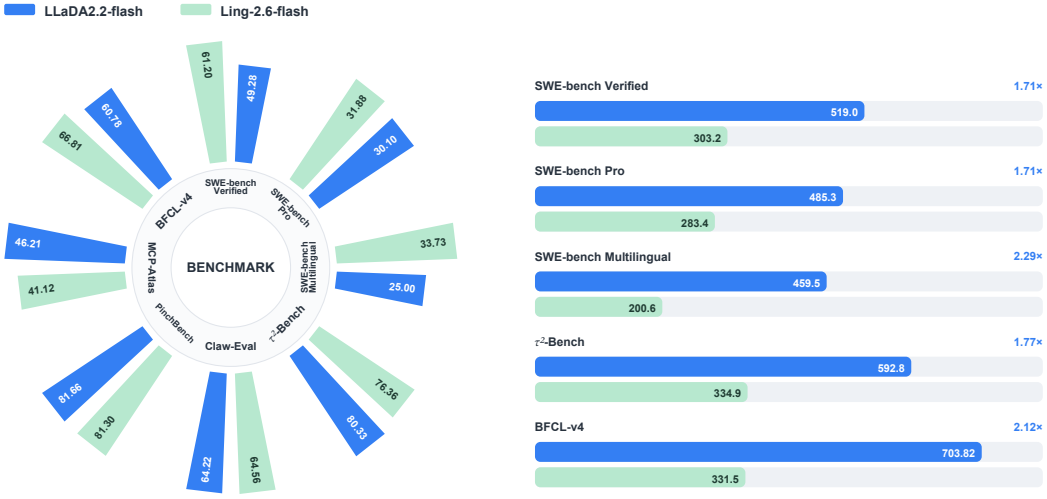
LLaDA2.2: Enabling Agentic Diffusion Language Models via Levenshtein Editing

Tiwei Bie[1], Jiansheng Cai[2], Maosong Cao[1], Xiang Cao[1], Binbin Chen[1], Fuyuan Chen[1], Kun Chen[1], Yuqi Ding[1], Lun Du[1], Zhuocheng Gong[1], Yanmei Gu[1], Kaiyuan Guan[1], Xiaoxu Guo[1], Kairong Han[1], Zenan Huang[1], Zijin Huang[1], Zhenzhong Lan[1,4,5†], Chengxi Li[1], Jianguo Li[1,†], Zhe Li[2], Ruiqi Liang[1], Xinyao Lin[1], Huabin Liu[1], Lin Liu[1], Xinyi Liu[1], Xuejie Liu[1], Guoshan Lu[1], Yuan Lu[1], Yuxin Ma[1], Chenhui Mao[1], Xiaojing Qi[1], Kaida Qiu[1], Xiaoyu Shi[2], Yiding Tian[1], Huanyu Wang[1], Qi Wang[1], Rui Wang[1], Shuaidi Wang[1], Lanning Wei[1], Tao Wu[1], Siyang Xiao[1], Yipeng Xing[1], Chenkai Xu[1], Song Yan[1], Ying Yan[2], Jingyi Yang[1], Yichun Yang[1], Liangyu Zha[1], Tianze Zhang[1], Yifan Zhang[2], Junbo Zhao[1,3,†], Wenqian Zhao[1], Da Zheng[1,†], Jianyuan Zhong[1], Jiahao Zhou[1], Junlin Zhou[1], Tianyu Zhou[1], Liwang Zhu[1], Yihong Zhuang[1]

[1]Inclusion AI, [2]Ant Digital Technologies, [3]Zhejiang University, [4]Westlake University, [5]Westlake Scitrain

Abstract

Diffusion language models (dLLMs) demonstrate strong performance and high efficiency across general tasks, yet their block-parallel decoding process makes them susceptible to error accumulation in multi-turn, long-horizon agentic settings. LLaDA2.1 partially mitigates this via token-to-token (T2T) editing, but its fixed-length substitution mechanism remains a critical bottleneck in agentic workflows. To address this, we present LLaDA2.2, which equips dLLMs with flexible Levenshtein editing through four primitive edit operations—KEEP, SUBSTITUTE, DELETE, and INSERT—with training labels derived via longest common subsequence (LCS) alignment between intermediate drafts and ground-truth sequences. We further propose L-EBPO, an agentic RL algorithm that optimizes editing decisions based on environmental feedback. For practical long-horizon deployment, LLaDA2.2 extends the context window to 128K tokens and introduces a block-routing mechanism to mitigate MoE inference overhead. Empirical results demonstrate that LLaDA2.2 achieves performance competitive with autoregressive baselines on long-horizon agentic benchmarks.



(a) Benchmark performance (b) Throughput (TPS)

Figure 1: Agent-Related benchmark results for **LLaDA2.2-flash** compared to **Ling-2.6-flash**.

Authors are listed alphabetically by last name. † indicates technical leads.

