YOUR OPENCLAW, BUT MUCH MORE EFFICIENT

Anonymous authors
Paper under double-blind review

ABSTRACT

General-purpose agents such as OpenClaw transmit their complete tool inventory, framework instructions, and interaction history to the model at every step, although each action typically depends on only a small subset of this context. We formalize this observation through the *sufficient exposure* of a request, defined as the least costly reduction of the full request that preserves task performance within a prescribed tolerance. Context beyond the sufficient exposure incurs two costs: redundant token consumption and distraction from irrelevant capabilities, which can degrade the model's decisions. Improving agent efficiency thus amounts to estimating the sufficient exposure online, a problem with asymmetric error costs: over-exposure incurs these costs gradually, whereas under-exposure withholds information required for correct execution. Motivated by this asymmetry, we propose *LeanClaw*, a request-level exposure controller for OpenClaw. LeanClaw initializes the exposed tool set from a compact prior over necessary tools and adapts it as execution feedback reveals new requirements or retires old ones. It extends the same criterion to interaction history through structured state preservation and duplicate removal, and it reverts to broader exposure whenever its estimate is uncertain. All runtime capabilities remain available; only the capabilities exposed in each request are altered. Across five benchmarks spanning mathematical reasoning, code generation, and reading comprehension, LeanClaw reduces token consumption by 46.64–59.29% relative to OpenClaw while improving task performance on every benchmark. On RealClawBench, it matches OpenClaw's success rate with 48.15% fewer tokens. Evaluations across multiple foundation models further demonstrate that these efficiency gains generalize beyond a single model route. Code is available at `https://anonymous.4open.science/r/leanclaw-08BF`.

1 INTRODUCTION

General-purpose agents have evolved into unified execution systems that inspect files, edit code, invoke tools, run commands, and interact with external environments through a shared agent loop (Yao et al., 2023; Schick et al., 2023; Qin et al., 2024; Yang et al., 2024; Li et al., 2023; Zhang et al., 2025; Hu et al., 2025b). OpenClaw exemplifies this design: it integrates a broad tool inventory, framework instructions, and interaction history into a single agent interface (OpenClaw contributors, 2026). This breadth is what allows one system to handle diverse tasks, but it has a direct consequence at the request level. Every step transmits the complete context to the model, although each action typically depends on only a small subset of it.

Existing efficiency methods have progressively moved the control point closer to the source of redundant context. Prompt- and reasoning-compression methods reduce local redundancy within a single request by shortening or rewriting task inputs and intermediate reasoning (Jiang et al., 2023; Pan et al., 2024; Li et al., 2025; Xia et al.). As interactions grow longer, trajectory- and memory-management methods extend this idea across requests by reducing, summarizing, or reorganizing accumulated tool calls, observations, and execution histories (Kang et al., 2025b; Xiao et al., 2026; Hu et al., 2025a; Xu et al., 2026; Kang et al., 2025a; Wang et al., 2026a; Ghulyani et al., 2026; Hu et al., 2026; Xiong et al., 2026). Tool-aware methods move further upstream by improving how external capabilities are represented, retrieved, and matched to tasks (Yuan et al., 2025; Lumer et al., 2026b; 2025). Together, these methods progressively reduce what the model sees, from request content to interaction history to tool representations. However, each reduces one component

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

of an already assembled request by a relevance or budget criterion, and doing so can remove task constraints, execution evidence, or tools that become necessary later. None asks the question that precedes all of them: *how much of a request does the agent actually need in order to act correctly?*

We make this question precise through the *sufficient exposure* of a request (Section 3). At each step, a conventional framework sends the full context: the framework prompt, the task, the schemas of every available tool, and the complete interaction history. A *reduction policy* instead sends a reduced request, which may select a subset of tools and instructions and replace raw history with structured state. A policy is ϵ-*sufficient* if its task performance stays within ϵ of full exposure, and the sufficient exposure is the request produced by the least costly such policy. Context beyond it incurs two costs. The first is redundant token consumption, and any policy observed to be sufficient certifies a lower bound on the tokens that full exposure wastes. The second is distraction: irrelevant capabilities compete for the model's attention and can degrade its decisions. Full context is therefore not necessarily optimal, and we define sufficiency one-sidedly so that a reduced request may outperform it.

The central problem is then to estimate the sufficient exposure online, from the task, the execution state, and feedback such as errors and newly discovered artifacts. This estimation has asymmetric error costs. Over-exposure incurs these costs gradually, whereas under-exposure can withhold information the task cannot do without, a loss that cannot always be detected and repaired. We show that any fixed or one-shot selection is therefore capped by the coverage of its initial guess, so the estimate must adapt as requirements surface during execution, fall back to broader exposure when in doubt, and keep its own control overhead small relative to the tokens it saves.

Following these requirements, we propose *LeanClaw*, a request-level exposure controller for OpenClaw. LeanClaw separates capability *availability* from capability *exposure*: the runtime retains the complete tool set together with the original model route, agent loop, workspace, and verifier, while the model at each step sees only an estimate of the sufficient exposure. An *adaptive tool adapter* initializes the exposed tool set from a compact prior over necessary tools and adapts it as execution feedback reveals new requirements or retires old ones. *Structured state preservation*, *history compaction*, and *duplicate-context removal* extend the same criterion to interaction history. *Fail-open recovery* reverts to broader exposure, less compressed history, or the original OpenClaw request whenever the estimate is uncertain. Across five benchmarks spanning mathematical reasoning, code generation, and reading comprehension, LeanClaw reduces token consumption by 46.64–59.29% relative to OpenClaw while improving task performance on every benchmark (Table 1). On RealClawBench, it matches OpenClaw's success rate with 48.15% fewer tokens. Evaluations across multiple foundation models further demonstrate that these efficiency gains generalize beyond a single model route. As predicted, a controlled diagnostic shows that fixed and one-shot selection fall below full exposure in accuracy, whereas LeanClaw exceeds it while exposing the fewest schemas.

Our contributions are threefold:

- We formalize the *sufficient exposure* of an agent request, showing that excess context wastes tokens and can distract the model, and that estimating it requires adaptation to execution feedback.
- We propose *LeanClaw*, an adaptive estimator of sufficient exposure that adjusts the exposed context as execution feedback reveals new requirements, while leaving the OpenClaw runtime unchanged.
- LeanClaw reduces token consumption by 46.64–59.29% on five benchmarks and by 48.15% on RealClawBench while matching or improving performance, with efficiency gains that generalize across multiple foundation models.

