UNDERSTANDING EVALUATION INCONSISTENCY IN DIFFUSION LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite the capability of parallel decoding, diffusion large language models (dLLMs) require many denoising steps to maintain generation quality, motivating recent research on efficient decoding strategies. However, existing studies have reported inconsistent benchmark results even in seemingly identical evaluation settings, undermining the reliability of conclusions about dLLM decoding. To understand this issue, we perform a comprehensive empirical study of existing dLLM decoding methods across diverse evaluation protocols. Our analysis reveals that the performance gains of decoding methods are highly prompt-dependent, resulting in inconsistent rankings of methods across prompt templates. We further attribute this evaluation inconsistency to the high sensitivity of parallel decoding methods to prompt design, where even minor prompt variations can significantly alter generation behavior. Notably, we find that a well-chosen prompt template can yield markedly greater accuracy gains than simply increasing the decoding budget. Beyond prompt templates, our experiments indicate that other commonly overlooked evaluation settings can also affect the assessment of decoding methods. Based on these findings, we provide practical guidelines for reliably evaluating decoding methods for diffusion large language models.

1 INTRODUCTION

Diffusion large language models (dLLMs) have emerged as a promising alternative to autoregressive large language models (Nie et al., 2025; Ye et al., 2025; Song et al., 2025). By adopting bidirectional attention, dLLMs enable any-order parallel decoding, overcoming the sequential generation bottleneck of autoregressive (AR) language models. Despite the capability of parallel decoding, dLLMs still require a large number of denoising steps to maintain generation quality, considerably limiting their inference efficiency. Recently, many studies have sought to accelerate dLLMs by leveraging key-value caches to reduce per-step computational cost, or by reducing the number of denoising steps (Wu et al., 2026a; Lu et al., 2026; Ma et al., 2025). These methods have been reported to achieve substantial inference speedups without degrading generation quality, and in some cases even improve downstream performance.

However, benchmark scores reported in existing studies exhibit marked inconsistency even in seemingly identical evaluation settings, undermining the reliability of the evaluation of decoding methods (Nie et al., 2025; Chen et al., 2026b; Liu et al., 2026). For example, the accuracy reported on GSM8K for LLaDA-8B-Instruct exhibits a discrepancy of up to 9 percentage points across different studies, despite using the same decoding method and generation configuration (Hu et al., 2025; Wu et al., 2026b). To investigate this inconsistency in evaluation, we performed a rigorous evaluation of current decoding methods across various benchmarks and models. Our evaluation reveals that the performance of decoding methods is highly prompt-dependent. This highlights a key pitfall in dLLM evaluation: the performance ranking of decoding methods can vary significantly across different prompt templates. Through comprehensive evaluation across common prompt templates, we find that aggressive acceleration often incurs task accuracy degradation.

In this work, we attribute this inconsistency in evaluation to the high sensitivity of decoding methods to minor prompt variations. As illustrated in Figure 1, minor variations in the prompt template can lead to non-trivial differences in downstream task performance. This effect is markedly more pronounced for parallel decoding methods. Specifically, parallel decoding is more prone to ampli-

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

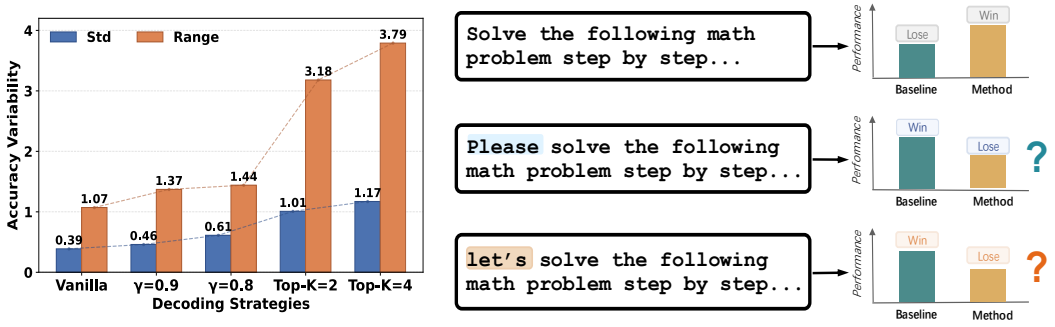


Figure 1: **Left**: Accuracy variability of single-token decoding (*Vanilla*) and parallel decoding methods (*Threshold-based decoding and Top-K*) across near-identical prompt templates. We evaluate *Threshold-based decoding* with $\gamma \in \{0.8, 0.9\}$ and *Top-K decoding* with $k \in \{2, 4\}$. Std and Range denote the standard deviation and range of accuracy across prompt templates. **Right**: Minor variations in prompt templates can produce noticeable differences in performance, which may lead to misleading conclusions in method comparison.

fying subtle prompt variations into inconsistent outputs, leading to performance fluctuations across prompts. Moreover, our analysis shows that prompt templates can play a dominant role in determining evaluation results. A well-chosen prompt template can yield markedly greater accuracy gains than simply increasing the decoding budget.

Beyond prompt templates, our experiments indicate that commonly overlooked evaluation settings can also confound assessment of dLLM decoding methods. Specifically, increasing the generation length budget does not consistently yield performance gains and can even degrade performance under certain prompt templates. In addition, few-shot evaluation settings generally improve accuracy. However, decoding methods exhibit different preferences for the number of few-shot examples, making performance comparisons sensitive to the chosen few-shot setting. Moreover, we find that evaluating decoding methods across different hardware platforms can yield inconsistent results, undermining fair comparisons. Based on these findings, we highlight several practical considerations for reliable evaluation of decoding methods in dLLMs, such as multi-prompt evaluations and detailed reporting of evaluation settings.

We summarize our main contributions as follows:

- We reveal that the performance gains of decoding methods are highly prompt-dependent, resulting in inconsistent rankings of methods across prompt templates.
- We attribute this evaluation inconsistency to the high sensitivity of parallel decoding to prompt design, where even minor prompt variations can alter generation behavior.
- We provide a systematic analysis of dLLM decoding methods across various benchmarks and evaluation settings. Our analysis shows that, beyond prompt templates, commonly overlooked evaluation settings can affect evaluation results.

2 Preliminary

2.1 Diffusion Large Language Models

Diffusion large language models formulate text generation as a forward process and a reverse process. The forward process gradually adds noise to clean text by masking tokens, while the reverse process recovers masked sequences by iteratively predicting masked tokens.

Forward Process Let x_0 denote a clean sequence consisting of N tokens. We denote by x_t the partially masked sequence, where each token is independently masked with probability $t \in [0, 1]$. The forward process progressively masks tokens independently in x_0, resulting in a corrupted x_t. At each time t, the forward process is defined by the transition probability:

$$q_{t|0}(x_t \mid x_0; t) = \prod_{i=1}^{N} q_{t|0}(x_t^i \mid x_0^i), \quad \text{where} \quad q_{t|0}(x_t^i \mid x_0^i; t) = \begin{cases} t, & \text{if } x_t^i = [\texttt{MASK}], \\ 1-t, & \text{if } x_t^i = x_0^i, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

where $q_{t|0}(x_t^i \mid x_0^i; t)$ denotes the transition probability for the i-th token, where x_t^i is independently replaced by [MASK] with probability t. The sequence-level transition probability $q_{t|0}(x_t \mid x_0; t)$ can be factorized as the product of token-level transition probabilities at each position.

Reverse Process The reverse process seeks to recover masked sequences by predicting masked tokens. Specifically, the mask predictor is parameterized by a neural network p_θ that takes x_t as input and predicts all masked tokens simultaneously. Formally, we define the reverse process as follows:

$$p_\theta(x_0 \mid x_t; t) = \prod_{i \in \mathcal{M}_t} p_\theta\big(x_0^i \mid x_t; t\big)\,, \tag{2}$$

where $\mathcal{M}_t := \{i \mid x_t^i = [\texttt{MASK}]\}$ represents masked token set. The core of dLLMs is to learn a mask predictor that recovers masked tokens from corrupted text. Consequently, given the noisy sequence, the model is trained to denoise x_t by maximizing a variational lower bound on the log-likelihood:

$$\log p_\theta(x_0) \geq \mathbb{E}_{t,x_t}\left[\frac{1}{t}\sum_{i=1}^{N} \mathbf{1}[x_t^i = \texttt{MASK}] \log p_\theta(x_0^i \mid x_t)\right], \tag{3}$$

where t is sampled from the uniform distribution $t \sim U(0,1)$, and the x_t is the corrupted text sampled from the distribution $q_{t|0}(x_t \mid x_0; t)$.

Generation At generation time, given a prompt q with a fully masked response, the generation process progressively denoises the masked response into plain text over T steps. To maintain generation quality, the mask predictor decodes only a subset of the masked tokens at each step, rather than decoding all masked tokens simultaneously. During generation, the model progressively decodes the masked response through iterative refinement over multiple steps. As a result, the masked response is gradually unmasked through iterative refinement, resulting in a coherent and fluent output.

3 Evaluation Inconsistency in Diffusion Large Language Models

3.1 Problem Setup

Accelerating decoding in dLLMs typically degrades generation quality. As a result, recent work has sought to develop decoding algorithms that improve inference efficiency without sacrificing generation quality. To assess the performance of decoding methods, researchers commonly evaluate them on benchmark datasets, where higher accuracy indicates better generation quality.

Formally, we denote by $\mathcal{A} = \{A_1, \ldots, A_n\}$ a set of decoding algorithms for dLLMs and $\mathcal{D}$ a benchmark dataset. Given a decoding algorithm A_i, we query the model for each benchmark question q with a prompt template p, and evaluate the response generated to obtain the performance score $s(A_i, p; \mathcal{D})$. The performance scores of all decoding algorithms induce a ranking over $\mathcal{A}$:

$$s(A_{(1)}, p; \mathcal{D}) \geq s(A_{(2)}, p; \mathcal{D}) \geq \cdots \geq s(A_{(n)}, p; \mathcal{D}), \tag{4}$$

where $A_{(r)}$ denotes the decoding algorithm with the r-th highest score. A higher score often indicates that the decoding algorithm produces better performance on the benchmark dataset. This evaluation typically assumes that the relative ranking of decoding algorithms remains unchanged across different prompt templates.

Definition 1 (*Evaluation Inconsistency*). Let $\mathcal{P}$ denote the set of prompt templates. For each prompt template $p \in \mathcal{P}$, let $s(A_i, p)$ denote the performance score of method A_i under prompt p. The evaluation procedure exhibits *evaluation inconsistency* if there exist $p, p' \in \mathcal{P}$ and $A_i, A_j \in \mathcal{A}$ such that

$$s(A_i, p) > s(A_j, p) \quad \text{and} \quad s(A_i, p') < s(A_j, p').$$

While performance scores may vary across prompt templates, a reliable evaluation is expected to produce a stable ranking of methods on the same benchmark. Evaluation inconsistency implies that single-prompt evaluations yield high-variance estimates of method performance, leading to potentially biased performance assessments.

Table 1: The accuracy of vanilla and current decoding methods on GSM8K and MATH500 under LLaDA-8B-Instruct, across various prompt templates. *Vanilla* baseline denotes the single-token decoding method. Each numerical subscript indicates the improvement or degradation compared to the vanilla decoding method.

