LATENT ACTION REPARAMETERIZATION FOR EFFICIENT AGENT INFERENCE

Qingwen Zeng[1,*], **Wenhao Huang**[2,*], **Zerui Xu**[3,*], **Zijie Guo**[4,*], **Yu Sun**[5], **Cheng Yang**[6], **Siru Ouyang**[7], **Jiri Gesi**[8], **Fang Wu**[9], **Jiayi Zhang**[6,10], **Bang Liu**[2], **Xiangru Tang**[5,†], **Chenglin Wu**[6,†]

[1]University of Sydney [2]Université de Montréal [3]University of Chicago [4]Fudan University
[5]Yale University [6]DeepWisdom [7]University of Illinois Urbana-Champaign [8]Amazon Science
[9]Stanford University [10]The Hong Kong University of Science and Technology (Guangzhou)

ABSTRACT

Large language model (LLM) agents often rely on long sequences of low-level textual actions, resulting in large effective decision horizons and high inference cost. While prior work has focused on improving inference efficiency through system-level optimizations or prompt engineering, we argue that a key bottleneck lies in the representation of the action space itself. We propose Latent Action Reparameterization (LAR), a framework that learns a compact latent action space in which each latent action corresponds to a multi-step semantic behavior. By reparameterizing agent actions into latent units, LAR enables decision making over a shorter effective horizon while preserving the expressiveness of the original action space. Unlike hand-crafted macros or hierarchical controllers, latent actions are learned from agent trajectories and integrated directly into the model, allowing both planning and execution to operate over abstract action representations. Across a range of LLM-based agent benchmarks, LAR significantly reduces the effective action horizon and improves inference efficiency under fixed compute budgets. As a consequence, our approach achieves substantial reductions in action tokens and corresponding wall-clock inference time, while maintaining or improving task success rates. These results suggest that action representation learning is a critical and underexplored factor in scaling efficient LLM agent inference, complementary to advances in model architecture and hardware. Source code is attached here [1].

1 INTRODUCTION

Large language model (LLM) agents have emerged as a powerful paradigm for solving complex tasks involving multi-step reasoning, tool use, and interaction with external environments (Wang et al., 2024). By repeatedly generating actions conditioned on intermediate observations, modern LLM agents can perform search, planning, and decision making across diverse domains (Liu et al., 2023; Gioacchini et al., 2024). However, as these agents are applied to increasingly complex tasks, inference efficiency has become a critical bottleneck (Liu et al., 2025). Agent execution often requires long sequences of decisions, leading to high inference latency and prohibitive computational cost, which in turn limits scalability, deployment, and real-time interaction (Gonzalez-Pumariega et al., 2025).

Prior work has primarily addressed agent inference efficiency through improvements in model architecture, hardware acceleration, system-level optimizations, or prompt engineering (Cai et al., 2025; Chen et al., 2024b; Debnath et al., 2025; Wan et al., 2023). These approaches reduce the cost of individual inference steps or improve throughput, but they largely operate orthogonally to the structure of the agent's decision process itself (Zhou et al., 2024). In particular, while per-token generation may become faster (Gim et al., 2024; Cai et al., 2025), the number of decision steps required to complete

[*]Equal contribution.
[†]Correspondence to: xiangru.tang@yale.edu, alexanderwu@deepwisdom.ai
[1]https://anonymous.4open.science/r/LAR-A2FD

a task often remains unchanged. As a result, the overall inference cost continues to scale poorly with task horizon, especially in settings that require multi-step reasoning or search (Chen et al., 2025).

In this work, we argue that inference efficiency in LLM agents is fundamentally constrained by the representation of the action space, particularly in sequential decision-making settings. In current agent systems, actions are typically realized as low-level textual outputs, where each generated token constitutes an explicit decision that conditions subsequent computation, planning, or interaction with the environment (Kim et al., 2025). Such token-level action representations induce excessively fine-grained decision making, resulting in unnecessarily large effective decision horizons even for semantically simple behaviors (Zhai et al., 2025; Chen et al., 2024a). Consequently, inference scaling is dominated not by model size alone, but by the granularity at which agent actions are represented and composed over time (Zheng et al., 2024; Yang et al., 2025b). We therefore posit that action representation should be treated as a first-class modeling choice in LLM-based agents, on par with model architecture and system-level design.

Motivated by this observation, we propose Latent Action Reparameterization (LAR), a framework that learns a compact latent action space for LLM agents. In LAR, each latent action corresponds to a multi-step semantic behavior that would otherwise be realized through a sequence of low-level textual actions. By reparameterizing agent decisions into these latent units, planning and execution can operate over a substantially shorter effective horizon while preserving the expressiveness of the original action space. Unlike hand-crafted macros or hierarchical controllers (Al-Emran, 2015; Amato et al., 2019; Bacon et al., 2017), latent actions in LAR are learned directly from agent trajectories and integrated into the model, enabling end-to-end decision making over abstract yet executable action representations.

A key challenge in action abstraction for LLM agents lies in balancing representational abstraction with action executability (Yao et al., 2022; Schick et al., 2023). Fully implicit latent representations are effective for internal reasoning and credit assignment, but they are insufficient for agent systems that must interact with external tools or environments through explicit, protocol-constrained interfaces (Schick et al., 2023; Hafner et al., 2023). In such settings, actions must remain decodable, interpretable, and executable by downstream systems. LAR addresses this challenge by explicitly modeling the latent–explicit boundary: latent actions provide higher-level abstraction while remaining directly realizable as concrete, executable action sequences. Specifically, our framework compresses low-entropy, structurally recurring patterns including system prompts, tool invocation syntax, and recurring configurations into latent units, while strictly preserving high-entropy, parameter-rich inputs (e.g., specific search queries or numerical entities) in the explicit output space. This design reflects a broader principle in agent modeling: increased abstraction is not always beneficial, as executability fundamentally constrains useful action representations (Yao et al., 2022).

Our main contributions are summarized as follows:

- **Significant Efficiency Gains:** We demonstrate that LAR significantly reduces the effective action horizon, leading to substantial reductions in action tokens and corresponding improvements in inference efficiency across diverse LLM agent benchmarks.
- **Preserved Task Performance:** Despite operating over a compressed latent action space, our approach maintains or improves task success rates compared to baselines using raw textual actions, proving that efficiency need not come at the cost of performance.
- **Analysis of Abstraction Limits:** We identify a distinct performance collapse threshold, empirically delineating the boundary between compressible structural redundancy and essential semantic content.
- **New Perspective on Scaling:** Our results highlight action representation learning as a critical and underexplored factor in scaling efficient LLM agent inference, offering a complementary path to advances in model architecture and hardware.

2 Related Work

A large body of prior work improves the efficiency of LLM-based agents by modifying different stages of the agent pipeline, including how inputs are conditioned, how tokens are generated, and how interaction histories are maintained. **Prompting and Input-Level Control:** Prompting and input-level

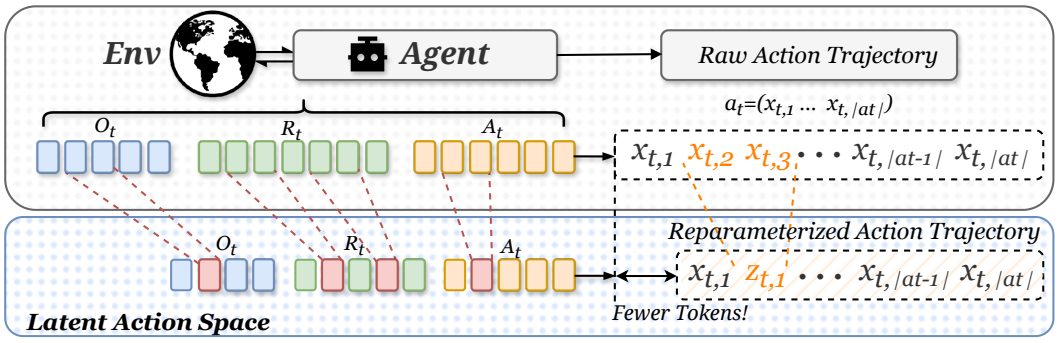


Figure 1: Overview of Latent Action Reparameterization (LAR). LAR reformulates agent decision making by collapsing recurrent, transition-equivalent action segments into executable latent actions, thereby reducing the effective decision horizon. Low-entropy structural components are abstracted into latent actions, while high-entropy, parameter-binding content remains explicit to preserve executability.

methods improve efficiency by shaping the conditioning signal before or during inference (Debnath et al., 2025). Techniques such as Chain-of-Thought elicit intermediate reasoning that improves accuracy but often increases generation length and latency (Wei et al., 2022; Wang et al., 2022). Subsequent prompt engineering constrains reasoning formats to reduce verbosity while preserving answer quality (Zhou et al., 2022; Li et al., 2023). **Token-Level Generation Control and Inference-Time Interventions:** Another line of work regulates token emission during inference to reduce redundant generation (Leviathan et al., 2023; Kim et al., 2023; Shridhar et al., 2023). Representative approaches include in-generation guidance that encourages shorter reasoning traces (e.g., ConciseHint-style methods (Tang et al., 2025)) and token scoring or pruning mechanisms that skip low-utility tokens (e.g., TokenSkip (Xia et al., 2025)). These methods optimize efficiency within the original token-level generation process. **Context and Memory Optimization for Agents:** For interactive and tool-using agents, efficiency bottlenecks often stem from long interaction histories carried as context (Shinn et al., 2023; Park et al., 2023; Packer et al., 2023; Zhang et al., 2025). Context and memory optimization methods reduce conditioning costs by compressing or summarizing histories. ACON is the representative optimizing the agent's memory representation via history and observation compression (Kang et al., 2025).

Collectively, the above approaches improve efficiency while leaving the decision interface fundamentally unchanged: the agent still reasons and acts at the level of token emissions, and efficiency gains arise from modifying inputs, regulating token generation, or compressing memory (Wan et al., 2023). As a result, the effective decision horizon remains dictated by token-level granularity. In contrast, our work challenges this assumption and targets inefficiency at its source by redefining the unit of decision-making itself. LAR reparameterizes the action space by collapsing multi-step action segments that induce transition-equivalent behaviors into single latent actions, thereby directly reducing the effective decision horizon. Crucially, reparameterization is constrained by executability: parameter-binding actions that determine environment-facing semantics are preserved explicitly, while only stable, context-invariant scaffolds are abstracted. This reframes efficiency not as a byproduct of shorter text, but as a consequence of operating over a more appropriate decision representation.

3 Methodology

3.1 Action Definition and Problem Setup

We consider an LLM-based agent operating in a sequential decision-making setting. An agent–environment interaction is represented as a trajectory $\tau = (o_1, a_1, o_2, a_2, \dots, o_T, a_T)$, where o_t denotes the observation at step t and a_t denotes the action produced by the agent at that step. Observations may include textual context, intermediate reasoning states, tool outputs, or environment feedback.

In contemporary LLM agents, actions are instantiated as explicit textual outputs. Each action a_t is a sequence of generated tokens $a_t = (x_{t,1}, \ldots, x_{t,|a_t|})$, with $x_{t,i}$ drawn from the model vocabulary. We treat all generated tokens that condition subsequent computation or interaction as action decisions, including system-level configurations and interaction scaffolds.

We define the effective action horizon of a trajectory as $H_{\text{eff}}(\tau) = \sum_{t=1}^{T} |a_t|$, which measures the total number of explicit generation decisions and directly determines inference cost. Our objective is to reduce this horizon by altering action representation, without modifying agent behavior or executability.

3.2 Latent Action Reparameterization

We propose Latent Action Reparameterization (LAR), which reformulates agent decision making over a compact action space as shown in Fig. 1. Instead of operating over token-level action primitives, LAR enables the agent to reason over higher-level action units that correspond to multi-step semantic behaviors.

Let $\mathcal{Z}$ denote the latent action space, and let $z_t \in \mathcal{Z}$ denote the latent action instantiated at step t. Each latent action represents a semantic decision unit that subsumes a sequence of low-level actions. Under this representation, the effective horizon becomes $H_{\text{lat}}(\tau) = \sum_t |z_t|$, where each latent action is treated as a single decision step.

LAR aims to preserve the functional behavior induced by original trajectories while eliminating redundant decision steps caused by overly fine-grained action representations. Planning and execution therefore operate directly over latent actions rather than token-level primitives.

3.3 Learning Latent Actions from Trajectories

Latent actions are learned directly from agent trajectories by identifying recurrent multi-step behaviors that function as stable decision units. Rather than treating each action a_t as atomic, we consider action segments, which may span multiple decision steps or structured sub-sequences within a single action. Such segments capture extended behaviors that recur across trajectories.

We characterize behavioral recurrence using transition equivalence. Let $\mathcal{T}$ denote the transition dynamics induced jointly by the agent and environment. Two trajectory segments a and a' are said to be transition-equivalent if, for any preceding action history h, the induced transitions satisfy $\mathcal{T}(h \circ a) \approx \mathcal{T}(h \circ a')$, where $\circ$ denotes sequence concatenation and $\approx$ denotes equivalence up to task-relevant outcomes. In practice, this equivalence is not enforced universally but is approximated on the empirical trajectory distribution induced by the agent.

A latent action $z \in \mathcal{Z}$ is defined as an equivalence class of trajectory segments under this relation. To integrate latent actions into the agent, each z is parameterized with a compact representation consumable by the language model during inference; in our implementation, this is realized via dedicated vocabulary symbols, though the modeling abstraction does not depend on this choice.

To preserve behavioral consistency, we align the model's predictive distributions conditioned on latent actions with those conditioned on their corresponding trajectory segments using trajectory-level distillation. This alignment ensures that latent actions faithfully represent the decision semantics of the original agent.

3.4 Executable Latent Actions

Not all latent actions are executable. In our framework, executability subsumes both syntactic validity under external interfaces and semantic correctness with respect to agent environment interaction. Specifically, while actions must remain decodable and interpretable by downstream systems, true executability further requires that replacing concrete trajectory segments with a latent action does not alter the induced transition behavior.

Formally, a latent action z is executable if all trajectory segments belonging to z are transition-equivalent. That is, for any two realizations $a, a' \in z$ and any preceding history h, executability requires that $\mathcal{T}(h \circ a) \approx \mathcal{T}(h \circ a')$. This condition ensures that replacing a concrete trajectory segment with the corresponding latent action does not change downstream behavior.

This executability condition defines a semantic constraint on the latent action space. Latent actions that satisfy this constraint correspond to behaviors whose effects are invariant across contexts, whereas actions whose effects depend on task-specific parameters or contextual bindings violate transition equivalence and cannot be safely abstracted.

3.5 Applicability and Failure Modes

LAR is effective when the agent's action space contains a substantial subset of executable latent actions. In such settings, multiple explicit action segments correspond to the same transition behavior, allowing action abstraction to reduce the effective decision horizon without affecting task execution.

Failure arises when abstraction merges action segments that are not transition-equivalent. In this case, a latent action no longer represents a single, well-defined behavior, and replacing concrete actions with the latent action alters the induced transitions. This leads to incorrect agent–environment interaction and results in sharp degradation in task performance.

Accordingly, LAR operates by restricting abstraction to latent actions that satisfy the executability condition. The method does not seek maximal compression, but instead identifies the largest subset of executable latent actions that preserves transition behavior, ensuring reliable decision making under a compressed action representation.

4 Experimental Setup

Agent Models: We evaluate LAR on two widely used instruction-tuned LLMs: Meta-Llama-3.1-8B-Instruct (Dubey et al., 2024) and Qwen3-8B (Yang et al., 2025a). These models differ in pretraining data, alignment procedures, and architectural choices, allowing us to assess whether action space reparameterization generalizes across model families. For each model, latent actions are learned exclusively from its own rollout trajectories.

Benchmarks: We consider a diverse set of LLM agent benchmarks covering different interaction patterns and action structures. TriviaQA (Joshi et al., 2017) represents multi-step reasoning tasks; HumanEval (Chen, 2021), MBPP (Austin et al., 2021), and KodCode (Xu et al., 2025) represent code-generation tasks with highly structured action patterns; and Mind2Web (Deng et al., 2023) represents web-based, tool-using tasks with rich interaction scaffolds. For code benchmarks, latent actions are learned jointly across datasets to evaluate cross-task generalization within a shared action style.

Baselines: We compare LAR against vanilla LLM agents and ReAct-style agents, as well as representative efficiency-oriented methods operating at different stages of the agent pipeline, including token-level generation control (TokenSkip (Xia et al., 2025), ConciseHint (Tang et al., 2025)) and context/memory optimization (ACON (Kang et al., 2025)).

Evaluation Metrics: We report task-specific performance metrics (e.g., accuracy or success rate) and the relative reduction in action token counts compared to the original (Vanilla) trajectories. All methods are evaluated using identical decoding settings and hardware configurations. More experimental information is detailed in Appendix A.

5 Results and Analysis

5.1 Main Results: Performance and Efficiency Analysis

Table 1 reports task performance alongside the relative reduction in action tokens (in parentheses, compared to the original Vanilla trajectories). We analyze the results from three perspectives: the accuracy–efficiency trade-off, robustness across interaction regimes and backbones, and overall comparison with baselines.

Accuracy–Efficiency Trade-off under Action Reparameterization: As shown in Table 1, LAR consistently achieves favorable accuracy–efficiency trade-offs across backbones and benchmarks. In most settings, LAR reduces the effective decision horizon, reflected in fewer action tokens, while preserving or improving task success rates. For instance, on TriviaQA with Qwen3-8B, LAR attains

Table 1: Main results on three representative LLM agent benchmarks. We compare Latent Action Reparameterization (LAR) with general prompting baselines (Vanilla, CoT, ReAct) and efficiency-oriented methods operating at the token or context level (TokenSkip, ACON, ConciseHint), across two backbone models. Numbers report task performance, with parentheses indicating the relative change in action tokens. The best performance for each backbone and benchmark is highlighted in **bold**.

Backbone	Method	TriviaQA	KodCode	Mind2Web
Qwen3-8B	Vanilla	67.40	34.44	36.73
	COT	69.43	35.10	34.15
	ReAct	78.67	**58.94**	-
	TokenSkip	57.02 (-28.66%)	29.80 (-28.44%)	31.13 (-16.57%)
	ACON	55.33 (-27.9%)	28.67 (-22.5%)	30.77 (-17.7%)
	ConciseHint	68.69 (-12.7%)	28.47 (-12.5%)	35.33 (+11.9%)
	LAR	**80.09** (-27.06%)	58.28 (-9.2%)	**39.84** (-2.9%)
Llama-3.1 8B-Instruct	Vanilla	73.63	31.13	24.40
	COT	**75.50**	22.52	13.85
	ReAct	59.88	33.11	-
	TokenSkip	68.87 (-11.98%)	26.03 (-5.79%)	14.27 (-1.87%)
	ACON	57.14 (-25.4%)	24.67 (-23.1%)	15.63 (-16.7%)
	ConciseHint	67.32 (-13.5%)	25.83 (-12%)	17.33 (-13%)
	LAR	71.26 (-23.3%)	**33.11** (-9.8%)	**28.30** (-20.81%)

80.09 accuracy while reducing tokens by 27.06%, outperforming Vanilla and CoT and slightly exceeding ReAct. On Mind2Web, LAR achieves the highest accuracy (39.84) despite a shorter decision horizon. These results suggest that LAR primarily eliminates redundant decision steps rather than semantically critical ones.

Unlike TokenSkip, ACON, and ConciseHint, which intervene at the token generation or conditioning stage and may destabilize decision semantics, LAR reparameterizes the action space itself into executable latent units, preserving task-relevant transition behavior. The results also reveal abstraction boundaries. On TriviaQA with Llama-3.1-8B-Instruct, LAR achieves a 23.3% token reduction but incurs a slight accuracy drop versus Vanilla and CoT, suggesting that when structurally redundant action segments are limited, further abstraction approaches the boundary of semantic decision-making rather than indicating instability. Overall, LAR occupies a more favorable region in accuracy–efficiency space, supporting the conclusion that the effective decision horizon, rather than token count alone, is the dominant factor governing LLM agent efficiency.

Robustness across Heterogeneous Interaction Regimes and Backbone Behaviors: Table 1 further shows that LAR generalizes robustly across benchmarks with distinct interaction structures: reasoning-intensive tasks (TriviaQA), highly structured code generation (KodCode), and tool-using web interaction (Mind2Web). These tasks differ in the relative proportions of structural redundancy and parameter-binding semantic content. TriviaQA combines multi-step reasoning with external retrieval calls; KodCode exhibits stable syntactic and semantic scaffolds; and Mind2Web interleaves protocol-constrained tool invocations, textual reasoning, and parameter specification.

On KodCode, LAR matches or closely approaches the strongest baselines across both backbones while reducing the decision horizon by approximately 9 to 10%, suggesting that code-generation tasks contain substantial executable structural redundancy that can be safely abstracted. On Mind2Web, the effect is more strongly modulated by backbone behavior: with Qwen3-8B, LAR achieves the highest accuracy with a modest token reduction, while with Llama-3.1-8B-Instruct, it delivers both a substantial accuracy improvement and a large reduction in decision horizon. This discrepancy reflects differences in generation style, as the amount of intermediate text produced by the backbone determines the fraction of action content that can be safely abstracted.

Overall, LAR demonstrates stable benefits across heterogeneous interaction regimes, with its effectiveness jointly determined by task-level action structure and backbone-specific generation behavior,

indicating that action representation learning serves as a general mechanism complementary to model- and system-level optimizations for efficient LLM agent inference.

Overall Performance Comparison: Across all benchmarks and backbones, LAR consistently achieves the best or near-best task performance among efficiency-oriented methods, while substantially outperforming TokenSkip, ACON, and ConciseHint. These baselines reduce inference cost by pruning token generation or compressing context, but frequently incur severe performance degradation on structured or interactive tasks such as KodCode and Mind2Web, highlighting a fundamental limitation of efficiency methods that do not preserve environment-facing transition semantics.

In contrast, LAR maintains strong task performance while operating over a reparameterized latent action space. On Qwen3-8B, LAR achieves the highest accuracy on TriviaQA and Mind2Web, and matches ReAct on KodCode, all under a significantly reduced effective decision horizon. This suggests that LAR gains arise from reparameterizing the unit of decision making rather than from token- or context-level compression. A similar trend holds on Llama-3.1-8B-Instruct: LAR remains competitive with the strongest prompting baselines and consistently outperforms efficiency methods lacking executability constraints. Even where LAR does not strictly exceed the best baseline, such as KodCode on Qwen3-8B, it achieves comparable task success while operating over a more compact decision space.

Overall, these results support our central claim that preserving transition semantics is critical for favorable efficiency–performance trade-offs. By constructing latent actions under an executability constraint, LAR eliminates structural redundancy in agent behavior while preserving environment-facing transition behavior, enabling efficient inference without sacrificing task success.

5.2 Cross-Benchmark Generalization of Latent Actions within the Code Generation Domain

This experiment evaluates whether latent actions learned by LAR capture reusable decision structure or merely encode dataset-specific artifacts. Specifically, we train latent actions using trajectories collected on KodCode and directly apply the resulting action reparameterization to other code-generation benchmarks, HumanEval and MBPP, without any retraining or adaptation. ReAct is used as the baseline under its standard prompting-based, multi-step interaction setting. This setup directly tests the core assumption underlying LAR's design.

If latent actions correspond to transition-equivalent and executable decision units (Section 3), they should generalize across benchmarks that share a common action domain even when surface distributions differ. Conversely, if they merely overfit to dataset-specific patterns, their effectiveness should deteriorate when transferred to unseen benchmarks.

As shown in Table 2, LAR demonstrates strong cross-benchmark generalization across both backbones. When trained on KodCode and evaluated on HumanEval and MBPP, LAR consistently matches or outperforms ReAct without task-specific retraining or prompt engineering, despite not being trained or tuned on these target benchmarks. This behavior indicates that LAR learns domain-level action abstractions for code generation, rather than benchmark-specific heuristics. In particular, the latent actions appear to encode stable structural behaviors such as code scaffolding and formatting patterns that remain valid across datasets.

Notably, this generalization is achieved without task-specific retraining or prompt engineering. While ReAct relies solely on prompting and interaction at test time, LAR reuses a fixed latent action space learned from KodCode trajectories. The fact that this learned representation remains effective on HumanEval and MBPP suggests that LAR's inductive bias arises from action reparameterization itself rather than from dataset alignment or tuning effects.

The magnitude of cross-benchmark gains also reflects differences in structural regularity across benchmarks. HumanEval and MBPP exhibit more standardized code patterns and shorter generation horizons than KodCode, which contains more diverse and complex interactions. Consistent with the executability constraint in LAR, benchmarks with higher structural regularity provide a larger subset of transition-invariant action segments that can be safely abstracted, leading to clearer performance improvements after transfer.

Table 2: Cross-benchmark generalization within the code generation domain. Latent actions are learned from KodCode trajectories and directly evaluated on HumanEval and MBPP without retraining or adaptation. LAR is compared against ReAct under identical backbone settings.

Backbone	Method	HumanEval	MBPP	KodCode
Qwen3-8B	ReAct	89.63	74.17	58.94
	LAR	91.46	75.50	58.28
Llama-3.1 8B-Instruct	ReAct	56.71	48.40	33.11
	LAR	60.37	46.60	33.11

At the same time, performance on KodCode remains stable, with LAR closely matching ReAct across both backbones. This parity indicates that LAR does not trade in-domain performance for cross-benchmark generalization. Instead, it preserves task-level behavior on the training benchmark while improving robustness and transferability elsewhere.

Overall, these results provide strong empirical evidence that latent actions learned by LAR are neither task-specific hacks nor benchmark-bound templates. Rather, they constitute reusable, executable decision abstractions that generalize across datasets within a shared action domain, validating the trajectory-based learning and executability assumptions central to the LAR framework.

5.3 Progressive Abstraction Ablation and the Boundary of Executable Latent Actions

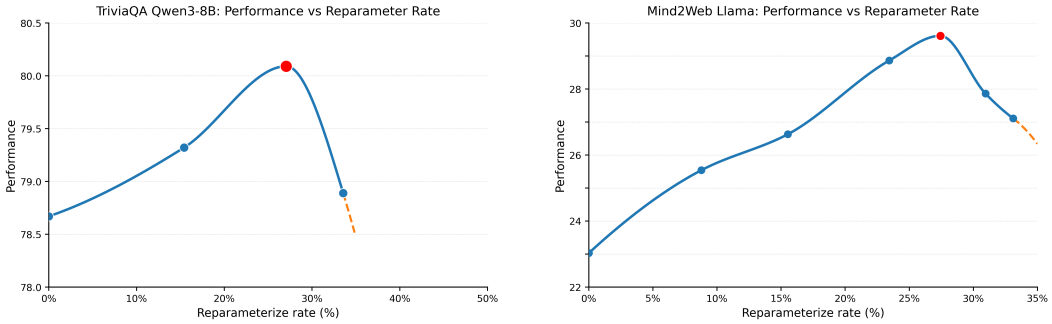
This experiment investigates the question: where the abstraction boundary lies for latent action reparameterization, through a structured ablation over abstraction strength. While previous results show that moderate abstraction can improve efficiency without harming performance, we conduct a progressive abstraction ablation to characterize both the beneficial regime and the failure mode of LAR by gradually increasing the reparameterization rate and observing when task performance begins to degrade. This analysis directly corresponds to the failure modes discussed in Section 3.5.

Experimental Design (Progressive Abstraction Ablation): We perform a controlled ablation by progressively increasing the reparameterization rate, defined as the proportion of action segments replaced by latent actions, while keeping the backbone model and decoding settings fixed. This ablation systematically varies the degree of abstraction applied to the agent's action space, allowing us to trace how performance evolves as abstraction moves from low-entropy structural components toward high-entropy parameterized components.

Phase I: Moderate Abstraction (Ablation Regime): In the low-to-moderate abstraction regime of this ablation, performance consistently improves alongside inference efficiency. This regime corresponds to selectively abstracting low-entropy structural components, such as recurring scaffolds, protocol formats, and stable interaction patterns. Removing these redundant decision steps shortens the effective decision horizon and reduces inference cost, while preserving task-relevant semantics. The observed performance gains in this ablation regime indicate that moderate abstraction removes redundant or noisy action fragments rather than eliminating essential decisions. This trend is visually illustrated in Fig. 2, which shows a consistent performance increase as low-entropy structural components are progressively abstracted.

Phase II: Abstraction Boundary: As the reparameterization rate continues to increase, performance reaches a peak marking the abstraction boundary. At this point, most transition-invariant, executable components have been safely abstracted and further abstraction yields no additional benefit. This peak empirically delineates the maximal scope of abstraction that preserves transition equivalence and executability.

Phase III: Performance Collapse: Beyond the abstraction boundary, performance degrades sharply. This collapse occurs when the ablation begins to include high-entropy parameterized components, such as queries, entity references, or task-specific arguments. Abstracting these components violates the executability condition, as latent actions no longer correspond to transition-equivalent behaviors across contexts. Notably, the degradation is abrupt rather than gradual, reflecting a semantic failure mode caused by broken environment-facing transitions rather than insufficient modeling capacity.



(a) TriviaQA (Qwen3-8B) (b) Mind2Web (Llama-3.1-8B-Instruct)

Figure 2: Progressive abstraction ablation results. Task performance as a function of the reparameterization rate for two representative settings. Moderate abstraction improves performance by eliminating low-entropy structural redundancy, while excessive abstraction leads to a sharp performance collapse once high-entropy, parameterized components are abstracted.

Cross-Task Consistency: The three-phase behavior along the abstraction ablation axis—performance improvement, peak, and collapse—appears consistently across both Mind2Web and TriviaQA. While the exact location of the abstraction boundary varies with task and backbone, the qualitative trend is shared across domains, suggesting that the abstraction boundary is a structural property of action reparameterization rather than an artifact of a specific task.

Method-Level Takeaway: This ablation provides direct empirical support for LAR's design choice to restrict abstraction to executable latent actions. Rather than aiming for maximal compression, LAR identifies the largest subset of low-entropy, transition-invariant action segments that can be safely abstracted. The observed performance collapse beyond the abstraction boundary validates this restriction and highlights why executability fundamentally constrains useful action representations.

6 Case Analysis

We conduct a case analysis on a multi-step reasoning instance from TriviaQA to illustrate how LAR reshapes the agent's action structure. As shown in Fig. 3, the vanilla agent generates a long sequence of fine-grained textual actions for information retrieval and answer construction, many of which correspond to recurrent structural scaffolds such as reasoning templates and tool invocation formats, rather than task-critical semantic decisions.

Under LAR, this trajectory is reformulated into two classes of actions: executable latent actions and explicit high-entropy parameterized components. Low-entropy, recurrent structural patterns are abstracted into latent actions representing transition-equivalent behaviors, while parameter-binding elements that determine task semantics, including the concrete search query and final answer, remain explicit to ensure executability.

This case study highlights that LAR shortens the effective decision horizon by abstracting transition-invariant structure, while preserving correct interaction with external tools by retaining parameter-rich components. Together, these effects illustrate how selective action reparameterization improves inference efficiency without compromising semantic correctness or executability.

7 Conclusion

We introduced Latent Action Reparameterization (LAR), a principled framework that improves LLM-agent efficiency by treating *action representation* as a first-class modeling choice. By learning executable latent actions that compress recurrent low-entropy structures while preserving high-entropy, parameter-binding content, LAR redefines the unit of decision making and directly addresses the inefficiency caused by overly fine-grained action interfaces, achieving favorable performance–efficiency trade-offs across diverse tasks and backbone models.

IMPACT STATEMENT

This paper introduces Latent Action Reparameterization (LAR), a general framework for improving the efficiency of LLM-based agents by learning executable latent action representations from trajectories. By reducing the effective decision horizon while preserving environment-facing semantics, LAR can lower inference cost and latency for agent systems in practical applications such as information retrieval, code generation, and tool-based workflows, potentially improving scalability and accessibility. We release our code to support reproducibility and independent assessment.

REFERENCES

Mostafa Al-Emran. Hierarchical reinforcement learning: a survey. *International journal of computing and digital systems*, 4(02), 2015.

Christopher Amato, George Konidaris, Leslie P Kaelbling, and Jonathan P How. Modeling and planning with macro-actions in decentralized pomdps. *Journal of Artificial Intelligence Research*, 64:817–859, 2019.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Yuxuan Cai, Xiaozhuan Liang, Xinghua Wang, Jin Ma, Haijin Liang, Jinwen Luo, Xinyu Zuo, Lisheng Duan, Yuyang Yin, and Xi Chen. Fastmtp: Accelerating llm inference with enhanced multi-token prediction. *arXiv preprint arXiv:2509.18362*, 2025.

Dingyang Chen, Qi Zhang, and Yinglun Zhu. Efficient sequential decision making with large language models. *arXiv preprint arXiv:2406.12125*, 2024a.

Hao Mark Chen, Wayne Luk, Ka Fai Cedric Yiu, Rui Li, Konstantin Mishchenko, Stylianos I Venieris, and Hongxiang Fan. Hardware-aware parallel prompt decoding for memory-efficient acceleration of llm inference. *arXiv preprint arXiv:2405.18628*, 2024b.

Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*, 2025.

Tonmoy Debnath, Md Nurul Absar Siddiky, Muhammad Enayetur Rahman, Prosenjit Das, Antu Kumar Guha, Muhammad Rezaur Rahman, and HM Kabir. A comprehensive survey of prompt engineering techniques in large language models. *TechRxiv*, 2025.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

Luca Gioacchini, Giuseppe Siracusano, Davide Sanvito, Kiril Gashteovski, David Friede, Roberto Bifulco, and Carolin Lawrence. Agentquest: A modular benchmark framework to measure progress and improve llm agents. *arXiv preprint arXiv:2404.06411*, 2024.

Gonzalo Gonzalez-Pumariega, Leong Su Yean, Neha Sunkara, and Sanjiban Choudhury. Robotouille: An asynchronous planning benchmark for llm agents. *arXiv preprint arXiv:2502.05227*, 2025.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. Acon: Optimizing context compression for long-horizon llm agents. *arXiv preprint arXiv:2510.00615*, 2025.

Jeonghye Kim, Sojeong Rhee, Minbeom Kim, Dohyung Kim, Sangmook Lee, Youngchul Sung, and Kyomin Jung. Reflact: World-grounded decision making in llm agents via goal-state reflection. *arXiv preprint arXiv:2505.15182*, 2025.

Minbeom Kim, Hwanhee Lee, Kang Min Yoo, Joonsuk Park, Hwaran Lee, and Kyomin Jung. Critic-guided decoding for controlled text generation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 4598–4612, 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Zekun Li, Baolin Peng, Pengcheng He, Michel Galley, Jianfeng Gao, and Xifeng Yan. Guiding large language models via directional stimulus prompting. *Advances in Neural Information Processing Systems*, 36:62630–62656, 2023.

Jiayu Liu, Cheng Qian, Zhaochen Su, Qing Zong, Shijue Huang, Bingxiang He, and Yi R Fung. Costbench: Evaluating multi-turn cost-optimal planning and adaptation in dynamic environments for llm tool-use agents. *arXiv preprint arXiv:2511.02734*, 2025.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: Towards llms as operating systems. 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. Distilling reasoning capabilities into smaller language models. *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 7059–7073, 2023.

Siao Tang, Xinyin Ma, Gongfan Fang, and Xinchao Wang. Concisehint: Boosting efficient reasoning via continuous concise hints during generation. *arXiv preprint arXiv:2506.18810*, 2025.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, et al. Efficient large language models: A survey. *arXiv preprint arXiv:2312.03863*, 2023.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*, 2025.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. *arXiv preprint arXiv:2503.02951*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Ruihan Yang, Yikai Zhang, Aili Chen, Xintao Wang, Siyu Yuan, Jiangjie Chen, Deqing Yang, and Yanghua Xiao. Aria: Training language agents with intention-driven reward aggregation. *arXiv preprint arXiv:2506.00539*, 2025b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Yuanzhao Zhai, Tingkai Yang, Kele Xu, Dawei Feng, Cheng Yang, Bo Ding, and Huaimin Wang. Enhancing decision-making for llm agents via step-level q-value models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 27161–27169, 2025.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025.

Ruijie Zheng, Ching-An Cheng, Hal Daumé III, Furong Huang, and Andrey Kolobov. Prise: Llm-style sequence compression for learning temporal action abstractions in control. *arXiv preprint arXiv:2402.10450*, 2024.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, et al. A survey on efficient inference for large language models, 2024. *URL https://arxiv. org/abs/2404.14294*, 2024.

A Detailed Experimental Setup and Design Rationale

A.1 Agent Models and Training Protocol

We conduct experiments using **Meta-Llama-3.1-8B-Instruct** and **Qwen3-8B** to ensure that observed efficiency gains are not model-specific. No architectural modifications or additional fine-tuning are applied to the base models. Latent action vocabularies are constructed separately for each model using trajectories generated by the same model, ensuring that reparameterization does not rely on cross-model transfer.

A.2 Benchmark Selection and Task Categorization

Benchmarks are selected to span distinct agent behaviors and action structures. **TriviaQA** emphasizes multi-step reasoning with relatively low structural repetition. Code benchmarks (**HumanEval**, **MBPP**, **KodCode**) exhibit strong syntactic regularities and recurring generation patterns, making them suitable for studying reusable semantic action units. **Mind2Web** involves complex tool interactions with extensive protocol-level scaffolds and repeated system-level configurations, providing a setting where executable action abstraction is particularly impactful.

A.3 Joint Latent Action Learning for Code Tasks

For code-generation benchmarks, we learn a shared latent action space across **HumanEval**, **MBPP**, and **KodCode**. This design reflects the observation that these tasks share similar action scaffolds (e.g., function definitions, formatting conventions, and control structures). Joint learning enables us to test whether latent actions capture task-agnostic semantic behaviors rather than dataset-specific artifacts, and to evaluate generalization across code tasks.

A.4 Baseline Methods

Baselines are chosen to represent distinct efficiency paradigms:

- **Vanilla LLM agents**, which operate directly over token-level actions;
- **ReAct-style agents**, which interleave reasoning and acting;
- **Token-level efficiency methods**, which regulate generation dynamics (TokenSkip, Concise-Hint);
- **Context and memory optimization methods**, which compress interaction histories (ACON).

All baselines preserve the original decision interface and do not alter the action space representation.