2 Related Work

General-purpose agent systems. Tool-using agents such as ReAct, Toolformer, and ToolLLM connect language models to environments and external APIs, and developer agents such as SWE-agent and OpenClaw, evaluated on benchmarks such as SWE-bench and RealClawBench, extend them to repositories, commands, and live workspaces (Yao et al., 2023; Schick et al., 2023; Qin et al., 2024; Yang et al., 2024; Jimenez et al., 2024; OpenClaw contributors, 2026; Lv et al., 2026). Multi-agent frameworks such as CAMEL, MetaGPT, Smolagents, OWL, and OAgents add task decomposition and collaboration (Li et al., 2023; Zhang et al., 2025; Hugging Face, 2025; Hu et al.,

2025b; Zhu et al., 2025). These systems study what an agent can do once its context is available. LeanClaw is orthogonal to the agent design and asks how much of that context each request must expose.

Agent context efficiency. Prompt and reasoning compression, such as LLMLingua, LLMLingua-2, 500xCompressor, and TokenSkip, removes low-contribution tokens from a single request (Jiang et al., 2023; Pan et al., 2024; Li et al., 2025; Xia et al.). For long-horizon agents, ACON and AgentDiet compress trajectories; HiAgent, A-Mem, Memory OS, and HyMem build hierarchical or structured memory; PACMS and ZipRL learn what to retain; and empirical studies examine how memory management changes agent behavior (Kang et al., 2025b; Xiao et al., 2026; Hu et al., 2025a; Xu et al., 2026; Kang et al., 2025a; Wang et al., 2026a; Ghulyani et al., 2026; Hu et al., 2026; Xiong et al., 2026). Tool-aware methods shorten tool instructions or select and retrieve tools and agents dynamically (Yuan et al., 2025; Lumer et al., 2026a;b; 2025). These methods reduce a request after it is assembled, by relevance or budget, and can remove evidence or tools that are needed later. LeanClaw instead targets sufficiency: every tool stays executable in the runtime, each request exposes an adaptive estimate of what the agent needs, and the estimate falls back to broader exposure whenever it is uncertain.

3 Sufficient Exposure

3.1 Requests, Reductions, and Sufficiency

An agent solves a task x by alternating model requests and tool executions. Under a conventional framework, the request at step t is the full context $c_t = (S, x, \mathcal{G}, h_t)$, where S is the framework prompt, $\mathcal{G}$ contains the schemas of every tool the runtime can execute, and h_t is the history of model outputs, tool calls, observations, and errors. A *reduction policy* $\phi = \{\phi_t\}_t$ replaces c_t with $\tilde{c}_t = \phi_t(c_t)$. We consider reductions that act componentwise and leave the task itself intact:

$$\phi_t(c_t) = \big(\phi_t^{\mathrm{inst}}(S),\ x,\ \phi_t^{\mathrm{tool}}(\mathcal{G}),\ \phi_t^{\mathrm{hist}}(h_t)\big), \qquad \phi_t^{\mathrm{tool}}(\mathcal{G}) = \mathcal{E}_t \subseteq \mathcal{G}. \tag{1}$$

The instruction component retains the parts of the framework prompt relevant to the current step, and the tool component selects the exposed schemas $\mathcal{E}_t$. The history component may drop, deduplicate, or summarize past turns and add a structured state summary, so its output need not be a subset of h_t. A reduction changes only what the model *sees*: the runtime still executes any tool in $\mathcal{G}$, so capability *availability* is decoupled from capability *exposure*.

Let $A(\phi)$ denote the expected task performance under ϕ on a task distribution $\mathcal{D}$, as judged by the verifier, and let $N(\phi)$ denote the expected number of provider tokens per task, counting every provider call made under ϕ, including those it issues for its own control. Full exposure is the identity policy ϕ_{full}. We take Φ to be the reductions of the form in Eq. (1) that depend only on information available at step t.

Definition 1 (Sufficiency)**.** *A reduction policy ϕ is* ϵ-sufficient *on $\mathcal{D}$ if $A(\phi) \geq A(\phi_{\mathrm{full}}) - \epsilon$.*

Definition 2 (Sufficient exposure)**.** *For a policy class Φ and tolerance $\epsilon \geq 0$, the* minimal sufficient cost *is $N_\epsilon^* = \inf\{N(\phi) : \phi \in \Phi,\ \phi$ is ϵ-sufficient$\}$. A policy whose cost attains or approaches N_ϵ^* issues the* sufficient exposure $\tilde{c}_t^*$*, and $N(\phi_{\mathrm{full}}) - N_\epsilon^*$ is the number of redundant tokens that full exposure spends beyond it.*

The definition has three consequences. First, sufficiency is one-sided: it admits $A(\phi) > A(\phi_{\mathrm{full}})$, and we call $A(\phi) - A(\phi_{\mathrm{full}})$ the *distraction gain*. A positive gain is consistent with irrelevant context harming the model, in which case excess context costs accuracy as well as tokens. Second, N_ϵ^* is defined over whole trajectories rather than single requests, because reducing one request changes every subsequent one. Third, because control overhead is part of $N(\phi)$, a policy earns no credit for savings that it spends on its own planning calls.

Remark 1 (Certified lower bound)**.** *N_ϵ^* is not computable, but any ϵ-sufficient $\hat{\phi} \in \Phi$ satisfies $N_\epsilon^* \leq N(\hat{\phi})$ by definition, so full exposure spends at least $N(\phi_{\mathrm{full}}) - N(\hat{\phi})$ redundant tokens. Every policy observed to be sufficient thus certifies a lower bound on the redundancy, and better estimators tighten it.*

Because A and N are estimated on finite benchmarks, we take sufficiency to mean a matched or improved pass count (F1 for DROP) on an identical case set ($\epsilon = 0$), and treat the resulting bounds as empirical estimates rather than statistical guarantees.

3.2 Why Estimation Must Be Adaptive and Fail-Open

The sufficient exposure is not observed online; it must be estimated from the task, the execution state, and feedback. Errors in this estimate are asymmetric: over-exposure raises N and, through distraction, can gradually lower A, whereas under-exposure can lower A irrecoverably. We make this precise for the tool component. Let $R(x) \subseteq \mathcal{G}$ be the set of tools the task requires over its trajectory.

