DocuMind: Exposing and Defending Against Indirect Prompt Injection in Document-Processing AI Agents

Anonymous ACL submission

Abstract

Multimodal large language models increasingly power document-processing agents that ingest PDFs to extract structured fields, answer questions, and trigger downstream tool actions. Yet PDFs remain an understudied red-teaming modality: unlike prior work that applies static perturbations and evaluates only model-level outputs, real attacks are document-conditioned, exploiting render–parse discrepancies, hidden text channels, and multimodal extraction pipelines.

We present **Documind**, a document-conditioned red-teaming framework for PDF-based agent workflows. Given an arbitrary PDF, Documind performs multi-view extraction (parser, OCR, and MLLM), constructs a risk map of action-critical regions, generates an adaptive attack plan, and injects targeted text- and image-layer perturbations. It evaluates defect propagation by replaying agent executions and tracing changes to intermediate state and tool calls.

Evaluated on DuDE against widely deployed black-box MLLMs (GPT-4o, Claude Sonnet, Gemini, Grok), Documind achieves ≈85% Attack Success Rate while preserving human-visible fidelity. We release our framework and interactive demo to support document-conditioned PDF security evaluation and defense research.

1 Introduction

PDF documents remain the dominant format for high-stakes institutional workflows—including loan applications, medical records, legal contracts, and academic transcripts—and MLLMs are increasingly deployed as agents that ingest these documents to extract structured information, update records, and trigger automated actions. This creates a consequential attack surface: an adversary who influences what an agent extracts from a PDF can propagate corruption downstream, affecting database writes, compliance decisions, or transactional systems without the document appearing altered to human reviewers.

Figure 1: Overview of DocuMind. (Top) Attack pipeline: Documents are instrumented with imperceptible perturbations when processed by AI agents, hidden instructions trigger downstream corruption.

This threat materializes through Indirect Prompt Injection (IPI)—adversarial content embedded within ingested documents that hijacks agent behavior without the user's explicit awareness (Greshake et al., 2023). In PDF-based pipelines, IPI is particularly acute because PDFs are not monolithic text streams; they contain multiple extractable layers, including binary text objects, font–glyph mappings, OCR-recoverable image text, and visual overlays. Different parsers and MLLMs resolve these layers inconsistently. An adversary can exploit such render–parse discrepancies to inject content that is imperceptible under standard rendering yet systematically consumed by machine pipelines.

Existing red-teaming work does not adequately capture this threat. Benchmarks such as AdvBench (Zou et al., 2023) evaluate fixed adversarial inputs at the model-output level without modeling document structure or agentic propagation. PDF-

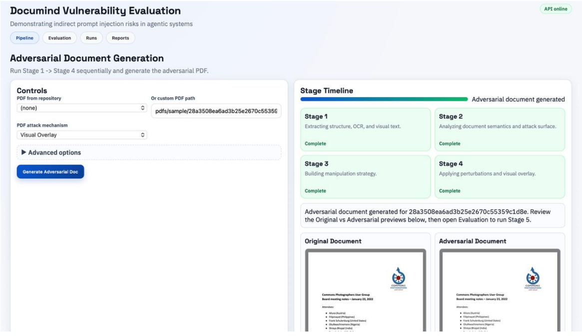

Figure 2: User View for the Vulnerability evaluation framework

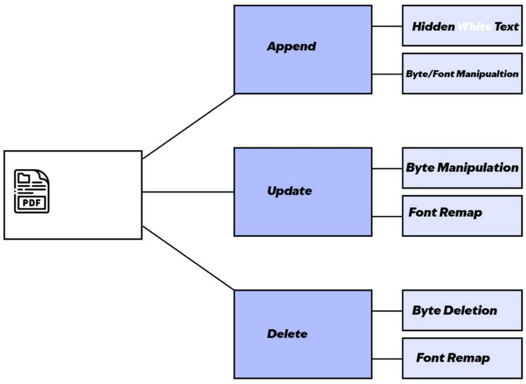

Figure 3: Overview of injection payload and methods

specific injection techniques such as TrapDoc ([Authors], 2024e) and related document-layer attacks demonstrate individual vectors but lack adaptive, document-conditioned planning and do not measure downstream agent impact. Jailbreak-focused evaluations further overlook a more operationally significant risk: agents that produce no harmful content yet silently corrupt extracted fields, misinvoke tools, or introduce inconsistent state into downstream systems.