1 Introduction

Diffusion language models (dLLMs) have emerged as a promising paradigm in natural language processing (Austin et al., 2021; Sahoo et al., 2024; Lou et al., 2024; Nie et al., 2025). By leveraging block-parallel denoising (Arriola et al., 2025), dLLMs can alleviate the sequential latency bottleneck of token-by-token generation in traditional autoregressive (AR) models. Our prior work, LLaDA2.1 (Ant Group, 2026), introduced a token-level editing mechanism that mitigates the trade-off between generation speed and quality in non-agentic tasks. However, when applications shift from single-turn static generation to multi-turn dynamic agent environments (Yang et al., 2024; Yao et al., 2024; Ye et al., 2026; Wei et al., 2025), the nature of the problem changes fundamentally. Whereas conventional text generation tasks focus on the quality of independent single-turn outputs, an autonomous agent follows a continuous, closed-loop interaction trajectory. It must track long interaction histories, formulate multi-step plans, invoke external tools (Yao et al., 2022; Schick et al., 2023; Patil et al., 2023; Qin et al., 2023), process environmental feedback, and correct its own behavior. In long-horizon reasoning, even a small local deviation in early decoding blocks can propagate across subsequent interactions, creating structural challenges for stable parallel denoising.
To dissect the root causes of these failure modes, we analyze dLLMs within complex tool-use environments and highlight two critical, unaddressed challenges:

Challenge I: Structural Rigidity and Reasoning-Path Collapse. In block-parallel generation, tokens within the same block are decoded without explicit sequential conditioning, often resulting in n-gram repetitions, malformed JSON, and ambiguous tool-call boundaries. These errors are difficult to repair because LLaDA2.1's one-to-one substitution fixes the output length and prevents token insertion or deletion. Beyond this structural rigidity, arbitrary-order denoising can also reduce reasoning diversity: by postponing high-uncertainty forking tokens while resolving easier positions first, it allows the surrounding context to prematurely determine critical decisions, thereby pruning alternative reasoning paths and inducing solution-space collapse (Ni et al., 2026).

Challenge II: Error Accumulation Destabilizes Long-Horizon Agentic Trajectories. In long-horizon, multi-turn interactions, erroneous outputs from early generation steps are incorporated into context as hard constraints for subsequent denoising blocks. Unlike conventional exposure bias, where training–inference mismatches degrade output quality, these errors directly corrupt the task state itself, causing goal drift and cascading failures throughout the trajectory. Compared to autoregressive models, which can implicitly condition on prior context and self-correct upon encountering anomalies, block diffusion models are structurally more vulnerable to such error accumulation. Recent empirical analysis confirms that this brittleness renders current dLLMs highly unreliable in agentic workflows requiring long-horizon causal planning (Lu et al., 2026).

To address these challenges, we present LLaDA2.2, which extends dLLMs beyond rigid token-level substitution to support flexible Levenshtein editing. The system is built around three interconnected components:

Efficient Infrastructure for MoE Diffusion Models. To ground these capabilities in real-world, large-scale agent scenarios, we extend the model's context window to 128K via progressive long-context continual pre-training. Crucially, we introduce a block routing mechanism that precisely regulates the computation of the Mixture-of-Experts (MoE) block-diffusion architecture (Fedus et al., 2022; Dai et al., 2024), effectively addressing the painful bottleneck of skyrocketing inference costs in long-context agent applications.

Levenshtein Editing Paradigm. To tackle Challenge I, we move beyond the fixed-length token substitution constraints of traditional dLLMs by introducing `DELETE` and `INSERT` edit-control tokens. Prior edit-based sequence generation work has shown that insertion and deletion operations can enable dynamic length changes (Stern et al., 2019; Gu et al., 2019); LLaDA2.2 integrates Levenshtein editing into the dLLM denoising process. By optimally aligning the model's intermediate drafts with the ground truth during training, LLaDA2.2 dynamically derives Levenshtein editing labels. This empowers the diffusion process to alter sequence lengths and manipulate sequence positions during parallel decoding, removing redundant spans through deletion and creating new positions through insertion.

Environment-Aware Agentic Reinforcement Learning System. To mitigate the long-horizon error accumulation identified in Challenge II, we model Levenshtein-based error-correction decisions in multi-turn interactions as an interactive reinforcement learning process. Building on recent progress in reinforcement learning for dLLMs (Zhao et al., 2026; Ou et al., 2025; Rojas et al., 2025), we propose Levenshtein Editing Evidence Lower Bound (ELBO)-based Block-level Policy Optimization (L-EBPO), which combines LCS-derived edit labels with environmental reward signals. L-EBPO trains the agent to make robust error-correction decisions in uncertain tool-use scenarios, thereby blocking error accumulation and preventing goal drift.

