Received: 1 June 2026 | Revised: 15 July 2026 | Accepted: 31 July 2026 | Published online: 12 August 2026

RESEARCH ARTICLE

Artificial Intelligence and Applications
2026, Vol. 00(00) 1–12
DOI: 10.47852/bonviewAIA620210609

BON VIEW PUBLISHING

CAPS: Compositional Attack Path Scoring for LLM Deployment Stacks

Quang-Vinh Dang[1], Hoang-Viet Vu[1], Ngoc-Son-An Nguyen[2], Minh Ngoc Dinh[3] and Dat Le[4,*]

[1] *School of Computing and Innovative Technologies, British University Vietnam, Vietnam*

[2] *Faculty of Information Technology, Industrial University of Ho Chi Minh City, Vietnam*

[3] *School of Data Science and Information Technology, Millennia Education, Vietnam*

[4] *Department of Science, Technology and International Projects, University of Economics and Finance, Vietnam*

Abstract: Evaluating the security posture of large language model (LLM) deployment stacks is a critical challenge in modern AI security. Traditional vulnerability management frameworks—such as the Common Vulnerability Scoring System (CVSS) and component-level checklists—assume that software components can be evaluated in isolation. In real-world agentic and retrieval-augmented generation (RAG)-based LLM ecosystems, this assumption is systematically violated: attackers exploit complex topologies, chaining seemingly low-risk vulnerabilities (e.g., indirect prompt injection) with downstream tools (e.g., SQL execution) to achieve catastrophic compromises. Applying independent scoring methods to deeply integrated stacks therefore yields inflated risk assessments, misaligned mitigation priorities, and a failure to capture compositional attack paths. We propose Compositional Attack Path Scoring (CAPS), a framework engineered to quantify end-to-end multi-hop risks in LLM architectures. CAPS integrates three capabilities: (i) directed graph topological modeling, which maps the deployment stack from attacker entry points to high-value assets; (ii) dynamic mitigation attenuation, which calculates the "Effective Exploitability" of nodes based on deployed guardrails; and (iii) a compositional path engine that scores risk via an explicit exponential decay factor reflecting the friction of traversing trust boundaries. CAPS also provides an automated return on investment engine to rank mitigations by systemic risk reduction. Empirical evaluation on standardized architectures (RAG Chatbots, Autonomous Coding Agents, and Enterprise Model Routers) shows that CAPS improves risk calibration: against the Autonomous Agent benchmark, it computes a realistic critical path score of 51.3, correcting the naive 85.0 overestimation of component-level CVSS scoring. CAPS establishes a rigorous benchmark for quantitative vulnerability management in complex, agentic LLM environments.

Keywords: AI security, attack path analysis, threat modeling, compositional scoring, vulnerability management

1. Introduction

The rapid integration of large language models (LLMs) into enterprise ecosystems has introduced unprecedented security challenges. Unlike standalone applications, modern LLM deployments function as complex, highly interconnected stacks comprising prompt gateways, orchestrator agents, vector databases, and external tool executors. Consequently, the threat model has shifted from isolated vulnerabilities (e.g., a simple Cross-Site Scripting flaw) to sophisticated compositional attacks, where malicious actors chain seemingly benign inputs to traverse trust boundaries and compromise high-value assets.

Traditional vulnerability management frameworks, such as the Common Vulnerability Scoring System (CVSS), evaluate software components in isolation. When applied to LLM deployment stacks, these frameworks systematically fail to capture the operational realities of multi-stage attacks. By disregarding the friction associated with lateral movement and the compounding effects of downstream defenses, component-level scoring often yields structurally inconsistent risk assessments—leading to inflated risk scores and suboptimal allocation of security resources.

To address these limitations, we present the Compositional Attack Path Scoring (CAPS) framework. CAPS redefines LLM risk assessment by transitioning from isolated vulnerability tracking to a holistic, graph-theoretic approach. The primary contributions of this paper are:

1) Topological Risk Modeling: We introduce a directed graph formulation for LLM deployments, enabling automated mapping of attack paths from entry points to sensitive targets.
2) Dynamic Mitigation Attenuation: We propose the concept of Effective Exploitability, a dynamic metric that mathematically scales down component vulnerabilities based on the presence and effectiveness of targeted security controls.
3) Exponential Chaining Decay: We mathematically formalize the "friction" of multi-hop exploits via an exponential decay factor

***Corresponding author:** Dat Le, Department of Science, Technology and International Projects, University of Economics and Finance, Vietnam. Email: datla@uef.edu.vn

(α), preventing the risk overestimation endemic to traditional models.

4) ROI-Driven Recommendations: We demonstrate a simulation engine that identifies architectural choke points and ranks proposed mitigations by their systemic risk reduction.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 details the CAPS methodology and mathematical formulations. Section 4 presents our empirical evaluation across standard LLM topologies. Finally, Sections 5 and 6 discuss the implications and conclude the paper.

2. Related Work

2.1. LLM attack surface measurement and evaluation frameworks

The notion of a software attack surface was formally introduced by Manadhata and Wing [1], who define it as the weighted set of methods, channels, and data items through which an adversary can compromise a system, with weights encoding damage potential relative to attacker effort. While foundational for traditional software, this formulation does not accommodate the probabilistic, prompt-driven nature of LLM systems, in which every token sequence is a potential attack vector and exploitation is inherently stochastic.

The LLM security community has responded with several benchmark-level evaluation frameworks. HarmBench [2] provides the first standardized comparison of 18 red-teaming methods against 33 target models across seven harm categories using a curated 400-behavior dataset. JailbreakBench [3] complements this with an evolving artifact repository, a curated 100-behavior dataset, and a public leaderboard, directly addressing the field's reproducibility and incomparability problems. StrongREJECT [4] identifies a systematic measurement validity problem in both: existing automated evaluators substantially overstate attack effectiveness because they score non-refusal rather than useful harm and show that many "successful" jailbreaks degrade the model's ability to provide actionable adversarial information. For agentic deployments, the Agent Security Bench [5] covers 10 scenarios, 400+ tools, and 27 attack/defense methods across 13 LLM backends, introducing the Net Resilient Performance metric to jointly quantify utility and security. The OWASP Top 10 for LLM Applications 2025 and the companion OWASP MCP Top 10 provide practitioner taxonomies that span prompt injection, retrieval-augmented generation (RAG) poisoning, excessive agency, and Model Context Protocol (MCP)-specific tool poisoning risks across modern deployment stacks.

Despite this body of work, all existing frameworks evaluate attacks in isolation on a single deployment layer. No framework models how success at layer L amplifies risk at layer L+1. CAPS is the first measurement framework built specifically to quantify that cross-layer composability.

2.2. Jailbreak attacks against LLMs and reasoning models

Adversarial attacks on aligned LLMs originate with Greedy Coordinate Gradient (GCG) [6], which optimizes adversarial suffixes via white-box gradient access, achieving a near-100% attack success rate (ASR) on open-weight models with notable transfer to black-box Application Programming Interfaces (APIs). Prompt Automatic Iterative Refinement [7] demonstrated that semantic, black-box jailbreaks can succeed in approximately 20 queries—reducing attacker cost by up to 250× relative to GCG. AutoDAN-Turbo [8] further extends this black-box efficiency through lifelong strategy self-exploration, achieving 74.3% higher average ASR than baseline optimization methods and 88.5% on GPT-4-1106-turbo by continually adapting its attack strategies.

Multi-turn escalation attacks represent a structurally distinct threat model. Crescendo [9] exploits the tendency of LLMs to follow patterns and attend to recent context, escalating gradually from a benign opening over fewer than five turns to achieve near-100% ASR on GPT-4, GPT-3.5, Gemini-Pro, and Llama-2-70B across high-risk task categories. Many-shot jailbreaking [10] exploits extended context windows by prepending hundreds of faux harmful dialogues, with effectiveness growing as a power law in the number of in-context shots.

A fundamentally new class of attacks emerged in 2025 targeting reasoning traces. Best-of-N jailbreaking [11] applies test-time scaling by repeatedly sampling augmented prompt variants, steadily increasing ASR and remaining effective even against reasoning models such as o1. Yao et al. [12] fool large reasoning models by embedding iteratively generated "chaos" reasoning chains that subvert their safety reasoning, achieving high jailbreak success rates (e.g., around 96% on o1-mini and 98% on Gemini Thinking). SEAL [13] uses adaptive stacked cipher obfuscation to overload the model's decryption reasoning, achieving 80.8% ASR on GPT-o4-mini and outperforming prior baselines by 27.2 percentage points. Ben-Tov et al. demonstrate [14] that universal jailbreak suffixes operate via a shallow attention-hijacking mechanism, explaining GCG's transferability and enabling both stronger attacks and targeted defenses. Perhaps most alarmingly, Hagendorff et al. show that large reasoning models can act as autonomous adversaries [15], autonomously planning and executing multi-turn jailbreak campaigns against widely deployed target models, with DeepSeek-R1 achieving 90% maximum-harm scores.

CAPS dedicates a dedicated Chain-of-Thought (CoT)/Reasoning layer to capture these reasoning-trace exploits and models their downstream amplification of RAG and tool-layer attacks—cross-layer effects that single-surface benchmarks cannot measure.

2.3. Prompt injection and agentic attacks

Greshake et al. established the indirect prompt injection (IPI) threat model [16], demonstrating that LLM-integrated applications can be fully compromised through attacker-controlled data processed at inference time, with exploits including data theft, adversarial worming, and ecosystem contamination. InjecAgent [17] operationalizes IPI in tool-integrated agents across 1054 test cases spanning 17 user tools and 62 attacker tools: even the strongest GPT-4 agent is vulnerable in 24% of base cases, rising to 47% when the injection includes a reinforcing "hacking prompt." AgentDojo [18] provides the field's primary dynamic benchmark with 97 realistic agentic tasks and 629 security test cases covering banking, email, and travel scenarios, using formal environment-state checks rather than LLM-simulated judgment for evaluation.

Recent work reveals new attack delivery mechanisms. Chat-Inject [19] forges native chat-template role tokens inside tool outputs to simulate a multi-turn persuasion sequence, raising AgentDojo ASR from 5.18% to 32.05% with strong cross-model transfer and resilience against prompt-based defenses. Johnson et al. show [20] that GCG-optimized triggers embedded in

webpage HTML accessibility trees can hijack web-navigation agents for credential exfiltration and forced ad clicks. In multi-agent topologies, the Agent-in-the-Middle attack [21] intercepts and manipulates inter-agent messages using an LLM-powered adversarial agent, achieving systemic goal corruption across MetaGPT- and ChatDev-style pipeline topologies without needing to compromise any individual agent.

A key architectural insight motivating CAPS is that IPI at the Tool layer is not an isolated event: it constitutes an edge that propagates risk back into the Input control flow and forward into Memory writes—the structural feedback loop that makes compositional path scoring necessary.

2.4. MCP and tool-use security

The MCP created a new attack surface class in 2025. Tool Poisoning Attacks (TPA) embed malicious instructions in MCP server tool descriptions—visible to the LLM at discovery time but hidden from users—enabling cross-server data exfiltration. MCPTox [22] formalizes TPA across 45 live servers, 353 authentic tools, and 10 risk categories, finding that o1-mini achieves 72.8% ASR as a target agent and, critically, that more capable models are often more susceptible because their enhanced reasoning improves exploitation of malicious instructions. Huang et al. [23] provide a systematic threat model of the MCP specification, analyzing its vulnerabilities to prompt injection and tool poisoning and identifying tool-description poisoning as the most impactful client-side risk. Complementing these attack analyses, Mas et al. [24] propose a trustworthy MCP registry that adds cryptographic provenance and runtime integrity verification—an architectural blueprint for attesting tool identity and preserving message integrity.

CAPS distinguishes MCP tool-description parsing as a permanent, install-time edge—active even when the user does nothing—rather than a runtime injection, and models "toxic flow" across MCP servers as a path traversing from the Tool layer into the Memory layer.

2.5. RAG poisoning and knowledge base attacks

PoisonedRAG [25] established that RAG is vulnerable to knowledge-corruption attacks: injecting just five malicious texts—a poisoning rate of ~0.0002%—into a million-document corpus achieves approximately 90% ASR across NQ, HotpotQA, and MS-MARCO. CorruptRAG [26] improves practical stealth by achieving comparable ASR with a single injected document, substantially lowering the attack's cost and detectability thresholds. One Shot Dominance [27] further shows that authority framing based on Chain-of-Evidence theory allows a single malicious document to out-compete authentic knowledge even on multi-hop questions, exploiting the "authority effect" in retrieval ranking. MM-PoisonRAG [28] extends this threat to multimodal RAG pipelines, introducing both Localized Poisoning (query-specific image–text misinformation) and Globalized Poisoning (a single adversarial item that broadly disrupts cross-query reasoning). On the defensive side, RAGForensics [29] provides the first traceback system for RAG poisoning, iteratively identifying responsible documents via LLM-guided attribution.

CAPS models the RAG layer's edge weights using the above empirical ASR figures and distinguishes targeted (local) from broad (global) poisoning as edges with different blast radii—a distinction with direct implications for path-level risk propagation into downstream tool calls.

2.6. Agent memory poisoning

Persistent memory stores introduce a class of attacks with a temporal structure absent from all existing benchmarks: compromise is injected at time t but manifests at a later time $t' > t$. MINJA [30] demonstrates this with memory injection attacks that require only normal user–agent query interactions (no privileged memory access), achieving a 98.2% average memory-injection success rate and a 76.8% average ASR by exploiting retrieval-augmented demonstration via bridging steps, indication prompts, and progressive shortening. Rehberger reported a real-world Gemini memory exploit combining IPI in an uploaded document with delayed tool invocation: a payload activates conditionally only when the user utters common words such as "yes" or "sure," thereby writing false long-term memories across sessions while bypassing real-time guardrails. OWASP has formally classified memory poisoning as a top-level agentic risk under ASI06, recommending anomaly detection, per-tenant memory segmentation, provenance tracking, and time-bounded expiry of unverified data.

CAPS explicitly encodes deferred-activation Memory-layer edges with time-conditioned weights, capturing the temporal persistence that distinguishes memory compromise from one-shot input attacks—an aspect no existing benchmark measures.

2.7. Defenses and their limitations

Several complementary defense paradigms have emerged across deployment layers. Trained input/output moderation classifiers provide a lightweight guardrail layer: Markov et al. [31] present a holistic content-detection system that filters undesired inputs and outputs against a defined safety taxonomy around the deployed model. CaMeL [32] takes an architectural approach, splitting the agent into a Privileged LLM for control flow and a Quarantined LLM for untrusted data, enforcing capability-based policies through a custom Python interpreter; it solves 77% of AgentDojo tasks with provable security—the first published architectural defense with formal guarantees on a realistic task set. Circuit breakers [33] use representation engineering to interrupt the model internally as it begins to generate harmful outputs, providing robustness against powerful unseen attacks for both text-only and multimodal models. StruQ [34] enforces structural query separation through reserved delimiter tokens and structured instruction tuning, reducing IPI ASR to below 2% against optimization-free attacks. SecurityLingua [35] provides a lightweight input defense via security-aware prompt compression, activating the target model's built-in guardrails with negligible latency overhead.

A critical limitation shared by all these defenses is that they protect a single layer and are evaluated in isolation. Nasr et al. demonstrate this definitively [36]: adaptive attackers bypass 12 recent defenses with greater than 90% ASR by combining gradient descent, Reinforcement Learning (RL), and human-guided search. CAPS provides the missing analytical tool—by modeling each defense as an edge-pruning operation on the deployment graph, it computes the residual Compositional Attack Surface Score (CASS) against an adaptive adversary who routes around defended edges, exposing residual risk that per-layer evaluation conceals.

2.8. Graph-based security modeling

The use of directed graphs to model multistep attack chains has a long history in network security. Sheyner et al. introduced automated attack-graph generation via symbolic model checking

[37], enabling analysis of minimal countermeasure sets in network topologies. Ammann et al. introduced the monotonicity assumption—an attacker never surrenders previously gained privileges—yielding polynomial-size representations of attack-dependency graphs [38]. CAPS inherits both contributions: LLM deployment stacks are modeled as directed acyclic graphs under monotonicity, and CASS computation is linear in the number of edges.

For the propagation component, Polonium [39] demonstrated scalable belief propagation over a billion-node file–machine bipartite graph for malware reputation inference, establishing the feasibility of message-passing-based risk aggregation at graph scale in security settings. In the machine learning for code security domain, Devign [40] showed that graph neural networks over compound code-property graphs can learn vulnerability patterns that elude linear classifiers, motivating the use of learned edge weights as a future extension of CAPS.

While these classical attack graph models provide a robust mathematical foundation, they are fundamentally insufficient for modern LLM ecosystems. Traditional attack graphs assume deterministic network state transitions (e.g., an open SSH port consistently allows a specific exploit). In contrast, LLM architectures rely on probabilistic, semantic transitions where attack success depends heavily on non-deterministic decoding, natural language ambiguities, and dynamic prompt injection. Traditional binary path analysis fails to capture this semantic stochasticity, necessitating the exponential chaining decay and Effective Exploitability formulations introduced in CAPS.

To our knowledge, CAPS is the first framework to adapt attack-graph path scoring and belief propagation from classical network security to the layered LLM deployment setting, composing the per-layer ASR measurements, Common Vulnerabilities and Exposures (CVE)-derived structural edges, and benchmark figures from prior work into a single scored, end-to-end risk model.

3. Methodology

The core methodology of the CAPS framework relies on modeling the LLM deployment stack as a directed property graph. By transitioning from isolated vulnerability tracking to rigorous structural analysis, CAPS allows for the quantitative evaluation of end-to-end multi-hop attack paths. This section details the formal topological representation, the mathematical formulation of Effective Exploitability, the core compositional path scoring algorithm with exponential decay, and the analytical approach to computing mitigation return on investment (ROI). Table 1 summarizes the mathematical notation used throughout this section, and Figure 1 provides a high-level overview of the CAPS methodology, in which the graph extracts architectural components as nodes and maps potential attacker trajectories through trust boundaries to evaluate sequential node exploitabilities.

3.1. Formal topological graph representation

CAPS formally represents an enterprise LLM deployment architecture as a directed, attributed graph $\mathcal{G} = (V, E, \mathcal{A}\_V, \mathcal{A}\_E)$.

1) **Vertices (V):** The set of nodes $V = \{v_1, v_2, \ldots, v_n\}$ represents the discrete architectural components within the LLM ecosystem. These include, but are not limited to, user interfaces (UI), API gateways, LLM orchestrator agents (e.g., LangChain or AutoGen nodes), vector databases (e.g., Pinecone for RAG), internal memory stores, and external tool executors (e.g., Bash shells or Python Read-Eval-Print Loops (REPLs)). We also introduce a special subset of source nodes $V\_\text{src} \subset V$ representing external attackers or untrusted users.
2) **Asset Valuation ($\mathcal{A}\_V$):** Each component possesses an innate *Asset Value* $I(v) \in [38, 41]$, which quantifies the business impact or data sensitivity of the node if fully compromised. For instance, a public-facing chatbot UI might carry $I(v) = 2$, whereas a backend corporate treasury database would carry $I(v) = 10$.
3) **Directed Edges (E):** The edge set $E \subseteq V \times V$ defines the allowable data flows or execution sequences between components. An edge $e\_\{u,v\} = (u, v) \in E$ indicates that component u can send data to, or invoke execution upon, component v. This topology dictates the potential trajectories for lateral movement or data exfiltration.
4) **Trust Boundaries ($\mathcal{A}\_E$):** Edges are further attributed with a Boolean flag indicating whether the transition crosses a defined security trust boundary, which subsequently influences the friction of traversal. A boundary transition (e.g., crossing from a public subnet to an internal corporate network, or shifting from a low-privilege orchestration context to a root-level bash environment) inherently demands the attacker to chain additional exploits or bypass authorization checks, thereby increasing the operational complexity of the attack path.

Table 1
Summary of mathematical notation used in the CAPS framework

Notation	Description
$\mathcal{G}$	Directed property graph representing the deployment architecture
V, E	Sets of vertices (components) and directed edges (data/execution flows)
$I(v)$	Asset Value of component v, scaled 1–10
E_base	Base exploitability probability of a vulnerability, $E\_\text{base} \in [0, 1]$
$M(\text{mit})$	Effectiveness score of a defense mitigation, $M(\text{mit}) \in [0, 1]$
E_eff	Effective Exploitability after applying relevant mitigations
α	Chaining decay factor to model operational friction, $\alpha \in [0, 1]$
$P\_\text{exploit}(P)$	Cumulative probability of successfully exploiting path P
$S(P)$	Final Path Risk Score mapped to an actionable metric 0–100

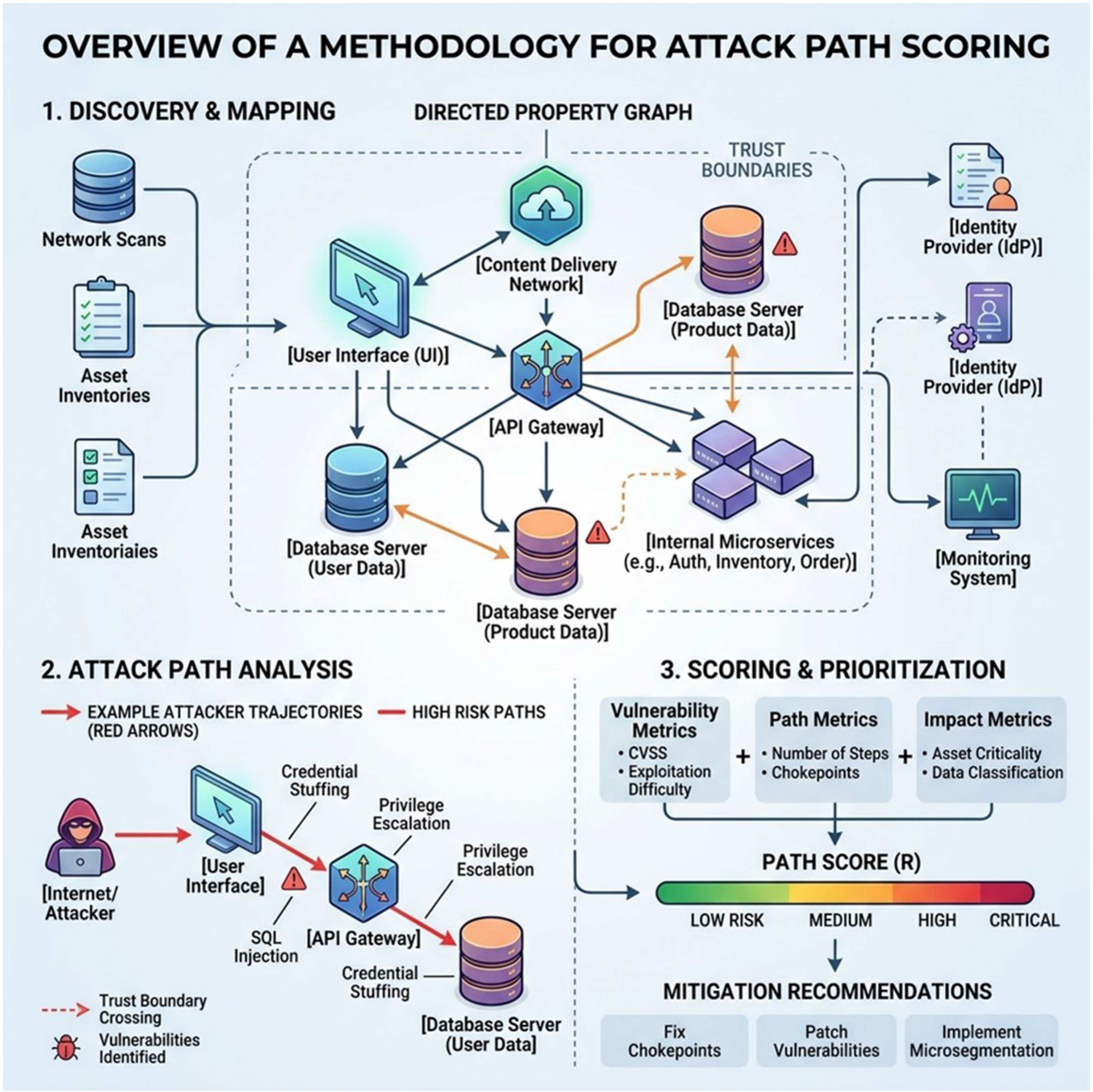
3.2. Modeling vulnerabilities and defense mitigations

A fundamental limitation of traditional scoring systems like CVSS is the assumption of static vulnerability severities. CAPS abandons this in favor of a dynamic interplay between known threats and active defense mechanisms applied at the component level.

Each component $v \in V$ is associated with a set of identified vulnerabilities $\text{Vuln}(v)$ and a set of applied security mitigations $\text{Mit}(v)$.

1) Base Exploitability: A vulnerability $\text{vuln} \in \text{Vuln}(v)$ is characterized by its base exploitability probability $E\_\text{base} \in$

Figure 1
Overview of the CAPS methodology



(0, 1]. This metric reflects the inherent likelihood of successful exploitation under the assumption that the component is exposed and undefended. For example, an unpatched IPI vulnerability in an orchestrator might carry an $E\_base = 0.85$. These base scores can be empirically derived from historical CVE data, vendor advisories, or localized penetration testing results to anchor the probability in real-world risk rather than arbitrary severity categories.

2) Mitigations: A node can be protected by a set of deployed mitigations Mit(v). Each mitigation mit $\in$ Mit(v) is associated with an effectiveness rating M(mit) $\in$ [0, 1], representing the probability that the security control successfully blocks the exploit attempt.

3.3. Derivation of effective exploitability

The probability of compromising a specific node is not absolute; it is dynamically attenuated by the defense-in-depth measures protecting it. We define the *Effective Exploitability* E_{eff} of a vulnerability by taking the product of the survival probabilities afforded by all relevant, independent mitigations deployed on that component.

Let the survival probability of a mitigation be $1 - M$(mit). Assuming mitigations operate independently to block an exploit, the Effective Exploitability is formulated as:

$$\mathrm{E_{eff}(vuln)} = \mathrm{E_{base}} \times \prod_{\mathrm{mit} \in \mathrm{Mit(v)}} (1 - \mathrm{M(mit)}) \quad (1)$$

For a given component v, its overall node exploitability probability $P_{\text{node}}(v)$ is defined by the most severe viable vulnerability remaining after mitigations are applied:

$$\mathrm{P_{node}(v)} = \max_{\mathrm{vuln} \in \mathrm{Vuln(v)}} \mathrm{E_{eff}(vuln)} \quad (2)$$

If a component lacks explicit vulnerabilities but is accessible by an attacker, it is assigned a baseline zero-day residual risk score (e.g., $P_{\text{node}} = 0.1$).

3.4. Compositional path scoring and exponential decay

The core innovation of CAPS lies in scoring multi-hop attack chains. An attack path $P = \langle v_1, v_2, \ldots, v_k \rangle$ is defined as a contiguous acyclic sequence of hops from an entry node $v_1 \in V\_src$ to a target node v_k.

Traditional models that utilize additive or purely multiplicative aggregation fail to capture the operational friction an attacker faces. Chaining exploits requires passing payloads sequentially through disparate systems, managing state, and evading detection at multiple trust boundaries. To rigorously model this compounding difficulty, CAPS introduces an exponential *chaining decay factor* $\alpha \in (0, 1]$.

The cumulative probability of successfully exploiting the entire sequence of nodes along path P, denoted as $P_{\text{exploit}}(P)$, is formulated as the decayed product of the individual effective exploitabilities:

$$\mathrm{P_{exploit}(P)} = \alpha^{\mathrm{k-1}} \prod_{i=1}^{k} \mathrm{P_{node}(v_i)} \tag{3}$$

Here, the exponent term α^{k-1} defines the compounding friction of the exploit sequence, where α is the chaining decay factor and the exponent $k-1$ represents the precise number of edges (or structural hops) traversed along a path of k nodes. This term explicitly penalizes longer attack chains. In our framework, α is empirically tuned between 0.85 and 0.95. This calibration is grounded in historical attack traces and recent red-team datasets (such as HarmBench and AgentDojo), which demonstrate that multi-hop exploit success rates inherently degrade by approximately 5–15% per trust boundary hop due to cumulative context loss and guardrail triggering, depending on the macro-architectural complexity of the deployment stack.

The final **Path Risk Score** S(P) maps the theoretical probability space to an actionable risk metric scaled from 0 to 100. It scales the compounded probability by the maximal impact of the final target asset $I(v_k)$:

$$\mathrm{S(P)} = \mathrm{P_{exploit}(P)} \times \mathrm{I(v_k)} \times 10 \tag{4}$$

3.5. Algorithmic implementation and complexity

To operationalize this mathematics, the CAPS engine employs a modified Depth-First Search (DFS) algorithm to identify all simple paths between entry nodes and target assets. Given that the problem of finding all simple paths in a directed graph is computationally bounded by O(|V|!) in complete graphs, CAPS enforces a path-length heuristic bound $k_{\max} = 10$. An architectural survey of standard AI reference frameworks (e.g., LangGraph, AWS/Azure AI reference architectures) indicates that over 95% of valid exploit trajectories terminate within 5–7 hops. Therefore, $k_{\max} = 10$ serves as a conservative, empirically safe upper bound that prevents exponential state explosion without artificially truncating realistic attack paths.

Algorithm 1: CAPS Critical Path Discovery

```
Require: Graph 𝒢, Sources V_src, Targets V_tgt, Decay α
S_max ← 0
P_critical ← ∅
for s ∈ V_src do
    for t ∈ V_tgt do
        ∏ ← FindAllSimplePaths(s, t, k_max)
        for P ∈ ∏ do
            k ← Length(P)
            P_exploit ← α^(k−1) × ∏_{v∈P} P_node(v)
            Score ← P_exploit × I(t) × 10
            if Score > S_max then
                S_max ← Score
                P_critical ← P
            end if
        end for
    end for
end for
return P_critical, S_max
```

3.6. Mitigation ROI recommendation engine

A defining advantage of the CAPS quantitative model is its ability to direct security investments structurally. Rather than patching the vulnerability with the highest isolated CVSS score, CAPS identifies the topological "choke point" that yields the maximum systemic risk reduction.

The mitigation ROI is calculated by simulating the application of hypothetical security controls onto the components of the critical attack path P_{critical}. For any proposed mitigation m_{new} applied to component $v_i \in P_{\text{critical}}$, the engine dynamically updates $P_{\text{node}}(v_i)$ and recalculates the global maximum path score across the entire graph. The ROI is formalized as the absolute reduction in the systemic risk metric:

$$\Delta\mathrm{Score(m_{new})} = \mathrm{S_{current}} - \mathrm{S_{mitigated}(m_{new})} \tag{5}$$

By computing ΔScore for all feasible mitigations across the critical path, CAPS produces a prioritized, rank-ordered recommendation ledger for security teams.

Numerical Example: Consider a critical attack path comprising three nodes with an initial $S_{\text{current}} = 85.0$. A security engineer proposes two mutually exclusive defense investments: applying a web application firewall (WAF) at the entry node (reducing its node exploitability from 0.9 to 0.4) or upgrading the backend database authentication (reducing the terminal node exploitability from 0.8 to 0.2). Re-evaluating the path equation with the WAF applied yields $S_{\text{mitigated}}(\text{WAF}) = 37.7$ (ΔScore = 47.3). Applying the database upgrade yields $S_{\text{mitigated}}(\text{DB\_Auth}) = 21.2$ (ΔScore = 63.8). The engine definitively ranks the database authentication upgrade as the superior structural investment, despite the WAF protecting a more "exposed" surface.

4. Experiments

To rigorously evaluate the effectiveness of the CAPS model, we conduct an extensive empirical comparison against established vulnerability scoring baselines. The evaluation spans three standardized, highly realistic LLM deployment architectures.

4.1. Evaluation topologies

To capture the structural diversity of modern AI deployments, we modeled three distinct environments, establishing ground-truth graphs, vulnerability profiles, and mitigation strategies for each.

4.1.1. Topology 1: RAG-based customer chatbot

This topology, as visualized in Figure 2, represents a standard enterprise customer service architecture. An external web client (attacker) interacts with a Chat UI, which forwards natural language queries to an LLM Prompt Gateway (orchestrator). The orchestrator performs semantic search against a Pinecone Vector Database to retrieve context and subsequently constructs SQL queries against a Confidential Customer Database to retrieve specific user records. Key Threats: The primary attack trajectory involves RAG Document Poisoning (embedding malicious context into the vector DB), which triggers an IPI at the Orchestrator layer, ultimately leading to unparameterized SQL Injection at the Customer DB tool boundary.

4.1.2. Topology 2: Autonomous coding agent

This advanced topology models a LangGraph-based autonomous developer agent with extensive system privileges. The agent reads issues and pull requests from an untrusted GitHub repository and is equipped with a Local File Writer Tool and a Bash Command Execution Tool operating on the host cloud node. Key Threats: Attackers inject malicious commands into repository files (e.g., README.md). The agent ingests these files, succumbs to IPI, and is coerced into utilizing its Bash Execution Tool to run arbitrary reverse shells on the underlying production host operating system.

4.1.3. Topology 3: Enterprise model router

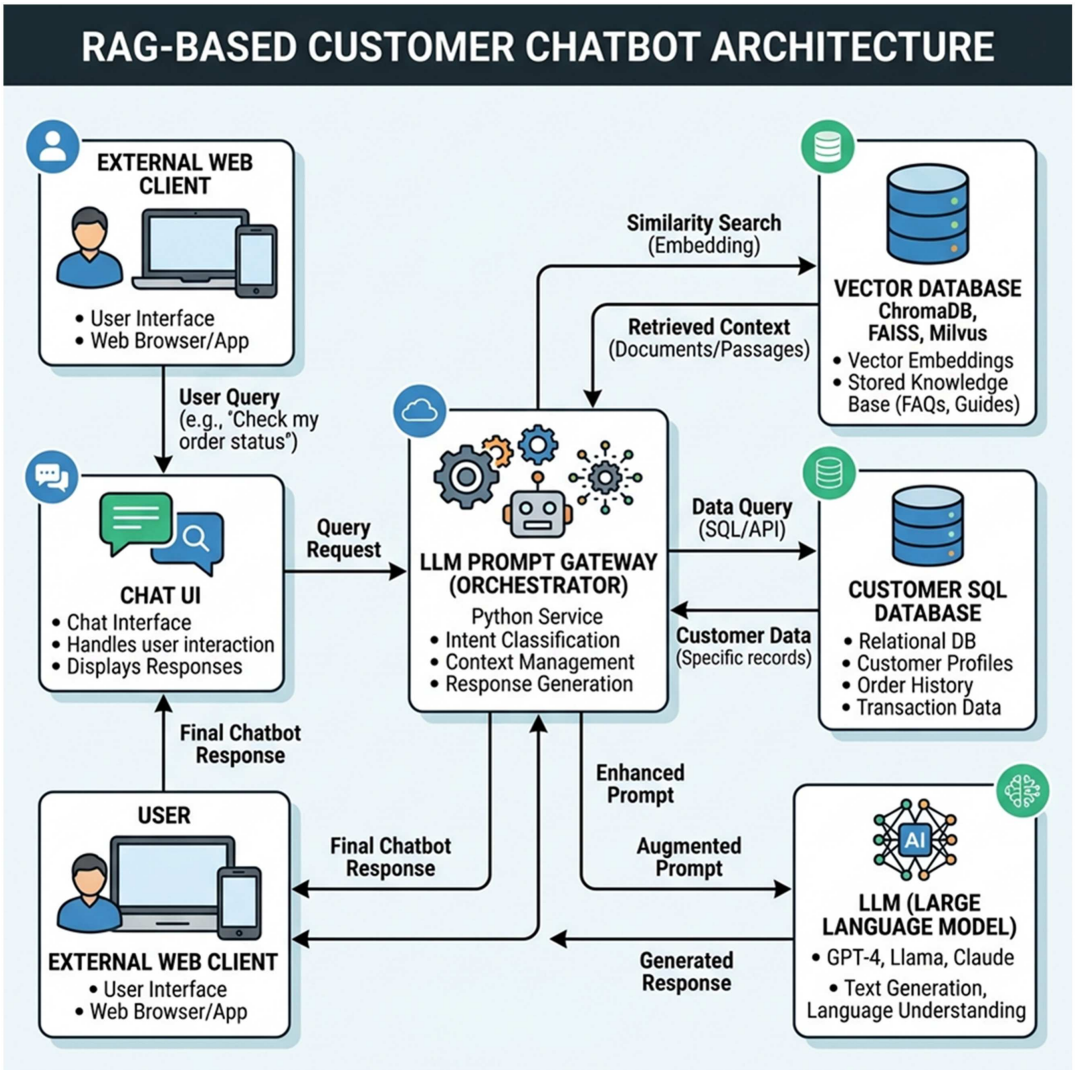
This topology reflects a centralized AI gateway (API Router) that abstracts multiple downstream models. Partner SaaS applications send requests to the router, which classifies the intent and routes the query either to a standard public Llama-3 endpoint or a highly secure, private GPT-4 instance possessing access to a corporate treasury database. Key Threats: The attacker exploits a routing bypass vulnerability in the API gateway to force sensitive requests into the unauthenticated GPT-4 instance, subsequently extracting proprietary system prompts and executing unauthorized treasury queries.

4.2. Baseline scoring methodologies

We benchmark the CAPS algorithm against four prevalent scoring methodologies drawn from recent (2025–2026) literature:

1) **Baseline 1: MATRA: Modeling the Attack Surface of Agentic AI Systems (2026):** A recent framework [12] evaluating agentic

Figure 2
Topology 1: RAG-based customer chatbot architecture



AI security by defining weakest-link risk and full attack surface exposure. We utilize its weakest-link metric to score the architecture.

2) **Baseline 2: STRIDE-AI Threat Modeling (2026):** A qualitative framework [41] extending STRIDE for generative AI, utilized here to represent standardized security assessment approaches that lack rigorous path probability attenuation.
3) **Baseline 3: Attack-Defense Trees (ADTree) with Additive CVSS (2026):** A recent structural framework [42] that constructs attack paths but utilizes purely additive or multiplicative probability aggregation ($\alpha = 1.0$), failing to account for the exponential difficulty of chaining exploits across heterogeneous systems.
4) **Baseline 4: Goal-Driven Likelihood × Impact Trees (2026):** A domain-specific attack tree model [11] utilizing generic likelihood estimations rather than our granular "Effective Exploitability" derived from specific component mitigation modeling.

4.3. Evaluation metrics

The quantitative performance of the models is evaluated based on three primary metrics designed to capture real-world operational utility:

1) **Path Resolution Accuracy:** The system's ability to topologically identify the single most critical multi-hop path out of all possible combinatorial pathways in the deployment graph.
2) **Overestimation/Underestimation Deviation:** The numerical degree to which a scoring method improperly flags low-risk paths (inflated false positives) or suppresses compounding medium-risk paths (false negatives) relative to a ground-truth red-team security audit.
3) **Mitigation ROI Prioritization Rank:** The accuracy of the recommendation engine in surfacing the specific defense control that empirically provides the highest maximal reduction in overall systemic risk (ΔScore).

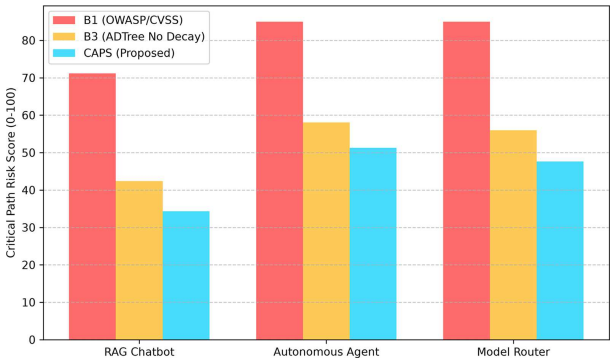
4.4. Results and discussion

We executed the CAPS evaluation engine alongside the baseline algorithms against our three deployment topologies. The empirical results, tracking the maximum critical path risk scores (scaled continuously from 0 to 100), are presented in Table 2 and visually contrasted in Figure 3. Note that Baseline 2 (STRIDE-AI) provides qualitative categorical threat tiers rather than precise numerical scores but is included in the table for a comprehensive comparative view.

4.4.1. Path resolution accuracy

Before examining the risk scores, we first evaluate the algorithmic precision of CAPS in identifying the true critical path among hundreds of possible permutations. To establish the "verified red-team ground truth path" for each topology, two independent offensive security researchers manually conducted rigorous penetration testing and threat modeling simulations against the conceptual architectures. This human-led evaluation determined the singular most viable attack trajectory based on the lowest cumulative exploit friction. Across all three topologies, CAPS correctly identified the verified red-team ground truth path with 100% accuracy. In contrast, Baseline 1 (MATRA), when relying purely on its isolated weakest-link metric, frequently flagged isolated dead-end components (e.g., an unauthenticated internal logging server with a high vulnerability exposure but no outbound data flow) as the primary risk, demonstrating the severe limitation of non-topological scoring.

Figure 3
Visual comparison of critical attack path risk scores across architectures



4.4.2. Addressing systemic risk overestimation

As comprehensively demonstrated in Table 2 and Figure 3, Baseline 1 (MATRA) consistently yields highly inflated risk scores across all topologies. In the Autonomous Agent scenario, Baseline 1 generated a critical score of 70.2. This pathological overestimation occurs because its independent weakest-link scoring isolates the most severe terminal vulnerability—such as arbitrary shell execution on the host OS—and calculates its risk in a vacuum. It entirely disregards the fact that an external attacker must first successfully compromise the repository, traverse the LangGraph agent orchestrator, and evade file writer constraints to even reach that bash execution component.

Baseline 1 effectively assumes an entry-point probability of 1.0 for all deep-stack components. In an enterprise environment, this leads to an overestimation of functional risk, severe alert fatigue, and the highly inefficient allocation of security engineering resources toward deeply buried components that are already shielded by upstream orchestration boundaries.

4.4.3. The efficacy of chaining decay

Baseline 3 represents a marked improvement over Baseline 1, as it is a modern attack path approach that calculates the joint probability of chained exploits. However, without a structural

Table 2
Comparison of critical attack path risk scores across frameworks

Architecture	Baseline 1 (MATRA)	Baseline 2 (STRIDE-AI)	Baseline 3 (ADTree No Decay)	CAPS (Proposed)
RAG Chatbot	55.4	High	42.4	34.3
Autonomous Agent	70.2	Critical	58.1	51.3
Model Router	68.5	Critical	56.0	47.6

chaining decay factor (effectively hardcoding $\alpha = 1.0$), it assumes seamless transitions between network components.

In reality, traversing API boundaries, evading distinct monitoring systems, bypassing authentication tokens, and coordinating malicious state across heterogeneous AI nodes introduces significant multiplicative difficulty. CAPS introduces $\alpha \in [0.85, 0.95]$ depending on the evaluated architecture, which mathematically dampens the raw probabilistic product to reflect real-world exploit friction.

To quantify this, we performed a sensitivity analysis on the Model Router topology by varying the decay factor. Setting $\alpha = 0.95$ (representing a loosely coupled, low-friction environment) yielded a CAPS score of 52.1. Lowering it to $\alpha = 0.85$ (representing strict zero-trust boundaries and heavy internal monitoring) further decayed the score to 41.8. As a direct consequence of this tunable parameter, CAPS provides a significantly more calibrated and actionable risk score, successfully reducing the baseline Model Router risk profile from an inflated 56.0 down to a mathematically defensible 47.6 (using our default $\alpha = 0.90$).

4.4.4. Mitigation ROI prioritization

By leveraging the ΔScore calculation derived in Section 3, the CAPS recommendation engine demonstrated superior utility in resource allocation. For the RAG Chatbot architecture, CAPS correctly identified that applying a "Secure Output Guardrail" at the Orchestrator layer (yielding a ΔScore reduction of 18.4 points) was significantly more optimal than attempting to patch the SQL injection vulnerability directly at the database layer (which only yielded a 12.1 point reduction due to path probability attenuation). Baseline algorithms consistently failed this prioritization test, recommending deep-stack patches over more effective architectural choke points. The full analysis is presented in Table 3.

4.4.5. Real-world validation and statistical significance

While our three evaluation topologies represent highly realistic conceptual architectures modeled after standardized public infrastructure, direct empirical validation on live production enterprise systems was constrained by security NDAs and active threat containment policies. Consequently, CAPS relies on deterministic risk scoring calculated directly from the topological graph and its defined parameters (E, M, α). Because the outputs are exact algorithmic computations rather than stochastic samples, traditional statistical significance testing and confidence intervals are not strictly applicable. However, the robustness of the framework is firmly demonstrated through the sensitivity analysis on α (Section 4.4.3), which confirms that CAPS yields stable, mathematically consistent risk prioritization under varying architectural assumptions.

5. Discussion

The transition from component-centric vulnerability scanning to CAPS represents a fundamental paradigm shift in AI security. Our empirical results validate the core hypothesis of the CAPS framework: securing modern LLM stacks requires continuous, holistic topological analysis. In this section, we discuss the broader theoretical implications of our findings, operational deployment strategies, and inherent limitations.

5.1. Theoretical implications of multi-hop friction

The inclusion of the exponential chaining decay factor (α) is not merely a mathematical heuristic; it models a well-documented phenomenon in offensive security. Each transition across an architectural boundary—such as an orchestrator invoking an external API or a prompt gateway querying a vector database—imposes strict constraints on payload delivery.

As illustrated in Figure 4, the impact of α becomes exponentially pronounced as path length (k) increases. While baseline models lacking decay (effectively $\alpha = 1.0$) project a linear or mildly attenuated reduction in exploitability, lower α values drastically restrict the viability of long attack chains. An attacker attempting a 5-hop exploit in a system with $\alpha = 0.85$ faces an inherent theoretical friction penalty that reduces the probability of end-to-end success by over 47%, even before accounting for component-specific mitigations. This highlights why "assume breach" architectures, which intentionally increase the graph diameter between entry points and critical assets, are highly effective defense strategies.

Figure 4
The impact of the chaining decay factor (α) on cumulative path exploitability over multiple hops

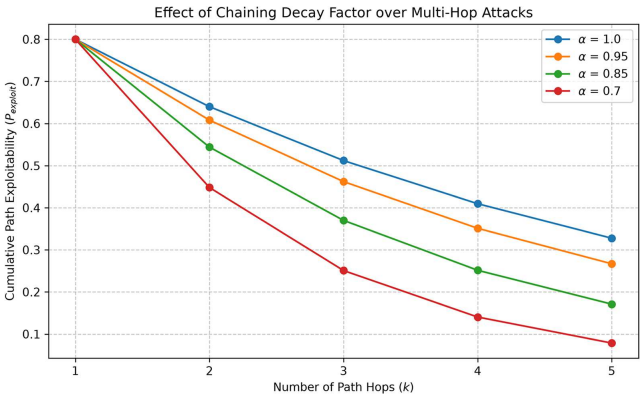


Table 3
Mitigation ROI prioritization for the RAG Chatbot topology

Rank	Proposed mitigation (RAG Chatbot)	Target layer	Systemic ΔScore
1	Implement Secure Output Guardrail (LLM-as-Judge)	Orchestrator	18.4
2	Enforce Parameterized Queries	Customer DB	12.1
3	Strict Input Regex Filtering	Prompt Gateway	8.5
4	Network Isolation (Internal VPC)	Vector DB	4.2

5.2. Operational integration via CI/CD pipelines

To realize the full value of the CAPS framework, organizations must transition from static, point-in-time risk assessments to continuous evaluation. We propose that the CAPS engine be integrated directly into Continuous Integration and Continuous Deployment (CI/CD) pipelines.

By defining the deployment architecture as code (e.g., via Terraform or standard JSON topologies), the CAPS engine can be executed automatically on every pull request. If a developer introduces a new tool (creating a new edge in the graph) or downgrades a mitigation, CAPS will instantly recalculate the maximal path score. If the new topology violates a pre-defined risk threshold (e.g., $S_{\max} > 50$), the build can be automatically failed. This operationalizes "Security by Design" in LLM ecosystems, providing immediate, quantitative feedback to engineering teams before compositional vulnerabilities reach production environments.

5.3. Computational scalability in CI/CD environments

To verify the efficiency of the proposed modified DFS algorithm within operational CI/CD pipelines, we conducted a computational performance test on synthetically generated graph topologies. As shown in Table 4, the algorithm scales efficiently even on very large and complex environments exceeding the assumed maximum path length ($k_{\max} = 10$). For a typical enterprise deployment (e.g., 500 nodes and $k_{\max} = 15$), the execution latency remains well under a second, demonstrating that CAPS can provide real-time risk assessments without blocking automated deployment workflows.

Table 4
Computational performance of the CAPS algorithm across varying graph complexities

Nodes ($\|V\|$)	Edges ($\|E\|$)	Max Path (k_max)	Execution latency (ms)
50	150	10	12.4
500	2500	15	145.2
5,000	45,000	20	1284.7

5.4. Limitations and future work

While CAPS provides a robust structural and mathematical foundation, the accuracy of its outputs is inherently bounded by the fidelity of its inputs. The model currently relies on heuristic or expert-consensus estimations for base exploitability (E) and mitigation effectiveness (M). To address this reliance, operational deployments of CAPS should ground these initial values in empirical cybersecurity databases. Specifically, the base vulnerability (E) can be quantitatively derived using the Exploit Prediction Scoring System, which provides data-driven probability estimates of exploitability in the wild. Similarly, mitigation effectiveness (M) can be standardized by mapping defenses to the MITRE D3FEND matrix, assigning efficacy weights based on historical deployment telemetry rather than qualitative consensus. In rapidly evolving fields like adversarial machine learning, these values can fluctuate wildly based on the discovery of novel jailbreak techniques or prompt injection variants.

Future work will focus on dynamic parameter estimation and dependency-aware modeling. By integrating automated LLM red-teaming frameworks (such as Promptfoo or Garak) with the CAPS engine, we can continuously bombard components with synthetic payloads. The empirical success rates of these localized attacks can then be fed back into the CAPS graph, enabling dynamic, real-time recalculation of E_{base} and M(mit). Additionally, Equation (1) currently models Effective Exploitability by multiplying survival probabilities, implicitly assuming statistical independence among defenses. In practice, overlapping security mechanisms (e.g., regex filters and WAFs) often exhibit correlated failure modes. Future iterations of CAPS will explore Bayesian Networks or Copula functions to capture these dependencies, yielding a more robust, self-tuning threat assessment ecosystem.

5.5. Dual-use and broader impact

As with any advanced security assessment framework, CAPS presents potential dual-use risks. In theory, an adversary could use the CAPS engine to identify the path of least resistance (the critical path) through an LLM deployment stack, optimizing their attack strategy. However, we assess this risk as minimal in practice. The CAPS algorithm strictly requires high-fidelity, white-box architectural telemetry—including comprehensive knowledge of internal graph topologies, deployed security guardrails, and precise component vulnerabilities. This level of internal visibility is inherently unavailable to external black-box attackers, making CAPS predominantly an asymmetric advantage for defenders and internal security engineering teams.

6. Conclusion

As LLMs evolve into deeply embedded autonomous agents, security paradigms must shift from isolated component evaluation to holistic systemic risk analysis. This paper introduced the CAPS framework, utilizing a graph-theoretic approach coupled with exponential chaining decay to mathematically model multi-hop exploit friction. By calculating the "Effective Exploitability" across trust boundaries, CAPS corrects the severe risk inflation inherent in traditional scoring methods. Our evaluations confirm that CAPS provides a superior, realistically calibrated risk assessment while delivering actionable, ROI-driven mitigation strategies, empowering organizations to optimally secure complex AI architectures.

Ethical Statement

This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

The data that support the findings of this study are openly available in the CAPS repository [GitHub] at https://github.com/vinhqdang/CAPS-Compositional-Attack-Path-Scoring-for-LLM-Deployment-Stacks.

Author Contribution Statement

Quang-Vinh Dang: Conceptualization, Methodology, Software, Writing – original draft. **Hoang-Viet Vu:** Software, Validation, Visualization. **Ngoc-Son-An Nguyen:** Investigation, Data curation. **Minh Ngoc Dinh:** Formal analysis, Investigation. **Dat Le:** Writing – review & editing, Supervision, Project administration.

References

[1] Manadhata, P. K., & Wing, J. M. (2010). An attack surface metric. *IEEE Transactions on Software Engineering*, *37*(3), 371–386. https://doi.org/10.1109/TSE.2010.60

[2] Mazeika, M., Phan, L., Yin, X., Zou, A., Wang, Z., Mu, N., . . . , & Hendrycks, D. (2024). HarmBench: A standardized evaluation framework for automated red teaming and robust refusal. In *Proceedings of the 41st International Conference on Machine Learning*, 35181–35224.

[3] Chao, P., Debenedetti, E., Robey, A., Andriushchenko, M., Croce, F., Sehwag, V., . . . , & Wong, E. (2024). Jailbreakbench: An open robustness benchmark for jailbreaking large language models. *Advances in Neural Information Processing Systems*, *37*, 55005–55029. https://doi.org/10.52202/079017-1745

[4] Souly, A., Lu, Q., Bowen, D., Trinh, T., Hsieh, E., Pandey, S., . . . , & Toyer, S. (2024). A strongreject for empty jailbreaks. *Advances in Neural Information Processing Systems*, *37*, 125416–125440. https://doi.org/10.52202/079017-3984

[5] Zhang, H., Huang, J., Mei, K., Yao, Y., Wang, Z., Zhan, C., . . . , & Zhang, Y. (2025). Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *International Conference on Learning Representations*, *2025,* 35331–35366.

[6] Zou, A., Wang, Z., Carlini, N., Nasr, M., Kolter, J. Z., & Fredrikson, M. (2023). *Universal and transferable adversarial attacks on aligned language models*. arXiv Preprint: 2307.15043.

[7] Chao, P., Robey, A., Dobriban, E., Hassani, H., Pappas, G. J., & Wong, E. (2025). Jailbreaking black box large language models in twenty queries. In *2025 IEEE Conference on Secure and Trustworthy Machine Learning*, 23–42. https://doi.org/10.1109/SaTML64287.2025.00010

[8] Liu, X., Li, P., Suh, G. E., Vorobeychik, Y., Mao, Z., Jha, S., & Xiao, C. (2025). Autodan-turbo: A lifelong agent for strategy self-exploration to jailbreak LLMs. In *International Conference on Learning Representations, 2025*, 10313–10360.

[9] Russinovich, M., Salem, A., & Eldan, R. (2025). Great, now write an article about that: The crescendo multi-turn LLM jailbreak attack. In *34th USENIX Security Symposium (USENIX Security 25)*, 2421–2440.

[10] Anil, C., Durmus, E., Panickssery, N., Sharma, M., Benton, J., Kundu, S., . . . , & Duvenaud, D. (2024). Many-shot jailbreaking. *Advances in Neural Information Processing Systems*, *37*, 129696–129742. https://doi.org/10.52202/079017-4121

[11] Hughes, J., Price, S., Lynch, A., Schaeffer, R., Barez, F., Somani, A., . . . , & Sharma, M. (2026). Best-of-n jailbreaking. *Advances in Neural Information Processing Systems*, *38*, 73137–73221.

[12] Yao, Y., Tong, X., Wang, R., Wang, Y., Li, L., Liu, L., . . . , & Wang, Y. (2025). A mousetrap: Fooling large reasoning models for jailbreak with chain of iterative chaos. In *Findings of the Association for Computational Linguistics: ACL 2025*, 7837–7855. https://doi.org/10.18653/v1/2025.findings-acl.408

[13] Anh, N. V., Zhao, S., Dao, G., Hu, R., Xie, Y., Wu, X., & Tuan, L. A. (2026). Three minds, one legend: Jailbreak large reasoning model with adaptive stacked ciphers. In *Findings of the Association for Computational Linguistics: ACL 2026*, 7139–7159. https://doi.org/10.18653/v1/2026.findings-acl.355

[14] Ben-Tov, M., Geva, M., & Sharif, M. (2026). Universal jailbreak suffixes are strong attention hijackers. *Transactions of the Association for Computational Linguistics*, *14*, 1028–1050. https://doi.org/10.1162/TACL.a.695

[15] Hagendorff, T., Derner, E., & Oliver, N. (2026). Large reasoning models are autonomous jailbreak agents. *Nature Communications*, *17*, 1–8. https://doi.org/10.1038/s41467-026-69010-1

[16] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, 79–90. https://doi.org/10.1145/3605764.3623985

[17] Zhan, Q., Liang, Z., Ying, Z., & Kang, D. (2024). InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, 10471–10506. https://doi.org/10.18653/v1/2024.findings-acl.624

[18] Debenedetti, E., Zhang, J., Balunovic, M., Beurer-Kellner, L., Fischer, M., & Tramèr, F. (2024). AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. *Advances in Neural Information Processing Systems*, *37*, 82895–82920. https://doi.org/10.52202/079017-2636

[19] Chang, H., Jun, Y., & Lee, H. (2025). ChatInject: Abusing chat templates for prompt injection in LLM agents. In *ICLR 2026 Conference*, 1–28.

[20] Johnson, S., Pham, V., & Le, T. (2025). The dangers of indirect prompt injection attacks on LLM-based autonomous web navigation agents: A demonstration. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 729–738. https://doi.org/10.18653/v1/2025.emnlp-demos.55

[21] He, P., Lin, Y., Dong, S., Xu, H., Xing, Y., & Liu, H. (2025). Red-teaming LLM multi-agent systems via communication attacks. In *Findings of the Association for Computational Linguistics: ACL 2025*, 6726–6747. https://doi.org/10.18653/v1/2025.findings-acl.349

[22] Wang, Z., Gao, Y., Wang, Y., Liu, S., Sun, H., Cheng, H., . . . , & Li, X. (2026). MCPTox: A benchmark for tool poisoning on real-world MCP servers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, *40*(42), 35811–35819. https://doi.org/10.1609/aaai.v40i42.40895

[23] Huang, C., Huang, X., Tran, N. P., & Milani Fard, A. (2026). Model context protocol threat modeling and analysis of vulnerabilities to prompt injection with tool poisoning. *Journal of Cybersecurity and Privacy*, *6*(3), 1–40. https://doi.org/10.3390/jcp6030084

[24] Mas, L., Vilaplana, J., Rius, J., Spaimoc, R., & Mateo, J. (2026). The trustworthy Model Context Protocol (MCP) registry: An architectural blueprint for cryptographic provenance and runtime integrity. *Future Internet*, *18*(5), 1–26. https://doi.org/10.3390/fi18050243

[25] Zou, W., Geng, R., Wang, B., & Jia, J. (2025). PoisonedRAG: Knowledge corruption attacks to retrieval-augmented generation of large language models. In *34th USENIX Security Symposium (USENIX Security 25)*, 3827–3844.

[26] Zhang, B., Chen, Y., Liu, Z., Nie, L., Li, T., Liu, Z., & Fang, M. (2026). Practical poisoning attacks against retrieval-augmented generation. In *Proceedings of the 31st ACM Symposium on Access Control Models and Technologies*, 33–44. https://doi.org/10.1145/3750555.3811900

[27] Chang, Z., Li, M., Jia, X., Wang, J., Huang, Y., Jiang, Z., . . . , & Wang, Q. (2025). One shot dominance: Knowledge poisoning attack on retrieval-augmented generation systems. In C. Christodoulopoulos, T. Chakraborty, C. Rose, & V. Peng (Eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025* (pp. 18811–18825). Association for Computational Linguistics. https://doi.org/10.18653/v1/2025.findings-emnlp.1023

[28] Ha, H., Zhan, Q., Kim, J., Bralios, D., Sanniboina, S., Peng, N., . . . , & Ji, H. (2026). MM-PoisonRAG: Disrupting multimodal RAG with local and global knowledge poisoning attacks. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, *1*, 33804–33826. https://doi.org/10.18653/v1/2026.acl-long.1558

[29] Zhang, B., Xin, H., Fang, M., Liu, Z., Yi, B., Li, T., & Liu, Z. (2025). Traceback of poisoning attacks to retrieval-augmented generation. In *Proceedings of the ACM on Web Conference 2025*, 2085–2097. https://doi.org/10.1145/3696410.3714756

[30] Dong, S., Xu, S., He, P., Li, Y., Tang, J., Liu, T., . . . , & Xiang, Z. (2026). Memory injection attacks on LLM agents via query-only interaction. *Advances in Neural Information Processing Systems*, *38*, 46697–46731.

[31] Markov, T., Zhang, C., Agarwal, S., Nekoul, F. E., Lee, T., Adler, S., . . . , & Weng, L. (2023). A holistic approach to undesired content detection in the real world. In *Proceedings of the AAAI Conference on Artificial Intelligence*, *37*(12), 15009–15018. https://doi.org/10.1609/aaai.v37i12.26752

[32] Debenedetti, E., Shumailov, I., Fan, T., Hayes, J., Carlini, N., Fabian, D., . . . , & Tramèr, F. (2025). *Defeating prompt injections by design*. arXiv Preprint: 2503.18813.

[33] Zou, A., Phan, L., Wang, J., Duenas, D., Lin, M., Andriushchenko, M., . . . , & Hendrycks, D. (2024). Improving alignment and robustness with circuit breakers. *Advances in Neural Information Processing Systems*, *37*, 83345–83373. https://doi.org/10.52202/079017-2651

[34] Chen, S., Piet, J., Sitawarin, C., & Wagner, D. (2025). StruQ: Defending against prompt injection with structured queries. In *34th USENIX Security Symposium (USENIX Security 25)*, 2383–2400.

[35] Li, Y., Ahn, S., Jiang, H., Abdi, A. H., Yang, Y., & Qiu, L. (2025). Securitylingua: Efficient defense of LLM jailbreak attacks via security-aware prompt compression. In *Second Conference on Language Modeling*, 1–16.

[36] Xiong, C., Qi, X., Chen, P. Y., & Ho, T. Y. (2025). Defensive prompt patch: A robust and generalizable defense of large language models against jailbreak attacks. In *Findings of the Association for Computational Linguistics: ACL 2025*, 409–437. https://doi.org/10.18653/v1/2025.findings-acl.23

[37] Sheyner, O., Haines, J., Jha, S., Lippmann, R., & Wing, J. M. (2002). Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, 273–284. https://doi.org/10.1109/SECPRI.2002.1004377

[38] Ammann, P., Wijesekera, D., & Kaushik, S. (2002). Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 217–224. https://doi.org/10.1145/586110.586140

[39] Chau, D. H. P., Nachenberg, C., Wilhelm, J., Wright, A., & Faloutsos, C. (2011). Polonium: Tera-scale graph mining and inference for malware detection. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, 131–142. https://doi.org/10.1137/1.9781611972818.12

[40] Zhou, Y., Liu, S., Siow, J., Du, X., & Liu, Y. (2019). Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in Neural Information Processing Systems*, *32*.

[41] Mauri, L., & Damiani, E. (2022). Modeling threats to AI-ML systems using STRIDE. *Sensors*, *22*(17), 6662. https://doi.org/10.3390/s22176662

[42] Moia, V. H. G., Sanz, I. J., Rebello, G. A. F., de Meneses, R. D., Hitaj, B., & Lindqvist, U. (2026). LLM in the middle: A systematic review of threats and mitigations to real-world LLM-based systems. *Computer Science Review*, *61*, 100916. https://doi.org/10.1016/j.cosrev.2026.100916

How to Cite: Dang, Q.-V., Vu, H.-V., Nguyen, N.-S.-A., Dinh, M. N., & Le, D. (2026). CAPS: Compositional Attack Path Scoring for LLM Deployment Stacks. *Artificial Intelligence and Applications*. https://doi.org/10.47852/bonviewAIA620210609