The Bitter Lesson of Diffusion Language Models for Agentic Workflows: A Comprehensive Reality Check

Qingyu Lu[♢✿†], **Liang Ding**[♡†], **Kanjian Zhang**[♢♠*], **Jinxia Zhang**[♢], **Dacheng Tao**[✿]
[♢]Southeast University [♡]Alibaba [♠]Southeast University Shenzhen Research Institute
[✿]College of Computing and Data Science at Nanyang Technological University, Singapore 639798
{luqingyu,kjzhang,jinxiazhang}@seu.edu.cn, {liangding.liam,dacheng.tao}@gmail.com
https://coldmist-lu.github.io/DiffuAgent/

Abstract

The pursuit of real-time agentic interaction has driven interest in Diffusion-based Large Language Models (dLLMs) as alternatives to auto-regressive backbones, promising to break the sequential latency bottleneck. **However, does such efficiency gains translate into effective agentic behavior?** In this work, we present a comprehensive evaluation of dLLMs (e.g., LLaDA, Dream) across two distinct agentic paradigms: *Embodied Agents* (requiring long-horizon planning) and *Tool-Calling Agents* (requiring precise formatting).

Contrary to the efficiency hype, our results on Agentboard and BFCL reveal a "***bitter lesson***": current dLLMs fail to serve as reliable agentic backbones, frequently leading to systematic failure. ***(1) In Embodied settings***, dLLMs suffer repeated attempts, failing to branch under temporal feedback. ***(2) In Tool-Calling settings***, dLLMs fail to maintain symbolic precision (e.g. strict JSON schemas) under diffusion noise. To assess the potential of dLLMs in agentic workflows, we introduce **DiffuAgent**, a multi-agent evaluation framework that integrates dLLMs as plug-and-play cognitive cores. Our analysis shows that dLLMs are effective in non-causal roles (e.g., memory summarization and tool selection) but require the incorporation of causal, precise, and logically grounded reasoning mechanisms into the denoising process to be viable for agentic tasks.

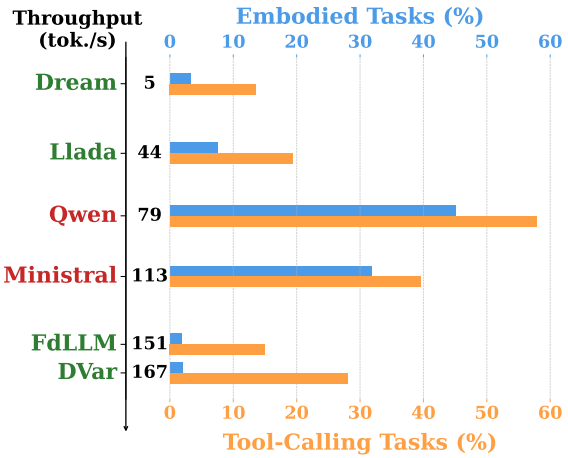

Figure 1: **Performance–Efficiency Trade-offs in Embodied and Tool-Calling Tasks.** Despite higher inference efficiency, FdLLM-7B and DVar-8B do not guarantee comparable agentic performance to autoregressive LLMs. Llada-8B and Dream-7B fall behind LLMs in both task performance and efficiency.

1 Introduction

Agents powered by large language models (LLMs, Yang et al., 2025a; Jiang et al., 2024) have demonstrated strong capabilities in planning and complex reasoning (Wang et al., 2024; Luo et al., 2025), particularly in embodied task-solving environments (Feng et al., 2025; Chang et al., 2024) and tool-calling scenarios (Liu et al., 2025a; Patil et al., 2025). However, such agentic systems often suffer from a **sequential latency bottleneck**, where multi-turn interactions incur substantial inference overhead, demanding faster reasoning and more efficient decision-making.

In this context, Diffusion-based Large Language Models (dLLMs, Nie et al., 2025) have attracted attention as alternatives to auto-regressive backbones, owing to their higher inference efficiency enabled by parallel decoding, while maintaining competitive general performance (Wu et al., 2025a; Ye et al., 2025). However, **does such efficiency gains translate into effective agentic behavior?** In this work, we present a comprehensive reality check of dLLMs, focusing on their long-horizon planning capabilities as *Embodied Agents* and precise formatting capabilities as *Tool-Calling Agents*.

Contrary to the efficiency hype, our results (as shown in Figure 1) on four representative dLLMs across AgentBoard (Chang et al., 2024) and BFCL

[*] Corresponding Author.
[†] Equal contribution. Work done while Qingyu was a visiting researcher at Nanyang Technological University.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 43997–44020
July 2-7, 2026

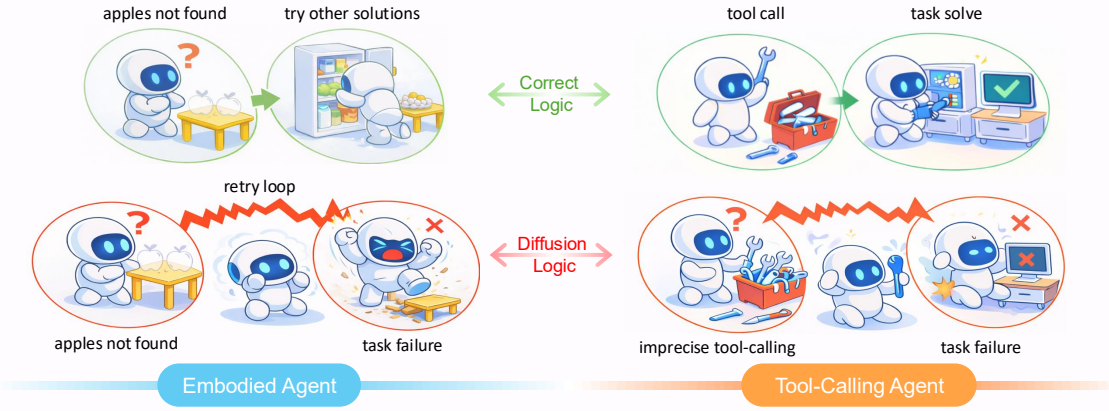

Figure 2: **An Overview of Systematic Failures** in dLLMs. In embodied agent settings, dLLMs tend to repetitively retry the same action instead of exploring alternative plans. In tool-calling agent settings, imprecise or unstable tool invocation further leads to execution failures.

(Patil et al., 2025) reveal a ***bitter lesson***: current dLLMs fail to serve as reliable agentic backbones, particularly in multi-turn interaction scenarios, exhibiting systematic failure behaviors. Specifically, (1) ***in embodied settings***, dLLMs tend to become trapped in repetitive action loops, failing to branch into alternative plans; (2) ***in tool-calling settings***, dLLMs struggle to maintain symbolic precision when generating tool invocations, frequently violating strict JSON schemas or hallucinating API parameters, potentially due to diffusion-induced noise, as illustrated in Figure 2.

To provide deeper insights into the agentic behavior of dLLMs, we further introduce **DiffuAgent**, a novel evaluation framework that treats dLLMs as *plug-and-play cognitive modules* for augmenting LLM-based agents, enabling systematic assessment under different agentic roles. Our results show that dLLMs can be effective when deployed in non-causal roles, such as memory summarization (Xu et al., 2025; Wang et al., 2025c), redundant trajectory detection (Lu et al., 2025), and relevant tool selection (Liu et al., 2025b).

Our contributions are three-fold:

- We present the first systematic study of dLLMs as agentic backbones, revealing consistent and previously underexplored failure modes in multi-turn agentic reasoning.
- We propose **DiffuAgent**, the first evaluation framework that integrates dLLMs as four distinct cognitive modules within a multi-agent setting to better assess agentic behavior.
- We provide extensive empirical evidence showing that dLLMs are effective mainly in non-causal roles, but remain weak in causal planning and formatting–critical scenarios.

This study serves as a foundational step toward **Diffusion-native Agents**. By bridging the gap between non-autoregressive generation and agentic workflows, we highlight a promising direction for future dLLM development, enabling real-time interaction without compromising causal, precise, and logically grounded reasoning capabilities.

2 Preliminaries

2.1 Embodied Agents

Embodied agents operate in interactive environments (e.g., household or virtual worlds), where an LLM acts as the central controller, selecting actions based on accumulated interaction history. This multi-turn decision process can be formalized as a Partially Observable Markov Decision Process (POMDP), in which the agent follows a policy π_θ at each time step t to choose the next action a_t:

$$a_t \sim \pi_\theta(\cdot \mid e_{1:t-1}, u_{\text{task}}),$$

where trajectory $e_{1:t-1} = (a_1, o_1, \ldots, a_{t-1}, o_{t-1})$ consists of past actions $(a_1, \ldots, a_{t-1})$ and corresponding observations $(o_1, \ldots, o_{t-1})$, and u_{task} denotes contextual information associated with the specific task and initial environment configuration.

To elicit reasoning capabilities, we adopt ReAct (Yao et al., 2023) for evaluating embodied agents, a widely-used agentic workflow that synergizes planning and decision-making in multi-turn interactions. This process can be formulated as:

$$[q_t, a_t] = \pi_\theta(\cdot \mid e_{1:t-1}, u_{\text{task}}),$$

where the LLM agent generates an intermediate thought q_t before producing action a_t.

2.2 Tool-Calling Agents

Agentic LLMs are expected to exhibit effective tool-calling (also referred to as *function-calling*) capabilities, where the agent is equipped with external tools and must decide whether, when, and how to invoke them to solve complex tasks (Patil et al., 2025). This setting can be viewed as a special case of the agentic interaction paradigm, where, at each interaction turn, the agent is provided with a set of available tool descriptions $\mathcal{D} = \{\tau_1, \ldots, \tau_N\}$ instead of interacting with a complex environment, and attempts to fulfill the user request u_{user} by generating one or more structured tool invocations as actions. This formats as

$$\mathcal{C} = \{(\tau_i, \alpha_i)\}_{i=1}^{K} \sim \pi_\theta(\cdot \mid u_{\text{user}}, \mathcal{D}), \quad (1)$$

where π_θ denotes the agent policy model, $\mathcal{C}$ is the set of generated tool calls, τ_i denotes the i-th selected tool, and α_i denotes argument. Each generated tool call is then executed by its associated tool, yielding a set of execution results:

$$\mathcal{O} = \text{Exec}(\mathcal{C}) = \{\, \tau_i(\alpha_i) \mid (\tau_i, \alpha_i) \in \mathcal{C} \,\}, \quad (2)$$

where the execution results $\mathcal{O}$ for all results are returned to the agent as feedback, based on which the agent decides whether further tool calls are required. The interaction terminates when the agent determines that the user request has been resolved or when a predefined step limit is reached.

2.3 Diffusion-based LLMs

Autoregressive LLMs follow a next-token prediction paradigm (Luo et al., 2025), in which tokens are decoded sequentially, one at a time. This inherent sequential nature limits decoding efficiency, particularly in generation-intensive scenarios. Inspired by diffusion probabilistic modeling (Yang et al., 2023), dLLMs originally developed for continuous domains such as images (Amit et al., 2021) and audio (Nam et al., 2025). Rather than relying on strict left-to-right generation, dLLMs generate tokens in parallel, offering greater potential for efficient inference acceleration (Wu et al., 2025b). As the parallel generation and sampling strategies differ slightly among the selected dLLMs, we explore different dLLMs and several optimization techniques during decoding stage.