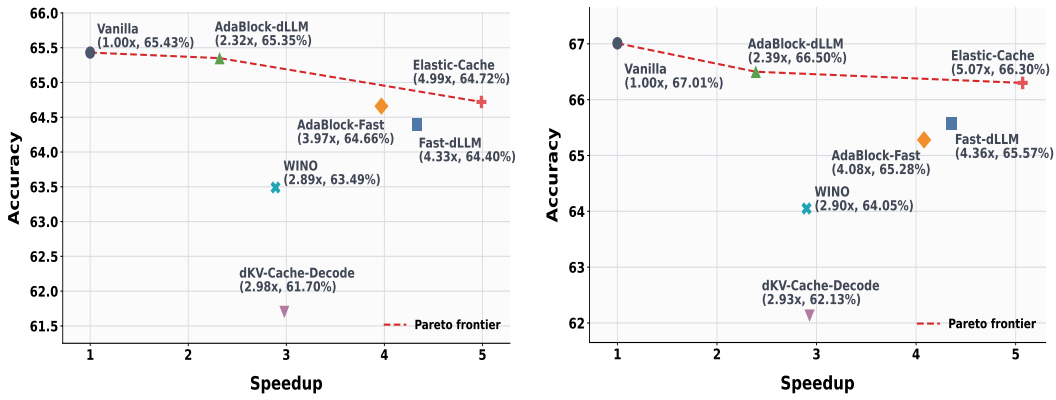
Method	GSM8K						MATH500		Avg.
	Template 1	Template 2	Template 3	Template 4	Template 5	Template 6	Template 7	Template 8	
Vanilla	**77.33**	74.60	71.72	**76.35**	74.98	76.88	**37.60**	34.00	65.43
Parallel decoding methods									
Fast-dLLM	$76.35_{-0.98}$	$73.16_{-1.44}$	$70.51_{-1.21}$	$75.74_{-0.61}$	$75.82_{+0.84}$	$74.60_{-2.28}$	$33.80_{-3.80}$	$35.20_{+1.20}$	$64.40_{-1.03}$
AdaBlock-dLLM	$76.88_{-0.45}$	$\mathbf{75.06}_{+0.46}$	$71.95_{+0.23}$	$75.74_{-0.61}$	$74.83_{-0.15}$	$\mathbf{76.95}_{+0.07}$	$37.00_{-0.60}$	$34.40_{+0.40}$	$65.35_{-0.08}$
AdaBlock-Fast	$76.65_{-0.68}$	$73.92_{-0.68}$	$71.27_{-0.45}$	$75.66_{-0.69}$	$75.21_{+0.23}$	$74.15_{-2.73}$	$34.20_{-3.40}$	$\mathbf{36.20}_{+2.20}$	$64.66_{-0.77}$
dKV-Cache-Decode	$72.93_{-4.40}$	$69.45_{-5.15}$	$66.94_{-4.78}$	$72.93_{-3.42}$	$72.63_{-2.35}$	$73.92_{-2.96}$	$31.80_{-5.80}$	$33.00_{-1.00}$	$61.70_{-3.73}$
Elastic-Cache	$77.10_{-0.23}$	$73.84_{-0.76}$	$\mathbf{72.18}_{+0.46}$	$75.97_{-0.38}$	$\mathbf{76.88}_{+1.90}$	$76.35_{-0.53}$	$33.60_{-4.00}$	$31.80_{-2.20}$	$64.72_{-0.71}$
WINO	$74.68_{-2.65}$	$72.48_{-2.12}$	$70.74_{-0.98}$	$74.45_{-1.90}$	$72.78_{-2.20}$	$75.21_{-1.67}$	$33.40_{-4.20}$	$34.20_{+0.20}$	$63.49_{-1.94}$

3.2 Evaluation Inconsistency Undermines Performance Metric Reliability

Experimental setup We evaluate recent decoding methods for dLLMs, comparing Low-confidence (Nie et al., 2025) as a vanilla baseline against Fast-dLLM (Wu et al., 2026a), AdaBlock-dLLM (Lu et al., 2026), AdaBlock-Fast (Lu et al., 2026), dKV-Cache-Decode (Ma et al., 2025), Elastic-Cache (Nguyen-Tri et al., 2026), and WINO (Hong et al., 2026), on commonly used math and code benchmarks using LLaDA (Zhu et al., 2025; Nie et al., 2025) and Dream (Ye et al., 2025) models. In our experiments, we set the generation length to 128 and adopt the semi-autoregressive decoding strategy. *To eliminate the effects of hardware and system configuration on evaluation results, our experiments are conducted on a single machine with a single GPU and a batch size of 1.* For each decoding method, we use the best-performing hyperparameter settings reported in the corresponding paper. Further details on the hyperparameter configurations are provided in Appendix B.1. We evaluate decoding methods across prompts commonly used in existing evaluation practice, as presented in Appendix D.1. In our experiments, *Vanilla* denotes the low-confidence decoding baseline, which decodes a single masked token with the highest confidence at each step. Parallel decoding methods (e.g., Fast-dLLM) decode multiple masked tokens at each step according to a designed parallel decoding strategy.

dLLM decoding methods exhibit pervasive evaluation inconsistency Table 1 and Table 2 report the accuracy of various decoding methods on math datasets across various prompt templates under LLaDA-8B-Instruct (Nie et al., 2025) and LLaDA-1.5 (Zhu et al., 2025), respectively. The results reveal that the performance of decoding methods is highly sensitive to prompt template choices, exhibiting substantial instability across prompt templates. For instance, in Table 1, Elastic-Cache achieves better performance than Vanilla under Template 3, but Vanilla substantially outperforms Elastic-Cache under Template 2. We also report evaluation results for the Dream model in Appendix C.4, where we observe a similar pattern. Figure 7a and Figure 7b in Appendix C.1 show the pairwise Kendall's tau correlations (see definition in Appendix B.3) among method rankings obtained under different prompt templates for LLaDA-8B-Instruct and LLaDA-1.5, respectively. Higher values indicate more consistent relative ranking of decoding methods, whereas lower values reflect greater disagreement in method ranking across prompt templates. The average pairwise Kendall's tau across prompt templates is only 0.539 for LLaDA-8B-Instruct and 0.477 for LLaDA-1.5, indicating pervasive evaluation inconsistency across dLLM decoding methods. In Appendix C.2, we report bootstrap confidence intervals, confirming that the observed evaluation inconsistency is not an artifact of a specific test sample. **These findings suggest that evaluating decoding methods under a single prompt template may undermine the reliability of evaluations.**

Performance gains of decoding methods are highly prompt-dependent A fundamental challenge in dLLM decoding methods is the dilemma between inference efficiency and generation quality. To address this, recent studies have explored parallel decoding methods designed to accelerate generation without sacrificing generation quality. Table 1 and Table 2 compare the accuracy of the



(a) LLaDA-8B-Instruct (b) LLaDA-1.5

Figure 2: The accuracy–speedup trade-offs of different decoding methods for (a) LLaDA-8B-Instruct and (b) LLaDA-1.5. The x-axis shows the average decoding speedup relative to Vanilla, and the y-axis shows the average accuracy across prompt templates.

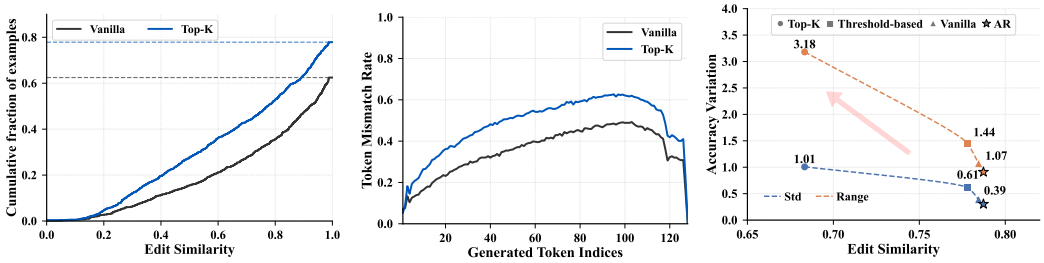
vanilla baseline and current parallel decoding methods across various prompt templates. Although some decoding methods occasionally produce marginal improvements over the vanilla baseline on certain prompt templates, degradation is widespread across the majority of prompt templates. The evaluation results on code benchmarks in Appendix C.3 also show a similar trend. These results show that the performance gains of parallel decoding methods are highly prompt-dependent and that aggressively accelerating generation can incur accuracy degradation on downstream tasks. More detailed results are provided in Appendix C.7. We further present the accuracy–speedup trade-off of decoding methods in Figure 2a and Figure 2b. The x-axis shows the average decoding speedup relative to Vanilla, computed from the measured end-to-end decoding times (Wu et al., 2026a), while the y-axis shows the average accuracy across prompt templates. We find that Vanilla, AdaBlock-dLLM, and Elastic-Cache form the Pareto frontier across both models, achieving the superior trade-offs between accuracy and decoding speed. **Our findings suggest that evaluating dLLM decoding methods relying on a single prompt template can undermine the reliability of evaluation.**

4 Performance Dynamics of dLLM Decoding Methods

4.1 Sensitivity of Decoding Methods to Minor Prompt Variations

Experimental setup Motivated by the high sensitivity of decoding methods to prompt templates, we systematically examine how minor prompt variations affect evaluation metrics. Specifically, we construct near-identical prompt templates differing only in individual words or characters (Appendix D.2). We evaluate three decoding strategies on GSM8K under LLaDA-8B-Instruct: single-token decoding (Vanilla), Top-K decoding, and threshold-based confidence decoding. We provide implementation details in Appendix B.2. Vanilla decoding employs low-confidence remasking, committing only the highest-confidence masked token at each step, and generally achieves superior performance. Parallel decoding accelerates generation by decoding multiple tokens per step, reducing denoising steps, but can degrade task performance. The Top-K decoding method selects the k masked tokens with the highest confidence scores, while the threshold-based confidence decoding method unmasks all tokens whose confidence exceeds a predefined threshold.

Parallel decoding transfers subtle prompt variations into inconsistent outputs We analyze how minor prompt variations affect dLLM generation behavior by measuring output similarity under pairs of near-identical prompts, as presented in Example 1. In our experiment, we use Top-K (k=2) decoding as a parallel decoding baseline, compared with single-token decoding baselines. We plot the cumulative distribution function (CDF) of edit similarities between generated outputs under the two prompt templates in Figure 3a. As defined in Appendix B.4, we employ edit similarity to quantify token-level agreement between generated sequences, where higher scores reflect greater output consistency across prompt templates. The higher CDF curve for Top-K decoding reflects a higher proportion of low-similarity outputs across prompts, suggesting that Top-K decoding increases token-level divergence under subtle prompt variations. We further examine the position-wise token mismatch across prompt pairs in Figure 3b. The results show that Top-K exhibits a



(a) Distribution of edit similarity (b) Token mismatch rate (c) Similarity vs. accuracy variation

Figure 3: Prompt sensitivity of parallel decoding methods under near-identical prompt templates. (a) Cumulative fraction of examples as a function of edit similarity under prompt variations for Vanilla and Top-K decoding. (b) Position-wise token mismatch rates for Vanilla and Top-K decoding across prompt pairs. (c) Edit similarity vs. accuracy variation across AR and dLLM decoding methods, including Vanilla, threshold-based, and Top-K.

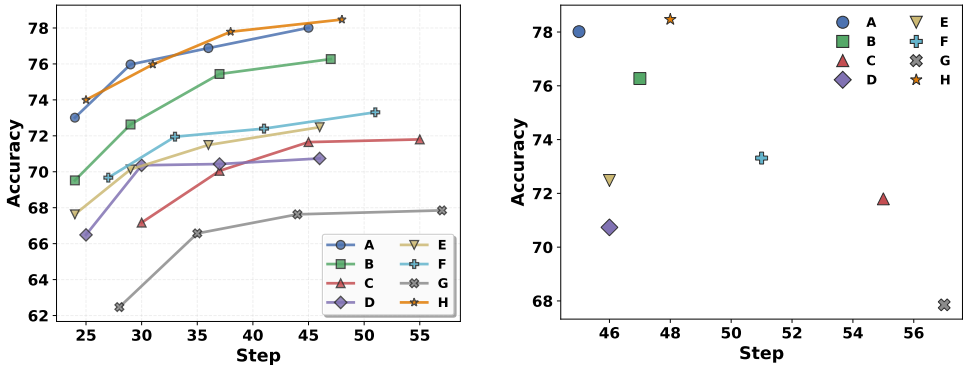
higher token mismatch rate at early output positions, which subsequently propagates and amplifies across later generation positions. Ultimately, the mismatch rate drops sharply toward the end of the sequence due to *EOS* token padding. These results indicate that parallel decoding is more prone to produce inconsistent outputs under subtle prompt variations. In practice, AR models generate text in a left-to-right order, while dLLMs iteratively determine both what tokens to generate and where to unmask at each decoding step. This flexibility makes the decoding trajectory jointly dependent on token prediction and position selection. In Appendix C.6, we present a case study illustrating how subtle prompt variations steer generation dynamics toward divergent trajectories.

Output divergence induces performance variations across prompts To examine how output divergence affects accuracy variation, we evaluate output consistency and accuracy variation across near-identical prompts (see Appendix D.2) for vanilla single-token decoding and for parallel decoding strategies. Figure 3c reveals a clear pattern between edit similarity and accuracy variation, where lower output similarity corresponds to larger performance fluctuations across prompt templates. Consequently, parallel decoding amplifies sensitivity to subtle prompt perturbations by heightening output divergence. In Appendix C.6, we provide a case study indicating that subtle prompt variations can induce divergent reasoning trajectories and yield different final answers. We further evaluate the autoregressive model using Qwen3-8B (Yang et al., 2025) across various prompt templates. The AR baseline exhibits substantially lower variation in accuracy than dLLMs, as the next-token prediction paradigm inherently offers less decoding flexibility. These findings indicate that the greater decoding freedom in dLLMs promotes divergent outputs, resulting in greater variation in task performance.

Parallel decoding is more susceptible to minor prompt variations Table 6 in Appendix C.5 presents the accuracy of the single-token decoding method and different parallel decoding strategies across various near-identical prompt templates on GSM8K under LLaDA-8B-Instruct. We quantify the sensitivity of decoding methods to prompt variation using the standard deviation and range of accuracy across prompt templates. Our results show that parallel decoding methods exhibit substantially greater sensitivity to minor variations of the prompt template. In practice, decoding more tokens per step leads to greater performance variability, as more concurrently decoded tokens are more prone to inter-token dependency conflicts. Moreover, Top-K methods exhibit considerably higher standard deviation and range than threshold-based decoding methods, suggesting that they are more susceptible to prompt template variations. This stems from Top-K decoding potentially sampling low-confidence tokens, which disrupts token dependencies under the conditional independence assumption, thereby amplifying sensitivity to minor prompt variations. **These findings indicate that parallel decoding methods are particularly susceptible to inconsistent evaluation results due to their high sensitivity to prompt templates.**

4.2 Effects of Decoding Budget and Prompt Templates on Evaluation

Experimental setup Diffusion large language models generate text through iterative denoising, progressively unmasking tokens across multiple denoising steps. While increasing the number of denoising steps is generally expected to improve generation quality, a natural question arises: *Do more denoising steps necessarily lead to better performance on downstream tasks?* In this subsection, we empirically investigate how the number of denoising steps and the choice of prompt template jointly



(a) Accuracy vs. decoding steps within prompts (b) Accuracy vs. decoding steps across prompts

Figure 4: Accuracy of parallel decoding method across a set of semantically equivalent prompt templates. (a) Accuracy of threshold-based parallel decoding method with varying thresholds $\gamma \in \{0.6, 0.7, 0.8, 0.9\}$ across various prompts. (b) Accuracy of threshold-based parallel decoding method ($\gamma = 0.9$) across various prompts.

affect evaluation performance. In our experiments, we construct a set of semantically equivalent prompt templates A–H with lexical and stylistic variations but identical semantics and output format instructions, as presented in Appendix D.3. We conduct experiments with threshold-based parallel decoding across varying confidence thresholds $\gamma \in \{0.6, 0.7, 0.8, 0.9\}$, obtaining varying numbers of denoising steps per generation. The detailed results of accuracy and numbers of denoising steps of each decoding method are presented in Appendix C.8.

Increasing decoding budget improves accuracy within each prompt template Figure 4a reports the accuracy of the threshold-based parallel decoding method across various prompt templates on GSM8K under LLaDA-8B-Instruct, where different confidence thresholds $\gamma \in \{0.6, 0.7, 0.8, 0.9\}$ produce varying numbers of denoising steps. The results show that increasing the number of denoising steps consistently improves accuracy within each prompt template. *This suggests an inherent trade-off between decoding budget and accuracy, motivating future methods that can overcome this limitation.*

A larger decoding budget does not necessarily yield better performance across prompts Figure 4b shows the accuracy of threshold-based decoding (γ=0.9) across different prompt templates (A-H), where each point corresponds to a prompt template, with the x-axis indicating the average number of denoising steps and the y-axis indicating the accuracy. Notably, even when the number of denoising steps is comparable, accuracy can vary substantially across prompt templates. For instance, template H achieves 78.47% accuracy with 48 steps, while template G achieves only 67.85% accuracy despite consuming 57 steps, a significant degradation of 10.62 percentage points. **This reveals that performance differences cannot be attributed solely to decoding budget, as a well-designed prompt template can achieve superior performance with fewer steps.**

Prompt templates dominate evaluation results Figure 5 shows the accuracy of threshold-based decoding under varying confidence thresholds, and the Vanilla baseline across four semantically equivalent prompt templates (i.e., G, F, B, H) on GSM8K under LLaDA-8B-Instruct. The experimental results show that prompt template choice has a dominant effect on evaluation metrics. Surprisingly, while increasing the confidence threshold yields moderate performance gains within a given prompt template, the choice of prompt template has a far greater impact on performance. For instance, parallel decoding under prompt template H with a low acceptance threshold of 0.6 achieves 74% accuracy within only 25 steps, substantially outperforming vanilla decoding under

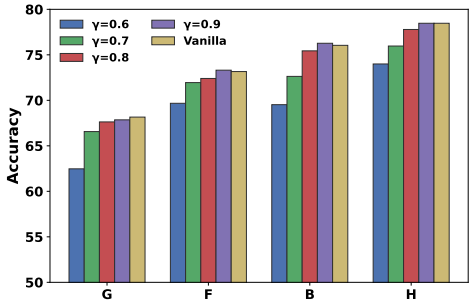


Figure 5: Accuracy variability of various decoding methods across a set of semantically equivalent prompt templates.

prompt template G, which demands 128 denoising steps while yielding a remarkably lower accuracy of 68.16%. An effective prompt template can consistently achieve superior performance across decoding configurations. These findings indicate that reported performance gains in decoding methods may stem from the choice of prompt template, underscoring the need for multi-prompt evaluation.

4.3 Potential Factors Contributing to Variation in Evaluation Metrics

How does few-shot evaluation affect the evaluation of decoding methods? To investigate the effect of few-shot prompting on the performance of decoding methods, we conduct experiments with current decoding methods under different few-shot prompting settings. Table 12 in Appendix C.8 reports the accuracy of vanilla and parallel decoding methods across varying few-shot settings on GSM8K under LLaDA-8B-Instruct. Few-shot evaluation consistently improves accuracy across all methods, yet the performance gain differs substantially between single token decoding and parallel decoding methods. The Vanilla baseline achieves superior performance with 2-shot examples, followed by a decline. In contrast, parallel decoding methods achieve their best performance in settings with more examples of few-shots, such as the 4-shot setting. The experimental results indicate that varying the number of few-shot examples markedly affects the evaluation results of dLLM decoding methods. Furthermore, we observe that KV-reuse decoding methods like Fast-dLLM exhibit larger inference speedup advantage as the number of few-shot examples grows. This is because longer prompts increase the overhead of repeatedly encoding the prefix across denoising steps, whereas KV reuse largely eliminates this redundant computation. Therefore, the choice of few-shot configuration dictates not only accuracy but also the comparative inference efficiency of decoding methods. Overall, the few-shot configuration substantially influences both the observed accuracy and the relative inference efficiency between the decoding methods.

How does generation length affect evaluation results? dLLMs generate text by iteratively denoising over a fixed generation length, and increasing the generation length is generally expected to improve generation quality. However, our experiments reveal that increasing generation length does not consistently yield performance gains. Specifically, we construct a set of zero-shot CoT prompt templates with identical task instructions and output format requirements, as presented in Appendix D.4. These prompt templates differ only in their reasoning instructions, ranging from no explicit reasoning instructions (Prompt A) to increasingly detailed ones (Prompts B–E). Figure 6 shows the accuracy of the threshold-based parallel decoding method ($\gamma = 0.8$) with varying generation length on GSM8K. The results show that accuracy improves as the generation length increases from 64 to 128 across all prompt templates, suggesting that an insufficient generation length may constrain the model's reasoning process and thereby harm accuracy. For prompt templates B–E, accuracy consistently improves with increasing generation length, stabilizing beyond 512. In contrast, for prompt template A, accuracy degrades when the generation length exceeds 128, as a larger generation length may create a mismatch between the predefined length and the task-required response length. We find that under template A, the model tends to skip intermediate reasoning steps and directly output final answers when the generation length exceeds 128, degrading generation quality. Our findings show that different prompt templates exhibit inconsistent preferences for generation length, with increasing generation length sometimes degrading performance for certain prompts.

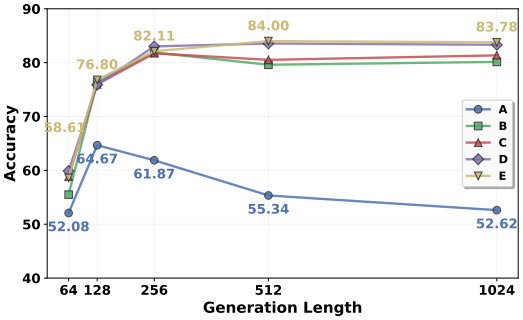


Figure 6: Accuracy of threshold-based parallel decoding under different generation lengths across prompt templates.

Conducting evaluations across GPU types risks unreliable evaluation results. Table 13 in Appendix C.9 reports the accuracy discrepancies across different GPU types, including L40, Pro6000, A100, and RTX 4090, under identical decoding method and evaluation settings. We observe that non-trivial accuracy discrepancies arise across GPU types even under identical experimental conditions, indicating that GPU types are a non-negligible factor in evaluation. This performance dif-

ference across GPU types stems from the non-associative nature of floating-point arithmetic under limited numerical precision (Yuan et al., 2025). Our analysis shows that evaluating decoding methods across different hardware platforms can introduce evaluation inconsistency, potentially affecting the reliability of conclusions. Furthermore, we also conduct experiments across GPU types under FP32 precision. Table 14 in Appendix C.9 reports the accuracy discrepancies across GPU types under identical decoding methods and prompt templates, evaluated in FP32 precision. The results show that the evaluation results on L40 and A100 are nearly identical, suggesting that adopting FP32 precision can substantially mitigate the evaluation variance introduced by hardware.

5 Discussion

Prior work on decoding methods for dLLMs has primarily focused on improving inference efficiency, with little attention paid to the reliability of their evaluation. By revealing evaluation concerns in our community, our goal is to establish more reliable evaluation practices that advance future research. Based on our findings, we propose the following practical suggestions for evaluating decoding methods in diffusion large language models.

- Report performance metrics across multiple prompt templates. Evaluating dLLM decoding methods with a single prompt template can undermine the reliability of the evaluations. Our experiments in Appendix C.10 show that multi-prompt evaluation can effectively mitigate this evaluation inconsistency.
- Conduct evaluations on identical GPU platforms. When conducting evaluations across different GPU platforms, we recommend adopting FP32 precision or other more efficient techniques for deterministic inference, such as LayerCast (Yuan et al., 2025), to mitigate evaluation variance introduced by hardware platforms.
- Provide detailed evaluation settings. We recommend thoroughly reporting the evaluation configurations, including the employed prompt template, algorithmic parameters, generation configurations (e.g., generation length), and hardware specifications (e.g., GPU type) to ensure reproducibility.

6 Conclusion

In this work, we conduct a rigorous evaluation of current decoding methods for dLLMs across evaluation settings. We find that the performance of parallel decoding methods is highly prompt-dependent, undermining the reliability of single-prompt evaluation protocols. Through comprehensive experiments, we attribute this evaluation inconsistency to the high sensitivity of parallel decoding methods to subtle prompt template variations. These findings highlight the importance of multi-prompt evaluation for dLLM decoding methods. Furthermore, our experiments indicate that other overlooked evaluation settings (e.g., hardware) can also notably affect the assessment of decoding methods. Overall, our work aims to provide guidelines for uncovering potential evaluation discrepancies and facilitating fair, reproducible evaluations. We hope that our study deepens the understanding of the evaluation challenges in dLLMs and advances future research.

Limitations Our work aims to reveal potential evaluation issues for evaluating dLLM decoding methods. Although we provide practical guidelines to promote reliable evaluation, a fully standardized, comprehensive evaluation framework remains an important direction for future work.

Ethics Statement

This work aims to improve the reliability and reproducibility of evaluations for diffusion large language models. All experiments are conducted on public benchmarks and involve no human subjects, private data, or personally identifiable information. We identify no direct societal risks beyond those broadly associated with language model research. Our work conducts evaluation using existing open-source models and releases no new models, posing no additional safety risks. This work aims to shed light on evaluation illusions in dLLMs and guide future research toward more reliable assessment practices.

Reproducibility Statement

To ensure reproducibility, we provide detailed implementation parameters in Appendix B. Additionally, we release experimental code in an anonymous repository. We will release the evaluation code to facilitate reproducibility and future research.

Use of Large Language Models

In our work, large language models were used solely for literature search and language polishing. The authors are responsible for all technical contributions, experimental implementations, and evaluation results.

References

Sudhanshu Agrawal, Risheek Garrepalli, Raghavv Goel, Christopher Lott, Fatih Porikli, and Mingu Lee. Structuring the future: Diffusion llm speculative decoding via calibrated draft graphs. *arXiv preprint arXiv:2509.18085*, 2025.

Wenrui Bao, Zhiben Chen, Dan Xu, and Yuzhang Shang. Learning to parallel: Accelerating diffusion large language models via learnable parallel decoding. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=bFJ8Sdr224.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=WBcBhT1NKO.

Xinhua Chen, Sitao Huang, Cong Guo, Chiyue Wei, Yintao He, Jianyi Zhang, Hai Helen Li, and Yiran Chen. DPad: Efficient diffusion language models with suffix dropout. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=0yOsSMU1eY.

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. dparallel: Learnable parallel decoding for dLLMs. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=hVOcstAURb.

Yifeng Gao, Ziang Ji, Yuxuan Wang, Biqing Qi, Hanlin Xu, and Linfeng Zhang. Self speculative decoding for diffusion large language models. *arXiv preprint arXiv:2510.04147*, 2025.

Feng Hong, Geng Yu, Yushi Ye, Haicheng Huang, Huangjie Zheng, Ya Zhang, Yanfeng Wang, and Jiangchao Yao. Wide-in, narrow-out: Revokable decoding for efficient and effective DLLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=XtLQHlNLxy.

Zhanqiu Hu, Jian Meng, Yash Akhauri, Mohamed S Abdelfattah, Jae-sun Seo, Zhiru Zhang, and Udit Gupta. Flashdlm: Accelerating diffusion language model inference via efficient kv caching and guided diffusion. *arXiv preprint arXiv:2505.21467*, 2025.

Andong Hua, Kenan Tang, Chenhe Gu, Jindong Gu, Eric Wong, and Yao Qin. Flaw or artifact? rethinking prompt sensitivity in evaluating llms. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 19900–19910, 2025.

Yuchu Jiang, Yue Cai, Xiangzhong Luo, Jiale Fu, Jiarui Wang, Chonghan Liu, et al. d^2cache: Accelerating diffusion-based llms via dual adaptive caching. In *International Conference on Learning Representations*, volume 2026, pp. 137511–137531, 2026.

Zhengbao Jiang, Frank F Xu, Jun Araki, and Graham Neubig. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438, 2020.

Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Yi Liang, Soroush Vosoughi, and Shiwei Liu. Diffusion language model knows the answer before it decodes. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=g88nt4ieTG.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, William B Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd workshop on knowledge extraction and integration for deep learning architectures*, pp. 100–114, 2022.

Xuejie Liu, Yap Vit Chun, Yitao Liang, and Anji Liu. Lookahead path likelihood optimization for diffusion llms. *arXiv preprint arXiv:2602.03496*, 2026.

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyuan Wei, Shaobo Wang, and Linfeng Zhang. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025.

Guanxi Lu, Hao Mark Chen, Yuto Karashima, Zhican Wang, Daichi Fujiki, and Hongxiang Fan. Semantic-aware diffusion LLM inference with adaptive block size. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=0Cv9PwL7cI.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8086–8098, 2022.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dKV-cache: The cache for diffusion language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Gppo2JImHs.

Moran Mizrahi, Guy Kaplan, Dan Malkin, Rotem Dror, Dafna Shahaf, and Gabriel Stanovsky. State of what art? a call for multi-prompt llm evaluation. *Transactions of the Association for Computational Linguistics*, 12:933–949, 2024.

Quan Nguyen-Tri, Mukul Ranjan, and Zhiqiang Shen. Attention is all you need for KV cache in diffusion LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=zkUbhdAiFJ.

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. URL https://openreview.net/forum?id=KnqiC0znVF.

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. True few-shot learning with language models. *Advances in neural information processing systems*, 34:11054–11070, 2021.

Felipe M Polo, Ronald Xu, Lucas Weber, Mírian Silva, Onkar Bhardwaj, Leshem Choshen, Allysson F de Oliveira, Yuekai Sun, and Mikhail Yurochkin. Efficient multi-prompt evaluation of llms. *Advances in Neural Information Processing Systems*, 37:22483–22512, 2024.

Amirhossein Razavi, Mina Soltangheis, Negar Arabzadeh, Sara Salamat, Morteza Zihayat, and Ebrahim Bagheri. Benchmarking prompt sensitivity in large language models. In *European Conference on Information Retrieval*, pp. 303–313. Springer, 2025.

Yusuke Sakai, Adam Nohejl, Jiangnan Hang, Hidetaka Kamigaito, and Taro Watanabe. Toward the evaluation of large language models considering score variance across instruction templates. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 499–529, 2024.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=RIu5lyNXjT.

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

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

Albert Webson and Ellie Pavlick. Do prompt-based models really understand the meaning of their prompts? In *Proceedings of the 2022 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pp. 2300–2344, 2022.

Qingyan Wei, Yaojie Zhang, Zhiyuan Liu, Puyu Zeng, Yuxuan Wang, Biqing Qi, Dongrui Liu, and Linfeng Zhang. Accelerating diffusion large language models with slowfast sampling: The three golden principles. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=Uh17FiwF4q.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=3Z3Is6hnOT.

Shutong Wu and Jiawei Zhang. Free draft-and-verification: Toward lossless parallel decoding for diffusion large language models. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025. URL https://openreview.net/forum?id=9Te9gWhJTT.

Tianyi Wu, Xiaoxi Sun, Yanhua Jiao, Yulin Li, Yixin Chen, Yun-Hao Cao, Yi-Qi Hu, and Zhuotao Tian. Dynamic-dLLM: Dynamic cache-budget and adaptive parallel decoding for training-free acceleration of diffusion LLM. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=SdnkB5pGbq.

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

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

Jiayi Yuan, Hao Li, Xinheng Ding, Wenya Xie, Yu-Jhe Li, Wentian Zhao, Kun Wan, Jing Shi, Xia Hu, and Zirui Liu. Understanding and mitigating numerical sources of nondeterminism in LLM inference. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Q3qAsZAEZw.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *International conference on machine learning*, pp. 12697–12706. Pmlr, 2021.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, et al. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025.

Kaijie Zhu, Jindong Wang, Jiaheng Zhou, Zichen Wang, Hao Chen, Yidong Wang, Linyi Yang, Wei Ye, Yue Zhang, Neil Gong, et al. Promptrobust: Towards evaluating the robustness of large language models on adversarial prompts. In *Proceedings of the 1st ACM workshop on large AI systems and models with privacy and safety analysis*, pp. 57–68, 2023.

Appendix

Table of Contents

A Related Work **14**

B Experimental Details **15**
B.1 Decoding methods for dLLMs 15
B.2 Implementation details 16
B.3 The Definition of Kendall's Tau 17
B.4 The definition of token-level edit similarity 17

C Experimental Results **17**
C.1 Ranking stability of decoding methods across different prompts 17
C.2 Statistical confidence analysis with bootstrap resampling 17
C.3 Experimental results on code benchmarks 18
C.4 Experimental results on Dream 18
C.5 Sensitivity of decoding methods to prompt templates 19
C.6 Effect of prompt variations on generation trajectories 19
C.7 Empirical analysis of the speed–accuracy trade-off 20
C.8 Experimental results across different prompt templates 21
C.9 Evaluation results with FP32 precision 22
C.10 Experiments on multi-prompt evaluation 22

D Examples of Prompt Templates and Responses **23**
D.1 Prompt templates for mathematical and code benchmarks 23
D.2 Near-identical prompt templates 24
D.3 Semantically equivalent prompt templates 24
D.4 Chain of thought prompt templates 25

A Related Work

Decoding methods for dLLMs Recent efforts to accelerate dLLM inference have focused on two primary challenges: reducing the computational cost per denoising step and decreasing the number of denoising steps (Hong et al., 2026; Li et al., 2026). The bidirectional attention of dLLMs precludes the use of standard autoregressive KV caching, motivating several works to propose approximate caching strategies. dKV-Cache proposes delayed and conditioned KV caching, while Fast-dLLM introduces block-wise approximate KV caching tailored for bidirectional diffusion models (Ma et al., 2025; Wu et al., 2026a). Other methods further reduce computational overhead per step by exploiting the stability of the token representation, applying selective cache refresh, or introducing sparse computation patterns (Liu et al., 2025; Nguyen-Tri et al., 2026; Jiang et al., 2026; Song et al., 2026; Chen et al., 2026a). Additionally, some approaches aim to reduce the number of denoising steps by unmasking multiple tokens per step (Wei et al., 2026; Lu et al., 2026; Li et al., 2026; Bao et al., 2026; Wu & Zhang, 2025; Agrawal et al., 2025; Gao et al., 2025). EB-Sampler uses an entropy-bounded unmasking strategy to control the number of decoded tokens per step (Ben-Hamu et al., 2025). WINO proposes a draft-and-verify decoding strategy that allows aggressively drafted tokens to be re-masked upon verification failure (Hong et al., 2026). These methods have been reported to achieve substantial inference speedups without degradation in generation quality. Our work reveals an evaluation illusion in dLLMs: method rankings vary significantly across prompt templates, undermining the reliability of single-prompt evaluations.

Evaluation robustness in LLMs LLM evaluation has been shown to be highly sensitive to seemingly minor design choices, such as prompt wording, formatting, and demonstration selection and ordering (Jiang et al., 2020; Webson & Pavlick, 2022; Perez et al., 2021; Liu et al., 2022; Zhao et al., 2021; Lu et al., 2022). Prior work on in-context learning has shown that prompt format and demonstration order can cause substantial variance in few-shot accuracy (Zhao et al., 2021; Lu et al., 2022). Recent work also demonstrates that LLM performance is sensitive to formatting changes (Sclar et al., 2024) and broader prompt formulation variations (Razavi et al., 2025). Adversarial perturbations at the character, word, sentence, and semantic levels further amplify this sensitivity (Zhu et al., 2023). To achieve reliable estimates, some work advocates evaluating models across multiple instruction templates and reporting performance scores aggregated over these templates (Mizrahi et al., 2024; Polo et al., 2024; Sakai et al., 2024). Recent analysis further shows that some observed prompt sensitivity may stem from evaluation artifacts such as rigid answer matching or likelihood-based scoring (Hua et al., 2025). While these studies reveal the instability of LLM evaluation, they primarily focus on evaluating the performance of autoregressive LLMs. By introducing an extra position selection dimension, dLLM decoding further amplifies prompt-induced generation instability compared to next-token prediction. In our work, we investigate how prompt templates affect the evaluation of dLLM decoding methods, showing that parallel decoding methods are highly sensitive to the choice of prompt template.

B Experimental Details

B.1 Decoding methods for dLLMs

Here, we summarize the decoding methods for diffusion language models examined in this work. For each method, we use the best-performing hyperparameter settings reported in the corresponding paper or provided in the official codebase. We carefully verify these configurations during implementation and report all detailed hyperparameter settings below.

Low-confidence The low-confidence remasking strategy is a widely used decoding method for dLLMs (Nie et al., 2025). During generation, the model predicts all masked positions but commits only the most confident token, leaving the remaining positions masked for subsequent refinement. The single-token decoding method achieves strong performance but requires a number of denoising steps equal to the generation length. In experiments, we use this method as the *Vanilla* baseline with generation length $L = 128$ and block size $B = 32$.

Fast-dLLM Fast-dLLM combines block-wise approximate KV caching with confidence-aware parallel decoding to accelerate inference in dLLMs (Wu et al., 2026a). The caching component reuses approximate KV states across denoising steps within a block-wise decoding process, while the decoding component commits all predictions exceeding a confidence threshold and keeps the remaining positions masked. Thus, Fast-dLLM reduces both redundant per-step computation and the number of denoising steps. Following the official implementation of Fast-dLLM (Wu et al., 2026a), we evaluate Fast-dLLM with confidence threshold $\tau = 0.9$, and block size $B = 32$. For the Dream model, following the official codebase, we adopt the same hyperparameter settings as those used for LLaDA.

AdaBlock-dLLM AdaBlock-dLLM addresses the fixed block size limitation of semi-autoregressive dLLM decoding by adaptively determining the block size during generation (Lu et al., 2026). At the start of each block, the model inspects a local context window over the predicted sequence and examines the associated confidence scores. If a semantic delimiter is identified with high confidence above the threshold, the block boundary is placed at that position. Otherwise, the predefined default block size is adopted. Following the official implementation of AdaBlock-dLLM (Lu et al., 2026), we evaluate AdaBlock-dLLM with predefined default block size $B = 32$, delimiter set $\mathcal{D} = [198]$, confidence threshold $\tau = 0.9$ and delimiter threshold $\tau_D = 0.3$. For the Dream model, following the official codebase, we use the delimiter set $\mathcal{D} = \{198, 271, 280, 319, 340, 382, 401, 532, 624, 630, 692, 698, 921, 1248, 1837, 1939, 2219, 2533, 3276, 3876, 4894, 5267, 14750, 68327\}$ and a delimiter threshold of $\tau_D = 0.5$.

AdaBlock-Fast AdaBlock-Fast (Lu et al., 2026) integrates the adaptive block size determination mechanism of AdaBlock-dLLM into the Fast-dLLM decoding framework. Following the official implementation of AdaBlock-dLLM (Lu et al., 2026), we evaluate AdaBlock-Fast with predefined default block size $B = 32$, delimiter set $\mathcal{D} = [198]$, confidence threshold $\tau = 0.9$ and delimiter threshold $\tau_D = 0.3$. For the Dream model, following the official codebase, we use the delimiter set $\mathcal{D} = \{198, 271, 280, 319, 340, 382, 401, 532, 624, 630, 692, 698, 921, 1248, 1837, 1939, 2219, 2533, 3276, 3876, 4894, 5267, 14750, 68327\}$ and a delimiter threshold of $\tau_D = 0.5$.

dKV-Cache-Decode dKV-Cache adapts KV caching to diffusion language model decoding through a delayed and conditioned caching strategy (Ma et al., 2025). The method exploits the observation that token representations stabilize at different rates during the denoising process. Instead of immediately reusing all KV states, it delays cache reuse and conditions caching decisions on the current token states. Following the official implementation of dKV-Cache-Decode (Ma et al., 2025), we evaluate dKV-Cache-Decode with a cache refresh step $s = 8$, and block size $B = 32$. To achieve parallel decoding, we commit the two tokens with the highest predicted confidence at each denoising step. For the Dream model, following the official codebase, we apply an entropy-based remasking strategy and set the cache refresh step to $s = 4$.

Elastic-Cache Elastic-Cache is a training-free KV caching method for diffusion LLMs that adaptively decides when to refresh cached states and from which layer recomputation should resume (Nguyen-Tri et al., 2026). By using attention-aware drift to invalidate caches, Elastic-Cache reuses stable shallow-layer and off-window `MASK` caches instead of recomputing all cache states at every denoising step. Following the official implementation of Elastic-Cache (Nguyen-Tri et al., 2026), we evaluate Elastic-Cache with a refresh threshold of 0.9, a window length of 32, a confidence threshold of 0.9, and a tracked-token count of 1.

WINO WINO achieves revocable parallel decoding via a wide-in narrow-out draft-and-verify mechanism (Hong et al., 2026). During the draft stage, multiple candidate tokens are unmasked in parallel. The verification stage evaluates each drafted token under full bidirectional context, identifying unreliable predictions. Tokens that fail verification are re-masked and refined in subsequent denoising steps, while reliable tokens remain committed. This mechanism allows for aggressive parallel decoding without permanently committing to erroneous predictions. Following the official implementation of WINO (Hong et al., 2026), we evaluate WINO with a forward confidence threshold of 0.6, a backward verification threshold of 0.9, and an acceptance window of 30 tokens. Additionally, consistent with the original implementation, we set the block size to half of the generation length.

B.2 Implementation details

Evaluation protocol In the main experiments, we evaluate the performance of current decoding methods on math and code benchmarks. To eliminate the effects of hardware and system configuration on evaluation results, our experiments are conducted on a single machine with a single GPU and a batch size of 1. All experiments are conducted on L40 GPUs unless otherwise stated. Experiments across hardware platforms are conducted on L40, Pro6000, A100, and RTX 4090 GPUs. The prompt templates used in our evaluation are sourced from lm-evaluation-harness and a set of manually crafted prompts (Appendix D). For mathematical datasets, we employ Math-Verify to assess the accuracy of generated answers. For code datasets, following prior work (Lu et al., 2026), we employ the Evaluate module from Hugging Face Evaluate to assess the correctness of generated solutions. Unless otherwise specified, for the main experiments, we set the generation length to 128 and adopt the semi-autoregressive decoding strategy with a block size of 32.

Ensuring reproducibility across multiple runs Our work aims to investigate the evaluation inconsistency in dLLMs. We first adopt rigorous experimental controls to eliminate the effect of computational randomness, and ensure that evaluation results are reproducible under identical configurations. Specifically, we adopt the following experimental settings: **(1) Deterministic computation.** We control sources of randomness in Python, NumPy, and PyTorch, and enforce deterministic computation in cuDNN. **(2) Consistent hardware platform.** All methods are evaluated on the same GPU to avoid variation caused by hardware differences. **(3) Single-GPU evaluation with a batch**

size of 1. Each evaluation is conducted on a single GPU with a batch size of 1 to avoid nondeterminism introduced by parallel execution. **(4) Deterministic decoding.** All decoding methods are evaluated using deterministic decoding without stochastic sampling. Under these controlled settings, repeated evaluations under the same prompt template produce identical results, confirming that the performance variations analyzed in our experiments are not caused by run-to-run computational randomness.

B.3 The Definition of Kendall's Tau

Kendall's tau quantifies the agreement between two rankings. Given two rankings r and r' over the same set of n items, let N_c and N_d denote the numbers of concordant and discordant pairs, respectively. We define Kendall's τ_b as

$$\tau_b(r, r') = \frac{N_c - N_d}{\sqrt{(N_0 - T_r)\,(N_0 - T_{r'})}}, \tag{5}$$

where $N_0 = \binom{n}{2}$ denotes the total number of item pairs, and T_r and $T_{r'}$ denote the numbers of tied pairs in rankings r and r', respectively. Higher values indicate stronger agreement between the two rankings, whereas lower values indicate greater disagreement.

B.4 The definition of token-level edit similarity

To quantify response variation across prompt templates, we compute the pairwise token-level edit similarity between generated responses. Given two responses, r and r', generated for the same query with different prompt templates, let $D_{\mathrm{Lev}}(r, r')$ denote their token-level Levenshtein distance, which is the minimum number of insertions, deletions, and substitutions of a single token required to transform r into r'. Formally, the token-level edit similarity is defined as

$$S(r, r') = 1 - \frac{D_{\mathrm{Lev}}(r, r')}{\max(|r|, |r'|)}, \tag{6}$$

where $|r|$ and $|r'|$ denote the number of tokens in the two responses, respectively. The similarity score $S(r, r') \in [0, 1]$, where $S(r, r') = 1$ indicates that r and r' are identical sequences, while smaller values indicate greater token-level differences.

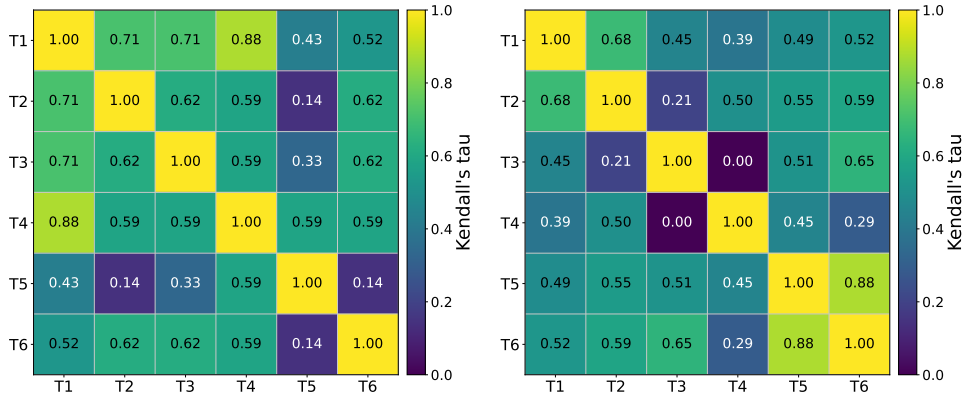
C Experimental Results

C.1 Ranking stability of decoding methods across different prompts

Figure 7a and Figure 7b show the pairwise Kendall's tau correlations (See Definition in Appendix B.3) among method rankings obtained under different prompt templates for LLaDA-8B-Instruct and LLaDA-1.5, respectively. The average pairwise Kendall's tau across prompt templates is only 0.539 for LLaDA-8B-Instruct and 0.477 for LLaDA-1.5, indicating pervasive evaluation inconsistency across dLLM decoding methods.

C.2 Statistical confidence analysis with bootstrap resampling

To quantify statistical uncertainty, we conduct example level bootstrap sampling with 1,000 iterations on GSM8K. For each run, we resample examples with replacement using identical indices across all decoding methods and prompt templates to ensure fair comparisons. We then recompute the average pairwise Kendall's τ across the prompt templates, reporting 95% percentile bootstrap confidence intervals. The average pairwise Kendall's τ is 0.539 with a 95% confidence interval of $[0.224, 0.619]$ for LLaDA-8B-Instruct, and 0.477 with a 95% confidence interval of $[0.206, 0.537]$ for LLaDA-1.5. The results show that even the upper bounds of the 95% confidence intervals do not exceed 0.62, indicating that evaluation inconsistency remains pronounced under sample variability. These findings confirm that the observed evaluation inconsistency across prompts is widespread rather than an artifact of sample resampling.



(a) LLaDA-8B-Instruct (b) LLaDA-1.5

Figure 7: Pairwise Kendall's tau correlation matrices across prompt templates on GSM8K for (a) LLaDA-8B-Instruct and (b) LLaDA-1.5. Each cell quantifies the ranking agreement of decoding methods between two prompt templates, with higher values indicating more consistent rankings.

Table 2: The accuracy of vanilla and current decoding methods on GSM8K and MATH500 under LLaDA-1.5, across various prompt templates. *Vanilla* baseline denotes the single-token decoding method. Each numerical subscript indicates the improvement or degradation compared to the vanilla decoding method.

Method	GSM8K						MATH500		Avg.
	Template 1	Template 2	Template 3	Template 4	Template 5	Template 6	Template 7	Template 8	
Vanilla	78.54	**77.86**	73.92	78.01	**77.79**	78.32	**38.20**	33.40	67.01
Parallel decoding methods									
Fast-dLLM	$77.33_{-1.21}$	$76.80_{-1.06}$	$72.55_{-1.37}$	$\mathbf{78.62}_{+0.61}$	$77.26_{-0.53}$	$73.77_{-4.55}$	$34.60_{-3.60}$	$33.60_{+0.20}$	$65.57_{-1.44}$
AdaBlock-dLLM	$77.63_{-0.91}$	$77.18_{-0.68}$	$\mathbf{74.15}_{+0.23}$	$76.04_{-1.97}$	$\mathbf{77.79}_{+0.00}$	$\mathbf{78.62}_{+0.30}$	$37.60_{-0.60}$	$33.00_{-0.40}$	$66.50_{-0.51}$
AdaBlock-Fast	$78.09_{-0.45}$	$\mathbf{77.86}_{+0.00}$	$71.80_{-2.12}$	$78.01_{+0.00}$	$76.50_{-1.29}$	$73.54_{-4.78}$	$34.20_{-4.00}$	$32.20_{-1.20}$	$65.28_{-1.73}$
dKV-Cache-Decode	$75.44_{-3.10}$	$73.84_{-4.02}$	$70.28_{-3.64}$	$72.78_{-5.23}$	$72.25_{-5.54}$	$71.04_{-7.28}$	$29.60_{-8.60}$	$31.80_{-1.60}$	$62.13_{-4.88}$
Elastic-Cache	$\mathbf{79.23}_{+0.69}$	$76.88_{-0.98}$	$\mathbf{74.15}_{+0.23}$	$77.94_{-0.07}$	$76.72_{-1.07}$	$76.04_{-2.28}$	$34.80_{-3.40}$	$\mathbf{34.60}_{+1.20}$	$66.30_{-0.71}$
WINO	$76.65_{-1.89}$	$76.72_{-1.14}$	$73.92_{+0.00}$	$73.09_{-4.92}$	$72.78_{-5.01}$	$73.46_{-4.86}$	$33.80_{-4.40}$	$32.00_{-1.40}$	$64.05_{-2.96}$

C.3 Experimental results on code benchmarks

Table 2 reports the accuracy of vanilla and current decoding methods on GSM8K and MATH500 under LLaDA-1.5. Table 3 and Table 4 report the accuracy of various decoding methods on code datasets across various prompt templates under LLaDA-8B-Instruct and LLaDA-1.5, respectively. The results show that current decoding methods achieve occasional marginal improvements over the vanilla baseline on certain prompt templates. However, degradation is widespread across the majority of prompts. These findings show that current decoding methods do not consistently preserve accuracy across prompt templates while accelerating generation.

C.4 Experimental results on Dream

We further evaluate Dream-v0-Instruct-7B to investigate whether the observed evaluation inconsistency extends across model architectures. Since the official WINO (Hong et al., 2026) codebase does not include an implementation for Dream, we do not report WINO results for this model. In practice, we adopt the best-performing hyperparameter settings reported in the corresponding paper or provided in the official codebase. Table 5 reports the accuracy of various decoding methods on math datasets across various prompt templates under Dream-v0-Instruct-7B. We find that the rankings of decoding methods can vary across prompt templates. Moreover, the results show that although some decoding methods achieve higher accuracy than the vanilla baseline under certain

Table 3: The accuracy of vanilla and current decoding methods on HumanEval and MBPP under LLaDA-8B-Instruct, across various prompt templates. *Vanilla* baseline denotes the single-token decoding method. Each numerical subscript indicates the improvement or degradation compared to the vanilla decoding method.

Method	HumanEval			MBPP			Avg.
	Template 1	Template 2	Template 3	Template 4	Template 5	Template 6	
Vanilla	**35.40**	29.90	**33.50**	36.60	**40.20**	37.60	35.53
Parallel decoding methods							
Fast-dLLM	**35.40**$_{+0.00}$	**32.90**$_{+3.00}$	31.70$_{-1.80}$	37.40$_{+0.80}$	36.40$_{-3.80}$	36.80$_{-0.80}$	35.10$_{-0.43}$
AdaBlock-dLLM	34.10$_{-1.30}$	31.70$_{+1.80}$	32.90$_{-0.60}$	37.20$_{+0.60}$	40.00$_{-0.20}$	**38.20**$_{+0.60}$	**35.68**$_{+0.15}$
AdaBlock-Fast	32.90$_{-2.50}$	29.30$_{-0.60}$	**33.50**$_{+0.00}$	37.60$_{+1.00}$	38.80$_{-1.40}$	38.00$_{+0.40}$	35.02$_{-0.52}$
dKV-Cache-Decode	23.80$_{-11.60}$	20.70$_{-9.20}$	26.20$_{-7.30}$	24.80$_{-11.80}$	29.00$_{-11.20}$	27.00$_{-10.60}$	25.25$_{-10.28}$
Elastic-Cache	32.30$_{-3.10}$	32.30$_{+2.40}$	29.90$_{-3.60}$	**39.40**$_{+2.80}$	37.20$_{-3.00}$	36.20$_{-1.40}$	34.55$_{-0.98}$
WINO	25.00$_{-10.40}$	25.60$_{-4.30}$	28.00$_{-5.50}$	36.40$_{-0.20}$	35.40$_{-4.80}$	36.00$_{-1.60}$	31.07$_{-4.47}$

Table 4: The accuracy of vanilla and current decoding methods on HumanEval and MBPP under LLaDA-1.5, across various prompt templates. *Vanilla* baseline denotes the single-token decoding method. Each numerical subscript indicates the improvement or degradation compared to the vanilla decoding method.

Method	HumanEval			MBPP			Avg.
	Template 1	Template 2	Template 3	Template 4	Template 5	Template 6	
Vanilla	**42.10**	38.40	**43.30**	**39.60**	**41.40**	**40.80**	**40.93**
Parallel decoding methods							
Fast-dLLM	36.00$_{-6.10}$	32.90$_{-5.50}$	38.40$_{-4.90}$	38.20$_{-1.40}$	40.00$_{-1.40}$	38.60$_{-2.20}$	37.35$_{-3.58}$
AdaBlock-dLLM	39.00$_{-3.10}$	**39.60**$_{+1.20}$	42.70$_{-0.60}$	38.60$_{-1.00}$	39.80$_{-1.60}$	40.60$_{-0.20}$	40.05$_{-0.88}$
AdaBlock-Fast	36.00$_{-6.10}$	33.50$_{-4.90}$	38.40$_{-4.90}$	37.20$_{-2.40}$	40.60$_{-0.80}$	39.80$_{-1.00}$	37.58$_{-3.35}$
dKV-Cache-Decode	28.70$_{-13.40}$	28.00$_{-10.40}$	32.90$_{-10.40}$	27.80$_{-11.80}$	31.80$_{-9.60}$	33.00$_{-7.80}$	30.37$_{-10.57}$
Elastic-Cache	37.20$_{-4.90}$	36.60$_{-1.80}$	38.40$_{-4.90}$	38.60$_{-1.00}$	39.60$_{-1.80}$	39.80$_{-1.00}$	38.37$_{-2.57}$
WINO	34.80$_{-7.30}$	33.50$_{-4.90}$	37.80$_{-5.50}$	33.20$_{-6.40}$	36.00$_{-5.40}$	39.20$_{-1.60}$	35.75$_{-5.18}$

prompt templates, they underperform the baseline across the majority of templates. These results suggest that current parallel decoding methods can accelerate generation but can come at a cost to accuracy on downstream tasks.

C.5 Sensitivity of decoding methods to prompt templates

Table 6 reports the accuracy of the decoding methods across various near-identical prompt templates on GSM8K under LLaDA-8B-Instruct. In practice, we quantify the sensitivity of decoding methods to prompt variation using the standard deviation and range of accuracy across prompt templates. The results show that parallel decoding methods exhibit substantially greater sensitivity to minor variations of the prompt template.

C.6 Effect of prompt variations on generation trajectories

In Figure 8, we illustrate how subtle prompt variations induce decoding dynamics in dLLMs. We find that a difference in position selection already emerges at the first decoding step, where prompt A chooses position 2, whereas prompt B chooses position 1. The first token mismatch occurs later

Table 5: The accuracy of vanilla and current decoding methods on GSM8K and MATH500 under Dream-v0-Instruct-7B, across various prompt templates. *Vanilla* baseline denotes the single-token decoding method. Each numerical subscript indicates the improvement or degradation compared to the vanilla decoding method.

Method	GSM8K						MATH500		Avg.
	Template 1	Template 2	Template 3	Template 4	Template 5	Template 6	Template 7	Template 8	
Vanilla	**76.27**	**78.01**	**76.19**	**72.71**	**73.39**	65.28	39.00	**42.60**	65.43
Parallel decoding methods									
Fast-dLLM	75.06$_{-1.21}$	76.72$_{-1.29}$	75.13$_{-1.06}$	72.10$_{-0.61}$	70.96$_{-2.43}$	**69.83**$_{+4.55}$	**41.20**$_{+2.20}$	40.00$_{-2.60}$	65.13$_{-0.31}$
AdaBlock-dLLM	75.89$_{-0.38}$	77.33$_{-0.68}$	**76.19**$_{0.00}$	71.72$_{-0.99}$	71.80$_{-1.59}$	64.90$_{-0.38}$	39.00$_{0.00}$	42.40$_{-0.20}$	64.90$_{-0.53}$
AdaBlock-Fast	74.68$_{-1.59}$	76.72$_{-1.29}$	74.98$_{-1.21}$	70.51$_{-2.20}$	71.11$_{-2.28}$	68.23$_{+2.95}$	40.60$_{+1.60}$	39.60$_{-3.00}$	64.55$_{-0.88}$
dKV-Cache-Decode	67.40$_{-8.87}$	69.75$_{-8.26}$	66.49$_{-9.70}$	64.06$_{-8.65}$	63.00$_{-10.39}$	63.08$_{-2.20}$	32.00$_{-7.00}$	34.60$_{-8.00}$	57.55$_{-7.88}$
dKV-Cache-Decode (Half)	73.39$_{-2.88}$	76.42$_{-1.59}$	73.84$_{-2.35}$	68.39$_{-4.32}$	69.29$_{-4.10}$	66.87$_{+1.59}$	38.80$_{-0.20}$	40.60$_{-2.00}$	63.45$_{-1.98}$
Elastic-Cache	70.51$_{-5.76}$	70.43$_{-7.58}$	69.67$_{-6.52}$	65.88$_{-6.83}$	65.58$_{-7.81}$	63.38$_{-1.90}$	35.60$_{-3.40}$	36.80$_{-5.80}$	59.73$_{-5.70}$

Table 6: Accuracy comparison across prompt templates (A–H) on the GSM8K dataset. Qwen3-8B serves as the autoregressive reference, while LLaDA-8B-Instruct is evaluated under various decoding strategies. *Vanilla* denotes the standard single-token decoding baseline. *Confidence Threshold* decodes all masked tokens whose confidence exceeds γ at each step. *Top-K* decodes the top k most confident masked tokens per step. **Avg** denotes the mean accuracy across templates. **Std** and **Range** denote the sample standard deviation and range of accuracy across templates, quantifying sensitivity to prompt variation.

Decoding Method	**Model**	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**Avg**	**Std**	**Range**
\	AR	91.74	91.96	91.36	91.74	91.81	91.89	91.74	91.05	91.66	0.304	0.910
Vanilla	dLLM	77.26	77.63	77.10	78.17	77.26	77.86	77.94	77.86	77.64	0.387	1.070
Threshold-based decoding methods												
Confidence Threshold ($\gamma = 0.9$)	dLLM	77.48	77.94	77.03	78.32	76.95	77.86	77.63	77.71	77.62	0.459	1.370
Confidence Threshold ($\gamma = 0.8$)		76.19	76.19	76.42	77.63	76.27	77.48	77.33	76.65	76.77	0.611	1.440
Top-K decoding methods												
Top-K ($k = 2$)	dLLM	75.36	73.69	74.45	74.83	75.06	74.68	74.00	72.18	74.28	1.007	3.180
Top-K ($k = 4$)		65.58	66.64	66.57	68.23	66.19	65.05	65.43	64.44	66.02	**1.170**	**3.790**

at position 20, after which token-level disagreements become increasingly frequent at subsequent decoding positions. dLLMs often determine both what tokens to generate and where to unmask at each decoding step. This flexibility makes the decoding trajectory jointly dependent on token prediction and position selection, thereby increasing sensitivity to subtle prompt variations. As shown in Example 1, we present a detailed case study illustrating how minor prompt changes can alter sequence generation. We find that even a subtle variation in the prompt can induce divergent reasoning trajectories and yield different final answers.

C.7 Empirical analysis of the speed–accuracy trade-off

Current dLLM decoding methods substantially improve generation efficiency, which is critical to the practical deployment of dLLMs. Tables 7 and 8 report the number of function evaluations (NFE) for decoding methods whose accuracy is presented in Tables 1 and 2, respectively. The NFE metric measures the number of model forward passes performed during decoding. The experimental results show that existing methods reduce the number of decoding steps and thereby improve generation efficiency. Additionally, Table 9 and Table 10 show the accuracy and decoding speedup of current decoding methods on LLaDA-8B-Instruct and LLaDA-1.5, across various prompt templates. The speedup is measured relative to the decoding latency of Vanilla under the same evaluation configuration. The experimental results show that the performance gains of parallel decoding methods are highly prompt-dependent, and that accelerating generation can degrade downstream task accuracy.

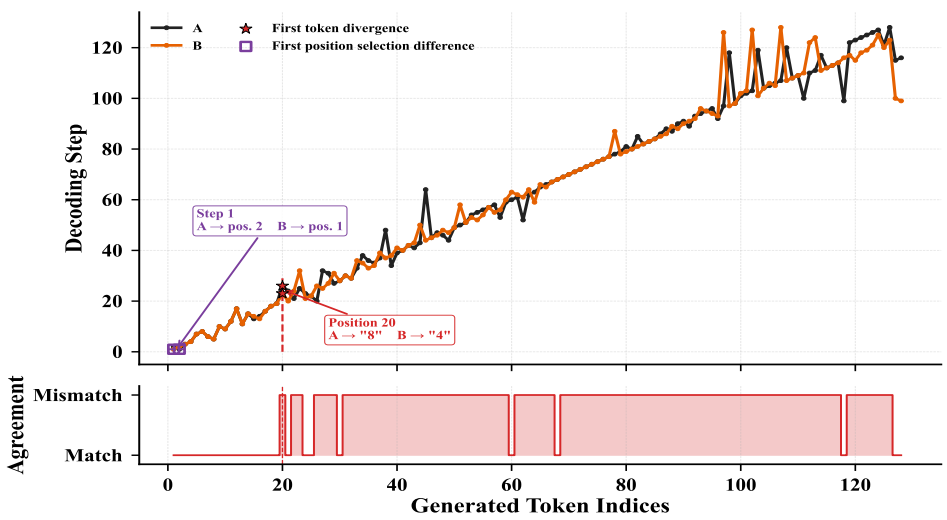


Figure 8: Illustration of prompt-induced decoding divergence. (Top) Comparison of decoding dynamics between two prompt templates: A difference in position selection already emerges at the first decoding step. (Bottom) Position-wise token agreement between the two generated sequences: the first token mismatch occurs at position 20.

Table 7: The decoding steps of vanilla and parallel decoding methods on GSM8K and MATH500 under LLaDA-8B-Instruct, across various evaluation prompt templates. *Vanilla* denotes the low-confidence decoding baseline, which decodes a single masked token with the highest confidence at each step.

Method	GSM8K						MATH500		Avg.
	Template 1	Template 2	Template 3	Template 4	Template 5	Template 6	Template 7	Template 8	
Vanilla	128	128	128	128	128	128	128	128	128.00
Parallel decoding methods									
Fast-dLLM	57	60	56	54	54	53	69	66	58.63
AdaBlock-dLLM	53	56	52	50	50	48	65	62	54.50
AdaBlock-Fast	57	61	57	55	55	55	70	67	59.63
dKV-Cache-Decode	64	64	64	64	64	64	64	64	64.00
Elastic-Cache	54	57	53	50	51	49	68	63	55.63
WINO	31	32	31	27	27	27	44	37	32.00

C.8 Experimental results across different prompt templates

Table 11 reports the accuracy and decoding steps of single-token decoding and confidence-threshold decoding strategies on GSM8K under LLaDA-8B-Instruct across a set of semantically equivalent prompt templates. The results show that prompt templates can play a dominant role in determining evaluation results. An effective template can yield high performance scores even with fewer denoising steps, substantially exceeding the marginal gains from increasing denoising steps.

How does few-shot prompting affect the evaluation of decoding methods? Table 12 reports the accuracy of vanilla and parallel decoding methods across varying few-shot settings on GSM8K under LLaDA-8B-Instruct. Our experimental results show that the number of few-shot examples can affect the performance of decoding methods during evaluation, suggesting that future evaluations should account for this factor.

Table 8: The decoding steps of vanilla and parallel decoding methods on GSM8K and MATH500 under LLaDA-1.5, across various prompt templates. *Vanilla* denotes the low-confidence decoding baseline, which decodes a single masked token with the highest confidence at each step.

Method	GSM8K						MATH500		Avg.
	Template 1	Template 2	Template 3	Template 4	Template 5	Template 6	Template 7	Template 8	
Vanilla	128	128	128	128	128	128	128	128	128.00
Parallel decoding methods									
Fast-dLLM	53	58	55	53	53	49	68	65	56.75
AdaBlock-dLLM	51	54	52	49	49	46	64	60	53.13
AdaBlock-Fast	54	59	56	54	54	53	68	66	58.00
dKV-Cache-Decode	64	64	64	64	64	64	64	64	64.00
Elastic-Cache	54	57	53	50	50	49	66	64	55.38
WINO	30	32	31	28	29	27	43	46	33.25

Table 9: The accuracy and decoding speedup of vanilla and current decoding methods on GSM8K and MATH500 under LLaDA-8B-Instruct, across various prompt templates. *Vanilla* denotes the single-token decoding method. For parallel decoding methods, the speedup relative to Vanilla is reported below each accuracy value.

Method	GSM8K						MATH500		Avg.
	Template 1	Template 2	Template 3	Template 4	Template 5	Template 6	Template 7	Template 8	
Vanilla	**77.33**	74.60	71.72	**76.35**	74.98	76.88	**37.60**	34.00	**65.43**
Fast-dLLM	76.35 (2.46×)	73.16 (2.37×)	70.51 (2.48×)	75.74 (6.85×)	75.82 (9.06×)	74.60 (3.55×)	33.80 (2.03×)	35.20 (5.83×)	64.40 (4.33×)
AdaBlock-dLLM	76.88 (2.39×)	**75.06** (2.31×)	71.95 (2.46×)	75.74 (2.54×)	74.83 (2.56×)	**76.95** (2.67×)	37.00 (1.78×)	34.40 (1.85×)	65.35 (2.32×)
AdaBlock-Fast	76.65 (2.42×)	73.92 (2.34×)	71.27 (2.43×)	75.66 (6.11×)	75.21 (7.73×)	74.15 (3.37×)	34.20 (2.02×)	**36.20** (5.35×)	64.66 (3.97×)
dKV-Cache-Decode	72.93 (1.98×)	69.45 (2.05×)	66.94 (2.00×)	72.93 (4.27×)	72.63 (4.65×)	73.92 (2.64×)	31.80 (2.04×)	33.00 (4.22×)	61.70 (2.98×)
Elastic-Cache	77.10 **(2.99×)**	73.84 **(2.94×)**	**72.18** **(3.05×)**	75.97 **(7.80×)**	**76.88** **(9.64×)**	76.35 **(4.46×)**	33.60 **(2.45×)**	31.80 **(6.59×)**	64.72 **(4.99×)**
WINO	74.68 (2.83×)	72.48 (2.77×)	70.74 (2.93×)	74.45 (3.32×)	72.78 (3.07×)	75.21 (3.95×)	33.40 (1.96×)	34.20 (2.25×)	63.49 (2.89×)

C.9 Evaluation results with FP32 precision

Table 13 shows the accuracy discrepancies across GPU types under identical decoding methods and prompt templates, evaluated in BF16 precision. Table 14 presents the accuracy discrepancies between L40 and A100 under the threshold-based decoding method ($\gamma = 0.8$) across various prompt templates, evaluated in FP32 precision. The results show that the evaluation results remain consistent between the GPU types at the precision of two decimal places, demonstrating that FP32 precision can substantially reduce evaluation variance across different hardware platforms.

C.10 Experiments on multi-prompt evaluation

In our work, we use prompt templates commonly adopted in prior studies to ensure that our evaluation reflects common practice and that the observed inconsistencies in evaluation are not due to specially designed prompts. To investigate the effectiveness of multi-prompt evaluation, we conduct leave-one-template-out experiments to assess the robustness of our evaluation. Concretely, for Templates 1–8 used in Table 1, we exclude each template in turn and calculate Kendall's τ between the rankings based on the average accuracy over the remaining 7 templates and all 8 templates. The leave-one-template-out analysis yields an average Kendall's τ of 0.94, suggesting that aver-

Table 10: The accuracy and decoding speedup of vanilla and current decoding methods on GSM8K and MATH500 under LLaDA-1.5, across various prompt templates. *Vanilla* denotes the single-token decoding method. For parallel decoding methods, the speedup relative to Vanilla is reported below each accuracy value.

Method	GSM8K						MATH500		Avg.
	Template 1	Template 2	Template 3	Template 4	Template 5	Template 6	Template 7	Template 8	
Vanilla	78.54	**77.86**	73.92	78.01	**77.79**	78.32	**38.20**	33.40	**67.01**
Fast-dLLM	77.33 (2.50×)	76.80 (2.35×)	72.55 (2.46×)	**78.62** (6.94×)	77.26 (9.20×)	73.77 (3.65×)	34.60 (1.96×)	33.60 (5.82×)	65.57 (4.36×)
AdaBlock-dLLM	77.63 (2.50×)	77.18 (2.35×)	**74.15** (2.45×)	76.04 (2.56×)	**77.79** (2.60×)	**78.62** (2.74×)	37.60 (1.86×)	33.00 (2.05×)	66.50 (2.39×)
AdaBlock-Fast	78.09 (2.51×)	**77.86** (2.35×)	71.80 (2.43×)	78.01 (6.35×)	76.50 (8.12×)	73.54 (3.36×)	34.20 (1.97×)	32.20 (5.51×)	65.28 (4.08×)
dKV-Cache-Decode	75.44 (1.95×)	73.84 (1.99×)	70.28 (1.96×)	72.78 (4.20×)	72.25 (4.63×)	71.04 (2.53×)	29.60 (1.96×)	31.80 (4.24×)	62.13 (2.93×)
Elastic-Cache	**79.23** **(3.11×)**	76.88 **(2.92×)**	**74.15** **(3.05×)**	77.94 **(7.95×)**	76.72 **(9.84×)**	76.04 **(4.60×)**	34.80 **(2.43×)**	**34.60** **(6.68×)**	66.30 **(5.07×)**
WINO	76.65 (3.07×)	76.72 (2.88×)	73.92 (2.98×)	73.09 (3.20×)	72.78 (2.95×)	73.46 (3.70×)	33.80 (2.11×)	32.00 (2.28×)	64.05 (2.90×)

Table 11: The accuracy and number of decoding steps of vanilla and current decoding methods on GSM8K datasets under LLaDA-8B-Instruct, across various semantically equivalent prompt templates used for evaluation. Vanilla denotes the low-confidence sampling baseline.

Method	**Metric**	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
Vanilla	Acc.	77.41	76.04	72.02	71.27	72.33	73.16	68.16	78.47
	Step	128	128	128	128	128	128	128	128
Threshold-based decoding ($\gamma = 0.6$)	Acc.	73.01	69.52	67.17	66.49	67.63	69.67	62.47	74.00
	Step	24	24	30	25	24	27	28	25
Threshold-based decoding ($\gamma = 0.7$)	Acc.	75.97	72.63	70.05	70.36	70.13	71.95	66.57	75.97
	Step	29	29	37	30	29	33	35	31
Threshold-based decoding ($\gamma = 0.8$)	Acc.	76.88	75.44	71.65	70.43	71.49	72.40	67.63	77.79
	Step	36	37	45	37	36	41	44	38
Threshold-based decoding ($\gamma = 0.9$)	Acc.	78.01	76.27	71.80	70.74	72.48	73.31	67.85	78.47
	Step	45	47	55	46	46	51	57	48

aging results across multiple templates produces a considerably more stable ranking. This further supports our recommendation to evaluate decoding methods using multiple prompt templates rather than relying on a single template.

D Examples of Prompt Templates and Responses

D.1 Prompt templates for mathematical and code benchmarks

Example D.1 presents the prompt templates employed in our mathematical benchmark evaluation. Specifically, Templates 1–6 are designated for GSM8K, and Templates 7–8 for MATH500, and the results in Table 1 and Table 2 are obtained using these templates. All templates are sourced from lm-evaluation-harness, except for Template 6, which is newly designed in this work. Example D.2 presents the prompt templates employed in code benchmark evaluation. Specifically, Templates 1–3 are designated for HumanEval, and Templates 4–6 for MBPP, and the results in Table 3 and Table 4

Table 12: The accuracy of vanilla and current decoding methods on GSM8K under LLaDA-8B-Instruct, across different few-shot settings. *Vanilla* denotes the low-confidence sampling baseline. Each numerical subscript indicates the improvement or degradation compared to the vanilla decoding method. Cells highlighted in yellow indicate the best result across few-shot settings for each method.

Method	GSM8K					Avg.
	0-shot	1-shot	2-shot	3-shot	4-shot	
Vanilla	71.72	76.04	**76.72**	76.35	75.44	75.25
Parallel decoding methods						
Fast-dLLM	$70.51_{-1.21}$	$73.77_{-2.27}$	$76.04_{-0.68}$	$75.74_{-0.61}$	$\underline{76.35}_{+0.91}$	$74.48_{-0.77}$
AdaBlock-dLLM	$71.95_{+0.23}$	$74.60_{-1.44}$	$75.82_{-0.91}$	$75.74_{-0.61}$	$76.12_{+0.68}$	$74.85_{-0.40}$
AdaBlock-Fast	$71.27_{-0.45}$	$74.22_{-1.82}$	$75.28_{-1.44}$	$75.66_{-0.68}$	$75.59_{+0.15}$	$74.40_{-0.85}$
dKV-Cache-Decode	$70.28_{-1.44}$	$70.74_{-5.31}$	$72.56_{-4.17}$	$72.78_{-3.56}$	$72.10_{-3.34}$	$71.69_{-3.56}$
Elastic-Cache	$72.18_{+0.45}$	$75.97_{-0.08}$	$76.12_{-0.61}$	$75.97_{-0.38}$	**$76.65_{+1.21}$**	$75.38_{+0.14}$
WINO	$70.74_{-0.98}$	$73.31_{-2.73}$	$74.75_{-1.97}$	$74.45_{-1.90}$	$75.13_{-0.31}$	$73.68_{-1.57}$

Table 13: Accuracy discrepancies across GPU types under identical decoding methods and prompt templates. $|\Delta|$ denotes the absolute accuracy deviation of the other GPU types relative to L40. Avg. and Max. denote the mean and maximum $|\Delta|$ across all prompt templates, respectively.