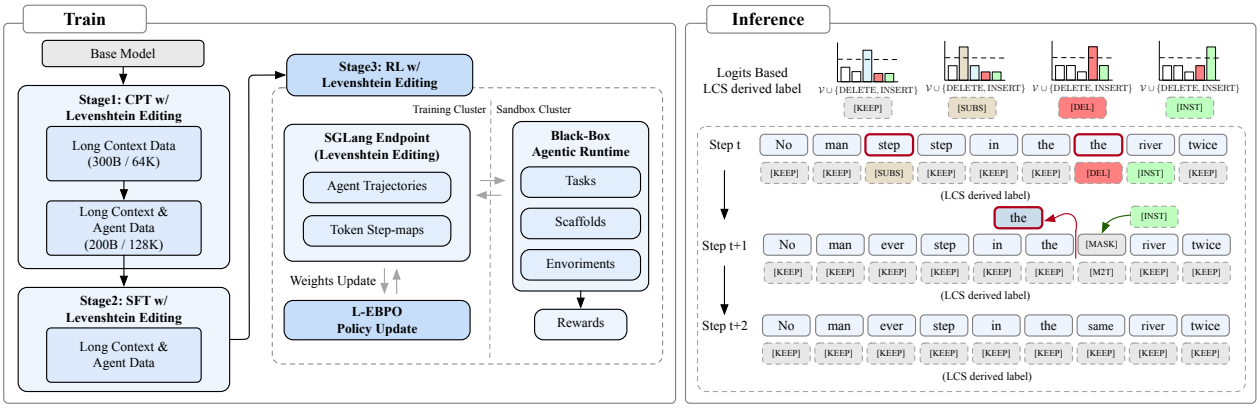


Figure 2: Overview of the training & inference framework for LLaDA2.2.

Across 7 agentic benchmarks (Jimenez et al., 2024; Deng et al., 2025; Zan et al., 2025; Yao et al., 2024; Ye et al., 2026; Bandi et al., 2026), LLaDA2.2-flash achieves an average score of 53.83, compared with 55.74 for the autoregressive Ling-2.6-flash baseline, and outperforms it on τ^2-Bench, PinchBench, and MCP-Atlas. Its average BF16 decoding throughput across 11 workloads is 1.64× that of Ling-2.6-flash. These results demonstrate a competitive balance between agentic performance and efficiency while also exposing limitations in long-horizon repository repair, strict function calling, instruction following, and knowledge-intensive reasoning.

2 Continued Pre-training

LLaDA2.2 is initialized from the LLaDA2.1 (Ant Group, 2026) base model and undergoes a dedicated continued pre-training (CPT) stage to lay the foundation for long-context agentic capabilities. This stage optimizes the base model's core representation and fundamental text-processing abilities, preparing it for subsequent instruction alignment and policy learning. The CPT stage serves two primary purposes: first, context extension, expanding the native context window of the LLaDA2.1 base model from 8K to 128K tokens; second, routing transition, switching the MoE routing mechanism from token-level routing to block-level routing.

2.1 Progressive Long-Context Extension

LLaDA2.2 extends the context window through a progressive long-context training schedule. Starting from the LLaDA2.1 8K base model, we first continue training at a 64K context length on 300B tokens, and then further extend the model to a 128K context length on another 200B tokens. This staged schedule avoids an abrupt transition from short to very long contexts, allowing the model to gradually adapt its positional representations, attention patterns, and block-diffusion denoising behavior.

The long-context data mixture is dominated by long-form documents and repository-level code data, especially examples whose effective lengths fall in the 64K–128K range. We retain document-aware packing and attention boundaries so that unrelated documents do not introduce spurious bidirectional dependencies during diffusion training. In the final 128K stage, we additionally introduce agentic data, including long software-engineering contexts, tool-use traces, and browsing trajectories. This places agentic supervision after the model has already acquired stable long-context behavior, making the final stage primarily responsible for adapting long-context capacity to interactive agent workflows.

Throughout both stages, the training objective remains compatible with block diffusion: the model denoises corrupted blocks while conditioning on valid preceding clean blocks and in-block noisy context. As a result, the final LLaDA2.2 model supports native 128K context without relying on inference-time context extrapolation alone.

2.2 Block Routing for MoE Block Diffusion

In a standard MoE transformer, each token independently selects its top-k experts. This is natural for autoregressive decoding, where each step usually processes a small number of new tokens. In block diffusion decoding, however, each step processes an entire block, commonly 32 or 64 tokens. If every token independently routes to top-k experts, the expert set touched by one block becomes the union of all

token-level choices. The active expert footprint can therefore grow from k experts in the AR case to as many as $\min(Bk, E)$ experts for block size B and total experts E.

This creates a systems bottleneck for MoE-dLLMs. A larger and input-dependent expert set increases HBM traffic, reduces expert weight reuse, and raises dispatch/combine communication cost under expert parallelism. The issue becomes more visible in long-context inference and limited-device deployment, where expert working-set predictability directly affects throughput, latency, and time to first token.

This design is motivated by an empirical observation on LLaDA2-series MoE dLLMs: although token-level routing appears dispersed, expert preferences within a block are highly concentrated, with a small fraction of experts accounting for most routing traffic. This makes a fixed-capacity block-level expert pool a natural fit for block diffusion inference.

LLaDA2.2 adopts **block routing** to align the routing unit with the diffusion generation unit. Instead of allowing each token to freely expand the expert set, block routing first performs expert admission at the block level, then applies token-wise routing inside the admitted expert pool.

