DLLM AGENT: SEE FARTHER, RUN FASTER

Huiling Zhen[1,*], **Weizhe Lin**[1,*], **Renxi Liu**[1], **Kai Han**[1], **Yiming Li**[1], **Yuchuan Tian**[1,4], **Hanting Chen**[1], **Xiaoguang Li**[1], **Xiaosong Li**[1], **Chen Chen**[1], **Xianzhi Yu**[1], **Mingxuan Yuan**[1], **Youliang Yan**[1], **Peifeng Qin**[1], **Jun Wang**[2], **Yu Wang**[3], **Dacheng Tao**[4], **Yunhe Wang**[1]
[1] Huawei Technologies Co., Ltd. [2] UCL [3] Tsinghua University [4] NTU [4] Peking University
$*$ Equal Contribution

ABSTRACT

Diffusion large language models (DLLMs) have emerged as an alternative to autoregressive (AR) decoding with appealing efficiency and modeling properties, yet their implications for agentic multi-step decision making remain underexplored. We ask a concrete question: when the generation paradigm is changed but the agent framework and supervision are held fixed, do diffusion backbones induce systematically different planning and tool-use behaviors, and do these differences translate into end-to-end efficiency gains? We study this in a controlled setting by instantiating DLLM and AR backbones within the same agent workflow (DeepDiver) and performing matched agent-oriented fine-tuning on the same trajectory data. This yields diffusion-backed *DLLM Agents* and directly comparable AR agents. Across agent benchmarks and case studies, we find that, at *comparable accuracy*, DLLM Agents are on average more than *30% faster* end to end than AR agents, with certain cases exceeding $8\times$ end-to-end speedup. Conditioned on solving the task correctly, DLLM Agents also require substantially fewer interaction rounds and tool invocations, consistent with higher planner hit rates that converge earlier to a correct action path and exhibit less backtracking. We further identify two practical considerations for deploying diffusion backbones in tool-using agents. First, naive DLLM policies are more prone to structured tool-call failures, necessitating stronger tool-call-specific training to reliably emit valid schemas and arguments. Second, for multi-turn agent inputs that interleave context and action spans, diffusion-style span corruption requires adjusting attention masking to avoid spurious context-action information flow; without such mask alignment, performance degrades. Finally, we analyze attention dynamics across workflow stages and observe paradigm-specific coordination patterns, providing evidence that diffusion-backed agents express stronger global planning signals during multi-step reasoning.

1 Introduction

Recent advances in large language models have led to the widespread adoption of language model agents, which extend single-shot generation to multi-turn reasoning, tool use, and structured workflows. Modern agent systems interact with external tools, maintain intermediate states, and revise decisions over multiple steps, forming complex action trajectories rather than isolated responses. As a result, agent performance is increasingly evaluated not only by final correctness, but also by efficiency, redundancy, and the structure of intermediate reasoning [1, 2].

Despite this shift, most existing agent research implicitly assumes that agent behavior is primarily determined by training data and agent workflow design, while the underlying generation paradigm of the language model plays a secondary role [3]. In practice, autoregressive (AR) decoding remains the de facto standard, and alternative generation paradigms are often treated as interchangeable components once embedded into a fixed agent framework. Under this assumption, improving agent performance is largely framed as a matter of better supervision, stronger base models, or more carefully engineered workflows [4, 5]. However, this perspective overlooks a fundamental question: *does the generation paradigm itself shape agent behavior in systematic ways that persist even under identical workflows and training data?*

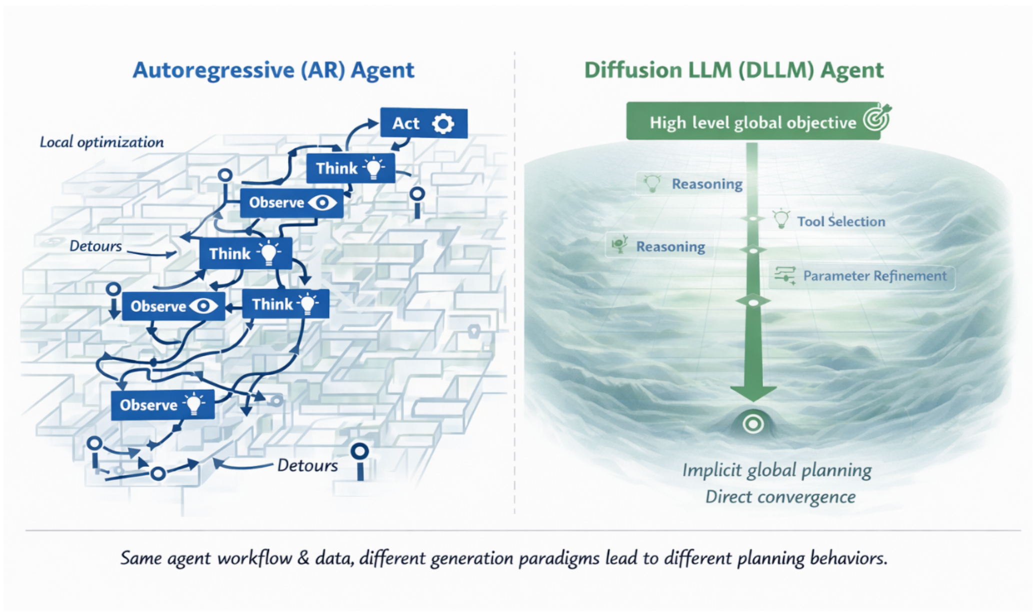

Autoregressive (AR) Agent
Local optimization
Act
Think
Observe
Detours
Think
Observe
Think
Observe
Detours
Diffusion LLM (DLLM) Agent
High level global objective
Reasoning
Tool Selection
Reasoning
Parameter Refinement
Implicit global planning
Direct convergence
Same agent workflow & data, different generation paradigms lead to different planning behaviors.

Figure 1: **Same agent workflow, different generation paradigms.** Under identical tool-use workflows and training data, DLLM Agents (diffusion) tend to exhibit earlier trajectory-level convergence and less redundant interaction, while AR agents more often follow incremental, locally conditioned decisions that can lead to detours and repeated tool calls.

In contrast to the AR paradigm, diffusion-based large language models (DLLMs) have emerged as a promising alternative paradigm for text generation. Unlike AR models, which generate tokens sequentially, DLLMs formulate generation as an iterative denoising process over the entire sequence, enabling repeated revision and global coordination [6]. Prior work has demonstrated the viability of diffusion for language modeling through masked discrete diffusion and latent diffusion formulations, achieving competitive perplexity and generation quality while enabling parallel decoding and iterative refinement [4, 7]. Subsequent studies further emphasize diffusion models' ability to revise early decisions and support global semantic control, including latent planning and structured paragraph generation [8, 9], soft-token evolution [10], and latent reasoning through continuous–discrete diffusion [11].

Notably, prior research on diffusion language models has primarily focused on single-shot generation settings, leaving their role in agent-based reasoning workflows largely unexplored [4, 7, 12]. This gap is important because agents are defined by trajectories of sequential decisions rather than isolated outputs. In practical agent workflows, both diffusion-based and autoregressive models generate actions step by step conditioned on accumulated interaction history [13, 14]. Nevertheless, their underlying training paradigms differ fundamentally. While AR models optimize strictly causal next-token prediction, DLLMs are trained through block-wise denoising with bidirectional attention within each block and iterative refinement across the sequence [15, 16].

Although bidirectional attention is locally constrained at the block level, the repeated denoising process enables *implicit global coordination through repeated local refinement*, allowing the model to progressively align long-range structure across blocks [17, 18]. Within each block, bidirectional context further promotes more consistent and well-formed action representations [18]. Together, these properties suggest that DLLMs can implicitly capture long-horizon dependencies and downstream consequences when producing individual decisions, whereas AR generation emphasizes immediate local commitments [5, 7]. From this perspective, paradigm differences manifest not in the external workflow structure, but in the *quality and global awareness of each local decision*. This can systematically influence planning efficiency, redundancy, and tool-use behavior, ultimately leading to distinct agent trajectories under otherwise identical workflows.

In this paper, we investigate how different generation paradigms translate into agent behavior, and introduce *Diffusion Large Language Model-based Agents (DLLM Agents)* by integrating diffusion models into standard agent workflows. We conduct a controlled comparison between diffusion-based and autoregressive language model agents by fine-tuning both paradigms on the same agent-oriented training data to equip them with comparable multi-turn dialogue and tool-use capabilities. Our analysis highlights consistent and systematic behavioral differences between DLLM Agents and their AR counterparts:

- **Earlier convergence to effective trajectories.** On tasks with known solutions, DLLM Agents tend to identify viable action paths earlier, committing to correct plans with fewer exploratory detours.
- **Reduced redundancy.** AR agents more frequently exhibit repeated reasoning steps and unnecessary tool invocations, whereas diffusion agents produce more compact and decisive interaction traces.
- **Workflow-level efficiency.** The advantages of DLLM Agents extend beyond token-level parallelism, manifesting as fewer interaction steps and shorter overall agent trajectories.
- **Distinct exploration behaviors on open-ended tasks.** On questions without a single ground-truth answer, diffusion agents tend to exhibit broader exploration and higher-level revision patterns, while AR agents follow more incremental, locally conditioned decision processes.

Importantly, these differences persist under identical workflows and training data, indicating that they arise from the generation paradigm itself rather than superficial decoding speedups.

Contributions. In summary, this work makes the following contributions:

- We present the first study of how diffusion and autoregressive generation paradigms differ at the level of agent behavior (Section 3).
- We demonstrate through extensive case studies (Section 4) and open-ended tasks that DLLM Agents achieve superior planning efficiency and reduced redundancy compared to AR-based agents.
- We analyze attention dynamics from an agent workflow perspective (Section 5), revealing that DLLM agents leverage early-stage global attention for action coordination, while AR agents rely on stepwise, token-level decision making.

