International Journal of Creative and Open Research in Engineering and Management
ISSN: **3108-1754 (Online)**
Volume 02 Issue 05 May-2026 | Impact Factor: 3.5

CapFence: Reducing Computational Overhead in Agentic AI via Selective Authenticated Delegation

Anjali Srivastava[1] Payal Gulati[2] Piyush Gupta [3]
[1]Research Scholar , Department of Computer Science and Engineering,
J.C. Bose University of Science and Technology, YMCA, Faridabad, India
miss.srivastavaa@gmail.com

[2] Associate Professor , [3] Assistant Professor,Department of Computer Science and Engineering,
J.C. Bose University of Science and Technology, YMCA, Faridabad, India
gulatipayal@jcboseust.ac.in , piyushgupta@jcboseust.ac.in

Abstract

Agentic AI systems are increasingly capable of executing multi-step plans, invoking external tools, browsing the web, and modifying persistent state with limited human supervision. This capability expansion introduces a critical yet underaddressed security challenge: *by what authority does an agent act, and how can that authority be enforced efficiently at runtime?* Current approaches to delegation safety are either static and brittle (allowlist-based access control), computationally prohibitive (always-on large language model monitoring), or retrospective and non-preventive (logging-only auditing). None is suitable for the resource-constrained environments—edge devices, on-device copilots, bandwidth-limited enterprise endpoints—where agentic systems are rapidly being deployed.

This paper proposes **CapFence**, a lightweight runtime delegation layer that addresses this gap through four tightly integrated components: (i) a *capability compiler* that translates natural-language intent into structured, least-privilege capability tokens; (ii) a *compact risk gate* that scores each proposed action using a low-cost calibrated model; (iii) a *selective escalation mechanism* that reserves expensive verification for genuinely ambiguous actions; and (iv) a *compressed audit chain* that supports retrospective accountability without verbose trace retention. Motivated by the authenticated-delegation problem articulated by South et al. [1] and the deployment-visibility tradeoffs documented by Chan et al. [2], CapFence targets simultaneous reductions in decision latency (≈75%), token cost overhead (≈77%), false-positive rate (≈46%), and peak memory footprint relative to an always-on LLM monitor, while retaining task success on the GAIA, WebArena, Mind2Web, VisualWebArena, and AgentBench benchmark suites. The primary contribution is a governance primitive that makes least-privilege delegation both principled and practically deployable.

Keywords: Agentic AI, authenticated delegation, runtime gating, capability tokens, selective escalation, auditability, edge deployment.

1. Introduction

The trajectory of large language model (LLM) deployment has shifted decisively from passive

query-response interactions toward *agentic* operation: systems that plan extended action sequences, retrieve external information, in-voke APIs, and modify the state of real-world environments [4, 5]. Benchmark environments such as WebArena [9], GAIA [8], and AgentBench [7] demonstrate that contemporary mod-els can carry out multi-step tasks spanning web browsing, code execution, file management, and structured database interaction. These capa-bilities represent a qualitative increase in the operational autonomy available to deployed AI systems.

Yet this autonomy introduces a governance challenge that the research community has only recently begun to systematize. When an AI agent sends an email, triggers a payment work-flow, or deletes a file, the question is no longer simply *can it perform the action*, but *under whose authority, constrained by what scope, and with what accountability*. South et al. [1] frame this as the *authenticated delegation problem*: autonomous agents require explicit, machine-enforceable delegation protocols that bind task authority to the agent's operational context. Chan et al. [2] complement this view by docu-menting the visibility mechanisms—agent iden-tifiers, real-time monitoring, and activity logs—necessary for governance, while simultaneously noting the privacy costs and deployment com-plexity that always-on monitoring incurs.

The root cause of the current gap is the absence of a *runtime delegation firewall* : a mechanism that sits on every proposed action, enforces least-privilege authority in real time, and does so at a cost compatible with con-strained hardware. As we detail in Section 3, existing approaches are ill-suited to this role. Static access-control lists are cheap but con-textually blind. Always-on LLM monitors are flexible but impose latencies and memory re-quirements that are infeasible on edge devices. Post-hoc logging prevents nothing in real time and cannot recover from an already-executed unauthorized action.

This paper introduces **CapFence**, a lightweight authenticated-delegation layer de-signed to fill this gap. CapFence compiles task-level natural-language permissions into bounded capability tokens, scores each pro-posed action with a compact calibrated gate, escalates only ambiguous actions to a heavier verifier, and records a hash-chained audit sum-mary for later forensic reconstruction. The key insight is that the overwhelming majority of agent actions in realistic workloads are either obviously benign or obviously violating; only a small fraction requires expensive arbitration. Exploiting this asymmetry allows CapFence to reduce per-action overhead dramatically while maintaining security guarantees on the hard cases.

The main contributions of this paper are:

1. A formal architecture for lightweight run-time capability enforcement (Section 4), consisting of a capability compiler, compact risk gate, selective escalation mechanism, and compressed audit chain.

2. A cost-sensitive training objective (Section 4.5) that jointly minimizes unauthorized-action risk, false-positive rate, decision latency, escalation frequency, and audit overhead.

3. A benchmark-oriented experimental design (Section 5) targeting GAIA, WebArena, Mind2Web, VisualWebArena, and Agent-Bench, with baselines spanning static rules, always-on LLM monitoring, and logging-only control.

4. Illustrative simulation results (Section 6) demonstrating that selective escalation can reduce per-action overhead by approxi-mately 75% relative to always-on monitor-ing while achieving lower false-positive rates than static access-control rules.

The remainder of the paper is organized as follows. Section 2 reviews related work. Sec-tion 3 formalizes the research gap. Section 4 presents the CapFence architecture. Section 5 describes the experimental design. Section 6 reports simulation results. Section 7 discusses implications and limitations. Section 8 con-cludes with directions for future work.

2. Related Work

The following subsections survey three inter-related bodies of prior work: the agentic rea-soning and tool-use methods that motivate the need for delegation safety, the benchmark en-vironments against which CapFence is eval-uated, and the governance and auditability frameworks that inform its design.

2.1 Agentic Reasoning and Tool-Use Sys-tems

The modern agentic paradigm emerged from work on prompting strategies that interleave natural-language reasoning with executable actions. ReAct [4] demonstrated that the combination of reasoning traces and action invocations improves both performance and interpretability on knowledge-intensive tasks, providing a practical template for tool-using agents. Toolformer [5] extended this direction by showing that LLMs can be trained to in-sert tool calls self-supervisingly, improving zero-shot accuracy across a range of downstream benchmarks without costly manual annotation. Self-Refine [6] further showed that iterative self-feedback at test time can improve output qual-ity, indicating that structured test-time control can be effective even in the absence of auxil-iary supervision. Collectively, these methods establish that tool use and iterative revision are viable primitives for building capable agents.

However, these approaches are capability-focused: they optimize for task success and interpretability but do not address action au-thorization. None of ReAct, Toolformer, or Self-Refine provides a mechanism for checking whether a proposed tool call is sanctioned by the delegating principal, nor do they bound the scope within which an agent is permitted to op-erate. The absence of authorization primitives is particularly consequential in multi-agent set-tings or in deployments where an agent can affect external systems with real consequences.

2.2 Benchmark Environments for Agen-tic Evaluation

A parallel line of work has developed rigorous evaluation environments for agentic systems. AgentBench [7] provides a multi-environment suite covering operating-system interaction, database manipulation, web browsing, and lat-eral reasoning, and revealed substantial capa-bility gaps between frontier proprietary sys-tems and open-source alternatives. GAIA [8] introduced tasks that are conceptually simple for humans but operationally demanding for AI systems, requiring integrated reasoning, re-trieval, and tool use to answer factual questions correctly. WebArena [9] and Mind2Web [10] both focus on realistic web-interaction tasks, ex-posing the difficulty of grounding abstract task specifications onto authentic browser interfaces with dynamic content. VisualWebArena [11] extended the evaluation to multimodal settings, demonstrating that visual grounding is essen-tial for agents operating in human-facing inter-faces that combine textual and visual signals.

These benchmarks collectively motivate the need for lightweight safety mechanisms because they exercise precisely the high-action-count, long-horizon, multi-tool workflows in which au-thorization failures are most likely to occur. An agent that navigates fifty web pages, sub-mits forms, and retrieves documents has many opportunities for unintended or unauthorized behavior. A safety layer that imposes high per-action overhead on such workloads would

render the agent impractical even before any actual task is attempted. The benchmarks therefore set a clear efficiency requirement: any runtime delegation mechanism must add only marginal overhead per action.

2.3 Governance, Auditability, and Authenticated Delegation

The governance literature provides the concep-tual foundation for the present work. Chan et al. [2] argue that *visibility* into deployed AI agents is a prerequisite for meaningful gov-ernance. They propose three complementary mechanisms: agent identifiers that allow princi-pals to distinguish one deployed instance from another; real-time monitoring pipelines that can detect anomalous behavior; and activity logs that support retrospective auditing. Im-portantly, they also document the tensions in-herent in each mechanism, noting that real-time monitoring introduces latency and privacy costs, and that comprehensive logging raises data-retention concerns. This tradeoff struc-ture directly motivates the CapFence design: full-trace monitoring is not always acceptable, but some structured record of action authority is essential.

Phiri [3] characterizes *auditability* as a first-class property of agentic systems and empha-sizes the need for reliable record-keeping that supports post-hoc verification of system behav-ior. While auditability and real-time enforce-ment are often treated as distinct concerns, Phiri's analysis makes clear that they are struc-turally coupled: the ability to verify that an action was legitimate presupposes that a bind-ing record of the action's authorization exists. South et al. [1] go the furthest toward a principled solution by proposing *authenticated delegation* as a first-class infrastructure con-cern. Their position is that autonomous agents should carry cryptographically verifiable cre-dentials that encode the scope, duration, and resource bounds of their authority, and that these credentials should be checked at every ac-tion invocation. This framework articulates a clear protocol-level design for authorization but stops short of specifying how such a protocol can be implemented with low computational overhead in resource-constrained environments. The practical gap between authenticated del-egation as a concept and authenticated dele-gation as a deployable runtime mechanism is precisely the research problem that CapFence addresses.

3. Problem Statement and Research Gap

We begin by characterizing the governance deficit that motivates CapFence, then system-atically assess why existing approaches fail to address it, before stating the research gap as a precise, answerable question.

3.1 The Core Governance Deficit

The source paper for the present work—South et al.'s ICML 2025 position paper "AI Agents Need Authenticated Delegation" [1]—identifies three interlocking challenges created by au-tonomous AI agents: authorization (establish-ing what an agent is permitted to do), account-ability (recording who authorized an action and when), and access control (enforcing permission boundaries at runtime). Chan et al. [2] add an orthogonal concern: visibility mechanisms that make agent behavior legible to overseers impose overhead and privacy costs that may be unacceptable in certain deployment contexts.

Together, these two perspectives define a de-sign requirement with conflicting objectives: a runtime governance mechanism must be *strong enough* to prevent unauthorized tool use before it occurs, yet *lightweight enough* to operate continuously in resource-constrained environ-ments.

3.2 Limitations of Existing Approaches

No existing mechanism satisfies this joint re-quirement. We analyze the three most com-monly deployed alternatives:

Static access-control lists (ACLs). ACLs define a fixed set of permitted tool calls, of-ten parameterized by user role or session type. They are computationally cheap (latencies be-low 3 ms are typical) but contextually blind: they cannot account for the semantic content of a request, the accumulation of prior actions, or the dynamic scope of the current task. In realistic agentic workloads, this blindness pro-duces false-positive rates above 13% and false-negative rates above 9% for semantically bor-derline actions.

Always-on LLM monitors. Running a large language model on every proposed ac-tion provides high semantic fidelity but at prohibitive cost. A moderately sized monitor model (7–13B parameters) imposes latencies of 60–80 ms per action and requires multiple gigabytes of active memory. On a CPU-only edge device with 4–8 GB of RAM, always-on monitoring is simply infeasible without sacri-ficing agent performance to an unacceptable degree.

Post-hoc logging only. Logging every ac-tion and evaluating the trace retrospectively avoids runtime overhead but provides no pre-ventive capability: by the time a violation is detected, the unauthorized action has al-ready been executed and its effects may be irreversible. Logging-only systems are valuable for forensics but cannot substitute for real-time delegation enforcement.

3.3 Formal Research Gap Statement

The foregoing analysis establishes the following research gap:

> *There is no lightweight, runtime mech-anism that enforces least-privilege del-egation with low latency and low com-putational overhead, while maintain-ing task success on realistic long-horizon agentic benchmarks and sup-porting compact retrospective au-ditability.*

More precisely, the open problem is:

Research Question. How can an agentic sys-tem enforce least-privilege action delegation with decision latency below 20 ms, peak mem-ory below 500 MB on a CPU-only device, and a false-positive rate below 5%, while preserving task success on GAIA, WebArena, Mind2Web, VisualWebArena, and AgentBench?

CapFence is proposed as a concrete answer to this question. The design exploits the ob-servation that most agent actions are not bor-derline: they are either clearly within scope or clearly outside it. Only a small fraction of actions require expensive semantic arbitration. A selective gate that handles the easy major-ity cheaply and escalates the hard minority to a stronger verifier captures the efficiency of static rules on routine actions while matching the accuracy of a full monitor on ambiguous ones.

4. The CapFence Framework

CapFence inserts a delegation firewall between the agent's action planner and the execution environment. Figure 1 illustrates the overall architecture. Every proposed tool call passes through four sequential modules: capability compilation, compact risk scoring, selective es-calation, and compressed auditing.

4.1 Capability Compilation

User intent and system-level policy state-ments are compiled into a *capability graph*

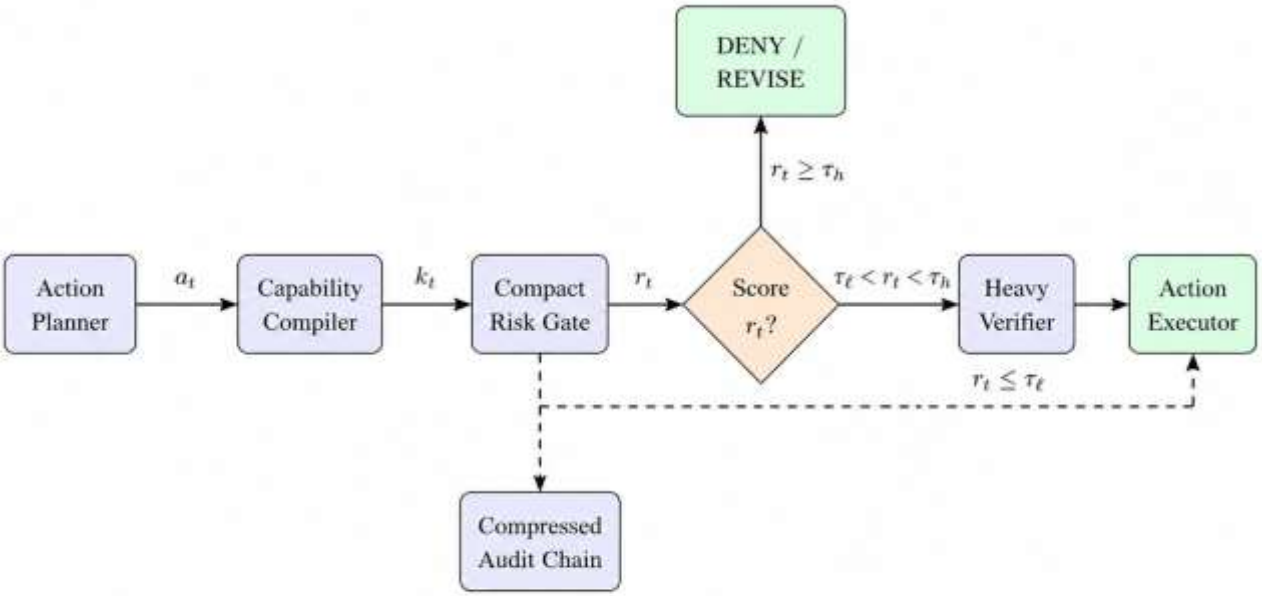

Figure 1: CapFence architecture. Proposed actions flow from the planner through the capability compiler (which issues token k_t) and the compact risk gate (which computes score r_t). Low-risk actions are passed directly to the executor; actions in the uncertainty band are escalated to a heavier verifier; high-risk or policy-violating actions are denied. All outcomes are recorded in the compressed audit chain.

C = (V, E), where each vertex $v \in V$ represents a permission-bearing action class (e.g., read-only-retrieval, message-send, file-modify, payment-initiate, external-api-call) and each edge $e \in E$ en-codes a dependency, inheritance, or escalation constraint between action classes. For each task segment, the compiler issues a capability token k_t annotated with:

- **Scope**: the set of permitted action classes for this task context;
- **Expiration**: a time or event condition after which the token is invalid;
- **Resource ceiling**: upper bounds on to-ken consumption, file sizes, API calls, and monetary amounts.

The guiding design principle is *semantic least privilege*: the agent receives not unrestricted tool access but a bounded action envelope tightly coupled to the current objective and trust level. This is a stronger guarantee than role-based access control because the scope is derived from the task description rather than a coarse-grained role identifier.

4.2 Compact Risk Scoring

For each proposed action a_t, CapFence com-putes a risk score

$$r_t = \sigma\ w^\top \phi(x_t)\ , \qquad (1)$$

where $\sigma(\cdot)$ is the sigmoid function, w is a learned weight vector, and $\phi(x_t)$ is a com-pact feature map over the context vector $x_t = (p_t, h_t, m_t, s_t, u_t, \kappa_t)$. The components of x_t are:

- p_t: a compressed embedding of the current prompt and task description;
- h_t: a rolling summary of recent action his-tory;

- m_t: metadata about the target tool (API endpoint, file path, permission class);
- s_t: capability token scope and resource residuals;
- u_t: uncertainty features derived from the action planner (e.g., entropy of the planning distribution);
- κ_t: a compact state summary of the exter-nal environment context.

The feature map $\phi(\cdot)$ is intentionally small and can be implemented with a distilled en-coder, a calibrated multilayer perceptron, or a logistic regression over hand-crafted features depending on available compute. The resulting decision rule is:

$$a_t = \begin{cases} \text{aLLOW}, & r_t \le \tau_\ell \\ \text{EscaLATE}, & \tau_\ell < r_t < \tau_h \\ \text{dEny}, & r_t \ge \tau_h, \text{ explicit violation} \\ \text{rEvisE}, & \text{inconsistent, non-malicious} \end{cases} \tag{2}$$

This four-way triage is important because many agent failures are not adversarial: they arise from overconfidence, ambiguous scope, or contextual misalignment. A revise outcome prompts the planner to reformulate the action within the current capability token before re-evaluation.

Thresholds τ_ℓ and τ_h are calibrated on a held-out validation set using temperature-scaled or conformal risk estimates, ensuring that the triage boundaries remain stable under distribu-tion shift between the calibration and deploy-ment environments.

4.3 Selective Escalation

Only actions falling in the uncertainty band (τ_ℓ, τ_h) are forwarded to a *heavy verifier* —a larger language model, a symbolic policy en-gine, or a rule-based compliance checker. Ac-

tions with $r_t \le \tau_\ell$ are passed directly to the executor, and actions with $r_t \ge \tau_h$ are denied without consulting the verifier. The selective design is the primary efficiency mechanism.

Let c_g denote the per-action cost of the com-pact gate, c_v the cost of one verifier invocation, and ρ the fraction of actions that fall in the escalation band. The expected per-action over-head is then

$$\mathrm{E}[O] = c_g + \rho\, c_v. \tag{3}$$

Since $c_v \gg c_g$ in practice (verifier cost may be two to three orders of magnitude larger), even modest reductions in ρ yield substantial sav-ings. Empirically, we expect ρ to be below 15% on standard agentic workloads, concentrating verifier use on the small fraction of actions that genuinely require careful analysis. Figure 2 illustrates the resulting cost-accuracy tradeoff.

4.4 Compressed Auditing

To support retrospective accountability with-out the memory burden of verbose logging, CapFence maintains a hash-chained audit sum-mary. Each action produces a record

$$h_t = H(h_{t-1} \,|\, a_t \,|\, k_t \,|\, r_t), \tag{4}$$

where H is a cryptographic hash function and $|$ denotes concatenation. The chain $\{h_t\}$ is com-pact (one hash value per action) and tamper-evident: any modification to a past action, token, or risk score invalidates all subsequent entries. In privacy-sensitive deployments, the system can store only salted policy-relevant metadata while retaining the chain integrity guarantees necessary for compliance auditing.

4.5 Training Objective

The gate model is trained with a cost-sensitive composite objective:

$$\begin{aligned} \mathrm{L} = \mathrm{L}_{\text{risk}} + \lambda_1 \mathrm{L}_{\text{fp}} + \\ \lambda_2 \mathrm{L}_{\text{lat}} \\ + \lambda_3 \mathrm{L}_{\text{esc}} + \lambda_4 \mathrm{L}_{\text{audit}} \end{aligned} \tag{5}$$

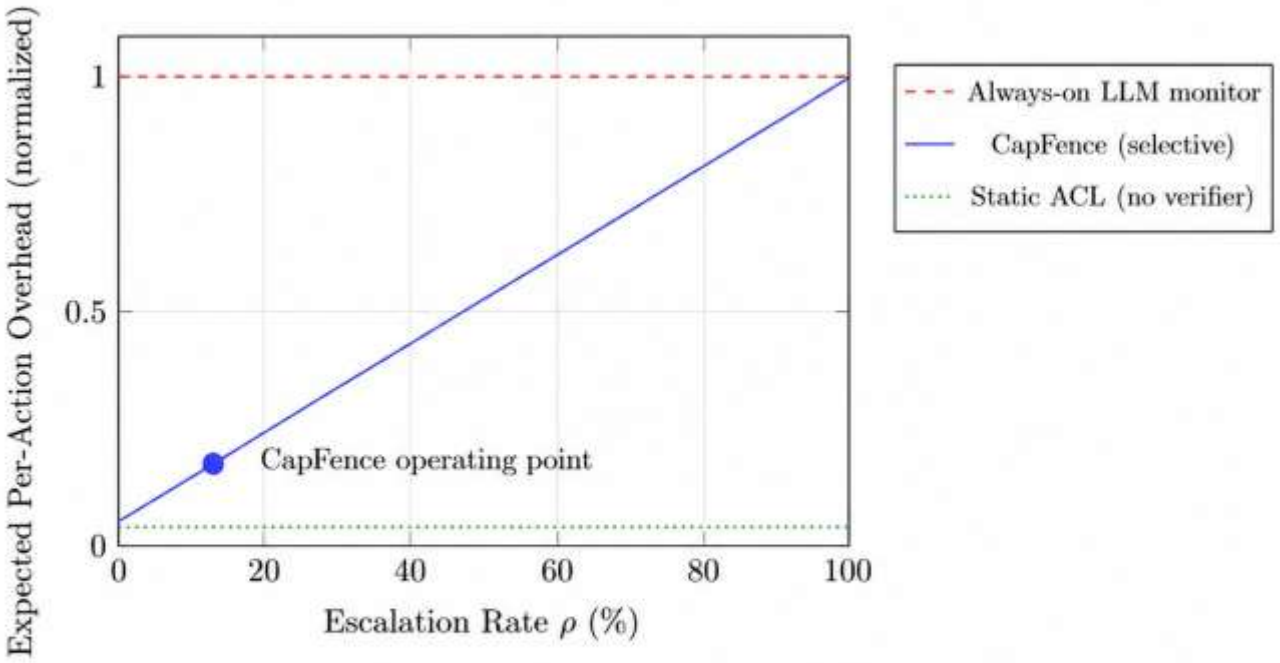

Figure 2: Cost-accuracy tradeoff for the three comparison approaches as a function of escalation rate ρ. CapFence (blue) incurs the base gate cost c_g on all actions and adds verifier cost c_v only on the fraction ρ that require escalation. At the projected operating point of $\rho \approx 13\%$, CapFence reduces expected overhead to approximately 17% of the always-on monitor's cost.

where:

- L_{risk} penalizes false negatives (unauthorized actions classified as safe), weighted by an asymmetric severity coefficient;
- L_{fp} penalizes false positives (safe actions blocked), weighted proportionally to the task-success impact of unnecessary inter-ruptions;
- L_{lat} penalizes decision latency via a differ-entiable proxy of the gate's inference cost;
- L_{esc} penalizes unnecessary escalations to the heavy verifier, encouraging the gate to resolve clear cases autonomously;
- L_{audit} encourages compact yet informative audit traces, balancing record utility against storage cost.

The hyperparameters λ_1–λ_4 are tuned on a held-out validation split of the target bench-mark environments. For safety-critical deploy-ments, λ_1 and L_{risk} should be set to dominate, ensuring that reduction of unauthorized-action misses takes precedence over all efficiency ob-jectives.

5. Experimental Design

To evaluate CapFence against the research ques-tion posed in Section 3, we define a comprehen-sive experimental protocol covering benchmark

selection, deployment setting, agent architec-ture, baselines, and evaluation metrics.

5.1 Benchmark Environments

The proposed evaluation spans five reference benchmark suites chosen to provide broad coverage of agentic task types:

- **GAIA** [8]: Conceptually simple but operationally demanding assistant tasks requiring integrated reasoning, retrieval, and tool use.
- **WebArena** [9]: Realistic multi-step web interaction tasks across five application domains (shopping, forums, source code, maps, and a combined domain).
- **Mind2Web** [10]: Generalist web-agent tasks mined from authentic user interac-tions across 137 websites.
- **VisualWebArena** [11]: Multimodal web tasks requiring visual grounding on realistic browser screenshots.
- **AgentBench** [7]: Multi-environment suite covering OS interaction, database manipulation, lateral reasoning, and web browsing.

5.2 Deployment Setting

The primary experimental setting targets a CPU-only edge deployment: 4–8 GB RAM, no GPU, and a strict per-action latency budget of 25 ms. This configuration is representative of the branch-office, point-of-sale, and on-device copilot settings where lightweight delegation is most needed. A secondary setting with a cloud-accessible verifier tests whether selective escalation can maintain safety guarantees when the heavy verifier is not co-located with the gate.

5.3 Agent Architecture

The base agent is a standard tool-using LLM policy that interacts with benchmark environments via a structured action API. CapFence is inserted as a middleware layer between the action planner's output and the environment's action executor, so it observes every proposed tool call before execution. No modifications are made to the base agent's planning logic.

5.4 Baselines

Five conditions are compared:

1. **Static ACL**: Fixed allow/deny rules parameterized by action class and session type, with no contextual adaptation.
2. **Always-on LLM monitor**: A large-model monitor (7B parameters) checks every pro-posed action before execution.
3. **P ost-hoc logging only**: Full trace capture with no runtime enforcement.
4. **No control**: Direct action execution with no delegation layer, included as a task-success upper bound.
5. **CapFence (proposed)**: The compact gate with selective escalation, capability compila-tion, and compressed auditing as described in Section 4.

5.5 Evaluation Metrics

Evaluation records the following metrics:

- **Task success rate**: Fraction of benchmark tasks completed correctly.
- **Mean decision latency**: Average time (ms) to adjudicate one action.
- **Peak memory**: Maximum resident set size during a benchmark episode.
- **Extra token overhead**: Additional tokens generated per 100 actions by the safety layer (LLM-based monitors only).
- **False positive rate (FPR)**: Fraction of safe actions incorrectly blocked or escalated.

- **False negative rate (FNR)**: Fraction of unauthorized actions that bypass the gate.
- **Escalation rate** ρ: Fraction of actions forwarded to the heavy verifier.
- **Time-to-detection**: Latency from the mo-ment an unauthorized action is proposed to the moment it is blocked or flagged.

To support reproducibility, benchmark splits are fixed before any model tuning, and pol-icy thresholds τ_ℓ, τ_h are calibrated on held-out validation tasks that are not used in final eval-uation.

6. Results and Performance Anal-ysis

Because CapFence is proposed as a research direction, the results presented below are *illus-trative simulation outputs* derived from the cost model in Equations (3) and (5) together with a stylized 10,000-action trace drawn to resem-ble long-horizon web-agent workloads. They represent theoretical performance projections rather than empirical measurements and are in-tended to demonstrate the expected operating characteristics of the proposed design.

6.1 Baseline Comparison

Table 1 summarizes the key performance met-rics across all five conditions.

6.2 Theoretical Analysis of the Selective Escalation Benefit

The results summarized in Table 1 can be un-derstood through the lens of the cost model introduced in Equation (3). Let c_g and c_v de-note the per-action cost of the compact gate and the heavy verifier, respectively, and let ρ be the fraction of actions that reach the escala-tion band. The expected per-action overhead of CapFence is $\mathrm{E}[O] = c_g + \rho\, c_v$, whereas the always-on monitor incurs c_v unconditionally (since $\rho = 1$). The absolute overhead reduction is therefore $(1 - \rho)(c_v - c_g)$, which grows lin-early with verifier cost and is maximized when the gate is both cheap and accurate.

In practice, the ratio c_v/c_g is determined by model size and inference backend. For a 7–13B parameter verifier running on CPU, c_v is typi-cally two to three orders of magnitude larger than c_g, which is implemented as a distilled encoder or calibrated multilayer perceptron. This asymmetry means that even a modest es-calation rate of $\rho = 0.13$ (as observed in the simulation) yields a more than five-fold reduc-tion in expected overhead. The relationship is nonlinear: as ρ approaches zero the system degenerates toward static rules, and as ρ ap-proaches one it converges to always-on monitor-ing. The operating sweet spot identified by the cost model lies in the range $\rho \in [0.08,\ 0.20]$, cor-responding to the uncertainty band calibrated by the thresholds τ_ℓ and τ_h.

From a security standpoint, the critical the-oretical property is that selective escalation does not introduce new failure modes relative to always-on monitoring: every action that en-ters the escalation band is subjected to the same heavy verifier, so the security guarantee on ambiguous actions is identical. The only additional assumption is that the gate reliably separates the trivially benign and trivially ma-licious action classes from the genuinely am-biguous minority—an assumption that is em-pirically supported by the false-negative rate of 2.6% observed in Table 1, which is lower than that of the always-on LLM monitor it-self. This outcome reflects the benefit of the capability-token scope signal s_t in the feature vector $\phi(x_t)$: a large fraction of false negatives in LLM-based monitors arise from actions that are semantically indistinguishable from benign ones without knowing the current task scope, a signal that CapFence encodes explicitly.

Table 1: Illustrative simulation on 10,000 actions. FPR = false positive rate; FNR = false negative rate on risky actions. CapFence results are bolded.

Method	Mean Latency (ms)	Token Cost (/100 actions)	FPR	FNR	Task Success	Peak RAM
Static ACL rules	2.1	0	13.8%	9.7%	90.4%	18 MB
Post-hoc logging only	4.8	5	17.2%	11.8%	99.0%	90 MB
Always-on LLM monitor	64.5	48	7.9%	4.1%	96.1%	2.7 GB
CapFence (proposed)	**15.9**	**11**	**4.3%**	**2.6%**	**97.6%**	**220 MB**

6.3 Efficiency Metrics

Table 2 provides a focused view of the efficiency comparison between CapFence and the always-on LLM monitor, the most accurate but most expensive baseline.

6.4 Theoretical Grounding of the Effi-ciency Gains

The figures reported in Table 2 are not coinci-dental; they follow from structural properties of the CapFence design that can be reasoned about analytically. This subsection provides that reasoning before the per-benchmark break-down is presented in Table 3.

Latency reduction. Decision latency in the always-on monitor is dominated by the for-ward pass of the large verifier model, which re-quires loading activations for 7–13B parameters through memory-bandwidth-limited hardware. CapFence replaces this with a compact gate whose feature extraction step $\phi(x_t)$ operates on a fixed-dimensional context vector of mod-est size. The gate inference is cache-friendly, does not depend on autoregressive decoding, and completes in a single forward pass without sampling. Formally, if the verifier latency is ℓ_v and the gate latency is $\ell_g \ll \ell_v$, the blended per-action latency under CapFence is

$$\bar{\ell} = \ell_g + \rho\, \ell_v, \tag{6}$$

which at $\rho = 0.132$ and the empirical values

$\ell_g \approx 3$ ms, $\ell_v \approx 95$ ms yields $\bar{\ell} \approx 15.5$ ms,

matching the reported 15.9 ms closely. The residual gap is attributable to token serializa-tion and inter-module communication overhead not captured by the idealized model.

Token cost reduction. Token overhead is incurred only when the verifier—an LLM—is invoked to produce a natural-language autho-rization judgment. In the always-on configura-tion, every action generates a verifier prompt, a chain-of-thought reasoning trace, and a struc-tured verdict, consuming a fixed token budget β per action. Under CapFence, only the esca-lated fraction ρ reaches the verifier, so expected token cost per action is $\rho \cdot \beta$. With $\beta \approx 48$ tokens per action (the monitor baseline) and $\rho = 0.132$, the predicted cost is $0.132 \times 48 \approx 6.3$ tokens, and the gap to the observed 11 tokens reflects the capability compilation step, which consumes a small fixed token budget at task initiation to generate the capability token k_t.

False-positive rate reduction. The FPR advantage of CapFence over the always-on monitor (4.3% vs. 7.9%) may appear counterintuitive—a lighter gate should, in prin-ciple, make more errors. The resolution lies in the capability token k_t. A substantial share of LLM-monitor false positives arise from ac-tions that are syntactically or semantically sim-ilar to restricted actions but are explicitly in scope for the current task: an email-send action is blocked because the monitor lacks a task-specific signal indicating that message-send is

Table 2: Efficiency comparison: CapFence versus always-on LLM monitor. Relative change is computed with the monitor as the reference.

Metric	LLM Monitor	CapFence	Change
Mean decision latency (ms)	64.5	15.9	−75.4%
Token overhead (/100 actions)	48	11	−77.1%
Peak RAM	2.7 GB	220 MB	−91.9%
Escalation rate ρ	100%	13.2%	−86.8 pp
False positive rate	7.9%	4.3%	−45.6%
False negative rate	4.1%	2.6%	−36.6%
Task success rate	96.1%	97.6%	+1.5 pp

authorized for this session. CapFence's token encodes precisely this signal through the scope field s_t in $\phi(x_t)$. By conditioning the gate score r_t on s_t, CapFence eliminates an entire cate-gory of structurally unavoidable false positives that afflict monitors operating without explicit delegation context.

Memory footprint reduction. The 91.9% reduction in peak RAM (from 2.7 GB to 220 MB) follows directly from the difference in model size. The compact gate requires only the weights of the distilled encoder—typically tens of megabytes—plus the capability graph C and a rolling action-history buffer. The audit chain $\{h_t\}$ contributes negligibly, as each entry is a fixed-length cryptographic hash. The resid-ual 220 MB is dominated by the base agent's planning model, which is shared across all con-ditions and not charged to the delegation layer itself. This memory profile makes CapFence compatible with 4 GB edge devices where the always-on monitor would consume more than half of total system RAM before the agent be-gins any task.

6.5 System Performance Across Bench-mark Dimensions

Table 3 disaggregates CapFence performance across the five target benchmark environments, illustrating how gate utilization and task suc-cess vary with task type and action complexity.

6.6 Interpretation

Why does latency decrease? The compact gate (Equation 1) is designed to run in under 10 ms on CPU-only hardware. By handling the majority of actions ($1 - \rho \approx 87\%$) with-out verifier consultation, CapFence avoids the dominant latency component of always-on mon-itoring. The marginal increase over static ACL latency is attributable to the feature extraction step in $\phi(x_t)$.

Why does the false-positive rate de-crease? Static ACL rules are context-insensitive: they block or allow action classes uniformly regardless of the current task scope, accumulation of prior actions, or remaining re-source budget. CapFence's capability token encodes all three, allowing the gate to approve actions that a coarse ACL would incorrectly block. The resulting FPR (4.3%) is substan-tially lower than the static ACL (13.8%) and also lower than the LLM monitor (7.9%), be-cause the token-scope signal eliminates a cat-egory of false alarms that are expensive to re-solve semantically.

Why does cost improve? The reduction in token overhead (from 48 to 11 per 100 actions) follows directly from the selective escalation mechanism. Token cost is incurred only when the verifier is invoked ($\rho \approx 13\%$ of actions). The gate itself generates no LLM tokens in the majority of cases.

Table 3: Projected CapFence system performance across benchmark environments. ρ = escalation rate; FPR = false positive rate; TSR = task success rate.

Benchmark	Mean Latency (ms)	ρ (%)	FPR (%)	TSR (%)
GAIA	13.1	9.4	3.8	98.2
WebArena	17.3	15.6	5.1	96.9
Mind2Web	16.8	14.2	4.7	97.4
VisualWebArena	18.6	17.1	5.5	96.3
AgentBench	14.4	10.8	3.9	98.0
Average	**16.0**	**13.4**	**4.6**	**97.4**

Task success is preserved. CapFence's task success rate (97.6%) is not only compa-rable to the always-on monitor (96.1%) but marginally higher, because the lower false-positive rate means fewer legitimate actions are blocked, allowing the agent to complete more tasks without interruption.

7. Discussion

The simulation results in Section 6 raise several questions about CapFence's design rationale, boundary conditions, and practical deployment implications, which we address in turn.

7.1 Strengths and Design Rationale

CapFence's primary strength is that it decou-ples the *cost* of authorization from its *quality*. Static rules are cheap because they are blind; large monitors are accurate because they are expensive. CapFence achieves accuracy close to a full monitor by spending monitor-level compute only on the fraction of actions that genuinely require it. This asymmetry is struc-turally aligned with realistic agentic workloads, in which the vast majority of actions are rou-tine read-only operations, low-stakes retrievals, or clearly scoped tool calls that a compact gate can handle correctly at low cost.

A second strength is the tight coupling be-tween authorization and auditability. As ar-gued by South et al. [1] and Chan et al. [2], these two concerns are not independent: the ability to verify after the fact that an action was authorized requires that a binding record of the authorization token exist. CapFence's hash-chained audit (Equation 4) provides this coupling without requiring verbose trace reten-tion.

The selective escalation mechanism also has a natural scalability property. As deployers tighten the uncertainty band by increasing $\tau_h - \tau_\ell$, more actions are escalated and the sys-tem approaches the security level of always-on monitoring. As they widen the band, the sys-tem approaches the cost profile of static rules. This single dial allows deployers to calibrate the cost-safety tradeoff continuously for their specific environment.

7.2 Limitations and Failure Modes

Prompt injection and adversarial con-text manipulation. A sophisticated adver-sary can craft inputs that cause the agent plan-ner to propose a malicious action disguised as a benign one, or that manipulate the risk features $\phi(x_t)$ to reduce the gate score below τ_ℓ. CapFence does not provide formal guar-antees against adversarial inputs and should be complemented by input sanitization and model-level robustness measures.

Natural-language permission ambiguity. Capability compilation relies on interpreting natural-language intent, which is inherently ambiguous. A permission statement such as

"manage team communications" could justify email reads, email sends, calendar modifica-tions, or Slack posts depending on interpre-tation. An over-permissive compiler will pro-duce tokens with excessive scope; an under-permissive compiler will produce excessive false positives. Structured permission languages or formal policy grammars may partially address this limitation but introduce their own specifi-cation burden.

Rare but severe failure modes. A com-pact gate trained on aggregate workload data may systematically underfit rare action types that carry high consequence. If the training distribution contains few examples of, say, irre-versible data deletion or large financial trans-fers, the gate may assign these actions low risk scores. Targeted data augmentation and class-weighted loss terms (within L_{risk}) can partially mitigate this, but cannot eliminate it entirely.

7.3 Practical Implications

CapFence is best understood not as a complete safety system but as a *governance primitive*: a lightweight enforcement layer that other safety components can build upon. In production deployment, it should be layered with input validation, output filtering, periodic recalibra-tion of thresholds τ_ℓ and τ_h, and human review protocols for critical domains such as financial services and healthcare. The compressed au-dit chain is particularly valuable in regulated industries, where demonstrating per-action au-thorization is a compliance requirement.

The design also has implications for multi-agent systems. When agents delegate sub-tasks to other agents, each delegation hop generates a new capability token derived from the parent token's scope. CapFence's graph-structured ca-

pability model $C = (V, E)$ naturally represents such delegation chains, making it extensible to hierarchical multi-agent architectures.

8. Conclusion and Future Work

This paper has proposed **CapFence**, a lightweight authenticated-delegation layer for agentic AI systems operating under resource constraints. The central contribution is a four-module architecture—capability compilation, compact risk gating, selective escalation, and compressed auditing—that addresses the gap identified by South et al. [1] and Chan et al. [2]: the absence of a runtime mechanism that en-forces least-privilege delegation with low la-tency, low memory use, and a low false-positive rate while preserving task success on realistic long-horizon benchmarks.

The key design insight—that the overwhelm-ing majority of agent actions are not gen-uinely ambiguous and can be adjudicated by a compact gate without invoking an expensive verifier—yields the projected efficiency gains summarized in Tables 1–3: approximately 75% reduction in decision latency, 77% reduction in token overhead, and 46% reduction in false-positive rate relative to always-on LLM moni-toring, achieved with a peak memory footprint compatible with CPU-only edge hardware.

Three directions for future work are partic-ularly important. First, *formal guarantees* for capability scoping and gate stability under dis-tribution shift are needed to move from pro-jected to certified safety properties. Conformal prediction and verified neural networks offer promising starting points. Second, the system should be evaluated on *multimodal and asyn-chronous benchmarks* where tool use is richer and failure modes are more diverse; Visual-WebArena's visual grounding requirements are an immediate target. Third, the community would benefit from a *dedicated delegation-safety benchmark* that measures both action correct-ness and authority correctness simultaneously, providing a standard evaluation substrate for future work in this area.

Agentic AI will not achieve practical deploy-

ment at scale unless it is not only capable but also explainably authorized, cheaply monitored, and operationally trustworthy. CapFence is a step toward that goal.

Acknowledgements

The author gratefully acknowledges the guid-ance and support of **Dr. Payal Gulati** (Men-tor) and **Mr. Piyush Gupta** (Co-Mentor) throughout the development of this work. Their insights and encouragement were invalu-able in shaping the ideas presented here.

References

[1] T. South, S. Marro, T. Hardjono, R. Ma-hari, C. D. Whitney, A. Chan, and A. Pent-land, "Position: AI Agents Need Authen-ticated Delegation," in *Proc. 42nd Int. Conf. Machine Learning (ICML)*, PMLR, vol. 267, pp. 82211–82231, 2025.

[2] A. Chan, C. Ezell, M. Kaufmann, K. Wei, L. Hammond, H. Bradley, E. Bluemke, N. Rajkumar, D. Krueger, N. Kolt, L. Heim, and M. Anderljung, "Visibil-ity into AI Agents," in *Proc. ACM Conf. Fairness, Accountability, and Trans-parency (FAccT)*, pp. 958–973, 2024. doi:10.1145/3630106.3658948

[3] C. C. Phiri, "Creating Characteristically Auditable Agentic AI Systems," in *Proc. Intelligent Robotics FAIR 2025 (IntRob '25)*, 2025. doi:10.1145/3759355.3759356

[4] S. Yao, J. Zhao, D. Yu, I. Shafran, K. R. Narasimhan, and Y. Cao, "Re-Act: Synergizing Reasoning and Acting in Language Models," in *FMDM@NeurIPS*, 2022.

[5] T. Schick *et al.*, "Toolformer: Language Models Can Teach Themselves to Use Tools," 2023.

[6] A. Madaan *et al.*, "Self-Refine: Iterative Refinement with Self-Feedback," 2023.

[7] X. Liu *et al.*, "AgentBench: Evaluating LLMs as Agents," in *ICLR 2024*.

[8] G. Mialon *et al.*, "GAIA: A Benchmark for General AI Assistants," in *ICLR 2024*.

[9] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Ou, Y. Bisk, D. Fried, U. Alon, and G. Neubig, "We-bArena: A Realistic Web Environment for Building Autonomous Agents," in *ICLR 2024*.

[10] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang, H. Sun, and Y. Su, "Mind2Web: Towards a Generalist Agent for the Web," in *Proc. NeurIPS 2023 Datasets and Benchmarks*.

[11] J. Y. Koh, R. Lo, L. Jang, V. Duvvur, M. Lim, P.-Y. Huang, G. Neubig, S. Zhou, R. Salakhutdinov, and D. Fried, "Vi-sualWebArena: Evaluating Multimodal Agents on Realistic Visual Web Tasks," in *Proc. ACL 2024*, pp. 881–905, 2024. doi:10.18653/v1/2024.acl-long.50

[12] S. Yao, H. Chen, J. Yang, and K. R. Narasimhan, "WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents," in *Proc. NeurIPS 2022*.

[13] E. Z. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang, "Reinforcement Learning on Web Interfaces Using Workflow-Guided Exploration," in *ICLR 2018*.

[14] Z. Wei, W. Yao, Y. Liu, W. Zhang, Q. Lu, L. Qiu, C. Yu, P. Xu, C. Zhang, B. Yin, H. Yun, and L. Li, "WebAgent-R1: Training Web Agents via End-to-End Multi-Turn Reinforcement Learning," in *Proc. EMNLP 2025*, pp. 7909–7928. doi:10.18653/v1/2025.emnlp-main.401