Let $s_i \in \mathbb{R}^E$ be the routing score vector for token i in a block of size B. We compute a block-level admission score by the **token racing** strategy:

$$g_j = \max_{i \in \{1,\ldots,B\}} s_{i,j}, \quad j = 1, \ldots, E. \tag{1}$$

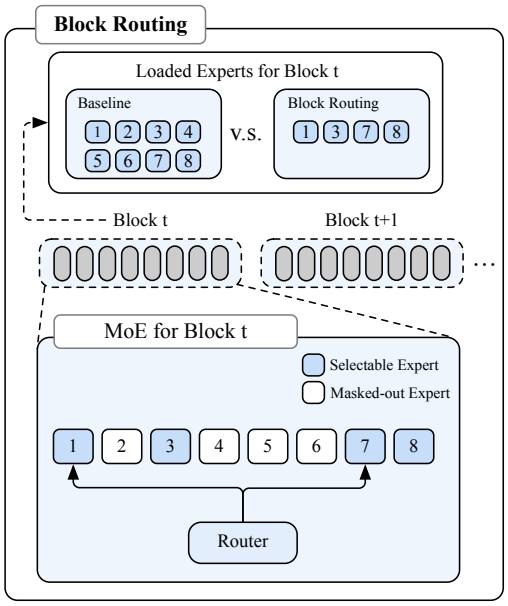


Figure 3: Illustration of Block Routing in a block-diffusion MoE.

This rule gives each token a chance to nominate experts with strong local preference. The block then admits the top-C experts according to g, forming a fixed-capacity expert pool $\mathcal{P}$. In our default setting, $C = 48$ for $E = 256$ routed experts. Token-wise top-k routing is then performed only inside $\mathcal{P}$, while scores outside the pool are masked.

Block routing gives every block a predictable expert working-set upper bound $O(C)$, which improves memory planning and expert-parallel execution. At the same time, because token-wise top-k selection is still performed inside the admitted pool, the model preserves token-level specialization rather than collapsing routing into a single block-level decision. In our training recipe, switching to block routing introduces little to no degradation in model quality after continued pre-training, while providing a more stable and efficient routing structure for block diffusion inference.

3 Post-training

LLaDA2.2 follows the overall post-training pipeline of LLaDA2.1, including diffusion-specific instruction tuning and policy optimization. We therefore focus this section on the main change introduced for agentic settings: extending the editable denoising process from fixed-length token substitution to Levenshtein editing within each diffusion block. This extension is motivated by failure modes that are difficult to repair with token-to-token (T2T) editing alone, such as repeated n-grams, missing tool arguments, incomplete code edits, and tool calls whose boundaries have been mixed together.

3.1 Supervised Fine-Tuning with Levenshtein Editing

Unlike T2T correction, which is restricted to position-wise *keep* and *substitute* operations, LLaDA2.2 supports four operations: *keep*, *substitute*, *delete*, and *insert*. `DELETE` and `INSERT` serve as two additional edit-control tokens. At position i, `DELETE` removes the current token x_i, whereas `INSERT` creates a new editable position immediately before x_i.

LCS-based label construction. Let x and y denote the valid draft and target token sequences in a block, with lengths m and n, respectively. In our block implementation, $m = n \leq B$. We compute their LCS and use its matches as anchors, obtaining operation labels

$$\mathbf{a} = (a_1, \ldots, a_m) \in \{\mathrm{K}, \mathrm{S}, \mathrm{D}, \mathrm{I}\}^m. \tag{2}$$

Except when used as an insertion site, a matched anchor yields a keep label. For each gap preceding an anchor, aligned positions yield substitute labels, surplus draft positions yield delete labels, and a target-surplus gap assigns one insert label to the following anchor. At most one insert label is assigned to such a gap in one editing round; additional missing positions are created over subsequent rounds. All draft positions after the final anchor are labeled delete; if no anchor exists, all positions are labeled delete. The resulting supervision label is

$$\ell_i = \begin{cases} y_{\sigma(i)} & \text{if } a_i \in \{\mathrm{K}, \mathrm{S}\}, \\ \texttt{DELETE} & \text{if } a_i = \mathrm{D}, \\ \texttt{INSERT} & \text{if } a_i = \mathrm{I}. \end{cases} \tag{3}$$

where $\sigma(i)$ is the target index assigned to a keep or substitute operation. During early editing stages, `[MASK]` may match any target token in the LCS. For masked positions selected during backtracking, the matched target index is adjusted to the nearest feasible position before its target-token label is assigned. The first training round uses ordinary target-token supervision, and these LCS-derived labels are recomputed from the updated draft in subsequent editing rounds.

Fixed-length block constraint. Edits are applied independently to the m valid positions in each block. Delete removes x_i and left-shifts subsequent tokens, whereas insert expands x_i into $(\texttt{[MASK]}, x_i)$ so that a later denoising round can fill the new slot. The result is padded with `[MASK]` tokens or truncated at the tail to restore m positions, keeping the block length fixed.

Effectiveness of Levenshtein Editing. We isolate the effect of Levenshtein editing by fine-tuning the same LLaDA2.2 base model on identical SWE trajectories, varying only whether Levenshtein editing is enabled. As shown in Figure 4, enabling Levenshtein editing improves SWE-bench Verified performance from 35.8 to 44.4, an absolute gain of 8.6 points. This result demonstrates that allowing insertions and deletions substantially improves performance on repository-level software engineering tasks.