Together, our findings highlight diffusion-based generation as a promising paradigm for building more efficient and structured language model agents. Conclusions and future works are in Section 6.

2 Related Work

LLM Agents & Tool-use Workflows. Language-model agents extend single-shot generation into multi-step interaction loops that combine reasoning, state tracking, and external tool use [3, 19]. Recent systems emphasize structured workflows (e.g., plan-act-reflect), explicit tool APIs, and environment feedback, shifting evaluation from final answer quality to trajectory-level properties such as the number of turns, tool-call cost, and recovery behaviors in long-horizon tasks [1, 2, 14]. In parallel, recent benchmarks and agentic evaluations have begun to operationalize tool-use competence and long-horizon reliability in more realistic settings (e.g., software engineering, web navigation, and general tool ecosystems), further highlighting the importance of workflow design and interaction efficiency [20, 21]. Beyond prompt-level scaffolding, these workflows increasingly resemble general decision-making pipelines where the model must coordinate intermediate states, choose actions, and revise plans under partial observability [21, 22]. Our work follows this line by using a fixed agent framework (*i.e.*, DeepDiver [23]) to isolate how the *generation paradigm* affects agent trajectories under controlled workflows and training data.

Decoding/Search for AR Agents. A complementary thread improves agent behavior by augmenting autoregressive (AR) generation with inference-time search and revision. These methods typically keep the AR backbone unchanged while modifying how candidates are proposed, evaluated, and aggregated. For example, sampling multiple reasoning traces and selecting consistent solutions (self-consistency), using explicit verification or reflection to iteratively refine outputs, and applying tree-structured search with learned or heuristic evaluators to explore alternative action sequences [20, 24]. Recent work also revisits RL-style test-time control and search for stronger decision-making, including planning-oriented rollouts and reward-guided inference that can be applied to multi-step problem solving and agent settings [25, 26]. While such approaches demonstrate that inference procedures can substantially influence agent reliability without changing the underlying modeling paradigm, our focus is orthogonal: we hold the agent workflow constant and compare diffusion vs. AR backbones under matched agent-oriented fine-tuning, asking whether the *generation paradigm itself* induces systematic differences in planning, redundancy, and trajectory efficiency.

Diffusion LLMs for Text Generation. Diffusion-based language modeling has re-emerged as a competitive alternative to AR generation by formulating text generation as an iterative denoising process over an entire sequence, enabling global coordination and repeated revision. Recent work shows that discrete or latent diffusion formulations can reach strong generation quality while offering parallel decoding and iterative refinement [4, 7, 18]. Building on these foundations, subsequent studies explore diffusion's ability to support higher-level semantic control and structured generation, such as latent planning mechanisms and paragraph-level organization [8,27], as well as continuous-discrete formulations that facilitate latent reasoning and progressive refinement [11]. However, this literature has largely remained within single-shot (or single-episode) generation settings and has not been systematically evaluated in tool-using, multi-turn agent workflows where behavior is defined by action trajectories rather than one-off outputs.

Agent Behavior Analysis & Attention/Trajectory Metrics. As agents become more widely deployed, an increasing number of studies analyze agent behavior beyond final accuracy, focusing on process metrics such as redundancy, tool-use efficiency, trajectory length, and failure modes across open-ended and interactive tasks [13,28,29]. Recent benchmark-driven analyses in realistic environments (e.g., web navigation and software engineering) also surface trajectory-level bottlenecks such as unnecessary actions, brittle recovery, and high interaction cost, motivating metrics that capture efficiency and robustness rather than only endpoint success [21,30]. Another line of work studies internal model dynamics (e.g., attention patterns and information flow) to explain how models coordinate decisions over multi-step interactions, though such analyses are often token-centric or single-generation-centric rather than aligned to explicit workflow stages [1,25,31]. Against the backdrop of growing interest in DLLMs as autoregressive backbone alternatives for low-latency real-time agentic interaction, recent work conducts a comprehensive evaluation of representative DLLMs on embodied and tool-calling agent paradigms, and uncovers that current DLLMs cannot act as reliable agentic backbones due to systematic failures in long-horizon planning and symbolic precision maintenance under diffusion noise [32]. Our analysis complements these efforts by aligning attention dynamics to the agent workflow and comparing diffusion vs. AR agents under controlled conditions, aiming to reveal paradigm-specific coordination mechanisms that manifest at the trajectory level.

3 Method

3.1 Overview: Controlled Agent Comparison

Our goal is to isolate how the *generation paradigm* of the policy backbone—autoregressive (AR) versus diffusion language models (DLLMs)—affects tool-using agent behavior in complex multi-turn workflows. To this end, we conduct a controlled comparison in which the entire agent pipeline is kept fixed and only the backbone used to generate each action is swapped.

Across both agents, we match: (i) the multi-agent orchestration framework; (ii) the tool set and APIs; (iii) prompt templates and state serialization; (iv) training data and optimization budgets; and (v) evaluation protocols and interaction budgets. Under these controls, the sole varying factor is the intra-step generation mechanism: AR decoding produces actions through left-to-right token prediction with irreversible commitment to early tokens, whereas DLLMs produce actions through iterative denoising and global refinement.

This controlled setup allows us to attribute differences in convergence to correct action trajectories, redundancy in intermediate reasoning and tool calls, and overall workflow efficiency directly to the backbone generation paradigm.

3.2 Agent Workflow and Tool Interface

In this work, we employ multi-agent deep research (*i.e.*, DeepDiver [2]) as a representative scenario and establish it as our standardized agent workflow for experimental validation. DeepDiver operates through hierarchical orchestration, where a *Planner Agent* decomposes the global objective into iterative sub-goals and dynamically coordinates specialized agents. Information Seeker agents are responsible for interacting with external knowledge sources and exploration tools, while Writer agents synthesize retrieved evidence and produce structured outputs.

The system is equipped with a heterogeneous tool ecosystem, encompassing both cognitive operators and environment-facing interfaces. Cognitive tools include operators such as `think` and `reflect` for explicit reasoning and self-correction, while environment-facing tools include `batch_web_search`, `url_crawler`, `document_qa`, `file_read`, and `file_write`, enabling information retrieval, document analysis, and external state manipulation.

A typical agent iteration follows a structured *think–act–observe* loop: the Planner Agent first reasons over the current workflow context to formulate the next sub-goal, invokes the appropriate tool through a specialized agent, and then integrates the returned observation into the shared history before proceeding to the next decision.

We abstract each coordinated multi-agent decision as a structured action:

$$a_t \in \{\mathrm{ToolCall}(u_t, \theta_t),\ \mathrm{Terminate}(y)\}, \tag{1}$$

where u_t denotes the selected tool and θ_t its parameters. Concrete operations such as web search, document QA, and file I/O correspond to particular ToolCall instances. $\mathrm{Terminate}(y)$ outputs the final response and ends the episode.

Executing an action yields an observation o_t, which is appended to the workflow history and conditions subsequent decisions.

3.3 Trajectory and Backbone Policy Formulation

Workflow state and trajectory. Given a user query q, the global execution context at round t is summarized by the accumulated history:

$$\mathcal{H}_{t-1} = \{q, (x_1, o_1), \ldots, (x_{t-1}, o_{t-1})\}, \tag{2}$$

where x_i is the textual serialization of the structured action issued at round i (including reasoning fields and tool invocations), and o_i is the corresponding observation.

An episode produces a workflow trajectory:

$$\tau = \big\{(\mathcal{H}_{t-1}, x_t, a_t, o_t)\big\}_{t=1}^{T^\star}, \tag{3}$$

with termination step $T^\star$.

This trajectory-level formulation allows us to analyze agent behavior at the workflow level, capturing not only final task success but also intermediate efficiency indicators such as the number of tool calls, redundant reasoning steps, and convergence speed to viable action paths.

AR policy factorization. Under an autoregressive backbone, each round is generated through a left-to-right factorized process:

$$r_t \sim \pi_{\mathrm{AR}}(r \mid \mathcal{H}_{t-1}), \quad a_t \sim \pi_{\mathrm{AR}}(a \mid \mathcal{H}_{t-1}, r_t), \tag{4}$$

where r_t denotes intermediate reasoning tokens and a_t the structured action outcome. This sequential commitment can amplify verbosity and often necessitates corrective actions across multiple rounds.

Diffusion-based policy refinement. With a diffusion backbone, the coordinated decision for round t is produced through iterative refinement over the entire structured action segment. Let x_t denote the serialized action proposal. The DLLM defines a denoising process:

$$x_t^{(0)} \to x_t^{(1)} \to \cdots \to x_t^{(K_t)}, \quad \text{where } K_t \text{ is adaptively determined by confidence-gated decoding.} \tag{5}$$

Implementation note: while this formulation treats x_t as a single sequence for clarity, the denoising process is applied over contiguous token blocks in practice, enabling scalable refinement under the same conditional objective.

The final refined output $x_t^{(K)}$ is parsed into the structured action a_t and executed. Across backbones, the orchestration logic and tool semantics remain unchanged; only the generation mechanism differs.

3.4 Agent-oriented Fine-tuning for DLLM

We adapt DLLMs as policy backbones through agent-oriented fine-tuning that directly targets structured action decisions. Although the supervision source—agent trajectories—matches standard agent SFT used for AR models, the learning objective differs fundamentally.

Action-level supervision. We construct a dataset of rollout pairs:

$$\tau = \big\{(\mathcal{H}_{t-1}, x_t)\big\}_{t=1}^{T^\star}, \tag{6}$$

where $\mathcal{H}_{t-1}$ denotes the interaction history up to round $t-1$ (including user messages, intermediate tool observations, and prior agent actions), x_t is the agent's *action segment* at round t, and $T^\star$ is the number of decision rounds in the trajectory. Each action segment x_t contains (i) optional reasoning tokens and (ii) exactly one structured outcome: either a `ToolCall` with arguments or a terminal response. Using action-level targets ensures that the model learns to produce coherent, complete decisions aligned with the agent workflow, rather than generic dialogue continuations.

Conditional denoising objective. Let $x \triangleq x_t$ and $c \triangleq \mathcal{H}_{t-1}$. DLLMs corrupt the target action segment with a *noise-level-indexed* corruption operator $\mathcal{C}_k$:

$$\tilde{x} = \mathcal{C}_k(x), \tag{7}$$

where $k \in \{1, \ldots, K\}$ indexes the corruption (noise) level, K is the maximum number of diffusion steps (or discrete noise levels) used in training, and $\mathcal{C}_k(\cdot)$ is a step-dependent stochastic operator that increasingly perturbs x as k grows (e.g., by masking or replacing a larger fraction of tokens or token spans at higher k).

The primary diffusion objective trains the model to recover the clean action segment conditioned on the corrupted input, the context, and the noise level:

$$\mathcal{L}_{\mathrm{MDM}} = \mathbb{E}_{(c,x)\sim\tau,\ k\sim\mathcal{U}(\{1,\ldots,K\})} \Big[-\log p_\theta(x \mid \tilde{x}, c, k) \Big], \tag{8}$$

where $\tilde{x} = \mathcal{C}_k(x)$, p_θ denotes the parameterized conditional distribution of the clean action segment, and $\mathcal{U}$ is the uniform distribution over discrete noise levels.[1]

In addition to the denoising objective, we incorporate a standard autoregressive cross-entropy loss over the clean action segment to stabilize training and preserve strong left-to-right generation capability:

$$\mathcal{L}_{\mathrm{AR}} = \mathbb{E}_{(c,x)\sim\tau} \Big[-\sum_{i=1}^{|x|} \log p_\theta(x_i \mid x_{<i}, c) \Big]. \tag{9}$$

The overall training objective is a weighted combination:

$$\mathcal{L} = \mathcal{L}_{\mathrm{MDM}} + \lambda \, \mathcal{L}_{\mathrm{AR}}, \tag{10}$$

where $\lambda = 0.5$ controls the contribution of the autoregressive term.

In practice, both corruption and denoising are applied over contiguous token blocks (spans) while preserving the same conditional objective.

Compared with naive continuation-based diffusion training on generic dialogue, this agent-oriented formulation aligns learning with the discrete decision structure of tool-using workflows, which is critical under strict interaction budgets.

3.5 Diffusion Masking for Multi-turn Agent Contexts

In multi-turn agent workflows, inference always conditions on a fully clean history and applies diffusion only to the next-step action segment. However, standard DLLM training typically corrupts entire sequences and allows bidirectional interactions across fixed token blocks, introducing severe train–inference mismatches. Such mismatches are particularly problematic in multi-turn settings, where the context encodes strict causal dependencies between previous agent outputs and subsequent tool responses, whereas these dependencies are absent in single-turn generation.

We introduce two complementary masking designs to address the mismatch problem in multi-turn settings. To explicitly align training with agent inference behavior, we decompose each input into a clean context prefix and a target action span: $c_t \parallel x_t$, where $c_t = \mathcal{H}_{t-1}$ encodes the full interaction history and x_t corresponds to the next structured action.

Context-clean corruption. Let $\mathcal{I}_c$ and $\mathcal{I}_x$ denote the token index sets corresponding to the clean context prefix c_t and the target action span x_t, respectively, with $\mathcal{I}_c \cap \mathcal{I}_x = \emptyset$. We define a step-dependent corruption operator $\mathcal{C}_k(\cdot)$ that applies masking or noise to a subset of tokens according to the diffusion step k. Under context-clean corruption, diffusion noise is applied exclusively to the action span:

$$\tilde{x}_{t,i} = \begin{cases} \mathcal{C}_k(x_{t,i}), & i \in \mathcal{I}_x, \\ x_{t,i}, & i \in \mathcal{I}_c, \end{cases} \tag{11}$$

yielding the denoising objective

$$\mathcal{L}_{\mathrm{MDM}} = \mathbb{E}_{(c_t,x_t)\sim\tau,\ k\sim\mathcal{U}(\{1,\ldots,K\})} \Big[-\sum_{i\in\mathcal{I}_x} \log p_\theta(x_{t,i} \mid \tilde{x}_t, c_t, k) \Big]. \tag{12}$$

This design ensures that the context prefix always remains fully observed during training, matching the inference-time input distribution in multi-turn workflows.

[1]Other sampling schedules for k are possible; we adopt uniform sampling for simplicity.

Span-aware attention alignment. While context-clean corruption aligns the input distribution, standard block diffusion still applies bidirectional self-attention uniformly within fixed-length blocks. When a block overlaps the boundary between context and action tokens, this introduces attention paths that do not exist at inference.

To remove such spurious interactions, we introduce a span-aware attention constraint.

Let $\mathcal{I}^{\text{loss}} = \mathcal{I}_x$ denote token positions participating in the denoising objective (i.e., action-span tokens), and let $\mathcal{I}^{\text{ctx}} = \mathcal{I}_c$ denote non-trainable context positions.

For any attention query position $i \in \mathcal{I}^{\text{loss}}$, we restrict attention to:

- all earlier context tokens $j \in \mathcal{I}^{\text{ctx}}$ with $j < i$ (causal access), and
- all tokens within the action span $\mathcal{I}^{\text{loss}}$ (bidirectional diffusion within the span).

Formally, we define the attention mask $M(i,j)$ as:

$$M(i,j) = \begin{cases} 0, & i \in \mathcal{I}^{\text{loss}},\ j \in \mathcal{I}^{\text{ctx}},\ j \geq i, \\ 1, & \text{otherwise}, \end{cases} \tag{13}$$

where $M(i,j) = 0$ removes the attention edge from query i to key j.

This constraint preserves valid causal context conditioning while eliminating non-existent bidirectional attention paths introduced by blockwise diffusion.

Orthogonality. Context-clean corruption aligns the input distribution, while span-aware attention aligns the attention structure. Both can be enabled independently or jointly.

3.6 Running a DLLM Agent in DeepDiver Workflow

Inference-time refinement. At round t, the history $\mathcal{H}_{t-1}$ is serialized into conditioning context c_t. Starting from an initialized action canvas $x_t^{(0)}$, the DLLM applies the refinement process in Section 3.3 for K steps to obtain $x_t = x_t^{(K)}$.

This procedure jointly refines reasoning structure, tool selection, argument construction, and termination decisions before execution.

Per-round adaptive decoding (DLLM only). The DLLM generates each action segment using a standard confidence-gated decoding procedure. At each iteration, masked tokens whose prediction confidence exceeds a threshold τ (we use $\tau = 0.9$) are committed and the process repeats until all tokens within the decoding window are resolved. This yields an adaptive number of refinement iterations per round while avoiding unnecessary computation once the action becomes confident.

Action execution and recovery. After denoising, the refined action segment x_t is parsed into a discrete executable action:

$$a_t \in \{\text{ToolCall}(u_t, \theta_t),\ \text{Terminate}(y_t)\}, \tag{14}$$

where u_t is the tool identifier, θ_t is the structured argument payload, and y_t is the final natural-language answer when the episode terminates. Malformed outputs are handled using identical recovery strategies across backbones to avoid confounding effects.

3.7 Evaluation Protocol and Budget Constraints

Both AR and DLLM agents operate under identical interaction budgets, including a strict global context-length cap (up to 32K tokens), maximum number of rounds $T_{\max}$, maximum number of tool calls $C_{\max}$, and total generated token limits for agent-produced text. If an episode would exceed these constraints, termination follows the same fallback policy for both backbones.

4 Experiments

4.1 Experimental Setup

This section describes the evaluation protocol for comparing AR and diffusion-based agents within the same DeepDiver workflow, with careful control over interaction budgets and context-length constraints.

Tasks and datasets. We evaluate agent performance in two complementary settings.

BrowseComp-zh. We adopt the public BrowseComp-zh benchmark [33], which requires multi-turn web browsing, iterative retrieval, and reasoning over dynamically collected evidence to answer user queries. The benchmark naturally induces long interaction histories due to repeated tool use and document accumulation.

Open-ended qualitative prompts. In addition to benchmark evaluation, we curate a small set of open-ended multi-step research questions designed to stress long-horizon reasoning, tool chaining, and action stability. These prompts are used exclusively for qualitative analysis, allowing direct inspection of agent trajectories, tool-use patterns, and failure modes that are not fully captured by automatic metrics.

Compared agent variants. All agents are instantiated within the same DeepDiver iterative RAG workflow, sharing identical orchestration logic, tool APIs, prompt templates, and action parsers.

We compare:

- **AR Agent.** A DeepDiver agent with an autoregressive LLM (openPangu-Embedded-7B [34]) backbone that generates each action span through left-to-right decoding. The AR backbone is fine-tuned on the same trajectory dataset under matched optimization budgets.
- **DLLM Agent.** A DeepDiver agent with a diffusion language model (openPangu-R-7B-Diffusion [18]) backbone that generates each action span via iterative denoising and refinement. The DLLM is trained using the agent-oriented fine-tuning objective and multi-turn masking strategies described in Section 3.4 and Section 3.5.

