zihao Huang, Yuxuan Song, Yong Shan, Yankai Lin, Wayne Xin Zhao, Chongxuan Li, and Ji-Rong Wen. Improved large language diffusion models. *arXiv preprint arXiv:2606.25331*, 2026.

Yuerong Song, Xiaoran Liu, Ruixiao Li, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. Sparse-dllm: Accelerating diffusion llms with dynamic cache eviction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33038–33046, 2026. doi: 10.1609/aaai.v40i39.40586.

Bian Sun, Kevin Zhai, Mubarak Shah, and Zhenyi Wang. Dystruct: Dynamically structured diffusion language model decoding via bayesian inference. *arXiv preprint arXiv:2605.09820*, 2026.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm. In *The Fourteenth International Conference on Learning Representations*, 2026a.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. In *The Fourteenth International Conference on Learning Representations*, 2026b.

Tianyi Wu, Xiaoxi Sun, Yanhua Jiao, Yulin Li, Yixin Chen, Yunhao Cao, Yiqi Hu, and Zhuotao Tian. Dynamic-dllm: Dynamic cache-budget and adaptive parallel decoding for training-free acceleration of diffusion llm. In *The Fourteenth International Conference on Learning Representations*, 2026c.

Zirui Wu, Lin Zheng, Zhihui Xie, Jiacheng Ye, Jiahui Gao, Shansan Gong, Yansong Feng, Zhenguo Li, Wei Bi, Guorui Zhou, and Lingpeng Kong. Dreamon: Diffusion language models for code infilling beyond fixed-size canvas. In *The Fourteenth International Conference on Learning Representations*, 2026d.

Jingyi Yang, Yuxian Jiang, and Jing Shao. ρ-EOS: Training-free Bidirectional Variable-Length Control for Masked Diffusion LLMs. *arXiv preprint arXiv:2601.22527*, 2026.

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

Andrew Zhang, Anushka Sivakumar, Chia-Wei Tang, and Chris Thomas. Flexible-length text infilling for discrete diffusion models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 31344–31359, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.1597.

Shun Zou, Yong Wang, Zehui Chen, Lin Chen, Chongyang Tao, Feng Zhao, and Xiangxiang Chu. Breaking block boundaries: Anchor-based history-stable decoding for diffusion large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 20637–20658. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.945.

A Method Details and Proofs

A.1 Decoding Algorithm

Algorithm 1: MaskHorizon

Require: Model p_θ, prompt $\mathbf{x}$, initial workspace N_0, cap $N_{\max}$, block size B, reserve R, growth increment G, threshold τ

$N \leftarrow N_0$; $\mathbf{z} \leftarrow$ `[MASK]`^{N_0}; $f \leftarrow 0$
while $f < N_{\max}$ **do**
  $e \leftarrow \min(f + B, N_{\max})$; $N_{\mathrm{req}}(f) \leftarrow \min(N_{\max}, e + R)$
  if $N < N_{\mathrm{req}}(f)$ **then**
    $N' \leftarrow \min(N_{\max}, \max(N_{\mathrm{req}}(f), N + G))$
    Append $N' - N$ `[MASK]` positions to the right of $\mathbf{z}$; $N \leftarrow N'$
  end if
  $\mathcal{U} \leftarrow \{i \in [f, e) : z_i =$ `[MASK]`$\}$
  while $\mathcal{U} \neq \varnothing$ **do**
    Forward $[\mathbf{x}; \mathbf{z}]$; compute $\hat{y}_i, q_i$ on $\mathcal{U}$ by Eq. (4)
    $\mathcal{S} \leftarrow \{i \in \mathcal{U} : q_i \geq \tau\}$
    If $\mathcal{S} = \varnothing$, set $\mathcal{S} \leftarrow \{i^*\}$ for one $i^* \in \arg\max_{i \in \mathcal{U}} q_i$
    Commit $z_i \leftarrow \hat{y}_i$ for all $i \in \mathcal{S}$ in parallel; $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{S}$
  end while
  If a stop starts in $\mathbf{z}_{[0,e)}$, return $\mathbf{z}_{[0,s)}$ for the first such position s
  $f \leftarrow e$
end while
return $\mathbf{z}_{[0,f)}$

A.2 Insertion–Append Equivalence

Let `M` denote the `[MASK]` token. We first establish the state invariant maintained by MaskHorizon at block boundaries.

Proposition A.1 (Masked-suffix invariant)**.** *At every workspace-growth boundary k, the exposed workspace has the form*

$$z^{(k)} = y^{(k)} \parallel \mathtt{M}^{s_k}, \tag{7}$$

where $y^{(k)}$ is fully resolved and $s_k \geq 0$.

Proof. At initialization,

$$z^{(0)} = \mathtt{M}^{N_0},$$

so the claim holds with $y^{(0)} = \varnothing$.