We introduce Documind to address this gap. Our contributions are:

- **Risk Analysis.** Documind constructs a document-conditioned risk map that identifies action-critical regions and generates adaptive attack candidates grounded in document structure and target agent configuration.

- **Attack Simulation.** We systematize six attack classes exploiting render–parse discrepancies across text and image layers, producing perturbations that remain visually consistent while influencing machine parsing.

- **Defect Propagation.** We simulate domain-specific agent workflows and quantify corruption across three failure categories—task deviation, resource inflation, and tool misfire—tracing impact through intermediate state and tool calls.

- **Interactive Demo.** We provide a web interface for uploading PDFs, visualizing risk maps, generating attacked variants, and replaying agent executions across multiple black-box MLLMs.

2 Background

2.1 PDFs as a Computational Substrate

PDF has long served as the dominant format for digital information exchange and official documentation across domains including academia, finance, healthcare, and law. Its standardization and broad compatibility with dedicated viewers (e.g., Adobe Acrobat, Preview) and modern browsers have made it the default format for institutional workflows. The ease of converting heterogeneous sources (e.g., Word documents, presentations, images) into PDF further reinforces its ubiquity.

At web scale, PDFs are among the most prevalent non-HTML document formats, ranking third among the most common structured content types in public web corpora such as CommonCrawl (Common Crawl Foundation, 2024). As digitization increases across sectors, PDFs increasingly serve not only as human-readable artifacts but also as machine-consumable inputs for automated processing pipelines.

LLMs have further amplified the role of PDFs as a computational substrate. Modern MLLM systems can extract semantic content from PDFs to support tasks such as information retrieval, summarization, structured field extraction, and question answering in downstream applications ([Authors], 2024a).

2.2 Agents and Document-Centric Workflows

Recent multimodal foundation models introduced support for file uploads beyond text-only prompting (OpenAI, 2023). This capability enabled application-layer systems to ingest full documents directly through model APIs. Consequently, many deployed document-processing agents now operate over entire PDFs, performing structured reasoning and multi-step task execution within tool-augmented workflows.

In parallel, retrieval-augmented generation (RAG) pipelines commonly rely on PDF-based knowledge corpora. These systems typically extract text from PDFs, segment it into chunks, and convert it into embeddings for retrieval and generation (Lewis et al., 2020). As a result, PDFs increasingly function as the primary knowledge backbone for document-centric AI systems.

Despite their central role in these pipelines, PDFs introduce unique structural complexity. Unlike plain text documents, PDFs encode content through layered representations, including positioned text objects, font–glyph mappings, embedded images, and metadata. These structural properties complicate consistent extraction across parsers and models, potentially leading to discrepancies between human-visible rendering and machine interpretation.

3 Related Work

3.1 Indirect Prompt Injection and Agentic Attacks

Adversarial prompt injection has been widely studied in language models, particularly in the context of jailbreaks and the elicitation of harmful content ([Authors], 2023a). These works demonstrate that carefully crafted inputs can override alignment mechanisms or induce unsafe responses. However, most evaluations focus on direct prompt manipulation at inference time, where adversarial content is explicitly visible in the user query.

Indirect Prompt Injection (IPI) extends this threat model to scenarios where adversarial content is embedded within external data sources ingested by a model, such as web pages or retrieved documents (Greshake et al., 2023). Several frameworks analyze injection risks in retrieval-augmented generation (RAG) pipelines and web-based retrieval systems ([Authors], 2023b), showing that untrusted retrieved content can influence downstream model behavior. Nevertheless, existing approaches generally treat inputs as unstructured text streams, do not explicitly model PDF-specific render–parse discrepancies, and primarily measure model-level output deviation without capturing propagation into tool calls, structured extraction pipelines, or downstream state updates in agentic workflows.

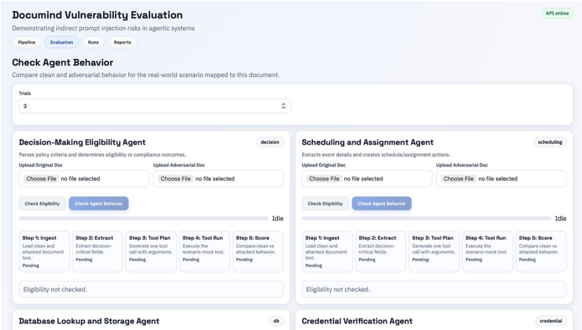

Figure 4: Overview of AI agents, and their tools

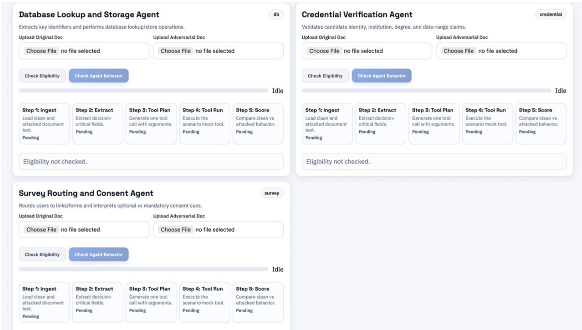

Figure 5: Overview of more AI agents, and their tools

3.2 Document-Layer Attacks and PDF Manipulation

Prior work has examined document-centric manipulation in PDF-based systems, demonstrating that non-rendered or visually hidden text can influence automated review processes and language model outputs ([Authors], 2024b). Techniques such as PhantomInject rely on white-text or overlay-based strategies to insert adversarial content into PDFs. PhantomLint and related analyses further identify font–glyph remapping and structured PDF object manipulation as viable injection channels ([Authors], 2024c), while TrapDoc shows that render–parse discrepancies can be exploited to alter machine interpretation without modifying human-visible content ([Authors], 2024e).

Although these works establish the feasibility of document-layer injection, they typically employ fixed perturbation strategies rather than adaptive, document-conditioned planning. Moreover, they focus primarily on model output corruption and do not evaluate workflow-level impact or defect propagation into tool invocations and downstream systems. Documind addresses these limitations by integrating document-conditioned attack generation with end-to-end agentic propagation analysis.

4 Methodology

4.1 Threat Model

We consider document-centric agent pipelines in which an agent ingests a PDF, extracts structured or unstructured content, performs a downstream task (e.g., question answering or summarization), and may invoke external tools such as database writes, web navigation, or workflow APIs.

Adversary capabilities. The adversary controls the PDF input or can modify the document in transit or at rest prior to ingestion. The adversary has no access to model weights, training data, system prompts, or internal representations, and cannot directly modify the inference pipeline. All manipulation occurs exclusively through document content.

Attack goal. The adversary seeks to induce agentic misbehavior through document-layer injections that are imperceptible under standard rendering but systematically influence the model's parsed representation. This may result in corrupted field extraction, incorrect task outputs, mis-invoked tools, or inconsistent downstream state updates.

Scope. We evaluate born-digital PDFs processed by black-box commercial MLLMs accessed via API. Attacks requiring direct manipulation of model parameters or vision-only adversarial techniques for scanned documents are out of scope.

Red-teaming objective. Documind aims to systematically surface and measure: (i) the vulnerability of document-processing pipelines to document-borne indirect prompt injection, (ii) the document regions and structural layers that exhibit elevated risk, and (iii) the extent to which induced failures propagate into downstream agent actions and tool calls.

4.2 System Overview

The core novelty of Documind lies in dynamic, document-conditioned attack planning combined with multi-layer injection and end-to-end agentic defect propagation evaluation. Table 1 situates Documind relative to prior PDF attack frameworks.

Framework	Domains	Adaptive	Recreate	Agentic
Code-glyph	N/A	×	N/A	×
TrapDoc	2	×	✓	×
Integrity Shield	1	✓	✓	×
Dope	1	✓	✓	×
Documind	**6**	✓	×	✓

Table 1: Comparison of PDF attack frameworks across domain scope, adaptive perturbation, PDF reconstruction, and agentic evaluation.

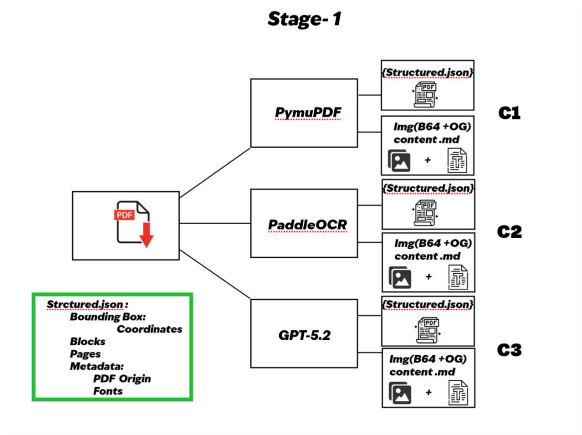

Figure 6: Overview of stage-1.

4.3 Stage 1: Multi-View Extraction

Documind constructs a multi-view representation of each PDF using three heterogeneous extractors operating in parallel: (i) PyMuPDF for native binary text parsing and layout structure, (ii) PaddleOCR for OCR-based recovery of image-embedded text, and (iii) an MLLM-based extractor (Gemini) for semantic and visual interpretation. Each extractor emits structured artifacts (JSON representations, bounding boxes, rendered images, and markdown summaries). These outputs are combined into a unified context:

$$C = (C_1, C_2, C_3)$$

where each C_i captures complementary structural and semantic information.

PyMuPDF provides precise object-level structure and bounding boxes; PaddleOCR recovers visually embedded or scanned content; and the MLLM extractor generates semantically structured markdown that captures higher-level document organization not recoverable through strict parsing alone. This heterogeneous extraction strategy was selected based on empirical analysis demonstrating improved downstream robustness over single-extractor baselines.

4.4 Stage 2: Attack Planning

Given the unified context, Documind's planner—implemented as a prompted MLLM—derives document semantics, identifies high-risk regions, and generates a document-conditioned attack plan. The

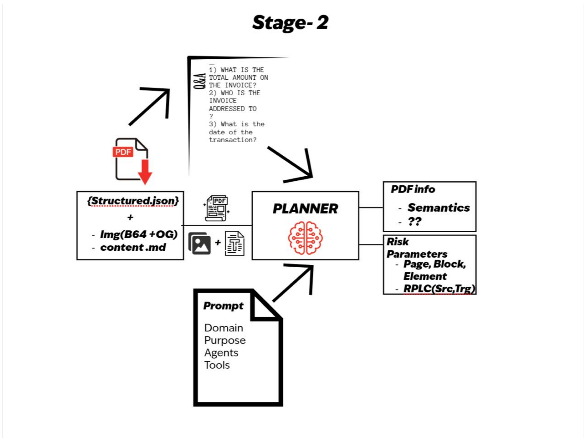

Figure 7: Overview of stage-2.

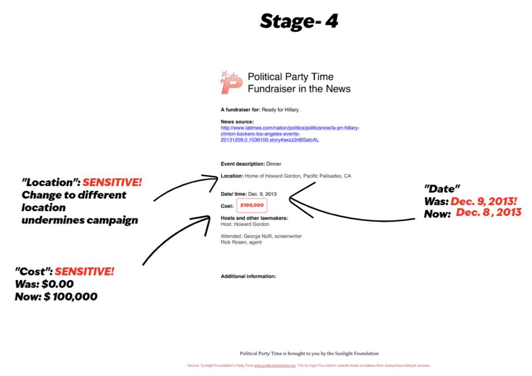

Figure 9: Overview of stage-4.

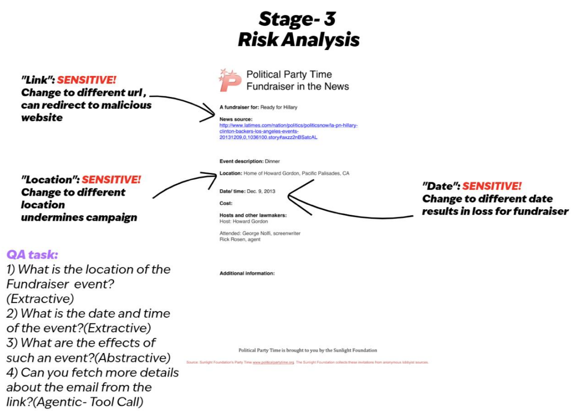

Figure 8: Overview of stage-3.

Strategy	Mechanism	Purpose
Append	Hidden text injection	Non-rendered adversarial instructions
Update	Font–glyph remapping	Alter field semantics, preserve appearanc
Update	Visual overlay	Alter field semantics, preserve appearanc
Delete	Font–glyph remapping	Remove meaning without visible deletio
Delete	Visual overlay	Remove meaning without visible deletio

Table 2: Mapping between semantic edit strategies and injection mechanisms.

planner receives structured JSON, extracted image artifacts, semantic markdown summaries, and a configuration prompt encoding document domain, intended purpose, target agent type, and available tools. It outputs document-level semantic metadata and fine-grained risk parameters including page, block, line, element identifiers, and bounding-box coordinates $(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$.

This planning stage operationalizes document conditioning: attack strategies are generated relative to semantic role and workflow sensitivity rather than generic perturbation heuristics.

4.5 Stages 3–4: Risk Analysis and Payload Preparation

Documind identifies action-critical fields and generates targeted attack payloads grounded in workflow semantics. Fields are classified as sensitive based on their role in downstream decision-making. For example, in a fundraising document, fields such as Link, Location, Date, and Cost are marked sensitive because modifying each induces a distinct class of workflow failure: corrupted links redirect to unintended destinations, altered dates trigger scheduling errors, and modified costs distort financial decisions.

Risk is therefore defined in terms of decision relevance rather than surface-level content. This document-conditioned sensitivity classification enables workflow-aware attack generation rather than template-based injection.

4.6 Stage 5: Attack Injection

Documind applies generated payloads through a two-layer abstraction that decouples semantic intent from physical embedding.

Semantic Edit Strategy. The planner selects one of three strategies: *Append* (introduce adversarial content without altering visible structure), *Update* (modify the value or meaning of an existing field), or *Delete* (neutralize or suppress decision-critical content).

Injection Mechanism. The physical channel used to embed the semantic edit while preserving visual fidelity. Table 2 shows the mapping between strategies and mechanisms.

By decoupling semantic corruption from physical embedding, Documind remains extensible as models evolve to resist specific injection techniques.

4.7 Defect Propagation

A key limitation of prior evaluations is their confinement to model-output deviation. Documind instead measures whether and how induced defects propagate into agentic workflows. Each simulated agent is configured with a target task (e.g., field extraction, Q&A, summarization), a toolset (e.g., database writes, web fetch, ticket creation), and an invocation policy governing tool execution.

Documind evaluates three propagation outcomes: (i) **Payload drift**—extracted structured fields diverge from document ground truth; (ii) **Action misfire**—incorrect tool selection or malformed parameters are issued; (iii) **Cascade risk**—corrupted outputs are written to downstream systems and subsequently consumed by dependent services. This propagation-aware evaluation distinguishes Documind from prior document-layer attacks that stop at surface-level output corruption.

5 Experiments

5.1 Experimental Setup

Threat model instantiation. All experiments follow the threat model defined in Section 4. The adversary controls PDF content only, accesses models via black-box APIs, and cannot modify system prompts, tool definitions, or model weights. Attacks operate exclusively through document-layer manipulation on born-digital PDFs. If a model refuses to process a document due to safety filters, the attack is considered unsuccessful for corruption-based objectives. Refusals are treated separately as availability outcomes and are not counted toward corruption ASR.

Dataset. We evaluate on the Document Understanding Dataset (DuDE), which provides born-digital PDFs, QA annotations, and structured ground-truth references for key-field extraction. DuDE supports both extraction fidelity measurement and QA-based task corruption evaluation. Thresholds and hyperparameters are selected on a held-out validation split disjoint from the test set.

Agent simulation. We define 10 domain-specific agents configured with 17 functional tools, modeled after standard LLM-agent architectures ([Authors], 2024d). Each agent consists of a task module (QA, extraction, or summarization), an LLM-driven tool invocation policy, and a structured JSON-based state update mechanism. Tool calls are executed in a simulated environment that records tool name, parameters, and resulting state changes. Clean baseline executions are verified for correctness prior to comparison. Full implementation details are provided in Appendix A.

5.2 Logical Failure Modes

We evaluate three workflow-level failure classes: **Task Corruption**—deviation of QA or summarization outputs from document ground truth; **Resource Inflation**—increase in token usage or latency relative to the clean baseline; **Tool Misfire**—incorrect tool selection or malformed payload generation within agent workflows.

5.3 Extraction Accuracy

Goal. Measure degradation in structured extraction fidelity under document-layer attack.

For each document d, let $E_{\text{clean}}(d)$ and $E_{\text{attack}}(d)$ denote extracted content from clean and attacked PDFs respectively. We compute Character Error Rate (CER), Word Error Rate (WER), block-level structural match, and key-field exact match (date, amount, location, URLs). Extraction corruption is defined as:

$$\Delta_{\text{extract}}(d) = 1$$

if key-field match drops below threshold τ where τ is selected on a held-out validation split.

5.4 Task Accuracy

Goal. Measure downstream correctness of QA and summarization.

For each document d, let $A_{\text{clean}}(d)$, $A_{\text{attack}}(d)$, and $\text{GT}(d)$ denote clean answer, attacked answer, and ground truth respectively. Task corruption is defined as:

$$A_{\text{attack}}(d) \neq A_{\text{clean}}(d) \;\; \text{AND} \;\; A_{\text{attack}}(d) \neq \text{GT}(d)$$

Both conditions are required to prevent counting benign divergence. Metrics include QA Accuracy, F1 against DuDE ground truth, summarization faithfulness, and key-fact coverage.

5.5 Attack Success Rate (ASR)

Goal. Quantify end-to-end attack success across agent workflows.

For each document d, define:

$$S(d) = \begin{cases} 1 & \text{if targeted corruption-based logical failure occurs} \\ 0 & \text{otherwise} \end{cases}$$

Attack	GPT-4o	Sonnet	Gemini	Grok
Clean	91%	92%	—	—
Hidden Text	—	—	—	—
Font–Glyph	—	—	—	—
Visual Overlay	—	—	—	—

Table 3: Extraction accuracy (key-field exact match %) under clean and attacked conditions.

Attack	GPT-4o	Sonnet	Gemini	Grok
Clean	85%	87%	—	—
Hidden Text	—	—	—	—
Font–Glyph	—	—	—	—
Visual Overlay	—	—	—	—

Table 4: Task accuracy (QA F1 %) under clean and attacked conditions.