Shared interaction budgets and constraints. To ensure a controlled comparison, we impose identical evaluation budgets across all agents:

- **Context-length cap.** All agents operate under a maximum context window of 32K tokens. If adding the next interaction would exceed this limit, the episode is terminated using the same fallback policy across methods (e.g., returning the most recent valid $\mathrm{Terminate}(y)$ action).
- **Interaction cap.** The maximum number of interaction rounds is fixed to $T_{\max} = 15$ for all agents.
- **Tool-use cap.** We apply the same upper bound on the number of ToolCall invocations per episode. Once the budget is exhausted, the agent is prompted to produce a terminal action.

Both models were trained for 5 epochs with a decaying learning rate from $5e^{-6}$ to 0. All action segments are extracted using the same `BEGIN_ACTION`/`END_ACTION` delimiters and parsed with identical deterministic rules to avoid confounding effects from formatting differences.

4.2 Benchmark Evaluation

We benchmark AR and DLLM agents on BrowseComp-zh under the shared interaction budgets described above. Our evaluation focuses on two complementary aspects.

End-task performance. We report standard task success metrics provided by the benchmark, measuring the accuracy of final answers produced after multi-turn browsing and reasoning.

Workflow efficiency and stability. Beyond final accuracy, we analyze trajectory-level properties that are critical for practical agent deployment, including:

- the number of interaction rounds required to reach a terminal decision;
- the number of ToolCall invocations per episode;
- redundancy in intermediate reasoning and repeated tool usage;
- robustness of action formatting and successful parsing across long horizons.

These metrics directly reflect how efficiently and stably different generation paradigms navigate complex multi-turn workflows.

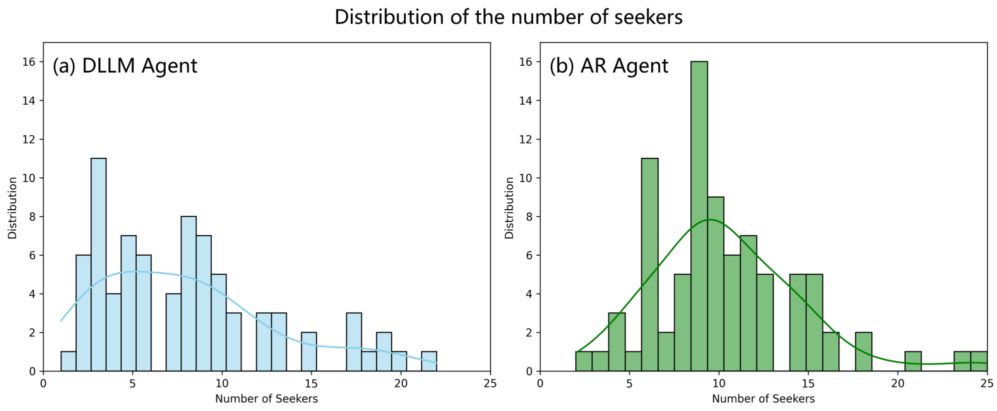


Figure 2: (a) and (b) show the distributions of the number of times the information seeker is invoked by the DLLM agent and the AR agent, respectively. The DLLM agent invokes the seeker significantly fewer times than the AR Agent.

Method	Accuracy (%) ↑	Tool Calls ↓	Turns Used ↓	Invalid Action Rate ↓
AR Agent	15.5	7.5	14.8	1.9%
DLLM Agent	15.5	6.7	13.0	6.4%
w/o context-clean corruption	14.5	6.8	15.0	6.2%
w/o span-aware attention alignment	14.5	7.1	14.6	7.1%

Table 1: Benchmarking evaluation on a 110-question subset sampled from BrowseComp-zh [33], under a shared 32K context constraint and a maximum of $T_{\max} = 15$ rounds. "Tool Calls" and "Turns Used" are averaged per information seeker over episodes. "Invalid Action Rate" denotes the fraction of episodes containing at least one unparsable action span (e.g., missing delimiters, malformed tool call, or invalid schema). All methods share the same tool API, action parser, and budgets; the only difference is the backbone (AR vs. DLLM).

Qualitative trajectory analysis. For open-ended prompts, we present representative agent trajectories to illustrate differences in planning behavior, tool selection patterns, and long-horizon coherence between AR and diffusion-based agents.

We evaluate on a 110-question subset uniformly sampled from BrowseComp-zh [33]. Table 1 reports the benchmark score (higher is better) together with stability/efficiency metrics: average tool calls per episode, average turns used, and the invalid action rate, defined as the fraction of episodes containing at least one unparsable action span (e.g., missing delimiters, malformed tool call, or invalid schema). As shown in Table 1, the AR agent provides a strong baseline under identical budgets.

The DLLM agent invokes the information seeker fewer times overall, leading to a 30% reduction in end-to-end latency compared to the AR agent, as shown in Figure 2. On average, the DLLM agent requires 8 seeker calls per query, whereas the AR agent requires 10.4 calls. This difference can be attributed to the fact that the planner of the DLLM agent generates more concise and structured solution strategies than that of the AR model. As a result, the DLLM agent is able to complete tasks with fewer intermediate information-seeking steps. This advantage will be further examined through detailed case studies in Subsection 4.3. Owing to this property, the DLLM agent achieves performance comparable to that of the AR agent while consuming significantly fewer computational resources.

Both masking modifications improve robustness and overall score: removing either of the two modifications resulted in a performance reduction of $\sim$1%. Overall, these results suggest that diffusion-style iterative refinement can be better compatible with multi-turn browsing agents when masking strategies are aligned in training and inference time.

Taken together, the results in Table 1 and Figure 2 support three practical takeaways:

- Under identical training data and interaction budgets, diffusion-backed planning can match AR accuracy while improving workflow efficiency, as reflected by fewer seeker calls and fewer overall turns.
- The higher invalid-action rate of the DLLM agent highlights a key deployment challenge: diffusion backbones require stronger tool-call-oriented supervision to reliably emit well-formed, parsable structured actions in multi-turn settings.

- The masking strategy serves as a lightweight alignment mechanism between training and inference in multi-turn agent prompting. It reduces distributional mismatch and modestly improves the DLLM agents.

4.3 Case Study

In this subsection, we present certain representative case studies where both backbones (AR and vanilla DLLM) produce the correct answer for the exact same browsing query, yet the end-to-end latency differs substantially. By inspecting the execution logs side by side, we highlight the behavioral differences in planning, tool-call patterns, and evidence consolidation that account for the speed advantage of the DLLM agent.

Case Study: DLLM agent on a multi-constraint browsing query.

User Query.

```
A tech company: its name comes from an animal; its parent company is a Chinese
internet company; it was formed by merging two in-house teams; and it involves
both software and hardware. What is the company called?
```

Planner → InformationSeeker task delegation (translated excerpt).

```
Action: assign_multi_objective_tasks_to_info_seeker -> Success
{
  "tasks": [{
    "task_content": "Search for a tech company that matches all constraints:
      (1) its name comes from an animal;
      (2) its parent company is a Chinese internet company;
      (3) it was formed by merging two in-house teams;
      (4) it involves both software and hardware.
     Provide the exact company name and relevant background.",
    "deliverable_contents": "Return the company that satisfies all constraints,
     including: (1) legal company name; (2) the animal-name origin;
     (3) parent-company information; (4) merger background (two teams);
     (5) business scope (software and hardware). Present results in a
     structured format.",
    "acceptance_checking_criteria": "You must find at least one company that
     fully satisfies all four constraints and provide sufficient evidence.
     If no perfect match is found, report the closest candidates and explain
     why they do not match.",
    "current_task_status": "Initial task: start from scratch to search for a
     company matching all constraints."
  }]
}
```

Evidence reading + termination (translated excerpt).

```
Action: file_read -> Success {"file_path": "research/findings_summary.md"}

Action: planner_objective_task_done -> Success
{
  "completion_status": "completed",
  "final_answer": "Based on the provided constraints, the company is:
    T-Head Semiconductor Co., Ltd. ...",
  "key_files": [{"file_path": "research/findings_summary.md",
              "is_final_output_file": true}]
}
```

Tool-call trace (episode-level).

```
1) think
2) assign_multi_objective_tasks_to_info_seeker
3) file_read
4) planner_objective_task_done
```

Final answer.

```
final_answer:
Based on your described constraints, the company is T-Head Semiconductor
(Pingtouge).
```

End-to-end latency on the same query (both correct): 140.95 s

Case Study: AR agent on the same multi-constraint browsing query.

User Query.

```
A tech company: its name comes from an animal; its parent company is a Chinese
internet company; it was formed by merging two in-house teams; and it involves
both software and hardware. What is the company called?
```

Artifacts produced .

```
key_files:
- todo_v1.md (task plan v1; decomposition and execution plan)
- todo_v2.md (task plan v2; verification-focused plan)
- research/china_internet_companies_animal_subcompanies_report.md
  (report: 23 animal-named Alibaba subsidiaries)
- research/tech_company_team_mergers_report.md
  (report: 12 examples of tech companies formed by team mergers)
- research/candidate_companies_report.md
  (final report: confirms T-Head/Alibaba is the best match) [final output]
```

Process characteristics.

```
Phase 1: Parallel multi-angle search (3 info-seeking agents dispatched)
  (1) animal-named subsidiaries of Chinese internet companies: succeeded
  (2) animal-named tech company search: failed (not finished within 30 iters)
  (3) companies formed by team mergers: succeeded

Phase 2: Focused verification (todo_v2.md)
  (1) detailed verification of T-Head: failed (not finished within 30 iters)
  (2) verification of other Alibaba subsidiaries: failed (<=30 iters)
  (3) alternative candidate search: succeeded

Phase 3: Integration and confirmation from completed reports; finalize answer.
```

Tool usage summary.

```
- Planning documents: 2 versions (todo_v1.md, todo_v2.md)
- Parallel search: 6 tasks dispatched via
    assign_multi_objective_tasks_to_info_seeker
- File management: multiple reports generated
- Integration: file_read used to confirm final answer from reports
```

Final answer.

```
final_answer:
Based on your described constraints, the company is T-Head Semiconductor
(Pingtouge).

Details:
- Company: T-Head Semiconductor Co., Ltd. (T-Head Semiconductor)
- Parent company: Alibaba Group (Chinese internet company)
- Founded: 2018
- Name origin: "Pingtouge" is a nickname for the honey badger (an animal)
- Merger background: formed by merging two Alibaba teams
```

```
  (C-SKY Microsystems / Zhongtianwei, and DAMO Academy in-house chip team)
- Scope: chip R&D and design; software-hardware co-design / co-optimization
```

End-to-end latency on the same query (both correct): 1152.68 s

These two episodes are intentionally comparable: the AR agent and the DLLM agent solve the *same* multi-constraint query and both return the correct entity (T-Head Semiconductor / Pingtouge). The difference lies in how each backbone behaves during multi-turn tool-based execution, which directly impacts end-to-end latency. Empirically, the vanilla DLLM agent finishes in 140.95s, while the AR agent requires 1152.68s on the identical query (an **8.18× wall-clock speedup** for DLLM).

