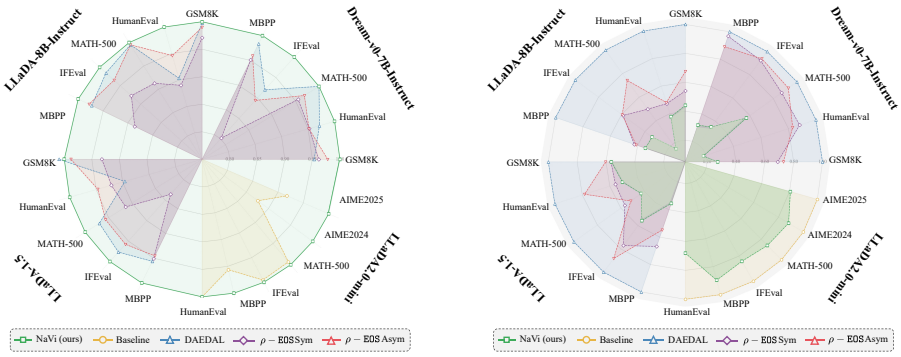
FROM FIXED TO FLEXIBLE: VIA LATENT LENGTH AWARENESS IN DLLMS FOR VARIABLE-LENGTH GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

The fixed-length generation paradigm of Diffusion Large Language Models (dLLMs) inherently limits their capability in tasks requiring variable-length outputs. Existing training-free approaches address this limitation through dynamic length adjustment, but often disrupt the model's learned distribution. In this work, we reveal that dLLMs possess latent length awareness: the remaining generation length can be encoded in the hidden state immediately following the given context. This latent information can be extracted from a single token, termed the Navigator Token, using a lightweight regression probe. Based on this observation, we propose NaVi, a training-free strategy that enables variable-length generation through predictive and minimally disruptive length recalibration. Specifically, NaVi initializes the generation budget via Navigator Token-based length prediction, and during decoding performs schedule-aware threshold adaptation and dynamic length recalibration with semantic prior injection to preserve distributional consistency. Experiments across diverse models and benchmarks demonstrate that NaVi improves generation quality and efficiency without task-specific tuning. Code and models will be made publicly available.



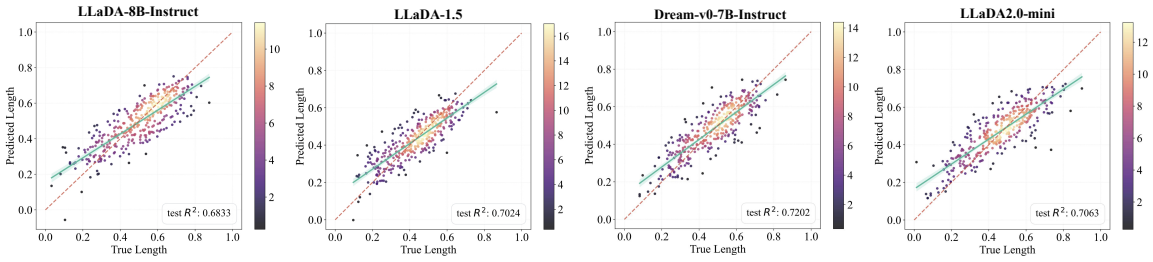
(a) Relative accuracy comparison. (b) Relative runtime comparison.

Figure 1: **Quantitative comparison.** Our method outperforms others across various benchmarks and four model architectures, reducing runtime while improving accuracy.

1 INTRODUCTION

Diffusion Large Language Models (dLLMs) such as LLaDA (Nie et al., 2025; Zhu et al., 2026; Bie et al., 2025; 2026) and Dream (Ye et al., 2025) are emerging as a compelling alternative to the dominant autoregressive (AR) paradigm. By formulating text generation as a parallel, multi-step denoising process over a masked

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.



(a) (b) (c) (d)

Figure 2: **Quantitative analysis of length predictability via Navigator Token.** Scatter plots show the correlation between the actual and predicted remaining sequence lengths using a linear ridge regression probe, which is fitted to the hidden states of the Navigator Token extracted at various intermediate decoding stages. Across four models, the consistently high R^2 values indicate that the Navigator Token accurately encodes the remaining length throughout the entire decoding process.

sequence, dLLMs offer unique advantages, including bidirectional modeling of global context and flexible generation orders. However, they suffer from a major limitation: the final output length is fixed to a predefined generation length. In fact, since task complexity varies, this fixed length creates a dilemma. Specifically, allocating an insufficient budget prevents the model from solving complex tasks, whereas an excessive budget causes unnecessary computational overhead and also degrades the model's performance (Li et al., 2026).

To address this issue, recent training-free methods have attempted to equip dLLMs with variable-length capabilities. DAEDAL (Li et al., 2026) employs a two-stage approach, utilizing a forward loop before denoising to determine the initial length, followed by internal mask insertions. To avoid the latency of two-stage pipelines, $\rho - \texttt{EOS}$ (Yang et al., 2026) operates in a single stage, dynamically adding or truncating mask tokens based on implicit `EOS` density during the denoising process.

However, these methods may introduce disruptions to the decoding distribution. Because dLLMs are trained on fixed-length trajectories, frequently adding or truncating mask tokens during the decoding process violates the expected Markovian transitions (Kim et al., 2025). This distribution shift forces the model to expend extra effort recovering from unexpected structural changes, slowing down inference or degrading output quality.

A natural approach to avoid such disruptions is to predict the full generation length initially and keep it fixed during decoding. However, this paradigm sacrifices flexibility, as it cannot adaptively adjust the sequence if the initial prediction proves to be insufficient or excessive. Therefore, the critical challenge remains: *How can we predict the generation length before decoding, while retaining the flexibility to dynamically expand without significantly disturbing the models' intrinsic distribution?*

Key Observations. To achieve dynamic length prediction while preserving the decoding distribution, we study whether the remaining generation length can be derived from the model itself. Specifically, given a context comprising the initial prompt and the intermediate decoded output, we append exactly one `[MASK]` token. This setup compels the dLLM to compress the content of the upcoming response into a single token. We hypothesize that the hidden states of this specific token, which we term the Navigator Token, can encode the expected remaining length, effectively capturing the dynamic latent length awareness within dLLMs.

To verify this, we extract the Navigator Token's hidden states across different decoding stages and fit a linear ridge regression probe. As shown in Fig. 2, the consistently high R^2 values indicate that the Navigator Token accurately encodes an awareness of the remaining distance throughout the entire decoding process. Consequently, the Navigator Token can be used to dynamically estimate the remaining length during decoding,

enabling flexible budget adjustment. Driven by these findings, we formulate a decoding strategy that leverages Navigator Token guidance to achieve variable-length generation while preserving the decoding distribution.

Our Solution. In this work, we propose **NaVi**, a training-free **Navi**gator-guided variable-length generation strategy. NaVi operates through three mechanisms: *1) Dynamic Length Recalibration*, *2) Semantic Momentum Injection*, and *3) Schedule-Aware Dynamic Threshold Decoding*.

Specifically, for dynamic length recalibration, we first establish the initial generation length budget via the probe on the initial Navigator Token, and dynamically adjust it via length recalibration when the currently allocated budget reaches its given boundary. Then, to preserve the distribution during expansion, we adopt semantic momentum injection, which embeds the Navigator Token's hidden state as a scaled semantic prior into newly added mask tokens. Additionally, with the Navigator's awareness of the remaining length, NaVi introduces Schedule Performance Index (SPI), the ratio of predicted to remaining length, to dynamically adjust the decoding threshold during generation, allowing more efficient token acceptance under high SPI, while raising thresholds under low SPI to ensure higher output quality.

Extensive experiments conducted on four dLLMs across diverse benchmarks demonstrate the effectiveness and generalization capabilities of NaVi. Our contributions are summarized as follows:

- We identify the existence of latent length awareness within dLLMs. We demonstrate that this awareness is linearly encoded in the hidden states of a specific token, termed the Navigator Token, which can be extracted using a lightweight linear regression probe.
- We propose NaVi, a training-free variable-length generation strategy. By synergizing initial budget prediction with distribution-preserving dynamic adjustments, NaVi enables efficient, dynamic generation under the guidance of the Navigator Token.
- We conduct extensive evaluations on four models, including LLaDA-8B-Instruct (Nie et al., 2025), LLaDA-1.5 (Zhu et al., 2026), Dream-v0-7B-Instruct (Ye et al., 2025), and LLaDA2.0-mini (Bie et al., 2025), using the same set of hyperparameters. The results on seven benchmarks demonstrate NaVi's effectiveness and generalization ability.

2 Background and Motivation

2.1 Preliminaries

Inference of dLLMs. Given a vocabulary $\mathcal{V}$ with a mask token `[MASK]`, dLLMs (Nie et al., 2025) generate a sequence $\mathbf{x}_{\text{gen}}$ of length L_g conditioned on a prompt $\mathbf{x}_{\text{prompt}}$ of length L_p via parallel denoising over $t \in \{T, \dots, 0\}$. The concatenated sequence $\mathbf{x} \in \mathcal{V}^L$, where $L = L_p + L_g$, is initialized as:

$$\mathbf{x}^T = \big(\mathbf{x}_{\text{prompt}}, \underbrace{\texttt{[MASK]}, \dots, \texttt{[MASK]}}_{L_g \text{ tokens}}\big). \tag{1}$$

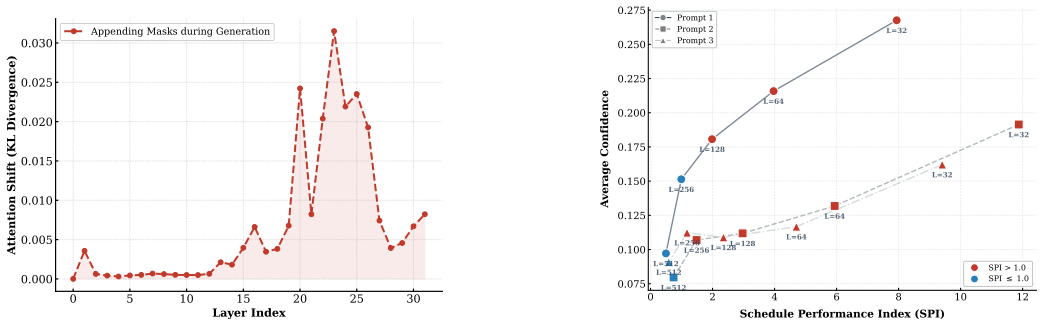
At each step, the dLLM predicts tokens for masked positions:

$$P(\mathbf{x}_i = v | \mathbf{x}^t) = \text{Softmax}(\mathbf{z}_\theta(\mathbf{x}^t)_i)_v, \tag{2}$$

where $\mathbf{z}_\theta(\cdot)_i \in \mathbb{R}^{|\mathcal{V}|}$ denotes the pre-softmax logits. The dLLM iteratively unmasks predicted tokens until $\mathbf{x}^0$ is obtained, but *requires the generation length L_g to be allocated before decoding*.

Extracting Hidden State of the Navigator Token. To enable variable-length generation, we investigate whether dLLMs' intrinsic representations encode the remaining output length. Specifically, let $\mathbf{x}_{<k}$ denote the partially denoised sequence up to the verified position k. We append one [MASK] token to construct:

$$\mathbf{x}_{\text{probe}} = (\mathbf{x}_{\text{prompt}}, \mathbf{x}_{<k}, [\texttt{MASK}]). \tag{3}$$



(a) (b)

Figure 3: **Empirical observations on LLaDA-1.5. (a)** We simulate a mid-generation expansion by suddenly adding 32 [MASK] tokens. The massive KL divergence shift in the attention distributions of deeper layers indicates that abruptly adding tokens during decoding disrupts the model's learned distribution. **(b)** We evaluate three different prompts under varying allocated generation budgets ($L \in \{32, \ldots, 512\}$). As the Schedule Performance Index (SPI, the ratio of predicted length to allocated budget) increases beyond 1.0, the model's average confidence rises correspondingly. This highlights the necessity for dynamic thresholding.

We then perform a single forward pass over $\mathbf{x}_{\text{probe}}$. Let $\mathcal{F}_\ell(\cdot)$ denote the transformation function up to the ℓ-th transformer layer of the dLLM. We extract the hidden state corresponding to the terminal

$$\mathbf{h}^{(\ell)} = [\mathcal{F}_\ell(\mathbf{x}_{\text{probe}})]_{-1}\,. \tag{4}$$

We hypothesize that this hidden state encodes the generation intent of the remaining length. Because this extracted representation can act as a guide for the dLLM to achieve variable-length generation, we designate this [MASK] token as the Navigator Token, and denote its latent representation as $\mathbf{h}^{(\ell)}_{\text{nav}}$.

2.2 Key Observations

Latent encoding of remaining generation length. To verify whether the model's intrinsic representations possess the capability to estimate the remaining output length, we collect inference samples using a single dataset. Specifically, we extract the Navigator Token's hidden state $\mathbf{h}^{(\ell)}_{\text{nav}}$ (as defined in Eq. equation 4) at predefined truncated ratios of the ground-truth response. We then fit a linear regression probe to predict the log-transformed remaining length based on these extracted states. As shown in Fig. 2, predictions from the linear regression probe closely match the actual remaining lengths in the validation set, consistently achieving high R^2 scores. This indicates $\mathbf{h}^{(\ell)}_{\text{nav}}$ effectively captures the dLLM's latent awareness of generation length.

The distribution disruption induced by blank masks. Furthermore, we observe an additional phenomenon regarding the disruption of the decoding distribution during token expansion. As illustrated in Fig. 3a, directly adding [MASK] tokens during decoding triggers attention shifts in deeper layers, degrading both generation quality and speed, even trapping the model in repetitive generation loops (see Appendix D.7). Therefore, another critical challenge in dynamic expansion during decoding is how to add new tokens without significantly disrupting the model's decoding distribution. Our study reveals that these distribution shifts can be effectively mitigated by introducing the Navigator Token's hidden state into the newly added masks (Sec. 4).

The above observations demonstrate two primary properties. First, the Navigator Token possesses the inherent ability to estimate remaining length during decoding. Second, introducing the Navigator Token's hidden state

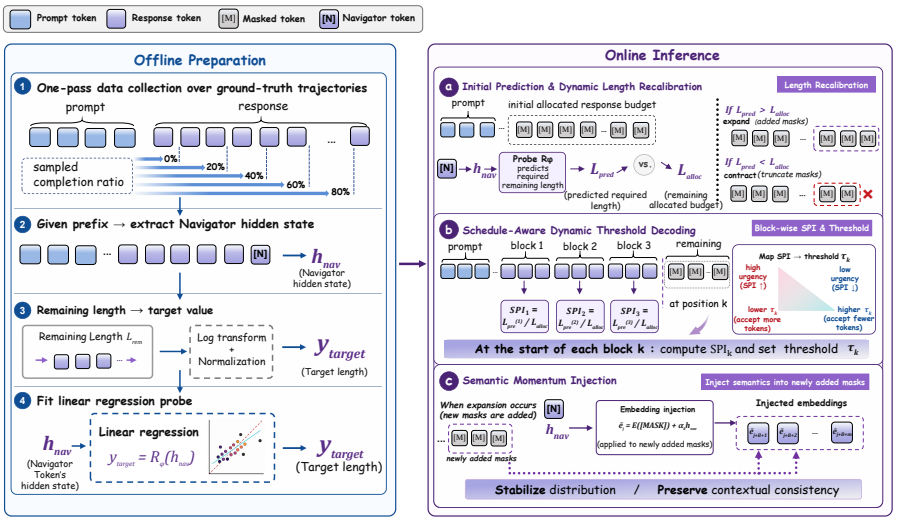


Figure 4: **Overview of NaVi.** *Offline Preparation* (Left): We perform a one-pass data collection on ground-truth trajectories across various predefined ratios. We extract the hidden state of the Navigator Token and fit a linear probe to map it to the normalized log-transformed remaining length. *Online Inference* (Right): During block-wise inference, NaVi uses the probe for *(a) Initial Length Prediction* and triggers *Length Recalibration* for expansion or truncation when the remaining budget falls below the recalibration threshold. When expanded, *(c) Semantic Momentum Injection* uses the Navigator Token's hidden state as a semantic prior to preserve the distribution. Concurrently, *(b) Schedule-Aware Dynamic Threshold Decoding* is applied to adjust the decoding threshold per block, balancing both the generation quality and speed dynamically.

during expansion can mitigate distributional disturbance. Therefore, these findings support our proposal that the Navigator Token can serve as a reliable indicator of remaining output, enabling variable-length generation without compromising the model's original distribution.

3 Methodology

3.1 Overview

Inspired by the insights discussed in Sec. 2.2, we propose NaVi to break the fixed-length constraint of dLLMs. As illustrated in Fig. 4, NaVi addresses the inefficiencies and distribution shifts common in existing variable-length generation strategies by incorporating a predictive and dynamic length adjustment mechanism guided by the Navigator Token. Specifically, NaVi operates in two stages:

First, during the *Offline Preparation* phase (Sec. 3.2), to transform the latent length awareness into an explicit length representation, we construct a lightweight regression probe $\mathcal{R}_\phi$ on ground-truth generation trajectories at various completion ratios. The probe can estimate the remaining length from the Navigator Token's hidden state, establishing a reliable predictive prior for online dynamic adjustment of the sequence length.

Second, in the *Online Inference* phase (Sec. 3.3), NaVi initializes the generation budget through *Initial Prediction*. During block-wise decoding, it introduces the *Schedule Performance Index (SPI)* to adaptively adjust decoding thresholds. When the allocated budget becomes insufficient, NaVi performs *Dynamic Length Recalibration* to expand or truncate the sequence based on updated predictions. During expansion, *Semantic*

Momentum Injection incorporates the Navigator Token's hidden state into newly added masks as a semantic prior, preserving contextual consistency and generation quality.

The details of NaVi are presented in the following sections, with the algorithm provided in Appendix A.

3.2 Offline Preparation

The objective of offline preparation is to fit a regressor that dynamically estimates the remaining length from the decoded context. Based on the observation in Sec. 2.2, the Navigator Token's hidden state encodes latent length awareness, which enables this prediction. We thus leverage this latent information to fit the regressor.

In detail, as shown in Fig. 4, to construct the fitting set $\mathcal{D}$, we sample correct responses from a single dataset (such as 700 MATH (Hendrycks et al., 2021) samples) with sufficient generation budgets (Appendix E.1). For each response $\mathbf{x}^*_{\text{gen}}$, we derive fitting pairs at various decoding stages by truncating it at ratios $\mathcal{R} = \{0.0, 0.2, \ldots, 0.8\}$. Specifically, for each $r \in \mathcal{R}$, we generate a prefix $\mathbf{x}_{<k}$ at index $k = \lfloor |\mathbf{x}^*_{\text{gen}}| \cdot r \rfloor$, with the corresponding target remaining length $L_{\text{rem}} = |\mathbf{x}^*_{\text{gen}}| - k$. For each prefix, we construct the probing sequence $\mathbf{x}_{\text{probe}}$ according to Eq. 3 and extract the $\mathbf{h}^{(\ell)}_{\text{nav}}$ following Eq. 4 as inputs for the regressor.

Given the heavy-tailed distribution of generation lengths (analysis is in Appendix D.4), we apply a logarithmic transformation followed by Min-Max normalization to the target variable:

$$y_{\text{target}} = \text{MinMax}\big(\log(1 + L_{\text{rem}})\big) \in [0, 1]. \tag{5}$$

We fit a ridge regression probe $\mathcal{R}_\phi(\cdot)$ with learnable parameters ϕ, to minimize the regularized squared error:

$$\min_{\phi} \sum_{i \in \mathcal{D}} \left\| \mathcal{R}_\phi\big(\mathbf{h}^{(\ell)}_{\text{nav},i}\big) - y_{\text{target},i} \right\|_2^2 + \lambda \|\phi\|_2^2, \tag{6}$$

where λ acts as the L_2 regularization coefficient. The optimal layer $\ell^* \in \{1, \ldots, L_{\text{last}}\}$ (where L_{last} denotes the final layer of the model architecture) and the coefficient λ are determined automatically by maximizing the R^2 on a held-out validation set without manual tuning (detailed in Appendix E.2).

3.3 Online Inference

During inference, the dLLM decodes block by block with block size B. Let k denote the number of currently verified tokens, and define the current decoding window as $[k, k + B)$. Within this, NaVi leverages the hidden state $\mathbf{h}^{(\ell^*,k)}_{\text{nav}}$ and probe $\mathcal{R}_\phi$ through three coordinated strategies achieve effective variable-length generation:

Initial Prediction and Dynamic Length Recalibration. As shown in Fig. 4 (a), NaVi initializes the generation budget $\hat{L}^{(0)}_{\text{pre}}$ by predicting the remaining length from the initial Navigator Token state $\mathbf{h}^{(\ell^*,0)}_{\text{nav}}$:

$$\hat{L}^{(0)}_{\text{pre}} = \exp\Big(\text{MinMax}^{-1}\big(\mathcal{R}_\phi(\mathbf{h}^{(\ell^*,0)}_{\text{nav}})\big)\Big) - 1. \tag{7}$$

The sequence is initialized with $\hat{L}^{(0)}_{\text{pre}}$ mask tokens. During inference, NaVi triggers recalibration when the allocated budget L_{alloc} falls below the margin threshold $L_{\text{margin}} = C_{\text{margin}} \times B$, where C_{margin} denotes the number of margin blocks. Specifically, when $L_{\text{alloc}} \leq L_{\text{margin}}$, NaVi extracts $\mathbf{h}^{(\ell^*,k)}_{\text{nav}}$ to update $\hat{L}^{(k)}_{\text{pre}}$ via Eq. 7. The sequence length is then adjusted by adding or truncating $\Delta L = |\hat{L}^{(k)}_{\text{pre}} - L_{\text{alloc}}|$ tokens.

Semantic Momentum Injection. As detailed in Sec. 2.2, directly adding [MASK] tokens often disrupts the distribution, thereby degrading performance. To address this, we inject informative priors for the newly added

Table 1: **Main results on LLaDA-8B-Instruct and LLaDA-1.5 across five benchmarks.** We compare NaVi (ours) against DAEDAL, $\rho - \texttt{EOS}$ (64 Sym and 64 Asym), and various fixed-length denoising baselines. ***Acc*** denotes task accuracy; $\boldsymbol{T}_{\text{runtime}}$ is the total evaluation time in seconds; $\boldsymbol{E}_{\text{token}}$ represents the effective tokens. The optimal baseline configuration for each benchmark is highlighted in gray, while DAEDAL, $\rho - \texttt{EOS}$, and our method are highlighted in yellow, green, and blue, respectively. The best results are **bolded**, and the second-best results are underlined.

Benchmark	Metric	Fixed-Length Denoising (Baseline)						DAEDAL	$\rho - \texttt{EOS}$		NaVi (ours)
		64	128	256	512	1024	2048	64	64 Sym	64 Asym	
LLaDA-8B-Instruct											
GSM8K	***Acc***	60.4	74.8	77.7	77.9	77.0	77.0	79.3	77.2	78.7	**79.5**
	$\boldsymbol{T}_{\text{runtime}}$	6854	14329	31752	101692	205201	896426	33328	18019	22490	**14686**
	$\boldsymbol{E}_{\text{token}}$	62.6	124.2	232.5	242.6	243.0	248.9	272.4	234.5	262.8	263.2
HumanEval	***Acc***	6.1	30.5	39.6	43.9	**45.1**	42.1	40.2	39.6	42.1	44.5
	$\boldsymbol{T}_{\text{runtime}}$	241	556	1502	4444	14616	55389	3122	1475	1502	**1179**
	$\boldsymbol{E}_{\text{token}}$	60.0	107.3	241.2	475.0	545.2	514.3	364.2	382.4	402.4	364.3
MATH-500	***Acc***	29.6	35.6	38.0	40.6	43.0	42.0	43.4	39.8	43.4	**43.6**
	$\boldsymbol{T}_{\text{runtime}}$	16365	25925	34181	44716	65502	98567	40182	20012	29840	**6479**
	$\boldsymbol{E}_{\text{token}}$	62.4	124.8	247.0	428.7	605.3	746.2	595.7	423.5	584.3	371.8
IFEval	***Acc***	65.6	68.0	70.0	70.7	68.6	69.7	70.1	65.7	68.7	**71.2**
	$\boldsymbol{T}_{\text{runtime}}$	647	1448	3877	13079	41853	96845	8987	5285	5345	**3036**
	$\boldsymbol{E}_{\text{token}}$	48.5	96.8	173.5	295.2	396.2	653.6	252.6	398.2	402.9	565.7
MBPP	***Acc***	33.0	34.8	**36.8**	35.4	34.0	34.8	35.8	32.6	36.0	**36.8**
	$\boldsymbol{T}_{\text{runtime}}$	2040	4323	10312	24595	69153	228290	10796	4582	4431	**3696**
	$\boldsymbol{E}_{\text{token}}$	54.3	107.5	127.7	228.0	300.5	157.9	144.7	157.8	165.2	179.2
LLaDA-1.5											
GSM8K	***Acc***	60.9	77.7	79.1	80.8	82.0	82.7	**84.4**	77.9	82.6	83.6
	$\boldsymbol{T}_{\text{runtime}}$	1615	14579	31986	96885	203739	874209	28170	15944	16978	**15851**
	$\boldsymbol{E}_{\text{token}}$	62.0	124.8	233.3	271.2	247.3	248.2	281.6	206.5	262.4	278.5
HumanEval	***Acc***	10.4	29.3	40.2	41.5	**47.0**	46.3	42.1	43.3	44.5	**47.0**
	$\boldsymbol{T}_{\text{runtime}}$	240	548	1482	4475	14601	55293	2427	1356	1902	**1232**
	$\boldsymbol{E}_{\text{token}}$	61.0	111.1	247.1	476.4	552.5	524.2	451.0	391.2	454.2	388.2
MATH-500	***Acc***	30.2	35.6	40.2	44.2	43.6	44.6	44.8	42.2	44.2	**46.2**
	$\boldsymbol{T}_{\text{runtime}}$	30993	33276	37939	45683	65573	99580	19862	11244	10233	**8466**
	$\boldsymbol{E}_{\text{token}}$	62.3	125.7	247.9	432.1	606.6	726.0	576.5	566.0	592.3	484.2
IFEval	***Acc***	69.9	72.2	73.1	73.5	**73.9**	70.5	71.9	61.5	70.5	73.6
	$\boldsymbol{T}_{\text{runtime}}$	648	1429	3847	13000	44180	90582	6609	5095	5831	**3674**
	$\boldsymbol{E}_{\text{token}}$	50.2	98.3	176.4	299.8	417.9	592.3	259.4	274.4	302.3	251.7
MBPP	***Acc***	35.0	36.0	**37.0**	34.0	32.8	32.8	35.0	34.8	34.6	36.6
	$\boldsymbol{T}_{\text{runtime}}$	2082	4489	10109	24322	69141	288930	12607	8421	6897	**4450**
	$\boldsymbol{E}_{\text{token}}$	55.9	115.8	167.7	283.0	463.8	223.0	203.4	245.1	252.3	230.2

tokens by leveraging the extracted Navigator Token's hidden state $\mathbf{h}_{\text{nav}}^{(\ell^*,k)}$ as a semantic momentum vector (in Fig. 4 (c)), which will be scaled adaptively by $\alpha_k \in [0, 1]$:

$$\alpha_k = \frac{\hat{L}_{\text{pre}}^{(k)} - L_{\text{alloc}}}{\hat{L}_{\text{pre}}^{(k)}}. \tag{8}$$

Here, $\hat{L}_{\text{pre}}^{(k)}$ denotes the predicted remaining length and L_{alloc} the allocated mask tokens. Thus, α_k measures the expansion intensity. A larger α_k indicates greater expansion and requires stronger semantic momentum to mitigate distribution shift, while a smaller one implies minor adjustments requiring weaker intervention.

Let $\mathcal{E}$ denote the dLLM's input embedding matrix, where $\mathcal{E}([\texttt{MASK}])$ is the embedding of the mask token. For each mask token at position $j \geq k$, where k is the verified position, we update its embedding $\widetilde{\mathbf{e}}_j$ as:

$$\widetilde{\mathbf{e}}_j = \mathcal{E}([\texttt{MASK}]) + \mathbb{I}(j \geq k + B) \cdot \left(\alpha_k \mathbf{h}_{\text{nav}}^{(\ell^*,k)}\right), \tag{9}$$

where $\mathbb{I}(\cdot)$ is the indicator function. The condition $\mathbb{I}(j \geq k + B)$ disables semantic momentum within the current decoding window $[k, k + B)$, thereby restricting momentum injection exclusively to the later decoding space $j \geq k + B$. For LLaDA 2.0 (Bie et al., 2025), we slightly adjust this mechanism to match its block-diffusion architecture to maintain functional consistency with other model architectures (Appendix E.7).

Schedule-Aware Dynamic Threshold Decoding. The threshold-based sampling strategy, which unmasks tokens that exceed a fixed confidence threshold (Wu et al., 2026a), has become a standard practice in dLLMs. However, we argue that applying a static threshold throughout the generation process is suboptimal. Instead, the unmasking threshold should adapt to the remaining token budget during decoding.

To quantify this budget constraint, we introduce the Schedule Performance Index (SPI) (Fig. 4 (b)), computed exactly once at the beginning of each decoding block. Let S_k denote the SPI at verified position k:

$$S_k = \frac{\hat{L}_{\text{pre}}^{(k)}}{L_{\text{alloc}}}, \tag{10}$$

This ratio reflects a budget constraint: a larger S_k indicates that the predicted length exceeds the available budget. As shown in Fig. 3b, more insufficient budget leads to higher confidence, motivating a lower threshold for faster decoding, while sufficient budget allows a higher threshold for better generation quality. Based on this observation, we dynamically map the SPI to an unmasking threshold. We establish two boundary hyperparameters, S_{low} and S_{high}, alongside two threshold bounds, $\tau_{\min} = 0.90$ and $\tau_{\max} = 0.99$ (see Appendix D.8). The dynamic threshold τ_k for the current block is formulated as a linear function:

$$\tau_k = \begin{cases} \tau_{\max}, & \text{if } S_k \leq S_{\text{low}} \\ \tau_{\min}, & \text{if } S_k \geq S_{\text{high}} \\ \tau_{\max} - \frac{S_k - S_{\text{low}}}{S_{\text{high}} - S_{\text{low}}}(\tau_{\max} - \tau_{\min}), & \text{otherwise.} \end{cases} \tag{11}$$

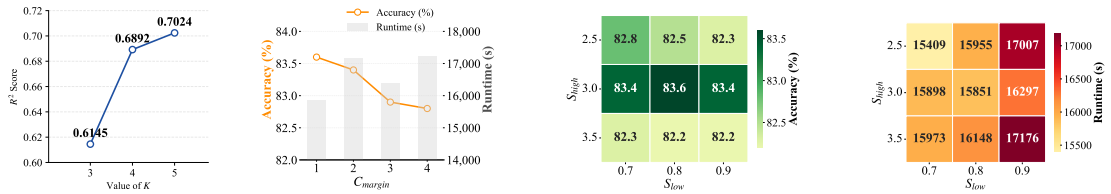
This formulation balances generation quality and speed. When the predicted length exceeds the remaining budget, the threshold approaches $\tau_{\min}$ to accelerate decoding. In contrast, it shifts the threshold toward $\tau_{\max}$ to prioritize output quality. Linear interpolation ensures a smooth transition between these operational bounds.

4 EXPERIMENTS

To comprehensively assess the effectiveness of NaVi, we conduct evaluations across three primary domains: Mathematical Reasoning (GSM8K (Cobbe et al., 2021) and MATH-500 (Lightman et al., 2024)), Code Generation (HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021)), and Instruction Following (IFEval (Zhou et al., 2023)). Furthermore, because LLaDA2.0-mini natively supports long-context output generation, we incorporate more challenging mathematical benchmarks, AIME2024 (AI-MO, 2024) and AIME2025 (OpenCompass, 2025), specifically to evaluate NaVi's robustness on long-context tasks. For each model, a regression probe is fitted offline on 700 sampled problems from MATH and remains frozen across all benchmarks. The effects of different numbers of samples are studied in Appendix D.5. Detailed experimental settings and baseline configurations are provided in Appendix E.3, E.4 and E.5.

4.1 MAIN RESULTS

NaVi obviates the need for task-specific length tuning by operating effectively under a unified configuration. The experimental results, summarized in Table 1, Table 2, and Table 3, demonstrate that NaVi consistently matches or surpasses both fixed-length denoising baselines and existing variable-length generation methods across nearly all benchmarks. Notably, compared to DAEDAL and $\rho - \texttt{EOS}$, NaVi achieves a maximum



(a) (b) (c) (d)

Figure 5: **Experimental results on hyperparameter sensitivity.** All experiments are conducted on LLaDA-1.5. (a) R^2 performance across different K values. (b) Sensitivity of C_{margin} to both accuracy and run time. (c) (d) Sensitivity of SPI bounds S_{low} and S_{high} to accuracy and runtime.

runtime reduction of 83.8% and an **absolute accuracy improvement of 2.7%**. Furthermore, the performance advantages of NaVi also extend to causal block-diffusion models like LLaDA2.0-mini, highlighting its superior cross-model generalizability. On this backbone, where previous methods fail to remain applicable (Appendix E.3), NaVi also enables an improvement in accuracy while enhancing inference efficiency.

4.2 Ablations and Analysis

Effect of predefined truncated ratios $\mathcal{R}$. We analyze the impact of truncated ratios $\mathcal{R}$ used for constructing the probe fitting data in Sec. 3.2. Specifically, we vary the number of intermediate sampling stages, denoted as K, and record the best validation R^2. For example, $K = 3$ corresponds to $\mathcal{R} = \{0\%, 33\%, 67\%\}$. As shown in Fig. 5a, increasing K improves R^2 from 0.6145 to 0.7024 on LLaDA-1.5, indicating that finer-grained sampling better captures the correlation between Navigator Token hidden states and remaining length. However, the gain diminishes with larger K, and we adopt $K = 5$ for all models.

Effect of Semantic Momentum Injection. We evaluate the necessity of the *Semantic Momentum Injection* mechanism. As shown in Table 6 (Appendix D.3), with momentum injection, NaVi maintains or improves performance on all benchmarks. More importantly, it substantially reduces the occurrences of repetitive loops (denoted as N), demonstrating its effectiveness in preserving the decoding distribution.

Impact of recalibration margin C_{margin}. We further analyze the effect of the recalibration margin C_{margin}, which determines when the model triggers length recalibration. As illustrated in Fig. 5b, increasing C_{margin} leads to higher runtime, while slightly degrading accuracy. In particular, $C_{\text{margin}} = 1$ achieves the best trade-off, yielding the highest accuracy with the lowest runtime, mitigating unnecessary recalibration overhead and preventing excessive interruptions to the generation process.

Sensitivity to SPI bounds S_{low} and S_{high}. We study the sensitivity of the Schedule Performance Index (SPI) by varying S_{low} and S_{high}. As shown in Fig. 5c and 5d, the performance remains stable across a wide range of configurations, with accuracy consistently around 83%. The best performance is achieved at $(S_{\text{low}}, S_{\text{high}}) = (0.8, 3.0)$, which is also the default setting. Notably, runtime exhibits more significant variance than accuracy, indicating that SPI primarily modulates generation efficiency. However, the overall performance remains stable, showing the robustness of the NaVi to SPI bounds.

5 Concluding Remarks

This paper identifies that the generation length is intrinsically encoded in dLLM's latent spaces. Building on this, our proposed training-free method, NaVi, enables variable-length generation, achieving superior performance and efficiency across various benchmarks and models.

AI USE STATEMENT

In this work, we used generative AI tools to assist with editing the manuscript to improve readability, clarity, and writing quality. We have not used generative AI tools for generating research ideas, formulating hypotheses, designing the methodology or experiments, implementing methods, analyzing or interpreting results, formulating mathematical claims, or other research tasks requiring disclosure. These tasks were performed by the authors. We have reviewed all AI-assisted writing modifications and verified their correctness. We take responsibility for the final content of this paper, including all text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

AI-MO. Aime 2024. https://huggingface.co/datasets/AI-MO/aimo-validation-aime, July 2024.

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:2108.07732*, 2021.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, Jianfeng Tan, Lanning Wei, Ji-Rong Wen, Yipeng Xing, Xiaolu Zhang, Junbo Zhao, Da Zheng, Jun Zhou, Junlin Zhou, Zhanchao Zhou, Liwang Zhu, and Yihong Zhuang. Llada2.0: Scaling up diffusion language models to 100b. *arXiv:2512.15745*, 2025.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, Zhuocheng Gong, Yanmei Gu, Jian Guan, Kaiyuan Guan, Hongliang He, Zenan Huang, Juyong Jiang, Zhonghui Jiang, Zhenzhong Lan, Chengxi Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Yuan Lu, Yuxin Ma, Xingyu Mou, Zhenxuan Pan, Kaida Qiu, Yuji Ren, Jianfeng Tan, Yiding Tian, Zian Wang, Lanning Wei, Tao Wu, Yipeng Xing, Wentao Ye, Liangyu Zha, Tianze Zhang, Xiaolu Zhang, Junbo Zhao, Da Zheng, Hao Zhong, Wanli Zhong, Jun Zhou, Junlin Zhou, Liwang Zhu, Muzhi Zhu, and Yihong Zhuang. Llada2.1: Speeding up text diffusion via token editing. *arXiv:2602.08676*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv:2107.03374*, 2021.

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: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.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv:2407.21783*, 2024.

Marton Havasi, Brian Karrer, Itai Gat, and Ricky T. Q. Chen. Edit flows: Variable length discrete flow matching with sequence-level edit operations. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025*, 2025.

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, 2021. URL `https://arxiv.org/abs/2103.03874`.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv:2410.21276*, 2024.

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

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 *Proceedings of the 14th International Conference on Learning Representations*, 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. In *Proceedings of the 12th International Conference on Learning Representations*, 2024.

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. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, et al. Gpt-4 technical report. *arXiv:2303.08774*, 2024.

OpenCompass. Aime 2025. `https://huggingface.co/datasets/opencompass/AIME2025`, February 2025.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288*, 2023.

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 *Proceedings of the 14th International Conference on Learning Representations*, 2026a.

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*, 2026b.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report. *arXiv:2407.10671*.

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

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv:2412.15115*, 2025b.

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

Yicun Yang, Cong Wang, Shaobo Wang, Zichen Wen, Biqing Qi, Hanlin Xu, and Linfeng Zhang. Diffusion llm with native variable generation lengths: Let [eos] lead the way. *arXiv:2510.24605*, 2025c.

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

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv:2311.07911*, 2023.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. LLaDA 1.5: Variance-reduced preference optimization for large language diffusion models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026.

CONTENTS

A Algorithm of NaVi . . . 14

B Related Work . . . 14

B.1 Diffusion Large Language Models . . . 14

B.2 Variable-Length Generation Strategies in dLLMs . . . 16

C More Evidence of the Latent Length Awareness in dLLMs . . . 16

D More Results and Analyses . . . 17

D.1 Results on Dream-v0-7B-Instruct and LLaDA2.0-mini . . . 17

D.2 Avg@K Performance Analysis . . . 17

D.3 Ablation Study of Semantic Momentum Injection on LLaDA-1.5 . . . 18

D.4 Ablation Study on Log-Transformation for Regression Probe Fitting . . . 18

D.5 Ablation Study on Different Numbers of Samples for Probe Fitting . . . 19

D.6 Ablation Study on SPI . . . 20

D.7 More Analyses of the Repetitive Phenomenon . . . 21

D.8 Discussion on the Selection of Threshold Bounds $\tau_{\min} = 0.90$ and $\tau_{\max} = 0.99$ . . . 22

D.9 Reasons and Analysis for Choosing the MATH Dataset for Probe Fitting . . . 23

D.10 Sensitivity to Decoding Process and Trajectory . . . 24

D.11 Sensitivity to Layer Choice . . . 25

D.12 Exploration to Multi-layer Aggregation . . . 25

D.13 Analysis of the Role of Semantic Momentum Injection . . . 25

E Details on Experimental Settings . . . 26

E.1 Details of Fitting Data Construction . . . 26

E.2 Details of Linear Ridge Regression Probe Fitting . . . 27

E.3 Decoding and Sampling Settings . . . 28

E.4 Implementation Details . . . 28

E.5 Details of Baseline Methods . . . 29

E.6 Details on Benchmarks . . . 29

E.7 Implementation Details of Semantic Momentum Injection in Architectures like LLaDA 2.0 . . . 29

F Case Study . . . 30

A ALGORITHM OF NAVI

The pseudocode of NaVi is shown in Algorithm 1 and 2.

Algorithm 1 Offline Preparation: Linear Regression Probe Fitting

Require: Ground-truth trajectories $\mathcal{D} = \{\mathbf{x}^*_{\text{gen},i}\}_{i=1}^{N}$, Truncation ratios $\mathcal{R}$, a dLLM with layers $\ell \in \{1, \ldots, L_{\text{last}}\}$, Regularization coefficient λ.

Initialize fitting set $\mathcal{D} \leftarrow \emptyset$.
for each $\mathbf{x}^*_{\text{gen}} \in \mathcal{D}$ **and each** $r \in \mathcal{R}$ **do**
  $k \leftarrow \lfloor |\mathbf{x}^*_{\text{gen}}| \cdot r \rfloor, \quad L_{\text{rem}} \leftarrow |\mathbf{x}^*_{\text{gen}}| - k$ ▷ *Calculate ground-truth remaining length*
  $\mathbf{x}_{\text{probe}} \leftarrow (\mathbf{x}_{\text{prompt}}, \mathbf{x}_{<k}, [\texttt{MASK}])$ ▷ *Construct probe sequence*
  for each layer $\ell \in \{1, \ldots, L_{\text{last}}\}$ **do**
    $\mathbf{h}^{(\ell)}_{\text{nav}} \leftarrow [\mathcal{F}_\ell(\mathbf{x}_{\text{probe}})]_{-1}$ ▷ *Extract Navigator Token hidden state*
  end for
  $y_{\text{target}} \leftarrow \text{MinMax}\big(\log(1 + L_{\text{rem}})\big)$ ▷ *Log-transformed target normalization*
  $\mathcal{D} \leftarrow \mathcal{D} \cup \big\{\big(\{\mathbf{h}^{(\ell)}_{\text{nav}}\}_{\ell=1}^{L_{\text{last}}}, y_{\text{target}}\big)\big\}$
end for
Split $\mathcal{D}$ into training set $\mathcal{D}_{\text{train}}$ and validation set $\mathcal{D}_{\text{val}}$.
// Automatically determine the hyperparameters (ℓ, λ) *that maximize* R^2 *on validation set*
$\ell^*, \lambda^* \leftarrow \arg\max_{\ell,\lambda} R^2\,(\mathcal{D}_{\text{val}}, \phi_{\ell,\lambda})$
where $\phi_{\ell,\lambda} = \arg\min_\phi \sum_{i \in \mathcal{D}_{\text{train}}} \|\mathcal{R}_\phi(\mathbf{h}^{(\ell)}_{\text{nav},i}) - y_{\text{target},i}\|_2^2 + \lambda\|\phi\|_2^2$
$\phi^* \leftarrow \phi_{\ell^*,\lambda^*}$
return $\mathcal{R}_{\phi^*}(\cdot), \ell^*$

B RELATED WORK

B.1 DIFFUSION LARGE LANGUAGE MODELS

Text generation with large language models has long been dominated by autoregressive models (ARMs). These models typically generate text token by token in a left-to-right manner, and representative architectures such as Qwen (Yang et al.; 2025b;a), GPT (OpenAI et al., 2024; Hurst et al., 2024), and LLaMA (Touvron et al., 2023; Dubey et al., 2024) have achieved remarkable success in open-domain question answering (Zhou et al., 2023), code generation (Chen et al., 2021), mathematical reasoning (Hendrycks et al., 2021), and other tasks. Despite the strong modeling capabilities and broad applicability demonstrated by the autoregressive paradigm, its inherently sequential decoding mechanism introduces non-negligible limitations. On the one hand, token-by-token generation restricts parallelization during inference, leading to increased generation latency. On the other hand, errors made in early decoding steps may be propagated and amplified as subsequent tokens are generated conditioned on previous outputs, resulting in error accumulation and degraded final output quality.

In recent years, Diffusion Large Language Models (dLLMs) (Nie et al., 2025; Zhu et al., 2026; Bie et al., 2025; 2026; Ye et al., 2025) have emerged as a promising alternative paradigm for language generation. Unlike autoregressive models, which decode tokens unidirectionally and sequentially, DLMs typically corrupt text through a forward noise-injection or masking process and then iteratively denoise it in the reverse process, progressively recovering the masked tokens. In this way, they enable non-autoregressive or partially non-autoregressive text generation. This paradigm allows models to predict tokens at multiple positions

Algorithm 2 Online Inference of NaVi

Require: Prompt $\mathbf{x}_{\text{prompt}}$, Block size B, Number of Margin blocks C_{margin}, Threshold bounds $\tau_{\min}, \tau_{\max}$, SPI bounds $S_{\text{low}}, S_{\text{high}}$, a dLLM with embedding matrix $\mathcal{E}$, Linear Probe $\mathcal{R}_{\phi^*}$, Optimal layer ℓ^*.

$k \leftarrow 0$, $L_{\text{margin}} \leftarrow C_{\text{margin}} \times B$
Extract initial hidden state $\mathbf{h}_{\text{nav}}^{(\ell^*,0)}$ from $\mathbf{x}_{\text{prompt}}$
$\hat{L}_{\text{pre}}^{(0)} \leftarrow \exp\left(\text{MinMax}^{-1}\left(\mathcal{R}_{\phi^*}(\mathbf{h}_{\text{nav}}^{(\ell^*,0)})\right)\right) - 1$ ▷ *Initial budget prediction*
$L_{\text{alloc}} \leftarrow \hat{L}_{\text{pre}}^{(0)}$
$\mathbf{x}_{\text{seq}} \leftarrow \mathbf{x}_{\text{prompt}} \oplus \{[\texttt{MASK}]\}_{i=1}^{L_{\text{alloc}}}$ ▷ $\oplus$ *denotes concatenation*
while generation not terminated **do**
  Extract current state $\mathbf{h}_{\text{nav}}^{(\ell^*,k)}$ given prefix $\mathbf{x}_{<k}$
  $\hat{L}_{\text{pre}}^{(k)} \leftarrow \exp\left(\text{MinMax}^{-1}\left(\mathcal{R}_{\phi^*}(\mathbf{h}_{\text{nav}}^{(\ell^*,k)})\right)\right) - 1$ ▷ *Update prediction*
  $\gamma_k \leftarrow \hat{L}_{\text{pre}}^{(k)} / L_{\text{alloc}}$ ▷ *Calculate Schedule Performance Index (SPI)*
  $\tau_k \leftarrow \max\left(\tau_{\min}, \min\left(\tau_{\max}, \tau_{\max} - \frac{S_k - S_{\text{low}}}{S_{\text{high}} - S_{\text{low}}}(\tau_{\max} - \tau_{\min})\right)\right)$ ▷ *Dynamic Threshold*
  if $L_{\text{alloc}} \leq L_{\text{margin}}$ **then** ▷ *Trigger Dynamic Recalibration*
    $\Delta L \leftarrow \hat{L}_{\text{pre}}^{(k)} - L_{\text{alloc}}$
    if $\Delta L > 0$ **then** ▷ *Expansion Phase*
      $\mathbf{x}_{\text{seq}} \leftarrow \mathbf{x}_{\text{seq}} \oplus \{[\texttt{MASK}]\}_{i=1}^{\Delta L}$
      $\alpha_k \leftarrow \Delta L / \hat{L}_{\text{pre}}^{(k)}$ ▷ *Semantic momentum scale*
      for each $j \geq k + B$ **do** ▷ *Semantic Momentum Injection*
        $\widetilde{\mathbf{e}}_j \leftarrow \mathcal{E}([\texttt{MASK}]) + \alpha_k \mathbf{h}_{\text{nav}}^{(\ell^*,k)}$
      end for
    else if $\Delta L < 0$ **then** ▷ *Truncation Phase*
      Truncate $|\Delta L|$ tokens of $[\texttt{MASK}]$ from the tail of $\mathbf{x}_{\text{seq}}$
    end if
    $L_{\text{alloc}} \leftarrow \hat{L}_{\text{pre}}^{(k)}$
  end if
  // Decoding for the current block $[k, k+B)$
  repeat
    $\mathbf{z}_i \leftarrow \mathbf{z}_\theta(\mathbf{x}_{\text{seq}})_i, \quad \forall i \in \{j \mid \mathbf{x}_j = \texttt{[MASK]}, j \in [k, k+B)\}$
    $\mathcal{P}_i \leftarrow \text{Softmax}(\mathbf{z}_i)$
    $\mathcal{I}_{\text{vld}} \leftarrow \{i \in [k, k+B) \mid \max_{v \in \mathcal{V}} \mathcal{P}_{i,v} > \tau_k\}$
    $\mathbf{x}_{\mathcal{I}_{\text{vld}}} \leftarrow \arg\max_{v \in \mathcal{V}} \mathcal{P}_{\mathcal{I}_{\text{vld}},v}$ ▷ *Decode high-confidence tokens*
  until all $\mathbf{x}_{i \in [k,k+B)} \neq \texttt{[MASK]}$
  $k \leftarrow k + B, \quad L_{\text{alloc}} \leftarrow L_{\text{alloc}} - B$
end while
return $\mathbf{x}_{\text{seq}}$

simultaneously at each decoding step, offering new opportunities to overcome the bottlenecks of conventional sequential decoding.

However, their generation process often requires the output length to be specified in advance and performs multiple rounds of denoising over a fixed-length sequence. Consequently, even when the target output is short or the task is relatively simple, the model may still need to carry out the full iterative computation over the predefined length range, incurring unnecessary computational overhead. Motivated by this limitation, our

work focuses on the efficiency bottleneck introduced by fixed-length inference and explores a more adaptive mechanism for diffusion-based language generation.

B.2 Variable-Length Generation Strategies in dLLMs

The predefined fixed-length constraint in masked dLLMs has recently prompted a surge of research aimed at equipping dLLMs with variable-length generation capabilities. Existing strategies can be categorized into training-based methods and training-free methods.

Training-Based Methods. Several recent works propose modifying the diffusion training objective or introducing auxiliary tokens to learn length control implicitly or explicitly. For instance, Kim et al. (2025) decouples the discrete diffusion process into explicit insertion and unmasking steps, training a Flexible Masked Diffusion Model (FlexMDM) to predict the expected number of mask tokens to insert. DreamOn (Wu et al., 2026b) augments the vocabulary with special control states (e.g., `[expand]` and `[delete]`), training the model to autonomously expand or contract the masked sequence based on self-predicted decisions. Similarly, dLLM-Var (Yang et al., 2025c) trains models to natively predict `[EOS]` tokens under a block-diffusion setup, dynamically padding blocks of masks until termination. Recent works further explore training-based approaches for variable-length generation through edit-based diffusion objectives and deletion-insertion mechanisms (Havasi et al., 2025; Ding et al., 2026). While these methods achieve promising variable-length capabilities, they inherently require computationally expensive re-training or fine-tuning, which limits their plug-and-play applicability to existing dLLM architectures.

Training-Free Methods. To avoid the overhead of re-training, concurrent training-free methods attempt to achieve variable-length generation only through inference-time heuristics. DAEDAL (Li et al., 2026) proposes a two-stage approach: it first iteratively expands a short initial sequence by appending masks if trailing `[EOS]` confidence is low, and subsequently performs iterative mask insertions during the actual denoising process to patch low-confidence regions. However, this multi-stage pipeline introduces additional latency and supports only unidirectional expansion, lacking the ability to contract overly long sequences. Alternatively, ρ-EOS (Yang et al., 2026) operates in a single stage, dynamically appending or truncating mask tokens globally based on the implicit `[EOS]` density monitored during the denoising steps.

Despite their efficiency improvements, these training-free methods fundamentally rely on the frequent and abrupt appending of blank `[MASK]` tokens during the generation trajectory. As we established in our study, because dLLMs are trained on fixed-length, continuous text trajectories, such operations inevitably violate the expected Markovian transitions (Kim et al., 2025), degrading both output quality and inference speed.

C More Evidence of the Latent Length Awareness in dLLMs

dLLMs like LLaDA-8B-Instruct (Nie et al., 2025), LLaDA-1.5 (Zhu et al., 2026), and Dream-v0-7B-Instruct (Ye et al., 2025) adopt a global bidirectional attention mechanism over the entire sequence. Under this paradigm, the generation process can be explicitly influenced by a predefined fixed generation length, allowing the model's latent length awareness to be directly reflected in its outputs. As illustrated in Fig. 6, when the generation budget is severely constrained, dLLMs tend to produce concise yet complete responses, whereas autoregressive (AR) models continue generating tokens until the maximum length is reached. This phenomenon provides a clear empirical manifestation of latent length awareness in dLLMs. It also reveals the limitations of using a fixed generation length, which motivates the need for adaptive, variable-length generation strategies.

In contrast, LLaDA-2.0 (Bie et al., 2025), which employs a block-wise diffusion architecture, no longer exhibits the behavior in Fig. 6. This different attention mechanism fundamentally alters its generation paradigm,

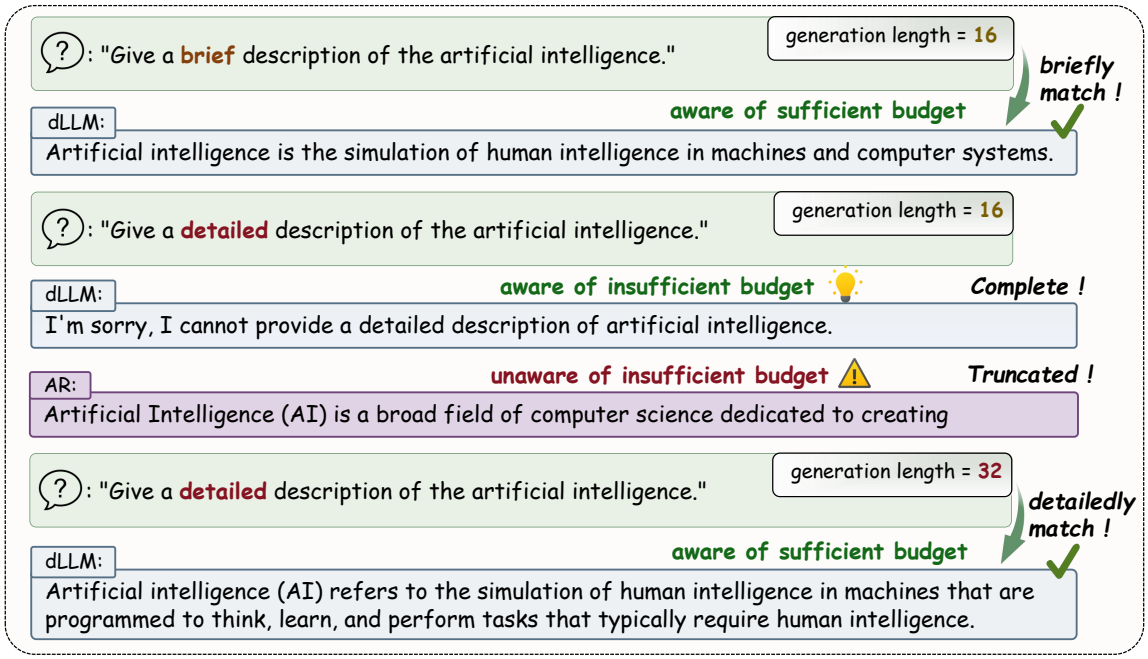


Figure 6: **Illustration of latent length awareness in dLLMs.** We conduct a qualitative comparison of generation under length constraints across dLLMs and ARs. While AR models generate text until they are truncated by the maximum length budget, dLLMs produce complete and coherent outputs that match the allocated budget.

making it more akin to AR models rather than conventional dLLMs which adopt a global bidirectional attention. Despite these architectural differences, NaVi consistently achieves superior performance across both paradigms. This suggests that our approach actually captures the latent length awareness at a more fundamental level, leading to improved robustness and generalization.

D More Results and Analyses

D.1 Results on Dream-v0-7B-Instruct and LLaDA2.0-mini

The detailed results on Dream-v0-7B-Instruct and LLaDA2.0-mini are summarized in Table 2 and 3.

D.2 Avg@K Performance Analysis

Table 4 presents the performance of our method on small-scale benchmarks (AI-MO, 2024; OpenCompass, 2025) using the Avg@30 metric. The results demonstrate consistent accuracy improvements across these tasks. Notably, the relatively high standard deviations observed on these smaller dataset demonstrate the importance of utilizing Avg@30 to ensure a more robust and reliable performance assessment.

Table 2: **Experimental results on Dream-0-7B-Instruct across five benchmarks.** We compare NaVi (ours) against DAEDAL, $\rho - \texttt{EOS}$ (Sym and Asym), and various fixed-length denoising baselines. ***Acc*** denotes task accuracy; $\boldsymbol{T}_{\text{runtime}}$ is the total evaluation time; $\boldsymbol{E}_{\text{token}}$ represents the effective tokens. The optimal baseline configuration is highlighted in gray, while DAEDAL, $\rho - \texttt{EOS}$, and our method are highlighted in yellow, green, and blue, respectively. The best results are **bolded**, and the second-best results are underlined.

Benchmark	Metric	Fixed-Length Denoising (Baseline)						DAEDAL	$\rho - \texttt{EOS}$		NaVi (ours)
		64	128	256	512	1024	2048	64	64 Sym	64 Asym	
GSM8K	***Acc***	48.5	70.8	**75.4**	73.8	73.8	73.5	71.4	72.0	73.2	74.9
	$\boldsymbol{T}_{\text{runtime}}$	6395	11276	24289	62296	120894	267894	27437	18909	20003	**7508**
	$\boldsymbol{E}_{\text{token}}$	62.3	112.7	129.5	126.4	125.7	124.9	121.1	131.4	136.9	123.8
HumanEval	***Acc***	49.4	54.9	59.2	54.9	57.9	59.8	61.0	59.8	59.8	**62.8**
	$\boldsymbol{T}_{\text{runtime}}$	304	669	1556	4642	12762	44361	3354	2956	2785	**607**
	$\boldsymbol{E}_{\text{token}}$	55.0	105.1	140.1	144.8	130.1	151.3	131.3	151.3	154.9	109.8
MATH-500	***Acc***	29.8	41.6	45.0	**45.8**	43.6	43.0	44.8	42.8	43.4	44.8
	$\boldsymbol{T}_{\text{runtime}}$	1877	4277	8524	19059	54623	130245	7106	6183	6589	**4052**
	$\boldsymbol{E}_{\text{token}}$	60.1	115.0	183.5	238.6	293.6	284.5	234.0	200.3	210.4	198.7
IFEval	***Acc***	44.7	**51.4**	50.8	48.9	46.8	47.8	46.0	40.1	44.7	50.0
	$\boldsymbol{T}_{\text{runtime}}$	831	1633	3996	10332	37085	69764	7464	6893	7001	**2610**
	$\boldsymbol{E}_{\text{token}}$	49.5	95.2	163.2	257.0	354.9	587.8	204.4	282.3	300.2	164.1
MBPP	***Acc***	41.4	46.0	47.2	47.2	47.8	47.8	48.0	46.4	46.8	**48.8**
	$\boldsymbol{T}_{\text{runtime}}$	1497	4018	7394	12173	43632	90234	4365	4255	3889	**1388**
	$\boldsymbol{E}_{\text{token}}$	45.1	49.4	50.9	55.9	67.2	66.9	55.2	102.0	104.5	55.7

Table 3: **Main results on LLaDA2.0-mini across various benchmarks.** We compare NaVi (ours) against the baseline. The best results are **bolded**.

	Metric	HumanEval	MBPP	IFEval	MATH-500	AIME2024	AIME2025
baseline	***Acc***	**84.2**	56.8	87.5	90.6	46.7	36.7
	$\boldsymbol{T}_{\text{runtime}}$	14882	85341	33132	184759	45197	49846
	$\boldsymbol{E}_{\text{token}}$	841.8	1228.5	379.3	2445.4	8247.9	8890.3
NaVi (ours)	***Acc***	**84.2**	**59.4**	**87.9**	**91.2**	**53.3**	**40.0**
	$\boldsymbol{T}_{\text{runtime}}$	**10148**	**76489**	**27865**	**158976**	**39824**	**40175**
	$\boldsymbol{E}_{\text{token}}$	774.4	1283.6	405.2	2131.6	7746.8	7714.9

D.3 Ablation Study of Semantic Momentum Injection on LLaDA-1.5

The ablation results of Semantic Momentum Injection are reported in Table 6.

D.4 Ablation Study on Log-Transformation for Regression Probe Fitting

In Sec. 3.2, we apply a logarithmic transformation to the target remaining length L_{rem} before performing Min-Max normalization. To validate the necessity of this operation, we conduct an ablation study comparing the best validation R^2 scores achieved by the linear regression probe with and without the log-transformation across four evaluated models.

As shown in Table 5, incorporating the log-transformation consistently yields a substantial improvement in the R^2 score across all models. For instance, on Dream-v0-7B-Instruct, the R^2 score increases from

Table 4: **Avg@30 performance with standard deviation on mathematical reasoning benchmarks.**

	AIME2024		AIME2025	
Method	**Avg@30↑**	**Std**	**Avg@30↑**	**Std**
LLaDA2.0-mini				
Baseline	46.6	0.0087	36.7	0.0150
NaVi (Ours)	**53.2**	0.0136	**39.0**	0.0222

Table 5: **Effect of log-transformation on regression probe fitting performance.** We report the optimal transformer layer (ℓ^*) and the highest validation R^2 score achieved by the linear ridge regression probe on four dLLMs, with and without the application of logarithmic transformation on the target length.

Model	**Configuration**	**Optimal Layer (ℓ^*)**	**Best R^2 Score**
LLaDA-8B-Instruct	without log-transform	19	0.6137
	with log-transform	**23**	**0.6833**
LLaDA-1.5	without log-transform	25	0.5834
	with log-transform	**22**	**0.7024**
Dream-v0-7B-Instruct	without log-transform	20	0.5212
	with log-transform	**19**	**0.7202**
LLaDA2.0-mini	without log-transform	12	0.5058
	with log-transform	**12**	**0.7063**

0.5212 to 0.7202. This significant gain stems from the intrinsic heavy-tailed nature of text generation lengths; the absolute variance is much larger for longer sequences. By applying the logarithmic transformation, we effectively compress the long-tail variance, rendering the target space more linear and stable. Consequently, the linear ridge regression probe can capture the underlying correlation much more accurately. The optimal transformer layers (ℓ^*) derived from the grid search for each configuration are also reported.

D.5 Ablation Study on Different Numbers of Samples for Probe Fitting

As described in Sec. 4, the linear regression probe $\mathcal{R}_\phi$ for NaVi is fitted offline using exactly 700 randomly sampled ground-truth trajectories. To justify this choice and verify the robustness of our length estimation mechanism to the size of the fitting data, we conduct an ablation study by varying the number of samples ($N \in \{600, 700, 800\}$).

The validation R^2 scores and the optimal transformer layers (ℓ^*) identified are presented in Table 7. We draw two conclusions from these results, as listed in the following.

First, the predictive performance of the linear probe is stable across different sample sizes, consistently achieving high R^2 scores around 0.67 to 0.72. Utilizing 700 samples generally yields the optimal or near-optimal peak performance across all evaluated models. Second, the optimal layer (ℓ^*) remains nearly invariant regardless of the sample size fluctuations (e.g., robustly anchored at layer 23 for LLaDA-8B-Instruct and layer 19 for Dream-v0-7B-Instruct). This phenomenon also provides empirical evidence that latent length awareness is an intrinsic property of the dLLMs.

Table 6: **Ablation study of Semantic Momentum Injection on LLaDA-1.5.** Red indicates degradation after removing semantic momentum, including lower performance and more repetitive loops (N).

Configuration	Metric	GSM8K	MATH-500	HumanEval	MBPP	IFEval
with Momentum	$N \downarrow$	0	9	0	1	1
	Acc (%) $\uparrow$	83.6	46.2	47.0	36.6	73.6
w/o Momentum	$N \downarrow$	4 (+4)	30 (+21)	1 (+1)	4 (+3)	49 (+48)
	Acc (%) $\uparrow$	82.3 (-1.3)	45.6 (-0.6)	47.0 (0.0)	36.2 (-0.4)	72.9 (-0.7)

Table 7: **Effect of fitting samples on probe performance.** The table reports the optimal transformer layer (ℓ^*) and the corresponding validation R^2 score under different numbers of fitting samples (600, 700, and 800) across four dLLMs. Our default configuration (700 samples) is highlighted in **bold**.

Model	# of Samples	Optimal Layer (ℓ^*)	Best R^2 Score
LLaDA-8B-Instruct	600	23	0.6678
	700	**23**	**0.6833**
	800	23	0.6743
LLaDA-1.5	600	22	0.7102
	700	**22**	**0.7024**
	800	21	0.6897
Dream-v0-7B-Instruct	600	19	0.7004
	700	**19**	**0.7202**
	800	19	0.6980
LLaDA2.0-mini	600	13	0.6653
	700	**12**	**0.7063**
	800	12	0.7112

D.6 Ablation Study on SPI

To evaluate the effectiveness of SPI, we conduct an ablation study comparing NaVi with a variant in which SPI is replaced by a standard fixed-confidence strategy, across two models: LLaDA-8B-Instruct and LLaDA-1.5. We compare them across three distinct domains. For the fixed-confidence variant, we use a threshold of $\tau = 0.9$ while keeping all other experimental settings identical to those used in Sec. 4. We additionally report the best fixed-length baseline for reference. The results are summarized in Table 8.

Overall, NaVi remains effective even without SPI. With the fixed-confidence strategy, it already achieves accuracy comparable to the best fixed-length baseline while substantially reducing runtime, confirming that NaVi's effectiveness doesn't depend on SPI. SPI prioritizes accuracy while keeping runtime comparable. Across all models and benchmarks, introducing SPI either improves or preserves accuracy. In some cases, these accuracy gains come with a moderate runtime overhead, while in others, SPI improves both accuracy and efficiency. In variable-length generation, accuracy is our primary concern. SPI better captures the dynamic variation in token confidence during generation, enabling NaVi to allocate the decoding budget more adaptively and achieve a superior balance between generation quality and efficiency, thus safeguarding accuracy at a comparable runtime against a fixed threshold.

Table 8: **Ablation study of SPI in NaVi.** The table compares NaVi with a fixed-confidence variant that removes SPI and adopts a fixed confidence threshold ($\tau = 0.9$), as well as the best fixed-length baseline, on three benchmarks using two dLLMs. For each method, we report both generation accuracy and runtime.

Model	Method	Metric	GSM8K	HumanEval	IFEval
LLaDA-8B-Instruct	Best Fixed-Length Baseline	Accuracy	77.9	45.1	70.7
		Runtime	10169	14616	13079
	NaVi (with confidence strategy)	Accuracy	78.8	44.5	70.7
		Runtime	13660	1262	2859
	NaVi	Accuracy	79.5	44.5	71.2
		Runtime	14686	1179	3036
LLaDA-1.5	Best Fixed-Length Baseline	Accuracy	82.7	47.0	73.9
		Runtime	874209	14601	44180
	NaVi (with confidence strategy)	Accuracy	82.5	46.3	73.6
		Runtime	13498	1426	3749
	NaVi	Accuracy	83.6	47.0	73.6
		Runtime	15851	1232	3674

D.7 More Analyses of the Repetitive Phenomenon

As briefly discussed in Sec. 2.2, expansion strategies that only add blank `[MASK]` tokens disrupt the decoding distribution of dLLMs. This disruption typically manifests as the repetitive generation phenomenon, which degrades both output quality and inference efficiency.

Through extensive qualitative analysis of the failure cases, we categorize this repetitive phenomenon into two distinct behavioral patterns: **Infinite Reasoning Loops** and **Redundant Conclusion Padding**. In the following, we provide two representative examples and analyze the mechanisms of each.

Pattern I: Infinite Reasoning Loops. In this pattern, the model loses its global awareness midway through the decoding process due to the abrupt token expansion. The sudden influx of blank masks causes an attention collapse in the deeper layers (Fig. 3a), forcing the model into a myopic state where it merely copies the most recent local context, resulting in an infinite loop. For instance, when solving a problem requiring the enumeration of two-digit prime numbers, the model successfully lists primes up to 89 but then gets trapped in a local loop, repeating the operation for the number 97 until the maximum generation length is hit:

```
[...]
- For 79:  7 + 9 = 16
- For 83:  8 + 3 = 11
- For 89:  8 + 9 = 17
- For 97:  9 + 7 = 16
- For 97:  9 + 7 = 16
- For 97:  9 + 7 = 16
..
(Repeats identical line infinitely until truncated)
```

This failure may be caused by out-of-distribution disruption from the newly added blank masks that destroy the Markovian transitions expected by the dLLM (Kim et al., 2025). The model forgets its ultimate objective (summing the valid primes in this example) and is trapped in an infinite loop, damaging generation quality and efficiency.

Pattern II: Redundant Conclusion Padding. In the second pattern, the model completes the logical reasoning and arrives at the correct final answer. However, because it is forced to fill an overly long, newly added mask buffer without a clear semantic prior indicating termination, it resorts to repeating the final concluding sentence to pad the remaining allocated length. For example consider the following mathematical derivation where the model correctly finds the answer:

> [...]
> This is because the correct product of the y-coordinates of all distinct solutions is:
>
> $$(-8) \times 7 \times (-1) = -56$$
>
> Thus, the correct final answer is:
>
> $$\boxed{-56}$$
>
> Thus, the correct final answer is:
>
> $$\boxed{-56}$$
>
> ..
> *(Repeats over 40 times)*
> Thus, the correct final answer is $\boxed{-56}$.

Unlike Pattern I, the reasoning trajectory here is complete. The repetition arises purely from a mismatch between the remaining generation budget and the model's actual response length requirement. The model is forced to hallucinate coherent but redundant text to bridge the gap to the end of the sequence. Interestingly, the model exhibits a slight variation at the very end of the sequence (*"Thus, the correct final answer is* $\boxed{-56}$."), attempting to formally close the generation as it finally reaches the sequence boundary. For this pattern, while the model eventually yields the correct answer, the intermediate reasoning trajectory is redundant and of degraded quality, which also compromises inference efficiency.

Summary and Connection to NaVi. These observations highlight the limitations of existing variable-length methods that add blank masks directly during the decoding process. By introducing Semantic Momentum Injection, NaVi effectively mitigates both failure modes as shown in Table 6. Injecting the Navigator Token's hidden state provides the newly added tokens with a semantic prior. For Pattern I, this prior maintains the global reasoning objective, preventing local attention collapse. For Pattern II, the accurate budget recalibration coupled with semantic momentum signals to the model whether to naturally conclude or continue reasoning, thereby mitigating the redundant padding.

D.8 Discussion on the Selection of Threshold Bounds $\tau_{\min} = 0.90$ and $\tau_{\max} = 0.99$

In the Schedule-Aware Dynamic Threshold Decoding mechanism (Sec. 3.3), we bound the dynamically computed threshold τ_k within the range $[\tau_{\min}, \tau_{\max}] = [0.90, 0.99]$. This specific range aligns with the established empirical properties of dLLMs (Wu et al., 2026a; Nie et al., 2025). The selection is guided by the trade-off between inference acceleration and generation quality:

For the lower bound ($\tau_{\min} = 0.90$), in iterative denoising paradigms, unmasking tokens with low prediction confidence severely degrades the final output quality, as early errors tend to compound in subsequent steps. Recent work on training-free acceleration, such as Fast-dLLM (Wu et al., 2026a), has conducted extensive empirical analyses and established that a static threshold of 0.90 serves as a safe boundary for most dLLMs.

Unmasking tokens above this threshold significantly accelerates inference while experiencing a negligible quality drop. Therefore, we adopt $\tau_{\min} = 0.90$ as our lower bound. This ensures that even under maximum generation urgency (where NaVi relaxes the threshold to catch up with the sequence budget), the model still commits tokens without compromising quality.

For the upper bound ($\tau_{\max} = 0.99$), original dLLMs such as LLaDA (Nie et al., 2025) typically employ a rigid Top-1 confidence remasking strategy. To guarantee the highest output quality, these baseline strategies prioritize committing only the highest-confidence tokens during each step. To emulate this strict quality-preservation behavior when the generation budget is abundant (low urgency), we set $\tau_{\max} = 0.99$. This nearly deterministic threshold forces the model to act conservatively to ensure the quality of the output for low decoding urgency.

By constraining the dynamic threshold within $[0.90, 0.99]$, NaVi safely interpolates between two paradigms: the efficient threshold-based acceleration (at 0.90) and the quality-maximizing sampling (at 0.99). This design avoids manual hyperparameter tuning while yielding stable performance across diverse benchmarks.

Table 9: **Effect of mixed domain fitting data on fitting performance.** The table reports the validation R^2 scores of different mixed strategies across two dLLMs, LLaDA-8B-Instruct and LLaDA-1.5.

Mixed Strategy	Metric	LLaDA-8B-Instruct	LLaDA-1.5
only math	R^2	0.7068	0.7073
only code	R^2	0.6459	0.6512
math:code = 1:2	R^2	0.6945	0.6712
math:code = 2:1	R^2	0.6509	0.6313

D.9 Reasons and Analysis for Choosing the MATH Dataset for Probe Fitting

Motivation for choosing the MATH dataset for probe fitting. We select MATH primarily for three reasons. First, unlike open-ended text generation datasets, MATH provides explicit ground-truth answers, allowing generated responses to be verified reliably. Second, compared with commonly used code benchmarks, MATH contains a relatively large number of training instances, making it easier to collect a sufficient number of correct generations across different dLLMs. Third, MATH features a well-defined difficulty gradient spanning Levels 1 to 5, which results in generated solutions with diverse response lengths, therefore increasing sample diversity and capturing the latent length awareness within dLLMs better. This diversity is particularly useful for fitting a probe that predicts the remaining generation length. Actually, our use of MATH is intended as a simple choice rather than a requirement of NaVi. More generally, the probe aims to extract a transferable length-related signal already encoded in the model's hidden representations, while the fitting dataset provides supervision for learning this mapping. A dataset with reliably verifiable outputs and sufficient response length diversity may therefore also be suitable.

Analysis with code and mixed domain fitting data. To examine the sensitivity of NaVi to the domain of the probe fitting data, we conduct an additional study using the LeetCodeDataset. Following the same construction procedure described above, we collect verified correct code responses and use them to fit linear probes for LLaDA-8B-Instruct and LLaDA-1.5. Since both models achieve relatively low accuracy on these code generation problems, only approximately 300 successful trajectories can be collected. For a fair comparison, we therefore control the total number of trajectories used in all settings to 300. In addition to probes fitted exclusively on math or code trajectories, we construct mixed-domain datasets with math-to-code

Table 10: **Effect of mixed domain fitting data on NaVi performance.** The table compares different mixed strategies across five benchmarks using LLaDA-8B-Instruct and LLaDA-1.5. Both accuracy and runtime are reported.

Model	Mixed Strategy	Metric	GSM8K	MATH-500	HumanEval	IFEval	MBPP
LLaDA-8B-Instruct	only math	Accuracy	78.9	43.2	42.7	70.7	38.4
		Runtime	13772	5971	909	2939	2894
	only code	Accuracy	78.3	43.4	41.5	69.3	38.4
		Runtime	12996	6022	873	3058	3233
	math:code = 1:2	Accuracy	78.8	43.4	42.1	70.1	37.0
		Runtime	14825	5884	924	2823	3050
	math:code = 2:1	Accuracy	79.2	43.2	42.1	69.7	37.0
		Runtime	14970	4962	1038	3742	2907
LLaDA-1.5	only math	Accuracy	82.3	44.6	44.5	73.6	36.2
		Runtime	14872	6184	1263	3826	4277
	only code	Accuracy	82.6	44.6	43.9	72.5	37.2
		Runtime	15903	6283	905	3864	4730
	math:code = 1:2	Accuracy	81.8	44.8	43.9	73.5	36.8
		Runtime	13778	6602	1133	3920	4828
	math:code = 2:1	Accuracy	82.1	44.2	45.1	73.1	36.6
		Runtime	15633	5967	1211	3529	4199

ratios of $1:2$ and $2:1$, each totaling 300 trajectories. We then fit a separate linear probe under each setting and evaluate NaVi on five benchmarks. The results are summarized in Table 9 and 10.

As shown in the tables, all settings achieve relatively high R^2 values ranging from approximately 0.63 to 0.71, indicating that the linear probe fits the model's inner representation well across different data domains. In the five benchmarks, their accuracy and runtime differ only slightly, and no setting consistently outperforms the others across all benchmarks. Therefore, code data can also serve as effective probe fitting data, as it can also provide clearly correct trajectories with diverse response lengths. Nevertheless, the math-only setting achieves the highest R^2 for both models and proves effective across various benchmarks. These results show that NaVi is robust to the fitting data domain, while MATH remains a simple and effective choice.

D.10 Sensitivity to Decoding Process and Trajectory

Table 11: **Effect of different temperatures on NaVi performance.** The table reports the accuracy of LLaDA-8B-Instruct under different sampling temperatures across five benchmarks: GSM8K, MATH-500, HumanEval, IFEval, and MBPP.

Model	Temperature	GSM8K	MATH-500	HumanEval	IFEval	MBPP
LLaDA-8B-Instruct	0.0	79.5	43.6	44.5	71.2	36.8
	0.2	79.1	43.4	43.4	71.1	36.8
	0.5	80.1	42.4	44.5	69.4	36.8
	0.8	78.8	43.8	43.9	71.2	36.4

Since different temperatures yield different token selections and hence different trajectories, we vary temperature, keeping all other settings identical, aiming to evaluate NaVi's sensitivity to the decoding process and trajectory. As shown in Table 11, accuracy remains stable across temperatures with only minor fluctuations, confirming robustness to decoding trajectories.

D.11 Sensitivity to Layer Choice

Table 12: **Effect of different layer choices on NaVi performance.** The table reports the probe R^2 score and benchmark accuracy across different layer choices using LLaDA-8B-Instruct.

Layer	R^2	GSM8K	MATH-500	HumanEval	IFEval	MBPP
22	0.68	79.5	43.6	44.5	71.2	36.8
23	0.67	78.7	43.0	45.7	71.2	38.0
21	0.68	79.3	42.8	44.5	70.1	38.2
32	0.66	78.8	42.0	42.7	70.1	37.8
10	0.56	79.0	42.6	40.9	69.3	37.8

To study NaVi's robustness to layer choice, we experiment on LLaDA-8B-Instruct. We compare the selected optimal layer 22 against its neighbors 21 and 23, an earlier layer 10, and the final layer 32, fitting a probe from each to evaluate. As shown in Table 12, Layers 21, 22, and 23 achieve stable performance with minor variations, and layer 32, which also attains a relatively high R^2, performs comparably. Layer 10, with a notably lower R^2, shows a larger gap on some benchmarks, though the overall degradation remains minor. This demonstrates both NaVi's robustness to layer choice and the effectiveness of held-out R^2 as the automatic selection criterion.

D.12 Exploration to Multi-layer Aggregation

To explore the potential of multi-layer representations, we combine the best-performing layers by averaging or concatenating their hidden states. As shown in Table 13, across all layer combinations, accuracy stays close to the single-layer baseline, improving on some benchmarks while degrading on others, with no combination consistently better. Realizing the potential of multi-layer representations would therefore require more carefully designed fusion strategies, and our single-layer design remains an effective and simpler choice.

D.13 Analysis of the Role of Semantic Momentum Injection

To clarify the role of semantic momentum injection, we add two complementary analyses: a random embedding token injection and a token distribution diagnostic based on KL divergence. Overall, these results show that the gains arise from the injection of the Navigator Token's hidden state rather than from an arbitrary additive bias, and that semantic momentum mitigates the distribution shift caused by sudden length expansion.

Random embedding token injection. We replace the Navigator Token hidden state with the embedding of a random vocabulary token. As shown in Table 14, a random bias produces more repetitive loops and lower accuracy than semantic momentum on nearly all benchmarks, and is even inferior to the baseline (without momentum) in several cases. This indicates that the improvement depends on the semantic information encoded in the Navigator state rather than merely on injecting a random vector.

Diagnostic of token distribution. We sample ten prompts from each of the five benchmarks and average the results. For each trigger position $k \in \{32, 64, 128, 256\}$, we first run the model with an initial budget

Table 13: **Effect of multi-layer aggregation strategies on NaVi performance.** The table compares different layer aggregation schemes and aggregation methods based on probe R^2 scores and benchmark accuracy.

Multi-layer aggregation	Method	R^2	GSM8K	MATH-500	HumanEval	IFEval	MBPP
21+22+23	Average	0.68	79.4	43.6	42.1	71.2	36.8
	Concatenate	0.70	79.8	42.2	45.1	70.1	37.2
21+22	Average	0.68	79.1	43.8	44.5	69.7	36.6
	Concatenate	0.70	78.3	44.0	43.9	71.2	37.6
22+23	Average	0.68	79.8	44.0	45.1	69.7	37.6
	Concatenate	0.69	79.0	46.2	44.5	68.7	36.2

Table 14: **Effect of semantic momentum and random bias injection.** The table compares the impact of different Navigator token injection strategies on repetitive loops and generation accuracy across five benchmarks.

Method	Metric	GSM8K	MATH-500	HumanEval	MBPP	IFEval
Baseline (without momentum)	Repetitive Loops	4	30	1	4	49
	Acc	82.3	45.6	47.0	36.2	72.9
with momentum	Repetitive Loops	0	9	0	1	1
	Acc	83.6	46.2	47.0	36.6	73.6
with a random bias	Repetitive Loops	8	15	0	6	18
	Acc	81.0	45.0	47.0	35.8	72.9

of $k + 32$. At k verified tokens, we invoke NaVi's probe to obtain a predicted total length L_{pred} and record the distributions over all undecoded positions, with and without semantic momentum. We then rerun from the beginning with the budget set directly to L_{pred} and take its distributions at the same position k as the reference. This run allocates the full budget upfront and involves no expansion. Finally, we compute the KL divergence of the two expansion distributions against this reference. All three runs share the same verified prefix and total generation length, differing only in whether that length is reached through abrupt expansion, so the measured divergence isolates the shift introduced by expansion itself. Results in Table 15 show that semantic momentum consistently reduces the KL divergence across both models, providing direct evidence that it mitigates the distribution shift caused by intermediate length expansion.

E Details on Experimental Settings

E.1 Details of Fitting Data Construction

Table 16 outlines the specific generation budgets allocated for the four models evaluated in our study. For LLaDA-8B-Instruct (Nie et al., 2025), LLaDA-1.5 (Zhu et al., 2026), and Dream-v0-7B-Instruct (Ye et al., 2025), which adopt a full-sequence generation strategy, we set a fixed generation budget of 4096 tokens. This length is sufficient to cover the reasoning trajectories of most MATH problems. In contrast, LLaDA2.0-mini (Bie et al., 2025) uses a block-wise autoregressive decoding mechanism that restricts bi-directional attention to the decoding block. Therefore, we scale its budget to the officially supported maximum context length of 32768 tokens to obtain its ground truth. Specifically, the data are obtained from trained dLLMs rather than from a simulation of uniformly random unmasking. For each evaluated model, such as LLaDA-8B-Instruct and LLaDA-1.5, we run inference on 2000 problems sampled from the MATH training split. Then

Table 15: **KL divergence analysis of token distributions under length expansion.** The table reports the KL divergence between the expanded generation distribution and the reference distribution at different trigger positions ($k = 32, 64, 128, 256$), with and without semantic momentum. Lower KL divergence indicates less distribution shift caused by intermediate length expansion.

Model	Method	32	64	128	256
LLaDA-8B-Instruct	w/o Momentum	0.70	0.74	7.59	5.03
	with Momentum	0.47	0.42	3.64	1.73
LLaDA-1.5	w/o Momentum	0.73	2.52	5.03	3.77
	with Momentum	0.38	0.82	1.40	1.89

Table 16: **Specific generation budgets for fitting data construction on four models.**

LLaDA-8B-Instruct	LLaDA-1.5	Dream-v0-7B-Instruct	LLaDA2.0-mini
4096	4096	4096	32768

we use the model's standard block-wise semi-autoregressive decoding procedure with a greedy strategy and allocate a maximum generation length of 4096 tokens to accommodate responses with diverse lengths. Out of these 2000 generated outputs, we retain only the correctly answered responses and randomly sample 700 of them, pairing each response with its corresponding prompt to form the successful generation trajectories.

E.2 Details of Linear Ridge Regression Probe Fitting

In Sec. 3.2, we outline the formulation of our offline linear regression probe $\mathcal{R}_\phi$. To provide a deeper understanding of our methodology, this section details the specific data preprocessing, hyperparameter search methods, and reasons for our choice of the ridge regression module.

Reasons for Choosing Linear Ridge Regression. While more complex non-linear models (e.g., Multi-Layer Perceptrons) could potentially yield lower fitting errors, we select a simple Linear Ridge Regression model. This choice is driven by two reasons. First, achieving high R^2 scores using a linear probe strongly proves our hypothesis: the latent length awareness is linearly encoded within the latent representation of the Navigator Token. Second, given the relatively small fitting dataset (700 trajectories), introducing an L_2 regularization penalty (Ridge) effectively prevents overfitting and ensures that the learned length estimator generalizes robustly to different benchmarks.

Data Construction and Preprocessing. As mentioned, we randomly sample 700 successful generation trajectories from the MATH dataset. For each trajectory, we extract intermediate states at 5 predefined truncation ratios ($\mathcal{R} = \{0.0, 0.2, 0.4, 0.6, 0.8\}$), yielding a total dataset $\mathcal{D}$ of 3500 samples (hidden state and target length pairs). Before fitting the regression model, we perform the following preprocessing steps: First, we process data splitting. The 3500 samples are randomly shuffled and partitioned into a training set and a validation set. Then, for hidden states, we make feature standardization. The raw hidden state vectors $\mathbf{h}_{\text{nav}}^{(\ell)} \in \mathbb{R}^{d_{\text{model}}}$ often exhibit varying magnitudes across different dimensions. To ensure that the L_2 regularization penalty applies uniformly across all features, we applied standard scaling (z-score normalization) to the hidden states using the mean and standard deviation computed exclusively from the fitting set.

Table 17: **Optimal layers in different models.**

	LLaDA-8B-Instruct	LLaDA-1.5	Dream-v0-7B-Instruct	LLaDA2.0-mini
Total Layers	32	32	28	20
Optimal Layer	23	22	19	12

Grid Search for Hyperparameters. The optimal transformer layer ℓ^* and the L_2 regularization coefficient λ are jointly determined by a grid search aimed at maximizing the validation R^2 score, which is automatic without manual tuning. The search operations are defined as follows:

- **Optimal Layer ℓ^*:** We sweep across all transformer layers of the dLLM, from $\ell = 1$ to $\ell = L_{\text{last}}$ and select the one with the highest R^2 score as the optimal layer. As a result, we consistently observe that representations from the middle-to-deep layers yield the most accurate length estimations (Table 17), likely because deeper layers encode more global planning semantics compared to shallow layers.
- **Regularization Coefficient λ:** We perform a grid search over λ in a predefined range. The parameter ϕ that yielded the highest R^2 on the validation split is frozen and utilized for all online inference experiments.

Fitting Computational Efficiency. Because the fitting process relies on a lightweight linear model and requires only 3500 data points, the computational overhead is minor. Generating the hidden states via forward passes and fitting the Ridge Regression takes only a few minutes on a single GPU. Once fitted, this probe can be directly applied to any benchmark evaluation or other scenarios without requiring further training or fine-tuning.

E.3 Decoding and Sampling Settings

We select LLaDA-8B-Instruct (Nie et al., 2025), LLaDA-1.5 (Zhu et al., 2026), Dream-v0-7B-Instruct (Ye et al., 2025), and LLaDA2.0-mini (Bie et al., 2025) as our evaluation models. For all models, the generation procedure follows a semi-autoregressive remasking strategy (Nie et al., 2025), configured with a default block size of $B = 32$. Crucially, the attention mechanisms differ across these models. LLaDA-8B-Instruct, LLaDA-1.5, and Dream-v0-7B-Instruct employ global bidirectional attention, meaning tokens within a block can attend to the entire sequence, including future mask tokens. In contrast, LLaDA2.0-mini adopts a localized, causal-like attention mechanism where tokens within a block can only attend to the current and preceding blocks. This difference renders existing variable-length methods such as DAEDAL and $\rho - \texttt{EOS}$ incompatible with LLaDA-2.0-mini, as they rely on global context for length estimation. However, NaVi relies on the Navigator Token and seamlessly adapts to LLaDA2.0-mini. Consequently, for the first three models, we compare our method against fixed-length baselines, DAEDAL and $\rho - \texttt{EOS}$; for LLaDA-2.0mini, we directly compare against its baseline.

E.4 Implementation Details

All experiments are performed on NVIDIA RTX PRO 6000 (Blackwell Server Edition) GPUs. To ensure reproducibility and fair comparisons, we utilize greedy decoding across all setups without any random sampling strategies. For the baseline models, including LLaDA-8B-Instruct, LLaDA-1.5, and Dream-v0-7B-Instruct, performance is highly sensitive to the manually predefined generation length, which varies per task (Li et al., 2026). Therefore, we evaluate and report their performance across six fixed-length configurations, ranging from 64 to 2048 tokens. For LLaDA-2.0-mini, which natively supports variable-length generation via a block-diffusion paradigm equipped with an early-stop mechanism, we directly integrate

NaVi into its generative pipeline as detailed in Sec. 3 and Appendix E.7. Throughout all our experiments, we maintain a unified set of hyperparameters across all models and benchmarks, demonstrating the generalization and adaptability of NaVi in diverse evaluation scenarios.

E.5 Details of Baseline Methods

In Sec. 4, we select two recent variable-length generation methods, DAEDAL (Li et al., 2026) and $\rho - \texttt{EOS}$ (Yang et al., 2026), as our baselines. Both methods are evaluated using their official code repository with the default configurations reported in their original papers. Specifically, for the key hyperparameters, DAEDAL uses $L_{\text{init}} = 64$, $\tau_{\text{high}} = 0.9$, $\tau_{\text{low}} = 0.1$, $\tau_{\text{eos}} = 0.5$, and $\tau_{\text{expand}} = 0.9$. $\rho - \texttt{EOS}$ uses $L_{\text{init}} = 64$, $\tau_{\text{high}} = 0.9$, and $\tau_{\text{low}} = 0.1$. And for Sym, we set $\rho_{\text{low}} = 0.4$ and $\rho_{\text{high}} = 0.6$; for Asym, we set $\rho_{\text{low}} = 0.4$ and $\rho_{\text{high}} = 0.8$. All of them are consistent with the settings used in their original evaluations. To ensure a fair comparison, all methods are evaluated on NVIDIA RTX PRO 6000 (Blackwell Server Edition) GPUs, maintaining hardware consistency across both the baselines and our method.

E.6 Details on Benchmarks

HumanEval (Chen et al., 2021): A code generation benchmark consisting of 164 hand-written Python programming problems, designed to measure the functional correctness of synthesized code through unit tests.

MBPP (Austin et al., 2021): The Mostly Basic Python Problems (MBPP) dataset contains a test set of 500 entry-level Python programming tasks, emphasizing basic concepts and standard library usage.

IFEval (Zhou et al., 2023): An instruction-following evaluation benchmark that focuses on objective constraints rather than subjective quality, utilizing a set of verifiable prompts.

GSM8K (Cobbe et al., 2021): A grade-school mathematics reasoning benchmark that separates 1,319 problems as a held-out test set for evaluation.

MATH-500 (Lightman et al., 2024): A curated set of 500 mathematical problems covering multiple difficulty levels, designed to provide a hierarchical test of reasoning ability.

AIME2024 (AI-MO, 2024): Composed of 30 problems from the 2024 American Invitational Mathematics Examination (AIME), this benchmark presents a difficult reasoning dataset, often calling for extensive chains of multi-step reasoning.

AIME2025 (OpenCompass, 2025): Derived from the 2025 AIME, consisting of 30 problems, and demands the same kind of intricate, multi-step reasoning as AIME2024.

E.7 Implementation Details of Semantic Momentum Injection in Architectures like LLaDA2.0

As briefly introduced in Section 3.3, the Semantic Momentum Injection mechanism requires the currently decoding tokens to access the semantic prior of the remaining sequence. In architectures with global bidirectional attention (LLaDA-8B-Instruct (Nie et al., 2025), LLaDA-1.5 (Zhu et al., 2026) and Dream-v0-7B-Instruct (Ye et al., 2025)), this is naturally achieved because the tokens within the decoding block $[k, k + B)$ can freely attend to the future unverified mask tokens within $j \geq k + B$, where the momentum is injected.

However, architectures like LLaDA-2.0-mini adopt a causal block-diffusion attention mask. This localized attention design strictly dictates that tokens within a specific block can only attend to themselves and the preceding blocks, rendering future tokens completely invisible to the current block. Consequently, if we

simply inject the semantic momentum into the future block ($j \geq k + B$), the current block $[k, k + B)$ will fail to absorb this semantic prior.

To overcome this architectural restriction while keeping the intervention as minimal as possible, we propose a *Single-Token Lookahead Strategy*. The implementation seamlessly adapts to the causal block structure through the following steps:

Window Expansion. During the forward pass for the current decoding block, we expand the input sequence window by exactly *one* token into the future. The block size becomes $B+1$, covering the interval $[k, k+B+1)$.

Targeted Injection. We inject the scaled semantic momentum vector $\alpha_k \mathbf{h}_{\text{nav}}^{(\ell^*,k)}$ into the initial input embedding of this specific $(B + 1)$-th token.

Prior Absorption. The model performs its forward pass over this expanded $B + 1$ sequence. Because this extra lookahead token is included in the same decoding block, the preceding B tokens within the block can attend to it. This allows the decoding block to effectively absorb the semantic prior of the remaining sequence.

This strategy maintains functional equivalence with the original mechanism used for global attention models. By introducing an extra lookahead token in the decoding block during the forward pass, NaVi bridges the architectural gap. It guarantees that newly unmasked tokens retain their expectation of the future generation length, fully preserving the stability of the decoding distribution on such architectures as LLaDA 2.0.

F CASE STUDY

Case Study of NaVi on LLaDA-8B-Instruct

Prompt.

Let a be a positive real number such that all the roots of $x^3 + ax^2 + ax + 1 = 0$ are real. Find the smallest possible value of a.

Initialization

Prompt Length: **59** Initial Predicted Length: **313**

Generation Trace (Block-wise Decoding).

[Block00] Given the polynomial equation $x^3 + ax^2 + ax + 1 = 0$, we need to find the smallest possible positive value of a such that all roots are real.

SPI 1.000 | τ **0.9818**

At the early stage, SPI remains 1.

[Block01] First, let's test some simple values to see if we can find an obvious root.

SPI 1.087 | τ **0.9783**

As the derivation begins, SPI increases slightly, indicating growing decoding pressure, and the threshold is also relaxed.

[Block02] Notice that $x = -1$ is always a root of the equation $x^3 + ax^2 + ax + 1 = 0$,

SPI 1.223 | τ 0.9727
SPI continues to rise, and the threshold drops further.

[Block03] since $(-1)^3 + a(-1)^2 + a(-1) + 1 = -1 + a - a + 1 = 0$.

SPI 1.397 | τ 0.9656
The threshold keeps decreasing as SPI grows; higher urgency calls for more permissive acceptance.

[Block04] Therefore, we can factor out $(x + 1)$ from the polynomial to get a quadratic factor.

SPI 1.630 | τ 0.9560
With SPI rising steadily, the threshold is adjusted downward because the model predicts generating far more than the existing remaining length.

[Block05] Performing polynomial division, we get: $(x + 1)(x^2 + (a - 1)x + 1) = 0$.

SPI 1.956 | τ 0.9427
The continued SPI increase indicates that the remaining budget is becoming tighter.

[Block06] For all roots of the original cubic to be real, the quadratic factor must have real roots.

SPI 2.445 | τ 0.9227
SPI enters a more elevated regime, so the threshold is reduced more aggressively, matching the model's higher decoding urgency and stronger demand for continuation.

[Block07] This means its discriminant must be non-negative. We calculate $\Delta = (a - 1)^2 - 4 \cdot 1 \cdot 1$.

SPI 3.260 | τ 0.9000
As SPI reaches the upper range, the threshold reaches a minimum level.

[Block08] Setting the discriminant greater than or equal to zero, we get $(a - 1)^2 - 4 \geq 0$.

SPI 4.891 | τ 0.9000
The threshold stabilizes at its lower bound while SPI remains high, showing the decoding is still urgent.

[Block09] Expanding this inequality, it reduces to $a^2 - 2a + 1 - 4 \geq 0 \implies a^2 - 2a - 3 \geq 0$.

SPI 1.000 | τ 0.9818
Length Recalibration is triggered.

Recalibration Triggered
NaVi updates remaining length according to the length prediction of the linear probe.

SPI 1.000 | τ 0.9818

Expansion
Budget expanded with the injection of semantic momentum.

EXPAND → +120 tokens

[Block10] This quadratic inequality factors as $(a + 1)(a - 3) \geq 0$.

SPI 1.267 | τ 0.9709
After recalibration, SPI drops back to a moderate level.

[Block11] Since a must be a positive real number, the only way this holds is if $a \geq 3$.

SPI 1.727 | τ 0.9521
SPI remains controlled and the threshold stays moderately high

[Block12] Thus, the smallest positive value that satisfies this is $\boxed{3}$.

SPI 2.714 | τ **0.9117**

As the answer becomes deterministic, SPI remains stable, and the threshold is lowered again to safely accept more tokens.

[Block13] `<|eot_id|><|endoftext|>`

SPI 16.333 | τ **0.9000**

Sequence termination, and the threshold is safe to be the lowest.

Figure 7: Case Study of NaVi on LLaDA-8B-Instruct