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

Anonymous ACL submission

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 systematically 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.

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.

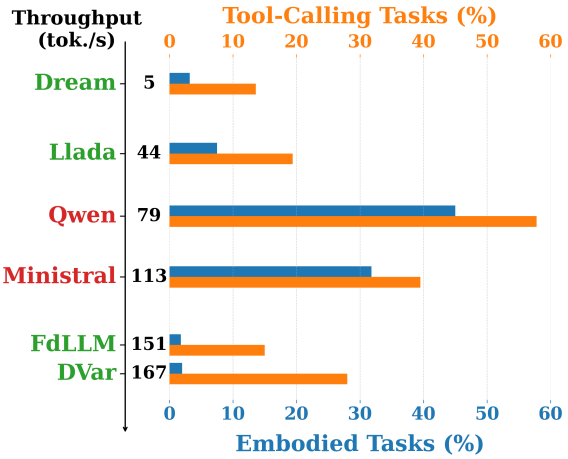

Figure 1: **Performance–Efficiency Trade-offs in Embodied and Tool-Calling Tasks.** Despite higher inference efficiency, `FdLLM-7B` and `DVar-8B` do not guarantee comparable agentic performance to autoregressive LLMs. `Llada-8B` and `Dream-7B` fall behind LLMs in both task performance and efficiency.

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 (Patil et al., 2025) reveal a ***bitter lesson***: current dLLMs fail to serve as reliable agentic backbones,

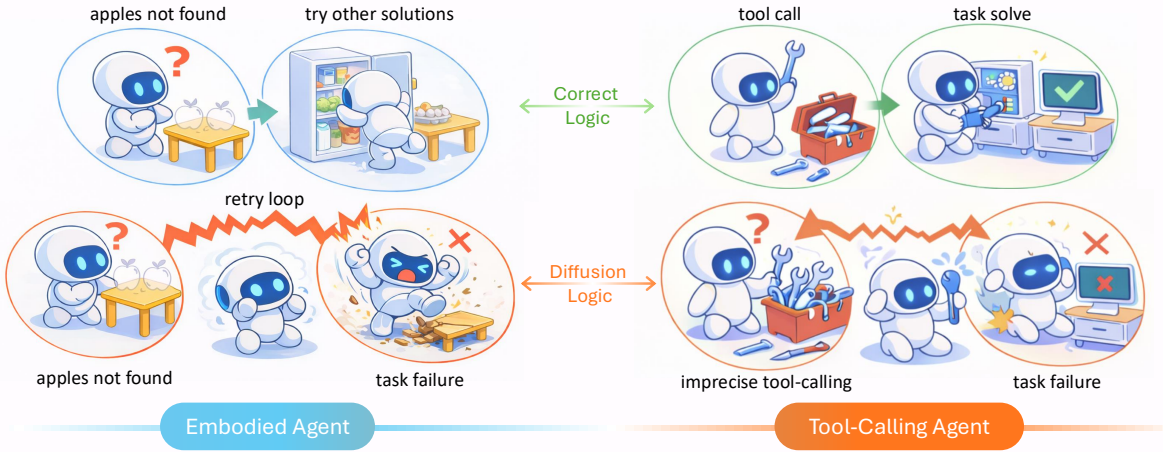

Figure 2: **An overview of systematic failure modes** in dLLMs. In embodied agent settings, dLLMs tend to repetitively retry the same action instead of exploring alternative plans. In tool-calling agent settings, imprecise or unstable tool invocation further leads to execution failures.

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., 2025), 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:

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

where trajectory $e_{1:t-1} = (a_1, o_1, \ldots, a_{t-1}, o_{t-1})$ denotes past actions and observations, and u_{task} represents task-specific context.

To elicit reasoning capabilities, ReAct (Yao et al., 2023) is a widely adopted agentic workflow that synergizes planning and decision-making in multi-turn interactions, where the LLM generates an intermediate thought before producing the next action. This process can be formulated as

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

with q_t denoting the thought generated at time step t. In this work, we adopt ReAct-style prompting when evaluating embodied agents.

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 embodied-agent paradigm. In each interaction, the agent processes the user request u_{user} together with a set of available tool descriptions $\mathcal{D} = \{\tau_1, \ldots, \tau_N\}$, and generates one or more structured tool invocations. This process can be formulated 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 its corresponding 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 $\mathcal{O}$ 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 provide a detailed summarization of their decoding strategies in Appendix A.