The log excerpts reveal complementary strengths and a clear efficiency trade-off. On the AR side, the PlannerAgent exhibits a thorough but heavy workflow: it creates multiple planning artifacts (`todo_v1.md`, `todo_v2.md`), dispatches 6 information-seeking tasks, and produces several intermediate research reports before integrating them into a final candidate report. Importantly, the AR summary also records repeated sub-task failures due to iteration caps (multiple tasks not completed within 30 iterations), after which the agent still succeeds by relying on other completed reports. This pattern is robust but computationally expensive: more planning rounds, more intermediate artifact generation, and longer-running verification loops accumulate substantial latency.

In contrast, the DLLM episode demonstrates a diffusion-friendly execution profile: it quickly compiles the query into an explicit multi-objective research contract (deliverables plus acceptance criteria) and closes the loop with a short, stable tool-call trace (`think` → `assign` → `file_read` → `done`). This matters because diffusion-style action generation can operate with a bounded denoising budget per turn and does not require long autoregressive decoding to emit structured tool calls. In multi-turn browsing—where each round must output a well-formed, parseable action span—this fixed-compute, refinement-based generation can translate into lower per-turn overhead and fewer opportunities for protracted verification loops, yielding a large reduction in wall-clock time while preserving correctness. Overall, the side-by-side logs suggest that the diffusion backbone's efficiency advantage manifests not only in faster generation, but also in a more direct, contract-driven execution path that reduces interaction and coordination overhead.

This second case study complements the earlier BrowseComp-zh episode by switching from a constraint-satisfaction browsing task to an *open-ended report-generation* task ("analyze Tesla 2024 Q3 earnings; estimate China deliveries and market share; issue an investment rating"). Here, both agents again complete the task successfully, but the efficiency gap narrows in a predictable way. The DLLM agent finishes end to end in 490.25s, whereas the AR agent requires 715.31s (1.46× latency; equivalently, DLLM is 31.5% faster).

Case Study: DLLM agent (Design 1) on an open-ended report-generation query.

User Query.

```
Analyze Tesla's 2024 Q3 earnings report. Look up China market deliveries and
market share, and then provide an investment rating.
```

Planner -> InformationSeeker delegation (contract-style; excerpt).

```
Action: assign_multi_objective_tasks_to_info_seeker -> Success
{
  "tasks": [{
    "task_content": "Obtain Tesla 2024 Q3 earnings materials (shareholder deck,
      10-Q, earnings call transcript). Extract key financials and guidance.
      Cite sources and page/section anchors."
  },{
    "task_content": "Find Tesla China deliveries in 2024 Q3 (or closest proxy if
      Tesla does not disclose directly). Provide methodology and sources."
  },{
    "task_content": "Estimate China EV market share for Tesla in 2024 Q3 using
      a credible denominator (e.g., BEV passenger car sales). Provide the
      data source, definitions, and calculation."
  },{
    "task_content": "Collect consensus / street views and major risks.
      Provide 3-5 credible sources and summarize divergences."
  }],
  "deliverable_contents": "Return a publishable report with: executive summary,
    financial highlights, China deliveries + market-share calculation, thesis,
    valuation notes (if feasible), risks, and a final rating (Buy/Hold/Sell)
    with justification. Include a sources section.",
  "acceptance_checking_criteria": "Every major numeric claim must be sourced.
    If an item cannot be verified from credible sources, mark it as
    'not disclosed' and use a proxy with clear caveats."
}
```

Evidence consolidation + report writing (excerpt).

```
Action: file_read -> Success {"file_path": "research/tesla_q3_2024_financials.md"}
Action: file_read -> Success {"file_path": "research/tesla_china_deliveries_proxy.
    md"}
Action: file_read -> Success {"file_path": "research/china_ev_market_share_calc.md"}

Action: file_read -> Success {"file_path": "research/risks_and_consensus.md"}

Action: planner_objective_task_done -> Success
{
  "completion_status": "completed",
  "final_answer": "Investment Memo: Tesla 2024 Q3 (Report)\n\nExecutive Summary:\n-
      ...\n\nChina deliveries & share:\n- ... (with sources and caveats)\n\nRating:
      HOLD\nRationale:\n- ...",
  "key_files": [
    {"file_path": "reports/tsla_2024q3_investment_memo.md", "is_final_output_file":
       true}
  ]
}
```

Tool-call trace (episode-level).

```
1) think
2) assign_multi_objective_tasks_to_info_seeker
3) file_read (x4)
4) planner_objective_task_done
```

Note (why the speedup is smaller here).

```
Open-ended report generation requires substantially more long-form writing.
Even if per-token generation is faster, end-to-end advantage can shrink when
total output length and evidence narration dominate runtime.
```

End-to-end latency on the same query (both correct): 490.25 s

Case Study: AR agent on the open-ended report-generation query.

User Query.

```
Analyze Tesla's 2024 Q3 earnings report. Look up China market deliveries and
market share, and then provide an investment rating.
```

Planning + verification-heavy workflow (excerpt).

```
Action: file_write -> Success {"file_path":"todo_v1.md", "desc":"plan v1"}
Action: assign_multi_objective_tasks_to_info_seeker -> Success {"tasks":[...]}
Action: file_write -> Success {"file_path":"todo_v2.md", "desc":"plan v2 (
    verification)"}

Action: file_read -> Success {"file_path":"research/tesla_q3_2024_financials.md"}
Action: file_read -> Success {"file_path":"research/tesla_china_deliveries_proxy.md
    "}
```

```
Action: file_read -> Success {"file_path":"research/china_ev_market_share_calc.md"}

Action: assign_multi_objective_tasks_to_info_seeker -> Success
{"tasks":[{
  "task_content":"Re-check China-deliveries proxy against an alternative source
    and reconcile discrepancies; if inconsistent, update the estimate and
    caveats."
},{
  "task_content":"Re-check market-share denominator definitions (BEV vs NEV vs
    passenger cars). Provide sensitivity analysis if needed."
}]}
```

Termination with long-form report (excerpt).

```
Action: planner_objective_task_done -> Success
{
  "completion_status": "completed",
  "final_answer": "Investment Memo: Tesla 2024 Q3 (Report)\n\nExecutive Summary:\n-
      ...\n\nChina deliveries & share:\n- ... (sources, caveats, sensitivity)\n\
      nRating: HOLD\nRationale:\n- ...",
  "key_files": [
    {"file_path": "todo_v1.md", "is_final_output_file": false},
    {"file_path": "todo_v2.md", "is_final_output_file": false},
    {"file_path": "reports/tsla_2024q3_investment_memo.md", "is_final_output_file":
        true}
  ]
}
```

Tool-call trace (episode-level).

```
1) think
2) file_write (todo_v1)
3) assign_multi_objective_tasks_to_info_seeker
4) file_write (todo_v2)
5) file_read (x3)
6) assign_multi_objective_tasks_to_info_seeker (re-check)
7) planner_objective_task_done
```

End-to-end latency on the same query (both correct): 715.31 s

The key contrast with the first case is that report generation is dominated less by discrete tool-action decisions and more by *long-form synthesis*. Even if diffusion-style action generation reduces overhead for producing well-formed tool calls and high-level plans, the runtime share attributable to writing (multi-section narratives, caveats, methodology explanations, and structured "memo" formatting) increases substantially. As a result, the per-turn/per-action efficiency advantage of the diffusion backbone is partially amortized by the unavoidable cost of generating a much larger volume of final text and weaving together multiple evidence streams.

Nevertheless, the DLLM agent retains a clear end-to-end advantage. Consistent with the execution patterns observed in the first case, the diffusion-backed agent tends to converge earlier to a stable "research contract" (deliverables plus acceptance criteria) and proceeds with a more direct retrieve-then-write trajectory, whereas the AR agent more frequently expands the workflow with additional planning and verification passes (e.g., re-checking definitions/denominators for market share and reconciling alternative sources). In short, when tasks are short and path-finding dominates, diffusion yields dramatic speedups (as in BrowseComp-zh); when tasks are open-ended and output-length dominates, the advantage persists but naturally compresses (490.25s vs. 715.31s), indicating that **DLLM efficiency gains arise at the *agent-workflow level* rather than solely from parallel token emission**.

5 Analysis of DLLM Agent Behavior

While Section 4 demonstrates that DLLM agents achieve substantial efficiency gains—particularly the ability to complete tasks with fewer tool invocations and shorter trajectories—the underlying mechanisms driving these workflow-level differences remain unclear. Are these advantages purely attributable to token-level parallelism, or do diffusion-style generation patterns manifest in qualitatively different planning and tool-use behaviors? To answer this question, we analyze the internal decoding dynamics of DLLM agents during multi-step reasoning workflows. Through fine-grained examination of token-level decoding order, confidence patterns, and entropy evolution within the multi-agent workflow, we reveal how the diffusion backbone shapes agent behavior at a structural level. We focus on two critical roles—the Planner Agent (responsible for task decomposition) and the Information Seeker Agent (handling information retrieval)—to illustrate paradigm-specific behavioral patterns. To make these patterns concrete, we conduct a detailed case study on the following query:

> A tech company: its name comes from an animal; its parent company is a Chinese internet company; it was formed by merging two in-house teams; and it involves both software and hardware. What is the company called?

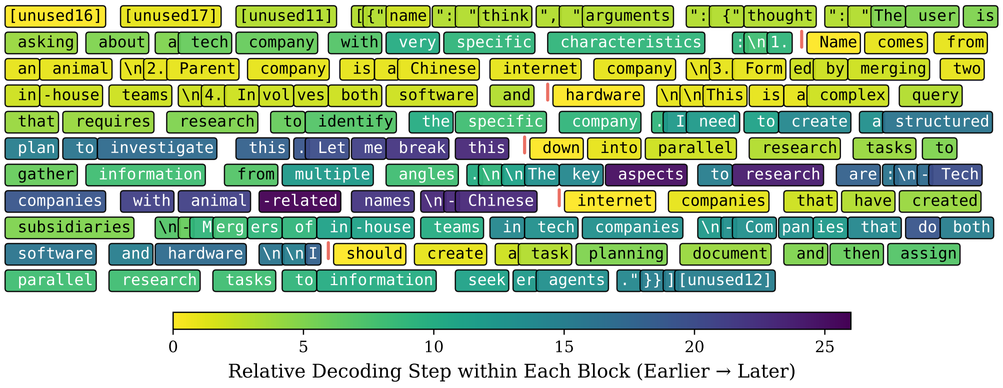


Figure 3: Heatmap showing the relative decoding order of each token within each block of an example Planner Agent response. Different blocks are separated by red lines.

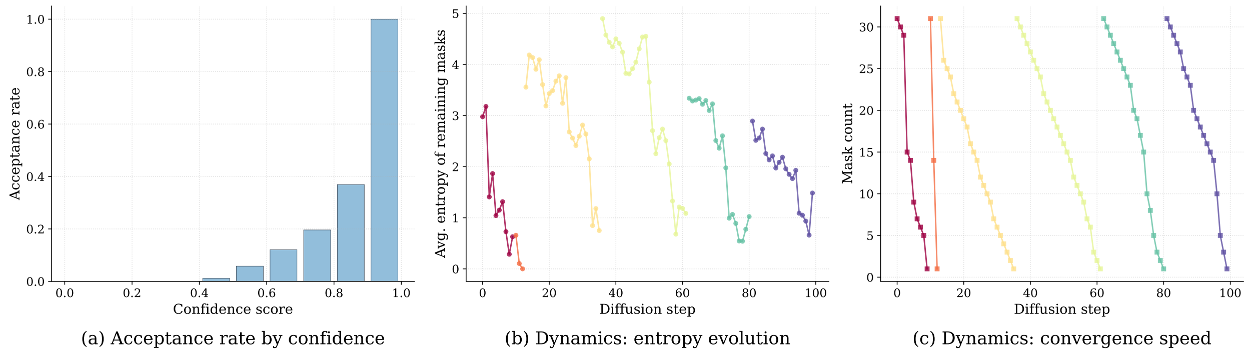


Figure 4: Analysis of the Planner Agent response shown in Figure 3. (a) Probability of tokens with different confidence scores being decoded during diffusion process. (b) Average entropy of remaining mask tokens across different diffusion steps. (c) Number of remaining mask tokens within each block across different diffusion steps.

5.1 Planner Agent

The Planner Agent operates at the workflow's inception, tasked with decomposing complex user queries into actionable research tasks. In the example shown in Figure 3 and Figure 4, we observe that the planner agent's response can be divided into two distinct phases: (1) understanding and summarizing the user's request (corresponding to blocks 1-2), and (2) performing task decomposition (corresponding to blocks 3-6). The following analysis examines the decoding dynamics of these two phases.

Phase 1: Understanding and Summarization (Blocks 1-2). The heatmap in Figure 3 reveals the relative decoding order of each token within each block. During this initial phase, which covers the first two blocks of the response, we observe that the planner agent can rapidly extract the four key pieces of information from the user query. Specifically, the first and second key points are decoded almost simultaneously within the same diffusion step, while the third and fourth key points are decoded together in another diffusion step. This behavior demonstrates that DLLM possesses strong information summarization and extraction capabilities, enabling it to rapidly output key points in parallel.

Phase 2: Task Decomposition (Blocks 3-6). As the response proceeds, the planner agent conducts task decomposition during blocks 3-6. As shown in Figure 4(c), the number of tokens decoded per step decreases during this phase, indicating that the agent engages in more intensive reasoning and planning. Notably, despite the reduced parallelism and a decoding order that tends to proceed from left to right, DLLM still differs from AR models. While AR models generate content sequentially token by token, DLLM progressively fills in critical tokens to establish the reasoning framework before gradually completing the remaining parts of the response. Besides, Figure 4(b) shows a distinctive pattern: the entropy of remaining mask tokens may suddenly increase at the beginning or during intermediate steps within each block, reflecting moments of greater uncertainty in the planning process. As decoding progresses, the entropy generally decreases, signifying increased confidence in the generated content.

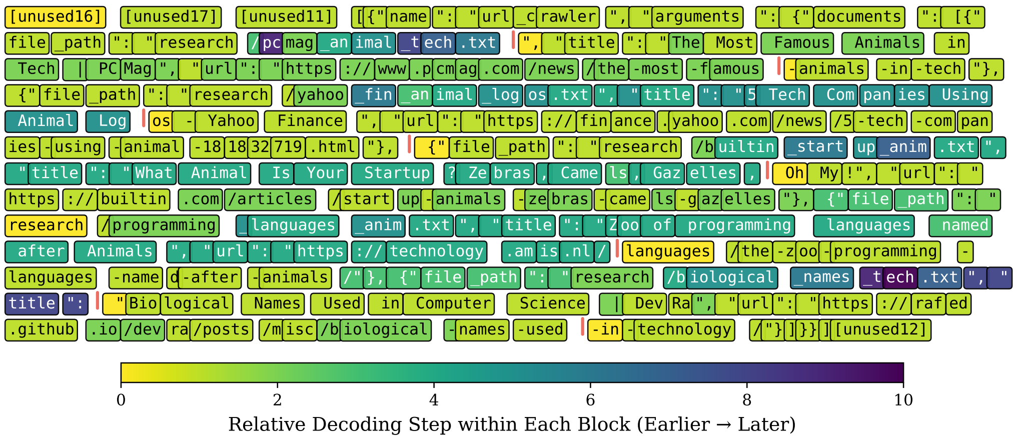
Contrast with AR Planning Behavior. Unlike AR models that must commit to early tokens sequentially and cannot easily revise high-level decisions once formed, the DLLM's iterative refinement allows it to establish a global planning framework (parallel key point extraction) before filling in details. This contrasts with AR planners which typically extract points incrementally and may need to backtrack or reformulate plans if early commitments misalign with later analysis, leading to more intermediate planning artifacts (e.g., multiple `todo_v1.md`, `todo_v2.md` as observed in case studies in Section 4) and redundant task decompositions.

Implications for Workflow Efficiency. The decoding patterns observed in the Planner Agent suggest that DLLM's efficiency advantage is not merely token-level parallelism. The ability to extract key query constraints in parallel (Phase 1) and establish reasoning frameworks before detailed completion (Phase 2) reflects a fundamentally different planning style: one that emphasizes global structural understanding before incremental elaboration. This reduces the need for multiple planning passes and intermediate artifact generation, explaining why DLLM planners produce more concise task assignments that information seekers can execute with fewer re-clarifications—consistent with the reduced information seeker agent invocation counts (8 vs. 10.4 calls per query) reported in Figure 2.

5.2 Information Seeker Agent

The Information Seeker Agent executes the specific research tasks assigned by the planner, primarily focusing on tool invocation and information retrieval. As shown in the example response in Figure 5, the seeker's behavior exhibits a distinctive **tool-oriented generation pattern** with high parallelism throughout the process.

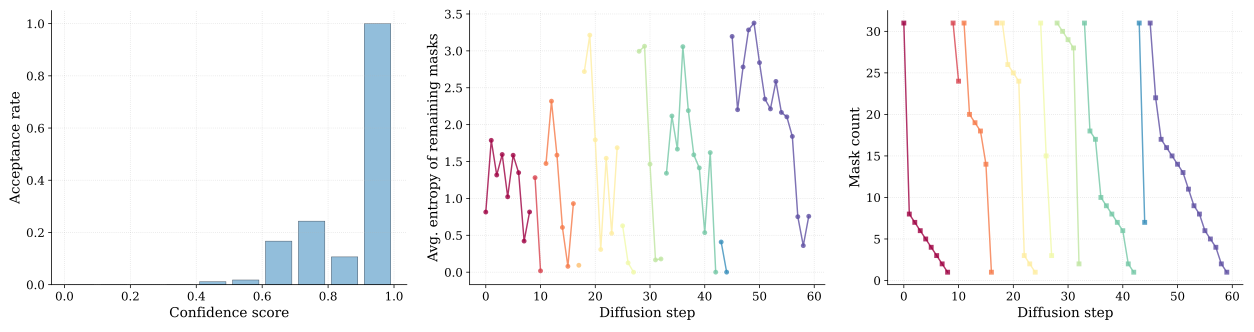
Tool Call Generation Pattern. The heatmap reveals a two-stage pattern in each tool invocation sequence. During the first block, the agent generates the decision to call the `url_crawler` tool, establishing the overall action direction. Subsequent blocks then generate the specific parameters required for this tool call. This separation between tool selection and parameter specification reflects the agent's structured approach to information gathering, where high-level action planning precedes detailed argument construction.



```
[unused16] [unused17] [unused11] [{"name": "url_crawler", "arguments": {"documents": [{"
file_path": "research/pcmag_animal_tech.txt", "title": "The Most Famous Animals in
Tech | PCMag", "url": "https://www.pcmag.com/news/the-most-famous-animals-in-tech"},
{"file_path": "research/yahoo_fin_animal_logos.txt", "title": "5 Tech Companies Using
Animal Logos - Yahoo Finance", "url": "https://finance.yahoo.com/news/5-tech-compan
ies-using-animal-181832719.html"}, {"file_path": "research/builtin_startup_anim.txt",
"title": "What Animal Is Your Startup? Zebras, Camels, Gazelles, Oh My!", "url": "
https://builtin.com/articles/startup-animals-zebras-camels-gazelles"}, {"file_path": "
research/programming_languages_anim.txt", "title": "Zoo of programming languages named
after Animals", "url": "https://technology.amis.nl/languages/the-zoo-programming -
languages-named-after-animals/"}, {"file_path": "research/biological_names_tech.txt", "
title": "Biological Names Used in Computer Science | DevRa", "url": "https://rafed
.github.io/devra/posts/misc/biological-names-used-in-technology/"}]}}][unused12]
```

Figure 5: Heatmap showing the relative decoding order of each token within each block of an example Information Seeker Agent response. Different blocks are separated by red lines.

High Parallelism During Information Retrieval. Compared to the Planner Agent, the Information Seeker demonstrates significantly higher decoding parallelism. As information search tasks typically require less complex reasoning and decision-making, the agent can generate multiple tokens simultaneously within each diffusion step. Figure 6(c) shows



(a) Acceptance rate by confidence (b) Dynamics: entropy evolution (c) Dynamics: convergence speed

Figure 6: Analysis of the Information Seeker response shown in Figure 5. (a) Probability of tokens with different confidence scores being decoded during diffusion process. (b) Average entropy of remaining mask tokens across different diffusion steps. (c) Number of remaining mask tokens within each block across different diffusion steps.

that the average number of tokens decoded per step remains consistently high throughout the entire response, confirming the seeker's efficient parallel processing capability.

Uncertainty Dynamics Across Blocks. Figure 6(b) reveals an interesting pattern of uncertainty within each block. At the beginning of each block, the model encounters increased uncertainty, indicated by higher entropy of remaining mask tokens. This reflects the initial exploration phase where the agent must decide which tool to invoke and how to structure the response. As decoding progresses within the block, uncertainty gradually decreases, signifying increased confidence as the agent fills in specific content and parameters. This within-block uncertainty pattern suggests that the most critical decisions—tool selection and action formulation—concentrate at the early stages of each generation block, followed by more deterministic content completion.

Contrast with AR Tool-Call Generation. In AR-based agents, tool invocation typically proceeds linearly: function name and all parameters are generated sequentially without opportunity for revision or reorganization. If an early token decision (e.g., the function name or parameter structure) proves suboptimal, the agent must continue the sequence and may need to issue corrective calls in subsequent rounds, increasing interaction overhead. The DLLM's two-stage pattern—first establishing tool direction, then filling parameters through parallel refinement—allows the agent to globally coordinate the entire tool invocation before committing to execution. This reduces formulation errors that would require correction across interaction rounds, contributing to the cleaner tool-call traces observed in Section 4's case studies.

5.3 Synthesis: Beyond Token-Level Parallelism

Together, the behavioral patterns observed across both agent roles provide a coherent answer to our central question: the efficiency advantages of DLLM agents extend beyond token-level parallelism and stem from qualitatively different planning and tool-use behaviors.

Two paradigm-specific signatures emerge from our analysis:

(1) Global Coordination Before Local Commitment. Both Planner and Information Seeker agents exhibit a pattern where critical high-level decisions are established early through parallel extraction or refinement, followed by deterministic completion of details. In the Planner Agent, this manifests as rapid parallel extraction of query constraints (four key points in two diffusion steps) during the summarization phase, before task decomposition. In the Information Seeker, this appears as tool selection being finalized before parameter specification within each block. By contrast, AR agents must make irreversible local commitments from the beginning, requiring multiple refinement passes when early decisions misalign with emerging context. This explains why DLLM agents produce fewer intermediate artifacts and converge to valid trajectories with fewer interaction rounds.

(2) Structured Action Formulation. The tool-oriented generation pattern in the Information Seeker reflects a diffusion-friendly interface where entire action spans are refined as structured units. This reduces formulation errors that would require correction across interaction rounds, whereas AR agents generate action components sequentially and must compensate through redundant verification loops—consistent with the heavier planning/verification patterns observed in AR case studies (Section 4). The planner's entropy spikes during task decomposition further indicate that DLLM agents engage in more intensive global reasoning at the action-segment level, rather than distributing uncertainty across incremental token-level decisions.

(3) Forward-aware Global Planning. Beyond local action formulation, the diffusion backbone exhibits a tendency to account for downstream and emerging workflow states when making each decision. Through bidirectional conditioning over partially specified representations and repeated refinement, the model learns to jointly reconcile past context with anticipated future consequences. This training dynamic potentially encourages decisions that are globally consistent across interaction rounds, rather than locally optimal under irreversible sequential commitments.

Together, these signatures indicate that the diffusion backbone's iterative refinement enables agents to "plan" at the *action-segment level* rather than the *token level*, systematically reducing workflow redundancy even after controlling for per-token generation speed. This paradigm shift—from sequential token commitments to structured segment refinement—provides the mechanistic explanation for the workflow-level efficiency gains observed in Section 4. We emphasize that these observations reflect tendencies exhibited under the current agent workflow, training setup, and evaluation regime. While diffusion-based backbones demonstrate strong potential for forward-aware and globally consistent planning, their advantages may vary across tasks, model scales, and system configurations. Exploring the generality of these behaviors in broader agent settings remains an important direction for future work.

6 Conclusions & Future Works

In this work, we investigate how the fundamental differences between diffusion LLMs and autoregressive (AR) LLMs translate into tool-using agent behavior. Rather than introducing a new agent framework, we adopt the DeepDiver workflow and perform continued training on its tool-use trajectories to strengthen both backbones for multi-turn dialogue and structured tool calling. This enables a controlled comparison in which observed differences can be attributed to backbone-induced execution dynamics rather than changes in the agent system itself.

Our results show that diffusion-style modeling can meaningfully manifest at the agent level. Under the evaluated settings, the resulting *DLLM Agent* achieves comparable or slightly improved task accuracy while exhibiting substantially higher workflow efficiency. Across benchmark tasks, the DLLM Agent attains over a 30% average efficiency improvement, with certain cases exceeding $8\times$ end-to-end speedup, all under identical interaction budgets and context constraints. Importantly, these gains are not solely attributable to parallel token generation. They emerge at the level of agent workflows: diffusion-based agents tend to establish higher-level plans earlier, converge to valid trajectories with fewer redundant verification loops, and terminate with shorter and cleaner tool-call traces even when both backbones ultimately reach the correct answer.

Beyond quantitative metrics, our case studies further highlight qualitative differences in exploration strategies, consolidation of evidence, and stopping behavior between AR and diffusion agents. We additionally provide a mechanistic analysis through attention dynamics, linking backbone-specific properties such as bidirectional conditioning and iterative refinement to observed planning and tool-use patterns. This perspective offers insight into how diffusion-based generation can support more globally consistent and forward-aware decision making in multi-step agent workflows. Overall, our findings suggest that diffusion backbones constitute a viable and promising alternative for tool-using agents, with the potential to improve end-to-end execution efficiency without sacrificing task success under appropriate training and system conditions. While further exploration across broader tasks and workflows is necessary to assess generality, this work points toward diffusion-based agents as an encouraging direction for building more efficient and streamlined interactive systems.

Looking ahead, several directions merit further investigation. First, while our study focuses on controlled multi-turn tool-use benchmarks, it remains an open question how diffusion-based agents scale to longer-horizon, more open-ended workflows with partial observability and richer tool ecosystems. Understanding whether the observed efficiency gains persist or compound in such settings is critical for assessing their practical impact. Second, our current training protocol relies on continued learning from autoregressive agent trajectories. Future work could explore diffusion-native training objectives, supervision schemes, or curricula that more explicitly leverage iterative refinement, intermediate planning states, or trajectory-level noise schedules, potentially further amplifying the advantages observed at the agent level. Third, the complementary strengths of diffusion and autoregressive backbones suggest promising hybrid designs, where diffusion models handle global planning or hypothesis consolidation while autoregressive models manage fine-grained execution and interaction. Investigating such mixed-backbone agents may yield additional gains in robustness and efficiency beyond either paradigm alone. Finally, deeper mechanistic analyses—such as causal interventions on refinement steps or systematic ablations of bidirectional conditioning—could further clarify how backbone-specific properties give rise to distinct agent behaviors, and help delineate the regimes in which diffusion-based generation is most beneficial.

Limitations. Our findings are primarily obtained in the DeepDiver agent workflow and focused on deep research-style multi-turn retrieval and tool-augmented reasoning tasks under fixed context and interaction budgets. While these settings enable controlled comparison between diffusion and autoregressive backbones, it remains unclear how well the observed efficiency and planning behaviors generalize to larger model scales, longer context windows, or more diverse agent scenarios such as long-horizon planning and embodied environments. Exploring these broader settings is an important direction for future work.

References

[1] W. Shi, H. Tan, C. Kuang, X. Li, X. Ren, C. Zhang, H. Chen, Y. Wang, L. Shang, F. Yu *et al.*, "Pangu deepdiver: Adaptive search intensity scaling via open-web reinforcement learning," *arXiv preprint arXiv:2505.24332*, 2025.

[2] openPangu Team, "openpangu deepdiver-v2: Multi-agent learning for deep information seeking," Huawei, Tech. Rep., 2025, accessed: 2025-12-03. [Online]. Available: https://ai.gitcode.com/ascend-tribe/openPangu-Embedded-7B-DeepDiver/blob/main/docs/openpangu-deepdiver-v2-tech-report.pdf

[3] R. Xu and J. Peng, "A comprehensive survey of deep research: Systems, methodologies, and applications," *arXiv preprint arXiv:2506.12594*, 2025.

[4] J. Ye, Z. Xie, L. Zheng, J. Gao, Z. Wu, X. Jiang, Z. Li, and L. Kong, "Dream 7b: Diffusion large language models," *arXiv preprint arXiv:2508.15487*, 2025.

[5] Y. Song, Z. Zhang, C. Luo, P. Gao, F. Xia, H. Luo, Z. Li, Y. Yang, H. Yu, X. Qu *et al.*, "Seed diffusion: A large-scale diffusion language model with high-speed inference," *arXiv preprint arXiv:2508.02193*, 2025.

[6] Y. Wang, K. Han, H. Zhen, Y. Tian, H. Chen, Y. Huang, Y. Cui, Y. Shu, S. Gao, I. Elezi *et al.*, "Top 10 open challenges steering the future of diffusion language model and its variants," *arXiv preprint arXiv:2601.14041*, 2026.

[7] S. Sahoo, M. Arriola, Y. Schiff, A. Gokaslan, E. Marroquin, J. Chiu, A. Rush, and V. Kuleshov, "Simple and effective masked diffusion language models," *Advances in Neural Information Processing Systems*, vol. 37, pp. 130 136–130 184, 2024.

[8] H. Kang, Y. Zhang, N. L. Kuang, N. Majamaki, N. Jaitly, Y.-A. Ma, and L. Qin, "Ladir: Latent diffusion enhances llms for text reasoning," *arXiv preprint arXiv:2510.04573*, 2025.

[9] Y. Zhang, J. Gu, Z. Wu, S. Zhai, J. Susskind, and N. Jaitly, "Planner: Generating diversified paragraph via latent language diffusion model," *Advances in Neural Information Processing Systems*, 2023.

[10] L. Zhong, L. Wu, B. Fang, T. Feng, C. Jing, W. Wang, J. Zhang, H. Chen, and C. Shen, "Beyond hard masks: Progressive token evolution for diffusion language models," *arXiv preprint arXiv:2601.07351*, 2026.

[11] C. Zhou, C. Yang, Y. Hu, C. Wang, C. Zhang, M. Zhang, L. Mackey, T. Jaakkola, S. Bates, and D. Zhang, "Coevolutionary continuous discrete diffusion: Make your diffusion language model a latent reasoner," *arXiv preprint arXiv:2510.03206*, 2025.

[12] B. Ma, Z. Zong, G. Song, H. Li, and Y. Liu, "Exploring the role of large language models in prompt encoding for diffusion models," *arXiv preprint arXiv:2406.11831*, 2024.

[13] X. Geng, P. Xia, Z. Zhang, X. Wang, Q. Wang, R. Ding, C. Wang, J. Wu, Y. Zhao, K. Li *et al.*, "Webwatcher: Breaking new frontier of vision-language deep research agent," *arXiv preprint arXiv:2508.05748*, 2025.

[14] X. Li, J. Jin, G. Dong, H. Qian, Y. Wu, J.-R. Wen, Y. Zhu, and Z. Dou, "Webthinker: Empowering large reasoning models with deep research capability," *arXiv preprint arXiv:2504.21776*, 2025.

[15] M. Arriola, A. Gokaslan, J. T. Chiu, Z. Yang, Z. Qi, J. Han, S. S. Sahoo, and V. Kuleshov, "Block diffusion: Interpolating between autoregressive and diffusion language models," *arXiv preprint arXiv:2503.09573*, 2025.

[16] T. Bie, M. Cao, K. Chen, L. Du, M. Gong, Z. Gong, Y. Gu, J. Hu, Z. Huang, Z. Lan *et al.*, "Llada2. 0: Scaling up diffusion language models to 100b," *arXiv preprint arXiv:2512.15745*, 2025.

[17] C. Wu, H. Zhang, S. Xue, S. Diao, Y. Fu, Z. Liu, P. Molchanov, P. Luo, S. Han, and E. Xie, "Fast-dllm v2: Efficient block-diffusion llm," *arXiv preprint arXiv:2509.26328*, 2025.

[18] Y. Tian, Y. Liang, S. Zhang, Y. Shu, G. Yang, W. He, S. Fang, T. Guo, K. Han, C. Xu, H. Chen, X. Chen, and Y. Wang, "From next-token to next-block: A principled adaptation path for diffusion llms," 2026. [Online]. Available: https://arxiv.org/abs/2512.06776

[19] Y. Zheng, D. Fu, X. Hu, X. Cai, L. Ye, P. Lu, and P. Liu, "Deepresearcher: Scaling deep research via reinforcement learning in real-world environments," *arXiv preprint arXiv:2504.03160*, 2025.

[20] W. Lin, H.-L. Zhen, S. Yang, X. Wang, R. Liu, H. Chen, W. Zhang, C. Zhou, Y. Li, C. Chen *et al.*, "Towards efficient agents: A co-design of inference architecture and system," *arXiv preprint arXiv:2512.18337*, 2025.

[21] Y. Jiang *et al.*, "Webarena: A realistic web environment for building autonomous agents," *arXiv preprint arXiv:2307.13854*, 2024.

[22] Y. Huang, Y. Chen, H. Zhang, K. Li, H. Zhou, M. Fang, L. Yang, X. Li, L. Shang, S. Xu *et al.*, "Deep research agents: A systematic examination and roadmap," *arXiv preprint arXiv:2506.18096*, 2025.

[23] H. Wan, C. Yang, J. Yu, M. Tu, J. Lu, D. Yu, J. Cao, B. Gao, J. Xie, A. Wang *et al.*, "Deepresearch arena: The first exam of llms' research abilities via seminar-grounded tasks," *arXiv preprint arXiv:2509.01396*, 2025.

[24] R. Ye, Z. Zhang, K. Li, H. Yin, Z. Tao, Y. Zhao, L. Su, L. Zhang, Z. Qiao, X. Wang *et al.*, "Agentfold: Long-horizon web agents with proactive context management," *arXiv preprint arXiv:2510.24699*, 2025.

[25] N. Shinn, B. Labash, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *arXiv preprint arXiv:2303.11366*, 2024.

[26] W. Lin, X. Li, Z. Yang, X. Fu, H.-L. Zhen, Y. Wang, X. Yu, W. Liu, X. Li, and M. Yuan, "Trimr: Verifier-based training-free thinking compression for efficient test-time scaling," *arXiv preprint arXiv:2505.17155*, 2025.

[27] L. Ma, Y. Cui, K. Han, and Y. Wang, "Diffusion in diffusion: Reclaiming global coherence in semi-autoregressive diffusion," 2026. [Online]. Available: https://arxiv.org/abs/2601.13599

[28] J. Wu, B. Li, R. Fang, W. Yin, L. Zhang, Z. Tao, D. Zhang, Z. Xi, G. Fu, Y. Jiang *et al.*, "Webdancer: Towards autonomous information seeking agency," *arXiv preprint arXiv:2505.22648*, 2025.

[29] D.-C. Zhang, Y. Zhao, J. Wu, L. Zhang, B. Li, W. Yin, Y. Jiang, Y.-F. Li, K. Tu, P. Xie *et al.*, "Evolvesearch: An iterative self-evolving search agent," in *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025, pp. 13 134–13 147.

[30] H. Luo, G. Chen, Q. Lin, Y. Guo, F. Xu, Z. Kuang, M. Song, X. Wu, Y. Zhu, L. A. Tuan *et al.*, "Graph-r1: Towards agentic graphrag framework via end-to-end reinforcement learning," *arXiv preprint arXiv:2507.21892*, 2025.

[31] X. Li, W. Jiao, J. Jin, G. Dong, J. Jin, Y. Wang, H. Wang, Y. Zhu, J.-R. Wen, Y. Lu *et al.*, "Deepagent: A general reasoning agent with scalable toolsets," *arXiv preprint arXiv:2510.21618*, 2025.

[32] Q. Lu, L. Ding, K. Zhang, J. Zhang, and D. Tao, "The bitter lesson of diffusion language models for agentic workflows: A comprehensive reality check," *arXiv preprint arXiv:2601.12979*, 2026.

[33] P. Zhou, B. Leon, X. Ying, C. Zhang, Y. Shao, Q. Ye, D. Chong, Z. Jin, C. Xie, M. Cao *et al.*, "Browsecomp-zh: Benchmarking web browsing ability of large language models in chinese," *arXiv preprint arXiv:2504.19314*, 2025.

[34] H. Chen, Y. Wang, K. Han, D. Li, L. Li, Z. Bi, J. Li, H. Wang, F. Mi, M. Zhu *et al.*, "Pangu embedded: An efficient dual-system llm reasoner with metacognition," *arXiv preprint arXiv:2505.22375*, 2025.