Assume

$$z^{(k)} = y^{(k)} \parallel \mathtt{M}^{s_k}.$$

Let the next active block contain b_k positions. Since commitment is restricted to this block, resolving it changes only its first b_k masked positions. For some resolved block $u^{(k)}$,

$$z = y^{(k)} \parallel u^{(k)} \parallel \mathtt{M}^{s_k - b_k}.$$

If the subsequent growth adds $\Delta_k \geq 0$ masked positions, then

$$z^{(k+1)} = \underbrace{\left(y^{(k)} \parallel u^{(k)}\right)}_{y^{(k+1)}} \parallel \mathtt{M}^{\,s_k - b_k + \Delta_k}.$$

Hence the invariant is preserved. The result follows by induction.

Proposition A.2 (Insertion–append equivalence)**.** *Suppose*

$$z = y \parallel \mathtt{M}^{s}.$$

For any $r \in \{0, \ldots, s\}$ and any $\Delta \in \mathbb{Z}_{\geq 0}$, let $\mathrm{Insert}_{r,\Delta}(z)$ insert Δ masked positions after the first r positions of the masked suffix. Then

$$\mathrm{Insert}_{r,\Delta}(z) = \mathrm{Append}_{\Delta}(z). \tag{8}$$

Proof. Directly,

$$\begin{aligned}\mathrm{Insert}_{r,\Delta}(z) &= y \parallel \mathtt{M}^{r} \parallel \mathtt{M}^{\Delta} \parallel \mathtt{M}^{s-r} && (9)\\ &= y \parallel \mathtt{M}^{s+\Delta} && (10)\\ &= \mathrm{Append}_{\Delta}(z). && (11)\end{aligned}$$

Implication. Propositions A.1 and A.2 show that, under the state structure maintained by MaskHorizon, insertion location is not an independent control variable. Once the net growth amount is fixed, every admissible insertion inside the masked suffix is equivalent at the token-sequence level and can be implemented by simply appending the same number of `[MASK]` tokens at the right boundary. MaskHorizon therefore removes insertion-location search and reduces workspace adjustment to deciding when and how much to grow.

If positional indices, attention masks, and other model-side metadata are constructed deterministically from the resulting token sequence in the same way, the subsequent model evaluation also receives identical inputs under these equivalent operations.

This equivalence is specific to the all-`[MASK]` suffix maintained by MaskHorizon. It does not apply to arbitrary insertion states containing resolved or prefilled non-mask tokens to the right of the insertion point.

A.3 Allocation Overhead Bound

For any fixed nondecreasing frontier sequence f_k, define

$$N_k^{\star} = \max(N_0, N_{\mathrm{req}}(f_k))\,, \tag{12}$$

which is the minimum monotone allocation retaining N_0 and satisfying the reserve requirement at block k. If N_k denotes MaskHorizon's allocation after the corresponding growth check, then

$$0 \le N_k - N_k^{\star} < G. \tag{13}$$

Assume $0 < N_0 \le N_{\max}$ and positive integer growth increment G. Since $N_{\mathrm{req}}(f_k)$ is nondecreasing, $N_k^{\star} = \max(N_0, N_{\mathrm{req}}(f_k))$ is the smallest monotone allocation that retains the initial workspace and satisfies every reserve constraint up to block k. We prove Eq. (13) by induction. If no expansion occurs, $N_k \ge N_{\mathrm{req}}(f_k)$ and the allocation is unchanged; the reference cannot decrease, so the excess cannot increase. Before the first expansion, $N_k = N_k^{\star} = N_0$. If expansion occurs, the previous allocation satisfies $N_{k-1} < N_{\mathrm{req}}(f_k)$ and $N_{k-1} \ge N_0$, hence $N_k^{\star} = N_{\mathrm{req}}(f_k)$. Equation (3) gives

$$0 \le N_k - N_{\mathrm{req}}(f_k) \le \max\{0, N_{k-1} + G - N_{\mathrm{req}}(f_k)\} < G. \tag{14}$$

The cap preserves the lower bound because $N_{\mathrm{req}}(f_k) \le N_{\max}$. The same argument applies to the first growth check with previous allocation N_0. For any matched nonnegative block forward counts T_k with $\sum_k T_k > 0$, define $\mathrm{CFP} = \sum_k T_k(P + N_k)$ and $\mathrm{CFP}^{\star} = \sum_k T_k(P + N_k^{\star})$. Multiplying the allocation bound by T_k and summing proves

$$0 \le \mathrm{CFP} - \mathrm{CFP}^{\star} < G \sum_k T_k. \tag{15}$$

If all $T_k = 0$, both costs are zero. This result conditions on the frontier sequence and forward counts; it does not assert that another exposure schedule would produce the same trajectory, outputs, or quality.

B Experimental Protocol

B.1 Models, Data, Metrics, and Stopping

Model interfaces. Adapters supply native tokenization, prompt templates, `[MASK]` and stop-token IDs, and logit alignment. They preserve the same workspace-growth and active-block commitment algorithm across both backbones.

Evaluation and baseline conventions. All evaluations perform test-time decoding with frozen pretrained models, without training or fine-tuning. Main comparisons, reserve and factorial ablations, shared-commitment controls, threshold sweeps, and reproduction diagnostics use the same task cohorts. Vanilla retains its fixed-budget schedule; other baselines retain their reproduced decoding and stopping rules. The final MaskHorizon defaults are specified in Section 4.1.

The anomalous Dream Fast-dLLM reproduction is excluded from rankings and reported in Appendix B.3. The native-decoder and controlled-growth pilots use the same fixed LLaDA subsets of 20 test examples each from GSM8K and MATH500. The earlier future-mask pilot and historical latency study also use 20 examples per task and backbone. These pilots are distinct from the main comparisons; no validation split is used. Their state-screening and configuration exceptions are specified below.

Metric aggregation. NFE is the number of model forward evaluations T. Without caching, each forward processes the prompt and every exposed position, so $\mathrm{CFP} = \sum_{t=1}^{T}(P + N_t)$. NFE, CFP, E, and N are sample means, with CFP in thousands (K). Effective output length E counts tokens before the first recognized stop, and utilization is $100 \sum_i E_i / \sum_i N_i$. NFE reduction factors divide mean Vanilla NFE by mean method NFE; they are not wall-clock speedups.

Termination details. The decoder checks for a recognized stop only after the active block is complete, returning tokens preceding the first such stop. An unresolved future prediction or a stop inside an unfinished block does not terminate decoding. At the allocation cap, the decoder still resolves active blocks until a recognized stop or completion of the last block. Effective output length counts tokens before the first recognized stop, while all executed forwards contribute to NFE and CFP.

B.2 Controlled Workspace/Controller Protocols

Targeted studies and reused results. The commitment-scope intervention covers all four tasks on both backbones to validate exposure–commitment separation, not to tune a hyperparameter. Block-granularity and initial-workspace sensitivity use the same task cohorts on LLaDA. The granularity sweep varies $B \in \{16, 32, 64\}$ under the coupling specified in Section 4.1, with each task's default N_0 fixed. The initialization sweep varies $N_0 \in \{128, 192, 256\}$ with other settings fixed, additionally retaining HumanEval's default $N_0 = 224$ as its reference. These measurements do not select new defaults after observing the results. Default rows reuse the corresponding main-result records after checking sample manifests, prompts, configurations, and token-level decoding parity; reused rows are not independent replications.

Threshold exploration. The threshold sweep tests $\tau \in \{0.70, 0.80, 0.90\}$ on the same test examples. Earlier LLaDA threshold choices also used these examples; the sweep is descriptive and does not provide independent validation.

Controlled experiments. R_k maintains a minimum reserve of kB beyond the active block; chunked growth may expose more. Full+Parallel starts at the cap but keeps the parallel resolver and block-completion stopping, unlike Vanilla's fixed-budget decoder. The factorial study crosses R_0/R_1 with highest-confidence single-position or parallel commitment. Its parallel arms are shared with the reserve ablation, not independent replications. Figure C1 pools samples within each model and arm, rather than averaging the four task means equally.

Controller comparison notation. Score is in percent and CFP in thousands. In Table B1, Δ denotes the requested append amount; a dash denotes native, method-specific operations and 8–64 denotes density-dependent growth. Derived labels identify adapted controllers, and Confidence-0.10/0.30/0.50 denotes confidence-based growth at the indicated threshold, not the commitment threshold. Signal variants have a separate $+64$ capacity safeguard. The default MaskHorizon row is the same reference in both blocks. Means are descriptive, not significance or non-inferiority claims.

Motivation analysis. Figure 1(a) preserves the earlier exposure pilot rather than replacing it with the full-cohort results in Appendix C.1. Its LLaDA intervention uses rollout traces at $\tau = 0.80$ and queries at $\tau = 0.90$. Same-state analyses retain only eligible states, without replacing ineligible

examples; the full-cohort eligibility procedure appears in Appendix C.1. Top-1 changes are averaged across positions within each eligible LLaDA state and then across states; commitment-set changes are averaged across states. Eligibility is established before querying alternative exposures.

Figure 1(b) uses the fixed controller-pilot IDs and the same shared-interface decoder with $B = 32$, $G = 64$, $\tau = 0.90$, $N_{\max} = 768$, and $N_0 = 192/256$ for GSM8K/MATH500. Fixed-R0 supplies only active-block capacity; Fixed-R1/R2 require one/two future blocks at block entry. All three inspect only the frontier and allocation and append masks on the right. Greedy token selection and active-block parallel commitment are shared with the signal controllers; no extra model evaluations are used for growth decisions. Fixed-R1 and the signal arms reuse configuration-matched results, not independent replications. Fixed-R0 triggers no growth on the observed trajectories and therefore serves as a no-reserve control, not evidence of continued expansion by itself. The points are descriptive pilot means, with no post-hoc rule selection.

Native-decoder reference. The native-decoder block of Table B1 reports complete DAEDAL and ρ-EOS runs on the same controller-pilot IDs and allocation cap. These runs retain method-specific initialization, commitment, and stopping: DAEDAL retains its two-stage length adjustment and internal insertion, while ρ-EOS retains density-dependent expansion, contraction, and its iteration gate. They are end-to-end references, not controlled changes of the growth criterion. The default MaskHorizon result is repeated in both blocks for reference, not generated twice.

Shared-interface controller pilot. The controlled-variant block of Table B1 compares growth criteria within a common append-only interface, fixing the initial workspace, active-block commitment rule, stopping condition, and capacity limit. The original comparison uses $\Delta = 64$ for progress, DAEDAL-derived, ρ-EOS-derived, and confidence-based controllers (C-0.10/0.30/0.50). The growth-amount controls additionally test $\Delta = 32$ for progress and both adapted controllers, $\Delta = 7$ for DAEDAL-derived, and the native density-dependent growth magnitude (8/16/32/64) for ρ-EOS-derived. Every signal controller retains a separate $+64$ capacity safeguard. DAEDAL-derived growth requires EOS confidence below 0.90 and an unselected active position with confidence below 0.10; ρ-EOS-derived growth retains density below 0.40 and the first-64-iteration gate. These are growth criteria, distinct from the common commitment resolver. All expansions append masks. The magnitude controls do not restore native insertion, contraction, or the complete native algorithms.

Smaller DAEDAL-derived increments substantially reduce CFP, so growth magnitude explains part of its cost gap. Progress still has the highest observed score and lower CFP than the tested signal settings on both tasks. The two progress increments tie in score but exchange cost advantages across tasks; the default remains $G = 64$. These controlled comparisons are not a native-method ranking, a significance claim, or post-hoc selection of new defaults. During completion, one cap-truncated MATH500 output ending in an incomplete `\frac` exposed an empty-fragment parser error. A guarded parser leaves this malformed expression unchanged; the existing equality rule marks it incorrect. Previously completed scores are unchanged.

B.3 Baseline Reproduction and Latency Details

Latency profiling. Profiling uses one measured run per method and example. All methods for a backbone share one model process on an NVIDIA RTX 4090 D, with method order rotated by example. Timings are CUDA-synchronized and exclude loading, tokenization, warm-up, and scoring. LLaDA profiles Fast-dLLM, AdaBlock-dLLM, DAEDAL, and MaskHorizon; Dream excludes the anomalous Fast-dLLM reproduction.

Dream Fast-dLLM Diagnostics. Our Dream Fast-dLLM reproduction produces abnormally short or empty outputs under the matched protocol. Table B2 preserves all four original result rows. The cause remains unresolved; these rows are excluded from main-table rankings, efficiency comparisons, and runtime claims. They diagnose this reproduction, not the general performance of Fast-dLLM.

Table B1: **Native references and growth-amount controls.** Score (%); CFP (K). Native decoders retain method-specific operations; controlled variants share a right-append interface. Derived controllers are not the native methods. The default MaskHorizon row is the same reference in both blocks, not an independent replication.

		GSM8K		MATH500	
Method / Variant	Δ	Score↑	CFP↓	Score↑	CFP↓
Native decoders					
DAEDAL	–	75.00	67.434	25.00	160.005
ρ-EOS	–	60.00	49.076	20.00	135.608
MaskHorizon	64	85.00	29.233	45.00	57.259
Controlled variants					
MaskHorizon	64	85.00	29.233	45.00	57.259
MaskHorizon	32	85.00	30.337	45.00	52.230
DAEDAL-derived	64	70.00	75.892	25.00	140.743
DAEDAL-derived	32	75.00	67.415	25.00	135.610
DAEDAL-derived	7	80.00	44.549	25.00	90.565
ρ-EOS-derived	64	75.00	57.489	25.00	130.049
ρ-EOS-derived	32	75.00	60.373	20.00	127.989
ρ-EOS-derived	8–64	80.00	59.205	25.00	135.373
Confidence-0.10	64	70.00	86.641	30.00	142.805
Confidence-0.30	64	75.00	89.131	25.00	158.290
Confidence-0.50	64	75.00	89.429	30.00	158.734

Table B2: **Anomalous Dream Fast-dLLM reproduction.** Low effective output lengths E accompany the collapsed scores. CFP is in thousands; utilization is descriptive, not ranked.

Task	Score (%)	NFE	CFP (K)	E	N	Util. (%)
GSM8K	34.00	175.15	158.88	35.51	768.00	4.62
MATH500	29.00	246.54	224.45	18.79	768.00	2.45
MBPP	2.00	226.65	214.14	1.32	768.00	0.17
HumanEval	1.22	240.71	239.29	0.61	768.00	0.08

C Additional Results

C.1 Same-State Intervention and Continuations

State selection and intervention. Before querying alternative exposures, we screen MaskHorizon traces and select the first block entry with enough capacity for the active block but less than a full future-block reserve. Each query pair keeps the prompt, existing tokens, frontier, and active block fixed, changing only the addition of future `[MASK]` positions at the right boundary. The intervention averages the fraction of changed top-1 predictions over unresolved active positions within each state, then over all eligible states within each backbone. Its second measure is the fraction of states whose parallel commitment set changes, including the one-position fallback. Changed sets need not be larger: their mean size decreases by 2.00 positions on LLaDA and 0.90 on Dream. The intervention establishes exposure sensitivity, not better predictions or faster commitment.

Paired continuations. From each selected state, S exposes the full workspace, C uses capacity-only growth, and R uses the default reserve rule. These arms share the already decoded prefix, parallel commitment, and block-completion stopping. C is therefore not a fresh R_0 run, and S is not Vanilla or full exposure from the start. Tables C1 and C2 report final quality and only the post-fork cost, excluding the common prefix and diagnostic queries.

The paired correctness changes show which gains and losses cancel in aggregate; neither equal totals nor these descriptive comparisons establish non-inferiority.

Although R lowers CFP in seven groups, it lowers NFE only on the two MATH500 groups. LLaDA–MBPP instead uses $1.05\times$ S's suffix CFP.

Table C1: **Matched-state continuation quality.** S uses full exposure at the fork, C capacity-only growth, and R reserve growth. S, C, and R report scores (%). W→C and C→W report the percentages of eligible states whose final answers change from wrong to correct and correct to wrong, respectively, from S to R.

Model	Task	S	C	R	W→C (%)	C→W (%)
LLaDA	GSM8K	88.75	83.75	86.25	0.00	2.50
LLaDA	MATH500	34.94	27.71	33.73	0.00	1.20
LLaDA	MBPP	38.96	36.36	38.96	0.00	0.00
LLaDA	HumanEval	41.67	34.03	40.28	0.69	2.08
Dream	GSM8K	77.46	74.65	78.87	1.41	0.00
Dream	MATH500	46.88	42.19	50.00	3.13	0.00
Dream	MBPP	59.49	56.96	59.49	0.00	0.00
Dream	HumanEval	50.72	37.68	53.62	2.90	0.00

Table C2: **Matched-state continuation cost.** Mean NFE and CFP after the shared fork, excluding prefix computation and diagnostic queries. S, C, and R use the same eligible states as Table C1.

		Suffix NFE			Suffix CFP (K)		
Model	Task	S	C	R	S	C	R
LLaDA	GSM8K	16.34	7.46	17.74	15.07	2.56	7.70
LLaDA	MATH500	40.59	10.04	33.55	37.41	4.15	17.69
LLaDA	MBPP	13.60	6.94	27.97	13.37	2.85	14.01
LLaDA	HumanEval	69.45	10.40	70.28	70.57	4.85	44.18
Dream	GSM8K	10.83	7.99	14.08	9.83	2.64	5.71
Dream	MATH500	35.78	10.06	28.25	33.45	4.19	15.71
Dream	MBPP	16.95	15.13	26.57	16.07	5.61	11.98
Dream	HumanEval	24.35	11.59	31.75	24.97	5.44	19.58

C.2 Commitment Diagnostics

Commitment scope. Table C3 extends the intervention in Table 4 to Dream. All-exposed commitment lowers score and increases NFE and CFP on every task, with the largest score losses on the two code tasks. The comparison does not isolate a specific failure mechanism.

Table C3: **Commitment scope on Dream.** Local: active block (MaskHorizon); All: all exposed positions. Both variants use the same workspace-growth policy, selector, and stopping rule.

Task	Scope	Score (%)↑	NFE↓	CFP (K)↓
GSM8K	Local	84.00	85.88	29.14
	All	48.00	100.77	33.27
MATH500	Local	52.00	117.10	49.49
	All	33.00	141.94	56.13
MBPP	Local	55.00	117.48	45.12
	All	6.00	140.46	51.84
HumanEval	Local	54.88	112.14	52.78
	All	3.05	146.67	65.45

Commitment rule. Table C4 compares parallel commitment with the historical stateful FCPC resolver on Dream, keeping workspace and stopping fixed. Commit/Fwd divides committed block tokens by total NFE. The broader adapted-DAEDAL workspace-policy control is reported below.

Table C4: **Commitment rules on Dream.** Parallel is the standard confidence-based resolver; FCPC is the historical stateful resolver. Bold marks the better value within each pair.

	Score (%)↑		NFE↓		CFP (K)↓		Commit/Fwd↑	
Dataset	Parallel	FCPC	Parallel	FCPC	Parallel	FCPC	Parallel	FCPC
GSM8K	**84.00**	83.00	**85.88**	91.38	**29.14**	31.07	**2.27**	2.13
MATH500	52.00	**53.00**	**117.10**	123.24	**49.49**	52.37	**2.39**	2.28
MBPP	55.00	**56.00**	**117.48**	122.70	**45.12**	47.19	**1.73**	1.66
HumanEval	**54.88**	54.27	**112.14**	117.26	**52.78**	55.28	**1.97**	1.89

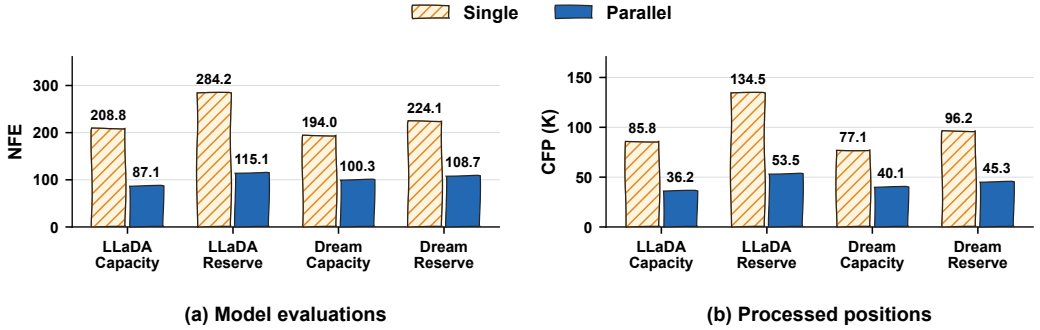


Figure C1: **Exposure and commitment costs.** (a) Forward calls. (b) Processed positions. Parallel commitment lowers both costs under either exposure policy.

C.3 Scheduling Sensitivity

These sensitivity measurements characterize the existing decoder; they are not used to select new defaults after observing the results. The main configurations in Section 4.1 remain unchanged.

Granularity cost details. Table C5 supplements the quality–CFP comparison in Table 7 with forward counts and final workspace. Reserve and growth remain coupled to block width as $R = B$ and $G = 2B$; these are the same runs, not an additional experiment.

Table C5: **Granularity cost details on LLaDA.** Shaded rows use the unchanged default. Quality and CFP are reported in Table 7.

Task	B	NFE↓	Final N
GSM8K	16	76.89	222.72
GSM8K	32	77.51	255.36
GSM8K	64	80.11	312.96
MATH500	16	116.13	303.68
MATH500	32	121.63	358.40
MATH500	64	130.25	446.72
MBPP	16	94.18	230.40
MBPP	32	97.34	273.28
MBPP	64	89.20	358.40
HumanEval	16	123.93	303.61
HumanEval	32	144.90	396.10
HumanEval	64	164.98	620.10

Initial workspace. The initial workspace produces a task-dependent quality–computation trade-off (Table C6). A smaller allocation saves CFP in this sweep but need not preserve quality: the most compact setting scores 78% on GSM8K versus the default's 87%, and 37% on MATH500 versus 42%. Starting MATH500 at 192 positions reduces CFP from 52.775K to 37.361K but lowers score

to 36%; increasing GSM8K's initial allocation raises cost without exceeding the default score. On MBPP, the default also has the highest tested score; larger initialization costs more and scores lower. On HumanEval, increasing the initial workspace from the default to 256 positions raises score from 43.29% to 43.90%, with CFP rising from 78.071K to 83.982K. Neither smaller initial workspaces nor the current task-specific defaults are universally preferable.

Table C6: **Initial-workspace sensitivity on LLaDA.** Only the initial allocation varies. Shading identifies each task's main configuration; final workspace can grow beyond the initial allocation.

Task	N_0	Score (%)↑	NFE↓	CFP (K)↓	Final N
GSM8K	128	78.00	61.80	18.490	200.32
GSM8K	192	87.00	77.51	27.664	255.36
GSM8K	256	85.00	91.91	38.292	316.80
MATH500	128	37.00	88.91	29.855	250.88
MATH500	192	36.00	100.60	37.361	286.72
MATH500	256	42.00	121.63	52.775	358.40
MBPP	128	40.00	89.27	32.336	255.36
MBPP	192	43.00	97.34	39.919	273.28
MBPP	256	41.00	111.80	51.899	323.20
HumanEval	128	38.41	125.34	59.667	331.32
HumanEval	192	40.85	136.78	71.407	372.29
HumanEval	224	43.29	144.90	78.071	396.10
HumanEval	256	43.90	148.85	83.982	412.88

Commitment threshold. Lowering τ permits more aggressive parallel commitment. In the tested sweep, this reduces NFE and CFP in every model–task pair, but the accompanying quality changes depend on the task (Figure C2).

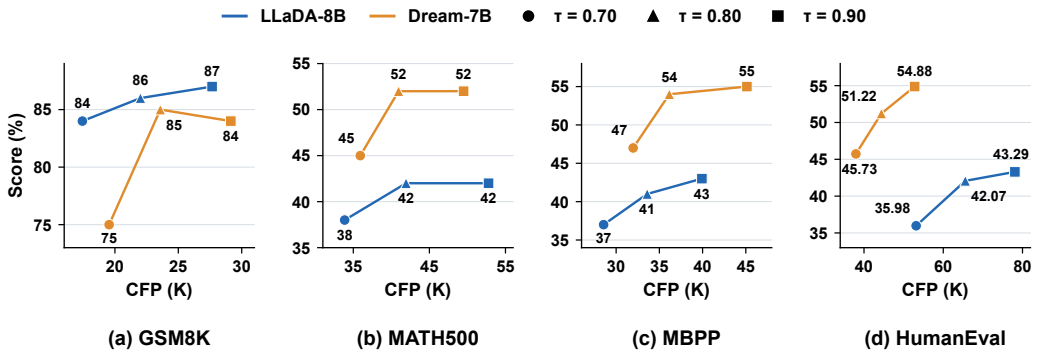


Figure C2: **Commitment-threshold sensitivity.** Score versus CFP on LLaDA (blue) and Dream (orange). Shapes denote thresholds; lines connect tested settings. Upper left is preferable.