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[1].

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 for fast response[2], and `Ministral-8B` (Jiang et al., 2024), an instruction-tuned 8B model. Both models are evaluated in text-only settings[3].

Diffusion-based LLMs We employ four recent dLLMs in our study: `Llada-8B` (Nie et al., 2025), a strong diffusion LLM with competitive general performance; `Dream-7B` (Ye et al., 2025), which uses 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 variable-length generation via native EOS prediction.

[1] For multi-turn scenarios, we adopt "Standard" denoting the `multi_turn_base` category, while "Challenge" reports the average performance over missing-parameter, missing-function, and long-context settings.

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

[3] 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**	**85.6**	**26.7**	**55.1**	32.1	45.7	**45.0**	**62.1**
Ministral-8B	45.5	66.2	13.3	52.0	**36.6**	**46.6**	31.8	54.9
Llada-8B	5.2	18.5	1.1	8.6	16.1	22.0	7.5	16.4
Dream-7B	0.7	6.0	0.0	5.3	8.9	14.8	3.2	8.7
FdLLM-7B	0.0	4.4	0.0	6.4	5.4	12.6	1.8	7.8
DVar-8B	0.7	10.0	0.0	1.9	5.4	15.0	2.0	8.9

Table 1: **Comparison of Success Rate (%) and Progress Rate (%)** across different **LLMs** and **dLLMs** on three *Embodied* tasks. Best results are highlighted in **bold**.

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., and P. denote simple, multiple, and parallel tool-calling tasks, respectively. Hallucination indicates whether a tool call is required (Rel.) or not required (Irrel.).

Device and Model Deployment All experiments are conducted on two NVIDIA A800 GPUs (80GB each). Auto-regressive models (`Qwen-8B` , `Ministral-8B`) are deployed with vLLM (Kwon et al., 2023) and accessed via OpenAI-compatible chat-completion APIs (Achiam et al., 2023). Diffusion LLMs (`Dream-7B` , `Llada-8B` , `FdLLM-7B`) are reproduced using NVIDIA Fast-dLLM[4] and served through FastAPI for high-throughput inference.

4 Failure of dLLM as Agent Backbone

4.1 The 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 in ScienceWorld. 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.

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

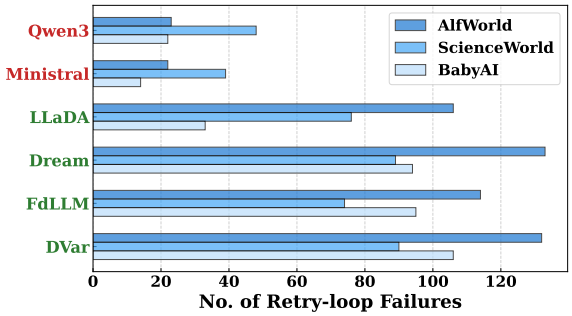

Figure 3: **Comparison of retry-loop failures across LLMs and dLLMs**. A retry-loop is defined as repeatedly executing the same action for more than three consecutive steps during task completion.

Retry Loops as a Systematic Failure Mode Following Shinn et al. (2023), we define repetitive actions as three or more consecutive identical 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 patterns and experiences (Sun et al., 2025) by leveraging prior interactions to branch into new actions.

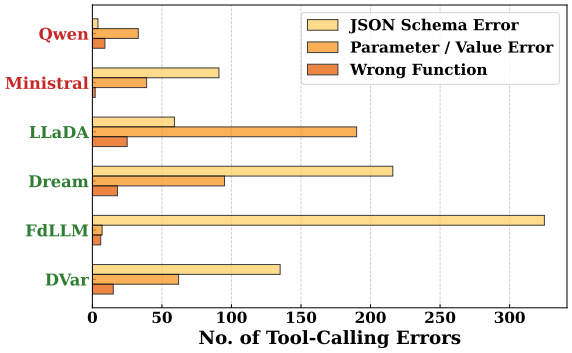

Figure 4: **Comparison of the number of tool-calling failure categories** across LLMs and dLLMs.