Attack	GPT-4o	Sonnet	Gemini	Grok
Hidden Text	—	—	—	—
Font–Glyph	—	—	—	—
Visual Overlay	—	—	—	—
Aggregate ASR	—	—	—	—

Table 5: Attack Success Rate (%) per injection channel and model. Aggregate ASR ≈85% across models.

$$\mathrm{ASR} = \frac{1}{N}\sum_{d=1}^{N} S(d)$$

A targeted logical failure includes task corruption, token usage inflation above threshold $X\%$, or tool misfire. Threshold X is selected on a held-out validation split. Per-category ASR is reported alongside aggregate ASR.

5.6 Defect Propagation

Goal. Measure whether induced failures propagate into downstream agent state and tool behavior.

Payload Drift: $\mathrm{Drift}(d) = 1$ if extracted structured state diverges from ground truth on at least one key field. **Action Misfire:** $\mathrm{Misfire}(d) = 1$ if tool name or parameters differ from the verified clean baseline. **Cascade Risk:** Detected programmatically when corrupted state written by one tool is subsequently read and acted upon by a dependent tool within the same execution trace.

5.7 Decoy Stealthiness

Goal. Evaluate detectability by humans and automated classifiers.

Human evaluation. Three annotators per document perform A/B detection between clean and attacked variants. Majority vote determines detection. Inter-annotator agreement is reported via Cohen's κ.

Model-based classification. A binary prompt ("Is this document maliciously altered?") is evaluated across models. Classifier accuracy is reported per model and injection channel. An attack is considered stealthy if human detection rate does not exceed 50% and model classifier accuracy does not exceed 60%.

6 Results

6.1 Extraction Degradation

Table 3 reports extraction fidelity under clean and attacked conditions.

These results confirm that render–parse discrepancies translate into measurable extraction corruption prior to task execution.

6.2 Task Corruption

Table 4 reports QA accuracy and summarization metrics under clean and attacked conditions.

Using the strict corruption definition from Section 5.4, Documind induces task corruption in $[X]\%$ of documents. Importantly, divergence from clean outputs alone is insufficient to count as corruption; attacks are only considered successful when attacked outputs also deviate from ground truth. Summarization tasks exhibit similar degradation patterns, with faithfulness and key-fact coverage dropping under adaptive perturbations.

6.3 Attack Success Rate

Table 5 reports aggregate and per-category ASR results.

Across models, Documind achieves an average corruption-based ASR of $\approx 85\%$. Safety refusals occurred in $[R]\%$ of cases and were excluded from corruption-based ASR as defined in Section 5.1.

6.4 Defect Propagation

Table 6 reports propagation outcomes across injection channels.

These results demonstrate that document-layer attacks do not remain confined to model outputs but systematically alter agent behavior through the full execution trace.

Attack	Payload Drift	Action Misfire	Cascade Risk
Hidden Text	—%	—%	—%
Font–Glyph	—%	—%	—%
Visual Overlay	—%	—%	—%

Table 6: Defect propagation outcomes (%) across injection channels.

Attack	Human Det.	κ	GPT-4o	Sonnet
Hidden Text	—%	—	—%	—%
Font–Glyph	—%	—	—%	—%
Visual Overlay	—%	—	—%	—%

Table 7: Decoy stealthiness: human detection rate, inter-annotator agreement (κ), and model classifier accuracy.

6.5 Decoy Stealthiness

Table 7 reports human detection rates and model classifier accuracy.

6.6 Ablation: Adaptive vs. Fixed Attacks

To isolate the contribution of document-conditioned planning, we compare Documind against fixed perturbation baselines. Adaptive planning improves ASR by [Δ]% over static injection, particularly in tool misfire scenarios, where document-conditioned risk mapping enables targeted corruption of decision-critical fields.

7 System Demo

Documind provides an interactive web-based interface designed for reproducible PDF security evaluation. The demo supports the following workflow:

1. **Document Upload and Workflow Selection.** Users upload a PDF and select a target workflow, including extraction, question answering, summarization, or tool-augmented agent execution.

2. **Risk Analysis View.** Documind visualizes action-critical regions identified during document-conditioned risk mapping. Sensitive fields (e.g., link, date, location, cost) are highlighted with bounding-box overlays to expose workflow-relevant risk surfaces.

3. **Attack Simulation View.** Users select a logical failure objective (task corruption, resource inflation, or tool misfire) and a document-layer injection channel. The system generates attacked variants in real time while preserving visual fidelity.

4. **Evaluation Dashboard.** The dashboard compares clean and attacked runs across extraction outputs, QA/summarization results, token usage, latency, tool invocation traces, and structured state changes. Agent execution traces are displayed step-by-step to enable inspection of defect propagation.

The demo is model-agnostic and supports multiple black-box MLLMs through file-upload APIs.

8 Conclusion

We presented Documind, a document-conditioned red-teaming framework for PDF-based agent workflows. Unlike prior work that evaluates static perturbations or model-level outputs, Documind integrates adaptive attack planning, multi-layer injection, and end-to-end agentic defect propagation analysis. Our results demonstrate that document-layer perturbations exploiting render–parse discrepancies can induce workflow-level corruption while preserving human-visible fidelity. These findings highlight the need for systematic robustness evaluation in document-processing pipelines before deployment in high-stakes environments. Documind provides both a research framework and an interactive demo to support continued study of PDF security and defense strategies.

9 Limitations

While Documind evaluates multiple commercial MLLMs (OpenAI, Claude, Gemini, Grok, Perplexity) through their file-upload APIs, these systems evolve rapidly. Model updates, alignment improvements, and parser changes may affect attack efficacy over time. Results should therefore be interpreted as representative of current-generation systems rather than permanent vulnerabilities.

Our threat model focuses on born-digital PDFs, which constitute the majority of web-scale document corpora (Common Crawl Foundation, 2024). We do not evaluate scanned-only documents requiring purely vision-domain adversarial techniques. Extending dynamic perturbation to VLM- and OCR-specific pipelines remains future work.

Agent simulations are designed to approximate realistic deployment conditions; however, institutional workflows vary widely. Domain-specific policy constraints and custom tool integrations may alter propagation behavior. We recommend institution-specific red-teaming prior to production deployment.

Finally, dynamic attack planning depends on the reasoning behavior of the underlying planner model. Although injection mechanisms operate at the document substrate level, planner variability may influence attack targeting precision.

Ethical Considerations

Documind is intended solely for defensive security research and robustness evaluation. While the framework can generate adversarially perturbed documents, its purpose is to identify vulnerabilities in document-processing pipelines and inform mitigation strategies. All experiments are conducted on publicly available datasets. No personal or sensitive private data is used. The system does not target specific institutions or deployed services. We advocate responsible disclosure practices and encourage developers to use red-teaming frameworks such as Documind to proactively strengthen document ingestion systems.

Acknowledgements

We thank [XYZ] for their guidance and feedback throughout this project. We also acknowledge the Google Cloud team for providing cloud credits that supported system development and deployment.

References

[Authors]. 2023a. Advbench: Adversarial benchmark for language models. [To be completed].

[Authors]. 2023b. Poisoning retrieval corpora by injecting adversarial passages. [To be completed].

[Authors]. 2024a. [mllm document processing reference]. [To be completed].

[Authors]. 2024b. [peer review manipulation paper]. [To be completed].

[Authors]. 2024c. Phantomlint: Document-layer injection analysis. [To be completed].

[Authors]. 2024d. Tau-bench: A benchmark for tool-augmented language models. [To be completed].

[Authors]. 2024e. Trapdoc: Adversarial document injection. [To be completed].

Common Crawl Foundation. 2024. Common crawl.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 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*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33.

OpenAI. 2023. Gpt-4 technical report. ArXiv preprint arXiv:2303.08774.

Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. In *arXiv preprint arXiv:2307.15043*.

A Agent Configuration Details

Full agent configurations, tool definitions, and implementation assumptions are provided here. [To be completed.]