3.2 Agentic Reinforcement Learning Training

The agentic RL post-training of LLaDA2.2 shares both the objectives and requirements of autoregressive agentic RL training, encompassing diverse cross-domain tasks that involve complex tool use, multi-turn environment interactions, and substantial demands on system resources and inference infrastructure. Notably, Levenshtein editing must also remain compatible with training across agentic trajectories.

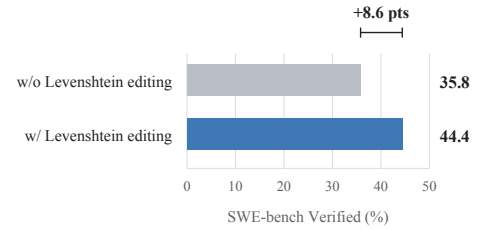


Figure 4: Effect of Levenshtein editing on SWE-bench Verified. Enabling editing improves the resolution rate from 35.8 to 44.4.

Training Paradigm To address this, we formulate the agentic RL problem as a two-level control hierarchy. The outer level, governed by EBPO, optimizes trajectory-level decisions across agent interaction rounds. The inner level manages intra-block splice editing—when and where to apply `DELETE` and `INSERT` within a single generation step. We propose L-EBPO, which unifies these two levels through a single extended action space:

$$\mathcal{A} = \mathcal{V} \cup \{\texttt{DELETE}, \texttt{INSERT}\}. \tag{4}$$

This action space enables both token prediction and Levenshtein editing decisions to be optimized under a common objective. The adapted log-ratio takes the following form:

$$\log \rho(y|x) \approx \sum_{n=1}^{N} w_n \sum_{b=1}^{B} \left(\log p_\theta(a_b \mid z_n, x; M) - \log p_{\theta_{old}}(a_b \mid z_n, x; M) \right), \quad a_b \in \mathcal{A}^{L_B}, \tag{5}$$

where a_b denotes the action vector for block b, encompassing both token predictions and Levenshtein editing decisions. Since `DELETE` and `INSERT` are transient—present during decoding but absent from the output sequence—the edit trace must be recovered for likelihood estimation. Following the Levenshtein editing algorithm introduced in Section 3.1, we derive the optimal edit labels at each noise level by aligning the noised rollout with the original action sequence, thereby enabling gradient signals to flow through structural

decisions. Attention masking for multi-turn agentic trajectories differs slightly from that for simple responses. Within a diffusion block, if there are multiple turns containing model outputs and tool-call responses, all turns except the first should be masked because of potential information leakage from intra-block full attention.

Agentic Training Environment For agentic tasks, we construct the RL training pipeline atop a collection of diverse task harnesses and isolated execution environments. Training and rollout workers are jointly orchestrated through ASystem (Ling Team et al., 2025), a unified reinforcement learning system designed for large-scale post-training. Rollout generation is handled by SGLang (Ant Group Team & SGLang Team), and execution environments are provisioned via a large-scale sandbox cluster managed by AKernel (Ant Group Team), which provides the controlled, reproducible execution contexts required by each task harness.

Reward signals are derived entirely from environmental feedback collected during agentic rollouts, and comprise three additive components: tool-call execution correctness, output format validity, and overall task completion. To reduce training wall-clock time, we adopt an asynchronous training scheme in which rollout collection, sandbox serving, environment execution, and policy optimization proceed concurrently across distributed workers.

Through these designs, the policy learns not only the syntactic conventions of Levenshtein editing established during supervised alignment, but also the strategic decision-making required to maximize marginal value within long-horizon agentic trajectories.

4 Evaluation

4.1 Benchmarks and Evaluation Protocol

Benchmark suite. Our suite comprises 17 benchmarks, split into 7 agentic and 10 general benchmarks, emphasizing agentic capabilities while maintaining broad general coverage.

Agentic benchmarks. The agentic subset includes three repository-level software engineering benchmarks: SWE-bench Verified, SWE-bench Pro, and SWE-bench Multilingual (Jimenez et al., 2024; Deng et al., 2025; Zan et al., 2025). It also features τ^2-Bench (Yao et al., 2024) for multi-turn tool–agent–user interaction, Claw-Eval (Ye et al., 2026) for autonomous agency, and PinchBench (AI, 2026) and MCP-Atlas (Bandi et al., 2026) for interactive tool use.

General benchmarks. The 10 general benchmarks cover core domains: mathematics (AIME 2026 (Art of Problem Solving Wiki, 2026), OlympiadBench (He et al., 2024a)), coding (LiveCodeBench (Jain et al., 2024)), instruction following (IFBench (Pyatkin et al., 2025), Multi-IF (He et al., 2024b)), reasoning (KOR-Bench (Ma et al., 2024)), scientific Q&A (GPQA-Diamond (Rein et al., 2024)), long-context understanding (LongBench v2 (Bai et al., 2024)), and function calling (BFCL v3/v4 (Patil et al., 2025)).

Evaluation protocol. We evaluate task performance and decoding efficiency. For performance, we compare LLaDA2.2-flash with Ling-2.6-flash across all 17 benchmarks, alongside LLaDA2.1-flash on 10 general benchmarks. For efficiency, we measure throughput across 11 representative workloads, comparing both BF16 and FP8-quantized variants of LLaDA2.2-flash against Ling-2.6-flash.

4.2 Overall Benchmark Performance

As shown in Table 1, LLaDA2.2-flash achieves an average score of 53.83 across 7 agentic benchmarks, 1.91 points below Ling-2.6-flash (55.74). It outperforms Ling-2.6-flash on τ^2-Bench, PinchBench, and MCP-Atlas, and nearly matches it on Claw-Eval. Its main weaknesses lie in repository-level software engineering: LLaDA2.2-flash trails by 8.73 points on SWE-bench Multilingual and 11.92 points on SWE-bench Verified. The latter comparison warrants caution as the models used different evaluation scaffolds. Overall, LLaDA2.2-flash is competitive in interactive tool-use tasks, though long-horizon repository repair remains challenging.

Across the 10 general benchmarks in Table 2, LLaDA2.2-flash obtains an average score of 56.81, compared with 65.90 for Ling-2.6-flash. It surpasses Ling-2.6-flash on LongBench v2 (45.13 vs. 42.94) and remains close on Multi-IF (73.67 vs. 74.80). The LongBench v2 result is consistent with the goal of extending the context window to 128K tokens. Larger gaps remain on BFCL v3 and v4, LiveCodeBench, IFBench, GPQA-Diamond, and AIME 2026, revealing weaknesses in structured function calling, code generation, instruction following,

and knowledge-intensive reasoning. These general capabilities therefore remain key priorities for future improvement.

Table 1: Task-performance scores on agentic benchmarks.

Benchmark	Ling-2.6-flash	LLaDA2.2-flash
Avg.	55.74	53.83
SWE-bench Verified	61.20*	49.28
SWE-bench Pro	31.88	30.10
SWE-bench Multilingual	33.73	25.00
τ^2-Bench	76.36	80.33
Claw-Eval	64.56	64.22
PinchBench	81.30	81.66
MCP-Atlas	41.12	46.21

Table 2: Task-performance scores on general benchmarks.

Category	Benchmark	Ling-2.6-flash	LLaDA2.1-flash	LLaDA2.2-flash
	Avg.	65.90	59.23	56.81
Function Calling	BFCL v4	66.81	41.04	60.78
	BFCL v3	76.05	75.61	67.17
Math	AIME 2026	73.65	74.01	62.24
	OlympiadBench	79.59	76.59	74.48
Coding	LiveCodeBench	60.96	45.37	44.77
Instruction Following	IFBench	57.40	37.07	30.20
	Multi-IF	74.80	76.41	73.67
Reasoning	KOR-Bench	66.48	65.12	60.96
Knowledge	GPQA-Diamond	60.35	67.30	48.67
Long Context	LongBench v2	42.94	33.80	45.13

Table 3: Throughput comparison between Ling-2.6-flash and LLaDA2.2-flash across different benchmarks. Throughput is measured in tokens per second (TPS).

Category	Benchmark	Ling-2.6-flash	LLaDA2.2-flash	
		BF16	BF16	w/ FP8 Quant
Agentic	SWE-bench Verified	303.20	519.00	601.50
	SWE-bench Pro	283.40	485.30	578.60
	SWE-bench Multilingual	200.60	459.50	535.70
	τ^2-Bench	334.90	592.80	705.30
Function Calling	BFCL-v4	331.50	703.82	846.60
Math	AIME 2026	351.24	514.10	588.50
Coding	LiveCodeBench	354.90	599.00	721.30
Instruction Following	IFBench	327.24	473.40	566.50
Reasoning	KOR-Bench	352.75	452.10	550.00
Knowledge	GPQA-Diamond	324.55	337.60	408.80
Long Context	LongBench-v2	145.50	281.20	320.80

For LLaDA2.2-flash, the SWE-bench series is evaluated with the Claude Code scaffold. Each task-performance score is the arithmetic mean of 5 runs.

The Ling-2.6-flash SWE-bench Verified score (*) is reported in Li et al. (2026) (evaluated via OpenHands), while the model's SWE-bench Pro and Multilingual scores were evaluated in our setup using the same Claude Code scaffold as LLaDA2.2-flash. For the decoding efficiency evaluation, multi-token prediction (MTP) is enabled with 4 draft tokens.

4.3 Inference Throughput

Table 3 reports the BF16 inference throughput of LLaDA2.2-flash and Ling-2.6-flash, with MTP enabled for the latter. Across the 11 evaluated workloads, LLaDA2.2-flash is consistently faster, delivering a 1.64× average speedup over Ling-2.6-flash. Quantizing LLaDA2.2-flash from BF16 to FP8 provides an additional 18.6% increase in average throughput.

5 Limitations & Outlook

Although our method marks a promising step toward developing agentic long-context block-diffusion language models, it remains at an early stage. Block routing enables more precise control over the number of activated experts during MoE block-diffusion inference, helping reduce system-level cost fluctuations in multi-turn agent scenarios. Furthermore, the Levenshtein editing mechanism provides the model with a local repair capability beyond fixed-length token-to-token replacement, while L-EBPO combines LCS-derived edit labels with environmental reward signals. Together, these mechanisms alleviate, to some extent, repetition, reasoning-path collapse, and error accumulation that destabilizes long-horizon agentic trajectories under block-parallel generation, though they do not fully resolve these issues.

Local Inconsistency in Block Diffusion Decoding. Block diffusion models still trail strong AR models in generating strictly structured outputs. Because multiple positions are predicted in parallel within each denoising step, they cannot condition on one another until subsequent refinement steps. This delayed dependency weakens local consistency in order-sensitive tasks such as nested JSON generation, complex SQL synthesis, and multi-argument tool calling, leading to errors such as mismatched delimiters, misaligned fields, and ambiguous boundaries. Levenshtein editing enables post-hoc repair, but it cannot fully recover the token-by-token prefix constraints of AR decoding or guarantee syntactic validity.

Asymmetric Edit Learning and Error Accumulation. Levenshtein editing directly addresses repetition and local structural defects, but its two core edit-control tokens are not equally learnable. `DELETE` often corresponds to locally visible redundancy, whereas `INSERT` requires the model to detect an omission, determine where new content is needed, and coordinate the inserted position with subsequent refinement steps. This asymmetry makes repetitions easier to remove than missing content or structural gaps to repair. More importantly, uncorrected errors enter the context and compound over time, leading to goal drift—local edits alone cannot eliminate long-horizon error accumulation.

RL Training Efficiency and MoE Train-Inference Discrepancy. The reinforcement learning stage also faces optimization difficulties that differ from those in AR models. Since a dLLM generation trajectory contains multiple rounds of block-level denoising and editing decisions, its exact sequence-level likelihood is difficult to compute efficiently. LLaDA2.1 therefore adopts EBPO, which uses an ELBO-based approximation as a tractable proxy for policy ratio estimation. L-EBPO follows the same approach and extends the EBPO framework to trajectories involving Levenshtein editing. However, throughout the RL training of the LLaDA2 series, several challenges remain to be addressed, including the training efficiency and optimization stability of EBPO, as well as two forms of train-inference discrepancy in MoE-based dLLM: the mismatch between ELBO-based likelihood estimation during training and confidence-based decoding during inference, and the discrepancy between expert routing behavior in MoE training and inference.

More broadly, the core training paradigm for dLLMs remains underexplored. Optimal strategies for data mixing and curriculum design between general and agentic corpora are yet to be established—while excessive agentic data risks degrading broad base capabilities, insufficient supervision constrains tool-use proficiency and long-horizon execution. Furthermore, extending techniques like Multi-Teacher On-Policy Distillation (MOPD) (Ma et al., 2026) to dLLMs presents an important yet unresolved challenge.

Outlook. Looking forward, several research frontiers offer significant promise. First, editing could integrate verify-while-decode objectives to continuously ensure consistency with previously generated tokens, aligning with the introspective modeling perspective of I-DLM (Yu et al., 2026). Second, continuous diffusion paradigms (Hu et al., 2026; Jo & Hwang, 2025) present a viable path beyond discrete mask-and-edit transitions. Third, online alignment warrants further exploration within block diffusion, particularly through lower-variance likelihood estimation, more targeted exploration, and finer-grained credit assignment. Together, these avenues may enhance the reliability and scalability of diffusion models in long-horizon interactions.

References

Kilo AI. Pinchbench: Real-world benchmarks for ai coding agents, 2026. URL https://github.com/pinchbench.

Ant Group. LLaDA2.1: Speeding up text diffusion via token editing, 2026. URL https://arxiv.org/abs/2602.08676.

Ant Group Team. Akernel: Programmable datacenter-scale infrastructure for agents. URL https://github.com/inclusionAI/AKernel.

Ant Group Team and SGLang Team. Power Up Diffusion LLMs: Day-0 Support for LLaDA 2.0 | LMSYS Org. URL https://lmsys.org/blog/2025-12-19-diffusion-llm.

Marianne Arriola, Aaron Gokaslan, Justin Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, volume 2025, pp. 50726–50753, 2025.

Art of Problem Solving Wiki. 2026 AIME I problems. https://artofproblemsolving.com/wiki/index.php/2026_AIME_I_Problems, 2026. Accessed: 2026-05-07.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. *arXiv preprint arXiv:2412.15204*, 2024.

Chaithanya Bandi, Razvan-Gabriel Dumitru, Ben Hertzberg, Divyansh Agarwal, Geobio Boo, Tejas Polakam, Sami Hassaan, Jeff Da, HiJae Kim, Vipul Gupta, Manasi Sharma, Andrew Park, Martin Dimakis, Ernesto Gabriel Hernandez Montoya, Dan Rambado, Ivan Salazar, Rafael Cruz, MohammadHossein Rezaei, Chetan Rane, Ben Levin, Daniel Yue Zhang, Brad Kenstler, and Bing Liu. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers, 2026. URL https://arxiv.org/abs/2602.00933.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1280–1297, 2024.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks?, 2025. URL https://arxiv.org/abs/2509.16941.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Jiatao Gu, Changhan Wang, and Junbo Zhao. Levenshtein transformer. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. OlympiadBench: A Challenging Benchmark for Promoting AGI with Olympiad-Level Bilingual Multimodal Scientific Problems. *arXiv preprint arXiv:2402.14008*, 2024a.

Yun He, Di Jin, Chaoqi Wang, Chloe Bi, Karishma Mandyam, Hejia Zhang, Chen Zhu, Ning Li, Tengyu Xu, Hongjiang Lv, et al. Multi-if: Benchmarking llms on multi-turn and multilingual instructions following. *arXiv preprint arXiv:2410.15553*, 2024b.

Keya Hu, Linlu Qiu, Yiyang Lu, Hanhong Zhao, Tianhong Li, Yoon Kim, Jacob Andreas, and Kaiming He. Elf: Embedded language flows, 2026. URL https://arxiv.org/abs/2605.10938.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Jaehyeong Jo and Sung Ju Hwang. Continuous diffusion model for language modeling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=VGv5y60sXC.

Ang Li, Ben Liu, and Bin Han etc. Ling and ring 2.6 technical report: Efficient and instant agentic intelligence at trillion-parameter scale, 2026. URL https://arxiv.org/abs/2606.15079.

Ling Team et al. Every step evolves: Scaling reinforcement learning for trillion-scale thinking model, 2025. URL https://arxiv.org/abs/2510.18855.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning*, 2024.

Qingyu Lu, Liang Ding, Kanjian Zhang, Jinxia Zhang, and Dacheng Tao. The bitter lesson of diffusion language models for agentic workflows, 2026. URL https://arxiv.org/abs/2601.12979.

Kaijing Ma, Xinrun Du, Yunran Wang, Haoran Zhang, Zhoufutu Wen, Xingwei Qu, Jian Yang, Jiaheng Liu, Minghao Liu, Xiang Yue, et al. Kor-bench: Benchmarking language models on knowledge-orthogonal reasoning tasks. *arXiv preprint arXiv:2410.06526*, 2024.

Wenhan Ma, Jianyu Wei, Liang Zhao, Hailin Zhang, Bangjun Xiao, Lei Li, Qibin Yang, Bofei Gao, Yudong Wang, Rang Li, Jinhao Dong, Zhifang Sui, and Fuli Luo. Mopd: Multi-teacher on-policy distillation for capability integration in llm post-training, 2026. URL https://arxiv.org/abs/2606.30406.

Zanlin Ni, Shenzhi Wang, Yang Yue, Tianyu Yu, Weilin Zhao, Yeguo Hua, Tianyi Chen, Jun Song, Cheng Yu, Bo Zheng, et al. The flexibility trap: Why arbitrary order limits reasoning potential in diffusion language models. *arXiv preprint arXiv:2601.15165*, 2026.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models, 2025. URL https://arxiv.org/abs/2502.09992.

Jingyang Ou, Jiaqi Han, Minkai Xu, Shaoxuan Xu, Jianwen Xie, Stefano Ermon, Yi Wu, and Chongxuan Li. Principled RL for Diffusion LLMs Emerges from a Sequence-Level Perspective, December 2025. URL http://arxiv.org/abs/2512.03759. arXiv:2512.03759 [cs].

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs, 2023. URL https://arxiv.org/abs/2305.15334.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs, 2023. URL https://arxiv.org/abs/2307.16789.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. GPQA: A Graduate-Level Google-Proof Q&Q Benchmark. In *First Conference on Language Modeling*, 2024.

Kevin Rojas, Jiahe Lin, Kashif Rasul, Anderson Schneider, Yuriy Nevmyvaka, Molei Tao, and Wei Deng. Improving Reasoning for Diffusion Language Models via Group Diffusion Policy Optimization, October 2025. URL http://arxiv.org/abs/2510.08554. arXiv:2510.08554 [cs].

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

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL https://arxiv.org/abs/2302.04761.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. Insertion transformer: Flexible sequence generation via insertion operations. In *International Conference on Machine Learning*, pp. 5976–5985. PMLR, 2019.

Jason Wei et al. BrowseComp: A simple yet challenging benchmark for browsing agents, 2025. URL https://arxiv.org/abs/2504.12516.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering, 2024. URL https://arxiv.org/abs/2405.15793.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, and Chenxin An. Claw-Eval: Towards trustworthy evaluation of autonomous agents, 2026. URL https://arxiv.org/abs/2604.06132.

Yifan Yu, Yuqing Jian, Junxiong Wang, Zhongzhu Zhou, Donglin Zhuang, Xinyu Fang, Sri Yanamandra, Xiaoxia Wu, Qingyang Wu, Shuaiwen Leon Song, et al. Introspective diffusion language models. *arXiv preprint arXiv:2604.11035*, 2026.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojian Zhong, Aoyan Li, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Kai Shen, and Liang Xiang. Multi-swe-bench: A multilingual benchmark for issue resolving, 2025. URL https://arxiv.org/abs/2504.02605.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *Advances in Neural Information Processing Systems*, 38: 56729–56762, 2026.