4.2 The 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.

dLLMs fail to generate precise tool-call formats We analyze tool-calling failures under single-turn Live and Non-Live settings by categorizing different error types. As shown in Figure 4, JSON schema errors and parameter/value errors dominate for both LLMs and dLLMs. However, dLLMs primarily 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 The Failure of Efficiency-Performance Trade-off

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 The 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 dependency and induces fuzzy intermediate states, hindering stable commitment to partial plans or structured outputs**. 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.

5 DiffuAgent: A Multi-Agent Evaluation Framework on Analyzing dLLM Behaviors

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, DiffuAgent isolates specific functions—such as memory, verification, and tool-related selection or format editing—rather than assigning dLLMs to the full agent loop, enabling fine-grained analysis without conflating module behavior with end-to-end 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

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

[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.

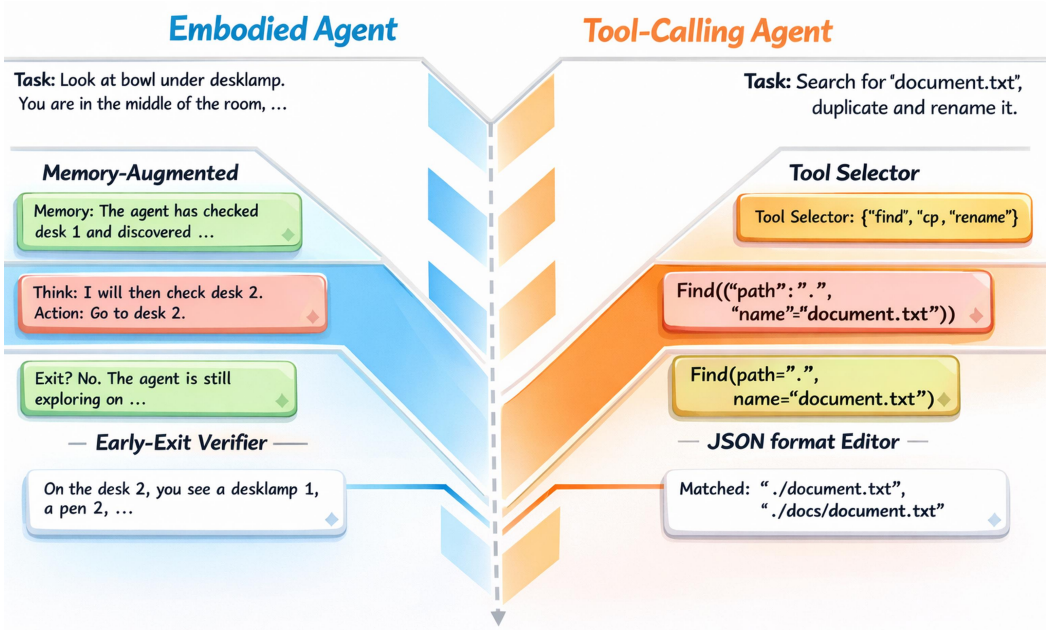

Figure 5: **Overview of DiffuAgent.** The framework integrates four modules. For embodied agents, we introduce a memory-augmented module for history compression and an early-exit verifier for global trajectory checking. For tool-calling agents, we include a tool selector over the library and a JSON format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

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.

Post-hoc: Early-Exit Verifier Following (Lu et al., 2025), we incorporate an early-exit verification module built on an LLM or dLLM backbone to assess an agent's self-awareness of being stuck. 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 can be formulated as:

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

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

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

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:

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

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:

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

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.

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.

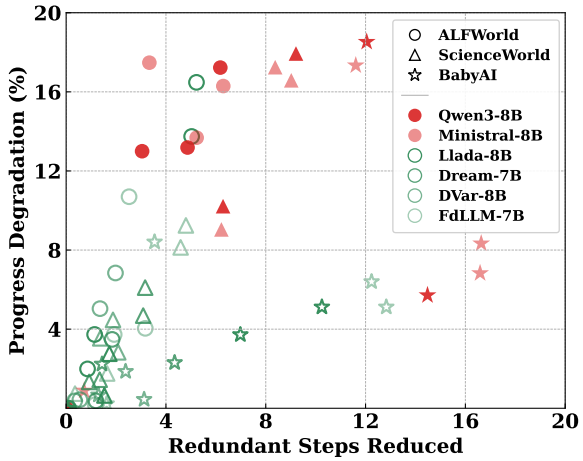

Figure 6: **Comparison of early-exit behavior across LLMs and dLLMs**. Filled and hollow markers denote LLM- and dLLM-based verifiers, respectively, while different marker shapes indicate different tasks.