Assumption 1. *(A1)* Non-substitutability*: if a required tool is never exposed, the task fails. (A2)* Detectability*: when a required tool $r \notin \mathcal{E}_t$ becomes needed, execution feedback (an out-of-set tool request, an error, or a newly revealed artifact) signals the miss before the step budget is exhausted, with probability at least $1 - \eta$.*

Proposition 1 (Fixed versus adaptive exposure)**.** *Under (A1), any fixed exposure $\mathcal{E}_t \equiv \mathcal{E}_0$ satisfies $A(\phi_{\text{fixed}}) \leq \Pr[R(x) \subseteq \mathcal{E}_0]$. Under (A1) and (A2), consider an adaptive policy that adds the signaled tools on every detected miss and never withdraws a required tool once exposed. It exposes every required tool before the step budget is exhausted with probability at least $1 - \eta\,\mathbb{E}\,|R(x) \setminus \mathcal{E}_0|$, performs at most $|R(x) \setminus \mathcal{E}_0|$ recovery expansions, and never exposes more than $\mathcal{G}$.*

The proof is in Appendix A. The proposition separates two regimes. A fixed or one-shot selector is capped by the coverage of its initial guess, because requirements that surface during execution can never be met. Feedback-driven adaptation removes this cap, and because required tools are never withdrawn, the number of recovery expansions is bounded by the required tools outside the prior rather than by the trajectory length. The coverage bound identifies the two quantities an estimator must keep small, which become LeanClaw's two design rules: a compact prior restricted to tools that are necessary with high confidence, which keeps $\mathbb{E}|R(x) \setminus \mathcal{E}_0|$ small without exposing speculative schemas, and a fail-open gate that treats ambiguous signals as misses and resolves them toward broader exposure, keeping η small.

History admits no analogous guarantee. A dropped observation, unlike a missing tool, rarely produces an explicit signal, so (A2) does not hold and lost information cannot be recovered by later expansion. LeanClaw therefore reduces history conservatively: it preserves execution state in structured form, keeps content that cannot be reconstructed verbatim, and restores less compressed history whenever the gate is uncertain (Section 4.2).

4 LeanClaw: Estimating Sufficient Exposure Online

LeanClaw instantiates a reduction policy $\hat{\phi}$ with the factorization of Eq. (1). At each step it sends

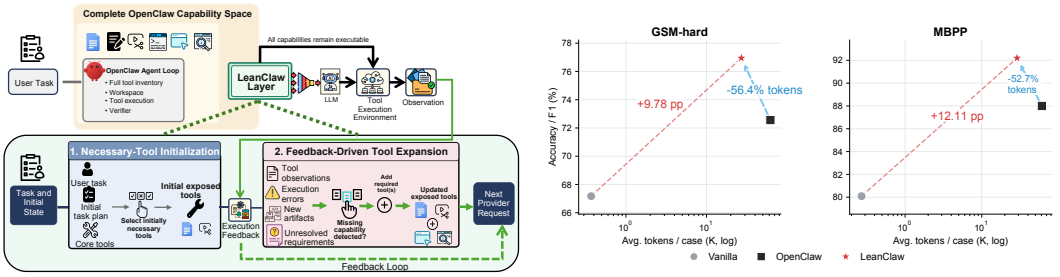
$$\hat{c}_t = \hat{\phi}_t(c_t) = \big(\widetilde{S}_t,\ x,\ \mathcal{E}_t,\ \widetilde{h}_t,\ z_t\big), \tag{2}$$

where $\mathcal{E}_t \subseteq \mathcal{G}$ are the exposed tool schemas, $\widetilde{S}_t$ are the instructions relevant to the current stage and to the exposed tools, $\widetilde{h}_t$ is the retained history, and z_t is a structured state capsule. The adaptive tool adapter estimates ϕ^{tool} from a necessary-tool prior that it adapts as execution feedback reveals new requirements (Section 4.1), the state manager estimates ϕ^{hist} (Section 4.2), and a fail-open gate decides when either estimate is trusted. The prior and the gate implement the two design rules of Section 3.2, keeping $\mathbb{E}|R(x) \setminus \mathcal{E}_0|$ and η small. LeanClaw sits at the provider interface and leaves the OpenClaw model route, agent loop, tool implementations, workspace, execution environment, and verifier unchanged, so $\mathcal{G}^{\text{LeanClaw}}_{\text{runtime}} = \mathcal{G}^{\text{OpenClaw}}_{\text{runtime}}$ (Figure 1a); Algorithm 1 summarizes the loop.

4.1 Tool Exposure: Necessary-Tool Prior and Adaptive Update

Necessary-tool prior. The adapter maintains an exposure profile $p_t = \text{Adapt}(x, h_t, z_t, \mathcal{G})$ that summarizes task requirements, tool dependencies, execution progress, protected constraints, and remaining goals. The initial exposure is

$$\mathcal{E}_0 = \mathcal{M}_{\text{core}} \cup \mathcal{M}_{\text{task}}(x) \subseteq \mathcal{G}, \tag{3}$$



(a) LeanClaw system overview **(b) Accuracy–token trade-off**

Figure 1: (a) LeanClaw keeps the full OpenClaw runtime and estimates the sufficient exposure at the provider interface, starting from a necessary-tool prior and adapting it on execution feedback. (b) On GSM-hard and MBPP, LeanClaw lies up and to the left of OpenClaw, so OpenClaw's extra tokens buy no accuracy. The leftward shift lower-bounds its redundant tokens (Remark 1); the upward shift is the distraction gain.

Algorithm 1 LeanClaw: online estimation of sufficient exposure

Require: task x, runtime tool set $\mathcal{G}$, framework prompt S, threshold τ

$z_0, h_0 \leftarrow \varnothing$; $p_0 \leftarrow \mathrm{Adapt}(x, h_0, z_0, \mathcal{G})$; $\mathcal{E}_0 \leftarrow \mathcal{M}_{\mathrm{core}} \cup \mathcal{M}_{\mathrm{task}}(x)$

for $t = 0, 1, 2, \ldots$ until the agent stops **do**

  $\hat{c}_t \leftarrow (\widetilde{S}_t, x, \mathcal{E}_t, \widetilde{h}_t, z_t)$ ▷ Eq. (2)

  $r_t \leftarrow \hat{c}_t$ if $q_t \geq \tau$, else $r_t^{\mathrm{conservative}}$ ▷ fail-open gate, Eq. (5)

  $a_t \leftarrow \textsc{Model}(r_t)$; $(o_t, e_t) \leftarrow \textsc{Runtime}(a_t)$ ▷ runtime can execute any tool in $\mathcal{G}$

  $z_{t+1} \leftarrow \mathrm{UpdateState}(z_t, a_t, o_t, e_t)$; $h_{t+1} \leftarrow h_t \oplus (a_t, o_t, e_t)$

  $p_{t+1} \leftarrow \mathrm{UpdateProfile}(p_t, o_t, e_t, z_{t+1})$; $\mathcal{E}_{t+1} \leftarrow (\mathcal{E}_t \setminus \Omega_t) \cup \Delta_t$ ▷ adaptive update

end for

where $\mathcal{M}_{\mathrm{core}}$ contains generally necessary tools and $\mathcal{M}_{\mathrm{task}}(x)$ contains tools inferred from the task and the initial execution state. Tools for workspace inspection, artifact creation or modification, testing, and verification are included when they are relevant to the initial stage. Following Proposition 1, the prior contains only tools that are necessary with high confidence; tools whose need is uncertain are left to feedback-driven adaptation rather than exposed in advance.

Feedback-driven adaptation. After each step, the adapter updates its profile with the latest observation o_t and any execution error or unresolved requirement e_t, $p_{t+1} = \mathrm{UpdateProfile}(p_t, o_t, e_t, z_{t+1})$. Feedback may reveal a new artifact, a workspace dependency, or a verification requirement that was unavailable at initialization. When the evidence indicates that the exposed set is insufficient, including the case where the model requests a tool outside $\mathcal{E}_t$, the adapter identifies the missing tools Δ_t, and tools that the updated profile marks as no longer relevant to the current stage form Ω_t:

$$\mathcal{E}_{t+1} = (\mathcal{E}_t \setminus \Omega_t) \cup \Delta_t, \qquad \Delta_t \subseteq \mathcal{G}, \;\; \Omega_t \cap \Delta_t = \varnothing. \tag{4}$$

This is the adaptive policy of Proposition 1: a tool is withdrawn only when the profile no longer requires it, so required tools stay exposed, and each recovery expansion restores a distinct required tool. Initialization and adaptation are dependency-closed: selecting a tool also exposes the dependencies and invocation instructions it needs, which form the part of $\widetilde{S}_t$ that varies with $\mathcal{E}_t$.

Quantifying the asymmetry. In OpenClaw the two error types differ by orders of magnitude. A tool schema costs about 410 tokens per request on average (77,900 tokens for 38 schemas over five requests; Appendix B), so one unnecessary schema kept over a ten-request trajectory costs roughly 4K tokens, about 3% of the average case in the diagnostic of Table 4. A missing required tool, by contrast, costs the task unless the miss is detected and repaired, and each repair adds a control call. Because a miss in the prior is recoverable whenever it is detected, the irrecoverable loss comes only from undetected misses. LeanClaw therefore keeps the prior compact but errs toward inclusion whenever the evidence of a miss is ambiguous, and withdraws a tool only when the profile no longer requires it.

4.2 History Compaction and Fail-Open Recovery

History lacks the detectability that makes tool expansion safe (Section 3.2), so LeanClaw reduces it conservatively and guards every reduction, of tools and history alike, with a fail-open gate.

Structured state capsule. The capsule z_t records completed steps, remaining goals, inspected and modified artifacts, unresolved errors, and verifier-relevant evidence. It is updated after each interaction as $z_{t+1} = \mathrm{UpdateState}(z_t, a_t, o_t, e_t)$, where a_t is the executed action. The capsule summarizes execution progress, so the model does not need to reread the full trajectory to know where it stands.

Compaction and deduplication. The retained history is $\widetilde{h}_t = \mathrm{Recent}_k(h_t) \cup \mathrm{Evidence}_t$. Recent turns remain verbatim, and $\mathrm{Evidence}_t$ keeps, in exact form, content that cannot be safely reconstructed: file paths, identifiers, JSON keys, numeric values, error messages, command arguments, and strict output requirements. Older turns whose content is already recorded in the capsule are replaced by the capsule or by compact references, and repeated observations and messages that add no new evidence are removed. The original task, the current observation, and tool-call arguments are never reduced.

Fail-open gate and cost accounting. Reduction is weakened or disabled whenever it might violate sufficiency: exact-copy requirements, strict output formats, unresolved errors, inconsistent state, uncertain tool selection, or verification conditions that require exact evidence. Writing q_t for the controller's confidence that the reduced request is sufficient, which any of these conditions lowers, and τ for its threshold,

$$r_t = \begin{cases} \hat{c}_t, & q_t \geq \tau, \\ r_t^{\mathrm{conservative}}, & q_t < \tau, \end{cases} \tag{5}$$

where the conservative request broadens $\mathcal{E}_t$, restores less-compressed history, or reverts to the original request c_t. The total cost is $N(\hat{\phi}) = N_{\mathrm{exposure}} + N_{\mathrm{control}}$, where the second term covers profile updates, expansion, and recovery; we measure it directly in Section 5.4.

5 Experiments

5.1 Experimental Setup

Benchmarks. We evaluate on mathematical reasoning (GSM-hard and AIME), code generation (HumanEval and MBPP), reading comprehension (DROP), and interactive developer-agent execution (RealClawBench) (Cobbe et al., 2021; HuggingFaceH4, 2024; Chen et al., 2021; Austin et al., 2021; Dua et al., 2019; Lv et al., 2026). RealClawBench turns real OpenClaw production sessions into 281 reproducible tasks with reconstructed environments and deterministic scorers. MBPP (500 problems) is the controlled setting for method comparison, ablation, and cross-model study. The cross-benchmark comparison uses Qwen3-32B (Yang et al., 2025); RealClawBench, the MBPP method comparison, and the ablation use DeepSeek V4 Flash (DeepSeek-AI, 2026); the cross-model study adds GPT-OSS-120B (OpenAI, 2025) and MiMo v2.5 (Xiaomi MiMo Team, 2026).

Protocol and metrics. For every paired OpenClaw–LeanClaw comparison, the task files, seeds, initial workspace, tool implementations, execution limits, verifier, and model route are identical; only the requests sent to the provider differ. This makes each pair in which LeanClaw matches or exceeds OpenClaw a direct instance of Remark 1 with $\epsilon = 0$. Token counts are provider/proxy usage (input plus output, with reasoning a subset of output) and include every control call, so $N(\hat{\phi})$ is measured in full. A case passes only if execution and verification both succeed; runtime-failed cases are rerun under the same configuration for both methods.

Compared methods. Besides OpenClaw, the cross-benchmark comparison includes direct generation (Vanilla), chain-of-thought with three-sample self-consistency (Wei et al., 2022; Wang et al., 2023), and the agent systems OWL, MetaAgent, and Smolagents (Zhang et al., 2025; Hugging Face, 2025; Hu et al., 2025b). RealClawBench additionally includes AWorld and OAgents (InclusionAI, 2025; Zhu et al., 2025). For exposure strategies, we adapt ToolLLM- and Re-Invoke-style retrieval

(Qin et al., 2024; Chen et al., 2024); for generic compression, we adapt EASYTOOL, AgentDiet, and SWE-Pruner (Yuan et al., 2025; Xiao et al., 2026; Wang et al., 2026b).

5.2 How Much of the Full Context Is Redundant?

Cross-benchmark lower bounds. Table 1 compares all methods with Qwen3-32B under the same protocol. On all five benchmarks, LeanClaw matches or exceeds OpenClaw's accuracy, so by Remark 1 its token reduction is a certified lower bound on the redundant tokens of OpenClaw in that setting. Figure 1b shows that OpenClaw is thus not a point on an accuracy–token trade-off but a dominated one: removing 56.4% and 52.7% of its tokens on GSM-hard and MBPP raises accuracy by 4.39 and 4.20 points. The bound is 46.64% on AIME, 52.66% on MBPP, 56.42% on GSM-hard, 57.39% on DROP, and 59.29% on HumanEval. It is substantial even on reasoning tasks that barely use tools, consistent with redundancy being driven by what the framework transmits rather than what the task needs: OpenClaw spends 36.8–69.2K tokens per case regardless of task family.

Real developer workloads. On RealClawBench (Table 2), OpenClaw and LeanClaw pass exactly the same number of cases (130/281), and LeanClaw reduces total provider tokens from 127.36M to 66.03M, which certifies that at least 48.15% of OpenClaw's tokens on this benchmark are redundant. The reduction is almost entirely on the input side (124.07M to 63.12M), where the redundant exposure lives. LeanClaw issues more responses (5.29K versus 4.23K): the control overhead N_{control} is paid in extra, much smaller requests. External agents cluster near OpenClaw's budget (114.97–138.15M), consistent with none of them reducing per-request exposure; OWL passes four more cases, at 1.88 times LeanClaw's tokens.

Consistency across model families. Table 3 repeats the paired comparison on MBPP with four model routes. LeanClaw matches or improves the pass count on every route, so each row certifies a lower bound: 42.79% for GPT-OSS-120B, 52.66% for Qwen3-32B, 58.89% for DeepSeek V4 Flash, and 66.60% for MiMo v2.5. Provider cost falls by 31.92–65.58%. The bound grows with the baseline budget, from 21.50M (GPT-OSS-120B) to 52.87M (MiMo v2.5), consistent with the removed tokens being budget spent on unused exposure.

5.3 Where Does the Redundancy Come From?

Exposed versus used tools. Successful RealClawBench trajectories use very few tools. In five cases from the 40-case diagnostic subset that pass under both OpenClaw and a tool-selection-only configuration of LeanClaw, each trajectory uses only one to three tool types, yet OpenClaw exposes all 38 schemas in every request. LeanClaw exposes six or seven and cuts total schema tokens by 94.5–95.8% with the outcome unchanged (Appendix B). Most of what the model reads about its capabilities is never used.

Distraction. Reduced exposure often does better than full exposure. With Qwen3-32B (Table 1), LeanClaw's distraction gain is positive on all five benchmarks: $+4.39$ on GSM-hard, $+4.20$ on MBPP, $+0.61$ on HumanEval, and $+0.65$ F1 on DROP. On MBPP it is positive for three of four model families, 10–21 additional passes out of 500, and zero for the fourth (Table 3). In the tool-exposure diagnostic (Table 4), exposing 5.35 schemas per request instead of 38 raises accuracy from 95.0% to 97.5%, one additional pass of 40; this comparison changes only the exposed tool set, so it isolates schema distraction from the effect of history management. The full context is thus not a neutral superset: the results are consistent with irrelevant schemas and stale history competing for attention. The effect is not universal (zero on the full281 run of RealClawBench), and we do not rest it on AIME, whose $+10.00$ is 3 of 30 problems.

5.4 Must Estimation Be Adaptive?

To isolate the tool component, we run a controlled diagnostic on 40 RealClawBench cases drawn from those OpenClaw passes in the full281 run and spanning all five task categories, where strategies differ only in how $\mathcal{E}_t$ is chosen. This selection favors full exposure, so any reduced policy that matches it does so on the baseline's own successes; on rerun, full exposure passes 38 of the 40, reflecting run-to-run variance. *Full exposure* is OpenClaw's default, with all 38 schemas in every

Table 1: Performance and average provider-token consumption on five benchmarks with Qwen3-32B. For DROP, we report all-case F1; other datasets report accuracy. LeanClaw changes are relative to OpenClaw. Green, blue, and red arrows denote resource reductions, performance gains, and performance losses, respectively.

Method	Metric	Reasoning		Code generation		Reading
		GSM-hard	AIME	HumanEval	MBPP	DROP
Direct and agentic baselines						
Vanilla	Acc / F1 (%)	67.17	26.67	76.82	80.09	76.36
	Avg. Tokens (K)	0.37	2.01	0.28	0.27	0.46
CoT SC (3-shot)	Acc / F1 (%)	69.01	30.00	77.78	81.26	77.72
	Avg. Tokens (K)	2.62	14.26	1.42	1.29	2.73
OWL	Acc / F1 (%)	72.48	33.33	90.74	79.08	79.85
	Avg. Tokens (K)	15.67	56.11	31.87	54.80	11.47
MetaAgent	Acc / F1 (%)	72.14	26.67	74.08	79.86	78.16
	Avg. Tokens (K)	4.35	6.24	2.59	6.39	1.43
Smolagents	Acc / F1 (%)	74.33	30.00	92.07	85.68	81.08
	Avg. Tokens (K)	11.59	59.14	40.91	111.07	12.01
OpenClaw-based control						
OpenClaw	Acc / F1 (%)	72.56	23.33	90.85	88.00	81.98
	Avg. Tokens (K)	62.08	52.77	69.19	60.89	36.80
Ours						
OpenClaw + LeanClaw (ours)	Acc / F1 (%)	**76.95** ↑4.39	**33.33** ↑10.00	91.46 ↑0.61	**92.20** ↑4.20	**82.63** ↑0.65
	Avg. Tokens (K)	27.06 ↓56.42%	28.16 ↓46.64%	28.17 ↓59.29%	28.83 ↓52.66%	15.68 ↓57.39%

Table 2: Agent baselines on RealClawBench full281 using DeepSeek V4 Flash. OWL, AWorld, and OAgents are external agent baselines (Hu et al., 2025b; InclusionAI, 2025; Zhu et al., 2025). All rows report completed full281 runs. The LeanClaw token reduction is measured against OpenClaw.

Method	Outcome			Provider usage				
	Cases	Pass	Acc (%)	Responses (K)	Input (M)	Output (M)	Reasoning (M)	Total (M)
OpenClaw-based control								
OpenClaw	281	130	46.26	4.23	124.07	3.29	1.45	127.36
External agent baselines								
OWL	281	134	47.69	5.46	121.05	3.00	1.10	124.06
AWorld	281	123	43.77	7.16	135.34	2.82	0.54	138.15
OAgents	281	122	43.42	3.81	111.75	3.22	1.00	114.97
Ours								
OpenClaw + LeanClaw	281	130	46.26	5.29	63.12	2.91	1.06	**66.03** ↓48.15%

request. *ToolLLM-style* and *Re-Invoke-style* adapt the one-shot retrieval principles of ToolLLM and Re-Invoke (Qin et al., 2024; Chen et al., 2024). *Necessary-only* is a fixed policy in the sense of Proposition 1: it exposes the same six core tools for file editing and execution in every request and never changes them. The *LeanClaw adapter* adds feedback-driven adaptation and the fail-open gate; history management is disabled for every strategy.

Fixed and one-shot selection are not sufficient. Table 4 matches the prediction of Proposition 1. All three non-adaptive policies expose 6–7 schemas per request and fall below full exposure, to 87.50–90.00%, so none of them is sufficient at $\epsilon = 0$. Necessary-only is the most informative row. Its compact core set covers the tools most tasks use, yet it loses 10 points relative to LeanClaw, which is the accuracy cost of requirements that a fixed set does not cover. Retrieval does not fix this, because a better one-shot guess still cannot anticipate requirements revealed later. The LeanClaw adapter is the only reduced policy that is sufficient. It reaches 97.50% while exposing the *fewest* schemas, 5.35 per request. Adaptivity therefore does not buy accuracy with a larger exposure set; it lets the controller start small and add tools only when they are needed.

The price of adaptation is bounded. Adaptation is not free. LeanClaw uses 3.411M tokens against 2.755M for Necessary-only, and since LeanClaw exposes fewer schemas per request, this 0.66M gap is mainly the control overhead N_{control} for profile updates, expansion, and recovery. It is the price of sufficiency, and it is small relative to the exposure it removes: total tokens remain 30.44% below full exposure, and tokens per passed case 32.23% below (129.06K to 87.46K). Because Proposition 1 bounds the number of recovery expansions by the required tools outside the

Table 3: Cross-model MBPP full500. Changes are relative to OpenClaw.

Model	Method	Outcome		Efficiency	
		Pass	Acc (%)	Total (M)	Cost (USD)
DeepSeek V4 Flash					
	OpenClaw	479	95.8	40.97	2.1723
	LeanClaw	489	97.8 ↑2.0	16.84 ↓58.89%	0.9689 ↓55.40%
Qwen3-32B					
	OpenClaw	440	88.0	30.45	3.0083
	LeanClaw	461	92.2 ↑4.2	14.41 ↓52.66%	1.6656 ↓44.63%
GPT-OSS-120B					
	OpenClaw	403	80.6	21.50	0.9784
	LeanClaw	423	84.6 ↑4.0	12.30 ↓42.79%	0.6661 ↓31.92%
MiMo v2.5					
	OpenClaw	473	94.6	52.87	2.2424
	LeanClaw	473	94.6	17.66 ↓66.60%	0.7718 ↓65.58%

Table 4: Tool-exposure strategies on 40 RealClaw-Bench cases. Changes are relative to full exposure. Per pass divides total tokens by passed cases.

Strategy	Outcome and exposure		Tokens	
	Acc (%)	Schemas/req.	Total (M)	Per pass (K)
Full exposure				
OpenClaw	95.00	38.00	4.90	129.06
Fixed or one-shot selection				
ToolLLM-style	90.00 ↓5.00	6.89 ↓81.88%	3.31	91.86
Re-Invoke-style	90.00 ↓5.00	6.89 ↓81.87%	2.87	79.72
Necessary-only	87.50 ↓7.50	6.00 ↓84.21%	2.76	78.71
Adaptive and fail-open				
LeanClaw	97.50 ↑2.50	5.35 ↓85.92%	3.41	87.46

Table 5: Generic efficiency methods on MBPP full500 (DeepSeek V4 Flash). Changes are relative to OpenClaw.

Method	Pass	Acc (%)	Total (M)
OpenClaw	479	95.8	40.97
+ EASYTOOL	470	94.0 ↓1.8	17.59 ↓57.08%
+ AgentDiet (θ=200)	470	94.0 ↓1.8	40.69 ↓0.69%
+ SWE-Pruner	468	93.6 ↓2.2	40.98 ↑0.03%
+ LeanClaw	**489**	**97.8** ↑2.0	**16.84** ↓58.89%

Table 6: Component ablation on MBPP full500 (DeepSeek V4 Flash). Changes are relative to full LeanClaw.

Variant	Pass	Acc (%)	Total (M)	Cost (USD)
Full LeanClaw	**489**	**97.8**	16.84	**0.97**
w/o compaction	471	94.2 ↓3.6	16.13	1.08 ↑11.16%
w/o state capsule	468	93.6 ↓4.2	16.03	1.06 ↑9.48%
w/o dedupe	471	94.2 ↓3.6	16.16	1.07 ↑9.91%
w/o tool adapter	473	94.6 ↓3.2	40.64	2.57 ↑164.60%

prior, the expansion part of this overhead is expected to scale with genuinely missing tools rather than with trajectory length, while profile updates grow with the number of steps.

5.5 Components and Generic Compression

Generic compression is not sufficiency-preserving. Table 5 compares LeanClaw with existing efficiency methods on MBPP under DeepSeek V4 Flash. None of the three baselines is sufficient at $\epsilon = 0$: each loses 9–11 passes relative to OpenClaw, so none of them certifies a lower bound. EASYTOOL shortens descriptions to a budget similar to LeanClaw's (17.59M versus 16.84M) but still exposes every tool and loses accuracy: it changes how capabilities are represented, not which are exposed. Trajectory methods (AgentDiet and SWE-Pruner) leave totals within 0.7% of OpenClaw's, which indicates that on this workload the redundancy lies in repeated exposure rather than in long histories. LeanClaw alone reduces tokens and improves accuracy (489 versus 479 passes).

Component ablation. Table 6 removes one component at a time. The two halves of the factorization in Eq. (1) play different roles. Removing the adaptive tool adapter returns total tokens to 40.64M, essentially OpenClaw's 40.97M, so on MBPP the tool component carries almost all of the token savings. Removing any history component (compaction, state capsule, or deduplication) leaves token totals nearly unchanged (16.03–16.16M) but costs 18–21 passes and raises provider cost by 9.48–11.16%, because a larger share of input misses the provider cache. The history mechanisms are thus needed for sufficiency rather than savings: without them, the reduced tool set is no longer paired with the state the model needs to use it.

6 Conclusion

We studied agent efficiency as a question about requests: how much of what a framework sends does the agent need to act correctly? Sufficient exposure shows that excess context costs both tokens and accuracy, and that any empirically sufficient policy bounds the redundant tokens from below. Because requirements surface during execution and errors are asymmetric, exposure must be estimated adaptively and fail open. LeanClaw does so on an unchanged OpenClaw runtime, removing 46.64–59.29% of provider tokens on five benchmarks and 48.15% on RealClawBench while matching or improving accuracy.

AI USE STATEMENT

We used AI-assisted tools for language polishing, code debugging, figure and table preparation, and routine implementation support. All research questions, methodological decisions, experimental configurations, analyses, reported results, and scientific claims were determined, verified, and approved by the authors. The authors take full responsibility for the accuracy and integrity of the manuscript.

ETHICS STATEMENT

This work studies the efficiency of general-purpose language-model agents by reducing redundant capability and context exposure during execution. The experiments use public benchmarks and reproducible benchmark environments for mathematical reasoning, code generation, reading comprehension, and developer-agent tasks. They do not involve human subjects, clinical decision-making, personally identifiable information, or private user data.

LeanClaw relies on API-based language models and executable tools. Such models may produce incorrect actions, insecure code, or inconsistent outputs, while tool execution may modify files or invoke external resources. Reducing the model-visible context may also omit information required by a later action. LeanClaw therefore preserves the complete runtime capability space and uses conservative recovery mechanisms, including exposure expansion, restoration of less-compressed history, and fallback to the original OpenClaw request. Deployments outside controlled benchmark environments should additionally use sandboxed execution, permission controls, task-specific validation, usage monitoring, and safeguards against exposing sensitive information to external model providers.

REPRODUCIBILITY STATEMENT

Section 3 defines sufficiency, sufficient exposure, and the assumptions behind Proposition 1, whose proof is in Appendix A. Section 4 and Algorithm 1 describe the adaptive tool adapter, feedback-driven adaptation, state preservation, history compaction, deduplication, the fail-open gate, and cost accounting. The experiments specify benchmarks, model routes, baselines, outcome counting, provider-token measurement, runtime-failure handling, and ablation settings. Paired OpenClaw–LeanClaw comparisons keep the task files, initial workspace, tool implementations, execution limits, model route, and verifier fixed. Appendix C gives capsule fields and compaction rules, Appendix D describes fail-open recovery and evaluation boundaries, and Appendix E clarifies the separation between the controller and the unchanged runtime.

REFERENCES

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. Re-invoke: Tool invocation rewriting for zero-shot tool retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4705–4726, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, 2019.

Manu Ghulyani, Arunabh Singh, Karan Bharadwaj, Ankit Nath, and Suranjan Goswami. PACMS: Submodular context selection as a pluggable engine for LLM agents, 2026.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. Hi agent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32779–32798, 2025a.

Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, et al. OWL: Optimized workforce learning for general multi-agent assistance in real-world task automation, 2025b.

Zhexin Hu, Li Wang, Xiaohan Wang, Jiajun Chai, Xiaojun Guo, Wei Lin, and Guojun Yin. Zip RL: Adaptive multi-turn context compression with hindsight response replay, 2026.

Hugging Face. smolagents: A barebones library for agents that think in code. `https://github.com/huggingface/smolagents`, 2025. Open-source software.

HuggingFaceH4. AIME 2024 dataset. `https://huggingface.co/datasets/HuggingFaceH4/aime_2024`, 2024.

InclusionAI. AWorld: The agent harness for your world. `https://github.com/inclusionAI/AWorld`, 2025. Open-source agent harness.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 13358–13376, 2023.

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

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25972–25981, 2025a.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing context compression for long-horizon LLM agents, 2025b.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society, 2023.

Zongqian Li, Yixuan Su, and Nigel Collier. 500xcompressor: Generalized prompt compression for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25081–25091, 2025.

Elias Lumer, Faheem Nizar, Anmol Gulati, Pradeep Honaganahalli Basavaraju, and Vamse Kumar Subbiah. Tool-to-agent retrieval: Bridging tools and agents for scalable llm multi-agent systems, 2025.

Elias Lumer, Anmol Gulati, Faheem Nizar, Dzmitry Hedroits, Atharva Mehta, Henry Hwangbo, Vamse Kumar Subbiah, Pradeep Honaganahalli Basavaraju, and James A Burke. Tool and agent selection for large language model agents in production: A survey. In *2026 IEEE Conference on Artificial Intelligence (CAI)*, pp. 701–708. IEEE, 2026a.

Elias Lumer, Anmol Gulati, Vamse Kumar Subbiah, Pradeep Honaganahalli Basavaraju, and James A Burke. Memtool: Optimizing short-term memory management for dynamic tool retrieval and invocation in llm agent multi-turn conversations. In *European Conference on Information Retrieval*, pp. 237–253, 2026b.

Zongwei Lv, Yaoming Li, Zhewen Tan, Yilun Yao, Yuxuan Tian, Lin Sun, Xiangzheng Zhang, Weihong Lin, Tong Yang, and Guangxiang Zhao. Realclawbench: Live openclaw benchmarks from real developer-agent sessions, 2026.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

OpenClaw contributors. Openclaw: A general-purpose developer-agent framework. `https://openclaw.ai`, 2026.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, et al. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression, 2024.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs, 2024.

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

XinQi Wang, Jinwei Xiao, Sijia Cui, Hongming Zhang, Yanna Wang, Qingyang Zhang, and Bo Xu. HyMem: Hierarchical context management for long-horizon agents via information isolation, 2026a.

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 *International Conference on Learning Representations*, 2023.

Yuhang Wang, Yuling Shi, Mo Yang, Rongrui Zhang, Shilin He, Heng Lian, Yuting Chen, Siyu Ye, Kai Cai, and Xiaodong Gu. Swe-pruner: Self-adaptive context pruning for coding agents. *arXiv preprint arXiv:2601.16746*, 2026b.

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. volume 35, pp. 24824–24837, 2022.

Heming Xia, Yongqi Li, Chak Tou Leong, Wenjie Wang, and Wenjie Li. Tokenskip: Controllable chain-of-thought compression in llms. volume 12067.

Yuan-An Xiao, Pengfei Gao, Chao Peng, and Yingfei Xiong. Reducing cost of LLM agents with trajectory reduction. volume 3, pp. 1241–1263. ACM New York, NY, USA, 2026.

Xiaomi MiMo Team. MiMo-V2.5. `https://mimo.xiaomi.com/mimo-v2-5/`, 2026.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts llm agents: An empirical study of experience-following behaviour. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 623–645, 2026.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. volume 38, pp. 17577–17604, 2026.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent–computer interfaces enable automated software engineering, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergising reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Kan Ren, Dongsheng Li, and Deqing Yang. Easytool: Enhancing llm-based agents with concise tool instruction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 951–972, 2025.

Yaolun Zhang, Xiaogeng Liu, and Chaowei Xiao. Metaagent: Automatically constructing multi-agent systems based on finite state machines. *arXiv preprint arXiv:2507.22606*, 2025.

He Zhu, Tianrui Qin, King Zhu, Heyuan Huang, Yeyi Guan, Jinxiang Xia, Hanhao Li, Yi Yao, Ningning Wang, Pai Liu, et al. Oagents: An empirical study of building effective agents., 2025.

A PROOFS

Proposition 1, fixed exposure. Under a fixed policy, the tools ever exposed are exactly $\mathcal{E}_0$. If $R(x) \not\subseteq \mathcal{E}_0$, some required tool is never exposed, and by (A1) the task fails. Hence the probability of success is at most $\Pr[R(x) \subseteq \mathcal{E}_0]$, and since A is the expected performance with values in $[0, 1]$, $A(\phi_{\text{fixed}}) \leq \Pr[R(x) \subseteq \mathcal{E}_0]$. □

Proposition 1, adaptive exposure. Fix a task x and let $M = R(x) \setminus \mathcal{E}_0$ be the required tools missing from the prior. A tool in $R(x) \cap \mathcal{E}_0$ is exposed from the start and, since required tools are never withdrawn, remains exposed. For each $r \in M$, let F_r be the event that the miss of r is not signaled before the budget is exhausted; by (A2), $\Pr[F_r] \leq \eta$. If F_r does not occur, the expansion rule adds r to Δ_t at the signaled step, and since required tools are never withdrawn, r remains exposed afterward. Thus every required tool is eventually exposed within the budget unless some F_r occurs. By the union bound, this happens with probability at most $\eta|M|$, and taking expectation over x gives the coverage bound $1 - \eta\,\mathbb{E}|R(x) \setminus \mathcal{E}_0|$. For the count, each recovery expansion adds a required tool in M that is not yet exposed (a signaled miss is by definition outside $\mathcal{E}_t$), and since required tools are never withdrawn, each tool in M is added at most once. Hence there are at most $|M| = |R(x) \setminus \mathcal{E}_0|$ recovery expansions, and $\mathcal{E}_t \subseteq \mathcal{G}$ throughout. □

Coverage is necessary but not sufficient for success. The proposition isolates the failure mode that adaptation removes, namely missing capabilities, and leaves the rest of task difficulty unchanged.

B CASE STUDY: EXPOSED VERSUS USED TOOLS

We inspect cases from the 40-case RealClawBench diagnostic subset that pass under both OpenClaw and a tool-selection-only configuration of LeanClaw. Although these tasks span data queries, code fixes, and file creation, each successful trajectory uses only one to three tool types, while OpenClaw exposes all 38 schemas in every provider request.

Case `30d14f6f99b18cbe`, for example, asks the agent to inspect `conversation-runtime.ts`, list its top-level exports, and verify several package export paths. The successful trajectory only reads the repository. OpenClaw exposes all 38 schemas across five requests, 77,900 schema tokens in total, whereas LeanClaw exposes six schemas across two requests, 3,252 schema tokens in total, and both runs pass. Unused tools remain executable in both systems; only OpenClaw repeatedly represents them in the provider request.

Table 7 reports the same pattern for five cases. Per-request exposure falls from 38 schemas to six or seven, and total schema tokens fall by 94.5–95.8%. Per request (89.6% for `30d14f`), the reduction exceeds the drop in schema count (84.2%) because the removed schemas are longer on average; the total reduction is larger still because LeanClaw issues fewer requests. These examples are a mechanism check, not an accuracy evaluation.

Table 7: Five RealClawBench cases that pass under both OpenClaw and LeanClaw with tool selection only. Schema counts are per request; the reduction is in schema tokens.

Case	Task type	Used tools	Schema exposure		
			OpenClaw	LeanClaw	Token reduction
`30d14f`	data query	read	38	6	↓95.8%
`259d7d`	code fix	edit/read	38	6	↓95.5%
`0a2064`	data query	exec/write	38	6	↓95.4%
`e2fe35`	code fix	exec/write	38	6	↓94.8%
`3a25e0`	file create	exec/read/write	38	7	↓94.5%

C Context Capsule and History Compaction

LeanClaw maintains a structured state capsule rather than forwarding the full interaction history at every request. The capsule contains the current task status, completed checkpoints, inspected and modified files, executed commands, test outcomes, unresolved errors, discovered artifacts, and verifier-relevant constraints. It is updated after each tool observation:

$$z_{t+1} = \mathrm{UpdateState}(z_t, a_t, o_t, e_t). \tag{6}$$

For request t, the compact history is assembled as

$$\widetilde{h}_t = \mathrm{Recent}_k(h_t) \cup \mathrm{Evidence}_t, \tag{7}$$

and is sent together with the capsule z_t (Eq. (2)). Recent turns remain verbatim, z_t preserves structured execution state, and $\mathrm{Evidence}_t$ stores exact content that cannot be safely reconstructed. Evidence includes file paths, identifiers, JSON keys, numeric values, error messages, command arguments, and strict output requirements.

History compaction is applied only to low-value historical content. Repeated observations or messages are removed when they add no new evidence. The original task, current observation, tool-call arguments, and exact verifier constraints remain intact. If the controller cannot determine whether an item is safe to compress, it preserves the item or falls back to a more conservative request.

D Recovery and Evaluation Boundaries

Exposure control is fail-open. If the model requests a capability outside the currently exposed set, or if execution feedback reveals a missing dependency, the controller expands the next exposure set and retries. If history compression produces an uncertain or inconsistent state, LeanClaw restores less-compressed history. For exact-copy requirements, strict output formats, unresolved tool errors, or low-confidence selection, the controller reverts to a conservative request as in Eq. (5).

These recovery paths preserve access to the complete runtime capability space. They also define the evaluation boundary: the task files, seeds, initial workspace, tool implementations, execution limits, model route, and verifier are held fixed. Runtime-failed cases are rerun under the same configuration for both methods, and a case passes only if execution and verification both succeed. Token and cost measurements are taken from provider or proxy usage rather than from diagnostic attribution buckets.

E Implementation Boundary

LeanClaw is implemented as a request-level layer around the OpenClaw execution loop. The controller observes outgoing provider requests, selects the request-local tool and instruction representation, updates structured execution state, and records provider usage. The underlying OpenClaw runtime continues to own tool invocation, workspace interaction, permissions, model routing, artifact production, and final verification.

Consequently, omitted capabilities are not deleted from OpenClaw. They remain installed and executable, and can be exposed in a later request when task feedback or dependency discovery makes them relevant. This separation is what allows the experiments to attribute the measured savings to request-level exposure control rather than to a reduced tool inventory or a modified benchmark.