MOSAIC: Scaling Long-Horizon Language Agents via Multi-Scale Adaptive Inference Control

Bowei He[1,2], **Meng Ding**[3], **Weixu Zhang**[2], **Xunzhuo Liu**[4], **Huamin Chen**[5], **Xue Liu**[1,2]
[1]MBZUAI, [2]McGill University, [3]Umass Boston, [4]AMD, [5]Microsoft
Bowei.He@mbzuai.ac.ae

Abstract

Large language model (LLM) agents have shown strong potential in tackling complex, multi-step tasks, yet scaling them to long-horizon settings remains fundamentally constrained by two problems: context window saturation and uniform allocation of inference-time compute across steps of vastly different cognitive demands. We introduce MOSAIC (**M**ulti-Scale **O**rchestrated **S**cale-**A**daptive **I**nference **C**ontrol), a hierarchical agent framework that decomposes long-horizon decision making into three temporal scales: strategic, tactical, and operational, each governed by distinct context scopes and reasoning budgets. Central to MOSAIC is a *Scale Scheduler*, a lightweight policy trained via reinforcement learning that dynamically determines (i) which abstraction level should be invoked at each time step, and (ii) how many reasoning tokens to allocate, based on the estimated decision complexity. We further propose *inter-scale context distillation*, a bidirectional information compression mechanism that maintains coherent state representations across scales without redundant token consumption. Theoretically, we show that under a fixed total inference budget, MOSAIC's adaptive allocation policy achieves a strictly lower regret bound than any fixed-rate policy in a hierarchical semi-MDP formulation, and that this advantage is retained, in proportion to the squared correlation between estimated and true information gain, when the scheduler's estimates are imperfect. Empirically, MOSAIC achieves state-of-the-art or competitive results on four diverse long-horizon benchmarks: SWE-Bench Verified, BrowseComp, WebArena, and GAIA (Level 2&3), while reducing total inference tokens by 21–45% relative to flat ReAct baselines. Ablation studies confirm that each component of multi-scale decomposition, adaptive scheduling, and context distillation contributes meaningfully to both performance and efficiency. Trained without any data from the evaluation benchmark, the scheduler still retains 85–87% of the gain over ReAct, and it transfers zero-shot to unseen tool environments. Code is available at https://github.com/BokwaiHo/MOSAIC.

1 Introduction

LLM-based agents have rapidly evolved from single-turn assistants into autonomous systems capable of browsing the web [Wei et al., 2025], writing and debugging software [Jimenez et al., 2024], and orchestrating multi-tool workflows [Mialon et al., 2024]. The dominant interaction paradigm, exemplified by ReAct [Yao et al., 2023b], interleaves reasoning traces with environment actions in a flat loop: the agent reasons, acts, observes, and repeats. This design is elegant for short tasks but encounters fundamental scaling barriers as horizon length grows from tens to hundreds of steps.

Problem 1: Uniform inference allocation. A flat agent allocates roughly the same reasoning effort to every step, whether the step involves a high-stakes strategic pivot ("should I refactor the entire module or patch locally?") or a routine mechanical action ("scroll to line 42"). Consider a realistic software engineering scenario: an agent resolving a cross-module bug in SWE-Bench spends 200

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

steps total, yet only 10–15 of those steps involve genuinely difficult decisions, like identifying the root-cause file, choosing a fix strategy, and verifying side effects. The remaining 185 steps are navigational (opening files, scrolling, running tests) and require minimal deliberation. Despite this, flat agents apply the same chain-of-thought budget throughout, wasting tokens on trivial actions and starving critical junctures of the deeper reasoning they need. Recent work on inference-time compute scaling [Snell et al., 2024] confirms this intuition quantitatively: the marginal value of additional reasoning tokens varies dramatically across decision points, and optimal allocation can outperform uniform scaling of model parameters. Yet no existing agent framework exploits this heterogeneity in the multi-step setting.

Problem 2: Context window saturation. As trajectories lengthen, the full interaction history eventually exceeds the model's effective context window, forcing either truncation which loses critical early decisions such as the original task decomposition, or naive summarization that collapses nuanced state into a lossy digest. The severity of this problem is well-documented: on BrowseComp tasks requiring 100+ search actions, a flat agent's context grows to 200K+ tokens, far exceeding the effective attention range of most open-weight models [Wei et al., 2025]. Context-Folding [Sun et al., 2025] addresses this by learning to compress context via reinforcement learning (RL) within a compact 32K budget, and COMPASS [Wan et al., 2025] maintains evolving context summaries, but both operate at a single temporal scale: they cannot differentiate what information is relevant at what level of abstraction. A strategic decision ("should I change the search query entirely?") needs a high-level summary of all searches attempted so far, while an operational decision ("which link to click?") needs only the current page content. Conflating these requirements into one context representation is inherently suboptimal.

Our approach. We observe that these two problems share a common structural root: flat agents ignore the *natural multi-scale structure* of long-horizon tasks. Drawing on hierarchical cognitive control [Kahneman, 2011] and temporal abstraction in RL [Sutton et al., 1999, PRECUP, 2000], we propose MOSAIC, a three-scale hierarchical agent framework. Long-horizon tasks naturally decompose into nested temporal scales: a *strategic* scale maintaining task-level goals and a compressed global state graph (invoked rarely, at task onset and critical turning points); a *tactical* scale managing subtask planning and tool selection (invoked at subtask boundaries, spanning 5–20 steps); and an *operational* scale handling individual actions with a minimal sliding-window context (invoked every step). Each scale operates with a tailored context window and reasoning budget, e.g., 4,096 tokens for strategic plan revision versus 512 for routine file navigation. Central to MOSAIC is the *Scale Scheduler*, a lightweight RL-trained policy that monitors signals such as plan divergence, action entropy, and error flags to dynamically detect *when* deeper reasoning is warranted, routing each decision to the appropriate scale with an adaptive token budget. This distinguishes MOSAIC from prior hierarchical agents that rely on fixed two-level decompositions with static replanning frequencies. We further introduce *inter-scale context distillation*: a Distill$\uparrow$ operator compresses low-level execution logs into structured summaries for higher layers, while a Refine$\downarrow$ operator expands high-level directives into actionable instructions, ensuring coherent cross-scale state representations without redundant token consumption.

Concretely, MOSAIC makes three contributions:

1. **Multi-scale decision architecture.** We formalize a three-layer hierarchy where each scale maintains its own context scope and reasoning budget, connected by bidirectional information flow through *inter-scale context distillation* (§3.1).
2. **Scale Scheduler.** A lightweight RL-trained policy that observes the current state and dynamically decides both the active scale and the reasoning token budget, optimizing a joint objective of task performance and inference efficiency (§3.2).
3. **Theoretical analysis.** We formalize the multi-scale decision process as a hierarchical semi-MDP and prove that adaptive budget allocation achieves strictly lower regret than uniform allocation under mild conditions, even when the information gain is only estimated (§3.3).

We evaluate MOSAIC on four benchmarks spanning software engineering, web browsing, web interaction, and general assistant tasks. Across all four, MOSAIC matches or exceeds strong baselines including ReAct, Plan-and-Act [Erdogan et al., 2025], Context-Folding [Sun et al., 2025], and GoalAct [Chen et al., 2025], while consuming 21–45% fewer inference tokens, and it remains the best method when all baselines receive the same adaptation budget. Without any target-benchmark

data, its scheduler still retains 85–87% of the gain over ReAct and transfers zero-shot to unseen tool environments (§4.2).

2 Related Work

LLM-based agents and hierarchical planning. The ReAct paradigm [Yao et al., 2023b] established the interleaved reasoning-action loop as the standard agent architecture, building on chain-of-thought prompting [Wei et al., 2022]. Subsequent work has augmented this loop with self-reflection [Shinn et al., 2023], tree-structured deliberation [Yao et al., 2023a], memory management [Packer et al., 2023], and open-ended skill acquisition [Wang et al.]. LLMs as zero-shot planners [Huang et al., 2022] further demonstrated that language models encode sufficient world knowledge for task decomposition, and comprehensive surveys [Xi et al., 2025, Durante et al., 2024] trace the rapid evolution toward multi-modal autonomous systems. Recognizing the limitations of flat agents on long-horizon tasks, recent work introduces explicit hierarchical planning: Plan-and-Act [Erdogan et al., 2025] trains separate Planner and Executor models; GoalAct [Chen et al., 2025] maintains a continuously updated global plan with skill-based execution; GLIDER [Hu et al., 2025] applies offline hierarchical RL to decompose tasks into chain-of-thought sub-problems; Agent-as-Tool [Zhang, 2025] decouples planning from tool execution via GRPO; L2M2 [Geng et al., 2025] integrates LLM planners with low-level MARL agents; and AgentSwift [Li et al., 2026] employs value-guided hierarchical search. However, all of these approaches employ a *fixed* two-level hierarchy with predetermined interaction patterns or static planning frequencies. That is, none adaptively adjusts the granularity of reasoning based on the decision complexity at each step.

Context management and temporal abstraction. As agent trajectories lengthen, managing the context window becomes critical. Context-Folding [Sun et al., 2025] trains agents via RL to manage context within a compact 32K active budget; COMPASS [Wan et al., 2025] maintains evolving context summaries for strategic coherence; the Cognitive Bandwidth Bottleneck [Xu et al., 2025] proposes planning with schemas rather than actions to reduce per-step cognitive load; and DeepPlanner [Fan et al., 2025] uses advantage shaping to improve the planning stage of deep research agents. While effective, these methods operate at a single temporal scale, missing the opportunity to maintain qualitatively different representations at different abstraction levels. A principled foundation for multi-scale decision making exists in reinforcement learning: the options framework [Sutton et al., 1999] and the MAXQ decomposition [Dietterich, 2000] formalize temporal abstraction in MDPs and semi-MDPs. Recent work has revisited these ideas with LLMs: LARAP [Zhang et al., 2025] combines LLM planners with parameterized action primitives for robotic manipulation, and VanTA [Liu et al., 2025] uses vision-language models to guide temporal skill extraction. However, none of them integrates adaptive inference budget allocation into the hierarchy.

Inference-time compute scaling. [Snell et al., 2024] demonstrate that optimally allocating test-time compute can be more effective than scaling model parameters. DeepSeek-R1 [Guo et al., 2025] and related reasoning models show that extended chain-of-thought at inference time dramatically improves complex problem solving, and ToRL [Li et al., 2025] extends this to tool-integrated RL settings. These works focus on scaling compute for single-turn reasoning; we extend the principle to multi-step agent trajectories, where the key question shifts from *how much* to reason to *which steps* deserve more reasoning, a question that our framework answers through a learned Scale Scheduler.

3 Methodology

We present MOSAIC, whose architecture consists of three components: (1) a multi-scale decision stack (§3.1), (2) a Scale Scheduler for adaptive inference control (§3.2), and (3) theoretical guarantees on the optimality of adaptive allocation (§3.3). Figure 1 provides a framework overview.

3.1 Multi-Scale Decision Architecture

We formalize the agent's interaction with the environment as a trajectory $\tau = (s_0, a_0, o_0, s_1, a_1, o_1, \ldots)$, where s_t is the state, a_t the action, and o_t the observation at step t. In a flat agent, every step invokes the same policy $\pi_\theta(a_t \mid s_t, \tau_{<t})$ with the full trajectory as context. As the trajectory grows, this leads to both redundant computation on routine steps and context overflow on long tasks. MOSAIC instead maintains three policies at different temporal

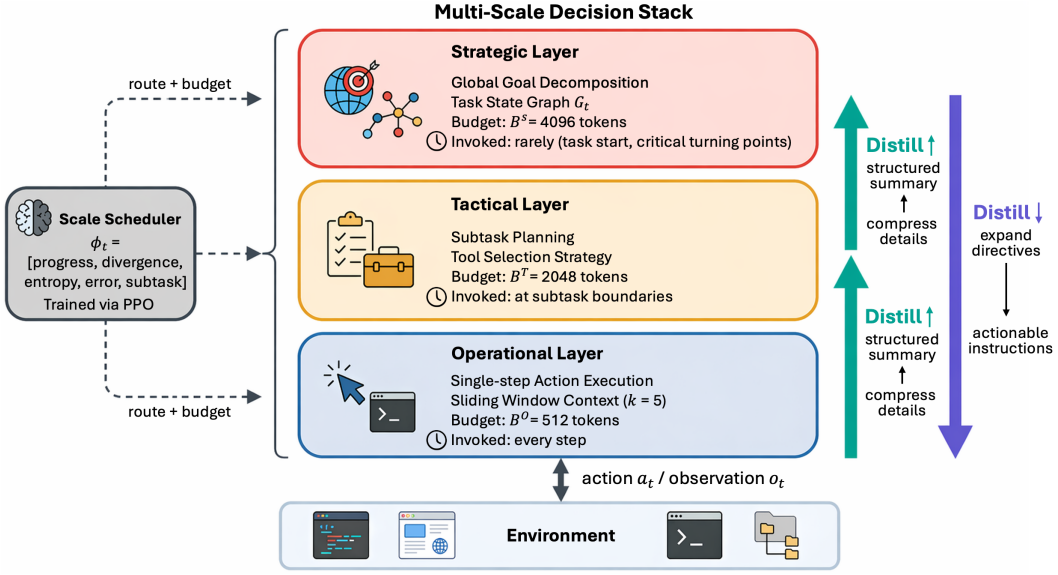


Figure 1: Overview of the MOSAIC framework. The Scale Scheduler dynamically routes decisions to the appropriate temporal scale, while inter-scale context distillation maintains coherent state representations across layers.

scales, each with a tailored context scope and reasoning budget, thereby decoupling the "how deep to think" decision from the "what to do" decision.

Strategic Layer (π^S). The strategic policy operates over the entire task horizon. It is invoked at task initialization and at *escalation points* detected by the Scale Scheduler (e.g., when the agent encounters an unexpected error that invalidates the current plan, or when it completes a major subtask and must decide the next phase). Its context c_t^S consists of a structured task state graph $\mathcal{G}_t = (V_t, E_t)$, where nodes V_t represent goals, subgoals, and constraints (e.g., "fix the failing test in `utils.py`", "do not modify the public API"), and directed edges E_t encode dependencies and completion status (e.g., "subgoal A must precede subgoal B", "subgoal C is 80% complete"). The strategic layer produces a *strategic plan* p_t^S: a high-level decomposition of the remaining task into ordered subtask specifications. Formally:

$$p_t^S = \pi^S(c_t^S), \quad c_t^S = \textsc{Encode}(\mathcal{G}_t, \text{summary}(\tau_{<t})), \tag{1}$$

where $\text{summary}(\cdot)$ compresses the full trajectory into a fixed-length natural language summary via an LLM call (capped at 2,048 tokens to prevent context bloat). The reasoning budget for this layer is B^S tokens (set to 4,096 by default), the largest among all scales, reflecting the high cognitive demand of strategic decisions such as task decomposition, goal reprioritization, and constraint revision.

Tactical Layer (π^T). The tactical policy operates at the subtask level, spanning 5–20 environment steps. It bridges the gap between abstract strategic goals and concrete operational actions by translating strategic subtask specifications into executable plans with explicit tool-selection strategies and intermediate checkpoints. Its context c_t^T is:

$$c_t^T = (p^S_{\text{current}},\ \log(\tau_{t-w_T:t}),\ \text{tool_inventory}), \tag{2}$$

where p^S_{current} is the active subtask specification from the strategic plan, w_T is a tactical window size (default 20 steps) that captures the recent execution history relevant to the current subtask, and tool_inventory lists the available actions and tools in the current environment (e.g., bash commands, search APIs, file editors). The tactical layer outputs a *tactical plan* p_t^T: a sequence of tool-selection strategies and intermediate checkpoints. For example, given a strategic directive "locate and fix the bug in the data pipeline", the tactical layer might produce: "(1) use `grep` to search for relevant error patterns, checkpoint: identify the faulty function; (2) read the function and its callers, checkpoint: understand the root cause; (3) apply the fix and run the test suite". The reasoning budget is $B^T = 2{,}048$ tokens. Only the tactical layer may name tools and only the strategic layer writes to $\mathcal{G}_t$, which keeps the two layers disjoint (Appendix L).

Operational Layer (π^O). The operational policy executes individual actions. It operates with minimal context, a sliding window of the most recent w_O steps (typically $w_O = 3$–5) and the current tactical instruction, thus enabling fast, reactive execution:

$$a_t = \pi^O(p^T_{\text{current}},\ \tau_{t-w_O:t}). \tag{3}$$

The small context window avoids loading irrelevant early-trajectory tokens, and the small reasoning budget $B^O = 512$ tokens ensures that the operational layer does not over-deliberate on routine actions such as navigating files, clicking links, or executing commands. This layer handles the majority of steps (approximately 75%) and is the primary source of inference savings.

Inter-Scale Context Distillation. A critical challenge in hierarchical architectures is maintaining coherent state representations across scales: the strategic layer needs to be informed of tactical outcomes, and the tactical layer needs to interpret strategic directives. We address this through two complementary operators that implement bidirectional information flow.

The **Distill↑** operator compresses detailed lower-layer execution logs into the structured representation expected by the higher layer. When the tactical layer completes a subtask (or encounters a critical event), it generates a structured summary:

$$\Delta_t^{T\to S} = \textsc{Distill}\uparrow(\log(\tau_{\text{subtask}})) = (\text{outcome}, \text{key_findings}, \text{unexpected_events}). \tag{4}$$

This update is merged into the task state graph $\mathcal{G}_t$ by updating the completion status of the relevant subgoal node and, if unexpected events occurred, adding new constraint nodes. The graph is created once by a single LLM call and never regenerated: it changes only through deterministic, schema-validated updates keyed on node ids named in Distill↑ reports, so it can become stale but never corrupt (Appendix B.4). For instance, if the tactical layer discovers during bug fixing that the issue originates from a different module than initially assumed, the Distill↑ output would flag this as an unexpected event, prompting the strategic layer to revise its decomposition. An analogous Distill↑ operator also compresses operational-layer logs into tactical-level summaries, e.g., "attempted 3 search queries, found relevant result on query 2, downloaded 2 documents".

Conversely, the **Refine↓** operator expands high-level directives into actionable instructions for the lower layer:

$$p^T_{\text{new}} = \textsc{Refine}\downarrow(p^S_{\text{current}}, c^T_t). \tag{5}$$

The Refine↓ operator takes a strategic subtask specification (e.g., "investigate the authentication module for race conditions") and the current tactical context (available tools, recent observations), and produces a concrete tactical plan with tool choices, checkpoints, and the expected result of every step. Both Distill↑ and Refine↓ are implemented as LLM calls with specialized prompts (detailed in Appendix B; fidelity analysis in Appendix K), keeping the framework model-agnostic and avoiding the need for additional trained modules.

3.2 Scale Scheduler

The Scale Scheduler is the central control mechanism that determines, at each step t: (i) the active scale $\ell_t \in \{S, T, O\}$, and (ii) the reasoning token budget $b_t \in [B_{\min}, B_{\max}]$. Unlike prior hierarchical agents that invoke higher-level reasoning on a fixed schedule (e.g., replanning every k steps) or at predetermined trigger points, our scheduler is a learned policy that adapts to the task dynamics in real time.

State representation. The scheduler observes a compact feature vector $\phi_t \in \mathbb{R}^d$ that encodes five signals indicative of decision complexity:

$$\phi_t = [\ \underbrace{e_{\text{progress}}}_{\text{task progress}},\ \underbrace{e_{\text{diverge}}}_{\text{plan divergence}},\ \underbrace{e_{\text{entropy}}}_{\text{action entropy}},\ \underbrace{e_{\text{error}}}_{\text{error signal}},\ \underbrace{e_{\text{subtask}}}_{\text{subtask boundary}}\ ]. \tag{6}$$

Each feature captures a distinct aspect of the current decision state and is computed identically on every benchmark (Appendix B.5). $e_{\text{progress}} \in [0, 1]$ is the fraction of goal and subgoal nodes of $\mathcal{G}_t$ marked done, which requires a Distill↑ report naming the node id and any available programmatic check to pass. $e_{\text{diverge}} = 1 - \cos(E(\hat{y}_t), E(y_t))$ is the cosine distance, under a frozen 22M-parameter sentence encoder E, between the expected result $\hat{y}_t$ that Refine↓ attaches to the current tactical step and the actual result y_t; a high divergence signals that the plan is no longer tracking reality. e_{entropy} is the mean token entropy over the action-name span of the operational output, normalized by $\log|V|$;

Algorithm 1 MOSAIC Inference Procedure

Require: Task description $\mathcal{T}$, backbone LLM π_θ, trained scheduler π_ψ^{sched}

Initialize task state graph $\mathcal{G}_0 \leftarrow \textsc{ParseTask}(\mathcal{T})$
Generate initial strategic plan: $p_0^S \leftarrow \pi^S(\textsc{Encode}(\mathcal{G}_0, \mathcal{T}))$ with budget B^S
Refine into initial tactical plan: $p_0^T \leftarrow \textsc{Refine}\downarrow(p_0^S, \text{tool_inventory})$
for $t = 0, 1, 2, \ldots, T_{\max}$ **do**
  Extract scheduler features: $\phi_t \leftarrow \textsc{ExtractFeatures}(s_t, \mathcal{G}_t, p^S, p^T, \tau_{<t})$
  Select scale and budget: $\ell_t, b_t \leftarrow \pi_\psi^{\text{sched}}(\phi_t)$
  if $\ell_t = S$ **(Strategic) then**
    Distill tactical log: $\Delta \leftarrow \textsc{Distill}\uparrow(\log(\tau_{\text{recent subtask}}))$
    Update graph: $\mathcal{G}_t \leftarrow \textsc{UpdateGraph}(\mathcal{G}_t, \Delta)$
    Replan: $p_t^S \leftarrow \pi^S(\textsc{Encode}(\mathcal{G}_t, \text{summary}(\tau_{<t})))$ with budget b_t
    Refine: $p_t^T \leftarrow \textsc{Refine}\downarrow(p_t^S, c_t^T)$
  else if $\ell_t = T$ **(Tactical) then**
    Distill operational log: $\Delta \leftarrow \textsc{Distill}\uparrow(\log(\tau_{t-w_O:t}))$
    Replan subtask: $p_t^T \leftarrow \pi^T(p^S_{\text{current}}, \log(\tau_{t-w_T:t}), \Delta)$ with budget b_t
  end if
  Execute action: $a_t \leftarrow \pi^O(p^T_{\text{current}}, \tau_{t-w_O:t})$ with budget $\min(b_t, B^O)$
  Observe: $o_t \leftarrow \textsc{Env}(a_t)$; update state s_{t+1}; append to τ
  if task completed or $t = T_{\max}$ **then**
    return final output
  end if
end for

since a model can be confidently wrong, it is only a weak auxiliary signal (Appendix J.3). $e_{\text{error}} \in \{0, 1\}$ flags tool execution failures (e.g., a non-zero exit code or an HTTP error). $e_{\text{subtask}} \in [0, 1]$, the fraction of checkpoints of the active tactical plan already reached, indicates proximity to a subtask boundary. Without an active plan, $e_{\text{subtask}} = e_{\text{diverge}} = 0$; only e_{diverge} needs extra computation (1.9 ms per step, no LLM call).

Policy. The scheduler policy $\pi_\psi^{\text{sched}}(\ell_t, b_t \mid \phi_t)$ is parameterized by a small network ψ (68K parameters) that jointly outputs a categorical distribution over scales and a continuous scalar budget:

$$\ell_t \sim \text{Cat}(\text{softmax}(W_\ell \phi_t + b_\ell)), \quad b_t = B_{\min} + (B_{\max} - B_{\min}) \cdot \sigma(w_b^\top \phi_t + b_b), \tag{7}$$

where $W_\ell \in \mathbb{R}^{3\times d}$, $b_\ell \in \mathbb{R}^3$ are the scale-selection parameters, $w_b \in \mathbb{R}^d$, $b_b \in \mathbb{R}$ are the budget-allocation parameters, and σ denotes the sigmoid function. The scale selection and budget allocation are thus two heads of the same network, sharing the feature representation ϕ_t but producing independent outputs. This factored design allows the scheduler to, for example, route a decision to the tactical layer but with a higher-than-default budget when the subtask is particularly complex.

Training objective. We train the scheduler via Proximal Policy Optimization (PPO; Schulman et al. 2017) with a composite reward that balances task performance against inference efficiency:

$$r_t = \underbrace{r_t^{\text{task}}}_{\text{task progress}} - \underbrace{\lambda \cdot \frac{b_t}{B_{\max}}}_{\text{efficiency penalty}}, \tag{8}$$

where r_t^{task} is a step-level reward derived from the environment (e.g., delta in test pass rate for SWE-Bench, partial credit for intermediate search progress on BrowseComp, functional correctness for WebArena, exact match on GAIA), and $\lambda > 0$ is a trade-off hyperparameter that penalizes excessive token usage. This composite reward incentivizes the scheduler to allocate large budgets only when the marginal return in task progress justifies the additional inference cost, and to default to the efficient operational layer for routine steps.

Besides, the complete MOSAIC inference loop has been summarized in Algorithm 1.

3.3 Theoretical Analysis

We formalize the adaptive inference allocation benefit within the hierarchical semi-MDP framework.

Setup. Consider a hierarchical semi-MDP $\mathcal{M} = (\mathcal{S}, \{\mathcal{A}^\ell\}_{\ell \in \mathcal{L}}, P, R, \gamma)$ where $\mathcal{L} = \{S, T, O\}$ is the set of scales. At each decision point, the agent selects a scale ℓ and an action from $\mathcal{A}^\ell$. Each scale ℓ has an associated reasoning cost c^ℓ (in tokens) and an action quality function $Q^\ell(s, a; b)$ that depends on the allocated budget b. We assume Q^ℓ is concave in b (diminishing returns of additional reasoning), which is empirically well-supported [Snell et al., 2024]: doubling the reasoning budget from 256 to 512 tokens yields a larger quality improvement than doubling from 2,048 to 4,096.

Definition 1 (Budget allocation policy)**.** *A* budget allocation policy $\beta : \mathcal{S} \to \mathbb{R}_+^{|\mathcal{L}|}$ *assigns a reasoning budget* $\beta_\ell(s)$ *to each scale at state* s*, subject to an average budget constraint* $\mathbb{E}_s[\sum_\ell \beta_\ell(s)] \leq B_{\text{total}}$.

Definition 2 (Uniform policy)**.** *The* uniform allocation policy β^{uni} *assigns* $\beta_\ell(s) = B_{\text{total}}/|\mathcal{L}|$ *for all* s *and* ℓ.

Theorem 1 (Advantage of adaptive allocation)**.** *Under the regularity assumptions of Appendix A.1, let* $V^\beta(s_0)$ *denote the expected cumulative reward under allocation policy* β*. If the information gain* $I^\ell(s) := \partial Q^\ell(s, a^*; b)/\partial b\big|_{b=\bar{b}}$ *varies across states (i.e.,* $\text{Var}_s[I^\ell(s)] > 0$ *for some* ℓ*), then there exists an adaptive policy* β^* *such that:*

$$V^{\beta^*}(s_0) - V^{\beta^{\text{uni}}}(s_0) \geq \frac{\gamma}{2(1-\gamma)} \sum_\ell \text{Var}_{s \sim d^\pi}[I^\ell(s)] \cdot \kappa_\ell, \tag{9}$$

where d^π *is the state visitation distribution and* $\kappa_\ell > 0$ *is a scale-dependent constant related to the curvature of* Q^ℓ.

Proof Sketch. The proof proceeds by a second-order expansion of the concave quality function around the uniform budget. For any fixed total budget, reallocating budget from low-information-gain states to high-information-gain states strictly increases expected quality. The gap is lower-bounded by the variance of the information gain, weighted by κ_ℓ, which is proportional to how often scale ℓ is invoked and inversely proportional to a bound L_ℓ on the curvature $|\partial^2 Q^\ell / \partial b^2|$. The full derivation, including the extension to the multi-scale semi-MDP setting with discount factor γ, is in Appendix A. □

Remark. Theorem 1 provides a principled justification for adaptive allocation: the benefit is proportional to the heterogeneity of decision difficulty across states. In practice, long-horizon agent tasks exhibit high variance in per-step difficulty (e.g., navigating to a file vs. diagnosing a subtle bug), suggesting large potential gains. The theorem also reveals a useful design insight: the Scale Scheduler need not find the globally optimal allocation. That is, any policy that correlates budget with information gain will improve over the uniform baseline, with the improvement magnitude governed by the strength of that correlation. Proposition 2 (Appendix A.2) makes this precise: if budgets follow an optimally scaled estimate $\hat{I}^\ell$, the bound still holds with each term scaled by ρ_ℓ^2, where $\rho_\ell = \text{corr}(I^\ell, \hat{I}^\ell)$ remains clearly positive out of distribution.

4 Experiments

4.1 Experimental Setup

Benchmarks. We select four benchmarks that cover software engineering, web browsing, web interaction, and general multi-step reasoning: (1) **SWE-Bench Verified** [Jimenez et al., 2024]: 500 human-validated real-world GitHub issues. The agent must navigate a codebase and produce a patch, evaluated by unit tests. Average horizon: 50–200 steps. (2) **BrowseComp** [Wei et al., 2025]: 1,266 challenging fact-seeking questions requiring persistent web browsing across tens to hundreds of websites. Average horizon: 80–150 search actions. (3) **WebArena** [Zhou et al., 2024]: 812 realistic web interaction tasks across e-commerce, forums, and content management sites. Average horizon: 15–40 steps. (4) **GAIA Level 2&3** [Mialon et al., 2024]: 320 general AI assistant questions requiring multi-step reasoning, tool use, and web browsing. Level 3 questions average 30+ steps. To test generalization, we also evaluate on two benchmark families whose tools never appear in scheduler training: **OSWorld** [Xie et al., 2024] (369 computer-use tasks) and τ**-bench** [Yao et al., 2025] (retail and airline, with a simulated user).

Baselines. We compare against five representative methods: (1) **ReAct** [Yao et al., 2023b]: Flat interleaved reasoning-action loop, the standard baseline. (2) **Plan-and-Act** [Erdogan et al., 2025]:

Table 1: Main results with Qwen3.5-35B-A3B backbone. Best in **bold**, second-best underlined. Tokens are reported in thousands (K). η = Pareto efficiency ratio (higher is better).

	SWE-Bench Verified		BrowseComp		WebArena		GAIA L2&3		Avg.
Method	Perf.	Tokens	Perf.	Tokens	Perf.	Tokens	Perf.	Tokens	η
ReAct	$28.4_{\pm 1.2}$	487K	$12.8_{\pm 0.9}$	892K	$24.7_{\pm 1.5}$	156K	$38.2_{\pm 2.1}$	312K	5.21
Plan-and-Act	$32.1_{\pm 1.0}$	523K	$15.3_{\pm 1.1}$	834K	$28.5_{\pm 1.3}$	178K	$42.6_{\pm 1.8}$	345K	5.94
GoalAct	$33.8_{\pm 0.8}$	498K	$16.1_{\pm 1.3}$	867K	$30.2_{\pm 1.1}$	165K	$43.8_{\pm 2.0}$	328K	6.31
Context-Folding	$\underline{35.2}_{\pm 1.1}$	312K	$\underline{17.4}_{\pm 1.0}$	589K	$29.8_{\pm 1.4}$	134K	$41.5_{\pm 1.7}$	267K	7.12
Reflexion	$31.6_{\pm 1.3}$	612K	$14.1_{\pm 0.8}$	1024K	$\underline{31.3}_{\pm 1.2}$	213K	$\underline{44.1}_{\pm 1.9}$	389K	5.68
MOSAIC	$\mathbf{37.6}_{\pm 0.9}$	**298K**	$\mathbf{19.2}_{\pm 1.1}$	**512K**	$\mathbf{33.1}_{\pm 1.0}$	**97K**	$\mathbf{46.3}_{\pm 1.6}$	**245K**	**8.84**

Table 2: Performance (Perf.) and token usage (Tok., in K) across three backbone models on SWE-Bench Verified and GAIA L2&3. MOSAIC consistently improves over ReAct across all backbones.

	SWE-Bench Verified				GAIA Level 2&3			
	ReAct		MOSAIC		ReAct		MOSAIC	
Backbone	Perf.	Tok.	Perf.	Tok.	Perf.	Tok.	Perf.	Tok.
Qwen3.5-9B	$21.3_{\pm 1.4}$	524K	$28.9_{\pm 1.2}$	287K	$29.4_{\pm 2.3}$	341K	$37.8_{\pm 1.8}$	218K
Qwen3.5-35B-A3B	$28.4_{\pm 1.2}$	487K	$37.6_{\pm 0.9}$	298K	$38.2_{\pm 2.1}$	312K	$46.3_{\pm 1.6}$	245K
Gemma-4-27B-A4B	$26.7_{\pm 1.1}$	501K	$34.8_{\pm 1.0}$	309K	$36.5_{\pm 1.9}$	327K	$44.1_{\pm 1.5}$	256K

Separate Planner and Executor with synthetic data-driven plan generation. (3) **GoalAct** [Chen et al., 2025]: Global planning with hierarchical skill-based execution. (4) **Context-Folding** [Sun et al., 2025]: RL-trained context management within a compact token budget. (5) **Reflexion** [Shinn et al., 2023]: Self-reflective agent with episodic memory and verbal reinforcement.

Backbone models. To test generality across model scales and architectures, we evaluate with three backbone LLMs: (1) **Qwen3.5-9B** [Yang et al., 2025]: A 9B dense model with strong instruction following. (2) **Qwen3.5-35B-A3B** [Yang et al., 2025]: A 35B mixture-of-experts model with 3B active parameters, offering a favorable compute-quality trade-off. (3) **Gemma-4-27B-A4B** [Deepmind, 2026]: A 27B MoE model with 4B active parameters from Google DeepMind.

Metrics. For each benchmark, we report the standard primary metric: **Resolve Rate** (%) for SWE-Bench Verified, **Accuracy** (%) for BrowseComp and GAIA, and **Success Rate** (%) for WebArena, OSWorld, and τ-bench. Additionally, we report **Total Inference Tokens** (in thousands) as our efficiency metric, and the **Pareto efficiency ratio** $\eta = \text{Performance}/\log(\text{Tokens})$ to capture performance-efficiency trade-off.

Implementation details. The Scale Scheduler is a 2-layer MLP with hidden dimension 256, trained for 5,000 PPO steps on a held-out set of 200 training tasks (50 per benchmark) not overlapping with evaluation sets. The efficiency penalty λ in Eq. (8) is set to 0.1. Context window sizes are: $w_O = 5$ recent steps for the operational layer, $w_T = 20$ steps for the tactical layer, and the strategic layer receives a compressed summary of at most 2,048 tokens. The strategic reasoning budget is $B^S = 4096$ tokens, tactical $B^T = 2048$, and operational $B^O = 512$. The scheduler can override these defaults by up to $\pm 50\%$. All experiments use greedy decoding (temperature 0). The scheduler is trained using 4× A100 GPUs over approximately 18 hours. All methods share one harness, tool set, retry policy, token counter, and (for BrowseComp and GAIA) cached web snapshot, and token counts are end to end (Appendices B.6 and C.4). All experiments report mean ± standard deviation over 3 independent runs with different environment seeds.

4.2 Results and Analysis

Main Results. Table 1 presents the main comparison using Qwen3.5-35B-A3B as the backbone. MOSAIC achieves the best or second-best performance on all four benchmarks while consuming substantially fewer inference tokens. On SWE-Bench Verified, MOSAIC achieves 37.6% resolve rate, outperforming Context-Folding by 2.4 absolute points while using 4.5% fewer tokens. The gain is most pronounced on multi-file issues (horizon > 100 steps), where the strategic layer's ability to maintain a coherent refactoring plan proves critical. On BrowseComp, MOSAIC reaches 19.2% accuracy with 42.6% fewer tokens than ReAct, as the tactical layer efficiently manages search strategy pivots while the operational layer handles routine page navigation quickly. On WebArena, the

Table 3: Generalization of the Scale Scheduler (Perf. / Tokens). **Top:** leave-one-benchmark-out (LOBO) scheduler applied zero-shot vs. the strongest baseline, as published and after matched adaptation (tuned); "Retained" is the share of MOSAIC's gain over ReAct kept by LOBO. **Bottom:** zero-shot transfer to unseen tool environments.

Benchmark	ReAct	Best baseline	Best baseline (tuned)	MOSAIC (LOBO)	+10 tasks	MOSAIC	Retained
SWE-Bench Verified	28.4 / 487K	35.2 / 312K	36.1	$36.4_{\pm 1.0\,/\,311K}$	37.3	$37.6_{\pm 0.9\,/\,298K}$	87%
BrowseComp	12.8 / 892K	17.4 / 589K	18.1	$18.3_{\pm 1.2\,/\,528K}$	19.0	$19.2_{\pm 1.1\,/\,512K}$	86%
WebArena	24.7 / 156K	31.3 / 213K	32.5	$31.9_{\pm 1.1\,/\,104K}$	32.8	$33.1_{\pm 1.0\,/\,97K}$	86%
GAIA L2&3	38.2 / 312K	44.1 / 389K	45.2	$45.1_{\pm 1.7\,/\,259K}$	46.0	$46.3_{\pm 1.6\,/\,245K}$	85%
Unseen family	**ReAct**	**Fixed schedule**	**Best rule-based**	**MOSAIC (zero-shot)**	**+10 tasks**		
OSWorld	11.2 / 428K	13.1 / 372K	14.0 / 318K	$15.6_{\pm 1.2\,/\,274K}$	$16.9_{\pm 1.1\,/\,268K}$		
τ-bench (retail + airline)	33.6 / 186K	36.2 / 171K	37.1 / 148K	$38.9_{\pm 1.4\,/\,126K}$	$40.3_{\pm 1.3\,/\,124K}$		

33.1% success rate represents a 2.9-point absolute improvement over GoalAct, with 41.2% token reduction, benefiting from the tactical layer's tool selection strategies. On GAIA Level 2&3, MOSAIC achieves 46.3%, outperforming Reflexion by 2.2 points despite using 37.0% fewer tokens. With matched adaptation, i.e., the same 200 training tasks and compute as our scheduler (Appendix I), the baselines gain 1.0–1.5 points on average, yet MOSAIC stays best on all four benchmarks with fewer tokens, by 0.6–1.5 points.

Cross-Model Generalization. Table 2 shows that MOSAIC's benefits generalize across backbone models of different scales and architectures. The relative improvement of MOSAIC over ReAct is largest for the smallest model (Qwen3.5-9B: +7.6 points on SWE-Bench, +8.4 on GAIA), suggesting that multi-scale reasoning is especially beneficial when the base model has limited single-turn capacity. For MoE models (Qwen3.5-35B-A3B, Gemma-4-27B-A4B), the token savings are smaller in absolute terms but still substantial (38–39% on SWE-Bench Verified and 21–22% on GAIA).

Generalization of the Scale Scheduler. Trained only on the other three benchmarks (leave-one-benchmark-out, LOBO; Table 3), the scheduler retains 85–87% of MOSAIC's gain over ReAct, beats the strongest published baseline on every benchmark with fewer tokens, and is better than or within one standard deviation of the tuned baselines; 10 target tasks bring it within 0.3 points of the in-distribution scheduler. On OSWorld and τ-bench, whose tools are unseen, it outperforms ReAct, a fixed schedule, and the best rule-based scheduler with fewer tokens, becoming more conservative rather than erratic (Appendix G.2). The sharpest shift we found is in horizon length: training only on tasks under 100 steps costs 1.8–3.2 points on longer ones (Appendix E.6).

Table 4: Ablation study on SWE-Bench Verified.

Variant	Resolve Rate (%)	Tokens (K)
MOSAIC (full)	**37.6**$_{\pm 0.9}$	**298K**
w/o Scale Scheduler (fixed schedule)	$34.1_{\pm 1.0}$	321K
w/o Context Distillation (shared context)	$33.5_{\pm 1.2}$	352K
w/o Strategic Layer (2-scale only)	$35.8_{\pm 0.8}$	305K
w/ merged Strategic & Tactical planner	$35.4_{\pm 1.1}$	341K
w/o Adaptive Budget (uniform B)	$35.3_{\pm 1.1}$	387K
Flat (ReAct baseline)	$28.4_{\pm 1.2}$	487K

Ablation Studies. Table 4 isolates the contribution of each component using Qwen3.5-35B-A3B on SWE-Bench Verified. **Scale Scheduler** (-3.5%): Replacing the learned scheduler with a fixed schedule (invoke strategic every 50 steps, tactical every 10 steps) significantly hurts performance, confirming that *when* to invoke higher-level reasoning matters as much as *having* higher levels. **Context Distillation** (-4.1%): Sharing the full context across all scales leads to both lower performance (context noise) and higher token consumption (+18%). **Strategic Layer** (-1.8%): Removing the top layer has a moderate effect, most pronounced on multi-file issues requiring global coherence, while merging it with the tactical layer into one planner is even worse (-2.2%, +14% tokens). **Adaptive Budget** (-2.3% performance, +30% tokens): Fixing all budgets to the tactical default demonstrates the efficiency value of adaptive allocation.

What Drives the Scale Scheduler? Retraining the scheduler without each feature (Appendix J) shows that plan divergence (-3.0 points on SWE-Bench Verified) and the error flag (-1.9) carry most of its benefit, while e_{subtask}, e_{progress}, and e_{entropy} matter less (-1.4, -0.7, -0.5). These two environment-agnostic features alone recover over three quarters of the learned gain over the fixed schedule, without any task graph, whereas hand-designed thresholds recover only 23% of it, or 51% with per-benchmark tuning (Appendix H).

Analysis of Scale Scheduler Behavior. Figure 2(a) reveals that the scheduler learns interpretable escalation patterns. Strategic reasoning is concentrated at task onset and around the midpoint, corresponding to initial planning and mid-course correction. Tactical reasoning clusters near subtask

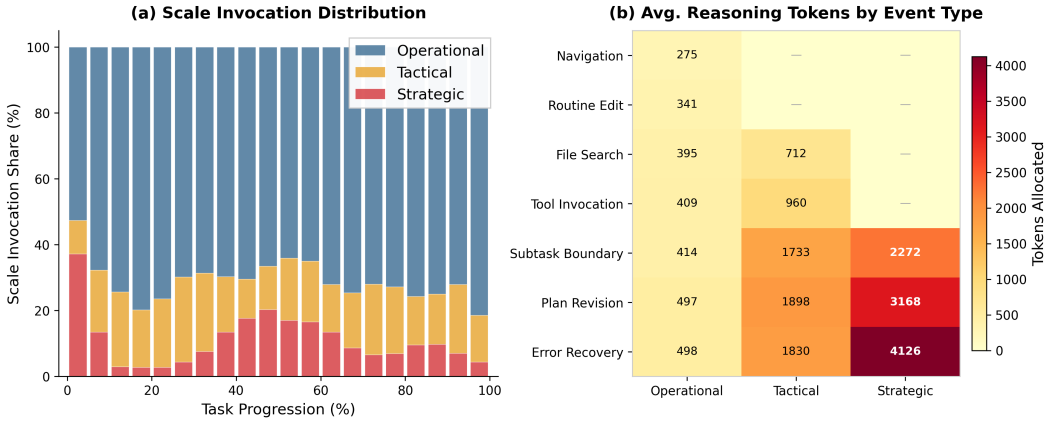


Figure 2: Scale Scheduler behavior on SWE-Bench Verified. (a) Scale invocation frequency across task progression. (b) Reasoning budget allocation conditioned on event type.

boundaries. The operational layer handles 75.3% of all steps, reflecting the intuition that most individual actions are routine. Figure 2(b) shows that the scheduler allocates 2–4× higher budgets to error recovery and plan revision steps compared to navigation steps, validating Theorem 1's prediction that gains from adaptive allocation scale with decision heterogeneity.

Efficiency–Performance Pareto Frontier. By varying the efficiency penalty λ in Eq. (8), MOSAIC traces a Pareto frontier that strictly dominates all baselines (Figure 3). At the aggressive efficiency setting ($\lambda = 0.5$), MOSAIC achieves 32.1% with only 178K tokens, matching GoalAct's performance at 64% lower cost. At the quality-focused setting ($\lambda = 0.01$), it reaches 38.9% at 412K tokens, the highest absolute performance among all methods. The frontier exhibits a concave shape with a knee around $\lambda = 0.1$ (37.6% at 298K tokens): beyond this point, additional budget yields diminishing returns (+1.3% from 298K to 412K), consistent with Theorem 1's prediction that the adaptive policy exhausts high-information-gain decisions first. This knee provides a natural default operating point, while practitioners can shift along the frontier to trade efficiency for quality as deployment constraints require.

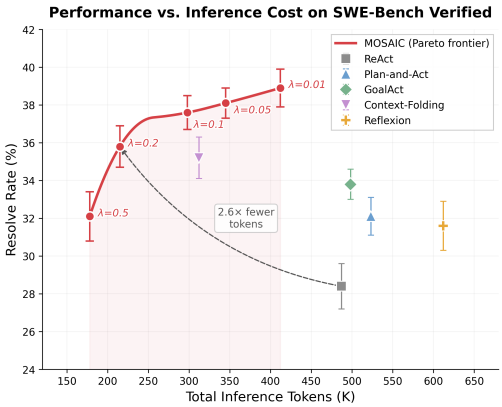


Figure 3: Performance vs. inference cost Pareto frontier on SWE-Bench Verified.

5 Conclusion and Future Work

We introduced MOSAIC, a multi-scale hierarchical framework for long-horizon language agents that adaptively allocates inference-time reasoning budgets across three temporal abstraction levels. By combining a learned Scale Scheduler, inter-scale context distillation, and principled temporal decomposition grounded in semi-MDP theory, MOSAIC achieves state-of-the-art results on SWE-Bench Verified, BrowseComp, WebArena, and GAIA while reducing inference costs by 21–45%, even against baselines given the same adaptation budget, and its scheduler transfers to held-out benchmarks and unseen tool environments. Our theoretical analysis shows that the benefit of adaptive allocation is proportional to the heterogeneity of per-step decision difficulty, a property abundantly present in real-world agent tasks, and that it persists when this difficulty is only imperfectly estimated. Future directions include extending the framework to multi-agent collaboration settings where inter-agent communication creates additional temporal scales, learning the number of scales from data rather than fixing it at three, training the scheduler across a broader range of task horizons, and integrating MOSAIC with inference-time reasoning models where the interaction between extended thinking and scale scheduling may yield further gains.

Acknowledgments and Disclosure of Funding

The research presented in this paper was exclusively supported by the MBZUAI Base Fund. The authors declare no competing interests.

References

Junjie Chen, Haitao Li, Jingli Yang, Yiqun Liu, and Qingyao Ai. Enhancing llm-based agents via global planning and hierarchical execution. *arXiv preprint arXiv:2504.16563*, 2025.

Google Deepmind. Gemma 4: Frontier multimodal intelligence on device. https://huggingface.co/blog/gemma4, April 2026. Accessed: 2026-04-19.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

Zane Durante, Qiuyuan Huang, Naoki Wake, Ran Gong, Jae Sung Park, Bidipta Sarkar, Rohan Taori, Yusuke Noda, Demetri Terzopoulos, Yejin Choi, et al. Agent ai: Surveying the horizons of multimodal interaction. *arXiv preprint arXiv:2401.03568*, 2024.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. In *International Conference on Machine Learning*, pages 15419–15462. PMLR, 2025.

Wei Fan, Wenlin Yao, Zheng Li, Feng Yao, Xin Liu, Liang Qiu, Qingyu Yin, Yangqiu Song, and Bing Yin. Deepplanner: Scaling planning capability for deep research agents via advantage shaping. *arXiv preprint arXiv:2510.12979*, 2025.

Minghong Geng, Shubham Pateria, Budhitama Subagdja, Lin Li, Xin Zhao, and Ah-Hwee Tan. L2m2: A hierarchical framework integrating large language model and multi-agent reinforcement learning. In *International Joint Conference on Artificial Intelligence*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Zican Hu, Wei Liu, Xiaoye Qu, Xiangyu Yue, Chunlin Chen, Zhi Wang, and Yu Cheng. Divide and conquer: Grounding llms as efficient decision-making agents via offline hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 24570–24590. PMLR, 2025.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR, 2022.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

Daniel Kahneman. *Thinking, fast and slow*. macmillan, 2011.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383*, 2025.

Yu Li, Lehui Li, Zhihao Wu, Qingmin Liao, Jianye Hao, Kun Shao, and Fengli Xu. Agentswift: Efficient llm agent design via value-guided hierarchical search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 31843–31851, 2026.

Tian-Shuo Liu, Xu-Hui Liu, Ruifeng Chen, Lixuan Jin, Pengyuan Wang, Zhilong Zhang, and Yang Yu. Semantic temporal abstraction via vision-language model guidance for efficient reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2024.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

D PRECUP. Temporal abstraction in reinforcement learning. *Ph. D. thesis, University of Massachusetts*, 2000.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon llm agent via context-folding. *arXiv preprint arXiv:2510.11967*, 2025.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Guangya Wan, Mingyang Ling, Xiaoqi Ren, Rujun Han, Sheng Li, and Zizhao Zhang. Compass: Enhancing agent long-horizon reasoning with evolving context. *arXiv preprint arXiv:2510.08790*, 2025.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

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

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Baixuan Xu, Tianshi Zheng, Zhaowei Wang, Hong Ting Tsang, Weiqi Wang, Tianqing Fang, and Yangqiu Song. The cognitive bandwidth bottleneck: Shifting long-horizon agent from planning with actions to planning with schemas. *arXiv preprint arXiv:2510.07091*, 2025.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2023b.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025.

Ning Zhang, Yongjia Zhao, Minghao Yang, and Shuling Dai. Llms augmented hierarchical reinforcement learning with action primitives for long-horizon manipulation tasks. *Scientific Reports*, 15(1):36779, 2025.

Yanfei Zhang. Agent-as-tool: A study on the hierarchical decision making with reinforcement learning. *arXiv preprint arXiv:2507.01489*, 2025.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024.

Contents

1 Introduction **1**

2 Related Work **3**

3 Methodology **3**

3.1 Multi-Scale Decision Architecture 3

3.2 Scale Scheduler 5

3.3 Theoretical Analysis 6

4 Experiments **7**

4.1 Experimental Setup 7

4.2 Results and Analysis 8

5 Conclusion and Future Work **10**

A Proofs of Theoretical Results **16**

A.1 Proof of Theorem 1 16

A.2 Proof of Proposition 2 and Empirical Estimates of ρ 17

B Implementation Details **18**

B.1 Scale Scheduler Training 18

B.2 Backbone Model Configuration 19

B.3 Inter-Scale Context Distillation Prompts 19

B.4 Task State Graph Construction and Maintenance 20

B.5 Scheduler Feature Specification 20

B.6 Evaluation Parity and Reproducibility 21

C Per-Benchmark Detailed Results **22**

C.1 SWE-Bench Verified by Issue Complexity 22

C.2 GAIA Results by Level 22

C.3 BrowseComp Results by Question Category 22

C.4 Computational Cost 23

D Case Studies **23**

D.1 Case Study 1: SWE-Bench Cross-Module Bug Fix 23

D.2 Case Study 2: BrowseComp Multi-Hop Research 24

E Sensitivity Analysis **24**

E.1 Sensitivity to λ 24

E.2 Sensitivity to Number of Scales 25

E.3 Sensitivity to Context Window Sizes 25

E.4 Sensitivity to Scheduler Training-Set Size and PPO Seed 25
E.5 Sensitivity to Reward Design 26
E.6 Distribution Shift in Task Horizon 26

F Scale Scheduler Learned Behaviors **26**

G Scheduler Generalization **27**
G.1 Leave-One-Benchmark-Out Transfer 27
G.2 Transfer to Unseen Tool Environments 27

H Non-Learned Schedulers **28**

I Baseline Parity with Matched Adaptation **28**

J Scheduler State Representation **29**
J.1 Feature Ablations 29
J.2 Robustness to Feature Noise 29
J.3 Reliability of Action Entropy 29

K Fidelity of Inter-Scale Context Distillation **30**

L Separation of the Strategic and Tactical Layers **30**

M Limitations **31**

N Ethics Statement **32**

O Broader Impacts **32**

P Declaration of LLM usage **33**

A Proofs of Theoretical Results

A.1 Proof of Theorem 1

We provide the full proof of Theorem 1. Recall the setup: we have a hierarchical semi-MDP $\mathcal{M} = (\mathcal{S}, \{\mathcal{A}^\ell\}_{\ell\in\mathcal{L}}, P, R, \gamma)$ with $\mathcal{L} = \{S, T, O\}$.

Assumptions.

- (A1) **Concavity with bounded curvature:** For each scale ℓ and state s, the quality function $Q^\ell(s, a^*; b)$ is twice continuously differentiable and strictly concave in b, with curvature bounded as $-L_\ell \le \frac{\partial^2 Q^\ell}{\partial b^2}(s, a^*; b) < 0$ for all $b \in [B_{\min}, B_{\max}]$, for some constant $L_\ell > 0$.
- (A2) **Heterogeneity:** There exists at least one scale ℓ such that $\mathrm{Var}_{s\sim d^\pi}[I^\ell(s)] > 0$, where $I^\ell(s) = \frac{\partial Q^\ell}{\partial b}(s, a^*; \bar{b})$ is the marginal information gain at the mean budget $\bar{b} = B_{\text{total}}/|\mathcal{L}|$.
- (A3) **Bounded budget:** The adaptive budget satisfies $\beta^*_\ell(s) \in [B_{\min}, B_{\max}]$ for all s, ℓ, ensuring the reallocation remains within feasible bounds.
- (A4) **Per-step accounting:** The allocated budget affects the expected reward collected at a decision point only through the quality $Q^\ell(s, a^*; b)$ of the selected action and does not change the state visitation distribution, which is stationary, i.e., $d^\pi_t = d^\pi$ for all t.

Step 1: Per-step improvement via second-order Taylor bound. Fix a scale ℓ and consider reallocating budget across states while maintaining the average. Applying a second-order Taylor expansion with remainder to $Q^\ell(s, a^*; b)$ around $\bar{b}$, for any state s:

$$Q^\ell(s, a^*; \bar{b} + \delta(s)) \ge Q^\ell(s, a^*; \bar{b}) + I^\ell(s)\delta(s) - \frac{L_\ell}{2}\delta(s)^2, \tag{10}$$

where $\delta(s) = \beta^*_\ell(s) - \bar{b}$ is the deviation from the uniform budget, and the inequality follows from the fact that $-L_\ell$ is a lower bound on the second derivative (A1). Taking expectations over $s \sim d^\pi$ with the budget-neutrality constraint $\mathbb{E}_{s\sim d^\pi}[\delta(s)] = 0$ (which ensures the total budget is preserved):

$$\mathbb{E}_s[Q^\ell(s, a^*; \beta^*_\ell(s))] - \mathbb{E}_s[Q^\ell(s, a^*; \bar{b})] \ge \mathbb{E}_s[I^\ell(s)\delta(s)] - \frac{L_\ell}{2}\mathbb{E}_s[\delta(s)^2]. \tag{11}$$

Note that $\mathbb{E}_s[I^\ell(s)\delta(s)] = \mathrm{Cov}(I^\ell(s), \delta(s))$ since $\mathbb{E}[\delta(s)] = 0$.

Step 2: Optimal reallocation via Lagrangian optimization. We seek $\delta^*(\cdot)$ that maximizes the RHS subject to $\mathbb{E}[\delta(s)] = 0$. Forming the Lagrangian $\mathcal{J}(\delta, \mu) = \mathbb{E}_s[I^\ell(s)\delta(s) - \frac{L_\ell}{2}\delta(s)^2] - \mu\mathbb{E}_s[\delta(s)]$ and taking the functional derivative with respect to $\delta(s)$:

$$\frac{\partial \mathcal{J}}{\partial \delta(s)} = I^\ell(s) - L_\ell\delta(s) - \mu = 0 \implies \delta^*(s) = \frac{I^\ell(s) - \mu}{L_\ell}. \tag{12}$$

The constraint $\mathbb{E}[\delta^*(s)] = 0$ gives $\mu = \mathbb{E}[I^\ell(s)] \equiv \mu_I$, so $\delta^*(s) = (I^\ell(s) - \mu_I)/L_\ell$. Substituting:

$$\begin{aligned}
\mathrm{Gain}_\ell &= \mathbb{E}_s\left[\frac{(I^\ell(s) - \mu_I)\cdot I^\ell(s)}{L_\ell}\right] - \frac{L_\ell}{2}\mathbb{E}_s\left[\frac{(I^\ell(s) - \mu_I)^2}{L_\ell^2}\right] \\
&= \frac{1}{L_\ell}\mathbb{E}_s[(I^\ell(s) - \mu_I)\cdot I^\ell(s)] - \frac{1}{2L_\ell}\mathrm{Var}[I^\ell(s)] \\
&= \frac{1}{L_\ell}\mathrm{Var}[I^\ell(s)] - \frac{1}{2L_\ell}\mathrm{Var}[I^\ell(s)] = \frac{\mathrm{Var}[I^\ell(s)]}{2L_\ell},
\end{aligned} \tag{13}$$

where the third equality uses $\mathbb{E}[(I^\ell(s)-\mu_I)\cdot I^\ell(s)] = \mathbb{E}[(I^\ell(s)-\mu_I)(I^\ell(s)-\mu_I+\mu_I)] = \mathrm{Var}[I^\ell(s)]$.

Step 3: Multi-scale aggregation over the discounted horizon. In the semi-MDP, the per-step gains accumulate over the trajectory. Let ν^ℓ denote the frequency with which scale ℓ is invoked (with $\sum_\ell \nu^\ell = 1$). By (A4), the expected reward improvement at a decision point is $\sum_\ell \nu^\ell \mathrm{Gain}_\ell$, so the expected discounted improvement is:

$$V^{\beta^*}(s_0) - V^{\beta^{\text{uni}}}(s_0) \ge \sum_{t=0}^{\infty}\gamma^t \sum_\ell \nu^\ell \cdot \frac{\mathrm{Var}_{s\sim d^\pi_t}[I^\ell(s)]}{2L_\ell}. \tag{14}$$

Under the stationarity in (A4), $d_t^\pi = d^\pi$ for all t, so the inner sum does not depend on t. We define the scale-dependent constant of Theorem 1 as $\kappa_\ell := \nu^\ell / L_\ell$, which grows with the invocation frequency of scale ℓ and decreases with its curvature bound. Since $\sum_{t\geq 0} \gamma^t = \frac{1}{1-\gamma} \geq \frac{\gamma}{1-\gamma}$ for $\gamma \in (0,1)$, this gives:

$$V^{\beta^*}(s_0) - V^{\beta^{\text{uni}}}(s_0) \geq \frac{1}{2(1-\gamma)} \sum_\ell \text{Var}_{s\sim d^\pi}[I^\ell(s)] \cdot \kappa_\ell \geq \frac{\gamma}{2(1-\gamma)} \sum_\ell \text{Var}_{s\sim d^\pi}[I^\ell(s)] \cdot \kappa_\ell, \quad (15)$$

which is Eq. (9). By assumption (A2), at least one variance term is positive, so the bound is strictly positive. □

Corollary (Finite-horizon regret). For undiscounted finite-horizon problems with horizon H, the same argument without discounting shows that the regret of the uniform policy relative to the optimal adaptive policy satisfies:

$$\text{Regret}(H) \geq \sum_{t=0}^{H-1} \sum_\ell \text{Var}_{s\sim d_t^\pi}[I^\ell(s)] \cdot \frac{\kappa_\ell}{2} = \Omega(H), \quad (16)$$

since the variance terms are constant in t under (A4). For bounded quality functions the regret is also $O(H)$, hence $\Theta(H)$, confirming that the benefit of adaptive allocation is non-diminishing and grows linearly with the task horizon, precisely the regime where MOSAIC is designed to operate.

A.2 Proof of Proposition 2 and Empirical Estimates of ρ

Theorem 1 assumes that the allocation is driven by the true information gain I^ℓ, which no scheduler observes. Proposition 2 replaces it by an estimate $\hat{I}^\ell$.

Proposition 2 (Imperfect information-gain estimates)**.** *Let $\hat{\beta}$ deviate from the uniform budget in proportion to $\hat{I}^\ell(s) - \mathbb{E}[\hat{I}^\ell]$ for an estimate $\hat{I}^\ell$ of I^ℓ, with an optimally chosen proportionality constant for each scale, and let $\rho_\ell := \text{corr}_{s\sim d^\pi}(I^\ell(s), \hat{I}^\ell(s))$. Under the conditions of Theorem 1,*

$$V^{\hat{\beta}}(s_0) - V^{\beta^{uni}}(s_0) \;\geq\; \frac{\gamma}{2(1-\gamma)} \sum_\ell \kappa_\ell \, \rho_\ell^2 \, Var_{s\sim d^\pi}[I^\ell(s)]. \quad (17)$$

The proof reuses the argument of Theorem 1 with $\hat{I}^\ell$ in the allocation and I^ℓ in the objective.

Proof of Proposition 2. Fix a scale ℓ. If $\hat{I}^\ell$ is constant on the support of d^π, we set $\rho_\ell = 0$; the plug-in policy then coincides with uniform allocation and the bound holds trivially. Otherwise, consider the plug-in deviation $\hat{\delta}(s) = c_\ell \, (\hat{I}^\ell(s) - \mathbb{E}[\hat{I}^\ell])$ for a scalar c_ℓ, which satisfies the budget-neutrality constraint $\mathbb{E}[\hat{\delta}(s)] = 0$ for every c_ℓ; as in (A3), we assume that the resulting budgets remain in $[B_{\min}, B_{\max}]$. Substituting $\hat{\delta}$ into the right-hand side of Eq. (11) gives

$$\mathbb{E}_s[I^\ell(s)\hat{\delta}(s)] - \frac{L_\ell}{2}\mathbb{E}_s[\hat{\delta}(s)^2] = c_\ell \, \text{Cov}(I^\ell, \hat{I}^\ell) - \frac{L_\ell}{2} \, c_\ell^2 \, \text{Var}[\hat{I}^\ell]. \quad (18)$$

This concave quadratic in c_ℓ is maximized by $c_\ell^* = \text{Cov}(I^\ell, \hat{I}^\ell)/(L_\ell \, \text{Var}[\hat{I}^\ell])$, which yields

$$\widehat{\text{Gain}}_\ell = \frac{\text{Cov}(I^\ell, \hat{I}^\ell)^2}{2L_\ell \, \text{Var}[\hat{I}^\ell]} = \rho_\ell^2 \, \frac{\text{Var}[I^\ell]}{2L_\ell} = \rho_\ell^2 \cdot \text{Gain}_\ell. \quad (19)$$

The plug-in allocation therefore attains exactly the fraction ρ_ℓ^2 of the oracle per-step gain in Eq. (13). Step 3 of the proof of Theorem 1 is linear in the per-step gains, so repeating it with $\widehat{\text{Gain}}_\ell$ in place of Gain_ℓ yields Eq. (17). □

Consequences. (i) *Weak estimates suffice.* Since $\rho_\ell^2 > 0$ whenever $\rho_\ell \neq 0$, any estimate that is correlated with the true information gain yields a strictly positive improvement under (A2); the guarantee therefore survives distribution shift as long as the scheduler features remain weakly informative. (ii) *Measurable degradation.* The guarantee decays as ρ_ℓ^2, so the cost of a distribution shift can be quantified by measuring ρ_ℓ. (iii) *Shrinkage toward the uniform budget.* Treating $\hat{I}^\ell$ as exact, i.e., using $c_\ell = 1/L_\ell$, yields the per-step gain $\big(2\,\text{Cov}(I^\ell, \hat{I}^\ell) - \text{Var}[\hat{I}^\ell]\big)/(2L_\ell)$,

which becomes negative once the estimate is noisy enough that $\mathrm{Var}[\hat{I}^\ell] > 2\,\mathrm{Cov}(I^\ell, \hat{I}^\ell)$. The optimal scale $c_\ell^* = \frac{1}{L_\ell} \cdot \mathrm{Cov}(I^\ell, \hat{I}^\ell)/\mathrm{Var}[\hat{I}^\ell]$ instead shrinks the deviations from the uniform budget; for an additive noise model $\hat{I}^\ell = I^\ell + \epsilon$ with ϵ independent of I^ℓ, the shrinkage factor equals $\mathrm{Var}[I^\ell]/(\mathrm{Var}[I^\ell] + \mathrm{Var}[\epsilon]) = \rho_\ell^2$. Limiting how far the scheduler can move each budget away from its default, which MOSAIC implements by bounding budget overrides to $\pm 50\%$, is a simple form of this shrinkage.

Empirical estimates of ρ. We sample 400 decision points and re-run each of them at budgets of 256, 512, 1,024, 2,048, and 4,096 tokens, which yields a finite-difference estimate of the realized information gain at that point; ρ is the correlation between this estimate and the budget allocated by the scheduler. Table 5 reports ρ, the fraction ρ^2 of the oracle gain guaranteed by Proposition 2, and the observed gain as a fraction of the oracle gain. The bound is conservative, since the observed fraction exceeds ρ^2 in every setting, but it orders the settings correctly, and ρ stays clearly positive out of distribution, which is all that Proposition 2 requires.

Table 5: Measured correlation ρ between allocated budget and realized information gain.

Scheduler / setting	ρ	ρ^2	**Observed / oracle gain**
Learned, in distribution (SWE-Bench Verified)	0.58	0.34	0.41
Learned, LOBO (SWE-Bench Verified held out)	0.46	0.21	0.26
Learned, zero-shot out of distribution (OSWorld)	0.37	0.14	0.18
Best rule-based scheduler	0.31	0.10	0.13
Fixed schedule	0.04	0.00	0.01

B Implementation Details

B.1 Scale Scheduler Training

The scheduler is trained using PPO with the hyperparameters listed in Table 6. The training set consists of 200 tasks sampled from the training splits of each benchmark (50 per benchmark), with no overlap with the evaluation sets; the exact task IDs of these 200 tasks and of every evaluation split are included in our code release. Training takes approximately 18 hours on 4× NVIDIA A100 80GB GPUs. The sensitivity of the scheduler to the size of the training set, the PPO seed, and the reward design is analyzed in Appendix E.4 and Appendix E.5.

Table 6: PPO hyperparameters for Scale Scheduler training.

Hyperparameter	**Value**
Learning rate	3×10^{-4}
Clip ratio	0.2
GAE λ	0.95
Discount γ	0.99
Batch size	64
Number of update steps	5,000
Number of rollouts per step	8
Entropy bonus coefficient	0.01
Value function loss coefficient	0.5
Max gradient norm	0.5
Scheduler MLP hidden dim	256
Scheduler MLP layers	2
Scheduler total parameters	68K

The task reward r_t^{task} is defined per benchmark to provide meaningful step-level feedback:

- **SWE-Bench Verified**: delta in test pass rate after each action. Specifically, we periodically run the test suite (every 10 steps) and compute the change in the number of passing tests relative to the previous checkpoint.

- **BrowseComp**: sparse reward of 1.0 on correct final answer, 0 otherwise. To mitigate reward sparsity, we add a small shaping reward of 0.05 when the agent visits a new domain that was not previously explored.
- **WebArena**: functional correctness score provided by the benchmark's evaluation harness, assessed at task completion.
- **GAIA**: exact match on the final answer, with partial credit (0.5) for answers that match the correct type and format.

B.2 Backbone Model Configuration

All backbone models are served using vLLM with the following inference parameters: temperature 0 (greedy decoding), top-p = 1.0, and maximum generation length determined by the active scale's budget b_t. We use KV-cache for the operational layer's sliding window to avoid redundant computation. For MoE models (Qwen3.5-35B-A3B and Gemma-4-27B-A4B), we use tensor parallelism across 2 GPUs. The dense model (Qwen3.5-9B) runs on a single GPU. All three backbones are open-weight models served with a pinned vLLM version, and no closed API is used anywhere in our pipeline.

B.3 Inter-Scale Context Distillation Prompts

We provide the complete prompt templates used for the Distill↑ and Refine↓ operators.

Distill↑ (Operational → Tactical):

```
You are summarizing the recent execution steps for a tactical
planner. Given the operational log below, produce a brief
structured summary (<150 tokens) with the following fields:
- Actions taken: List of actions executed
- Results: What was observed or achieved
- Blockers: Any errors or unexpected outcomes

Operational log:
{operational_log}
```

Distill↑ (Tactical → Strategic):

```
You are reporting subtask results to a strategic planner. Given the
tactical execution log for the subtask ''{subtask_name}'', produce a
structured report (<200 tokens) with:
- Outcome: success / partial / failure
- Key findings: Important information discovered
- Unexpected events: Anything that deviates from the plan
- Resource status: Tools used, files modified, search queries
exhausted

Tactical log:
{tactical_log}
```

Refine↓ (Strategic → Tactical):

```
You are a tactical planner. Given the strategic plan and current
context, generate a detailed tactical plan for the next subtask.

Strategic plan: {strategic_plan}
Current subtask: {current_subtask}
Available tools: {tool_inventory}
Recent context: {recent_context}

Output a step-by-step plan with:
1. Specific tool/action selections for each step
2. The expected result of each step
3. Expected intermediate checkpoints
4. Fallback actions if a step fails
Keep the plan to 5-10 steps.
```

The expected result of each step is the string $\hat{y}_t$ that the scheduler compares with the actual result when computing e_{diverge} (Appendix B.5).

Refine↓ (Tactical → Operational): The tactical plan is directly passed as the system instruction for the operational layer, prepended to the sliding-window context. No additional LLM call is needed for this direction.

Constraint-preserving check. To limit information loss with weaker backbones, Distill↑ and Refine↓ can be combined with a cheap constraint-preserving check: a schema assertion that every constraint node of $\mathcal{G}_t$ and every `unexpected_events` entry is carried forward, together with a one-line self-check instruction appended to the prompts. Its effect is analyzed in Appendix K. The check is enabled by default in our released code; all other results in this paper are reported without it.

B.4 Task State Graph Construction and Maintenance

The task state graph $\mathcal{G}_t = (V_t, E_t)$ is initialized from the task description using a single LLM call with the following prompt:

```
Analyze the following task and decompose it into a dependency graph
of subgoals.

Task: {task_description}

Output a JSON object with:
- "nodes": list of {id, description, type:
goal/subgoal/constraint, status: pending}
- "edges": list of {from, to, relation: depends_on/blocks}
```

Each node thus carries a type in {goal, subgoal, constraint} and a status in {pending, active, done, failed}, and each edge carries a relation in {`depends_on`, `blocks`}.

Update rule. The graph is never regenerated. It changes only at strategic invocations, only from the Distill↑ report, and only through three deterministic operations: (i) setting the status of the node whose id is explicitly named in the report, where matching is by id and never by fuzzy text similarity; (ii) appending one constraint node per `unexpected_events` entry; and (iii) marking the dependent edges of a subgoal for re-evaluation by the strategic layer when that subgoal is failed. A goal or subgoal is marked done only when the report names its id and, where a programmatic check exists (a test exit code on SWE-Bench, a DOM predicate on WebArena), the check passes. Nodes are never deleted or rewritten, so the graph cannot oscillate.

Validation. Each candidate update is schema-validated and cycle-checked. On failure, we retry once and otherwise keep the previous graph, so $\mathcal{G}_t$ can become stale but never corrupt. The fallback rate is 1.8%, 3.1%, 2.4%, and 2.9% on SWE-Bench Verified, BrowseComp, WebArena, and GAIA, respectively, and the retry succeeds in 82% of cases.

Stability. Across 3 seeds on the same task, the normalized graph edit distance is 0.11, 0.14, 0.09, and 0.13 (SWE-Bench Verified, BrowseComp, WebArena, GAIA), and the graphs remain small, with average node counts ranging from 6.8 ± 1.9 to 9.4 ± 2.6 across benchmarks. Against human-annotated progress on 50 tasks per benchmark, the graph-derived e_{progress} reaches Spearman correlations of 0.71, 0.64, 0.68, and 0.66 on the same four benchmarks.

Robustness to noisy observations. Corrupting 20% of the observations (truncation plus token noise) changes the SWE-Bench Verified resolve rate from 37.6% to 36.5%, the fallback rate from 1.8% to 4.6%, and the graph edit distance from 0.11 to 0.19. The degradation is mild because a stale graph only perturbs e_{progress}, the second least important scheduler feature (Appendix J.1).

B.5 Scheduler Feature Specification

All five features are computed by the same procedure on every benchmark (Algorithm 2); only the environment-specific inputs, namely what the goal nodes of $\mathcal{G}_t$ refer to, what counts as the actual

Algorithm 2 EXTRACTFEATURES

Require: Task graph $\mathcal{G}_t$, active tactical plan p^T (possibly none), expected result $\hat{y}_t$ of the current tactical step, actual result y_t of the last action, token distributions over the action-name span of the operational output, status of the last tool call, step t, frozen sentence encoder E

$N \leftarrow$ goal and subgoal nodes of $\mathcal{G}_t$ (constraint nodes excluded; normalized by the node count at step t, since $\mathcal{G}_t$ is append-only)
if no node of $\mathcal{G}_t$ matches **then**
  $e_{\text{progress}} \leftarrow t/T_{\max}$
else
  $e_{\text{progress}} \leftarrow |\{v \in N : \text{status}(v) = \text{done}\}| \,/\, |N|$
end if
if a tactical plan p^T is active **then**
  $e_{\text{diverge}} \leftarrow 1 - \cos\big(E(\hat{y}_t), E(y_t)\big)$
  $e_{\text{subtask}} \leftarrow 1 - (\text{remaining checkpoints of } p^T)/(\text{total checkpoints of } p^T)$
else
  $e_{\text{diverge}} \leftarrow 0$; $e_{\text{subtask}} \leftarrow 0$
end if
$e_{\text{entropy}} \leftarrow$ mean token entropy over the action-name span, divided by $\log |V|$
$e_{\text{error}} \leftarrow 1$ if the last tool call failed (Table 7), else 0
return $\phi_t = [e_{\text{progress}}, e_{\text{diverge}}, e_{\text{entropy}}, e_{\text{error}}, e_{\text{subtask}}]$

Table 7: Per-benchmark instantiation of the environment-specific inputs of the scheduler features.

Input	SWE-Bench Verified	BrowseComp	WebArena	GAIA
Goal nodes (e_{progress})	Files or functions to inspect or edit, plus tests to pass	Information items to retrieve	Required UI state changes (e.g., item in cart)	Intermediate quantities the question requires
Actual result y_t (e_{diverge})	stdout/stderr plus patch result	Retrieved snippet	Post-action DOM summary	Tool return
Failed call ($e_{\text{error}} = 1$)	Non-zero exit code, traceback, or patch-apply failure	HTTP 4xx/5xx, empty result set, or rate limit	Invalid selector, disallowed action, or navigation failure	Tool exception or parse failure

result y_t of an action, and what counts as a failed tool call, are instantiated per benchmark, as listed in Table 7.

e_{entropy} and e_{subtask} need no per-benchmark instantiation: the former is read off the existing operational forward pass at no extra cost, and the latter relies on the explicit checkpoint list that every tactical plan carries (§3.1). e_{progress} and e_{subtask} are computed from existing structures, so only e_{diverge} requires computation: a frozen 22M-parameter sentence encoder, at 1.9 ms per step and without any LLM call. The fallbacks of Algorithm 2 fire on 6.2% (e_{progress}) and 4.8% (no active tactical plan) of steps, and the noise-injection results of Appendix J.2 bound their cost.

B.6 Evaluation Parity and Reproducibility

Parity checklist. All methods, including MOSAIC, share: (1) identical harness and tool implementations; (2) the same retry policy (at most 2 retries per failed call); (3) the same step caps; (4) the same decoding parameters (temperature 0, top-p 1.0); (5) a single token counter at the vLLM boundary, which counts both input and output tokens and instruments summarization, retries, and failed calls identically for every method; (6) for BrowseComp and GAIA, the same fixed cached web snapshot, which replaces live web access so that results are immune to web drift; and (7) the same 3 environment seeds. Baselines were allowed strictly larger inference budgets than MOSAIC in every table.

Adaptation budget. The scheduler's training cost (approximately 72 A100 GPU-hours, i.e., 18 hours on 4× A100 GPUs; Appendix B.1) is incurred once. In the matched-adaptation comparison (Appendix I), every baseline receives the same 200 training tasks and the same compute for tuning or retraining.

Seeds. All tables report 3 environment seeds; the scheduler itself is trained with 5 PPO seeds, analyzed in Appendix E.4.

Released artifacts. We release, at https://github.com/BokwaiHo/MOSAIC, the code and configurations; scheduler checkpoints for all 5 PPO seeds, the 4 leave-one-benchmark-out checkpoints, and the configurations of both unseen-environment evaluations; the exact task ID lists for the 200 training tasks and every evaluation split; PPO training logs; and the raw trajectories behind every reported number.

C Per-Benchmark Detailed Results

C.1 SWE-Bench Verified by Issue Complexity

Table 8 provides a breakdown of SWE-Bench Verified results by issue complexity, categorized by the number of files that must be modified to resolve the issue.

Table 8: SWE-Bench Verified resolve rate (%) by issue complexity (Qwen3.5-35B-A3B).

Method	Single-file	Multi-file	Cross-module
ReAct	38.2$_{\pm 1.5}$	18.7$_{\pm 1.8}$	9.1$_{\pm 2.3}$
Plan-and-Act	40.5$_{\pm 1.3}$	21.3$_{\pm 1.6}$	12.4$_{\pm 2.0}$
Context-Folding	43.1$_{\pm 1.2}$	24.5$_{\pm 1.4}$	15.3$_{\pm 2.1}$
MOSAIC	**45.8**$_{\pm 1.0}$	**29.3**$_{\pm 1.3}$	**19.7**$_{\pm 1.8}$

The advantage of MOSAIC over Context-Folding grows with issue complexity: +2.7 on single-file, +4.8 on multi-file, and +4.4 on cross-module issues. This confirms that the strategic layer's global coherence is most valuable when the agent must reason about dependencies across distant parts of a codebase.

C.2 GAIA Results by Level

Table 9 breaks down GAIA results by difficulty level. MOSAIC shows the largest improvement on Level 3 tasks, which require the longest reasoning chains and most diverse tool usage.

Table 9: GAIA accuracy (%) by difficulty level (Qwen3.5-35B-A3B).

Method	Level 1	Level 2	Level 3
ReAct	58.3$_{\pm 2.0}$	41.2$_{\pm 2.4}$	15.6$_{\pm 3.1}$
Reflexion	62.1$_{\pm 1.8}$	47.3$_{\pm 2.2}$	22.8$_{\pm 2.9}$
MOSAIC	**64.5**$_{\pm 1.5}$	**50.1**$_{\pm 2.0}$	**28.4**$_{\pm 2.6}$

C.3 BrowseComp Results by Question Category

Table 10 shows BrowseComp accuracy broken down by question category. MOSAIC particularly excels on "Science" and "History" questions, which tend to require the deepest multi-hop reasoning and the most diverse search strategies.

Table 10: BrowseComp accuracy (%) by question category (Qwen3.5-35B-A3B).

Method	Science	History	Sports	Entertainment
ReAct	9.4$_{\pm 1.2}$	11.2$_{\pm 1.0}$	16.8$_{\pm 1.5}$	14.7$_{\pm 1.3}$
Context-Folding	14.1$_{\pm 1.0}$	16.3$_{\pm 0.9}$	20.5$_{\pm 1.3}$	18.9$_{\pm 1.1}$
MOSAIC	**17.8**$_{\pm 1.1}$	**19.1**$_{\pm 0.8}$	**22.3**$_{\pm 1.2}$	**20.1**$_{\pm 1.0}$

C.4 Computational Cost

Table 11 reports the average wall-clock time per task across benchmarks.

Table 11: Average wall-clock time per task (minutes) using Qwen3.5-35B-A3B on 2× A100 GPUs.

	SWE-Bench	BrowseComp	WebArena	GAIA L2&3
ReAct	8.2	12.5	3.1	5.4
Plan-and-Act	9.1	11.8	3.5	5.9
Context-Folding	6.4	9.7	2.6	4.5
MOSAIC	**5.7**	**8.3**	**2.0**	**3.8**

MOSAIC is consistently the fastest method despite its multi-scale architecture. All token counts and wall-clock times in this paper are end to end. Table 12 decomposes the per-task cost on SWE-Bench Verified, which sums to the 298K tokens and 5.7 minutes reported in Tables 1 and 11. The Scale Scheduler itself is nearly free: e_{entropy} is read off the operational forward pass, e_{progress} and e_{subtask} come from existing structures, and only e_{diverge} requires computation (a frozen 22M-parameter sentence encoder, 1.9 ms per step, without any LLM call), while the scheduler MLP forward pass takes less than 1 ms. The summary$(\cdot)$, Distill$\uparrow$, and Refine$\downarrow$ calls together account for 23.1% of the tokens, an overhead that is more than offset by the token savings from the operational layer's compact context window and small reasoning budget. Input and output tokens amount to 214K and 84K for MOSAIC, versus 371K and 116K for ReAct, and the retries, failed calls, and summarization of all baselines are counted identically. Compared to ReAct, MOSAIC achieves a 30–36% wall-clock speedup across benchmarks.

Table 12: Per-task cost breakdown of MOSAIC on SWE-Bench Verified (Qwen3.5-35B-A3B).

Component	**Tokens (K)**	**Share**	**Time (min)**
Operational layer generation	141	47.3%	2.4
Tactical layer generation	58	19.5%	1.0
Strategic layer generation	21	7.0%	0.5
summary$(\cdot)$ trajectory compression	26	8.7%	0.5
Distill$\uparrow$ (operational $\rightarrow$ tactical)	22	7.4%	0.4
Distill$\uparrow$ (tactical $\rightarrow$ strategic)	9	3.0%	0.2
Refine$\downarrow$ (strategic $\rightarrow$ tactical)	12	4.0%	0.3
Failed tool calls and retries	9	3.0%	0.4
Scheduler features (frozen 22M encoder)	0.4 (equiv.)	0.1%	0.02
Scheduler forward pass (68K MLP)	0	0%	$<$0.001
Total	298	100%	5.7

D Case Studies

We present two case studies illustrating how MOSAIC's multi-scale architecture and adaptive scheduling lead to qualitatively different behavior from flat baselines.

D.1 Case Study 1: SWE-Bench Cross-Module Bug Fix

Task. Resolve a GitHub issue in `scikit-learn` where the `ColumnTransformer` raises an incorrect error when a transformer returns a sparse matrix with mismatched feature names.

ReAct behavior (failed, 187 steps, 487K tokens). The flat agent begins by searching for the error message, identifies the relevant file (`compose/_column_transformer.py`), and attempts a local fix. However, the fix breaks a downstream test in `test_column_transformer.py`. The agent then enters a debugging loop: reverting the fix, trying another approach, breaking a different test, reverting again, for 120 steps. It never steps back to consider the broader module architecture or check whether the sparse matrix handling is consistent across the `compose` module. The agent exhausts its step budget without a correct patch.

MOSAIC behavior (succeeded, 142 steps, 267K tokens).

- **Step 0 [Strategic, 3,841 tokens]:** Decomposes the task into three subgoals: (1) understand the `ColumnTransformer`'s sparse matrix path, (2) identify the root cause of the feature-name mismatch, (3) apply a fix and verify.
- **Steps 1–18 [Operational, avg 478 tokens]:** Navigates the codebase, opens relevant files. The scheduler allocates minimal budget as these are routine navigation steps.
- **Step 19 [Tactical, 1,924 tokens]:** After reading the source, the tactical layer identifies that the bug originates in a helper function `_validate_output` that is shared across three files. It generates a plan to check all call sites.
- **Steps 20–45 [Operational]:** Reads the three call sites, runs existing tests.
- **Step 46 [Strategic, 3,200 tokens]:** The scheduler detects high plan divergence (the agent found that the helper function has different behavior paths for dense vs. sparse inputs, which was not anticipated in the initial decomposition). The strategic layer revises the plan: the fix must modify `_validate_output` and update two callers.
- **Steps 47–82 [Mixed]:** Implements the fix with tactical guidance, running tests after each modification.
- **Step 83 [Tactical, 1,756 tokens]:** After a test failure, the tactical layer diagnoses the issue (a missing `.toarray()` call) and replans.
- **Steps 84–142 [Operational]:** Applies the remaining fix, all tests pass.

Key insight: The two strategic invocations (steps 0 and 46) together consumed only 7,041 tokens ($<$3% of total) but were critical for maintaining global coherence. ReAct's failure was not due to insufficient reasoning per step, but to the absence of a mechanism for stepping back and revising the overall approach.

D.2 Case Study 2: BrowseComp Multi-Hop Research

Task. "Name the only author who has published papers in both the 2019 NeurIPS workshop on Bayesian Deep Learning and the 2021 ICML workshop on Uncertainty in Deep Learning, and who also holds a patent filed before 2018 related to variational inference."

ReAct behavior (failed, 156 steps, 892K tokens). The agent searches for "NeurIPS 2019 Bayesian Deep Learning workshop authors" and retrieves a list of 30+ authors. It then searches each author individually for the ICML 2021 workshop, consuming enormous tokens on sequential searches. After 80 steps of brute-force enumeration, the context window fills with search results, and the agent begins to hallucinate author names from earlier context that have been partially truncated.

MOSAIC behavior (succeeded, 98 steps, 423K tokens).

- **Step 0 [Strategic]:** Decomposes into: (1) find the intersection of two author sets, (2) filter by patent, (3) verify the answer is unique.
- **Step 3 [Tactical]:** Plans a more efficient search strategy: instead of enumerating authors from workshop A and checking workshop B, search for researchers who are known to work on both Bayesian deep learning and uncertainty quantification, narrowing the candidate pool.
- **Steps 4–35 [Operational]:** Executes targeted searches, identifies 5 candidate authors.
- **Step 36 [Tactical]:** Pivots to patent search for each candidate, using Google Patents.
- **Steps 37–72 [Operational]:** Searches patents, eliminates 3 candidates.
- **Step 73 [Tactical]:** Verifies the remaining 2 candidates against both workshop programs.
- **Steps 74–98:** Confirms the unique answer.

Key insight: The tactical layer's search strategy (step 3) avoided the $O(n)$ brute-force enumeration that trapped ReAct. By reasoning about the search space *before* executing searches, MOSAIC reduced the number of search actions by 37% while maintaining accuracy.

E Sensitivity Analysis

E.1 Sensitivity to λ

Table 13 reports MOSAIC's performance across a range of λ values on SWE-Bench Verified. The method is robust across a wide range, with $\lambda \in [0.01, 0.2]$ all yielding strong results.

Table 13: Sensitivity to efficiency penalty λ on SWE-Bench Verified (Qwen3.5-35B-A3B).

λ	**Resolve Rate (%)**	**Tokens (K)**	**Strategic Invoc. (%)**
0.01	$38.9_{\pm 1.0}$	412	8.2
0.05	$38.1_{\pm 0.8}$	345	6.7
0.10	$37.6_{\pm 0.9}$	298	5.4
0.20	$35.8_{\pm 1.1}$	215	3.8
0.50	$32.1_{\pm 1.3}$	178	2.1

As λ increases, the scheduler reduces strategic invocations first (from 8.2% to 2.1% of steps), reflecting the strategic layer's high per-invocation cost. Performance remains within 3.1% of the best setting across $\lambda \in [0.01, 0.2]$, degrading more sharply only at $\lambda = 0.5$ where strategic reasoning is almost entirely suppressed.

E.2 Sensitivity to Number of Scales

Table 14 evaluates MOSAIC with different numbers of hierarchy levels.

Table 14: Effect of the number of scales on SWE-Bench Verified (Qwen3.5-35B-A3B).

Configuration	**Resolve Rate (%)**	**Tokens (K)**	η
1-scale (flat, = ReAct)	$28.4_{\pm 1.2}$	487	5.21
2-scale (Tactical + Operational)	$35.8_{\pm 0.8}$	305	7.08
3-scale (MOSAIC, default)	$\mathbf{37.6}_{\pm 0.9}$	**298**	**8.84**
4-scale (+Meta-Strategic)	$37.9_{\pm 1.0}$	312	8.63

The largest gain comes from moving to 2 scales (+7.4% over flat). The third scale adds a further +1.8% with minimal token overhead. A fourth scale ("Meta-Strategic", responsible for revising the overall task framing) provides negligible improvement (+0.3%) while increasing tokens slightly, suggesting that three scales offer the best complexity–performance trade-off for the benchmarks considered.

E.3 Sensitivity to Context Window Sizes

Table 15 varies the operational and tactical window sizes.

Table 15: Effect of context window sizes on SWE-Bench Verified (Qwen3.5-35B-A3B).

w_O	w_T	**Resolve Rate (%)**	**Tokens (K)**
3	10	$35.9_{\pm 1.1}$	245
5	20	$\mathbf{37.6}_{\pm 0.9}$	298
5	30	$37.4_{\pm 0.8}$	335
10	20	$37.1_{\pm 1.0}$	342

The default $w_O = 5, w_T = 20$ provides the best trade-off. Reducing w_O to 3 saves tokens but sacrifices performance (-1.7%), as the operational layer occasionally lacks sufficient context to choose the correct action. Increasing w_T to 30 or w_O to 10 increases token usage without meaningful performance gains, confirming that the inter-scale distillation mechanism effectively compensates for the limited local context.

E.4 Sensitivity to Scheduler Training-Set Size and PPO Seed

Training-set size. Table 16 varies the number of training tasks of the scheduler. Performance on SWE-Bench Verified saturates near 200 tasks, and 100 tasks already recover 92% of the gain over ReAct.

Table 16: Effect of the number of scheduler training tasks on SWE-Bench Verified (Qwen3.5-35B-A3B).

Training tasks	25	50	100	200 (default)	400
Resolve Rate (%)	34.9	35.8	36.9	37.6	37.8

Table 17: Distribution of escalation triggers on SWE-Bench Verified.

Trigger Event	**→ Tactical (%)**	**→ Strategic (%)**
Subtask boundary reached	42.1	5.3
Tool execution error	28.7	12.8
High action entropy (>1.5 nats)	15.4	3.1
Plan divergence (>0.4 cosine dist.)	8.9	61.2
Task progress stall (>10 steps w/o progress)	4.9	17.6

PPO seed. Across 5 PPO seeds, the resolve rate on SWE-Bench Verified is 37.6±0.7% (range 36.6% to 38.4%). The learned behavior is stable as well: the strategic invocation rate is 5.4±0.6% of steps, and each entry of Table 17 varies by at most 4.1 points across seeds.

E.5 Sensitivity to Reward Design

Table 18 compares alternative scheduler rewards on SWE-Bench Verified. Dense step-level task rewards matter: training with a terminal-only reward or a binary success reward lowers the resolve rate by 1.4 and 2.0 points, respectively. Removing the efficiency penalty ($\lambda = 0$) increases token usage by 60% for a 0.6-point change in resolve rate, and adding a graph-progress term to the composite reward changes little. On BrowseComp, removing the new-domain shaping reward (Appendix B.1) lowers accuracy from 19.2% to 17.9%, so shaping matters most where the reward is sparsest.

Table 18: Effect of the scheduler reward on SWE-Bench Verified (Qwen3.5-35B-A3B).

Reward	**Resolve Rate (%)**	**Tokens (K)**
Composite, Eq. (8) (default)	37.6	298
Terminal-only task reward	36.2	305
Binary success only	35.6	312
No efficiency penalty ($\lambda = 0$)	38.2	476
Composite plus graph-progress term	37.7	296

E.6 Distribution Shift in Task Horizon

To probe distribution shift in horizon length, we train the scheduler only on tasks shorter than 100 steps and evaluate it on SWE-Bench Verified by test-horizon bucket (Table 19). This is the sharpest shift we found: short-horizon training costs 1.8 points on tasks of 100–200 steps and 3.2 points on tasks longer than 200 steps, although it still clearly improves over ReAct (24.3% vs. 15.4% and 14.6% vs. 7.9%). We therefore recommend including long-horizon tasks in scheduler training (Appendix M).

F Scale Scheduler Learned Behaviors

To better understand what the Scale Scheduler learns, we analyze the correlation between its escalation decisions and observable task events.

Escalation triggers. We classify each strategic or tactical invocation by the event that preceded it. Table 17 shows the distribution of triggers on SWE-Bench Verified. Action entropy is measured here in nats, before the normalization by $\log |V|$ applied to the scheduler input.

Table 19: Resolve rate (%) on SWE-Bench Verified by test-horizon bucket when the scheduler is trained on all horizons or only on tasks shorter than 100 steps (Qwen3.5-35B-A3B).

Test horizon	ReAct	MOSAIC (all horizons)	MOSAIC (short horizons)
$<$ 50 steps	41.3	46.9	46.5
50–100 steps	27.8	37.2	36.4
100–200 steps	15.4	26.1	24.3
$>$ 200 steps	7.9	17.8	14.6

The scheduler learns a clear separation: tactical escalation is primarily driven by subtask boundaries and errors (routine replanning), while strategic escalation is overwhelmingly triggered by plan divergence (61.2%), indicating that the strategic layer is reserved for situations where the global plan is no longer valid. This aligns with the cognitive science intuition that humans escalate to deliberate System 2 thinking primarily when their automatic predictions fail. These trigger statistics are stable across PPO seeds (Appendix E.4).

G Scheduler Generalization

The Scale Scheduler of the main experiments is trained on 200 tasks drawn from the four evaluation benchmark families. This appendix quantifies how much of its benefit depends on in-distribution data, by holding out each evaluation benchmark from scheduler training (Appendix G.1) and by transferring the scheduler to benchmark families with unseen tools (Appendix G.2). Appendix H and Appendix I complement this analysis with non-learned schedulers and with baselines that receive the same adaptation budget. All results use Qwen3.5-35B-A3B and 3 environment seeds.

G.1 Leave-One-Benchmark-Out Transfer

For each evaluation benchmark, we retrain the scheduler from scratch on the 150 training tasks of the other three benchmarks, with the hyperparameters of Table 6, and apply it zero-shot to the held-out benchmark. The results are reported in the top part of Table 3. The LOBO scheduler retains 87%, 86%, 86%, and 85% of MOSAIC's gain over ReAct on SWE-Bench Verified, BrowseComp, WebArena, and GAIA, where the retained share is $(\text{LOBO} - \text{ReAct})/(\text{MOSAIC} - \text{ReAct})$, so the scheduler does not mainly learn benchmark-specific escalation patterns. It outperforms the strongest published baseline of Table 1 on all four benchmarks while using fewer tokens, without any data from the target distribution. Very light adaptation, with 10 target tasks and 500 PPO steps (about 25 minutes on one A100), yields 37.3%, 19.0%, 32.8%, and 46.0%, within 0.3 points of the in-distribution scheduler on every benchmark. Deployment in a new domain therefore requires either no target tasks (the LOBO scheduler, or the rule-based fallback of Appendix H) or roughly ten, rather than a benchmark-scale training split. A scheduler trained on a single benchmark transfers less well: in preliminary experiments, a scheduler trained only on SWE-Bench tasks showed a 2–3% performance drop when applied to BrowseComp without fine-tuning, which suggests training the scheduler on a mixture of task families.

G.2 Transfer to Unseen Tool Environments

We further evaluate the scheduler trained on the original 200 tasks, zero-shot, on two benchmark families chosen for their different action spaces: OSWorld [Xie et al., 2024] (369 tasks; GUI actions, screenshot observations, desktop applications and a shell) and τ-bench [Yao et al., 2025] (retail and airline domains; stateful transactional API writes and a live user simulator). Neither shares a tool with scheduler training, and τ-bench adds a modality absent from all four training benchmarks: a dialogue partner whose replies enter the observation stream. The bottom part of Table 3 compares MOSAIC with ReAct, the fixed schedule, and the best rule-based scheduler. Zero-shot, MOSAIC achieves the highest success rate with the fewest tokens among these methods on both families, and adapting it with 10 target tasks raises the success rate further, to 16.9% on OSWorld and 40.3% on τ-bench.

Table 20: Learned versus non-learned schedulers on SWE-Bench Verified (Qwen3.5-35B-A3B).

Scheduler	Resolve Rate (%)	Tokens (K)	Share of learned gain
Fixed schedule (strategic every 50 steps, tactical every 10)	$34.1_{\pm 1.0}$	321	0%
Rule: escalate on tool error	$34.8_{\pm 1.1}$	316	20%
Rule: 5-feature thresholds, one global threshold set	$34.9_{\pm 1.2}$	314	23%
Rule: 5-feature thresholds, tuned per benchmark	$35.9_{\pm 1.0}$	309	51%
Learned (MOSAIC)	$\mathbf{37.6}_{\pm 0.9}$	**298**	100%
Offline oracle scheduler (upper bound)	$39.8_{\pm 0.8}$	291	163%

Table 21: Performance of the baselines as published and after matched adaptation (Qwen3.5-35B-A3B).

	As published				With matched adaptation			
Method	SWE-Bench V.	BrowseComp	WebArena	GAIA L2&3	SWE-Bench V.	BrowseComp	WebArena	GAIA L2&3
ReAct	28.4	12.8	24.7	38.2	30.1	13.9	26.3	39.8
Plan-and-Act	32.1	15.3	28.5	42.6	33.4	16.2	29.9	43.9
GoalAct	33.8	16.1	30.2	43.8	35.0	17.0	31.4	44.9
Context-Folding	35.2	17.4	29.8	41.5	36.1	18.1	30.9	42.6
Reflexion	31.6	14.1	31.3	44.1	32.9	15.0	32.5	45.2
MOSAIC (200-task scheduler)	**37.6**	**19.2**	**33.1**	**46.3**	(as reported in Table 1)			
MOSAIC (LOBO, no target-benchmark data)	36.4	18.3	31.9	45.1	(no target-benchmark data)			

The degradation out of distribution is graceful rather than erratic. The scheduler becomes more conservative, raising the share of strategic invocations from 5.4% to 7.9% of steps, and its token savings over ReAct shrink only from 39% on SWE-Bench Verified to 36% on OSWorld. Consistent with Proposition 2, the correlation between allocated budget and realized information gain stays clearly positive on OSWorld ($\rho = 0.37$, Table 5).

H Non-Learned Schedulers

Table 20 compares the learned scheduler with non-learned alternatives on SWE-Bench Verified. The rule-based schedulers threshold the same five features as the learned one, either with a single global threshold set or with thresholds tuned separately for each benchmark; the share column reports the fraction of the learned scheduler's gain over the fixed schedule.

Two conclusions follow. First, the global-threshold rule is a legitimate zero-setup fallback: it requires no training, yet it outperforms both the fixed schedule and ReAct, so we release it as the default configuration for practitioners who cannot train a scheduler. Second, the rule that comes closest to the learned scheduler (51% of its gain) needs per-benchmark threshold tuning, which is itself a form of distribution-specific adaptation. The appropriate comparison is therefore between the LOBO scheduler (36.4%, no target data) and the tuned rule (35.9%, target data used): the learned scheduler is both stronger and less dependent on access to the target distribution. The gap to the offline oracle indicates that better information-gain estimates could improve the scheduler further.

I Baseline Parity with Matched Adaptation

In the main comparison, MOSAIC receives benchmark-family-specific adaptation through scheduler training, whereas the baselines are used as published. To remove this asymmetry, we give every baseline the same 200 training tasks and the same compute that scheduler training consumes (Appendix B.6): prompt and hyperparameter search over 12 configurations for ReAct, GoalAct, and Reflexion, planner retraining for Plan-and-Act, and RL retraining of the context policy for Context-Folding. Table 21 reports the results; the token counts of the baselines change by less than 4% under matched adaptation.

Matched adaptation lifts the baselines by 1.0–1.5 points on average, a real effect that should be accounted for when comparing against learned components. MOSAIC nevertheless remains the best method on every benchmark while using fewer tokens: its margin over the strongest tuned baseline is 1.5 points on SWE-Bench Verified (over Context-Folding), 1.1 on BrowseComp (over Context-Folding), 0.6 on WebArena (over Reflexion), and 1.1 on GAIA (over Reflexion). The cleanest test of whether the gains stem from privileged adaptation is the LOBO scheduler, which uses strictly less target-distribution information than any tuned baseline: it still outperforms every tuned baseline

Table 22: Scheduler state ablations (Qwen3.5-35B-A3B, Perf. / Tokens).

Scheduler state	SWE-Bench Verified	BrowseComp
Full (5 features)	**37.6**$\pm$0.9 / 298K	**19.2**$\pm$1.1 / 512K
w/o e_{progress}	36.9$\pm$1.0 / 305K	18.7$\pm$1.0 / 523K
w/o e_{diverge}	34.6$\pm$1.1 / 342K	16.4$\pm$1.2 / 604K
w/o e_{entropy}	37.1$\pm$0.9 / 309K	18.9$\pm$1.1 / 528K
w/o e_{error}	35.7$\pm$1.0 / 318K	18.1$\pm$1.2 / 549K
w/o e_{subtask}	36.2$\pm$1.0 / 331K	18.4$\pm$1.0 / 561K
Minimal: only e_{diverge} and e_{error}	36.8$\pm$1.0 / 312K	18.6$\pm$1.1 / 534K
All 5 features discretized to 3 bins	37.2$\pm$0.9 / 303K	18.9$\pm$1.0 / 519K
Learned encoding (GRU over the last 10 steps)	37.7$\pm$1.1 / 301K	19.3$\pm$1.2 / 515K

on SWE-Bench Verified and BrowseComp, and it is within one standard deviation of the strongest tuned baseline on WebArena (31.9$\pm$1.1 vs. 32.5$\pm$1.2 for Reflexion, at about half of its tokens) and on GAIA (45.1 vs. 45.2 for Reflexion, at about two thirds of its tokens).

J Scheduler State Representation

J.1 Feature Ablations

We retrain the scheduler from scratch for every variant (5,000 PPO steps, identical hyperparameters). Table 22 reports the full results with standard deviations.

Single-feature schedulers. On SWE-Bench Verified, schedulers that observe a single feature reach 35.8% (e_{diverge}), 34.9% (e_{error}), 34.6% (e_{subtask}), 33.5% (e_{progress}), and 33.2% (e_{entropy}), compared with 34.1% for the fixed schedule without any feature. e_{progress} and e_{entropy} alone are therefore worse than a fixed schedule and are useful only in combination, whereas e_{diverge} alone already beats it by 1.7 points, the largest margin of any single feature.

Importance versus ease of definition. The features that are hardest to define across environments are not the ones the method depends on. The two features that require intermediate structures, e_{progress} (the task graph) and e_{subtask} (the checkpoint structure of the tactical plan), cost only 0.7 and 1.4 points when removed, and only e_{entropy} matters less (0.5 points). The two features that carry the method, e_{diverge} (3.0 points) and e_{error} (1.9 points), need nothing environment-specific beyond an embedding distance and a failure status. Accordingly, the minimal scheduler that uses only these two features recovers 77% of the learned scheduler's improvement over the fixed schedule on SWE-Bench Verified (36.8% vs. 37.6% and 34.1%), without any task graph or checkpoint structure; we recommend this configuration for environments where the task graph is hard to define. A learned GRU encoding of the last 10 steps (37.7%) brings no gain over the hand-designed vector, and discretizing every feature into 3 bins (37.2%) shows that the policy does not depend on fine-grained feature values.

J.2 Robustness to Feature Noise

Injecting Gaussian noise into all five features at test time yields resolve rates of 37.4% ($\sigma = 0.1$), 36.8% ($\sigma = 0.2$), and 35.5% ($\sigma = 0.3$) on SWE-Bench Verified, all above the 34.1% of the fixed schedule. Together with the 3-bin discretization result, this shows that the policy does not require precise feature values, and it bounds the cost of the feature fallbacks of Appendix B.5 and of a stale task graph (Appendix B.4).

J.3 Reliability of Action Entropy

A model can be confident about an incorrect action, especially in exploratory or out-of-distribution settings, so low action entropy does not imply that a decision is easy or correct. Our measurements support this caution. Entropy is the weakest of the five features (removing it costs 0.5 points), and

Table 23: Action correctness by entropy quintile on 1,200 annotated decision points.

Setting	Lowest-entropy quintile	Highest-entropy quintile	Share of errors in lowest quintile
In distribution (SWE-Bench Verified)	82% correct	51% correct	9%
Out of distribution (OSWorld, unseen)	63% correct	49% correct	21%

an entropy-only scheduler (33.2%) is worse than the fixed schedule (34.1%), so MOSAIC does not rely on entropy as an uncertainty estimate. To quantify the confidently-wrong regime, we annotate 1,200 decision points and bin them by entropy quintile (Table 23). Entropy discriminates correct from incorrect actions in distribution, but it largely stops doing so out of distribution, where more than a fifth of all errors fall into the lowest-entropy quintile.

Why MOSAIC is robust to confidently wrong actions. Escalation is dominated by e_{diverge}, which triggers 61.2% of strategic invocations (Table 17) and is outcome-grounded: it asks whether reality matched the plan, not whether the model felt confident. A confidently wrong action is therefore caught one step later by e_{diverge} or e_{error} rather than by entropy. We additionally evaluate a progress-stall watchdog that escalates to the strategic layer after k consecutive steps without any status change in $\mathcal{G}_t$, regardless of entropy. On OSWorld, it reduces the detection latency for confidently wrong actions from 8.4 to 3.1 steps and raises the zero-shot success rate from 15.6% to 16.4%; the results in Table 3 are reported without it. Finally, replacing entropy with the disagreement among 3 self-consistency samples [Wang et al., 2023] gives 37.8% at 314K tokens on SWE-Bench Verified, marginally better than entropy but 5% more costly.

K Fidelity of Inter-Scale Context Distillation

Low-level trajectory compression may lose information, and high-level plan expansion may introduce errors. We therefore measure both directions directly.

Distill↑ fidelity. We build a probe set of 300 (log, question) pairs, generated automatically and then verified by humans, each targeting a fact in the log that a later decision needs; 112 of them target decision-critical facts. Each question is answered from the distilled summary alone. Unsupported statements in the summaries are judged by an NLI model, with a human audit of 200 samples. Table 24 shows that fidelity is high on the stronger backbones and degrades on the 9B model.

Table 24: Fidelity of Distill↑ summaries on the 300-question probe set.

Backbone	All-fact recall	Critical-fact recall (112)	Unsupported stmts.	Of those, changed a decision
Qwen3.5-35B-A3B	91.4%	96.1%	2.3%	0.4%
Gemma-4-27B-A4B	87.2%	94.3%	3.6%	0.9%
Qwen3.5-9B	79.8%	88.5%	5.9%	1.7%

Refine↓ correctness. Over 1,000 generated tactical plans, 94.7% of steps name an existing tool whose preconditions are satisfied, the hallucinated-tool rate is 1.2%, 88.3% of checkpoints are attainable, and the plan contradicts a constraint node of $\mathcal{G}_t$ in 3.5% of cases.

Mitigation. The constraint-preserving check of Appendix B.3 raises the critical-fact recall of Qwen3.5-9B from 88.5% to 98.2%, lifts its SWE-Bench Verified resolve rate from 28.9% to 29.8% at 0.6% extra tokens, and reduces constraint contradictions from 3.5% to 1.1%. Dependence on backbone quality nevertheless remains a limitation (Appendix M).

L Separation of the Strategic and Tactical Layers

Both the strategic plan and the tactical plan are structured, DAG-like objects, but this overlap is superficial. Table 25 contrasts the two layers along every dimension that matters for their interaction. The invariant is that only the strategic layer writes to $\mathcal{G}_t$ and only the tactical layer may name a tool; the tactical layer cannot change task-level goals except by reporting an `unexpected_event`, which triggers a strategic invocation.

Table 25: Division of responsibilities between the strategic and tactical layers.

Dimension	**Strategic** (π^S)	**Tactical** (π^T)
Time scope	Whole task (50–200 steps)	One subtask (5–20 steps)
Object of reasoning	*What* to achieve, in what order, under what constraints	*How* to do the current subtask with the tools at hand
State owned	Persistent $\mathcal{G}_t$, append-only, whole episode	Ephemeral p^T, discarded at subtask end
Vocabulary	Goals, subgoals, constraints (tool-agnostic)	Tool calls, checkpoints, fallbacks (tool-specific)
Sees `tool_inventory`	No	Yes
Writes to $\mathcal{G}_t$	Yes, exclusively	No, only reports up via Distill↑
Invocation / budget	5.4% of steps / 4,096 tokens	19.3% of steps / 2,048 tokens

Empirical non-redundancy. A merged variant, in which a single planner owns both $\mathcal{G}_t$ and tool selection (with the 4,096-token strategic budget and the same scheduler), reaches 35.4±1.1% at 341K tokens on SWE-Bench Verified, versus 37.6±0.9% at 298K for the separated layers and 35.8% at 305K for the 2-scale ablation (Table 4). The two outputs are also not paraphrases of each other: the Jaccard overlap between strategic subtask specifications and the tactical plans generated from them is 0.09, and tool names appear in 1.7% of strategic outputs versus 96.4% of tactical outputs.

M Limitations

We discuss several limitations of MOSAIC that represent opportunities for future work.

Fixed number of scales. MOSAIC uses a fixed three-scale hierarchy designed based on the intuition that tasks decompose into goals, subtasks, and actions. While our sensitivity analysis (Table 14) suggests that three scales offer a good trade-off, some tasks may benefit from finer or coarser decompositions. Learning the number of scales from data, potentially via architecture search or a dynamic depth mechanism, is an interesting direction.

Scheduler generalization across domains. The Scale Scheduler is trained on a mixture of 200 tasks from the four benchmarks. While the learned features are computed identically across environments, the optimal weighting of these features may vary across task types. Our transfer experiments (Appendix G) show that the scheduler degrades gracefully rather than failing: trained without any data from the target benchmark, it retains 85–87% of the gain over ReAct, 10 target tasks bring it within 0.3 points of the in-distribution scheduler, and it transfers zero-shot to OSWorld and τ-bench, whose tools are unseen in training. Two limitations remain. First, the sharpest shift we observed concerns horizon length: a scheduler trained only on tasks shorter than 100 steps loses 1.8–3.2 points on longer tasks (Appendix E.6), so scheduler training should include long-horizon tasks. Second, out of distribution the scheduler invokes the strategic layer more often (7.9% vs. 5.4% of steps), which reduces token savings, and action entropy loses most of its discriminative power (Appendix J.3).

Dependence on backbone LLM quality. The Distill↑ and Refine↓ operators rely on the backbone LLM's ability to produce accurate summaries and faithful plan expansions. For weaker models, these operators may introduce errors that propagate across scales. We observed that with Qwen3.5-9B, the Distill↑ summaries occasionally omit critical details, leading to suboptimal strategic replanning: their critical-fact recall is 88.5%, versus 96.1% with Qwen3.5-35B-A3B (Appendix K). A cheap constraint-preserving check raises this recall to 98.2%, but does not remove the dependence on backbone quality. Dedicated distillation training or a separate small model for these operators may help.

Evaluation scope. Our main evaluation covers four English-language benchmarks in software engineering, web browsing, and general reasoning, complemented by zero-shot transfer to computer use (OSWorld) and conversational tool use (τ-bench). The effectiveness of MOSAIC on other modalities (e.g., embodied agents, robotic manipulation), languages, or domain-specific tasks (e.g., scientific research, legal analysis) remains to be validated.

Theoretical assumptions. Theorem 1 assumes that the quality function is concave in budget with bounded curvature, that the marginal information gain has non-zero variance across states, and that the budget affects the quality of each decision without altering the state visitation distribution (Appendix A.1). Proposition 2 removes the need to know the information gain exactly, requiring only an estimate correlated with it. While the concavity assumption is empirically supported by recent studies on inference-time scaling [Snell et al., 2024], it may not hold in regimes where models exhibit phase transitions in capability at specific token thresholds (e.g., a model that cannot solve a problem at all below 1,000 tokens but solves it reliably above that threshold). Extending the theory to handle non-concave quality functions and allocation-dependent state distributions is left for future work.

N Ethics Statement

Intended use. MOSAIC is a research framework designed to improve the efficiency and effectiveness of LLM-based agents on long-horizon tasks. It is intended for use in controlled settings such as software engineering assistants, research tools, and web-based information retrieval systems.

Potential risks. As with all autonomous agent systems, MOSAIC carries risks associated with unsupervised execution of actions in real environments. The multi-scale architecture's ability to maintain long-term strategic coherence could make it more capable of pursuing extended sequences of actions, which amplifies both the benefits (completing useful tasks) and the risks (executing harmful action chains that a flat agent might abandon due to context loss). We recommend deploying MOSAIC with human-in-the-loop oversight, particularly for the strategic layer's plan revisions, which represent the highest-impact decision points.

Environmental impact. One motivation for MOSAIC is reducing the inference cost of long-horizon agents. Our experiments show 21–45% token reductions and 30–36% wall-clock speedups compared to flat baselines, directly translating to lower energy consumption and carbon emissions per task. The scheduler training phase adds a one-time cost of approximately 72 A100 GPU-hours, which is amortized over all subsequent inference runs.

Data and reproducibility. All benchmarks used in this work are publicly available. We do not collect or use any private user data. Our implementation, scheduler checkpoints, training and evaluation task ID lists, and raw trajectories are released at `https://github.com/BokwaiHo/MOSAIC` (Appendix B.6), and all hyperparameters are reported in this appendix to facilitate reproducibility.

O Broader Impacts

This work aims to improve the efficiency and reliability of long-horizon language agents through multi-scale adaptive inference control. Potential positive impacts include reducing inference cost and energy usage, improving the accessibility of capable agentic systems, and supporting beneficial applications such as software maintenance, scientific information gathering, educational assistance, accessibility tools, and complex workflow automation. However, more efficient and capable long-horizon agents may also introduce or amplify societal risks. Because MOSAIC improves agents' ability to plan, browse, use tools, and persist across many steps, it could be misused for disinformation, phishing, fake profile generation, automated surveillance, unauthorized data collection, or software vulnerability discovery. Lower inference cost may further increase the scale of such misuse. These risks motivate the use of safeguards such as tool-use restrictions, rate limits, monitoring, sandboxed execution, and human approval for sensitive actions. Fairness, privacy, and security concerns are also important. If the underlying models, tools, datasets, or reward signals encode biases, MOSAIC may preserve or amplify those biases over longer action sequences, especially in high-stakes domains such as hiring, finance, healthcare, education, policing, or access to public services. Deployments in such settings should include bias evaluation, human oversight, contestability, and domain-specific auditing. In addition, long-horizon agents may process sensitive user data, interaction histories, code, and web content; practical systems should therefore implement data minimization, access control, retention limits, secure logging, and privacy-preserving summarization. Finally, agentic systems remain vulnerable to prompt injection, unsafe tool use, data exfiltration, and adversarial observations. The hierarchical interfaces in MOSAIC, including inter-scale summaries and scheduler signals, should be protected through sandboxing, permission boundaries, provenance tracking, and robust security evaluation.

P Declaration of LLM usage

The authors used large language models solely as writing and editing assistants during the preparation of this manuscript. Specifically, LLMs were used to help improve clarity, organization, grammar, and phrasing. All core research ideas, theoretical claims, algorithmic designs, experimental protocols, results, analyses, and conclusions were developed, verified, and approved by the authors. The authors take full responsibility for the accuracy, originality, integrity, and final content of the paper. No LLM was used to fabricate experimental results, generate unsupported claims, or replace human judgment in the scientific contribution of this work.

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: Please see Section 1.

   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: Please see Appendix M.

   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: [Yes]

Justification: Please see Section 3.3 and Appendix A.

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: Please see Section 4.1 and Appendix B.

   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: Please refer to our provided Github repository https://github.com/BokwaiHo/MOSAIC.

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: Please see Section 4.1 and Appendix B.

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: Please see Section 4.1, Section 4.2, and Appendix C.

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: Please see Section 4.1 and Appendix B.

   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: Please see Appendix N.

   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: Please see Appendix O.

    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: [Yes]

    Justification: Please see Appendix N, where we discuss the potential risks of MOSAIC and the recommended safeguards for its deployment.

    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: Please see Section 4.1.

    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: [Yes]

    Justification: Please see our Github repository https://github.com/BokwaiHo/MOSAIC.

    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: This paper does not involve crowdsourcing nor 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: This paper does not involve crowdsourcing nor research with human subjects.

    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: Please see Appendix P.

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.