6 Analysis on Agentic Behaviors on 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. 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

Based on memory-augmented trajectories, we select the four best-performing memory configurations[8] and apply early-exit verifiers implemented with different dLLMs. We evaluate redundancy reduction and progress degradation across embodied tasks (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

[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.

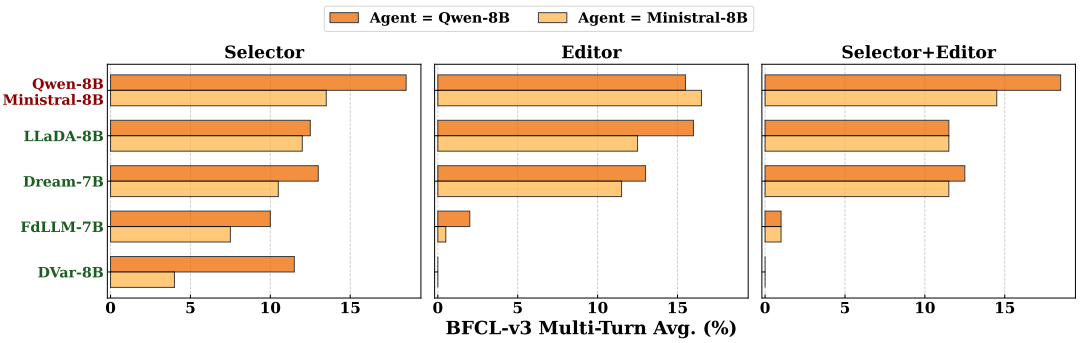

Figure 7: **Ablation performance of tool-calling agents** on BFCL-v3 Multi-Turn benchmark, evaluated with different backbone models for the agent modules (*Selector* and *Editor*). 0 indicates no successful instance.

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 and Table 4 in Appendix B, LLM-based modules consistently outperform dLLMs. Among dLLMs, `Llada-8B` and `Dream-7B` serve as relatively effective 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-8B` improves 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.

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). However, dLLM behavior under multi-turn, causally grounded agentic interaction remains underexplored; we therefore systematically study dLLMs as cognitive modules within agentic workflows.

LLM-based Agents LLM-based 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), AgentFLAN (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., 2025) recover agent capabilities under limited context windows. In contrast, we propose a multi-agent evaluation framework that treats dLLMs as cognitive modules.

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.

- **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.

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*.

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*.

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*.

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 arXiv:2510.07248*.

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*.

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*.

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*.

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

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. 2025. 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*.

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, 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 Introduction of dLLM Inference Strategies

This section provides a concise overview of the inference (decoding) strategies adopted by 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 sched-

ule 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

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

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 Detailed Results on BFCL-v3 Benchmark

We provide detailed results on the BFCL-v3 benchmark in Table 4, consistent with Fig. 7, for reference and reproducibility.

C Prompt Context

C.1 Embodied Agentic Workflow

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-style prompting, we follow Song et al. (2024) to design a ReAct-style prompt format. For ALFWorld and ScienceWorld, we observe that providing valid actions substantially affects performance (approximately 10%–20% in success rate). We therefore include valid action lists for these two datasets to ensure fair comparison with prior work (Song et al., 2024; Fu et al., 2025a). For memory-augmented agents, we use `gpt-5.1-2025-11-13` to construct three memory exemplars by transforming original ReAct trajectories.

C.2 Tool-Calling Agentic Workflow

For the main results, we follow BFCL (Patil et al., 2025) and use their prompts without any modification to ensure faithful reproduction. For DiffuAgent, we present the prompts used for the four agentic modules below.

ReAct-Style Prompt

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}
ASSISTANT: OK.
USER:

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}

Important ##: Your thought should be short, clear and concise.
{intrinsic early-exit instruction}

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

Memory Update Prompt

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:

Early-Exit Verification Prompt

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 Editor Prompt

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 explanations 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:

Tool Selection Prompt

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: