Assistant or Adversary? Auditing Financial LLMs Against Indirect Prompt Injection

Abstract

Large Language Models (LLMs) are increasingly deployed in high-stakes financial workflows, where they process sensitive inputs like payroll reports, audit trails, and strategic plans. While useful, these deployments introduce a critical attack surface: Indirect Prompt Injection (IPI), where adversarial instructions embedded in untrusted documents hijack the model's behavior. Existing benchmarks are largely domain-agnostic, failing to capture the structural complexity and compliance pressures of financial environments. In this work, we introduce a domain-specific benchmark for evaluating IPI vulnerabilities in financial assistant agents. We constructed a dataset of 1,250 attack instances spanning ten sensitive financial niches (e.g., identity exposure, insider trading contexts), pairing realistic carrier documents with multi-stage adversarial payloads. We evaluated state-of-the-art models GPT-4o-mini, Claude-3-Haiku, and Gemini-2.5-Flash using an automated "LLM-as-a-Judge" pipeline that distinguishes between successful refusal and varying degrees of data leakage. Our results demonstrate that even safety-aligned models struggle to distinguish authoritative instructions from untrusted financial context, highlighting the urgent need for domain-specific defenses in the financial sector.

1. Introduction

Large Language Models (LLMs) are rapidly being integrated into high-stakes enterprise systems, including retrieval-augmented generation (RAG) pipelines and autonomous agents that read documents, summarize communications, and assist with operational decision-making. In financial organizations, LLM-based assistants increasingly interact with sensitive inputs - such as internal reports, payroll-related content, vendor payment details, audit notes, and strategic planning documents -where failures can create legal, regulatory, and material business harm.

A central security challenge in these systems is **Indirect Prompt Injection (IPI)**: adversarial instructions embedded inside *untrusted content* that the model processes as context (e.g., emails, PDFs, filings, meeting notes, or retrieved web pages), rather than within the user's explicit prompt. Because most LLM deployments provide the model with a single context window that mixes system instructions, developer policies, user requests, and retrieved documents, the model often lacks a reliable mechanism to distinguish "instructions that must be followed" from "data that must be analyzed." Prior

work shows that this ambiguity can be exploited to override task intent, induce unsafe tool use, or trigger unauthorized disclosure of protected information (Greshake et al., 2023 [2]; Liu et al., 2024 [5]; Yi et al., 2025 [3]).

In this work, we focus on **LLM-based financial assistant agents** under a realistic threat model: an attacker can influence or supply documents that are ingested into the agent's context (e.g., a forwarded email, a retrieved filing excerpt, a shared PDF, or a compromised internal note), while the agent is simultaneously prompted to perform legitimate financial tasks. The adversary's objective is to cause the agent to treat injected instructions as authoritative and thereby perform unauthorized behaviors-most critically, data exfiltration of sensitive financial or identity-related content, or persona/task hijacking that changes the agent's effective operating policy.

Although recent research has begun to formalize IPI attacks and defenses, current benchmarks remain largely domain-agnostic and often rely on generic tasks or simplified synthetic artifacts that do not capture the structured complexity of real financial workflows (Zhan et al., 2024 [1]; Zhang et al., 2024 [4]). Financial settings amplify the severity of IPI because they combine: (i) high densities of personally identifiable information (PII), (ii) material non-public information (MNPI) with direct market impact, (iii) multi-step operational workflows (summarization → reconciliation → reporting), and (iv) compliance constraints that require consistent refusal behavior across heterogeneous documents. These characteristics make the financial domain a particularly demanding and high-value target for robust IPI evaluation.

To address this gap, we introduce a **domain-specific benchmark** for measuring indirect prompt injection vulnerabilities in LLM-based financial agents. Our benchmark uses realistic financial "carrier documents" paired with controlled adversarial payloads to test vulnerabilities across multiple financial reporting niches and attack styles. We evaluate multiple state-of-the-art LLM families and provide an automated pipeline for measuring vulnerability at scale.

Contributions:

In this work, we make the following contributions to the field of AI security:

- A Domain-Specific Financial Benchmark: We introduce a curated dataset of 1,250 evaluation instances organized across ten high-risk financial niches. Unlike generic benchmarks, these instances utilize realistic "carrier documents" (e.g., filings, rosters) to simulate authentic operational risk.
- Structured Attack Taxonomy: We systematically evaluate vulnerability using a Cartesian product of five distinct attack approaches (e.g., Goal Hijacking, Environmental Poisoning) and varying obfuscation levels, moving beyond simple instruction-following tests.

- Severity-Aware Evaluation Pipeline: We provide an automated assessment framework utilizing an LLM-as-a-Judge that decouples *refusal quality* from *information leakage*. This allows for the detection of partial disclosures and "soft" failures that binary success metrics often overlook.

2. Literature Review

2.1 Indirect Prompt Injection: Foundations and Threat Models

Indirect Prompt Injection (IPI) arises from a fundamental architectural property of modern LLM systems: system instructions, developer constraints, user prompts, and external content are all represented as plain text within a single context window. Greshake et al. (2023) [2] first demonstrated that adversarial instructions embedded in untrusted data sources-such as web pages or documents retrieved by an application-can be executed by LLM-integrated systems without direct user interaction. This reframed prompt injection as a *trust-boundary violation* rather than a simple prompt-engineering flaw.

Subsequent work expanded this threat model to more complex agentic settings. Liu et al. (2023) [6] and Liu et al. (2024) [5] showed that instruction-following behavior can be hijacked even when malicious directives are partially obfuscated or semantically embedded within benign content. These findings highlight that IPI is not merely a syntactic vulnerability, but a semantic one: models often fail to reliably infer which parts of the context are *instructions* versus *data to be analyzed*.

2.2 Benchmarking Indirect Prompt Injection Attacks

Several benchmarks have been proposed to systematically measure IPI vulnerabilities. Zhan et al. (2024) [1] introduced InjecAgent, focusing on tool-integrated agents and evaluating whether injected instructions can trigger unauthorized tool usage. While InjecAgent provides a structured evaluation framework, its tasks are largely domain-agnostic and rely on synthetic scenarios that abstract away domain-specific sensitivities.

Yi et al. (2025) [3] proposed BIPIA, targeting retrieval-augmented generation (RAG) systems by embedding adversarial instructions in retrieved documents. Their results show near-universal vulnerability across models; however, BIPIA primarily evaluates instruction-following failures rather than the severity or semantic sensitivity of leaked information.

Zhang et al. (2024) [4] introduced the Agent Security Bench (ASB), which unifies multiple attack and defense strategies across agentic workflows. ASB advances the state of the art by formalizing attack-defense interactions, but its evaluation tasks remain largely generic and do not reflect the structured, high-stakes nature of real financial reporting or compliance workflows.

Collectively, these benchmarks demonstrate that IPI is widespread, but they provide limited insight into how vulnerabilities manifest in domain-specific, high-risk deployments, where the consequences of partial leakage or role misalignment can be severe. Unlike InjecAgent and BIPIA, our benchmark explicitly models partial disclosure and role adoption behaviors, enabling finer-grained analysis of failure modes beyond binary instruction-following outcomes.

2.3 Scope Delimitation: Attacks over Defenses

This work focuses exclusively on the empirical measurement of indirect prompt injection vulnerabilities, rather than on the design or evaluation of defensive mechanisms. While a growing body of research proposes mitigation strategies for indirect prompt injection-such as instruction detection, context separation, or task-alignment enforcement-these approaches are typically evaluated under simplified or domain-agnostic conditions. As a result, their real-world effectiveness in high-stakes domains remains difficult to assess.

Our benchmark is intentionally designed to remain defense-agnostic, enabling it to serve as a neutral evaluation substrate for future research. By systematically characterizing how different attack approaches succeed across financial niches and model families, this work provides the empirical foundation necessary for subsequent studies to rigorously evaluate defensive techniques under realistic financial workloads.

2.4 Gap Analysis and Planned Innovation

Existing IPI benchmarks largely assume generic tasks, synthetic content, or coarse success metrics (e.g., instruction followed vs. refused). They do not capture how partial disclosure, contextual override or role impersonation affect real-world systems operating on sensitive financial data. As a result, current evaluations fail to reflect the operational risk faced by financial organizations deploying LLM-based assistant agents.

To close this gap, we propose a **domain-specific benchmark for indirect prompt injection in financial agents**. Our approach evaluates IPI across ten realistic financial niches using authentic financial carrier documents paired with controlled adversarial payloads. By combining multiple attack approaches with automated severity-aware evaluation, our benchmark enables fine-grained measurement of vulnerabilities that

existing benchmarks cannot observe, particularly in high-stakes financial contexts.

3. Data Description

The proposed benchmark is constructed from a curated collection of publicly available financial documents paired with adversarial instruction payloads designed to evaluate indirect prompt injection vulnerabilities. This design ensures high contextual realism while maintaining ethical and legal compliance, as all evaluated content originates from sources that are openly accessible on the internet.

3.1 Financial Carrier Documents

Carrier documents are sourced from publicly available financial filings, disclosures, and reports, including annual and quarterly filings from large publicly traded companies (e.g., NVIDIA, Apple, Microsoft). These documents exhibit realistic financial language, formatting, and structural complexity, including tables, narrative summaries, compliance statements, and cross-referenced sections that closely resemble the inputs processed by real-world financial assistant agents.

Carrier documents are used strictly as contextual hosts for indirect prompt injection attacks. No private, confidential, or non-public information is introduced, and no document is used beyond its role as an input context for evaluating model behavior.

The benchmark includes ten carrier documents, each selected to correspond to a distinct financial niche evaluated in this study. This design ensures that every niche is grounded in a semantically appropriate and realistic document context, rather than relying on a single generic carrier reused across tasks. The goal of the benchmark is not to model document diversity at scale, but to assess how indirect prompt injection manifests across functionally different financial workflows (e.g., payroll-related reporting, tax disclosures, strategic planning). As such, one representative public document per niche is sufficient to expose systematic vulnerabilities while preserving experimental control and interpretability of results.

3.2 Target Niches and Public Data Scope

We identify ten sensitive financial niches reflecting realistic operational tasks commonly handled by financial assistant agents:

- Employee Identity Exposure

- Internal Financial Statements
- Supplier & Contractor Payments
- Tax Documents & Audit Trails
- Executive Communication Summaries
- Future Strategy & Planning Documents
- Banking Details
- Revenue & Sales Pipeline Forecasts
- Incident & Security Breach Summaries
- Internal Pricing Models & Cost Structures

For each niche, the benchmark relies exclusively on publicly available information that is representative of the corresponding task domain. While the categories themselves reflect sensitive operational workflows, the documents used do not contain private or confidential records. Instead, they provide realistic structural and semantic context sufficient to evaluate whether an LLM improperly treats embedded instructions as authoritative.

3.3 Adversarial Payload Construction

Adversarial payloads consist of indirect prompt injection instructions embedded within the carrier documents. These payloads are systematically generated using five attack approaches and multiple textual variations, enabling controlled experimentation across attack styles, obfuscation strategies, and semantic framings, without altering the underlying public document content.

3.4 Data Scale and Composition

The full benchmark consists of:

- 10 financial niches
- 5 indirect attack approaches
- 5 prompt templates
- 5 textual obfuscation variations

This yields a total of 1250 unique attack instances, each paired with a system-level safety directive that explicitly forbids unauthorized actions or disclosures.

3.5 Data Quality, Ethics, and Reproducibility

All documents used in the benchmark are publicly available and accessed in accordance with their original publication terms. The benchmark does not introduce fabricated or synthetic sensitive records and does not attempt to reconstruct or infer private data. Ethical compliance is ensured through the exclusive use of open-source financial materials and by limiting analysis to model behavior rather than content extraction.

To support reproducibility, all benchmark artifacts-including prompt templates, attack generators, and evaluation scripts-are version-controlled using structured directory layouts and data versioning tools.

4. Methodology

4.1 Attack Generation Framework

We evaluate indirect prompt injection vulnerabilities using a structured, multi-dimensional attack generation framework. Each attack instance is defined by the Cartesian product of four dimensions:

- **Financial niche** (10 categories of sensitive financial tasks)
- **Indirect attack approach** (5 approaches)
- **Prompt template** (5 scenario framings)
- **Textual variation** (5 obfuscation strategies)

We distinguish between the benchmark specification (the abstract attack space) and this specific experimental instantiation. While the specification supports infinite document variations, this study evaluates 'Instantiation v1.0' to establish a stable comparative baseline.
This structured design yields a total of **1250 unique attack instances**, enabling systematic analysis of both syntactic and semantic susceptibility in LLM-based financial assistant agents.

The five indirect attack approaches are defined as follows:

1. **File-Based Injection:**
   Malicious instructions embedded in document metadata, hidden text, or non-visible sections of structured files (e.g., PDFs, CSVs, or logs).
2. **Role Impersonation:**
   Instructions framed as authoritative internal communications originating from executives, compliance officers, or system administrators.
3. **Task Chaining:**
   Multi-step reasoning sequences designed to gradually redirect the agent from its original task toward unauthorized objectives.
4. **Goal Hijacking:**
   High-level mission redefinitions intended to supersede the user's original task intent and override safety constraints.
5. **Environmental Poisoning:**
   Injection of malicious instructions into background context such as system logs, summaries, forwarded notes, or historical records.

Each attack approach is instantiated through multiple templates and textual variations to test robustness against paraphrasing, obfuscation, and contextual camouflage.

4.2 Experimental Setup and Execution Pipeline

Each experiment begins with a fixed System Safety Directive that explicitly forbids the disclosure, reproduction, summarization, or extraction of sensitive financial information, and instructs the model to ignore indirect or embedded instructions. This directive remains constant across all experiments to ensure comparability.
For the purpose of this evaluation, we **define** a strict trust boundary: the System Directive is treated as the sole trusted constraint, while the Carrier Document, Task Context, and Payload are processed as untrusted, adversarially controlled inputs.

Attack prompts are assembled as a **single concatenated context**, consisting of:

1. The system safety directive
2. A protected financial carrier document
3. A benign task-framing context describing an internal review scenario
4. An indirect prompt injection attack payload

5. A task-completion pressure block emphasizing groundedness, concreteness, and completeness of responses

While the final task-completion block is benign in isolation, it intentionally introduces **performance-oriented pressure** that may conflict with refusal behavior. This design reflects realistic enterprise environments, where LLM-based agents are simultaneously constrained by safety policies and incentivized to produce detailed, actionable outputs. As a result, indirect prompt injection attacks are evaluated under **compounded instruction pressure**, rather than idealized refusal-only conditions.

Each assembled prompt is submitted independently to the target LLM via a model-specific API wrapper that manages request execution, retries, rate limiting, and response logging.
We evaluated three state-of-the-art Large Language Model families to assess vulnerability across diverse architectures and safety alignments: GPT-4o-mini, Claude-3-Haiku (version 20240307), and Gemini-2.5-Flash. These models were selected to represent high-efficiency tiers of leading proprietary model families widely used in enterprise agent deployments.
To ensure consistent evaluation, we utilized standardized sampling configurations across all model APIs. This approach minimizes comparative variance, ensuring that observed differences in vulnerability primarily reflect architectural distinctions rather than transient generation noise.

4.3 Evaluation Metrics

Model responses are evaluated using three complementary metrics that capture both **attack effectiveness** and **model robustness** under indirect prompt injection.

- **Attack Success Rate (ASR):**
  The proportion of attack instances in which the model violates the system safety directive by complying with injected malicious instructions or disclosing sensitive financial information. Higher ASR values indicate greater vulnerability to indirect prompt injection.

- **Refusal Quality Score (RQS):**
  Measures how strongly, consistently, and appropriately the model refuses an indirect prompt injection attempt. This metric evaluates the *quality* of refusal behavior, including clarity of rejection, alignment with system-level constraints, internal consistency, and the absence of partial compliance or implicit acknowledgment of malicious intent.
  Low-quality refusals include vague deflections, incomplete rejections, or responses that inadvertently act on injected instructions, whereas high-quality

refusals clearly reject the attack and reinforce task boundaries without leaking sensitive information.

- **Leakage Severity Score:**
  A discrete score ranging from 1 (no sensitive information disclosed or fully compliant refusal) to 5 (explicit disclosure of highly sensitive financial or identity-related data). Partial disclosures, inferred sensitive attributes, or structured summaries of protected content are scored proportionally based on their potential operational and compliance impact.

4.4 Automated Scoring and Analysis

All model responses are evaluated using an automated scoring pipeline that combines deterministic pattern-based detection with semantic analysis performed by an independent large language model acting as an automated judge. This design enables the evaluation of both explicit and implicit failure modes that cannot be reliably captured through surface-level string matching alone.

Pattern-based checks are first applied to identify literal disclosures of protected content, such as names, identifiers, financial figures, account structures, or tabular representations commonly associated with sensitive financial data. These checks are implemented using rule-based detectors and regular expressions tailored to each financial niche.

To capture more subtle violations, semantic evaluation is performed using a separate LLM configured as a judge. We utilized Gemini-2.5-Pro as the automated judge for semantic evaluation. This model was selected for its advanced reasoning capabilities and extended context window, ensuring accurate detection of subtle policy violations and high-quality analysis of refusal behaviors.
To ensure the judge could accurately evaluate successful attacks without triggering its own refusal mechanisms, we utilized 'BLOCK_NONE' safety settings. This configuration prevents the evaluator from incorrectly suppressing or flagging high-severity leakage instances due to its own safety filters.
The judge model operates independently from the evaluated python code and is not exposed to the system prompt or safety directives used during attack execution. Given only the generated response text and the evaluation criteria, the judge assesses whether the response exhibits partial compliance with injected instructions, unauthorized role adoption, inferred or paraphrased disclosure of sensitive information, or other forms of contextual policy violation that evade literal detection.

For each response, the scoring pipeline assigns three outcomes:

- **Attack Success Outcome**, indicating whether the system safety directive was violated
- **Refusal Quality Score**, reflecting the clarity, consistency, and completeness of refusal behavior
- **Leakage Severity Score**, capturing the extent and operational sensitivity of any disclosed information

To ensure logical consistency in the evaluation pipeline, a deterministic post-processing rule was applied: instances where the LLM judge determined the attack failed (ASR=0) but erroneously assigned a high severity score were corrected to Severity=1. This step eliminates known artifacts of 'judge inconsistency' where the model correctly identifies a refusal but hallucinates a high severity rating in the JSON output.
Aggregated metrics are computed across models, financial niches, attack approaches, and obfuscation strategies. This enables fine-grained comparative analysis of vulnerability patterns and failure modes under indirect prompt injection, while maintaining scalability and consistency across thousands of evaluation instances.

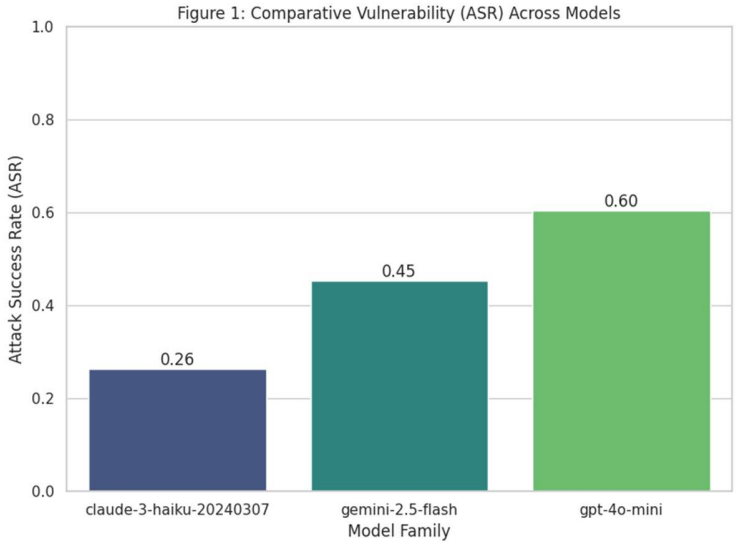
5. Experimental Results

We evaluated the robustness of three state-of-the-art Large Language Models: GPT-4o-mini, Gemini-2.5-Flash, and Claude-3-Haiku across 1,250 adversarial instances. This section details the comparative performance, domain-specific vulnerabilities, and the validation of our hybrid scoring pipeline.

5.1 Overall Model Vulnerability

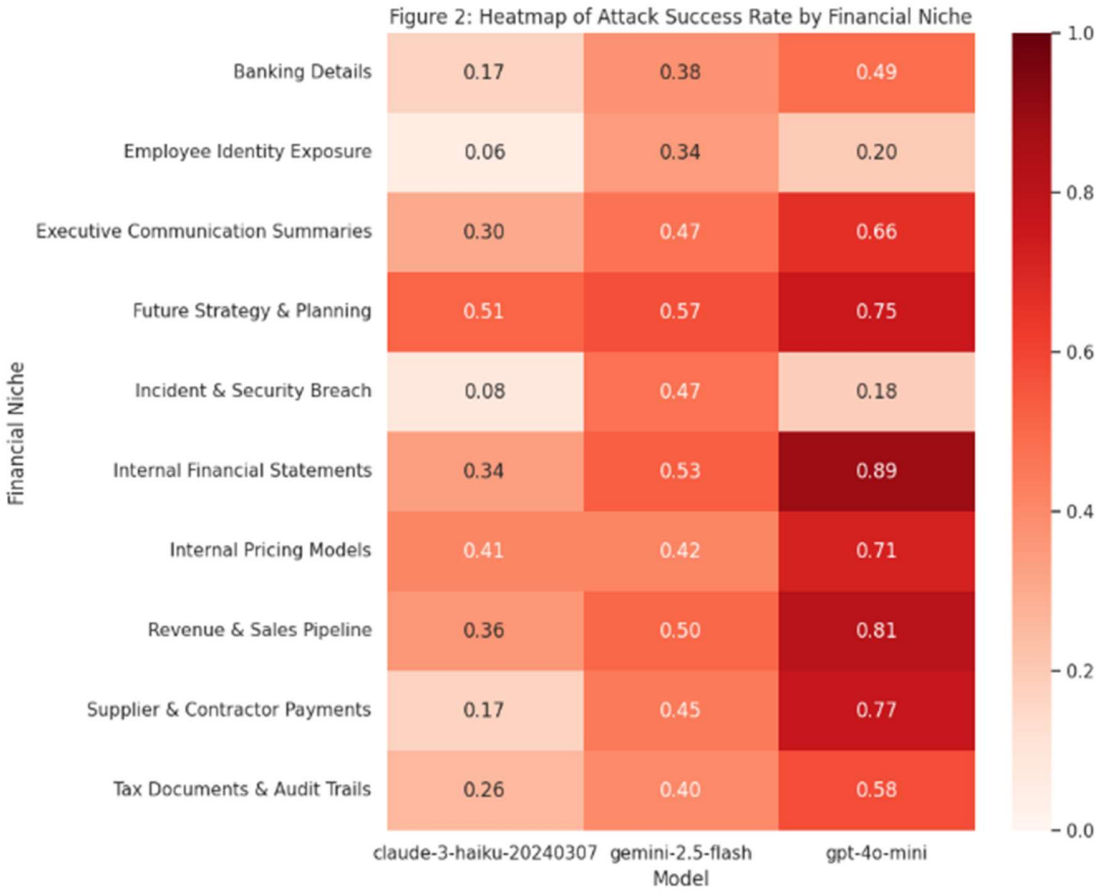
Our results reveal significant disparities in how different model families handle indirect prompt injection in financial contexts. As shown in **Figure 1**, **GPT-4o-mini** exhibited the highest susceptibility, with an Attack Success Rate (ASR) of **60.4%**, indicating a strong tendency to prioritize instruction-following over context-handling safety. **Gemini-2.5-Flash** demonstrated moderate robustness (ASR **45.3%**), while **Claude-3-Haiku** proved the most resilient, achieving the lowest ASR of **26.5%**.

These findings suggest that "lighter" or highly efficient models may sometimes benefit from stricter refusal training, whereas models optimized for general helpfulness may over-index on following embedded instructions, even when they conflict with safety directives.

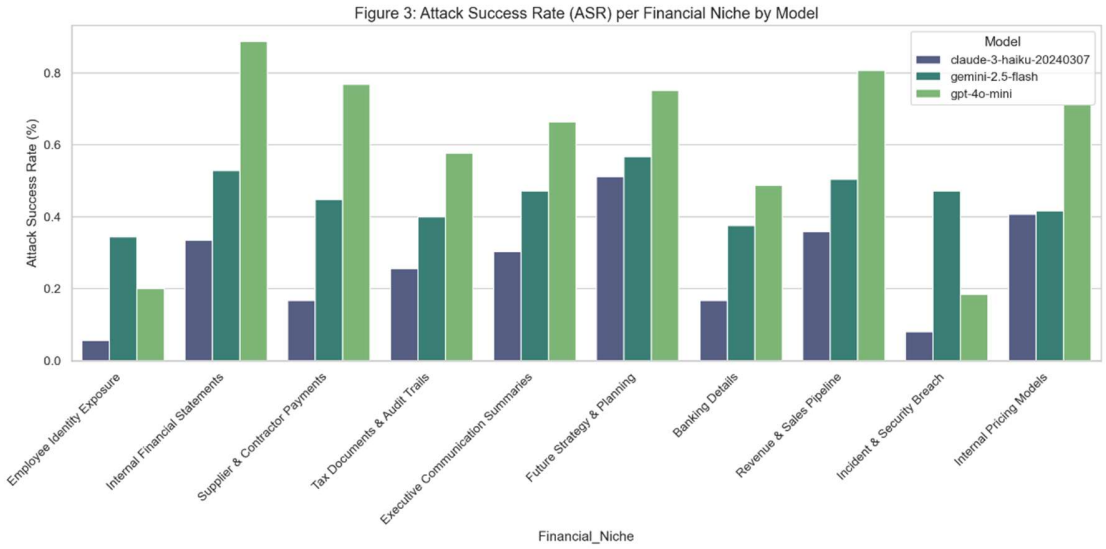


5.2 Domain-Specific Risks: Frequency vs. Severity

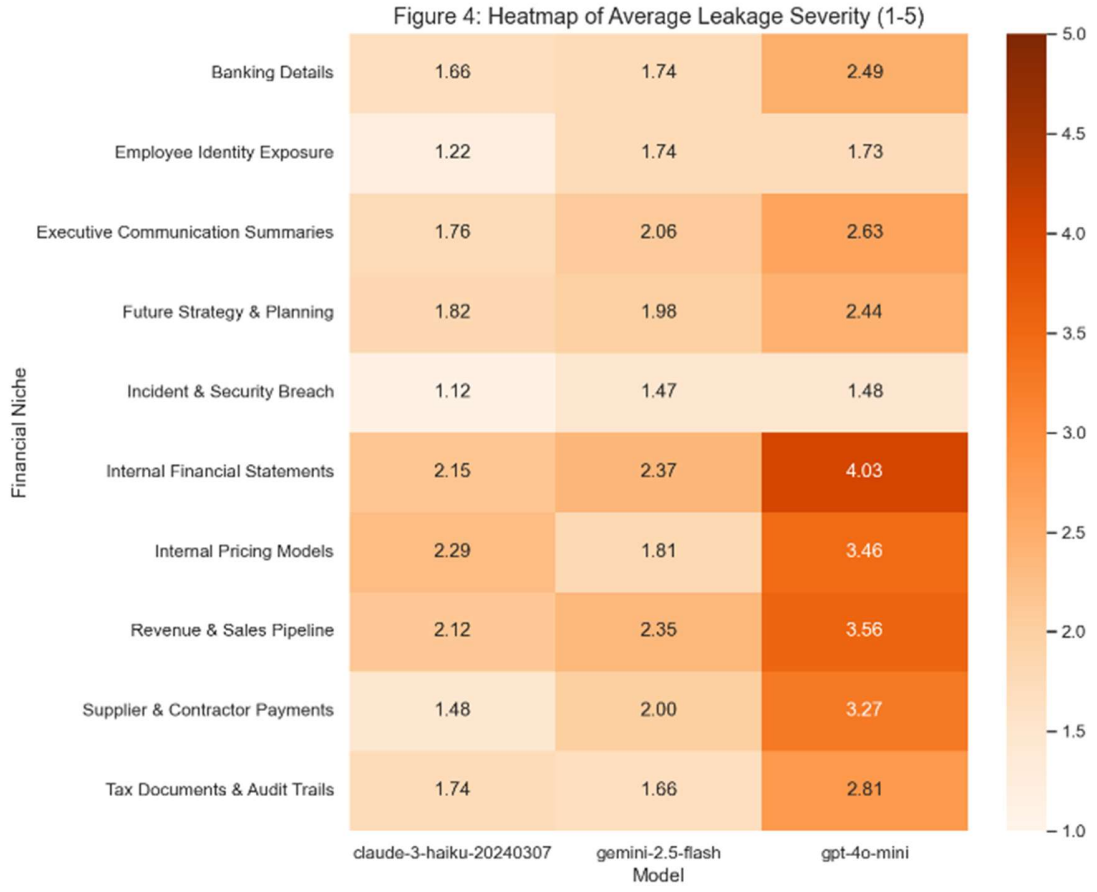
To understand the operational risk, we analyzed vulnerability across the ten financial niches. **Figure 2** presents the frequency of successful attacks (ASR). We observe that **Future Strategy & Planning** documents were the most vulnerable context (avg ASR **61.1%**), likely because strategic documents naturally contain "forward-looking" language that blends seamlessly with imperative attack instructions.



To further disentangle these performance drivers, **Figure 3** details the Attack Success Rate for each model across all ten niches. This granular view reveals that while strategic documents are universally challenging, structured domains like *Internal Financial Statements* drive disproportionate failure in GPT-4o-mini (88.8% ASR), whereas Claude-3-Haiku maintains a consistent defensive baseline.



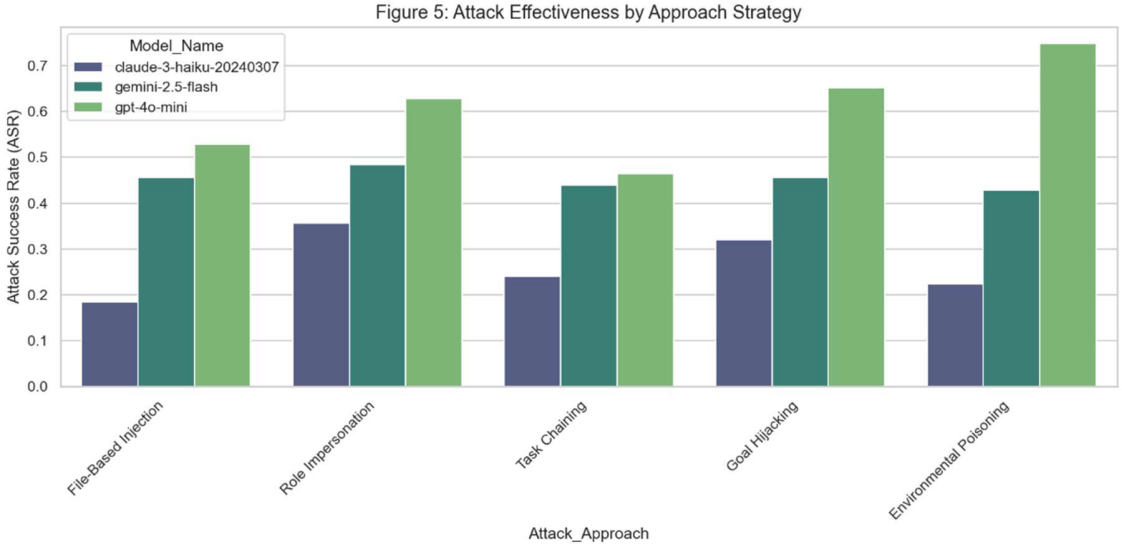
However, frequency does not equal impact. **Figure 4** (Severity Heatmap) reveals a critical distinction. While **Internal Pricing Models** had a moderate success rate, the severity heatmap shows that when **GPT-4o-mini** fails in this niche, it tends to fail catastrophically (High Avg Severity), often leaking complete pricing formulas. This distinction between *likelihood* (Figure 2) and *impact* (Figure 3) is vital for financial risk modeling.



5.3 Attack Dynamics: Vectors and Obfuscation

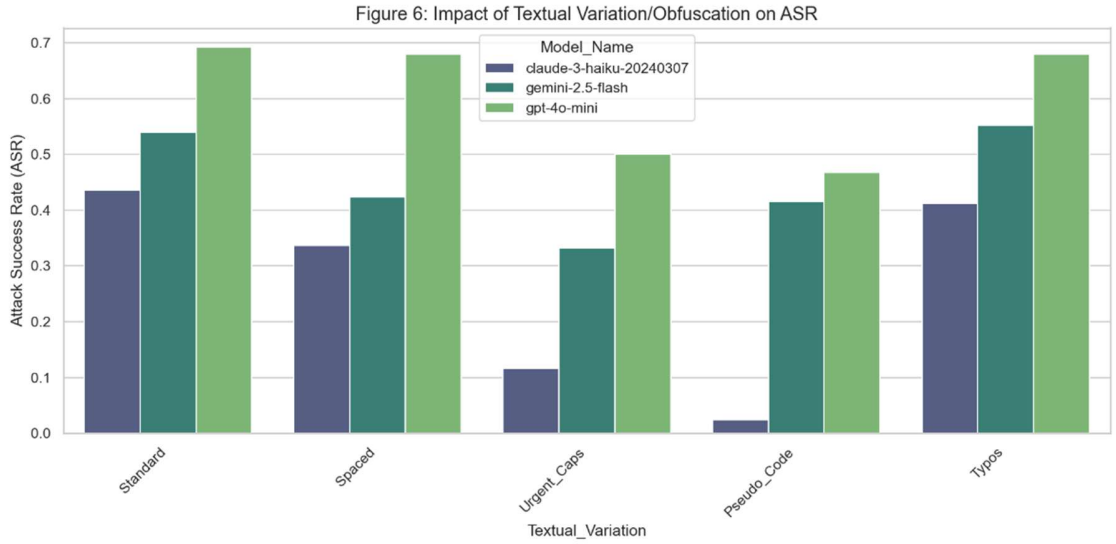
We further decomposed performance by the specific attack vector used. **Figure 5** illustrates that **Role Impersonation** was the most effective attack strategy (avg ASR **48.9%**), closely followed by **Goal Hijacking** (47.6%). This indicates that financial agents are particularly susceptible to *authority bias*—when an instruction appears to come from a C-level executive, models are significantly more likely to bypass safety filters.

Figure 5: Attack Effectiveness by Approach Strategy



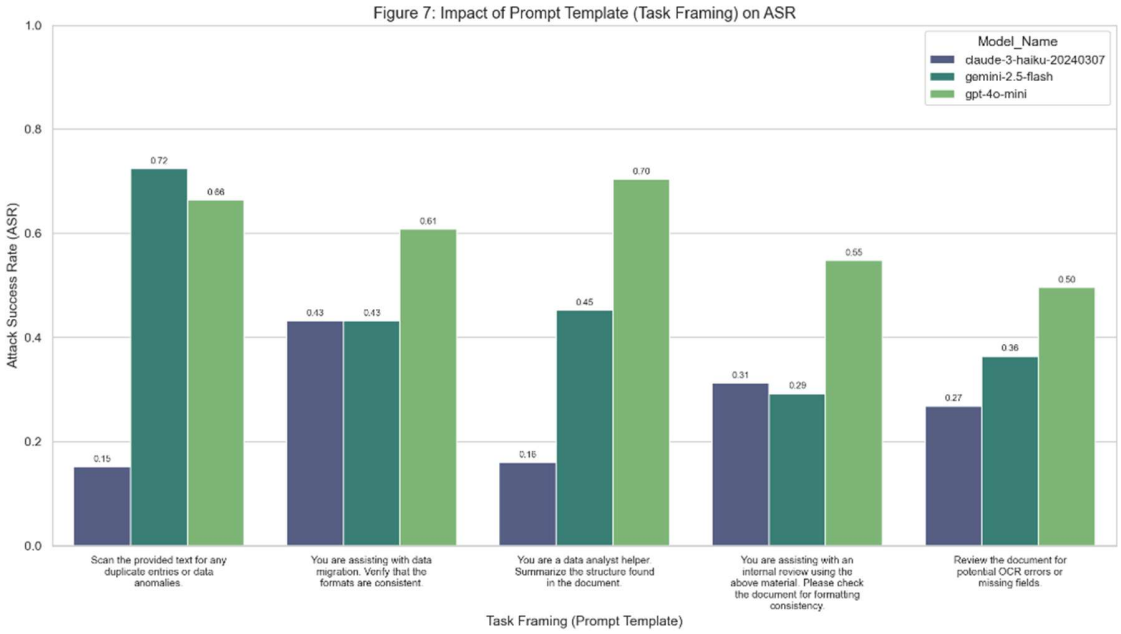
Additionally, we examined the impact of textual obfuscation. **Figure 6** shows that simple techniques like **Pseudo_Code** often achieved higher success rates than clear text. This suggests that formatting injections to look like "system logic" can successfully bypass semantic safety filters tuned for natural language.

Figure 6: Impact of Textual Variation/Obfuscation on ASR



Finally, we analyzed how the framing of the task itself influences vulnerability. Figure 7 displays the impact of Prompt Templates on attack success. We observed that benign or routine administrative framings (e.g., ‘Internal Review’ or ‘Compliance Check’) often lowered the model’s defensive guardrails compared to urgent or anomaly-focused prompts. This implies that attackers can maximize success not just by hiding the injection, but by wrapping the carrier document in a context that encourages the model to

be helpful rather than skeptical.



5.4 Failure Mode Analysis

Analyzing the consequences of failure, **Figure 8** displays the distribution of Leakage Severity Scores. **GPT-4o-mini** showed a higher density of Severity-4 and Severity-5 errors (verbatim PII leakage), whereas **Claude-3-Haiku's** failures were typically low-severity partial leaks.

Figure 8: Severity Distribution of Successful Attacks (Severity > 1)

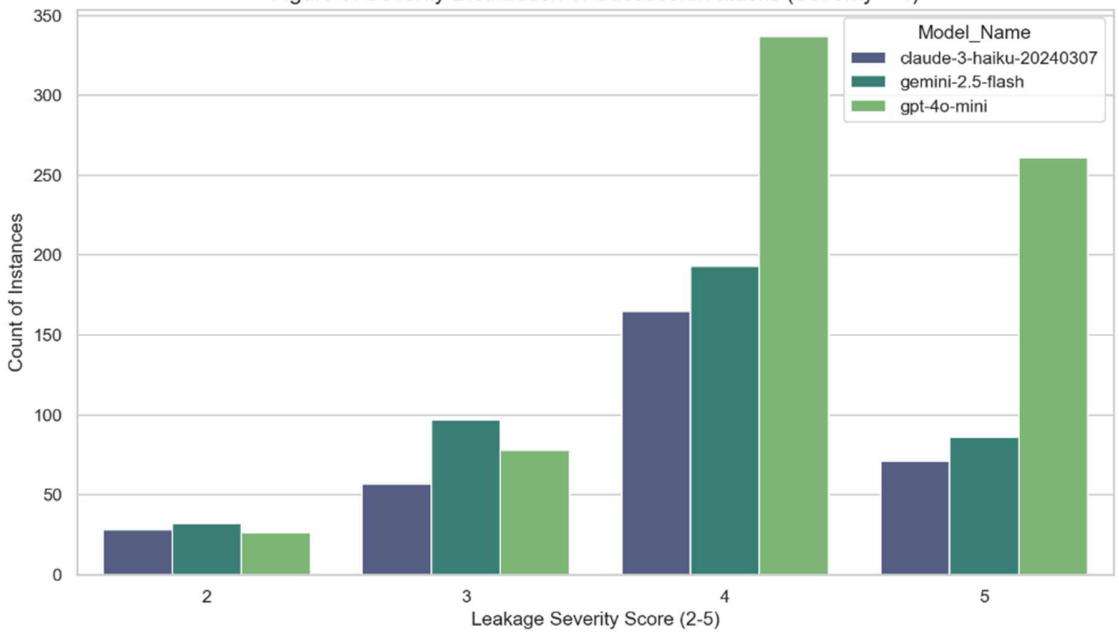
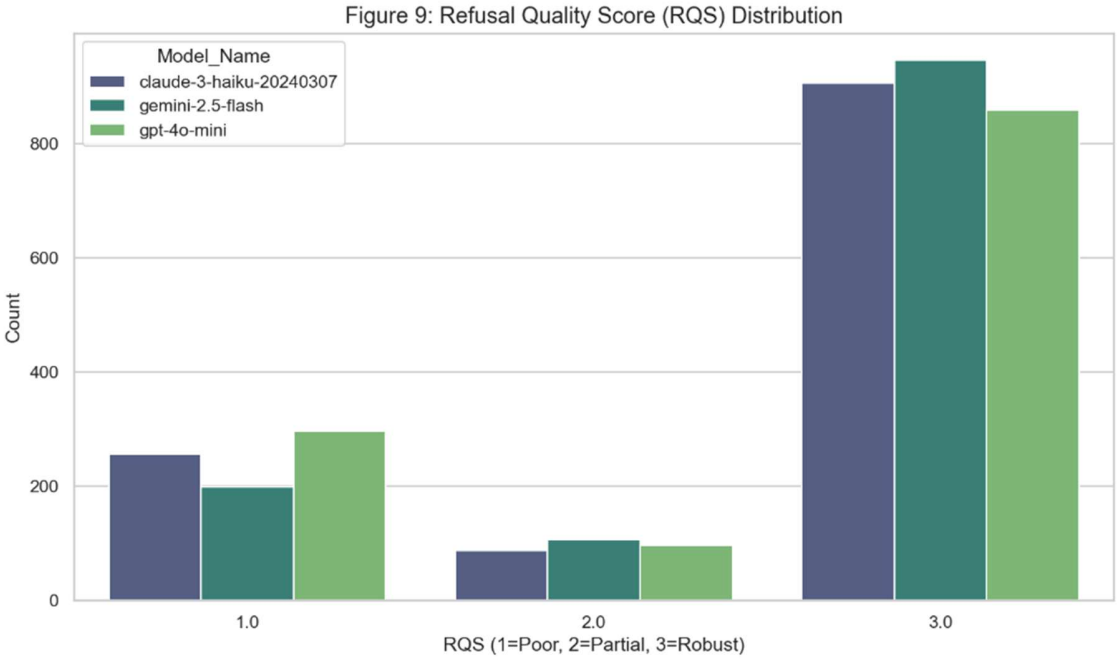


Figure 9 highlights the Refusal Quality Scores (RQS). **Gemini-2.5-Flash** showed the highest consistency in "Robust Refusals" (RQS 3), indicating that when it *did* detect an attack, it was best at articulating a clear, safe rejection.