On LLaDA–MATH500, $\tau = 0.80$ matches the default's 42.00% score while reducing NFE from 121.63 to 96.83 and CFP from 52.78K to 41.94K. On Dream it improves GSM8K and matches MATH500, but lowers both code scores. The more aggressive $\tau = 0.70$ lowers quality in all eight model–task pairs. Moderate reductions can therefore save computation without changing aggregate accuracy on some tasks, but the trade-off remains task-dependent.

C.4 Qualitative Examples

These post-hoc cases use the main-table runs and identical prompts across methods. Under a fixed evaluation order, we select the first example for each displayed model–task pair where all three methods are correct and MaskHorizon uses at least 20% fewer output tokens, with lower NFE and CFP than both baselines. Tokens count the full response before stopping, including reasoning and code; costs cover full generation. MaskHorizon reaches a recognized stop in every case. Excerpts are marked below; these examples do not establish a typical gain or uniform quality preservation.

Case 1: GSM8K **LLaDA**

Question

James decides to run 3 sprints 3 times a week. He runs 60 meters each sprint. How many total meters does he run a week?

Ground truth: 540 meters

Selected passages, with formatting normalized. [. . .] marks omitted text.

Vanilla

236 tokens **NFE: 768** **CFP: 683.52K**

1. Calculate the distance James runs in one sprint.
2. Calculate the distance James runs in one day.
3. Calculate the distance James runs in one week.

[. . .]

Finally, James runs 3 days a week. Therefore, the distance he runs in one week is: 180 meters/day $\times$ 3 days/week $=$ 540 meters/week So, the total distance James runs in a week is 540 meters.

DAEDAL

232 tokens **NFE: 110** **CFP: 39.19K**

1. Calculate the distance James runs in one sprint.
2. Calculate the distance James runs in one day.
3. Calculate the distance James runs in one week.

[. . .]

Finally, James runs 3 times a week. Therefore, the distance he runs in one week is: 180 meters/day $\times$ 3 days/week $=$ 540 meters/week So, the total distance James runs in a week is $\boxed{540}$ meters.

MaskHorizon

181 tokens **NFE: 68** **CFP: 21.74K**

1. Calculate the distance James runs in one day:
- James runs 3 sprints each day.
- Each sprint is 60 meters.
- Therefore, the distance he runs in one day is $3 \times 60 = 180$ meters.
2. Calculate the distance James runs in one week:
- James runs 3 times a week.
- Therefore, the distance he runs in one week is $180 \times 3 = 540$ meters.

So, the total distance James runs in a week is 540 meters.

Compared with DAEDAL: 22.0% fewer tokens, 38.2% fewer calls, and 44.5% lower CFP. All answers are correct; MaskHorizon combines planning and calculation.

Case 2: MATH500 **Dream**

Question

Find all values of x that satisfy the equation $x = \sqrt{11 - 2x} + 4$.

Ground truth: $x = 5$

Complete reasoning and final answers; response-format tags omitted and math formatting normalized.

Vanilla

349 tokens **NFE: 768** **CFP: 672.77K**

To solve the equation $x = \sqrt{11 - 2x} + 4$, we start by isolating the square root term. Subtract 4 from both sides to get $x - 4 = \sqrt{11 - 2x}$. Next, we square both sides to eliminate the square root, resulting in $(x - 4)^2 = 11 - 2x$. Expanding the left side gives $x^2 - 8x + 16 = 11 - 2x$. Rearranging the terms, we get $x^2 - 6x + 5 = 0$. Factoring this quadratic equation, we find $(x - 1)(x - 5) = 0$. Therefore, the solutions are $x = 1$ and $x = 5$. However, we must check these solutions in the original equation to ensure they are valid. Plugging $x = 1$ into the original equation gives $1 = \sqrt{11 - 2(1)} + 4$, which simplifies to $1 = \sqrt{9} + 4$ or $1 = 3 + 4$, which is false. Plugging $x = 5$ into the original equation gives $5 = \sqrt{11 - 2(5)} + 4$, which simplifies to $5 = \sqrt{1} + 4$ or $5 = 1 + 4$, which is true. Therefore, the only valid solution is $x = 5$.

Final answer: $\boxed{5}$

DAEDAL-port

342 tokens **NFE: 166** **CFP: 69.85K**

To solve the equation $x = \sqrt{11 - 2x} + 4$, we start by isolating the square root term. Subtract 4 from both sides to get $x - 4 = \sqrt{11 - 2x}$. Then, square both sides to eliminate the square root: $(x - 4)^2 = 11 - 2x$. Expand the left side to get $x^2 - 8x + 16 = 11 - 2x$. Move all terms to one side to form a quadratic equation: $x^2 - 6x + 5 = 0$. Factor the quadratic equation: $(x - 1)(x - 5) = 0$. This gives us two potential solutions: $x = 1$ and $x = 5$. However, we must check these solutions in the original equation to ensure they are valid. For $x = 1$, we have $1 = \sqrt{11 - 2(1)} + 4$, which simplifies to $1 = \sqrt{9} + 4$ or $1 = 3 + 4$, which is not true. For $x = 5$, we have $5 = \sqrt{11 - 2(5)} + 4$, which simplifies to $5 = \sqrt{1} + 4$ or $5 = 1 + 4$, which is true. Therefore, the only valid solution is $x = 5$.

Final answer: $\boxed{5}$

MaskHorizon

251 tokens **NFE: 120** **CFP: 44.58K**

To solve the equation $x = \sqrt{11 - 2x} + 4$, we start by isolating the square root term. Subtract 4 from both sides to get $x - 4 = \sqrt{11 - 2x}$. Next, we square both sides to eliminate the square root, resulting in $(x - 4)^2 = 11 - 2x$. Expanding the left side gives $x^2 - 8x + 16 = 11 - 2x$. Rearranging the terms, we get $x^2 - 6x + 5 = 0$. Factoring this quadratic equation gives $(x - 1)(x - 5) = 0$. Therefore, the solutions are $x = 1$ and $x = 5$. However, we need to check if these solutions satisfy the original equation. Plugging $x = 1$ into the original equation does not satisfy it, but plugging $x = 5$ into the original equation does satisfy it. Therefore, the only solution is $x = 5$.

Final answer: $\boxed{5}$

Compared with DAEDAL-port: 26.6% fewer tokens, 27.7% fewer calls, and 36.2% lower CFP.

All three responses derive $(x - 1)(x - 5) = 0$ and reject the extraneous root $x = 1$. MaskHorizon states the validity check more briefly; it does not skip it.

Case 3: MBPP **LLaDA**

Question

Write a function to remove characters from the first string which are present in the second string.

Benchmark outcome: all three implementations pass

Complete generated code blocks; preceding reasoning and response-format tags are omitted.

Vanilla

256 tokens **NFE: 768** **CFP: 715.78K**

```
def remove_dirty_chars(str1, str2):
    # Initialize an empty string to store the result
    result = ""
    # Iterate through each character in the first string
    for char in str1:
        # Check if the character is not in the second string
        if char not in str2:
            # Append the character to the result string
            result += char
    return result
# Test cases
assert remove_dirty_chars("probasscurve", "pros") == 'bacuve'
assert remove_dirty_chars("digitalindia", "talent") == 'digiidi'
assert remove_dirty_chars("exoticmiles", "toxic") == 'emles'
```

DAEDAL

260 tokens **NFE: 141** **CFP: 81.19K**

```
def remove_dirty_chars(str1, str2):
    # Initialize an empty string to store the result
    result = ""
    # Iterate through each character in the first string
    for char in str1:
        # Check if the character is not in the second string
        if char not in str2:
            # Add the character to the result string
            result += char
    return result
# Test cases
assert remove_dirty_chars("probasscurve", "pros") == 'bacuve'
assert remove_dirty_chars("digitalindia", "talent") == 'digiidi'
assert remove_dirty_chars("exoticmiles", "toxic") == 'emles'
```

MaskHorizon

134 tokens **NFE: 61** **CFP: 21.72K**

```
def remove_dirty_chars(str1, str2):
    return ''.join(char for char in str1 if char not in str2)
```

Compared with DAEDAL: 48.5% fewer tokens, 56.7% fewer calls, and 73.3% lower CFP.

All three implementations pass the same externally executed benchmark tests. The baselines repeat the prompt tests in their output; MaskHorizon does not. Omitting these assertions does not bypass evaluation, but part of the token saving reflects less repeated test code and commentary, not necessarily a shorter underlying algorithm.

Case 4: HumanEval **Dream**

Question

Implement `string_xor(a, b)`: return the binary XOR of two strings containing 0s and 1s. Example: `string_xor('010', '110')` returns `'100'`.

Benchmark outcome: all three implementations pass

Task summarized above. Complete generated code blocks below; preceding reasoning is omitted.

Vanilla

306 tokens **NFE: 768** **CFP: 718.08K**

```
def string_xor(a: str, b: str) -> str:
    """ Input are two strings a and b consisting only of 1s and 0s.
    Perform binary XOR on these inputs and return result also as a string.
    >>> string_xor('010', '110')
    '100'
    """
    a_list = list(a)
    b_list = list(b)
    result = []
    for a_char, b_char in zip(a_list, b_list):
        result.append(str(int(a_char) ^ int(b_char)))
    return ''.join(result)
```

DAEDAL-port

307 tokens **NFE: 235** **CFP: 124.08K**

```
def string_xor(a: str, b: str) -> str:
    """ Input are two strings a and b consisting only of 1s and 0s.
    Perform binary XOR on these inputs and return result also as a string.
    >>> string_xor('010', '110')
    '100'
    """
    result = ''
    for bit_a, bit_b in zip(a, b):
        result += str(int(bit_a) ^ int(bit_b))
    return result
```

MaskHorizon

217 tokens **NFE: 140** **CFP: 55.38K**

```
def string_xor(a: str, b: str) -> str:
    a_list = list(a)
    b_list = list(b)
    result = []
    for i in range(len(a_list)):
        if a_list[i] != b_list[i]:
            result.append('1')
        else:
            result.append('0')
    return ''.join(result)
```

Compared with DAEDAL-port: 29.3% fewer tokens, 40.4% fewer calls, and 55.4% lower CFP.

Both baselines repeat the prompt docstring and example. MaskHorizon omits that documentation. Its full response is shorter even though its implementation has more lines than DAEDAL's.