A.5 Baseline Evaluation

- **Vanilla LLM agents**, which operate directly over token-level actions;
- **ReAct-style agents**, which interleave reasoning and acting;
- **TokenSkip** is adapted to the COT prompt template of ours, and the cutoff lengths for LoRA adapter training are set to 4096 for **TriviaQA** and **Mind2Web**, and 8192 for **KodCode**; the compression ratio is set to 0.7, while the other settings are left unchanged.
- **ACON** was modified to use Qwen3-8B as both the compressor and generator, where we set the maximum generated tokens to 8192, and we tested n cases for the benchmarks.
- **Context and memory optimization methods**, which compress interaction histories (ACON).

A.6 Metrics and Evaluation Protocol

In addition to task performance, we measure efficiency at both the decision and system levels. The **effective decision horizon** is defined as the total number of explicit generation decisions, aligning

Table 3: The experimental results for the three Mind2Web sub-test sets.

Backbone	Method	Cross-Task	Cross-Website	Cross-Domain
Qwen3-8B	Vanilla	39.56	17.58	30.69
	LAR	**45.05**	**26.37**	**35.64**
Llama-3.1 8B-Instruct	Vanilla	22.76	8.13	22.44
	LAR	**27.64**	**11.38**	**26.69**

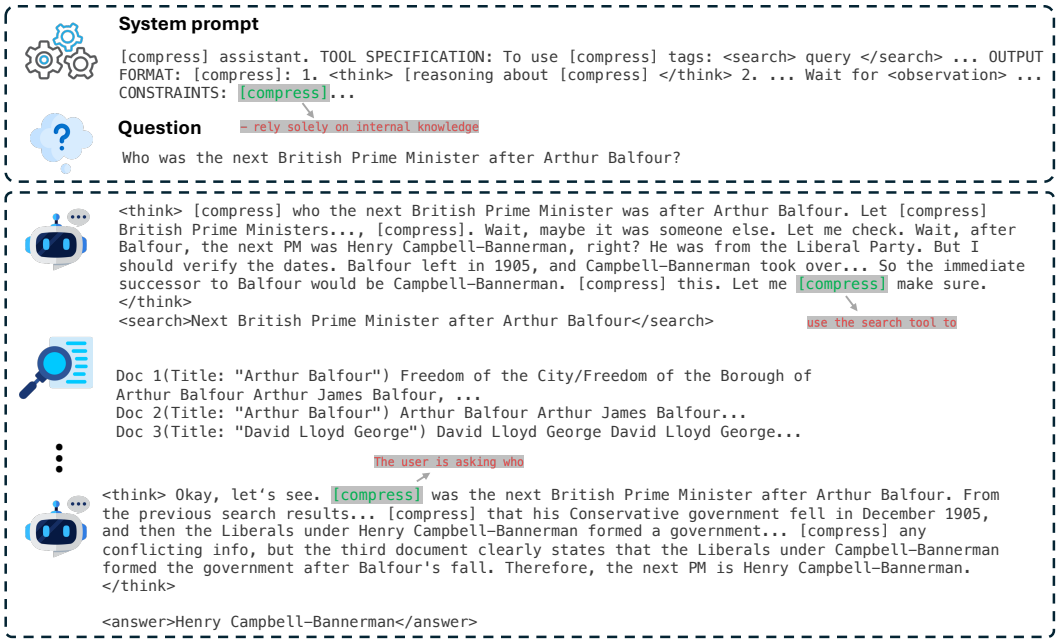


Figure 3: Case analysis of LAR on a TriviaQA example. LAR abstracts low-entropy structural components into executable latent actions while preserving high-entropy parameterized content (e.g., the search query), reducing the effective decision horizon without altering task execution.

with the formal definition in Section 3. Wall-clock inference time is measured under identical hardware, decoding parameters, and maximum context lengths for all methods. Token reduction is reported as an outcome of action reparameterization rather than an explicit optimization objective.

A.7 Reproducibility and Implementation Details

All models and baselines are trained and tested on servers with 8×H200 140GB cards. For LAR training, we set batch_size $=$ 16, epochs $=$ 3, learning_rate $=$ 1e-4. We use vLLM and set temperature $=$ 0 for deterministic decoding. All experiments are conducted with fixed random seeds.

A.8 Detailed experimental result of Mind2Web

Mind2Web provides a unique opportunity to evaluate generalizability at three different levels: cross domains, cross websites, and cross tasks. The experimental result is shown in Table 3, which shows that LAR can effectively compress redundant HTML context and help LLM-based agents generate more precise actions.

A.9 Detailed Case Analysis and Trajectory Example

Figure 3 displays a complete trajectory on a TriviaQA task. The task requires the agent to identify a famous person based on a description. The Vanilla agent follows a standard ReAct-style approach,

generating a sequence of thoughts, tool actions, and a final answer. This process involves generating a large number of tokens, many of which are structural and serve only to format the interaction.

LAR reparameterizes this trajectory by identifying and abstracting these recurrent structural patterns into latent actions. As shown in the figure, the lengthy sequence of tokens corresponding to the search action is compressed into a single latent token. Crucially, the high-entropy content, the search query "Next British Prime Minister after Arthur Balfour", is preserved explicitly to maintain executability.

This transformation results in a significantly shorter effective decision horizon. The Vanilla trajectory requires many steps to express the intent, whereas the LAR trajectory expresses the same high-level decisions in far fewer steps. By operating over these latent actions, the agent can plan and execute at a higher level of abstraction, reducing computational cost while preserving the integrity of the interaction with the environment and the final output. This example highlights how LAR selectively compresses structural redundancy while maintaining the necessary granularity for effective task performance.