☞ *See Appendix A for a detailed summary of the decoding strategies of the selected dLLMs.*

3 Experimental Setup

3.1 Evaluation Data

Datasets We evaluate embodied agents using AgentBoard (Chang et al., 2024) across three interactive environments: AlfWorld (Shridhar et al., 2021) (134 household tasks), ScienceWorld (Wang et al., 2022) (90 scientific experiments), and BabyAI (Chevalier-Boisvert et al., 2019) (112 grid-based navigation and interaction tasks). Tool-calling agentic ability is assessed on BFCL-v3 (Patil et al., 2025). We sample at most 50 instances per BFCL-v3 category (using all samples when fewer than 50 are available), yielding 758 evaluation examples in total covering all categories.

☞ *See Appendix B for detailed dataset descriptions, including their settings and example instances.*

Metrics For embodied agents, we report both success rate and progress rate. **Success rate** measures the proportion of tasks successfully completed by an agent, while **progress rate** (Chang et al., 2024) quantifies how much an agent advances toward the task goal, making it a more informative metric for evaluating incremental improvements. For tool-calling evaluation, we adopt the official BFCL evaluation suite and report the percentage of successful instances as our primary metric.

3.2 Agentic Backbones

LLMs We consider open-source LLMs under 10B parameters for reproducibility and efficiency. Specifically, we use Qwen-8B (Yang et al., 2025a), adopting the non-thinking variant to meet real-time latency constraints[1], and Ministral-8B (Jiang et al., 2024), an instruction-tuned 8B model. Both models are evaluated in text-only settings[2].

dLLMs We employ four recent dLLMs in our experiments: Llada-8B (Nie et al., 2025), a strong diffusion LLM with general performance competitive with Llama3-8B; Dream-7B (Ye et al., 2025), which is initialized from Qwen2.5-7B weights and adopts token-level noise rescheduling for context-adaptive denoising; FdLLM-7B (Fast-dLLM v2; Wu et al., 2025a), a block-diffusion model enabling parallel decoding within each block for efficient inference; and DVar-8B (dLLM-Var; Yang et al., 2025b), which supports native variable-length generation via accurate EOS prediction.

[1] https://huggingface.co/Qwen/Qwen3-8B

[2] https://huggingface.co/mistralai/Ministral-3-8B-Instruct-2512

Embodied Agent	AlfWorld		ScienceWorld		BabyAI		Avg.	
	Success	Progress	Success	Progress	Success	Progress	Success	Progress
Qwen-8B	**76.1** ±0.8	**85.6** ±0.3	**26.7** ±2.2	**55.1** ±0.3	32.1 ±0.9	45.7 ±1.0	**45.0** ±0.2	**62.1** ±0.5
Ministral-8B	45.5 ±1.1	66.2 ±0.6	13.3 ±1.7	52.0 ±0.1	**36.6** ±1.4	**46.6** ±1.0	31.8 ±0.7	54.9 ±0.5
Llada-8B	5.2 ±0.4	18.5 ±0.6	1.1 ±0.0	8.6 ±0.4	16.1 ±0.9	22.0 ±1.3	7.5 ±0.2	16.4 ±0.4
Dream-7B	0.7 ±0.0	6.0 ±0.1	0.6 ±0.6	5.3 ±0.2	8.9 ±0.5	14.8 ±0.5	3.4 ±0.1	8.7 ±0.2
FdLLM-7B	3.3 ±3.3	7.8 ±3.3	0.7 ±0.6	6.4 ±1.2	5.4 ±0.5	12.6 ±2.2	3.1 ±1.3	8.9 ±1.5
DVar-8B	0.7 ±0.0	10.0 ±0.0	0.0 ±0.0	1.9 ±0.0	5.4 ±0.0	15.0 ±0.0	2.0 ±0.0	8.9 ±0.0

Table 1: **Comparison of Success Rate (%) and Progress Rate (%)** across different LLMs and dLLMs on three *Embodied* tasks. Best results are highlighted in **bold**. The lower-right fluctuation values indicate the variability estimated from 3 runs, showing the possible effect of error propagation.

Tool-Calling Agent	Non-Live	Single-Turn Live					Multi-Turn			Hallucination		Overall
	Avg.	S.	M.	P.	PM.	Avg.	Standard	Challenge	Avg.	Rel.	Irrel.	
Qwen-8B	**87.5**	**82.0**	**80.0**	**75.0**	**75.0**	**78.0**	**20.0**	**10.0**	**12.5**	**94.4**	68.0	**57.8**
Ministral-8B	49.8	74.0	70.0	50.0	45.8	60.0	2.0	4.7	4.0	66.7	58.0	39.5
Llada-8B	23.0	8.0	26.0	0.0	12.5	11.6	0.0	0.0	0.0	66.7	56.0	19.4
Dream-7B	4.2	2.0	4.0	0.0	0.0	1.5	0.0	0.0	0.0	27.8	77.0	13.6
FdLLM-7B	1.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.6	**99.0**	15.0
DVar-8B	35.0	56.0	22.0	37.5	20.8	34.1	0.0	0.0	0.0	44.4	63.0	28.0

Table 2: **Comparison of Success Rates (%)** across different LLMs and dLLMs as *Tool-Calling* agents. **S.**, **M.**, **P.** and **PM.** denote simple, multiple, parallel, and a combination of parallel and multiple tool-calling tasks, respectively. Hallucination indicates whether a tool call is required (**Rel.**) or not required (**Irrel.**). Best results are in **bold**.

3.3 Deployment Details

For fair comparison, we allocate one NVIDIA A800 (80GB) GPU per model. When two models are used in multi-agent scenarios (Section 6), two GPUs are deployed accordingly. We do not employ distributed inference. AR models (Qwen-8B, Ministral-8B) are deployed with vLLM (Kwon et al., 2023) and accessed via OpenAI Chat APIs (Achiam et al., 2023). Diffusion LLMs (Dream-7B, Llada-8B, FdLLM-7B) are reproduced using NVIDIA Fast-dLLM[3] and served through FastAPI.

3.4 Prompts

For embodied agents, we adopt a ReAct-style prompt format for multi-turn planning and action generation. For BFCL, we follow the official implementation and build an OpenAI API version to match the input templates of different models.
☞ *See Appendix F for the prompts used.*

4 Failure of dLLMs as Agent Backbone

4.1 Failure of Replan: Embodied Agents

dLLMs significantly underperform LLMs Table 1 summarizes the performance of embodied agents with LLM and dLLM backbones. Across all environments, dLLMs consistently underperform LLMs, achieving success rates below 10% in most settings, with the only exception being Llada-8B on BabyAI; in some cases, they fail to solve any tasks (0.0%) in ScienceWorld. Progress-rate performance exhibits a similar pattern: almost all progress rates fall below 20%, suggesting that dLLM agents are unable to complete even one subgoal on average. This gap is striking given the competitive performance of dLLMs on general language benchmarks, demonstrating that such gains fundamentally fail to transfer to agentic scenarios requiring long-horizon planning.

Retry loops as a systematic failure mode To further investivate the failure mode of dLLMs as embodied agents, we follow Shinn et al. (2023) to define *retry loop* as three or more consecutive repetitive actions and report their frequency across different backbones. As shown in Figure 3, dLLMs exhibit significantly more frequent *retry loops* than auto-regressive LLMs, repeatedly generating the same action without exploring alternatives. This indicates an over-reliance on recent context, whereas LLM-based agents exhibit more causal decision

[3] https://github.com/NVlabs/Fast-dLLM

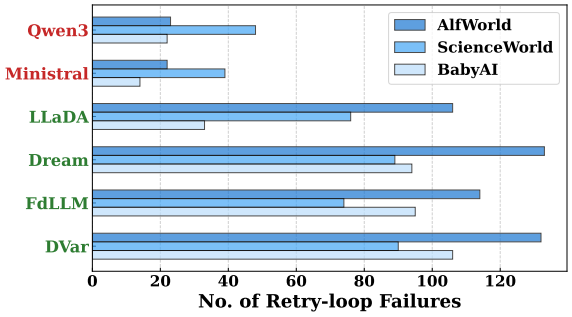

Figure 3: **Comparison of Retry-Loop Failures** across LLMs and dLLMs. A retry-loop is defined as repeatedly executing the same action for more than 3 consecutive steps during task completion.

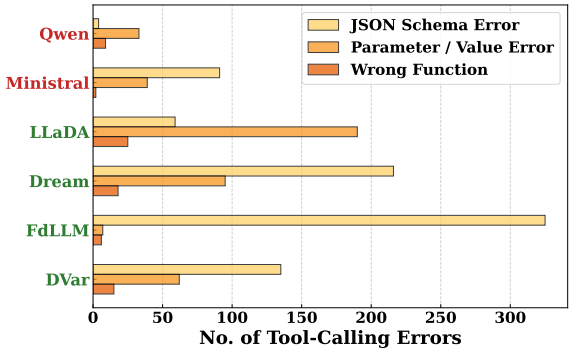

Figure 4: **Comparison of the number of tool-calling failure categories** across LLMs and dLLMs.

patterns and experiences (Sun et al., 2025) by leveraging prior interactions to branch into new actions.

Effect of Error Propagation Since embodied agents may interact with the environment for up to 30 iterations, they are likely to accumulate error propagation over time. We therefore run each embodied setting for 3 runs and report the fluctuation values in Table 1. Most success-rate changes remain within 1%, and most progress-rate fluctuations stay within 2%. The most volatile case is FdLLM-7B, whose success rate shows a fluctuation of $\pm 3.8\%$, suggesting that decoding changes may substantially affect the final outcome. This is consistent with Appendix D.1: changing FdLLM-7B from vanilla decoding to Deferred Commitment Decoding (DCD, Shu et al., 2026) raises its ALFWorld success rate from 3.3 to 10.4.[4]

4.2 Failure of Precision: Tool-Calling Agents

dLLMs underperform LLMs on both single–turn and multi-turn tool-callings Table 2 summarizes the tool-calling results. Consistent with embodied settings, dLLMs underperform auto-regressive LLMs in both single-turn[5] and multi-turn scenarios. Among dLLMs, DVar-8B achieves better single-turn performance but remains suboptimal. Notably, the multi-turn setting is particularly challenging for dLLMs, as none succeeds on any test instance. The high irrelevance score (Irrel.) of FdLLM-7B arises from frequent incorrect tool calls that are classified as irrelevant actions.

Imprecise tool-call formats as a systematic failure mode We further summarize tool-calling failures under single-turn Live and Non-Live settings by categorizing their error types according to Abstract Syntax Tree (AST) evaluation. As shown in Figure 4, JSON schema errors and parameter/value errors dominate for both LLMs and dLLMs. However, compared to LLMs, dLLMs are more prone to produce malformed JSON schemas, except for Llada-8B, which more often exhibits missing parameters or values. These fuzzy or ill-formed formats lead to tool execution failures, indicating that dLLMs struggle to adhere to the strict structural constraints required for tool invocation.

4.3 Failure of Efficiency-Performance Trade-off

As efficiency has become a key consideration for dLLMs, Figure 1 compares efficiency and performance across models. Despite achieving high throughput (above 150 tokens/s), FdLLM-7B and DVar-8B exhibit the worst embodied-task performance, with average success rates below 2%. In contrast, auto-regressive LLMs such as Qwen-8B and Ministral-8B achieve stronger tool-calling and embodied reasoning performance while maintaining acceptable latency. These results show that efficiency gains in dLLMs do not directly translate into improved agentic performance.

4.4 Bitter Lesson: Non-causal and Fuzzy Nature of dLLMs

From the above analysis, we observe a fundamental limitation of dLLMs: despite their efficiency gains, **parallel decoding weakens causal depen-**

[4]Since the BFCL test set has been rebuilt in the latest update and BFCL contains many single-turn cases, making the chance of error propagation is relatively low compared with embodied tasks. Therefore, we do not report multi-run fluctuation for BFCL.

[5]The single-turn average is computed by excluding hallucination categories, which differs from the original BFCL implementation.

dency and induces fuzzy intermediate states, hindering stable commitment to partial plans or structured outputs. This aligns with established challenges in non-autoregressive generation, where the conditional independence assumption has been shown to cause 'uncoordinated' structural predictions in slot filling (Wu et al., 2020) and 'lexical choice errors', particularly for low-frequency tokens, in machine translation (Ding et al., 2021). As a result, dLLMs perform poorly on long-horizon reasoning and strictly structured tasks, serving as a bitter lesson that they should be used with caution as backbone models in agentic workflows requiring strong temporal or symbolic consistency.

Importantly, these results do not suggest that dLLMs are ineffective in agentic systems. Since agentic tasks often require heterogeneous capabilities, we further examine the collaboration between dLLMs and LLMs in multi-agent workflows to clarify the role of dLLMs in agentic scenarios.
☞ *See Appendix C for more rigorous definitions and theoretical explanations of the non-causal and fuzzy failure patterns of dLLMs.*

4.5 Extended Validation

dLLM Optimization Techniques We further examine whether recent dLLM optimization techniques could change our main conclusion. These methods improve performance, but do not close the agentic gap. In particular, Adaptive Parallel Decoding (APD) and Discrete Diffusion Forcing (D2F) bring only limited gains for Dream-7B on embodied tasks, while DCD improves FdLLM-7B on ALFWorld but remains limited on ScienceWorld and BabyAI. On tool-calling tasks, APD, D2F, and DCD all improve BFCL accuracy, yet a substantial gap from strong AR LLMs remains.

Agent-Level Optimization We also examine whether agent-level optimization, such as external refinement or feedback from AR LLMs, could mitigate the weakness of dLLM agents. These strategies provide limited gains and do not overturn our conclusion. AR self-refine raises the success rate from 0.7% to 1.5%, while periodic or step-wise AR feedback raises it to 2.2%, and the progress rate from 10.0% to 15.2–15.4%. These gains suggest that external feedback can partially stabilize dLLM agents, but remain insufficient to close the gap.

Other Agentic Benchmarks We further examine whether the same limitation extends to other agentic benchmarks through Tau-Bench mock. Qwen-8B is the only model that achieves a non-zero score, while the tested dLLM settings either remain at zero pass rate or fail because of response-format incompatibility.

Schema-Checking Methods We further examine whether lightweight schema-checking heuristics or structural guardrails could remove the main weakness of dLLMs in tool calling. Even when all three guardrails are combined, only 21% of outputs achieve syntactic recovery, only 14% reach semantic correctness, and 86% still fail.
☞ *See Appendix D for detailed settings and additional experimental results.*

5 DiffuAgent: A Multi-Agent Evaluation Framework on Analyzing Agentic Behaviors in dLLMs

To better understand the agentic potential of dLLMs, we introduce **DiffuAgent**, which integrates dLLMs as plug-and-play cognitive modules to augment auto-regressive LLMs. As shown in Figure 5, rather than letting dLLMs run the entire agent loop, DiffuAgent assigns them to individual functional modules, including memory, verification, and tool-related selection or format editing. This multi-agent modular design allows us to study their strengths and weaknesses more clearly, without confusing them with overall agent failures.

5.1 Modules in Embodied Agents

Pre-hoc: Memory We incorporate a memory-augmented module to compress long interaction histories while preserving salient information for agentic decision-making. The agent periodically summarizes past trajectories into a textual memory every k_{mem} steps[6], reusing the existing memory otherwise. This process is formulated as

$$m' = \text{Memory}(m,\, e_{t-k_{\text{mem}}:t-1}, u_{\text{task}})\,. \quad (3)$$

During the subsequent k_{mem} decision steps, the policy conditions on the compressed memory together with a short-term interaction history e_{latest} consisting of the most recent steps:

$$[q_t, a_t] = \pi_\theta\big(\cdot \mid m',\, e_{\text{latest}},\, u_{\text{task}}\big)\,. \quad (4)$$

This design facilitates evaluation of agents under memory compression, where inaccurate memory updates may hinder information preservation and induce erroneous or cyclic behaviors.

[6]The memory module is invoked every $k_{\text{mem}} = 5$ steps in our experiments, while the last two interactions are always retained to preserve recent context.

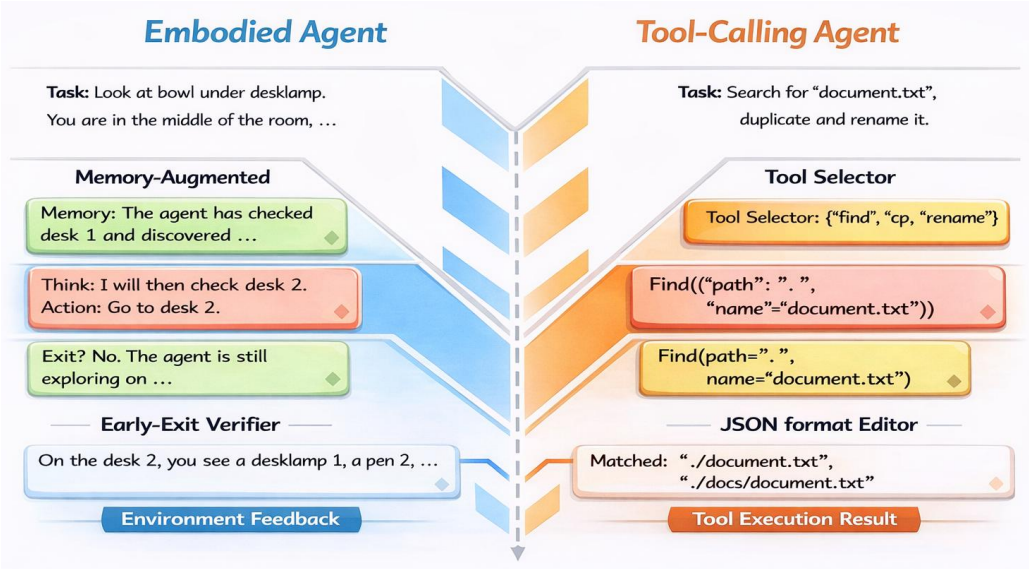

Figure 5: **Overview of DiffuAgent.** The framework integrates four modules. For embodied agents, we introduce a memory-augmented module for history compression and an early-exit verifier for global trajectory checking. For tool-calling agents, we include a tool selector over the library and a JSON format editor.

Post-hoc: Early-Exit Verifier To evaluate an agent's self-awareness of being stuck, we follow (Lu et al., 2025) to incorporate an early-exit verification module built on an LLM or dLLM backbone. The verifier is triggered every $k_{\text{earlyexit}}$ steps[7] and is prompted to determine whether the agent has entered a deadlock or repetitive loop. This verification process can be formulated as:

$$\text{Verifier}(e_{1:t}, u_{\text{task}}) \in \{0, 1\} \tag{5}$$

A binary decision is then used to terminate the episode early, reducing unnecessary generation steps and improving overall efficiency.

5.2 Modules in Tool-Calling Agents

Pre-hoc: Tool Selector Existing tool-calling workflows suffer from a mismatch between large tool libraries and task-specific needs, which increases decision complexity and leads to inefficient or erroneous tool use. We introduce a pre-hoc tool selection module that filters the full tool set and provides a condensed subset of relevant tools prior to tool calling. Formally, at each interaction turn, given the user message u_{user} and the full tool set $\mathcal{D}$, the tool selector produces a reduced tool subset:

$$\mathcal{D}' \subseteq \mathcal{D}, \quad \mathcal{D}' = \text{Selector}(u_{\text{user}}, \mathcal{D}), \tag{6}$$

where $\mathcal{D}'$ contains only tools deemed relevant to the current user request. The selected tool subset $\mathcal{D}'$ is then provided to the tool-calling agent, which performs tool invocation conditioned on the reduced action space:

$$\mathcal{C}' \sim \pi_\theta(\cdot \mid u_{\text{user}}, \mathcal{D}'). \tag{7}$$

Post-hoc: Tool-Call Editor Although tool calls may select correct tools and parameters, they often violate the required JSON schema, leading to execution failures. We therefore introduce a tool-call editor that post-processes malformed outputs into schema-compliant formats, enabling post-hoc evaluation of structural adherence without altering the selected function or parameters.

6 Analysis of Agentic Behaviors in dLLMs

We analyze the behavior of dLLMs within the DiffuAgent framework under multi-agent settings.

6.1 dLLMs Are Competitive Memory Modules for Memory-Augmented Agents

In *memory-augmented agents*, incorporating a memory module generally improves performance over the **w/o** baseline across tasks (Table 3), indicating effective preservation of useful information. An exception is BabyAI, where long observation strings at each step may hinder effective memory summarization and lead to marginal or negative gains. Comparing memory backbones, dLLMs achieve performance comparable to Qwen-8B, suggesting their potential as memory modules.

[7] We set $k_{\text{earlyexit}} = 5$ in our experiments.

Model		AlfWorld		ScienceWorld		BabyAI		Avg.	
Agent	Memory	Success	Progress	Success	Progress	Success	Progress	Success	Progress
Qwen-8B	**w/o**	36.6	59.5	20.0	53.2	**28.6**	**39.0**	28.4	50.6
	Qwen-8B	54.5	72.8	28.9	59.5	21.4	32.2	34.9	54.8
	Llada-8B	**67.2**	**81.1**	**31.1**	**61.9**	23.2	35.9	**40.5**	**59.6**
	Dream-7B	64.9	77.4	**31.1**	60.8	20.5	33.6	38.9	57.3
	FdLLM-7B	57.5	72.8	28.9	56.5	20.5	35.2	35.6	54.8
	DVar-8B	61.2	76.6	26.7	58.4	24.1	35.4	37.3	56.8
Ministral-8B	**w/o**	22.4	43.7	17.8	57.6	33.0	44.2	24.4	48.5
	Ministral-8B	32.8	60.7	**34.4**	65.7	**33.9**	**45.5**	**33.7**	**57.3**
	Llada-8B	37.3	62.5	28.9	**67.2**	29.5	41.3	31.9	57.0
	Dream-7B	**39.6**	**63.5**	26.7	60.9	25.9	37.3	30.7	53.9
	FdLLM-7B	27.6	50.6	16.7	51.0	24.1	35.7	22.8	45.8
	DVar-8B	35.8	59.7	24.4	58.0	29.5	39.8	29.9	52.5

Table 3: Performance comparison of *memory-augmented agents* across different LLMs and dLLMs on three embodied environments. "**w/o**" indicates no memory, retaining only recent interactions.

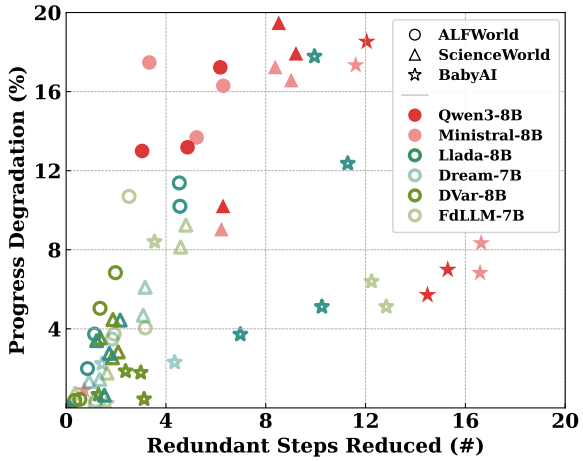

Figure 6: **Comparison of early-exit behavior across LLMs and dLLMs**. Filled and hollow markers denote LLM- and dLLM-based verifiers, respectively, while different marker shapes indicate different tasks.

However, performance varies across environments: Ministral-8B performs better on BabyAI and ScienceWorld but worse on AlfWorld, which we attribute to its tendency to generate longer thoughts in ReAct, making summarization more challenging. This suggests that dLLMs may be less suitable for lengthy and complex reasoning traces.

6.2 LLM Verifiers Tend to Trigger Premature Early Exits, Whereas dLLMs Terminate More Reliably

To better assess whether dLLMs can provide early termination through self-awareness based on existing trajectories, we select the four best-performing trajectories from memory-augmented embodied agents[8] and apply early-exit verifiers implemented with different LLMs or dLLMs. We compute the two efficiency metrics, redundancy reduction and progress degradation[9] across embodied tasks with different backbones and report them in Figure 6. An interesting phenomenon is that Auto-regressive LLMs exhibit more aggressive early exits, sharply reducing redundancy but causing severe progress loss, whereas dLLM-based verifiers behave more conservatively, achieving smaller redundancy reductions with less degradation, likely due to their global trajectory awareness.

6.3 dLLMs Are Effective Tool Selectors but Struggle as Tool-Call Editors

To further investigate the effectiveness of dLLMs as tool-calling modules, we adopt the BFCL-v3 multi-turn benchmark (Patil et al., 2025), using 50 randomly selected instances to construct a 200-sample test set. The Standard setting involves multi-step tool interactions across multiple user turns, while the Challenge setting includes missing functions, missing parameters, and long-context inputs.

We perform ablations by replacing the selector and/or editor modules in BFCL multi-turn setting. As shown in Figure 7, LLM-based modules consistently outperform dLLMs. Among dLLMs, Llada-8B and Dream-7B serve as relatively effec-

[8]For the first two trajectories, we use Ministral-8B and Qwen-8B as both the agent and memory module, and Llada-8B as the memory module for the remaining settings.

[9]As illustrated in Lu et al. (2025), redundant steps quantify the potential for efficiency improvement, while progress degradation measures the loss in performance. These metrics jointly capture the trade-off between efficiency and performance.

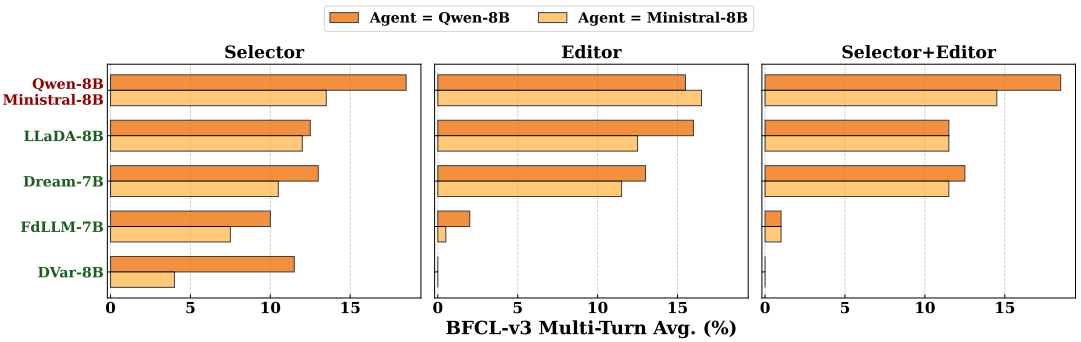

Figure 7: **Ablation performance of tool-calling agents** on BFCL-v3 Multi-Turn benchmark, evaluated with different backbone models for the agent modules (*Selector* and *Editor*). 0 indicates no successful instance.

tive selectors and editors, achieving performance comparable to LLM baselines, whereas FdLLM-7B and DVar-8B degrade performance as editors, possibly due to imprecise tool calls. Interestingly, DVar-8Bimproves performance when used as a selector for Qwen-8B but harms Ministral-8B. This behavior can be attributed to DVar-8B's tendency to generate weakly filtered tool subsets, benefiting Qwen-8B with strong selection capacity but overwhelming Ministral-8B and reducing task success. ☞ *See Appendix E and Table 10 for detailed BFCL multi-turn ablation results.*

7 Related Work

Diffusion-based LLMs dLLMs enable non-autoregressive generation via parallel denoising, offering substantial speedups over autoregressive LLMs. Models such as LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025) achieve competitive standalone performance, with further improvements from block-based diffusion, KV-cache reuse, and confidence-aware decoding (Arriola et al., 2025; Wu et al., 2025b,a). Recent decoding advances further improve diffusion inference through APD (Israel et al., 2025), DCD (Shu et al., 2026), D2F (Wang et al., 2025b), and Residual Context Diffusion (RCD, Hu et al., 2026). However, dLLM behavior under multi-turn, causally grounded agentic interaction remains underexplored; we therefore systematically study dLLMs as cognitive modules within agentic workflows.

LLM Agents LLM agents show strong performance in embodied reasoning, tool use, and interactive decision-making, enabled by expert trajectory training (e.g., ETO (Song et al., 2024), Agent-FLAN (Chen et al., 2024)), prompt-based reasoning frameworks (e.g., ReAct (Yao et al., 2023), PreAct (Fu et al., 2025b), StateFlow (Wu et al., 2024)), and planning or imitation signals (Lin et al., 2023). Tool-calling agents further focus on tool selection (Lumer et al., 2025), planner design (Liu et al., 2025b), and schema alignment (Lee et al., 2025), but largely assume autoregressive backbones and overlook inference efficiency. In contrast, we investigate efficiency-oriented dLLMs as agentic backbones and reveal systematic failures in causally dependent decision processes.

Agent Verification and Memory Recent work studies model- or agent-based verification across text (Zheng et al., 2023; Lu et al., 2024), code (Chen et al., 2024), and autonomous agents (Pan et al., 2024), while memory compression methods (Xu et al., 2025; Wang et al., 2025a,c) recover agent capabilities under limited context windows. In contrast, we propose a multi-agent evaluation framework that treats dLLMs as cognitive modules to better measure its capabilities.

8 Conclusion

We conduct a systematic evaluation of dLLMs in agentic settings. Despite their inference efficiency, we first demonstrate the "bitter lesson": dLLMs are unreliable agentic backbones under multi-turn interaction, exhibiting repetitive action loops in embodied tasks and imprecise tool calls under strict formatting constraints. We then introduce **DiffuAgent**, a modular evaluation framework that decomposes agentic workflows into plug-and-play cognitive roles. Our analysis shows that dLLMs struggle with causal planning and formatting-critical tasks, but remain effective in non-causal roles, such as memory summarization and tool selection, motivating diffusion-native agent designs.

Limitations

The limitations of our work are as follows:

- **Limited Coverage of dLLMs and Benchmarks:** Our study evaluates a representative but limited set of diffusion-based LLMs on AgentBoard and BFCL. We focus on a subset of the test suites, considering only embodied AI tasks in AgentBoard while excluding other scenarios such as web-based tasks. For BFCL, we restrict our evaluation to versions v1–v3 to capture the core challenges of tool calling. While we believe our findings are indicative of the current behavior of dLLMs, they may not fully generalize to future architectures or broader agentic settings. We leave a more comprehensive evaluation to future work.

- **Inference-Only Analysis:** We focus on the agentic behavior of post-training dLLMs without incorporating task-specific fine-tuning or reinforcement learning. Although this setting enables a controlled comparison, targeted training objectives or architectural adaptations may alleviate some of the observed failure modes, which we leave for future work. At the same time, prior work suggests that simple fine-tuning often behaves more like behavior cloning and may not substantially improve agentic capabilities (Song et al., 2024). More meaningful gains may require multi-stage agentic training, such as continued pre-training or reinforcement learning, which incurs substantially higher computational cost.

- **Ablation Completeness:** Our ablation study covers only a subset of selector–editor configurations under the DiffAgent framework, and assumes an LLM-based main workflow agent. We focus on evaluating dLLMs as auxiliary cognitive modules rather than as primary agents. Exploring dLLMs as the main workflow agent is left for future work.

- **Fixed Agentic Workflow Assumptions:** DiffuAgent evaluates dLLMs by inserting them into predefined cognitive roles within a fixed agent pipeline. This modular design facilitates systematic analysis but may underestimate the potential of diffusion models in end-to-end or co-designed agentic systems optimized for diffusion-native reasoning.

- **LLM Self-Awareness:** One possible explanation for the observed improvements is the limited self-verification capability of a single LLM, where using the same model for both generation and verification may hinder timely error detection. In contrast, multi-agent settings with heterogeneous models introduce distributional diversity that can implicitly facilitate error correction and improve agentic performance. While we acknowledge that this effect may exist, we do not believe it substantially affects our main conclusions. Due to experimental budget constraints, we do not explicitly isolate this "self-awareness" effect and leave a systematic investigation for future work.

Ethics Statement

We take ethical considerations seriously and conduct this research in accordance with established ethical standards. This work focuses on evaluating diffusion-based large language models (dLLMs) in agentic settings and introducing a modular evaluation framework for analyzing their behaviors. The proposed evaluation framework and analyses do not introduce prompts or mechanisms intended to elicit harmful, unsafe, or deceptive outputs from the models.

All datasets, environments, and models used in this study are publicly available and widely adopted in prior research. Our experiments are conducted in simulated embodied and tool-calling environments, and no human participants are involved as evaluators or subjects. The study does not collect, process, or infer any personal or sensitive information.

Our findings highlight limitations and failure modes of dLLMs in multi-turn agentic interactions, with the goal of improving transparency, reliability, and safety in future agentic system design. We report all results and conclusions accurately and objectively, without exaggerating model capabilities or risks.

Acknowledgments

We thank the anonymous reviewers and the area chair for their insightful comments and suggestions. This work was completed while Qingyu Lu was a visiting scholar at Nanyang Technological University, Singapore, and we thank the China Scholarship Council for its sponsorship. This research is supported by the Fundamental Research Funds for the Central Universities under

Grant 2242025F20002, the National Natural Science Foundation of China under Grant 61973083, and the Shenzhen Science and Technology Program under Grant JCYJ20210324121213036. Dr Tao's research is partially supported by NTU RSR and Start Up Grants.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint*.

Tomer Amit, Tal Shaharbany, Eliya Nachmani, and Lior Wolf. 2021. Segdiff: Image segmentation with diffusion probabilistic models. *arXiv preprint*.

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Jiaqi Han, Zhihan Yang, Zhixuan Qi, Subham Sekhar Sahoo, and Volodymyr Kuleshov. 2025. Interpolating autoregressive and discrete denoising diffusion language models. In *ICLR*.

Ma Chang, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *NeurIPS*.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024. Agent-FLAN: Designing data and methods of effective agent tuning for large language models. In *ACL*.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2019. Babyai: A platform to study the sample efficiency of grounded language learning. In *ICLR*.

Liang Ding, Longyue Wang, Xuebo Liu, Derek F Wong, Dacheng Tao, and Zhaopeng Tu. 2021. Understanding and improving lexical choice in non-autoregressive translation. In *ICLR*.

Zhaohan Feng, Ruiqi Xue, Lei Yuan, Yang Yu, Ning Ding, Meiqin Liu, Bingzhao Gao, Jian Sun, Xinhu Zheng, and Gang Wang. 2025. Multi-agent embodied ai: Advances and future directions. *arXiv preprint*.

Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma GongQue, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. 2025a. Agentrefine: Enhancing agent generalization through refinement tuning. In *ICLR*.

Dayuan Fu, Jianzhao Huang, Siyuan Lu, Guanting Dong, Yejie Wang, Keqing He, and Weiran Xu. 2025b. PreAct: Prediction enhances agent's planning ability. In *COLING*.

Yuezhou Hu, Harman Singh, Monishwaran Maheswaran, Haocheng Xi, Coleman Hooper, Jintao Zhang, Aditya Tomar, Michael W. Mahoney, Sewon Min, Mehrdad Farajtabar, Kurt Keutzer, Amir Gholami, and Chenfeng Xu. 2026. Residual context diffusion language models. *arXiv preprint*.

Daniel Israel, Guy Van den Broeck, and Aditya Grover. 2025. Accelerating diffusion llms via adaptive parallel decoding. In *NeurIPS*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint*.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *SOSP*.

Jonggeun Lee, Woojung Song, Jongwook Han, Haesung Pyun, and Yohan Jo. 2025. Don't adapt small language models for tools; adapt tool schemas to the models. *arXiv preprint*.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. 2023. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *NeurIPS*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, et al. 2025a. Toolace: Winning the points of llm function calling. In *ICLR*.

Yanming Liu, Xinyue Peng, Jiannan Cao, Yuwei Zhang, Xuhong Zhang, Sheng Cheng, Xun Wang, Jianwei Yin, and Tianyu Du. 2025b. Tool-planner: Task planning with clusters across multiple tools. In *ICLR*.

Qingyu Lu, Liang Ding, Siyi Cao, Xuebo Liu, Kanjian Zhang, Jinxia Zhang, and Dacheng Tao. 2025. Runaway is ashamed, but helpful: On the early-exit behavior of large language model-based agents in embodied environments. In *EMNLP*.

Qingyu Lu, Baopu Qiu, Liang Ding, Kanjian Zhang, Tom Kocmi, and Dacheng Tao. 2024. Error analysis prompting enables human-like translation evaluation in large language models. In *ACL*.

Elias Lumer, Anmol Gulati, Faheem Nizar, Dzmitry Hedroits, Atharva Mehta, Henry Hwangbo, Vamse Kumar Subbiah, Pradeep Honaganahalli Basavaraju, and James A. Burke. 2025. Tool and agent selection for large language model agents in production: A survey. *Preprints*.

Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, et al. 2025. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *NeurIPS*.

KiHyun Nam, Jongmin Choi, Hyeongkeun Lee, Jungwoo Heo, and Joon Son Chung. 2025. Diffusion-link: Diffusion probabilistic model for bridging the audio-text modality gap. *arXiv preprint*.

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

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. 2024. Autonomous evaluation and refinement of digital agents. In *COLM*.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *NeurIPS*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. Alfworld: Aligning text and embodied environments for interactive learning. In *ICLR*.

Yingte Shu, Yuchuan Tian, Chao Xu, Yunhe Wang, and Hanting Chen. 2026. Deferred commitment decoding for diffusion language models. *arXiv preprint*.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization of LLM agents. In *ACL*.

Zeyi Sun, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiaoyi Dong, Tong Wu, Dahua Lin, and Jiaqi Wang. 2025. Seagent: Self-evolving computer use agent with autonomous learning from experience. *arXiv preprint*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*.

Qingyue Wang, Yanhe Fu, Yanan Cao, Shuai Wang, Zhiliang Tian, and Liang Ding. 2025a. Recursively summarizing enables long-term dialogue memory in large language models. *Neurocomputing*.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. ScienceWorld: Is your agent smarter than a 5th grader? In *EMNLP*.

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, Kai Yu, and Zhijie Deng. 2025b. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing. *arXiv preprint*.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. 2025c. Mem-{\alpha}: Learning memory construction via reinforcement learning. *arXiv preprint*.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. 2025a. Fast-dllm v2: Efficient block-diffusion llm. *arXiv preprint*.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. 2025b. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint*.

Di Wu, Liang Ding, Fan Lu, and Jian Xie. 2020. Slotrefine: A fast non-autoregressive model for joint intent detection and slot filling. In *EMNLP*.

Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. 2024. Stateflow: Enhancing llm task-solving through state-driven workflows. In *COLM*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *NeurIPS*.

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

Ruihan Yang, Prakhar Srivastava, and Stephan Mandt. 2023. Diffusion probabilistic modeling for video generation. *Entropy*.

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

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *ICLR*.

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *NeurIPS*.

A Description of dLLM Inference Strategies

This section presents a concise overview of the parallel decoding strategies of the dLLMs evaluated in this work.

A.1 Llada-8B: Parallel Reverse Sampling with Confidence-Based Remasking

LLaDA (Nie et al., 2025) is trained from scratch as a masked diffusion language model, learning a Transformer-based mask predictor under random masking. During SFT, prompt tokens remain unmasked while response tokens are masked; `|EOS|` is treated as a normal token during training and used for truncation at inference.

At inference, LLaDA performs reverse diffusion from a fully masked sequence. Given reverse steps T and response length L, the process starts from

$$x^{(T)} = (M, \ldots, M) \in \{M\}^L.$$

At each step $t = T, \ldots, 1$, all masked positions (M) are predicted in parallel:

$$q_i(\cdot) = p_\theta(x_i \mid p, x^{(t)}), \quad x_i^{(t)} = M,$$

followed by sampling and partial remasking to obtain $x^{(t-1)}$.

Instead of random remasking, LLaDA applies *low-confidence remasking*, where tokens with the smallest confidence (e.g., lowest $\max_v q_i(v)$) are remasked, improving generation quality while maintaining parallelism. The final output is $x^{(0)}$, truncated at the first `|EOS|`.

A.2 Dream-7B: Discrete Diffusion Inference and Parallel Denoising

Dream 7B (Ye et al., 2025) follows a discrete diffusion-based generation paradigm, performing inference via iterative denoising over a fixed-length masked sequence. Starting from an initial state filled with `[MASK]` (or noise-corrupted) tokens, the model progressively refines the sequence over multiple diffusion steps, predicting all token positions in parallel at each step.

Formally, given a total of T diffusion steps, inference transforms an initial noisy sequence $x^{(T)}$ into a clean output $x^{(0)}$. At each step,

$$x^{(t-1)} \sim p_\theta(x \mid x^{(t)}, p), \quad t = T, \ldots, 1,$$

where updates may be applied to all or a subset of masked positions according to a predefined schedule or confidence-based criterion. As multiple tokens can be updated simultaneously, Dream naturally supports parallel decoding as well as infilling-style generation.

This iterative denoising procedure exposes a flexible latency–quality trade-off: fewer diffusion steps yield faster inference at the cost of potential degradation in generation quality. The Dream implementation provides configurable step counts and sampling strategies to accommodate different deployment requirements.

A.3 Fast-dLLM: Training-Free Acceleration via KV Caching and Confidence-Aware Parallel Decoding

Fast-dLLM (Wu et al., 2025b) is a **training-free** inference acceleration framework built on top of existing diffusion LLMs (e.g., LLaDA and Dream), modifying only the decoding procedure. It targets two challenges: enabling approximate KV caching under bidirectional attention and reducing quality degradation from parallel unmasking. In our experiments, it is applied to both Llada-8B and Dream-7B.

Fast-dLLM adopts **block-wise decoding** to make caching feasible: the generation is partitioned into blocks, where KV states of the fixed context (prompt and completed blocks) are cached and reused across denoising steps, and refreshed only at block boundaries. DualCache further improves reuse by caching both prefix and masked suffix blocks.

To mitigate the curse of parallel decoding, Fast-dLLM employs **confidence-aware parallel decoding**. For each masked position i,

$$c_i = \max_{v \in \mathcal{V}} p_\theta(x_i = v \mid \text{context}),$$

and only tokens with $c_i \geq \tau$ are unmasked (or at least the single highest-confidence token to ensure progress). A factor-based variant selects the largest

K satisfying

$$(K+1)\big(1-c_{(K)}\big) < \gamma.$$

Together with (Dual) KV caching, this strategy achieves large inference speedups with minimal accuracy loss.

A.4 FdLLM-7B: Block-Diffusion Inference with Hierarchical Caching

Fast-dLLM v2 (Wu et al., 2025a) adapts a pretrained autoregressive LLM (e.g., Qwen2.5-Instruct (Yang et al., 2025a)) into a block-diffusion decoder via light fine-tuning. The core design enforces causal generation across blocks while allowing bidirectional refinement within each block, preserving global left-to-right semantics while enabling parallel token updates locally.

At inference time, blocks are generated sequentially. For block b, the prefix $x_{<b}$ is fixed, and decoding starts from an all-mask block

$$x_b^{(0)} = (M, \ldots, M) \in \{M\}^B.$$

Masked positions are iteratively refined using confidence-aware parallel decoding. For each masked index i,

$$c_i = \max_{v \in \mathcal{V}} p_\theta(x_i = v \mid x_{<b}, x_b),$$

and tokens with $c_i \geq \tau$ are unmasked (or at least the single highest-confidence token is selected to ensure progress).

Efficiency is achieved via **hierarchical caching**: a block-level cache reuses KV states for fully decoded past blocks, while a sub-block cache (DualCache-style prefix–suffix reuse) reduces recomputation during within-block refinement. This design achieves up to $\sim 2.5\times$ speedup over standard autoregressive decoding with minimal quality degradation.

A.5 DVar-8B: EOS-Led Variable-Length Block Diffusion

dLLM-Var (Yang et al., 2025b) enables native variable-length decoding for diffusion LLMs, removing the fixed-length constraint of vanilla dLLMs. It is obtained by lightly fine-tuning Llada-8B to adjust EOS behavior while preserving the diffusion modeling.

At inference, decoding follows an EOS-led block-diffusion process under bidirectional attention. Starting from a prompt, a block of B masked tokens is appended and denoised in parallel; if no `|EOS|` is produced, additional masked blocks are appended iteratively until EOS appears. At block k,

$$x^{(k)} = [x_{\text{prompt}}, \hat{x}_{1:(k-1)B}, \underbrace{M, \ldots, M}_{B}],$$

and decoding terminates at the earliest EOS position.

Efficiency comes from simple KV caching: the prompt and completed blocks are cached as fixed context, while computation focuses on the current block. Parallel updates can be gated by a confidence threshold,

$$\max_v p_\theta(x_i = v) \geq \tau \ (\tau \approx 0.9),$$

achieving substantial speedups without specialized attention masks or complex cache refresh.

B Description of Benchmarks

We provide a brief description of the benchmarks used in our experiments to facilitate a clear understanding of the task requirements and to improve reproducibility and comparability with prior work.

B.1 AgentBoard

We adopt three embodied AI task environments from AgentBoard (Chang et al., 2024): AlfWorld, ScienceWorld, and BabyAI. Table 11 reports the number of tasks ("#Tasks") in each test set, along with a representative example trajectory from each environment, collected using Qwen-8B as the agentic backbone.

B.2 BFCL-v3

We evaluate tool-calling agents on the widely used tool-calling benchmark BFCL-v3 (Patil et al., 2025). Table 12 summarizes the detailed configuration of the test set. Since the original BFCL test set contains over 4,000 samples, we randomly select 50 samples from each category defined in the original dataset (see the "Categories II" column) using a fixed random seed of 42. For categories with fewer than 50 samples, we include all available samples for evaluation. This procedure yields a total of 758 test samples. To accommodate space constraints, we further regroup the original categories into a set of consolidated labels ("Categories I") when reporting the main results in Table 2.

Step	Thought	Action	Observation
1	I should start by checking the cabinets for finding a pan.	go to cabinet 1	On cabinet 1, you see a bowl 1 and a plate 1.
2	I should start by checking the cabinets for finding a pan.	go to cabinet 1	Nothing happens.
3	I should start by checking the cabinets for finding a pan.	go to cabinet 1	Nothing happens.

Table 4: **Retry loop example** from FastdLLM-7B on AlfWorld #44. **Goal:** Cool a pan and put it on the countertop.

Field	Content
User Query	Can you search for the song 'Shape of You'?
Function	`play_spotify_song` — searches for a song on Spotify using a query and plays it.
Model Output (Raw)	`[play_spotify_song(query','':': 'Shape of You',`
Correct Function Call	`[play_spotify_song(query="Shape of You")]`

Table 5: **Schema violation example** from FastdLLM-7B on BFCL live_simple_239-125-2.

C Definitions and Theoretical Explanations of the Non-Causal and Fuzzy Nature of dLLMs

To support the analysis in Section 4.4, we provide more rigorous definitions, case demonstrations, and theoretical explanations for the non-causal and fuzzy failure patterns of dLLMs.

C.1 Non-Causal

Definition *Non-causal* refers to the contrast with standard autoregressive (AR) causal decoding, where tokens are generated sequentially based on previous context ($p(x_t \mid x_{<t})$), enabling reasoning and Chain-of-Thought (CoT) capabilities. Diffusion-based LLMs (dLLMs) generate tokens in parallel, which may weaken this causal assumption and reduce the model's ability for incremental reasoning.

Theoretical Explanation As shown in Table 4, dLLMs may repeatedly preserve an outdated action hypothesis even after receiving new environmental feedback. In AR decoding, the causal attention mask ensures that each position only attends to previous tokens, so new observations (e.g., "On cabinet 1, see a bowl and a plate") directly and exclusively update all subsequent actions, allowing the agent to correct its plan when previous attempts fail. In diffusion decoding, however, full bidirectional attention is applied across the entire sequence, diluting the influence of any single observation and weakening the model's sensitivity to local feedback. As a result, the model may repeat the same action ("go to cabinet 1") despite receiving negative environmental feedback, leading to failure modes such as retry loops in embodied agent tasks.

C.2 Fuzzy

Definition *Fuzzy* refers to the tendency of dLLMs to produce structurally malformed outputs due to their continuous-to-discrete generation process. Unlike AR decoding, which enforces local syntactic constraints token by token, dLLMs iteratively refine continuous latent representations of the entire sequence before projecting to discrete tokens, making them prone to schema violations such as malformed function calls or broken key-value syntax.

Theoretical Explanation As shown in Table 5, unlike AR decoding, where the causal attention mask ensures each structural token (e.g., `=`, `"`) is uniquely constrained by its prefix, diffusion decoding applies bidirectional attention across the entire sequence, lacking position-specific syntactic constraints. This leads to structurally inconsistent outputs — the model produces garbled syntax (`query','':': 'Shape of You'`) instead of the valid schema (`query="Shape of You"`), rendering the function call unparseable despite correctly identifying the target function and argument.

D Discussion of Other dLLMs and Optimization Techniques

This appendix discusses three additional questions raised during the rebuttal stage that are closely related to the scope and interpretation of our main results.

D.1 Can dLLM Optimization Techniques Solve Failures in Agentic Workflows?

Motivation Our main experiments focus on a restricted set of dLLM optimization strategies (e.g.,

dLLM	Inference	AlfWorld		ScienceWorld		BabyAI		BFCL
		Success	Progress	Success	Progress	Success	Progress	Single-Live
Dream-7B	Vanilla	0.7	6.0	0.6	5.3	8.9	14.8	1.5
	APD	0.0	10.9	0.0	3.3	4.5	9.4	31.4
	D2F	2.9	13.8	2.2	6.7	9.8	23.6	34.3
FdLLM-7B	Vanilla	3.3	7.8	0.7	6.4	5.4	12.6	0.0
	DCD	10.4	30.8	0.0	6.4	8.0	13.3	30.7

Table 6: **Comparison of recent dLLM optimization techniques** on embodied and tool-calling tasks. We report Success Rate (%) and Progress Rate (%) on ALFWorld, ScienceWorld, and BabyAI, and Accuracy (%) on BFCL Single-Live.

FastdLLM and DLLMVar). To broaden this coverage, we further incorporate several recent decoding advances as supplementary experiments and examine whether they materially change the conclusions of our main study.

Settings For clarity, we list the relevant methods below together with the implementation settings adopted in our evaluation:

- **APD** (Israel et al., 2025): We implement Dream-7B + APD using the balanced configuration to preserve both performance and efficiency ($R = 0.7$, $W = 16$, $M = 100$), with Qwen2.5-0.5B as the approximate AR model.
- **DCD** (Shu et al., 2026): We adopt DCD with FdLLM-7B (dual cache), corresponding to the best-performing configuration reported in the original paper.
- **D2F** (Wang et al., 2025b): We apply D2F on top of Dream-7B, as this combination achieves the strongest performance in the original study.

Results From Table 6, we observe that on embodied tasks, APD and D2F yield only limited gains for Dream-7B. On ALFWorld, they modestly improve progress rate while leaving success rate low, whereas on ScienceWorld and BabyAI, D2F is consistently stronger than APD. For FdLLM-7B, DCD substantially improves ALFWorld and BFCL performance, but its gains on ScienceWorld and BabyAI remain limited. On tool-calling tasks, APD, D2F, and DCD all raise the original baseline to above 30% accuracy, placing them in a similar performance tier to the stronger dLLM variants in our main results. Overall, although these techniques lead to notable improvements, a substantial gap between dLLMs and AR LLMs remains, and our overall conclusion still holds.

D.2 Can Agent-Level Optimization Improve dLLM Agent Performance?

Motivation Our main experiments do not consider two related directions. First, we do not cover agentic workflows beyond standard ReAct-style execution, such as Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023). Second, we do not study the use of AR models as feedback signals to assist a dLLM agent, but only include AR models as sub-modules within the DiffuAgent framework. A reasonable concern is that if AR feedback can help a dLLM agent converge to a reasonable multi-step action chain, then the weakness of dLLMs may reflect unstable planning rather than a systematic defect. We emphasize that incorporating techniques such as multi-trial refinement (e.g., self-refine) or in-trajectory autoregressive feedback is beyond the main scope of this paper, since it shifts the focus from evaluating native dLLM behavior to optimizing agent-level performance and also reduces the runtime efficiency that motivates the use of dLLMs. Nevertheless, examining such settings provides a more complete validation.

Settings To examine this concern, we conduct additional experiments on the ALFWorld test set. We use DVar-8B as the primary agent backbone and Qwen-8B (Yang et al., 2025a) as the AR refinement or feedback module. We consider three augmentation settings:

1. AR self-refine with up to 2 trials, where the AR model performs reflection and replanning based on the trajectory generated by the dLLM;
2. AR feedback every 5 steps within a single trajectory;
3. AR feedback at every step, forming a tightly coupled hybrid loop.

Agent Optimization Method	SR (%)	PR (%)
DVar-8B (Baseline)	0.7	10.0
+ AR Self-Refine (2 trials)	1.5	13.5
+ AR Feedback (every 5 steps)	2.2	15.2
+ AR Feedback (every step)	2.2	15.4

Table 7: **Comparison of agent-level optimization methods** on ALFWorld. We report Success Rate (%) and Progress Rate (%) with DVar-8B as the primary dLLM backbone and Qwen-8B as the AR refinement or feedback module.

Model	Status	Eval.	Reward	Pass	Infra
Qwen-8B	valid	8	0.25	0.25	1
FdLLM-7B + DCD	valid	9	0.00	0.00	0
Dream-7B + D2F	invalid	0	-	-	9
DVar-8B	valid	9	0.00	0.00	0

Table 8: **Results on Tau-Bench mock.** We report the number of evaluated tasks, average reward, pass rate, and infrastructure errors after excluding one broken task from the original 10-task split, leaving 9 effective tasks.

Results Table 7 shows that these agent-level optimization strategies do improve embodied performance, but the gains remain limited. AR self-refine raises the success rate from 0.7% to 1.5%, while periodic or step-wise AR feedback increases it to 2.2%. The progress rate also rises from 10.0% to 15.2–15.4%. These results suggest that external autoregressive refinement can partially mitigate the identified failure modes, but it still does not close the gap.

D.3 Can the Bitter Lesson Generalize to Other Agentic Benchmarks?

Motivation Our main experiments focus on BFCL as the primary tool-calling benchmark and therefore do not directly cover conversational multi-turn tool-use settings such as Tau-Bench (Yao et al., 2024). In our preliminary attempts on Tau-Bench and SWE-Bench (Jimenez et al., 2023), we found that most samples exceed the context window of the tested dLLMs, making direct evaluation invalid. Therefore, we only report Tau-Bench mock, the lightest Tau-Bench domain, as a basic sanity-check benchmark for lightweight interactive tool use, and leave broader evaluation on larger Tau-Bench domains and SWE-Bench to future work.

Settings We evaluate on the Tau-Bench mock with max_steps=20 and max_errors=5. Since the original split contains one broken task (update_task_with_user_tools), we exclude it and report results on the remaining 9 tasks. We compare Qwen-8B, FdLLM-7B with DCD, Dream-7B with D2F, and DVar-8B.

Results Table 8 shows that Qwen-8B is the only model that achieves a non-zero score on Tau-Bench mock, indicating that the benchmark pipeline and text-based tool-calling fallback are workable. In contrast, FdLLM-7B with DCD and DVar-8B both finish valid runs but remain at zero pass rate, typically producing repetitive natural-language responses instead of entering the required tool-calling protocol. Dream-7B with D2F fails earlier in the stack, with all 9 tasks ending as infrastructure errors because the benchmark receives assistant turns with neither usable content nor tool calls. Overall, these results suggest that the main bottleneck is not only serving stability, but also policy following and action formatting under the Tau-Bench interaction loop, which further strengthens our bitter lesson.

D.4 Can Schema-Checking Methods Improve dLLM Tool Calling?

Motivation Our main experiments do not separately evaluate whether simple schema-checking heuristics or lightweight structural guardrails could materially close the tool-calling gap. A natural concern is that if such post-hoc repairs were already sufficient to recover most failures, then the observed weakness of dLLMs might mainly come from superficial formatting errors rather than deeper decoding problems.

Settings To assess whether lightweight structural guardrails could close the tool-calling performance gap, we conduct an additional controlled study on 100 uniformly sampled BFCL outputs across four dLLMs. We consider three simple guardrails:

- G1: Repetition truncation, which removes repetitive trailing content;
- G2: First-bracket extraction, which extracts the first complete bracketed structure from the raw output;
- G3: Dict-unpacking repair, which repairs malformed Python-style dict-unpacking patterns.

We also evaluate their combined version.

Results Table 9 shows that even when combining all three guardrails (G1+G2+G3), only 21%

Guardrail	Parse (%)	Semantic (%)	Fail (%)
G1: Truncate	9	8	92
G2: Extract	20	12	88
G3: Repair	1	1	99
G1+G2+G3	21	14	86

Table 9: **Results of lightweight schema guardrails on BFCL outputs.** Each number denotes the percentage of sampled cases in the corresponding category over 100 BFCL outputs. Parse denotes syntactic recovery, Semantic denotes semantic correctness, and Fail denotes the remaining failures. G1 denotes truncation, G2 denotes first-bracket extraction, and G3 denotes dict-unpacking repair.

of BFCL outputs achieve syntactic recovery and only 14% reach semantic correctness, while 86% still fail. Among these remaining failures, 79% still break at the parsing stage due to deeply corrupted or incomplete AST structures that cannot be repaired by local structural heuristics, while the remaining 7% are syntactically valid but semantically incorrect, such as wrong function names, incorrect argument values, or mismatched argument counts. We further note that BFCL already evaluates schema adherence through its built-in `ast_decoder` verification. This means the benchmark itself already provides a strong structural validity check, and our additional analysis shows that the dominant failure modes arise from deeper decoding and semantic errors rather than superficial format violations.

E Detailed Results on the BFCL-v3 Benchmark under the DiffuAgent Framework

To facilitate clearer comparisons and ensure reproducibility, we report detailed statistical results on the BFCL-v3 benchmark in Table 10, reported separately for the "Standard" and "Challenge" categories. The results are consistent with Figure 7.

F Prompt Context

F.1 Embodied Agentic Workflow

ReAct Prompt Following Chang et al. (2024), we use the provided task instruction, task goal, and in-context example for each dataset. As Chang et al. (2024) adopt an act-only prompting style rather than ReAct prompting, we follow Song et al. (2024) to design a ReAct-style prompt format. For ALFWorld and ScienceWorld, we include valid action lists for these two datasets to ensure fair comparison with prior work (Song et al., 2024; Fu et al., 2025a). See the corresponding `PROMPT` with AlfWorld as an example.

DiffuAgent: Memory For memory-augmented agents, we use *gpt-5.1-2025-11-13* to construct 3 memory `EXEMPLARS` by transforming original ReAct trajectories, and using a direct `PROMPT` for summarizing the previous memory.

DiffuAgent: EarlyExit We follow (Lu et al., 2025) to use the same early-exit `PROMPT` for external verification.

F.2 Tool-Calling Agentic Workflow

Tool-Calling Prompt For the experiments in this work, we follow BFCL (Patil et al., 2025) and adopt their `PROMPT` without any modification to ensure faithful reproduction. Although Qwen-8B provides a dedicated tool-calling mode, we do not use it in this work, as our initial experiments indicate that many tools provided in BFCL do not strictly adhere to the requirements of the tool-calling mode.

DiffuAgent: Tool Selector We employ a tool selector prior to tool-call generation; the corresponding prompt is provided in `PROMPT`. The selector takes the available functions and the interaction history as input, and outputs a subset of tools to be selected. We then update the tool descriptions to include only this selected subset.

DiffuAgent: Too-Call Editor We introduce a tool editor to refine generated tool calls; the corresponding prompt is provided in `PROMPT`. It is prompted with examples of various broken tool-call cases and outputs either a corrected tool call, "UNCHANGED" if the original call is valid, or "NO_VALID_TOOL_CALLS" when the output contains only textual explanations.

Qwen-8B Agent					Ministral-8B Agent				
Modules		BFCL Multi-Turn (%)			Modules		BFCL Multi-Turn (%)		
Selector	Editor	Standard	Challenge	Avg.	Selector	Editor	Standard	Challenge	Avg.
Agent + Selector									
Qwen-8B	-	**26.0**	**16.0**	**18.5**	Ministral-8B	-	**18.0**	**12.0**	**13.5**
Llada-8B	-	16.0	11.3	12.5	Llada-8B	-	16.0	10.7	12.0
Dream-7B	-	14.0	12.7	13.0	Dream-7B	-	12.0	10.0	10.5
FdLLM-7B	-	16.0	8.0	10.0	FdLLM-7B	-	12.0	6.0	7.5
DVar-8B	-	20.0	8.7	11.5	DVar-8B	-	2.0	4.7	4.0
Agent + Editor									
-	Qwen-8B	**26.0**	12.0	15.5	-	Ministral-8B	**14.0**	**17.3**	**16.5**
-	Llada-8B	20.0	**14.7**	**16.0**	-	Llada-8B	12.0	12.7	12.5
-	Dream-7B	16.0	12.0	13.0	-	Dream-7B	12.0	11.3	11.5
-	FdLLM-7B	2.0	2.0	2.0	-	FdLLM-7B	0.0	0.7	0.5
-	DVar-8B	0.0	0.0	0.0	-	DVar-8B	0.0	0.0	0.0
Agent + Selector + Editor									
Qwen-8B	Qwen-8B	**28.0**	**15.3**	**18.5**	Ministral-8B	Ministral-8B	**20.0**	**12.7**	**14.5**
Llada-8B	Llada-8B	16.0	10.0	11.5	Llada-8B	Llada-8B	16.0	10.0	11.5
Dream-7B	Dream-7B	22.0	9.3	12.5	Dream-7B	Dream-7B	12.0	11.3	11.5
FdLLM-7B	FdLLM-7B	2.0	0.7	1.0	FdLLM-7B	FdLLM-7B	2.0	0.7	1.0
DVar-8B	DVar-8B	0.0	0.0	0.0	DVar-8B	DVar-8B	0.0	0.0	0.0

Table 10: **Ablation performance of tool-calling agents** on BFCL-v3 Multi-Turn benchmark, evaluated with different backbone models for the agent modules (*Selector* and *Editor*).

Test Set	#Tasks	Example Task & Trajectory
AlfWorld	134	**Task:** *Look at the bowl under the desklamp.* **T:** I should check the desks to see if there's a desklamp and a bowl nearby. I'll start with desk 1. **A:** go to desk 1 **O:** On the desk 1, you see a desklamp, a keychain, a mug, pens, and a pencil. **T:** I found a desklamp on desk 1. I should check if there's a bowl nearby. **A:** examine desk 1 **O:** No bowl found on desk 1. **T:** I didn't find a bowl on desk 1. I'll check desk 2 next. **A:** go to desk 2 **O:** On the desk 2, you see an alarm clock, a bowl, and CDs. **T:** I found a bowl on desk 2 and can use the desklamp from desk 1. **A:** take bowl from desk 2 **O:** You pick up the bowl from desk 2. **T:** I need to return to desk 1 where the desklamp is. **A:** go to desk 1 **O:** The desklamp is available on desk 1. **T:** I should use the desklamp to look at the bowl. **A:** use desklamp **O:** You turn on the desklamp.
ScienceWorld	90	**Task:** *Find the animal with the longest life span.* **T:** I need to focus on the animal with the longest lifespan. Since the bee hive is open and contains multiple adult bees, I should focus on the bees first. **A:** focus on adult bee **O:** Ambiguous request: multiple adult bees are present in the bee hive. **T:** I need to determine which animal has the longest lifespan. I will focus on one bee to gather information. **A:** focus on adult bee (in bee hive, in greenhouse) **O:** Unknown action. **T:** I need to examine the bee to determine its lifespan. I will try to look at the bee for more information. **A:** look at adult bee (in bee hive, in greenhouse) **O:** No known action matches that input. **T:** I need to inspect the bee hive more closely to determine information about the bees. **A:** look in bee hive **O:** Inside the bee hive are several adult bees. **T:** The bee hive does not provide lifespan information. I should explore other locations for animals with longer lifespans. **A:** go to outside **O:** You move to the outside. **T:** I need to explore the outside location to find animals with long lifespans. **A:** look around **O:** You see various animals, including a baby dragonfly, a baby wolf, and a giant tortoise egg. **T:** Tortoises are known for long lifespans, so I should examine the giant tortoise egg first. **A:** focus on giant tortoise egg **O:** You focus on the giant tortoise egg.
BabyAI	112	**Task:** *go to the red ball.* **T:** I need to find the red ball, so I'll start by exploring forward. **A:** move forward **O:** There is a barrier in front of you, you can't move forward. **T:** There's a wall ahead. I'll turn to look for a clearer path toward the red ball. **A:** turn right **O:** In front of you in this room, you can see several objects: The room has walls around you. You are facing a wall 2 steps away. You are not carrying anything. **T:** There's more space in this direction. I'll move forward to see if I can spot the red ball. **A:** move forward **O:** In front of you in this room, you can see several objects: The room has walls around you. You are facing a wall 1 steps away. You are not carrying anything. **T:** This wall is close. I'll try turning again to scan for the red ball. **A:** turn left **O:** In front of you in this room, you can see several objects: The room has walls around you. You are facing a wall 1 steps away. You are not carrying anything. **T:** I need to find a different direction to explore since turning left and right is not allowed. I'll try moving forward again to see if there's a new path. **A:** move forward **O:** There is a barrier in front of you, you can't move forward. **T:** I need to find a different direction to explore since moving forward is blocked. I'll try turning left to see if there's a new path. **A:** turn left **O:** In front of you in this room, you can see several objects: There is a grey box 1 3 steps in front of you and 1 steps to your left. There is a grey ball 1 2 steps in front of you and 2 steps to your left. There is a red ball 1 right in front of you 5 steps away. There is a grey key 1 4 steps in front of you and 1 steps to your left. The room has walls around you. You are facing a wall 6 steps away. You are not carrying anything. **T:** I can see the red ball now! Let's head toward it. **A:** go to red ball 1 **O:** In front of you in this room, you can see several objects: There is a red ball 1 right in front of you 1 steps away. There is a grey key 1 0 steps in front of you and 1 steps to your left. The room has walls around you. You are facing a wall 2 steps away. You are not carrying anything. The task is completed.

Table 11: **Overview of embodied agent tasks in the AgentBoard (Chang et al., 2024) benchmark**. In the example trajectories, "T", "A", and "O" denote **T**hought, **A**ction, and **O**bservation, respectively.

Categories I (This Work)	**Categories II** (Original Testset)	**#Used** (#Original)	**Example & Tool-Call**
			Non-Live (300)
Non-Live	simple	50 (400)	**U:** Convert 150 Euros to Canadian dollars. **T:** currency_conversion.convert **TC:** [currency_conversion.convert(amount=150, from_currency='EUR', to_currency='CAD')]
	java	50 (100)	**U:** Can I determine if the symbol 'getVersion' is readable in the native function interface library associated with the current object? **T:** NFILibrary.isMemberReadable **TC:** [NFILibrary.isMemberReadable(symbol='getVersion')]
	javascript	50 (50)	**U:** Help me reset a state property called 'userSession' to 'null' in a React component? **T:** resetStateProperty **TC:** [resetStateProperty(stateProperty=üserSession)]
	multiple	50 (200)	**U:** What's the area of a circle with a radius of 10? **T:** geometry.area_circle, plot_sine_wave **TC:** [geometry.area_circle(radius=10)]
	parallel	50 (200)	**U:** How to save game progress at stage 7 in easy mode and stage 3 in hard mode? **T:** game.save_progress **TC:** [game.save_progress(stage=7, mode='easy'), game.save_progress(stage=3, mode='hard')]
	parallel_multiple	50 (200)	**U:** Invest $2000 in Google and withdraw $1000 from Apple. **T:** investment.withdraw, investment.invest **TC:** [investment.invest(company="Google", amount=2000), investment.withdraw(company="Apple", amount=1000)]
			Single-Turn Live (140)
S.	live_simple	50 (258)	**U:** Order me pizza. **T:** ChaFod **TC:** [ChaFod(TheFod="PIZZA")]
M.	live_multiple	50 (1053)	**U:** Can you find me a Family Counselor in Gilroy? **T:** Services_4_BookAppointment, Services_4_FindProvider, Weather_1_GetWeather **TC:** [Services_4_FindProvider(city="Gilroy, CA", type="Family Counselor")]
P.	live_parallel	16 (16)	**U:** What's the snow like in the two cities of Paris and Bordeaux? **T:** get_snow_report **TC:** [get_snow_report(location="Paris, France"), get_snow_report(location="Bordeaux, France")]
PM.	live_parallel_multiple	24 (24)	**U:** interviewers list for Python and Java **T:** get_interviewer_list, review_of_interviewer **TC:** [get_interviewer_list(skill="Python"), get_interviewer_list(skill="Java")]
			Multi-Turn (200)
Standard	multi_turn_base	50 (200)	**U:** Hey, I've just filled my car up with 13.2 gallons of fuel. How much is that in liters? **TC:** [gallon_to_liter(gallon=13.2)] **U:** Once you've converted it to liters, please guide me in filling the tank to the max limit. After that's sorted, I need to ensure that all the doors are securely locked and the parking brake is firmly set. **TC:** [fillFuelTank(fuelAmount=36.8), lockDoors(unlock=False, door=["driver", "passenger", "rear_left", "rear_right"]), activateParkingBrake(mode="engage")] ...
Challenge	multi_turn_miss_func	50 (200)	**U:** I've been thinking of visiting Autumnville for a while now, but I'm not sure how far it is from here in Crescent Hollow. Can you help me figure this out so I can plan my trip accordingly? **TC:** [get_zipcode_based_on_city(city="Crescent Hollow"), get_zipcode_based_on_city(city="Autumnville"), estimate_drive_feasibility_by_mileage(distance=100)], [estimate_distance(cityA="69238", cityB="51479")] **U:** Oh, and by the way, there's something else I need help with. I want to calculate the logarithm of the distance you've just told me about, considering a base 10 with a precision of 5 digits. Could you provide me with this value as well? **TC:** [logarithm(value=630.0, base=10, precision=5)]]
	multi_turn_miss_param	50 (200)	**U:** Hey there, I noticed that all of my car doors seem to have locked themselves up, and with my schedule being pretty tight today, I'm in quite a pinch. I could really use your help to get some of them unlocked. **TC:** [lockDoors(unlock=True, door=["driver", "passenger", "rear_left", "rear_right"])], [lockDoors(unlock=False, door=["driver", "passenger", "rear_left", "rear_right"])] **U:** I mean could you help me get all those doors unlocked? It'd be fantastic if you could also switch on the headlights. It's getting a bit darker out here than expected, and visibility's not great! **TC:** [lockDoors(unlock=True, door=["driver", "passenger", "rear_left", "rear_right"]), setHeadlights(mode="on")]
	multi_turn_long_context	50 (200)	**U:** It'd be great if you could pop Zeta Corp's stock onto my watchlist. I've come across some fascinating insights into their recent performance that I want to monitor. **TC:** [add_to_watchlist(stock="ZETA")] **U:** With Zeta Corp's stock now on my radar, let's pull up the complete list of stocks I'm watching. I want to double-check that all my chosen stocks are properly listed for my review. **TC:** [get_watchlist()]
			Hallucination (118)
Rel.	live_relevance	18 (18)	**U:** Hi, could you get me a house to stay for 4 in London? **T:** Hotels_2_BookHouse, Hotels_2_SearchHouse **TC:** [Hotels_2_SearchHouse(where_to="London, UK", number_of_adults=4)]
Irrel.	irrelevance	50 (240)	**U:** What defines scientist? **T:** get_historical_figure_info
	live_irrelevance	50 (882)	**U:** dsfsdf **T:** calculate_tax

Table 12: **Overview of tool-calling tasks in the BFCL benchmark** (Patil et al., 2025). "U", "T", and "TC" denote the **U**ser message, available **T**ools, and the generated **T**ool **C**all, respectively.

Embodied Agent: ReAct-Style

SYSTEM:
You are a helpful assistant.
USER:
Your task is to interact with a virtual household simulator to accomplish a specific task. With each interaction, you will receive an observation. Your role is to ... {task instruction}

Here is the example: {example}

Now, it's your turn. You should perform thoughts and actions to accomplish the goal. Your response should use the following format:

Thought: <your thoughts>
Action: <your next action>

Your task is: {task goal}
You are in the middle of a room. Looking quickly around you, ... {init observation}

{interaction history}

The next action could be chosen from these valid actions: {valid actions}

DiffuAgent: 3 Memory Exemplars

Example 1:
Memory: (empty)

Thought: I should check nearby storage spaces for a spraybottle.
Action: go to cabinet 1
Observation: On the cabinet 1, you see a cloth 1, a soapbar 1, a soapbottle 1.

Example 2:
Memory: The agent has checked cabinet 1 and found no spraybottle, then moved to cabinet 2 and discovered it was closed. Cabinet 2 is still unopened and uninspected inside.

Thought: I need to open this cabinet to see what's inside.
Action: open cabinet 2
Observation: You open the cabinet 2. The cabinet 2 is open. In it, you see a candle 1, and a spraybottle 2.

Example 3:
Memory: The agent has searched cabinet 1 (without finding a spraybottle), opened cabinet 2 (where the spraybottle was found), picked up spraybottle 2, and moved to the toilet 1 (where a soapbottle was present).

Thought: It's time to place the spraybottle on the toilet to complete the task.
Action: put spraybottle 2 in/on toilet 1
Observation: You put the spraybottle 2 in/on the toilet 1.

DiffuAgent: Pre-hoc Memory Update

SYSTEM:
You are a memory updater. Update the memory_str to reflect what the agent has done and learned so far. Include important actions taken, locations visited, and key observations. Keep the summary concise, chronological, and consistent. Do not invent new facts or omit relevant past actions. Write the memory in third-person, concise past tense, like a mission log.

USER:
Memory_str: {previous memory_str}
Recent_steps: {recent interaction steps}

Please output the updated Memory_str only — a short narrative summary of what has been done and observed so far. No explanations or formatting other than plain text.

Memory_str:

DiffuAgent: Post-hoc Exit Verification

SYSTEM:
You are a helpful assistant.
USER:
You will be given a historical scenario in which you are placed in a specific environment with a designated objective to accomplish.
Task Description: Your task is to interact with a virtual household simulator to accomplish a specific task. With each interaction, you will receive an observation.

Your role is to ... {task instruction}
Your Objective:
{task goal}
Your Current History:
{interaction history}
Instructions:
{extrinsic early-exit instruction}
Do not include any additional text or explanations in your response.

Tool-Call Agent: BFCL default Prompts

SYSTEM:
You are an expert in composing functions. You are given a question and a set of possible functions. Based on the question, you will need to make one or more function/tool calls to achieve the purpose.
If none of the functions can be used, point it out. If the given question lacks the parameters required by the function, also point it out. You should only return the function calls in your response.

If you decide to invoke any of the function(s), you MUST put it in the format of [func_name1(params_name1=params_value1, params_name2=params_value2...), func_name2(params)]

You SHOULD NOT include any other text in the response.

At each turn, you should try your best to complete the tasks requested by the user within the current turn. Continue to output functions to call until you have fulfilled the user's request to the best of your ability. Once you have no more functions to call, the system will consider the current turn complete and proceed to the next turn or task.

Here is a list of functions in JSON format that you can invoke. {function descriptions}

USER:
{user message}

DiffuAgent: Pre-hoc Tool Selection

SYSTEM:
You are a tool selector for a function-calling agent.

Task:
Given a user message ([User Message]), the previous tool call ([Tool Call]) and its results ([Tool Execution Results]), you must select a minimum of 3 distinct functions from the provided list.

Rules:

- Output at least 3 function names, and no more than 10 functions.
- Use ONLY names from the provided function list.
- Output ONLY function names. No explanations or extra text.
- Prioritize the [USER MESSAGE] above all else; use previous tool calls and results only as supplementary context.

USER:
Functions:
{available functions}

{interaction history: user message, tool calls, tool execution results}

Selected Functions:

DiffuAgent: Tool-Call Editor

SYSTEM:
You are a strict tool-call format auditor and repairer.
Your task: Repair or validate a broken tool-call and output a final call that strictly follows TOOL_CALL_FORMAT.
Rules:

- If the tool-call is already valid and correct, output UNCHANGED.
- If the tool-call is textual explanations, output NO_VALID_TOOL_CALLS.
- If the tool-call contains both explanations and tool-calls, remove the expla-

nations and correct the tool-calls.

- If the tool-call does not conform to TOOL_CALL_FORMAT, repair any format or schema errors and output the corrected tool-call only; do not invent functions or parameters.

TOOL_CALL_FORMAT:
[func_name1(param_name1=param_value1, param_name2=param_value2, ...), func_name2(param_name3=param_value3, ...)]

Examples:
BROKEN_TOOL_CALL 1:
[cd(folder="academic_venture")]
Output: UNCHANGED

BROKEN_TOOL_CALL 2:
cd(folder="academic_venture")
Output: [cd(folder="academic_venture")]

BROKEN_TOOL_CALL 3:
"cd": "folder": "academic_venture"
Output: [cd(folder="academic_venture")]

BROKEN_TOOL_CALL 4:
The task is now complete.
Output: NO_VALID_TOOL_CALLS

BROKEN_TOOL_CALL 5:
The task is now complete. The final tool-call is "ls":
Output: [ls()]

USER:
BROKEN_TOOL_CALL (to be audited and possibly corrected):
{model response}

Now produce the final output according to the rules above. No explanations, markdown, or extra text.

Output: