When Further Realization Is Unnecessary: Amortized Reasoning for Long-Horizon LLM Agents

Rongzheng Wang[1], Jiakai Li[1], Renzhong Wang[1],
Rongwei Wang[2], Yihong Huang[1], Dingyuan Rao[3],
Jielei Wang[1]†, Shuang Liang[1]*, Ke Qin[1]†
[1]Institute of Intelligent Computing,
University of Electronic Science and Technology of China, Chengdu, China
[2]Tsinghua Shenzhen International Graduate School, Shenzhen, China
[3]R&D Dept, Chengdu Jinfa Edge Intelligence Technology Co., Ltd.
wangrongzheng@std.uestc.edu.cn, shuangliang@uestc.edu.cn

Abstract

Long-horizon LLM agents solve interactive tasks by repeatedly choosing the next action from the current state. At many steps, a compact decision is sufficient to specify the action, yet the base agent still emits a full realization in its native output format (e.g., code or action text). Once the compact decision determines the executable content of the step, subsequent generation mostly adds formatting and elaboration, yielding redundant realization. This separation is reflected inside the decoder: compact decisions become locally recoverable in middle layer representations before full realizations, and attention over its token span becomes concentrated when the decision is sufficiently supported as the current step output. Building on this separation, we propose **MIRA**, a training-free inference method for Model-Internal Reasoning Amortization. MIRA amortizes realization through two complementary stages: Consensus-Guided Decision retrieves successful reference steps in the middle layer representation space to propose a compact decision candidate. Attention-Guided Commitment evaluates its token span using token confidence and attention concentration. The candidate is used as the step output only when both signals support it for the current state; otherwise, the agent falls back to the base agent's full realization path. Experiments on long-horizon interactive agents show that MIRA improves task performance while reducing redundant realization, yielding a 47.5% reduction in output tokens and a 24.7% reduction in inference time across tasks and agent formats.

1 Introduction

Long-horizon LLM agents [30] extend language models from single response generation to interactive decision making [37, 25]. Across interactive benchmarks [26, 31, 36], a long-horizon agent repeatedly chooses the next action from the current state using observations, memory, and task context. At many steps, this next action is specified by a compact decision, such as which tool to call, which API arguments to issue, or which structured action to emit [22, 15, 20, 19]. Yet the base agent may still realize this decision in its full native output format, such as code or action text [33, 6, 32]. Once the compact decision has been determined, the remaining generation mainly repeats this settled executable content through formatting and elaboration, adding realization cost and latency.

*Corresponding author.
†Also affiliated with the Ubiquitous Intelligence and Trusted Services Key Laboratory of Sichuan Province.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Existing efficiency methods for agents mainly reduce cost through external interventions, such as prompt compression [10], interface redesign [9], smaller policies [27, 34], or stopping criteria based on intermediate outputs [23, 35]. Although these approaches improve efficiency, they leave a central model-internal question unresolved: when has the compact decision at a step been determined, and when does the model still need the base agent's full realization to determine the step output? Rather than treating every step as requiring the same native output format, efficient reasoning should distinguish states where the compact decision can serve as the step output from states that still require full realization. This gives long-horizon agent inference an amortization objective [7, 12, 8]: full realization should be used only when the compact decision lacks sufficient support.

The decoder exposes signals that guide this amortization. During the generation of a step, compact decisions become locally recoverable from middle decoder representations before full realizations are resolved. However, local recoverability alone does not justify using a candidate as the current step output. Once a candidate is proposed, attention over its token span becomes more concentrated when it is supported for the current state, whereas rejected candidate token spans show weaker concentration. These patterns suggest that efficient agent reasoning should amortize realization selectively, using a compact decision only when it is both stabilized in middle layer representations and supported by decoder evidence.

Based on this view, we propose **MIRA** (**M**odel-**I**nternal **R**easoning **A**mortization), a training-free inference method for amortizing realization in long-horizon agents. MIRA amortizes realization through two complementary stages: Consensus-Guided Decision (CGD) retrieves successful reference steps in the middle layer representation space to propose a compact decision candidate. Attention-Guided Commitment (AGC) then evaluates its token span based on token confidence and attention concentration. The candidate is used as the step output only when both signals support it for the current state; otherwise, the agent falls back to the base agent's full realization path.

We evaluate MIRA on ALFWorld [26], ScienceWorld [31], and WebShop [36], covering three representative agent formats: direct-planning, code-based, and skill-based agents. MIRA preserves or improves task scores while reducing redundant realization, yielding a 47.5% reduction in output tokens and a 24.7% reduction in inference time relative to the corresponding base agents. Our contributions are as follows: (1) we formulate efficient long-horizon agent inference as amortized reasoning over compact decisions and full realizations; (2) we identify decoder signals that distinguish whether a compact decision has stabilized from whether it is supported as the step output in the current state; (3) we instantiate these signals in MIRA and show that amortized realization reduces inference cost while improving task performance across tasks and agent formats.

2 Related Work

Long-Horizon Agents and Benchmarks. Long-horizon LLM agents use language models as interactive decision makers that repeatedly condition on task context, state, observations, and memory. Existing agents differ in how they realize each step. ReAct [37] represents the next step as textual reasoning and action, where the executable part is a structured action text. ReCode [38] realizes steps through executable code and recursive plan-action expansion, where the compact decision can correspond to a code stage or procedure skeleton. SkillNet [17] represents reusable procedures as skill invocations, where the compact decision specifies the skill and its arguments. These output formats define where compact decisions and full realizations arise in long-horizon interaction. Benchmarks such as ALFWorld [26], ScienceWorld [31], WebShop [36], WebArena [40], Mind2Web [5], and VisualWebArena [13] evaluate such agents across household, science, shopping, and web tasks.

Efficient Inference. Efficiency methods for language models reduce generation cost by shortening inputs, allocating computation adaptively, or stopping generation when further tokens are unlikely to help. Prompt compression reduces context length [10], early exiting reduces computation through intermediate exits [27, 34], and confidence based adaptive generation stops or continues decoding according to output confidence [23, 35]. In long-horizon agents, efficiency methods often operate through the agent interface. AutoTool [9], for example, improves efficiency by optimizing tool selection and tool use. These methods mainly control context size, computation budget, or tool interface. However, the step-level distinction between a compact executable decision and its full realization remains a separate source of efficiency.

Memory Reuse and Decoder Signals. Retrieval and representation reuse provide precedents for matching a current state with successful references. Case based reasoning stores previous solutions [1], and kNN-LM uses nearest neighbors in representation space to support language modeling [11]. Retrieval-augmented generation [14] and RETRO [2] condition generation on retrieved context, while prefix conditioning [16] injects guidance through latent states without changing the base model. Decoder signals provide another basis for support checking: attention analyses study model-internal computation [28, 3, 29], attention entropy measures concentration [24], and confidence based adaptive generation uses token probabilities for generation control [23, 35].

3 When Can Realization Be Amortized?

This section studies when full realization can be amortized at an agent step. We analyze this through two observations. The first concerns local recoverability: compact decisions become recoverable in middle decoder representations before full realizations. The second concerns candidate support: once a compact decision candidate is proposed, attention concentration over its token span distinguishes candidates that can be used as the step output from candidates that should be rejected.

3.1 Preliminaries

Stepwise Agent Interaction. At step t, the base agent operates on the current state s_t and action space Γ_t. The state s_t contains the task context, such as history, observation, and memory, while Γ_t specifies the actions, tools, or API calls available to the agent. The LLM input is $x_t = (s_t, \Gamma_t)$, and the generated step output is $y_t \sim p_\theta(\cdot \mid x_t)$. We distinguish two output forms:

$$y_t = \begin{cases} \delta_t, & \delta_t \in \mathcal{Y}^{\mathrm{dec}}, \\ \rho_t, & \rho_t \in \mathcal{Y}^{\mathrm{real}}. \end{cases} \tag{1}$$

where δ_t and ρ_t denote the compact decision and the full realization, respectively. The compact decision δ_t specifies the action to execute at the current step. The full realization ρ_t expresses that decision in the base agent's native output format. We define δ_t as the unit of amortization, whereas ρ_t is generated when the current model state does not sufficiently support using δ_t as the step output.

Consensus Entropy. We use Consensus entropy to measure whether an output form is locally recoverable from a decoder layer. For each layer ℓ, we retrieve successful reference steps using the pooled layer representation of the current input. Let $p_t^{\ell,\mathcal{Y}}(y)$ denote the label distribution of the retrieved references under output space $\mathcal{Y}$. We define the Consensus entropy of the current step at layer ℓ in output space $\mathcal{Y}$ as:

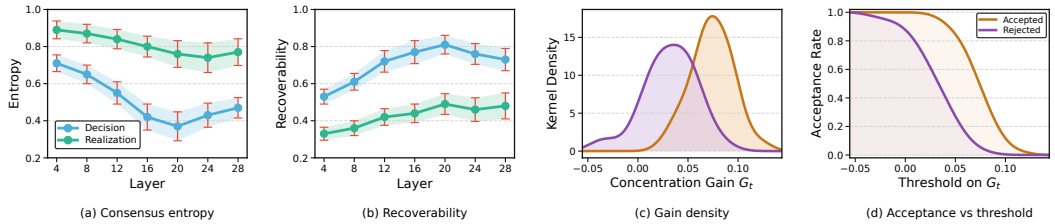
$$H^{\ell}_{\mathrm{cons},t}(\mathcal{Y}) = -\frac{1}{\log \min(K, |\mathcal{Y}|)} \sum_{y \in \mathcal{Y}} p_t^{\ell,\mathcal{Y}}(y) \log p_t^{\ell,\mathcal{Y}}(y). \tag{2}$$

Low Consensus entropy indicates that retrieved successful steps at layer ℓ concentrate on the same label. We compute $H^{\ell}_{\mathrm{cons},t}(\mathcal{Y})$ under the compact decision labeling and the full realization labeling using the same query steps and the same retrieved reference sets. This comparison tests which output form is more locally recoverable from the same layerwise retrieval result. Appendix F gives the retrieval construction.

Attention Entropy. Local recoverability alone does not determine whether a proposed candidate should be used as the step output. Candidate support is therefore evaluated from the decoder computation over its token span. Since each attention row is a probability distribution over visible positions [28], its entropy provides a normalized measure of attention concentration [24, 3, 29]. We use attention concentration as a diagnostic signal for candidate support. Let $D_t = (j_1, \ldots, j_m)$ denote the candidate token span under Γ_t, where j_1 and j_m are the first and last positions of the span. For the current input x_t, head h, layer l, and position $j \in D_t$, the normalized Attention entropy is:

$$H^{h,l}_{\mathrm{att},t}(j) = -\frac{1}{\log |\Omega_{t,j}|} \sum_{r \in \Omega_{t,j}} a^{h,l}_{t,j}(r) \log a^{h,l}_{t,j}(r), \tag{3}$$

where $a^{h,l}_{t,j}$ is the attention distribution over the visible positions $\Omega_{t,j}$. Lower Attention entropy indicates that attention is concentrated on fewer visible positions. We average $H^{h,l}_{\mathrm{att},t}(j)$ over heads



(a) Consensus entropy (b) Recoverability (c) Gain density (d) Acceptance vs threshold

Figure 1: Observation results on ALFWorld with Qwen3-8B. Panels (a) and (b) report the average over three base agents (ReAct [37], ReCode [38], and SkillNet [17]) with standard deviations, and compare layerwise Consensus entropy and recoverability under the compact decision labeling and the full realization labeling. Panels (c) and (d) pool accepted and rejected candidate cases and show the distribution and threshold behavior of span concentration gain G_t.

$h \in \mathcal{H}$ and layers $l \in \mathcal{L}$, and denote the result by $\bar{H}_{\text{att},t}(j)$. We then define the span concentration gain from the first position j_1 to the last position j_m as:

$$G_t = \frac{\bar{H}_{\text{att},t}(j_1) - \bar{H}_{\text{att},t}(j_m)}{\bar{H}_{\text{att},t}(j_1) + \epsilon}. \tag{4}$$

A larger G_t indicates a stronger decrease in Attention entropy across the candidate token span. We use this quantity as span-level evidence for whether a compact decision candidate is supported as the current step output. Appendix F gives the averaging rule and candidate construction.

3.2 Observation 1: Compact Decisions Become Recoverable Before Full Realizations

We first examine whether compact decisions become locally recoverable before full realizations. For the same query steps and retrieved reference sets, we compute Consensus entropy under two labelings: compact decisions and full realizations. As a complementary measure, we compute recoverability as the rate at which the majority label among retrieved references matches the query label under the same output form. Figure 1 (left two panels) shows a clear separation between the two labelings. In middle decoder layers, compact decisions exhibit lower Consensus entropy and higher recoverability than full realizations. This pattern indicates that compact decisions become locally recoverable in middle decoder representations before full realizations. The paired recovery-order diagnostic in Appendix C.3 additionally supports their earlier recovery across decoder layers.

3.3 Observation 2: Supported Candidates Show Stronger Span Concentration

We then examine whether a compact decision candidate is supported as the step output. For each step, we compare Decision Accepted cases, where the candidate contains the action and arguments required by the state, with Decision Rejected cases, where the candidate has a similar token span but contains missing, mismatched, or incorrect arguments. Figure 1 (right two panels) shows that Decision Accepted cases exhibit larger span concentration gain than Decision Rejected cases. This pattern indicates that, once a candidate is fixed, stronger concentration over the token span is associated with acceptance as the step output. These two observations suggest an amortization view of long-horizon agent inference: a compact decision can replace full realization when it is locally recoverable and supported for the state; otherwise, the agent retains the base realization path.

4 Method

We propose **MIRA** (**M**odel **I**nternal **R**easoning **A**mortization), a training-free inference method for amortizing realization in long-horizon agents. At each step, MIRA operates on the current state s_t and the available action space Γ_t. It first proposes a compact decision candidate from consensus among retrieved reference steps, and then evaluates whether the candidate is supported as the current step output. This determination is implemented by two complementary stages. *Consensus-Guided Decision* (CGD) relies on a Prefix Memory built from successful trajectories, which provides successful reference steps retrieved using the current middle layer representation. CGD then proposes a compact decision candidate from the retrieved reference steps. *Attention-Guided Commitment*

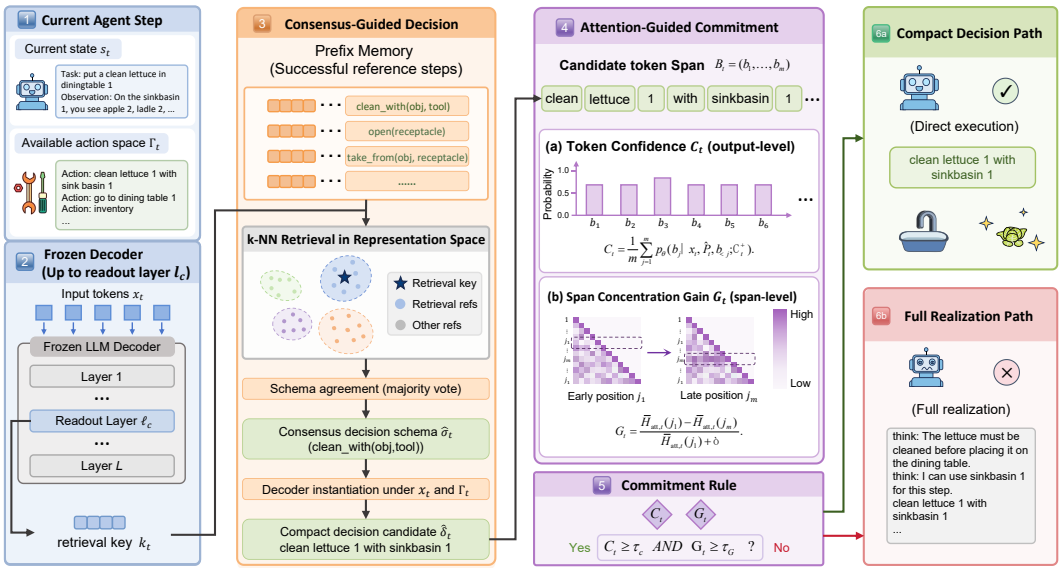


Figure 2: Overview of MIRA. MIRA forms a retrieval key from a middle decoder layer, retrieves successful reference steps from Prefix Memory, and uses CGD to instantiate a consensus decision schema into a compact decision candidate. AGC scores the candidate token span with token confidence C_t and span concentration gain G_t, using the candidate as the step output only when both signals support it and falling back to the base agent's full realization path otherwise.

(AGC) evaluates the candidate token span with token confidence and span concentration gain. MIRA uses the candidate as the step output when both signals support it for the current state. Otherwise, the agent falls back to the base agent's full realization path. Figure 2 gives an overview of this proposal and commitment process.

4.1 Consensus-Guided Decision

Consensus-Guided Decision (CGD) operationalizes local recoverability as the candidate proposal stage of MIRA. CGD represents each successful step by a state-conditioned decision form. The decision schema specifies the reusable operator and argument roles of a compact decision. The current context x_t and action space Γ_t define the valid candidate serializations, and the decoder instantiates the schema into concrete arguments under this constraint. Across action, code, and skill based agents, the schema records the reusable decision pattern at the current decision granularity, such as an action form, a code stage, or a skill invocation form. CGD first stores successful steps in Prefix Memory, then uses fixed-K retrieval to select a consensus schema, and performs decoder-side instantiation under Γ_t to produce a compact decision candidate. The resulting candidate is passed to AGC for the candidate support check.

Prefix Memory Construction. Prefix Memory stores successful steps in a form that can be retrieved by representation, compared by schema, and reused for instantiation. For each successful step of the same base agent, CGD stores:

$$m_i = (k_i, P_i, \sigma_i, \delta_i), \tag{5}$$

where δ_i is the successful compact decision instance, σ_i is the decision schema obtained by canonicalizing δ_i into its reusable operator and argument roles, k_i is the retrieval key obtained from the layer-ℓ_c hidden states by masked mean pooling followed by unit normalization, and $P_i \in \mathbb{R}^{L_p \times d}$ contains the last L_p valid unpooled hidden states at the same layer, where L_p is the fixed prefix length. These final input states provide compact latent support from the context immediately preceding candidate generation. The schema records the reusable decision pattern, while δ_i keeps the concrete compact decision instance that succeeded. Appendix D details schema extraction rules and memory statistics.

Consensus Gate. CGD processes the current input x_t up to the readout layer ℓ_c, obtaining hidden states $h_{t,1:n_t}^{(\ell_c)}$, and forms the retrieval key k_t using the same pooling and normalization rule as k_i.

Using the retrieval size K, it retrieves a reference set $\mathcal{R}_t$ from Prefix Memory using k_t. CGD then computes the majority schema in the retrieved reference set:

$$\mu_t = \mathrm{Maj}(\{\sigma_i : i \in \mathcal{R}_t\}) . \tag{6}$$

where $\mathrm{Maj}(\cdot)$ returns a schema only when that schema has strict-majority support among the retrieved references. The consensus gate succeeds when the reference set returns a non-empty majority schema. When the consensus gate fails, CGD produces no candidate, and the agent falls back to the base agent's full realization path. Otherwise, the shared schema is denoted by $\hat{\sigma}_t$.

Latent Prefix Instantiation. The consensus schema specifies the reusable decision pattern, and CGD performs decoder-side instantiation under the current context and action space. CGD uses the retrieved references assigned to $\hat{\sigma}_t$ within the reference set to construct the latent prefix, following prefix conditioning for generation:

$$\mathcal{R}_t^+ = \{i \in \mathcal{R}_t : \sigma_i = \hat{\sigma}_t\}, \qquad \hat{P}_t = \frac{1}{|\mathcal{R}_t^+|} \sum_{i \in \mathcal{R}_t^+} P_i, \tag{7}$$

The aggregated prefix $\hat{P}_t$ retains the fixed prefix length L_p and provides the latent context used by the remaining decoder layers during instantiation. CGD then performs constrained decoder instantiation of the consensus schema under Γ_t:

$$u_t = f_{\ell_c+1:L}\Big([\hat{P}_t; h_{t,1:n_t}^{(\ell_c)}]\Big) , \qquad \hat{\delta}_t = \phi_{\Gamma_t}(x_t, \hat{\sigma}_t, u_t) . \tag{8}$$

where $f_{\ell_c+1:L}$ denotes the remaining decoder layers after the CGD readout layer, and $[\hat{P}_t; h_{t,1:n_t}^{(\ell_c)}]$ denotes layer-ℓ_c concatenation. Here ϕ_{Γ_t} denotes a constrained decoder readout over the current action space Γ_t. It uses the decoder state u_t to select argument tokens and serialize the consensus schema $\hat{\sigma}_t$ into the compact decision candidate $\hat{\delta}_t$. The action space provides the validity constraint over candidate serializations, while the argument selection is driven by the prefix-conditioned decoder state. The candidate is then passed to AGC for the candidate support check.

4.2 Attention-Guided Commitment

CGD proposes a compact decision candidate from consensus among retrieved reference steps. Attention-Guided Commitment (AGC) implements the candidate support stage. Starting from the latent prefix state, AGC scores the candidate token span and computes two signals: token confidence and span concentration gain. MIRA uses the candidate as the step output when both signals support it for the current state; otherwise, the agent falls back to the base agent's full realization path.

Candidate Scoring. AGC scores a proposed compact decision candidate before generating the full realization. For a candidate $\hat{\delta}_t$, we serialize it under Γ_t as the candidate token span $\mathcal{B}_t = (b_1, \ldots, b_m)$. Starting from the latent prefix state produced by CGD, AGC creates a temporary suffix branch $\mathcal{C}_t^+$ and evaluates the likelihood of this span:

$$\log p_\theta(\mathcal{B}_t \mid x_t, \hat{P}_t) = \sum_{j=1}^{m} \log p_\theta(b_j \mid x_t, \hat{P}_t, b_{<j}; \mathcal{C}_t^+). \tag{9}$$

where $\mathcal{C}_t^+$ denotes the temporary cache used for scoring $\mathcal{B}_t$. The branch provides the logits and attention statistics used to compute the support signals for the subsequent commitment rule.

Commitment Rule. AGC evaluates whether the candidate $\hat{\delta}_t$ should be used as the step output through two support signals on its token span $\mathcal{B}_t$. The first signal is token confidence, which measures output-level support from the decoder distribution. Following confidence-based adaptive generation and calibration analyses [23] [35], token confidence over $\mathcal{B}_t$ is defined as the average probability assigned to its tokens:

$$C_t = \frac{1}{m} \sum_{j=1}^{m} p_\theta(b_j \mid x_t, \hat{P}_t, b_{<j}; \mathcal{C}_t^+). \tag{10}$$

Averaging over tokens makes C_t comparable across candidate token spans of different lengths. The second signal is span concentration gain G_t, which measures span-level support from attention concentration. We compute G_t over the token span corresponding to $\mathcal{B}_t$, using decoder layers after ℓ_c, as defined in Eq. 4. The commitment rule selects between the two output forms in Eq. 1:

$$y_t^{\mathrm{MIRA}} = \begin{cases} \hat{\delta}_t, & C_t \geq \tau_c \text{ and } G_t \geq \tau_G, \\ \rho_t, & \text{otherwise.} \end{cases} \tag{11}$$

where τ_c and τ_G are the thresholds for token confidence and span concentration gain. The first branch emits the candidate $\hat{\delta}_t$ as the step output. The second branch discards the temporary suffix branch and falls back to the base agent's full realization path, producing ρ_t. Appendix E describes the cache-level implementation.

5 Experiments

5.1 Experimental Setup

Benchmarks and Data Splits. We evaluate MIRA on three long-horizon interactive benchmarks: ALFWorld [26], ScienceWorld [31], and WebShop [36]. These environments cover household interaction, scientific experiment tasks, and web shopping. ALFWorld and ScienceWorld are reported on seen and unseen evaluation sets. WebShop is evaluated on its test set. Additional data split and preprocessing details are provided in Appendix A.

Baselines. The main comparison uses base agents from three representative formats: direct-planning agents, code-based agents, and skill-based agents. ReAct emits textual actions in an observation and action loop [37]. CodeAct [32] and ReCode [38] realize steps through executable code, with ReCode further supporting recursive plan and action expansion. SkillNet represents skill-level procedure agents [17]. For each base agent, we evaluate the Vanilla version and the version equipped with MIRA under the same backbone, environment interface, and action space. We further compare with two plug-in methods built on ReAct to position MIRA against prior methods for inference efficiency and agent performance: AutoTool [9] improves efficiency through tool use optimization, while ExpeL [39] improves performance through learning from experience.

Evaluation Protocol. We report task score at the episode level. For ALFWorld, the score is binary task success. ALFWorld scores are macro-averaged across task families within each split. For ScienceWorld and WebShop, the score follows the official benchmark reward protocol. Tables use *Score* as a compact label for this task score. The efficiency metric is average time per episode. We also report input tokens, output tokens, amortized ratio, and false accept rate. Amortized ratio is the fraction of correct MIRA step decisions that are produced through the compact decision path. False accept rate measures how often MIRA incorrectly uses a compact decision candidate for a step that should have fallen back to full realization path.

Implementation Details. MIRA keeps each base agent frozen and uses no parameter updates. The main experiments use Qwen3-8B as the backbone [21], run with the Hugging Face Transformers implementation on 8*NVIDIA A800 80GB GPUs; Prefix Memory is constructed from successful training trajectories of the same base agent, with evaluation trajectories excluded from memory construction. The token confidence threshold is selected once per benchmark on the held-out calibration subset of the training episodes to account for differences in action spaces and candidate serializations. The selected values are $\tau_c = 0.45$ for ALFWorld, $\tau_c = 0.35$ for ScienceWorld, and $\tau_c = 0.40$ for WebShop. The remaining settings are fixed across benchmarks and agent formats: $\ell_c = 20$, $\tau_G = 0.06$, $K = 4$, and $L_p = 4$. All reported results are averaged over three runs.

5.2 Experimental Results

Main Comparison. Table 1 compares the Vanilla and MIRA variants of each base agent under the same backbone, environment interface, and action space. Across four base agents and five evaluation splits, MIRA improves task score by an average of 3.0 points and reduces aggregate inference time by 24.7%. The time reduction is substantial across settings, ranging from 16.5% to 32.8%. This

Table 1: Main results of Vanilla agents and MIRA on Qwen3-8B. ALF and Sci denote ALFWorld and ScienceWorld. Each split reports task score and average inference time.

Method	Variant	ALF-seen		ALF-unseen		Sci-seen		Sci-unseen		WebShop	
		Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
ReAct	Vanilla	60.5	122.2s	64.5	129.8s	23.5	36.0s	18.6	38.3s	35.6	31.5s
	MIRA	63.9	84.7s	68.0	87.2s	27.9	28.9s	22.1	31.0s	38.7	26.3s
CodeAct	Vanilla	82.9	25.9s	82.1	26.8s	46.4	44.5s	40.8	35.1s	36.5	41.7s
	MIRA	85.0	19.7s	84.6	20.0s	49.6	33.0s	43.5	27.1s	39.1	31.2s
ReCode	Vanilla	81.7	30.4s	84.3	29.8s	48.1	29.3s	39.6	29.7s	45.8	37.5s
	MIRA	84.1	24.6s	87.0	23.6s	51.2	23.2s	42.7	24.0s	48.6	31.1s
SkillNet	Vanilla	74.7	72.1s	79.4	78.1s	32.8	76.1s	28.9	81.8s	44.7	48.9s
	MIRA	77.1	52.8s	82.9	55.6s	36.1	59.4s	31.5	64.2s	47.3	39.4s

Table 2: Comparison with methods built on ReAct using Qwen3-8B. Input and Output denote average input and output tokens per episode.

Method	ALFWorld				ScienceWorld				WebShop			
	Score	Time	Input	Output	Score	Time	Input	Output	Score	Time	Input	Output
ReAct	62.5	126.0s	37973	5994	21.1	37.2s	14537	1648	35.6	31.5s	16524	371
ReAct w/o Thought	47.8	80.6s	28137	846	12.6	26.8s	10691	413	25.9	27.4s	13608	229
ReAct+ExpeL	**67.3**	132.1s	56857	5653	24.7	47.4s	19942	1596	**39.1**	35.0s	21462	360
ReAct+AutoTool	64.8	102.9s	**27680**	2945	20.3	30.8s	**11234**	1147	35.0	27.0s	**12402**	277
ReAct+MIRA	66.0	**86.0s**	29846	**2655**	**25.0**	**29.9s**	11764	**846**	38.7	**26.3s**	13617	**218**

pattern matches the goal of realization amortization: MIRA keeps the backbone frozen and preserves the external environment interface and retains full realization as the fallback, and reduces the full realization of steps whose compact decision is locally recoverable and supported for the current state. The task score gains mainly come from reducing errors introduced during full realization, such as formatting mistakes, repeated continuations, or drift from the selected executable content, while unresolved steps still follow the base agent's full realization path. The time reduction tracks the length of native realizations. Across benchmarks, the reduction is largest on ALFWorld and smallest on WebShop, where native actions are more compact. Skill-based agents still show strong reductions, especially when skill invocations accompany longer procedure descriptions. Appendix B reports transfer results across backbones with different parameter scales.

Comparison with agent enhancement methods. Table 2 compares MIRA with ReAct enhancement and compression variants. ReAct w/o Thought reduces output length by removing long thinking, but this direct compression substantially lowers task score. ExpeL uses experience memory to improve task score, reaching the highest scores on ALFWorld and WebShop, but increases average inference time from 64.9s to 71.5s and input tokens from 23.0k to 32.8k. AutoTool improves efficiency through tool selection and tool use optimization, reducing average time to 53.6s and input tokens to 17.1k, while its score drops below ReAct on ScienceWorld and WebShop. MIRA uses memory at the model-internal level: it retrieves successful reference steps in middle layer representation space and reduces the full realization of supported compact decisions. Relative to ReAct, MIRA raises average score from 39.7 to 43.2, reduces average time from 64.9s to 47.4s, reduces input tokens from 23.0k to 18.4k, and reduces output tokens from 2.67k to 1.24k. This comparison shows that MIRA improves efficiency through both input-side history reduction and output-side realization amortization, while preserving more stable task score than AutoTool.

Efficiency Decomposition. Table 3 reports input-side changes, output-side savings, amortized ratio, and false accept rate. Across benchmarks and agent formats, MIRA reduces input tokens by 9.2% on average because compact decisions shorten ReAct history and thinking traces while reducing error-induced context growth in ReCode and SkillNet. This input-side reduction is accompanied by output-side savings: average output tokens decrease from 1.88k to 0.93k, with a mean per-setting output reduction of 47.5%. The amortized ratio averages 0.53, while the false accept rate remains 0.05,

Table 3: Efficiency decomposition across benchmarks and baselines. Amortized denotes the amortized ratio, and False denotes the false accept rate following the definitions in Section 5.1.

Method	Score		Time		Input		Output		Amortized		False	
	Vanilla	MIRA	Vanilla	MIRA	Vanilla	MIRA	Vanilla	MIRA	Vanilla	MIRA	Vanilla	MIRA
ALFWorld												
ReAct	62.5	66.0	126.0	86.0	37973	29846	5994	2655	/	0.61	/	0.04
ReCode	83.0	85.5	30.1	24.1	8160	7956	1226	670	/	0.63	/	0.04
SkillNet	77.1	80.0	75.1	54.2	32957	31968	1598	757	/	0.66	/	0.03
ScienceWorld												
ReAct	21.1	25.0	37.2	29.9	14537	11764	1648	846	/	0.50	/	0.07
ReCode	43.9	47.0	29.5	23.6	8119	7808	1201	612	/	0.52	/	0.06
SkillNet	30.9	33.8	79.0	61.8	51706	49482	2204	1085	/	0.54	/	0.06
WebShop												
ReAct	35.6	38.7	31.5	26.3	16524	13617	371	218	/	0.43	/	0.04
ReCode	45.8	48.6	37.5	31.1	14334	13876	1426	855	/	0.41	/	0.05
SkillNet	44.7	47.3	48.9	39.4	19654	18942	1246	696	/	0.44	/	0.04

Table 4: Component ablation across representative baselines on ALFWorld. Top-1 schema + AGC replaces CGD consensus with the nearest schema. CGD + commit all removes AGC support checking. Confidence only and concentration only keep one AGC signal. Amort. denotes amortized ratio, and False denotes false accept rate.

Variant	ReAct					ReCode					SkillNet				
	Score	Time	Out	Amort.	False	Score	Time	Out	Amort.	False	Score	Time	Out	Amort.	False
Full MIRA	66.0	86.0	2655	0.61	0.04	85.5	24.1	670	0.63	0.04	80.0	54.2	757	0.66	0.03
Vanilla	62.5	126.0	5994	/	/	83.0	30.1	1226	/	/	77.1	75.1	1598	/	/
Top-1 schema + AGC	63.7	83.2	2485	0.66	0.11	83.8	23.4	626	0.68	0.08	77.6	52.3	685	0.71	0.11
CGD + commit all	60.8	77.0	2252	0.82	0.17	81.2	22.7	522	0.78	0.20	76.6	49.6	514	0.84	0.15
CGD + confidence only	64.5	80.0	2312	0.73	0.09	85.1	24.0	603	0.69	0.11	79.0	50.5	604	0.76	0.08
CGD + concentration only	63.8	79.3	2380	0.72	0.06	83.0	23.4	638	0.68	0.07	77.0	49.1	650	0.74	0.05

showing that MIRA uses the compact decision path for a substantial fraction of local steps without frequently accepting unsupported candidates. These results support the model-internal mechanism: MIRA reduces the cost of full realization for supported compact decisions, while unresolved steps remain on the base agent's full realization path.

Component Ablation. Table 4 isolates candidate proposal and candidate support in MIRA. Full MIRA gives the best balance between task score and time. Replacing CGD with top-1 schema retrieval weakens task score, showing that candidate proposal benefits from consensus over retrieved reference steps rather than a single nearest schema. Accepting all CGD candidates reduces cost more aggressively but increases false accepts, indicating that candidate proposal alone is insufficient. The two single-signal variants further show that token confidence and span concentration provide complementary support: token confidence measures output-level support from the decoder distribution, while span concentration measures whether support over the candidate span has settled. Additional ablation results are provided in Appendix C. These results show that MIRA depends on both consensus-based proposal and decoder-based support checking.

Threshold Sensitivity. Figure 3 studies the two commitment thresholds on ALFWorld. Lower thresholds accept more candidates and increase the amortized ratio, but overly permissive commitment lowers task score by admitting unsupported candidates. As τ_G and τ_c increase, the score curves quickly enter a stable region, while the amortized ratio continues to decrease. The selected values, $\tau_G = 0.06$ and $\tau_c = 0.45$, lie at the left side of this stable region. This choice preserves more compact decision steps while avoiding a higher false accept regime.

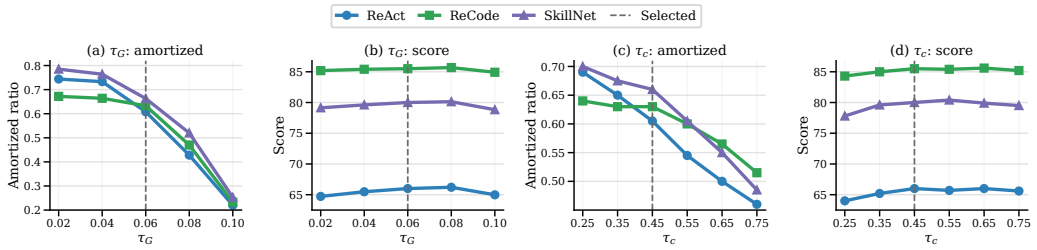


Figure 3: Sensitivity analysis on ALFWorld. Panels (a,b) vary the span concentration threshold τ_G, and panels (c,d) vary the token confidence threshold τ_c.

6 Conclusion

We study redundant realization in long-horizon LLM agents and formulate efficient inference as realization amortization between compact decisions and full realizations. MIRA implements this idea with two training-free stages: CGD proposes compact decision candidates from middle-layer reference consensus, and AGC uses token confidence and span concentration to decide whether a candidate is supported as the current step output. Across interactive benchmarks and agent formats, MIRA reduces inference cost while improving task score, with the base agent frozen and the original environment interface and action space retained. These results show that agents can skip substantial redundant generation when the decoder state already supports a compact decision.

Acknowledgements

This work is supported by National Natural Science Foundation of China No. 62406057, the Noncommunicable Chronic Diseases-National Science and Technology Major Project No. 2023ZD0501800, the Fundamental Research Funds for the Central Universities No. ZYGX2025XJ042, and the Sichuan Science and Technology Program under Grant No. 2024ZDZX0011. The authors declare no competing interests.

References

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.

[2] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In *ICML*, pages 2206–2240, Baltimore, Maryland, USA, 2022.

[3] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does BERT look at? an analysis of BERT's attention. In *ACL Workshop BlackboxNLP*, pages 276–286, Florence, Italy, 2019.

[4] Alexis Conneau, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In *ACL*, pages 2126–2136, Melbourne, Australia, 2018.

[5] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *NeurIPS*, New Orleans, LA, USA, 2023.

[6] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: program-aided language models. In *ICML*, pages 10764–10799, Honolulu, Hawaii, USA, 2023.

[7] Samuel Gershman and Noah D. Goodman. Amortized inference in probabilistic reasoning. In *CogSci*, Quebec City, Canada, 2014.

[8] Alex Graves. Adaptive computation time for recurrent neural networks. *CoRR*, abs/1603.08983, 2016.

[9] Jingyi Jia and Qinbin Li. AutoTool: Efficient tool selection for large language model agents. In *AAAI*, pages 31265–31273, Singapore, 2026.

[10] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *EMNLP*, pages 13358–13376, Singapore, 2023.

[11] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *ICLR*, Addis Ababa, Ethiopia, 2020.

[12] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *ICLR*, Banff, AB, Canada, 2014.

[13] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *ACL*, pages 881–905, Bangkok, Thailand, 2024.

[14] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NeurIPS*, Virtual Event, 2020.

[15] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *EMNLP*, pages 3102–3116, Singapore, 2023.

[16] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL/IJCNLP*, pages 4582–4597, Virtual Event, 2021.

[17] Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Xin Xu, Tongtong Wu, Kun Wang, Yang Liu, Zhen Bi, Jungang Lou, Yuchen Eleanor Jiang, Hangcheng Zhu, Gang Yu, Haiwen Hong, Longtao Huang, Hui Xue, Chenxi Wang, Yijun Wang, Zifei Shan, Xi Chen, Zhaopeng Tu, Feiyu Xiong, Xin Xie, Peng Zhang, Zhengke Gui, Lei Liang, Jun Zhou, Chiyu Wu, Jin Shang, Yu Gong, Junyu Lin, Changliang Xu, Hongjie Deng, Wen Zhang, Keyan Ding, Qiang Zhang, Fei Huang, Ningyu Zhang, Jeff Z. Pan, Guilin Qi, Haofen Wang, and Huajun Chen. SkillNet: Create, evaluate, and connect AI skills. *CoRR*, abs/2603.04448, 2026.

[18] Llama Team. The Llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

[19] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *NeurIPS*, Vancouver, BC, Canada, 2024.

[20] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *ICLR*, Vienna, Austria, 2024.

[21] Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025.

[22] 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. In *NeurIPS*, New Orleans, LA, USA, 2023.

[23] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *NeurIPS*, New Orleans, LA, USA, 2022.

[24] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(4):623–656, 1948.

[25] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, New Orleans, LA, USA, 2023.

[26] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *ICLR*, Virtual Event, Austria, 2021.

[27] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *ICPR*, pages 2464–2469, Cancún, Mexico, 2016.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, Long Beach, CA, USA, 2017.

[29] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *ACL*, pages 5797–5808, Florence, Italy, 2019.

[30] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

[31] Ruoyao Wang, Peter A. Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *EMNLP*, pages 11279–11298, Abu Dhabi, United Arab Emirates, 2022.

[32] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *ICML*, pages 50208–50232, Vienna, Austria, 2024.

[33] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, New Orleans, LA, USA, 2022.

[34] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *ACL*, pages 2246–2251, Online, 2020.

[35] Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Zheng Lin, Li Cao, and Weiping Wang. Dynamic early exit in reasoning models. *CoRR*, abs/2504.15895, 2025.

[36] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *NeurIPS*, New Orleans, LA, USA, 2022.

[37] 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 *ICLR*, Kigali, Rwanda, 2023.

[38] Zhaoyang Yu, Jiayi Zhang, Huixue Su, Yufan Zhao, Yifan Wu, Mingyi Deng, Jinyu Xiang, Yizhang Lin, Lingxiao Tang, Yingchao Li, Yuyu Luo, Bang Liu, and Chenglin Wu. ReCode: Unify plan and action for universal granularity control. *CoRR*, abs/2510.23564, 2025.

[39] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *AAAI*, pages 19632–19642, Vancouver, Canada, 2024.

[40] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *ICLR*, Vienna, Austria, 2024.

A Additional Experimental Details

ALFWorld is a text-only household environment with binary task success. It contains 3553 training episodes, 140 seen evaluation episodes, and 134 unseen evaluation episodes. ScienceWorld is a text-based science environment with dense rewards over task progress. It contains 1483 training episodes, 194 seen evaluation episodes, and 211 unseen evaluation episodes. WebShop is an online shopping environment with dense product-matching rewards, where success corresponds to a final reward of 1. It contains 1824 training episodes and 200 test episodes.

Table 5 lists the hyperparameters used in the main experiments. The token-confidence threshold τ_c is selected once per benchmark on a held-out calibration subset split from the training episodes and then fixed across agent formats and evaluation splits of that benchmark. This selection accounts for differences in action spaces, candidate serializations, and benchmark-specific token distributions. The span-concentration threshold τ_G is shared across benchmarks because it is defined as a normalized entropy decrease over the candidate span. The remaining hyperparameters, ℓ_c, K, and L_p, control the backbone readout layer, retrieval consensus, and latent-prefix support, and are shared across benchmarks and agent formats.

Table 5: Hyperparameters used in the main experiments.

Symbol	Value	Stage	Description
ℓ_c	20	CGD/AGC	Readout layer used for retrieval, latent prefix construction, and the AGC scoring branch.
K	4	CGD	Fixed number of retrieved reference steps used for the CGD majority vote.
L_p	4	CGD	Prefix length; it specifies how many tail valid unpooled hidden states at ℓ_c are stored as latent prefix support for memory item.
τ_c	0.45 / 0.35 / 0.40	AGC	Token confidence threshold for using a compact decision candidate as the step output, selected for ALFWorld / ScienceWorld / WebShop, respectively.
τ_G	0.06	AGC	Span concentration gain threshold for using a compact decision candidate as the step output.

For fair comparison, the Vanilla and MIRA variants of each base agent use the same backbone, environment interface, action space, and prompt examples. MIRA keeps the base agent frozen and introduces no parameter updates. Prefix Memory is built only from successful trajectories in the memory-construction portion of the training episodes, with calibration and evaluation trajectories excluded from memory construction. We record inference time, input tokens, and output tokens from the same execution logs. The recorded episode time includes retrieval, candidate instantiation, AGC scoring, parser and action-space checks, fallback decoding, and environment interaction. A step output is valid if it passes the parser, action-space, and environment feedback checks used to validate the base agent's step output. Amortized ratio measures the fraction of valid MIRA step outputs that are produced through the compact decision path. All reported runs are executed with Hugging Face Transformers implementation on an $8\times$A800 server.

B Additional Backbone Transfer Results

B.1 Backbone Transfer Across Model Scales

Table 6 evaluates whether MIRA transfers across backbone scales on ReAct under the same evaluation protocol as the main experiments. The backbones include Qwen3 models from 4B to 32B [21], together with Llama 3.1-8B [18]. The Vanilla scores generally increase with model scale, giving a range of base-agent strengths for transfer evaluation. MIRA remains effective when the backbone can both recover compact decisions and instantiate supported candidates under the current state and action space. On Qwen3-4B, the gains are limited and sometimes mixed. This smaller model can still form schema consensus, but its weaker step-level reasoning and argument grounding make the instantiated compact decision candidates less reliably useful. In such cases, realization amortization can reduce redundant generation only when the underlying step decision is already correct; it cannot compensate for insufficient base task ability. Among stronger backbones, MIRA reduces inference

time more when the native full realization is verbose, such as repeated action formatting, short code stubs, and routine skill invocations.

Table 6: Backbone transfer results on ReAct across model scales. ALF and Sci denote ALFWorld and ScienceWorld.

Model	Variant	ALF-seen		ALF-unseen		Sci-seen		Sci-unseen		WebShop	
		Score	Time	Score	Time	Score	Time	Score	Time	Score	Time
Qwen3-32B	Vanilla	72.8	246.8s	75.1	274.2s	34.7	66.5s	29.3	65.1s	44.8	50.3s
	MIRA	77.1	136.7s	79.3	150.0s	39.5	41.1s	33.8	42.0s	48.9	39.4s
Qwen3-14B	Vanilla	67.4	162.1s	70.0	174.4s	29.1	44.9s	24.0	45.8s	40.3	35.9s
	MIRA	70.7	97.3s	74.3	104.3s	33.6	31.8s	28.0	32.6s	42.8	31.1s
Qwen3-8B	Vanilla	60.5	122.2s	64.5	129.8s	23.5	36.0s	18.6	38.3s	35.6	31.5s
	MIRA	63.9	84.7s	68.0	87.2s	27.9	28.9s	22.1	31.0s	38.7	26.3s
Qwen3-4B	Vanilla	53.2	89.9s	56.0	94.2s	18.4	27.4s	14.7	29.8s	28.6	27.1s
	MIRA	54.0	63.1s	56.2	68.5s	18.1	21.9s	15.2	24.0s	28.2	26.1s
Llama 3.1-8B	Vanilla	54.6	131.5s	57.8	142.0s	19.6	35.5s	15.8	37.8s	30.0	34.4s
	MIRA	55.8	80.3s	60.1	90.1s	21.7	26.9s	17.0	30.1s	31.3	29.1s

B.2 Readout Layer Selection Across Qwen Backbones

Figure 4 studies readout layer selection on ALFWorld for Qwen3 backbones. The Qwen3-8B result anchors the main setting at $\ell_c = 20$. As backbone size increases, the layer with low compact-decision Consensus entropy and clear decision-realization separation shifts later in the decoder layers, giving $\ell_c = 20$ for Qwen3-4B, $\ell_c = 24$ for Qwen3-14B, and $\ell_c = 40$ for Qwen3-32B. Qwen3-4B also shows a clear separation between the compact decision and full realization curves, but its lower base task ability makes candidate instantiation less reliable, limiting the practical gain from realization amortization on this smaller backbone.

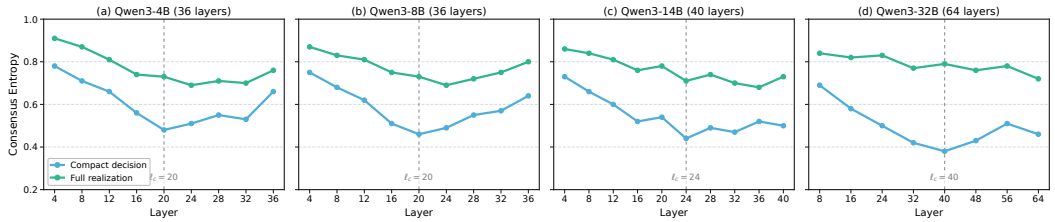


Figure 4: Readout layer analysis across Qwen3 backbones on ALFWorld. Each panel reports Consensus entropy under the compact decision labeling and the full realization labeling. Vertical dashed lines mark the selected readout layers used by MIRA.

C Additional Ablation Results

We provide additional ablations on ALFWorld across three representative base agents. The main text isolates the two stages of MIRA, candidate proposal and candidate support. This section further separates nearest-reference reuse from consensus-based proposal, examines paired recovery order, and studies how prefix length affects realization amortization.

C.1 Candidate Proposal Controls

Table 7 studies whether compact candidates can be obtained by prompt-level compression alone, or require the proposal components of CGD. The controls separate compact-only prompting, nearest compact decision copy, schema consensus, and latent prefix support. Compact-only prompting keeps the base agent format fixed but asks the model to directly emit the shortest executable decision. Nearest compact decision copy directly copies the compact decision from the nearest retrieved

reference step. Memory-free compact + AGC removes Prefix Memory and schema consensus, directly decodes a compact candidate under the current action space, and applies the same AGC support check. The variant without latent prefix removes $\hat{P}_t$ while keeping the consensus schema and AGC support check. The shuffled-prefix variant keeps the same consensus schema but uses prefixes from mismatched reference steps, testing whether latent prefix support must align with the retrieved schema.

Table 7: Candidate proposal controls on ALFWorld with Qwen3-8B. All MIRA variants keep AGC unless otherwise specified; Compact-only prompting is a prompt-level control without MIRA modules, and Memory-free compact + AGC removes Prefix Memory while retaining AGC. Nearest compact decision copy tests direct nearest-reference reuse. w/o latent prefix removes $\hat{P}_t$. Shuffled prefix uses prefixes from mismatched reference steps.

Variant	ReAct					ReCode					SkillNet				
	Score	Time	Out	Amort.	False	Score	Time	Out	Amort.	False	Score	Time	Out	Amort.	False
Vanilla	62.5	126.0	5994	/	/	83.0	30.1	1226	/	/	77.1	75.1	1598	/	/
Compact-only prompting	47.8	80.6	846	/	/	65.8	22.9	526	/	/	59.6	52.8	684	/	/
Memory-free compact + AGC	61.4	116.4	4457	0.27	0.12	81.7	26.3	934	0.25	0.15	74.6	68.4	1238	0.24	0.13
Nearest compact decision copy	59.8	78.9	2613	0.70	0.18	80.9	23.7	599	0.71	0.17	74.8	50.0	647	0.76	0.16
w/o latent prefix	63.8	119.1	4648	0.24	0.10	83.9	27.3	970	0.28	0.08	77.9	65.2	1118	0.31	0.09
Shuffled prefix	62.4	114.6	4276	0.30	0.15	81.2	27.7	997	0.26	0.14	77.3	66.3	1157	0.28	0.15
Full MIRA	66.0	86.0	2655	0.61	0.04	85.5	24.1	670	0.63	0.04	80.0	54.2	757	0.66	0.03

These controls test whether candidate proposal requires more than prompt-level compression or nearest-reference reuse. Compact-only prompting reduces output length but sharply lowers task score, showing that simply asking the base agent to emit a short decision is not sufficient. Memory-free compact + AGC recovers part of this loss but remains below Full MIRA, indicating that AGC can filter unsupported candidates but cannot replace memory-based proposal. Directly copying the nearest compact decision lowers task score and increases false accepts, while removing or shuffling the latent prefix reduces the quality of candidate instantiation. Such a pattern supports the role of CGD: retrieved reference steps provide schema consensus and latent support, while the compact decision candidate is instantiated by the decoder under the current state and action space.

C.2 Span Concentration Controls

Table 8 examines whether span concentration gain provides candidate-support evidence beyond candidate length, schema frequency, and token confidence. We use the same ALFWorld setting as the main ablations and pool candidate cases from ReAct, ReCode, and SkillNet under Qwen3-8B. For each supported candidate, we construct hard rejected variants by preserving the decision schema and keeping the candidate token span close in length, while omitting, swapping, or mismatching one argument. These variants form unsupported cases with the same surface form but incorrect argument grounding. After filtering for these requirements, the retained unsupported candidates and supported candidates have an approximate ratio of 1:2. We then evaluate whether supported and unsupported candidates can be separated by token confidence C_t, span concentration gain G_t, or their combination. For the combined row, C_t and G_t are normalized on the validation pool and summed as a ranking score. False@60 is the fraction of unsupported candidates among the top 60% ranked candidates. Span concentration gain remains informative after matching token length, decision schema, schema frequency, or token confidence, although its separation becomes weaker under these stricter controls. Combining C_t and G_t gives the strongest supported-vs-unsupported separation and the lowest false-accept rate. These results indicate that G_t provides complementary span-level evidence to token confidence in AGC.

C.3 Paired Recovery Order

We supplement Observation 1 with a paired recovery-order diagnostic on ReAct in ALFWorld using frozen Qwen3-8B representations. The compact-decision label records the executable action, whereas the full-realization label records the corresponding normalized ReAct step output. Both labelings use the same pre-output input and masked mean pooling to obtain frozen layerwise representations. We

Table 8: Controls for span concentration gain on ALFWorld with Qwen3-8B. AUROC and AUPRC measure supported-vs-unsupported candidate separation.

Signal / Control	AUROC	AUPRC	False@60
Token confidence C_t only	0.84	0.87	0.12
Span concentration gain G_t only	0.90	0.91	0.10
Combined C_t, G_t	**0.94**	**0.95**	**0.08**
G_t, length matched	0.88	0.89	0.16
G_t, schema matched	0.86	0.88	0.17
G_t, frequency balanced	0.87	0.86	0.16
G_t, confidence matched	0.83	0.85	0.15

evaluate recovery across consecutive decoder layers and define the recovery depth as the last layer of the first four consecutive layers with correct predictions.

We evaluate reference-supported steps from successful ReAct trajectories in ALFWorld. Historical diagnostic references include benchmark validation trajectories, are task-disjoint from queries, and are separate from deployment Prefix Memory. We use linear probes with the same regularization setting for both labelings, evaluating $C = 1$ and $C = 10$, together with three operator-preserving control labelings matched to compact-label class counts and frequencies in the reference pool.

Table 9 reports paired recovery-depth differences with 97.5% trajectory-bootstrap confidence intervals. Comparisons use steps recovered under both labelings and all three controls within each setting. Compact decisions recover earlier than full realizations and matched controls under both regularization settings, supporting earlier compact-decision recovery on the paired subsets.

Table 9: Paired recovery order on ReAct in ALFWorld. Positive layer lead indicates earlier compact-decision recovery. Control denotes the mean recovery depth over three matched labelings. All confidence intervals are 97.5%.

	Recovered		Full − Decision		Control − Decision	
Readout	Steps	Traj.	Mean	CI	Mean	CI
Linear probe, $C = 1$	521	302	1.68	[1.21, 2.19]	3.05	[2.49, 3.65]
Linear probe, $C = 10$	549	304	1.45	[1.10, 1.82]	3.25	[2.78, 3.72]

C.4 Prefix Length

Table 10 studies the prefix hyperparameter that controls Prefix Memory usage. The prefix length L_p determines how many valid unpooled hidden states each memory item contributes as latent prefix support. For prefix length, $L_p = 0$ removes latent prefix support and sharply reduces useful amortization, confirming that schema consensus alone is not sufficient for reliable decoder-side instantiation. Increasing L_p to 4 improves score and reduces false accepts, while $L_p = 8$ brings slightly more amortization but also a higher false accept rate.

Table 10: Sensitivity to prefix length on ALFWorld with Qwen3-8B.

Variant	ReAct				ReCode				SkillNet			
	Score	Time	Amort.	False	Score	Time	Amort.	False	Score	Time	Amort.	False
Vanilla	62.5	126.0	/	/	83.0	30.1	/	/	77.1	75.1	/	/
$L_p = 0$	63.8	119.1	0.24	0.10	83.9	27.3	0.28	0.08	77.9	65.2	0.31	0.09
$L_p = 2$	64.7	93.6	0.46	0.07	84.4	25.9	0.50	0.06	78.8	59.7	0.51	0.06
$L_p = 4$	66.0	86.0	0.61	0.04	85.5	24.1	0.63	0.04	80.0	54.2	0.66	0.03
$L_p = 8$	65.2	84.8	0.66	0.08	84.7	23.8	0.67	0.07	79.0	53.2	0.70	0.07

D Schema Extraction and Statistics

This appendix details the offline constructions used by CGD. Prefix Memory is built from successful trajectories of the same base agent. Each successful step is stored with a retrieval key, a latent prefix, a decision schema, and a compact decision instance. The schema inventory is obtained by canonicalizing valid compact decisions into decision schemas that record reusable operators and argument roles under each benchmark action space. These constructions are deterministic post-processing steps over successful trajectories and do not introduce extra trainable components.

D.1 Prefix Memory Construction and Schema Agreement

We first parse each successful step into the smallest executable decision exposed by the agent, and then canonicalize that decision into a decision schema, i.e., a reusable operator with typed argument roles. The schema records the reusable decision pattern, while the compact decision instance keeps the concrete arguments that succeeded. At inference time, CGD first infers a consensus schema $\hat{\sigma}_t$ from Prefix Memory and then provides it to the decoder for constrained instantiation.

Prefix Memory Fields. For each successful step i, let x_i be the original agent input, δ_i the successful compact decision, and ρ_i the base-agent realization that produced it. We run x_i through the frozen backbone at the CGD readout layer ℓ_c and store:

$$\begin{aligned} &\sigma_i = \mathrm{Schema}(\delta_i), \qquad m_i = (k_i, P_i, \sigma_i, \delta_i), \\ &k_i = \mathrm{Norm}\Big(\mathrm{Pool}(H_i^{(\ell_c)})\Big), \qquad P_i = \mathrm{Tail}_{L_p}(H_i^{(\ell_c)}). \end{aligned} \tag{12}$$

where $H_i^{(\ell_c)}$ denotes the layer-ℓ_c hidden states of the reference input x_i. The key k_i indexes the layerwise neighborhood used for retrieval. The prefix $P_i \in \mathbb{R}^{L_p \times d}$ stores the last L_p valid unpooled hidden states at the readout layer, where L_p is the fixed prefix length. The operator $\mathrm{Tail}_{L_p}(\cdot)$ selects the final L_p valid input positions before step output generation, rather than generated tokens from the reference step. These positions are used because their hidden states have integrated the preceding prompt under causal attention and are adjacent in sequence to candidate generation. Thus P_i provides compact latent support for decoder-side instantiation, while the decision identity is represented by the schema σ_i and the concrete decision δ_i. The realization ρ_i is used only to extract and validate the successful decision, and is not replayed during inference.

Schema Agreement Rule. For a query step t, CGD processes the current input x_t up to the readout layer ℓ_c, obtains $H_t^{(\ell_c)}$, and forms a retrieval key k_t using the same pooling and normalization rule as k_i. It then retrieves a fixed-K reference set:

$$\mathcal{R}_t = \mathrm{TopK}_i^K \, \mathrm{sim}(k_t, k_i), \tag{13}$$

where the main experiments use $K = 4$. CGD computes the strict-majority schema in this retrieved reference set:

$$\mathrm{Maj}(\mathcal{R}_t) = \begin{cases} \sigma, & \frac{1}{|\mathcal{R}_t|} \sum_{i \in \mathcal{R}_t} \mathbf{1}[\sigma_i = \sigma] > \frac{1}{2}, \\ \perp, & \text{if no such schema exists.} \end{cases} \tag{14}$$

The consensus gate succeeds when $\mathrm{Maj}(\mathcal{R}_t) \neq \perp$. With $K = 4$, strict majority requires at least three retrieved references to share the same schema. When the consensus gate fails, CGD produces no candidate, and the agent falls back to the base agent's full realization path. Otherwise, the majority schema is denoted by $\hat{\sigma}_t$.

Latent Prefix Aggregation. After schema agreement, CGD aggregates latent prefixes only from retrieved references that support the consensus schema:

$$\mathcal{R}_t^+ = \{i \in \mathcal{R}_t : \sigma_i = \hat{\sigma}_t\}, \qquad \hat{P}_t = \frac{1}{|\mathcal{R}_t^+|} \sum_{i \in \mathcal{R}_t^+} P_i. \tag{15}$$

The aggregation is position-wise, so the aggregated prefix $\hat{P}_t$ retains shape $\mathbb{R}^{L_p \times d}$ rather than growing with the number of retrieved references. This keeps the latent prefix length fixed while allowing multiple successful references to contribute support for the same schema.

Decoder-Side Instantiation. At inference time, the consensus schema $\hat{\sigma}_t$ specifies the operator and argument roles of the compact decision candidate. The current action space Γ_t defines the valid serializations compatible with this schema, and the prefix-conditioned decoder state u_t provides the token-level evidence used to select concrete arguments. The constrained readout is:

$$u_t = f_{\ell_c+1:L}\Big([\hat{P}_t; H_t^{(\ell_c)}]\Big), \qquad \hat{\delta}_t = \phi_{\Gamma_t}(x_t, \hat{\sigma}_t, u_t). \tag{16}$$

We implement ϕ_{Γ_t} as schema-conditioned constrained greedy readout from the frozen decoder. The schema $\hat{\sigma}_t$ and the current action space Γ_t induce a schema-compatible serialization space $\mathcal{L}_{\Gamma_t}(\hat{\sigma}_t)$, whose grammar follows the candidate format of the corresponding agent type. This grammar provides admissible continuation masks and complete serialization boundaries, while token selection is scored by the prefix-conditioned decoder distribution.

We represent schema-compatible serializations as a prefix trie, whose valid continuations are induced by the schema grammar and the same constraints used by the base agent. Let $\mathcal{A}_{\Gamma_t}(b_{<j}, \hat{\sigma}_t)$ denote the trie-valid next-token set after the partial serialization $b_{<j}$. At token position j, the decoder distribution is evaluated under the temporary suffix branch $\mathcal{C}_t^+$, and the next token is selected by:

$$b_j = \arg \max_{v \in \mathcal{A}_{\Gamma_t}(b_{<j}, \hat{\sigma}_t)} p_\theta\Big(v \mid x_t, \hat{P}_t, b_{<j}; \mathcal{C}_t^+\Big). \tag{17}$$

Decoding stops when the trie reaches a terminal node corresponding to a complete serialization, and the compact decision candidate is obtained as:

$$\hat{\delta}_t = \mathrm{Detok}(b_{1:m}). \tag{18}$$

After $\hat{\delta}_t$ is decoded, MIRA applies the same parser and interface-validity checks as the base agent. If the trie has no valid continuation before reaching a complete serialization, or if the decoded candidate fails the validity check, instantiation aborts and the system falls back to the base realization path. Thus, Γ_t constrains admissible serialization forms, while concrete argument choices are determined by the prefix-conditioned decoder distribution. The resulting candidate is then passed to AGC, which evaluates its token confidence C_t and span concentration gain G_t before commitment.

D.2 Schema Statistics

Schemas record the reusable operator and argument roles of compact decisions in MIRA. Table 11 summarizes the size of the reusable decision space under the deterministic schema extraction rules.

ALFWorld. ALFWorld consists of household manipulation tasks in text-based indoor environments, where the agent navigates rooms, interacts with receptacles, transforms objects, and places them at target locations. Its compact decisions are naturally expressed as parameterized action forms over objects, receptacles, and tools. Example schemas include *go_to(location)*, *open(receptacle)*, *take_from(obj, receptacle)*, *clean_with(obj, tool)*, and *put_in_on(obj, receptacle)*.

ScienceWorld. ScienceWorld evaluates science-oriented interactive tasks, including state changes, conductivity tests, object classification, plant growth, chemistry mixing, and measurement. The action space is broader because the agent must inspect objects, manipulate containers, use devices, and operate task-specific instruments. Example schemas include *focus_on(obj)*, *look_at(obj)*, *take(obj)*, *pour(obj, target)*, *mix(obj, target)*, *connect_electrical(obj_1, obj_2)*, and *activate(obj)*.

WebShop. WebShop is an interactive shopping benchmark where the agent searches for products, opens product pages, selects required options, and submits a purchase. Although the environment exposes a small primitive interface, the reusable decisions separate search, product selection, option selection, navigation, and final purchase. Example schemas include *search(query)*, *click_product(item)*, *click_option(value)*, *click_detail(section)*, *click_nav(target)*, and *click_buy_now()*.

E KV-Cache Implementation

The KV-cache implementation avoids redundant decoding when MIRA falls back to the base agent's full realization path. MIRA reuses the prefill cache of the current input x_t, opens a temporary suffix

Table 11: Schema size across the three benchmarks.

Dataset	ALFWorld	ScienceWorld	WebShop
Number of schemas	7	24	7

branch for decoder-side candidate instantiation and AGC scoring, and discards this branch when the candidate is rejected. This lets the agent resume the base agent's full realization path without reprocessing the full prompt.

Staged Prefill. Let the transformer have L layers and let ℓ_c be the CGD readout layer. During prefill, MIRA runs the current input x_t through the lower layers once and keeps the layer-ℓ_c hidden states $H_t^{(\ell_c)}$. CGD forms the retrieval key from these hidden states and applies the consensus gate. If the consensus gate proposes no schema, the agent continues along the base agent's full realization path. If the consensus gate returns a schema $\hat{\sigma}_t$ and constructs the latent prefix $\hat{P}_t$, MIRA opens a temporary suffix branch:

$$\tilde{H}_t^{(\ell_c)} = [\hat{P}_t; H_t^{(\ell_c)}], \qquad \tilde{H}_t^{(l)} = f_l(\tilde{H}_t^{(l-1)}), \quad l = \ell_c + 1, \ldots, L. \tag{19}$$

Only the suffix layers after ℓ_c are evaluated in this branch. The lower-layer prefill of x_t is shared with the base path. The temporary branch provides the decoder states used for constrained decoder instantiation and for the AGC candidate support check.

Prefix KV Slots. The latent prefix is prepended as hidden-state positions in the temporary suffix branch, rather than inserted as text tokens or used to replace current hidden states. At each suffix layer, the prefix positions in $\tilde{H}_t^{(l-1)}$ are projected into key-value slots:

$$K_{t,l}^p = W_l^K \tilde{P}_t^{(l-1)}, \qquad V_{t,l}^p = W_l^V \tilde{P}_t^{(l-1)}, \qquad l > \ell_c, \tag{20}$$

where $\tilde{P}_t^{(l-1)}$ denotes the prefix part of the branch hidden states before layer l. The current input positions and later candidate token positions can attend to these prefix slots through the causal attention mask. Logits are read from the original input and candidate token positions.

Candidate Token Span Scoring. Using the temporary suffix branch, CGD performs constrained decoder readout to instantiate the consensus schema under the current action space. Given the resulting compact decision candidate $\hat{\delta}_t$, AGC scores the corresponding candidate token span $\mathcal{B}_t = (b_1, \ldots, b_m)$ on the same temporary branch:

$$\log p_\theta(\mathcal{B}_t \mid x_t, \hat{P}_t) = \sum_{j=1}^{m} \log p_\theta(b_j \mid x_t, \hat{P}_t, b_{<j}; \mathcal{C}_t^+), \tag{21}$$

where $\mathcal{C}_t^+$ is the temporary KV-cache branch. During this scoring pass, AGC stores the logits for the candidate tokens used to compute token confidence C_t, and stores the attention distributions at the span positions used to compute span concentration gain G_t. The scoring pass is limited to $\mathcal{B}_t$; full realization remains on the fallback path when AGC rejects the candidate.

Commit or Discard. If AGC accepts the candidate, MIRA emits $\hat{\delta}_t$ as the step output and commits the temporary branch to the compact decision path. If AGC rejects the candidate, the temporary branch is discarded:

$$\mathcal{C}_t = \begin{cases} \mathcal{C}_t^+, & y_t^{\mathrm{MIRA}} = \hat{\delta}_t, \\ \mathcal{C}_t^0, & y_t^{\mathrm{MIRA}} = \rho_t, \end{cases} \tag{22}$$

where $\mathcal{C}_t^+$ denotes the temporary cache branch used for the compact decision path, and $\mathcal{C}_t^0$ denotes the ordinary cache used by the base agent's full realization path. In the rejection case, the agent resumes the base agent's full realization path without committing to the candidate branch.

F Observation Protocols and Metrics

This appendix gives the complete metric construction used in Section 3. The observation study uses the same frozen backbone and the same agent input as the base agent. It compares two output forms, compact decisions and full realizations, through two diagnostic signals: Consensus entropy for local recoverability and span concentration gain for candidate support.

Output Labeling. The observation study follows the output-form definition in Eq. 1. For each successful reference step, we attach two labels derived from the same base-agent step output. The compact-decision label records the executable decision δ_i, such as the action, tool call, code stage, or skill invocation used at the step. The full realization label records the corresponding native step output ρ_i, such as the complete action text or code realization produced by the base agent. These two labels are assigned to the same frozen layerwise representation; only the label space changes between $\mathcal{Y}^{\mathrm{dec}}$ and $\mathcal{Y}^{\mathrm{real}}$. The observation study then asks whether compact-decision labels become locally recoverable before full realization labels, and whether a proposed compact decision candidate is supported as the current step output.

F.1 Local Recoverability and Consensus Entropy

Consensus entropy measures whether an output form is locally recoverable from a decoder layer. The measure follows a layerwise locality premise: when a layer representation carries an output form, successful reference steps that are close in that representation space should concentrate on the same label.

For the current step t, at decoder layer ℓ, we represent the input x_t by:

$$q_t^{(\ell)} = \mathrm{Norm}\Big(\mathrm{Pool}(h_{t,1:n_t}^{(\ell)})\Big), \tag{23}$$

where $h_{t,1:n_t}^{(\ell)}$ are the layer-ℓ hidden states of x_t. We use masked mean pooling over valid tokens to form a layerwise state summary, followed by ℓ_2 normalization for retrieval.

We use successful trajectories of the same base agent as reference steps, following the general use of case-based and nearest-neighbor memory in nonparametric reuse [1, 11]. At decoder layer ℓ, each reference step i provides a frozen representation $q_i^{(\ell)}$. The same reference step is assigned two labels: $y_i^{\mathcal{Y}^{\mathrm{dec}}}$ under the compact decision labeling and $y_i^{\mathcal{Y}^{\mathrm{real}}}$ under the full realization labeling. The representations are shared across the two analyses; only the label space changes.

For a query step t, retrieval is performed once at layer ℓ. We retrieve the top-K reference steps by cosine similarity:

$$\mathcal{R}_t^{(\ell)} = \mathrm{TopK}_i^K \,\mathrm{sim}\Big(q_t^{(\ell)}, q_i^{(\ell)}\Big). \tag{24}$$

Given a label space $\mathcal{Y}$, the retrieved reference set induces the label distribution:

$$p_t^{\ell,\mathcal{Y}}(y) = \frac{1}{K} \sum_{i \in \mathcal{R}_t^{(\ell)}} \mathbf{1}[y_i^{\mathcal{Y}} = y], \qquad y \in \mathcal{Y}. \tag{25}$$

where $y_i^{\mathcal{Y}}$ denotes the label of reference step i under label space $\mathcal{Y}$. Consensus entropy is then defined as:

$$H_{\mathrm{cons},t}^{\ell}(\mathcal{Y}) = -\frac{1}{\log \min(K, |\mathcal{Y}|)} \sum_{y \in \mathcal{Y}} p_t^{\ell,\mathcal{Y}}(y) \log p_t^{\ell,\mathcal{Y}}(y). \tag{26}$$

Terms with $p_t^{\ell,\mathcal{Y}}(y) = 0$ are treated as zero. The normalization by $\log \min(K, |\mathcal{Y}|)$ accounts for the finite support of the retrieved-label distribution and places the entropy on a comparable scale across global label spaces. We set the entropy to zero when $\min(K, |\mathcal{Y}|) = 1$. Low Consensus entropy indicates that the retrieved successful reference steps concentrate on the same label. We compute $H_{\mathrm{cons},t}^{\ell}(\mathcal{Y})$ under the compact decision labeling and the full realization labeling using the same query steps and the same retrieved reference sets. A lower value under $\mathcal{Y}^{\mathrm{dec}}$ indicates stronger local recoverability of compact decisions at that layer.

We also report recoverability as a complementary majority-label measure:

$$\mathrm{Rec}^{\ell}(\mathcal{Y}) = \frac{1}{|\mathcal{Q}|} \sum_{t \in \mathcal{Q}} \mathbf{1}\left[\arg\max_{y \in \mathcal{Y}} p_t^{\ell,\mathcal{Y}}(y) = y_t^{\mathcal{Y}} \right], \tag{27}$$

where $\mathcal{Q}$ is the query set and $y_t^{\mathcal{Y}}$ is the query label under label space $\mathcal{Y}$. Higher recoverability means that the majority label among retrieved references agrees more often with the query label.

F.2 Candidate Support and Span Concentration Gain

Local recoverability alone does not determine whether a proposed candidate should be used as the step output. We therefore evaluate candidate support from the decoder computation over the candidate token span.

Let $\mathcal{B}_t = (b_1, \ldots, b_m)$ denote the compact decision candidate serialized under the current action space Γ_t, and let $D_t = (j_1, \ldots, j_m)$ denote the corresponding token positions, where j_1 and j_m are the first and last positions of the span. For the current input x_t, head h, layer l, and position $j \in D_t$, we define the normalized Attention entropy as:

$$H_{\mathrm{att},t}^{h,l}(j) = -\frac{1}{\log |\Omega_{t,j}|} \sum_{r \in \Omega_{t,j}} a_{t,j}^{h,l}(r) \log a_{t,j}^{h,l}(r), \tag{28}$$

where $\Omega_{t,j}$ denotes the selected visible positions used for attention analysis at token j, and $a_{t,j}^{h,l}$ is the attention distribution renormalized over these positions. Since each attention row is a probability distribution over visible positions [28], this entropy measures how dispersed the attention distribution is. Lower Attention entropy indicates concentration on fewer visible positions.

We summarize Attention entropy at position j by averaging over decoder layers and attention heads:

$$\bar{H}_{\mathrm{att},t}(j) = \frac{1}{|\mathcal{H}|\,|\mathcal{L}|} \sum_{h \in \mathcal{H}} \sum_{l \in \mathcal{L}} H_{\mathrm{att},t}^{h,l}(j), \qquad j \in D_t, \tag{29}$$

where $\mathcal{L}$ is the set of decoder layers used for the analysis, and $\mathcal{H}$ is the set of heads within each layer. In MIRA, this layer set is taken from the suffix layers after the CGD readout layer.

The span concentration gain compares the averaged Attention entropy at the first and last positions of the candidate token span:

$$G_t = \frac{\bar{H}_{\mathrm{att},t}(j_1) - \bar{H}_{\mathrm{att},t}(j_m)}{\bar{H}_{\mathrm{att},t}(j_1) + \epsilon}. \tag{30}$$

A larger G_t means that Attention entropy decreases more strongly from the first position to the last position of the candidate token span. We use this quantity as span-level evidence for whether a compact decision candidate is supported as the current step output.

F.3 Observation Protocols

For Observation 1, we keep the frozen backbone and the original agent input for each query step. We extract $q_t^{(\ell)}$ at each decoder layer and retrieve successful reference steps using the same layerwise representation space. If the query step appears in the reference set, it is removed before retrieval. The same retrieved reference set is then evaluated twice, once under the compact decision labeling and once under the full realization labeling. This protocol tests which output form is recoverable from the same representation neighborhood.

For Observation 2, the diagnostic analysis appends a compact decision candidate to the original agent input and treats the appended tokens as the candidate token span under Γ_t. This construction exposes the candidate span for measuring decoder evidence; online candidate instantiation remains the CGD decoder-side procedure. We evaluate the frozen decoder computation over this span and construct two cases. In Decision Accepted cases, the candidate contains the executable action and arguments required by the current state and action space, and is accepted as the step output. In Decision Rejected cases, the candidate remains close in length and form but contains missing, mismatched, or incorrect arguments, and is therefore not accepted as the step output. We compare these two cases using the span concentration gain G_t.

Table 12: Retrieval robustness controls for Observation 1 on ReAct in ALFWorld. We evaluate layers $\{4, 8, 12, 16, 20, 24, 28\}$. Mean gaps are averaged over evaluated layers after family-macro aggregation.

Retrieval pool	Mean Δ_{rec}	Mean Δ_{ent}
Standard	0.246	0.269
Trajectory-excluded	0.213	0.231
Held-out	0.187	0.218

G Additional Validation for Local Recoverability

This appendix provides two additional validations for Observation 1 on ReAct in ALFWorld. Both validations use the same frozen backbone and layerwise representations as the main observation study. The first validation tests whether the decision–realization gap in local recoverability is robust to task-family imbalance and trajectory leakage in the retrieval pool. The second validation addresses a different concern: nearest-neighbor neighborhoods may reflect contextual similarity, such as task family, observation content, prompt structure, or environment state, rather than the target output form alone. We therefore train linear classifiers on frozen layerwise representations under the compact-decision and full-realization labelings. This supervised probe tests whether compact decisions are more directly accessible from the representations than full realizations, beyond the neighborhood structure used by KNN retrieval.

G.1 Retrieval Robustness Controls

We use successful ReAct trajectories from ALFWorld and replay each step through the frozen backbone. For each query step, we extract the layerwise representation with the same pooling and retrieval rule as in Appendix F. We evaluate the decoder layers used in the main observation analysis and retrieve $K = 4$ successful reference steps at each layer.

Each reference step is assigned two labels. Under the compact decision labeling, the label is the executable ALFWorld action after canonicalization under the environment action interface. Under the full realization labeling, the label is the corresponding normalized ReAct step output in the base agent's native format. The retrieved reference set is shared across the two labelings, so the comparison measures which output form is more locally recoverable from the same representation neighborhood.

We evaluate three retrieval pools:

- **Standard.** Only the query step itself is removed from the reference pool.
- **Trajectory-excluded.** All steps from the same trajectory as the query are removed before retrieval.
- **Held-out.** We split trajectories into folds and require each query to retrieve only from trajectories assigned to other folds.

To control for task-family imbalance, we report family-macro metrics. For each retrieval pool and decoder layer, we first compute recoverability and Consensus entropy within each ALFWorld task family, and then average the values across families with equal weight. We report two gaps:

$$\Delta^{\ell}_{\mathrm{rec}} = \mathrm{Rec}^{\ell}(\mathcal{Y}^{\mathrm{dec}}) - \mathrm{Rec}^{\ell}(\mathcal{Y}^{\mathrm{real}}), \qquad \Delta^{\ell}_{\mathrm{ent}} = H^{\ell}_{\mathrm{cons}}(\mathcal{Y}^{\mathrm{real}}) - H^{\ell}_{\mathrm{cons}}(\mathcal{Y}^{\mathrm{dec}}). \tag{31}$$

Positive Δ_{rec} indicates higher average compact-decision recoverability. Table 12 summarizes the robustness results. The mean recoverability gap remains positive under all three retrieval pools, although it decreases under stricter retrieval restrictions. The trajectory-excluded pool removes same-trajectory neighbors, and the held-out pool further forces retrieval across disjoint trajectory splits.

G.2 Linear Probe Validation

The retrieval robustness controls evaluate local recoverability through nonparametric neighborhoods. Such neighborhoods may also reflect contextual similarity, including task family, observation content,

prompt structure, or environment state, which can correlate with output labels. We therefore add a supervised linear-probe validation, following the common use of linear probes for analyzing information in frozen representations [4]. This validation tests whether compact decisions are more directly accessible from the frozen layerwise representations than full realizations, beyond the neighborhood structure used by nearest-neighbor retrieval.

For each step and decoder layer, the input representation is the masked-mean pooled layer representation used in the Consensus entropy analysis. The compact decision target is the canonicalized executable ReAct action under the ALFWorld action interface. The full realization target is the corresponding normalized ReAct step output in the base agent's native format. For each evaluated layer, we train an L2-regularized multinomial logistic regression separately under the compact-decision labeling and the full-realization labeling. Splits are performed at the trajectory level so that train and test examples do not come from the same trajectory. Both labelings share identical folds and test steps, and we report the mean of per-fold macro-F1 over five folds. We evaluate layers $\{4, 8, 12, 16, 20, 24, 28\}$.

The probe metrics are macro-F1 and predictive entropy. Macro-F1 averages per-label F1 scores with equal weight over labels present in both the training and evaluation folds, denoted by $\mathcal{Y}_{\mathrm{eval}}$. Per-label F1 is computed on all test steps.

$$\mathrm{MacroF1}^{\ell}(\mathcal{Y}) = \frac{1}{|\mathcal{Y}_{\mathrm{eval}}|} \sum_{y \in \mathcal{Y}_{\mathrm{eval}}} \mathrm{F1}_y^{\ell}, \qquad \mathrm{F1}_y^{\ell} = \frac{2\,\mathrm{Prec}_y^{\ell}\,\mathrm{Recall}_y^{\ell}}{\mathrm{Prec}_y^{\ell} + \mathrm{Recall}_y^{\ell}}. \tag{32}$$

Predictive entropy measures the concentration of the probe's predicted label distribution, normalized by the logarithm of its output label-space size. Lower predictive entropy indicates that the probe assigns probability mass more strongly to one label. We emphasize macro-F1 because the full-realization labeling can have a larger and more imbalanced label space than the compact-decision labeling.

Table 13 reports the layerwise linear-probe comparison. We report:

$$\Delta_{\mathrm{F1}}^{\ell} = \mathrm{MacroF1}^{\ell}(\mathcal{Y}^{\mathrm{dec}}) - \mathrm{MacroF1}^{\ell}(\mathcal{Y}^{\mathrm{real}}), \qquad \Delta_{\mathrm{Ent}}^{\ell} = \mathrm{Ent}^{\ell}(\mathcal{Y}^{\mathrm{real}}) - \mathrm{Ent}^{\ell}(\mathcal{Y}^{\mathrm{dec}}). \tag{33}$$

The layerwise results complement the neighborhood-based Consensus entropy analysis. Instead of asking whether nearby successful reference steps concentrate on the same label, the linear probe asks whether a simple trained classifier can recover the label from the same frozen representation. The positive Δ_{F1} values across layers indicate that stronger compact-decision recoverability over train-supported evaluation labels is not specific to the nonparametric retrieval rule. Predictive entropy is reported descriptively.

Table 13: Layerwise linear-probe comparison of compact decisions and full realizations on ReAct in ALFWorld.

Layer ℓ	Dec. F1	Real. F1	ΔF1	Dec. Ent.	Real. Ent.	ΔEnt.
L4	0.602	0.397	0.205	0.742	0.946	0.204
L8	0.684	0.415	0.269	0.654	0.941	0.287
L12	0.729	0.476	0.253	0.608	0.911	0.303
L16	0.791	0.501	0.290	0.526	0.879	0.353
L20	0.803	0.541	0.262	0.493	0.835	0.342
L24	0.776	0.512	0.264	0.541	0.872	0.331
L28	0.721	0.491	0.230	0.629	0.886	0.257

H Limitations

Several aspects of MIRA remain for further study. First, constrained by available hardware resources, our backbone transfer study covers models up to 32B parameters; whether the same realization amortization behavior scales to substantially larger backbones remains a direction for future evaluation. Second, our experiments focus on text-based interactive benchmarks and representative direct-planning, code-based, and skill-based agent formats. Extending the same realization amortization

principle to multimodal agents and related benchmarks mainly requires reliable extraction of compact decisions and candidate schema under the corresponding action space. Third, our implementation uses a standard decoding stack with a temporary suffix branch for candidate instantiation and AGC scoring. Further systems-level optimization, such as batching, kernel-level cache reuse, or serving-specific scheduling, may further reduce the overhead of the candidate support check.

I MIRA Behavior Example

Figures 5–7 illustrate how MIRA changes native realization windows on ALFWorld across ReAct, ReCode, and SkillNet.

Base ReAct

```
Current ALFWorld state
  Goal: put some tomato on microwave
  Observation: you pick up tomato 3
    from fridge 1
  Target receptacle: microwave 1

Native realization points
  Step t
    Thought:
      I need to go to microwave 1.
    Action: go to microwave 1

  Step t+1
    Action: open microwave 1

  Step t+2
    Thought:
      The microwave is open, so I can
      place the tomato inside.
    Action:
      put tomato 3 in/on microwave 1

Execution
  same environment trace, 632 output tokens
```

ReAct+MIRA

```
Same current state and interface

Each compact decision is committed
independently by CGD and AGC.

  Step t
    schema: go_to(location)
    output: go to microwave 1

  Step t+1
    schema: open(receptacle)
    output: open microwave 1

  Step t+2
    schema: put_in_on(obj, receptacle)
    obj        = tomato 3
    receptacle = microwave 1
    output:
      put tomato 3 in/on microwave 1

AGC support check at each step
  C_{t+k} >= tau_c
  G_{t+k} >= tau_G

Execution
  same environment trace, 405 output tokens
```

Figure 5: Illustrative ReAct behavior on ALFWorld before and after MIRA.

Base ReCode

```
Current ReCode node
  Task: cool some lettuce and put it
    in sinkbasin
  Variables:
    obj_ID             = lettuce 2
    cooling_location_ID = fridge 1

Native code realization points
  Step t
    Thought:
      I need to navigate to the fridge.
    Execute:
      run(f"go to {cooling_location_ID}")

  Step t+1
    Thought:
      Cooling is valid with a fridge.
    Execute:
      run(f"cool {obj_ID} with "
          f"{cooling_location_ID}")

Executor trace
  go to fridge 1
  cool lettuce 2 with fridge 1
```

ReCode+MIRA

```
Same function inputs and executor

Each compact code decision is
instantiated by the frozen decoder.

  Step t
    schema: go_to(location)
    location = fridge 1
    output:
      run("go to fridge 1")

  Step t+1
    schema: cool_with(obj, receptacle)
    obj        = lettuce 2
    receptacle = fridge 1
    output:
      run("cool lettuce 2 with fridge 1")

AGC support check at each step
  C_{t+k} >= tau_c
  G_{t+k} >= tau_G

Executor trace
  go to fridge 1
  cool lettuce 2 with fridge 1
```

Figure 6: Illustrative ReCode behavior on ALFWorld before and after MIRA.

Base SkillNet

```
Current SkillNet state
  Task: put some soapbottle on cabinet
  Observation:
    You pick up soapbottle 1 from
    countertop 1.
  Target receptacle: cabinet 1

Native realization points
  Step t
    decide next controller step
    Action: go to cabinet 1

  Step t+1
    react to closed receptacle
    Action: open cabinet 1

  Step t+2
    complete placement
    Action:
      put soapbottle 1 in/on cabinet 1

Execution
  success in the same trajectory
```

SkillNet+MIRA

```
Same state and skill interface

MIRA commits one compact skill
decision at the realization point.

  Step t
    schema:
      place_in_receptacle(
        obj, receptacle)
    obj        = soapbottle 1
    receptacle = cabinet 1
    output:
      place_in_receptacle(
        obj="soapbottle 1",
        receptacle="cabinet 1")

AGC support check
  C_t >= tau_c
  G_t >= tau_G

Executor trace, not extra LLM output
  go to cabinet 1
  open cabinet 1
  put soapbottle 1 in/on cabinet 1
```

Figure 7: Illustrative SkillNet behavior on ALFWorld before and after MIRA.

J Broader Impacts and Asset Licenses

J.1 Broader Impacts

MIRA studies inference-time realization amortization for long-horizon language agents. Its main positive impact is to reduce unnecessary model-side realization tokens and inference cost for benchmark agents, which can make controlled evaluation of interactive agents more accessible and reduce repeated inference compute. The work does not collect new human-subject data, run crowdsourcing studies, or release newly scraped raw data. Prefix Memory is constructed from successful training trajectories of the corresponding benchmark and base-agent setting, with evaluation trajectories excluded from memory construction. Potential negative impacts include lowering the cost of running automated agents in undesirable applications, reproducing biases or unsafe behaviors already present in the underlying models and benchmark data, and encouraging over-trust in compact decisions when false accepts occur. We mitigate these risks by evaluating only public text-based research benchmarks, reporting false accept rate and limitations, retaining the base-agent fallback path.

J.2 Licenses for Existing Assets

This work builds on existing public research assets. The benchmark environments are ALFWorld [26], ScienceWorld [31], and WebShop [36]. The agent implementations and formats considered in the experiments are based on ReAct [37], CodeAct [32], ReCode [38], and SkillNet [17]. The model backbones include Qwen3 [21] and Llama 3 [18]. All third-party assets are used under their respective official terms. Third-party dependencies retain their corresponding license notices, and benchmark data and model weights are obtained from their original sources.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: The abstract and introduction state the scope of MIRA as inference-time realization amortization for long-horizon LLM agents, and the claims are supported by the observation study, method design, and experiments in Sections 3–5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: Section H discusses limitations on backbone scale, benchmark modality, and systems-level optimization of the temporary suffix branch.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A].

Justification: The paper does not present theoretical theorems or formal proofs. The equations in the paper define metrics, retrieval operations, and the MIRA inference procedure.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes].

Justification: Sections 3 and 4 define the algorithm, metrics, benchmarks, baselines, backbone, and thresholds. Appendix A provides split statistics, hyperparameters, memory construction details, and execution settings.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes].

   Justification: The experiments use public benchmarks and open model backbones, and the code will be released.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes].

   Justification: Section 5.1 describes benchmarks, baselines, metrics, backbone, and threshold selection. Appendix A reports split sizes, hyperparameters, and logging details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes].

   Justification: The main result tables report mean values averaged over three runs. The observation study reports standard deviations across agent formats.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes].

   Justification: Section 5.1 and Appendix A state that the experiments use Hugging Face Transformers on 8×NVIDIA A800 80GB GPUs, and the tables report average inference time per episode.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: The work uses public benchmarks and existing LLM backbones for algorithmic evaluation, does not involve human subjects or sensitive personal data, and is presented as an inference-efficiency method.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes].

Justification: Appendix J discusses broader impacts. It identifies positive impacts from reducing inference-time realization tokens and repeated agent-evaluation compute, and negative impacts such as lowering the cost of undesirable automated-agent use, inheriting biases or unsafe behaviors from base models and benchmark data, and over-trusting compact decisions when false accepts occur. The appendix also describes mitigations, including evaluation on public research benchmarks, false-accept reporting, limitations, and the base-agent fallback path.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A].

    Justification: The paper does not release a new high-risk pretrained model, image generator, or scraped dataset. It evaluates an inference-time method on existing public benchmarks and model backbones.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes].

Justification: Appendix J credits the benchmarks, agent implementations, and model backbones used in the experiments, states that they are used under their official terms, and clarifies that license notices will be preserved while third-party data and model weights are obtained from their original sources.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A].

Justification: The paper does not introduce or release a new dataset, model checkpoint, or benchmark asset. MIRA is presented as an inference-time method evaluated on existing assets.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A].

Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: The paper does not involve human subjects, so IRB approval or equivalent review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes].

    Justification: LLMs are central to the method and experiments. The paper describes the base-agent setting, the decoder-side use of MIRA, the Qwen3-8B backbone, and additional backbone transfer results.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.