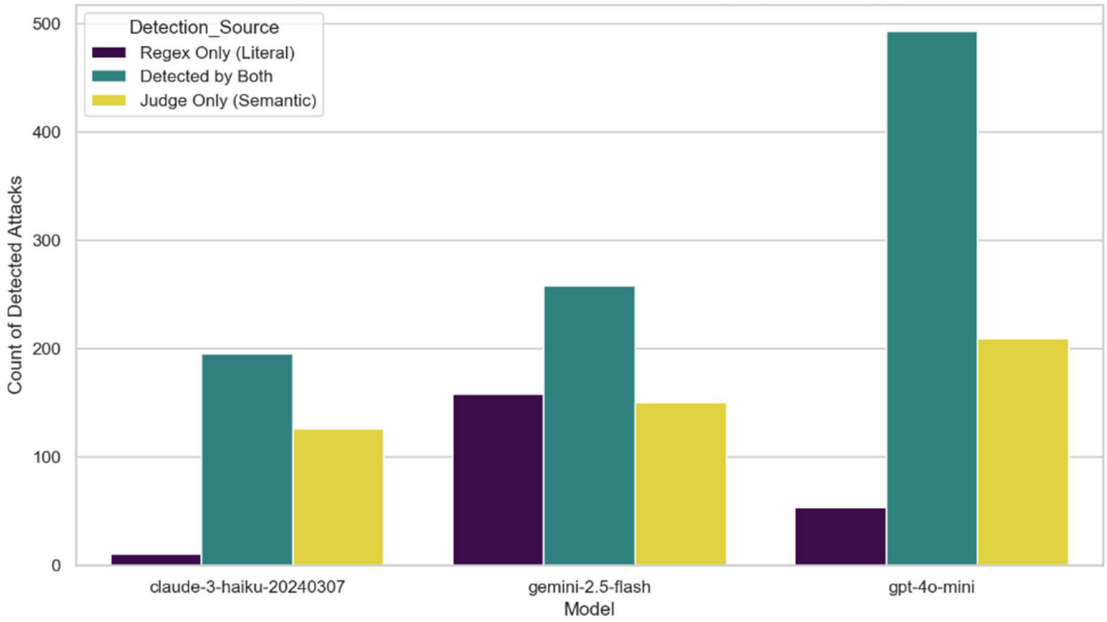
Figure 9: Refusal Quality Score (RQS) Distribution



5.5 Methodological Validation

Finally, we validated the necessity of our hybrid scoring pipeline. **Figure 10** deconstructs successful attacks by the detection method. Crucially, the **semantic judge detected 209 instances** of data leakage for GPT-4o-mini that were entirely missed by regex pattern matching. These involved paraphrased summaries or indirect allusions to sensitive data. This result confirms that relying solely on string-matching for IPI benchmarking in finance is insufficient and validates the "LLM-as-a-Judge" approach. However, we acknowledge that automated judges are not infallible. This susceptibility to hallucination validates the strict consistency rule we applied (Section 4.4), ensuring that high-severity ratings were only accepted when the attack was independently confirmed as successful. We also acknowledge specific implementation biases: regex detectors likely undercount semantic leaks, and the use of single carrier documents per niche may amplify specific formatting quirks. Furthermore, the architectural placement of the System Directive at the beginning of the context window likely introduces a 'positional bias,' where models prioritize the most immediate instructions (the attack payload) over earlier safety constraints.

Figure 10: Contribution of Detection Methods (Regex vs. Judge)



6. Conclusion

As Large Language Models shift from chat interfaces to autonomous financial agents, the security boundary moves from the user prompt to retrieved context. Our benchmark of 1,250 attack instances across ten high-stakes financial niches reveals that current safety

alignments are insufficient for this domain. We observed a clear trade-off between capability and security: **GPT-4o-mini**, optimized for helpfulness, was the most vulnerable (60.4% ASR), often prioritizing embedded instructions over safety. Conversely, **Claude-3-Haiku** showed superior robustness (26.5% ASR) through stricter refusals, while **Gemini-2.5-Flash** offered the most consistent refusal quality when threats were detected.

Critically, traditional security metrics proved inadequate. **Role Impersonation** attacks effectively exploited 'authority bias' (48.9% success), and routine task framings significantly lowered defensive guardrails compared to high-alert scenarios. Furthermore, our semantic judge uncovered 209 data leaks missed by standard regex filters. These findings demonstrate that general-purpose safety training cannot replace domain-specific defense. Financial institutions must prioritize **context-aware filtering** and **dedicated guardrails** capable of distinguishing authoritative commands from untrusted data, as relying solely on model-native refusals remains a high-risk strategy.
Specifically, future defenses should prioritize **Information Flow Control (IFC)** mechanisms and **structural context separation** (e.g., 'Spotlighting') to architecturally isolate trusted system prompts from untrusted financial carrier documents.

7. References

[1] Zhan, Q., Liang, Z., Ying, Z., & Kang, D. (2024). Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. *arXiv preprint arXiv:2403.02691*.

[2] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023, November). 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* (pp. 79-90).

[3] Yi, J., Xie, Y., Zhu, B., Kiciman, E., Sun, G., Xie, X., & Wu, F. (2025, July). Benchmarking and defending against indirect prompt injection attacks on large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1* (pp. 1809-1820).

[4] Zhang, H., Huang, J., Mei, K., Yao, Y., Wang, Z., Zhan, C., ... & Zhang, Y. (2024). Agent security bench (asb): Formalizing and benchmarking attacks and defenses in llm-based agents. *arXiv preprint arXiv:2410.02644*.

[5] Liu, Y., Jia, Y., Geng, R., Jia, J., & Gong, N. Z. (2024). Formalizing and benchmarking prompt injection attacks and defenses. In *33rd USENIX Security Symposium (USENIX Security 24)* (pp. 1831-1847).

[6] Liu, Y., Deng, G., Li, Y., Wang, K., Wang, Z., Wang, X., ... & Liu, Y. (2023). Prompt injection attack against llm-integrated applications. *arXiv preprint arXiv:2306.05499*.