GPU	Template								Avg.	Max.
	A	B	C	D	E	F	G	H		
Single-token decoding										
L40	77.26	77.63	77.10	78.17	77.26	77.86	77.94	77.86	–	–
Pro6000	77.56	77.03	77.10	77.41	78.01	77.86	78.09	77.79	–	–
$\|\Delta\|$	0.30	0.61	0.00	0.76	0.76	0.00	0.15	0.08	0.33	0.76
RTX 4090	77.41	77.48	76.65	77.94	77.48	78.32	77.41	77.71		
$\|\Delta\|$	0.15	0.15	0.45	0.23	0.22	0.46	0.53	0.15	0.29	0.53
A100	77.10	77.10	76.72	77.79	76.42	77.56	78.01	78.62		
$\|\Delta\|$	0.16	0.53	0.38	0.38	0.84	0.30	0.07	0.76	0.43	0.84
Threshold-based parallel decoding (γ=0.8)										
L40	76.19	76.19	76.42	77.63	76.27	77.48	77.33	76.65	–	–
Pro6000	76.27	76.72	76.19	77.94	76.95	77.10	76.80	77.03	–	–
$\|\Delta\|$	0.08	0.53	0.23	0.31	0.68	0.38	0.53	0.38	0.39	0.68
RTX 4090	76.35	76.50	76.12	77.41	76.95	77.71	76.35	76.72	–	–
$\|\Delta\|$	0.16	0.31	0.30	0.22	0.68	0.23	0.98	0.07	0.37	0.98
A100	76.27	76.27	75.82	77.94	76.27	77.03	76.72	77.71		
$\|\Delta\|$	0.08	0.08	0.60	0.31	0.00	0.45	0.61	1.06	0.40	1.06

are obtained using these templates. All templates are adapted from lm-evaluation-harness, with minor modifications to enhance template diversity.

D.2 Near-identical prompt templates

Example D.3 presents a set of near-identical prompt templates constructed for the GSM8K benchmark evaluation, where each template differs from the others only in individual words or characters. For clarity, characters that differ from the first prompt are highlighted in red. Table 6 reports the accuracy of different parallel decoding strategies across this set of prompt templates.

D.3 Semantically equivalent prompt templates

Example D.4 presents a set of semantically equivalent prompt templates constructed for the GSM8K benchmark evaluation, where each template maintains identical semantics while varying in lexical

Table 14: Accuracy discrepancies across GPU types under identical decoding methods and prompt templates, evaluated in float32 precision. $|\Delta|$ denotes the absolute accuracy deviation of A100 relative to L40. Avg. and Max. denote the mean and maximum $|\Delta|$ across all prompt templates, respectively.

GPU	Template								Avg.	Max.
	A	B	C	D	E	F	G	H		
L40	76.65	76.57	76.42	76.72	77.33	77.18	77.03	77.26	–	–
A100	76.65	76.57	76.42	76.72	77.33	77.18	77.03	77.26		
$\|\Delta\|$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

and stylistic expression. Table 11 reports the accuracy and number of decoding steps of each parallel decoding strategy across this set of prompt templates.

D.4 Chain of thought prompt templates

Example D.5 presents a set of prompt templates for the GSM8K benchmark, where the prompts have identical task instructions and output format requirements, differing only in the chain-of-thought elicitation phrase, ranging from no explicit reasoning instruction to increasingly detailed step-by-step guidance.

Example D.1: Prompt templates for GSM8K and Math500 benchmarks

Template 1:
```
Q: {{question}}
A:
```

Template 2:
```
Q: {{question}}
A: Let's think step by step.
```

Template 3:
```
Question: {{question}}
Answer:
```

Template 4:
```
Question: {{fewshot_question_1}}
Answer: {{fewshot_answer_1}}

Question: {{fewshot_question_2}}
Answer: {{fewshot_answer_2}}

Question: {{fewshot_question_3}}
Answer: {{fewshot_answer_3}}

Question: {{question}}
Answer:
```

Template 5:
```
Question: {{fewshot_question_1}}
Answer: {{fewshot_answer_1}}

Question: {{fewshot_question_2}}
Answer: {{fewshot_answer_2}}

Question: {{fewshot_question_3}}
Answer: {{fewshot_answer_3}}
```

```
Question: {{fewshot_question_4}}
Answer:{{fewshot_answer_4}}

Question: {{fewshot_question_5}}
Answer: {{fewshot_answer_5}}

Question: {{question}}
Answer:
```

Template 6:

```
Solve the following math problem step by step.
IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly: The answer is <NUMBER>
- Output ONLY the number on the final line
  (no '$', no commas, no units, no extra words).
- Put the final line on its own line.
Problem:
{{question}}
```

Template 7:

```
Problem: {{question}}

Solution:
```

Template 8:

```
Problem: {{fewshot_question_1}}

Solution: {{fewshot_answer_1}}

Problem: {{fewshot_question_2}}

Solution: {{fewshot_answer_2}}

Problem: {{fewshot_question_3}}

Solution: {{fewshot_answer_3}}

Problem: {{fewshot_question_4}}

Solution: {{fewshot_answer_4}}

Problem: {{question}}

Solution:
```

Example D.2: Prompt templates for HumanEval and MBPP benchmarks

Prompt 1:

```
{{Question}}
```

Prompt 2:

```
You are a Python coding assistant. Complete the function.
{{Question}}
```

Prompt 3:

```
You are a Python coding assistant. Follow the docstring and
write the correct implementation.
```

```
{{Question}}
```

Prompt 4:

```
You are an expert Python programmer, and here is your task:
{{Question}} Your code should pass these tests:
{{test_list[0]}}
{{test_list[1]}}
{{test_list[2]}}
[BEGIN]
```

Prompt 5:

```
You are an expert Python programmer.  Read the programming
problem below and produce a correct Python solution:
{{Question}} Your code should pass these tests:
{{test_list[0]}}
{{test_list[1]}}
{{test_list[2]}}
[BEGIN]
```

Prompt 6:

```
You are an expert Python programmer.  Your goal is to
implement a Python function that satisfies the specification
and tests:  {{Question}} Your code should pass these tests:
{{test_list[0]}}
{{test_list[1]}}
{{test_list[2]}}
[BEGIN]
```

Example D.3: Near-identical prompt templates

Prompt A:

```
Solve the following math problem step by step.

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly:  The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
- Put the final line on its own line.

Problem:  {{question}}
```

Prompt B:

```
'Solve the following math problem step by step.

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly:  The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
- Put the final line on its own line.

Problem:  {{question}}
```

Prompt C:

```
' Solve the following math problem step by step.

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly:  The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
```

```
- Put the final line on its own line.

Problem: {{question}}
```

Prompt D:

```
** Solve the following math problem step by step

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly: The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
- Put the final line on its own line.

Problem: {{question}}
```

Prompt E:

```
$ Solve the following math problem step by step

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly: The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
- Put the final line on its own line.

Problem: {{question}}
```

Prompt F:

```
?? Solve the following math problem step by step

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly: The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
- Put the final line on its own line.

Problem: {{question}}
```

Prompt G:

```
please solve the following math problem step by step

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly: The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
- Put the final line on its own line.

Problem: {{question}}
```

Prompt H:

```
let's solve the following math problem step by step

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly: The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
- Put the final line on its own line.

Problem: {{question}}
```

Example D.4: Illustrative case of prompts

Prompt A:

```
Solve the following math problem step by step.

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly: The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
- Put the final line on its own line.

Problem:
{{question}}
```

Prompt B:

```
Problem: {{question}}

Solve it step by step. Last line must be exactly: The
answer is <NUMBER>
```

Prompt C:

```
You are an expert mathematician. A student has asked you the
following question:

{{question}}

Please solve this carefully, showing each step of your
reasoning.
At the end, state your final answer in the format: The
answer is <NUMBER>
```

Prompt D:

```
Think step by step to solve the following problem.

Problem: {{question}}

Work through the problem carefully. After your reasoning,
your final answer must follow this exact format:
The answer is <NUMBER>
```

Prompt E:

```
Answer the math problem below.

{{question}}

End your response with: The answer is <NUMBER>
```

Prompt F:

```
The following is a mathematical problem that requires a
step-by-step solution.
Analyze the problem systematically and derive the answer
through logical reasoning.

Problem Statement: {{question}}

Required output format for the final line: The answer is
<NUMBER>
```

Prompt G:

```
Hey, can you help me solve this math problem?

{{question}}
```

```
Walk me through how you'd solve it, and make sure your last
line says exactly:  The answer is <NUMBER>
```

Prompt H:

```
Solve the math problem below.  Show your work.

{{question}}

WARNING: Your response MUST end with the following line and
nothing after it:
The answer is <NUMBER>
Do NOT include units, symbols, or extra text in the final
line.
```

Example D.5: CoT prompt templates

Prompt A:

```
Solve the following math problem.

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly:  The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
- Put the final line on its own line.

Problem:
{{question}}
```

Prompt B:

```
Solve the following math problem.

Let's think step by step.

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly:  The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
- Put the final line on its own line.

Problem:
{{question}}
```

Prompt C:

```
Solve the following math problem.

Reason step by step before giving the final answer.

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly:  The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
- Put the final line on its own line.

Problem:
{{question}}
```

Prompt D:

```
Solve the following math problem.
```

```
Work through the solution carefully step by step before
giving the final answer.

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly:  The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
- Put the final line on its own line.

Problem:
{{question}}
```

Prompt E:

```
Solve the following math problem.

Show your reasoning and work through the solution very
carefully step by step before giving the final answer.

IMPORTANT OUTPUT FORMAT:
- Your final line must be exactly:  The answer is <NUMBER>
- Output ONLY the number on the final line (no $, no commas,
no units, no extra words).
- Put the final line on its own line.

Problem:
{{question}}
```

Example 1: An illustration of prompt-induced trajectory divergence

Prompt A:
Solve the following math problem step by step.
IMPORTANT OUTPUT FORMAT:

- Your final line must be exactly: `The answer is <NUMBER>`
- Output ONLY the number on the final line (no '$', no commas, no units, no extra words).
- Put the final line on its own line.

Problem:
Carla is downloading a 200 GB file. Normally she can download 2 GB/minute, but 40% of the way through the download, Windows forces a restart to install updates, which takes 20 minutes. Then Carla has to restart the download from the beginning. How long does it take to download the file?

Response:
1. Calculate the time taken to download the first 40% of the file: 80 GB / 2 GB/minute = 40 minutes.
2. Calculate the time taken to download the remaining 60% of the file: 120 GB / 2 GB/minute = 60 minutes.
3. Add the time for the restart: 20 minutes.
4. Sum the times for all the steps: 40 minutes + 60 minutes + 20 minutes = 120 minutes.
The answer is 120

Prompt B:
** Solve the following math problem step by step
IMPORTANT OUTPUT FORMAT:

- Your final line must be exactly: `The answer is <NUMBER>`
- Output ONLY the number on the final line (no '$', no commas, no units, no extra words).
- Put the final line on its own line.

Problem:
Carla is downloading a 200 GB file. Normally she can download 2 GB/minute, but 40% of the way through the download, Windows forces a restart to install updates, which takes 20 minutes. Then Carla has to restart the download from the beginning. How long does it take to download the file?

Response:
1. Calculate the time taken to download the first 40% of the file: 40% of 200 GB / 2 GB/minute = 80 minutes.
2. Calculate the time taken to download the remaining 60% of the file: 60% of 200 GB / 2 GB/minute = 60 minutes.
3. Add the time for the restart and the initial download: 80 minutes + 60 minutes + 20 minutes = 160 minutes.
The answer is 160.