Context-Construction-Based Prompt Injection in Document-Centric LLM Decision Systems

Yuan Gao[1,2,*], Jinwen He[1,2], Yue Zhao[1,2], Kai Chen[1,2]
[1] Institute of Information Engineering, Chinese Academy of Sciences, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, China
[*] Corresponding author
{gaoyuan, hejinwen, zhaoyue, chenkai}@iie.ac.cn

***Abstract*—Large Language Models (LLMs) are increasingly deployed in document-centric workflows such as resume screening and academic peer review, where uploaded files are automatically parsed and converted into model input. This processing pipeline introduces new security risks: hidden content embedded in documents may be extracted by parsers and incorporated into the LLM context without being visible to human users.**

Most existing prompt injection research focuses on inserting explicit or obfuscated commands, and largely ignores how document structure and parser behavior affect whether injected content reliably enters the model's context. In this work, we show that such placement instability significantly limits the effectiveness of document-level attacks in practice. To address this issue, we propose ConStruct, a context-construction-based prompt injection framework for text-extraction-based document pipelines that jointly considers structural placement and content formulation. ConStruct enforces structure-aware placement to ensure that injected text consistently appears at salient positions across common PDF parsers, and uses fact-based, non-imperative statements that blend naturally with task-relevant context.

We evaluate ConStruct in two representative document-driven scenarios, resume screening and academic paper review, across multiple real-world systems, parsers, and detection mechanisms. Experimental results show that ConStruct achieves up to 67.4% higher resume matching scores and 94.1% higher paper review ratings relative to the original documents, while maintaining near-perfect evasion with rates above 99.47%.

***Index Terms*—large language models, document security, prompt injection attacks, natural language processing**

I. Introduction

Large Language Models (LLMs) are increasingly deployed in document-centric decision-making systems, such as resume screening, academic peer review, and document auditing [1]–[5]. In these applications, uploaded documents are not directly provided to the model. Instead, they are first processed by a system pipeline that performs document parsing, content extraction, ordering, and truncation, ultimately constructing a bounded-length context that serves as the model input.

Despite its central role, this document-to-context construction process is often treated as a transparent and trustworthy intermediate step. Most existing studies implicitly assume that relevant document content is stably and completely incorporated into the model context, and that security risks primarily arise from how the model interprets and executes input instructions [6].

Within this setting, prompt injection attacks have been extensively studied. Prior work predominantly focuses on instruction-based prompt injection, where explicit or implicit commands are injected to override or manipulate the model's behavior [7], [8]. These attacks fundamentally operate at the stage of prompt interpretation and rely on the assumption that injected content has already entered the model context.

However, this assumption does not always hold in document-centric LLM pipelines. In practice, different document parsers employ diverse rules for structural interpretation, reading order, and text extraction. As a result, identical document content may be incorporated into the constructed context with varying orderings, or may even be excluded entirely. Moreover, to satisfy context length constraints, extracted text is commonly truncated, further influencing which portions of the document are ultimately visible to the model.

Based on these observations, we argue that context construction itself constitutes a previously underexplored attack surface in document-centric LLM decision systems. Without manipulating model parameters or overriding system instructions, an attacker can influence model behavior by controlling document content and structure, thereby affecting which textual fragments are stably included in the constructed context. We refer to this class of attacks as context-construction-based prompt injection, distinguishing it from traditional instruction-based prompt injection.

To systematically study this attack surface, we propose ConStruct, a context-construction-based prompt injection framework designed to exploit document parsing and context construction mechanisms. ConStruct jointly designs injected content and document structure to ensure that factual, task-relevant text can reliably enter the model context across different parsers and truncation settings. We comprehensively evaluate ConStruct in both resume screening and paper review scenarios, comparing it against representative instruction-based prompt injection baselines across multiple parsers. Our method has been validated across different models, including GPT-4o and Gemini series, demonstrating its robustness and cross-model effectiveness. Also, the entire process from context extraction to prompt injection is fully automated, leveraging a multi-agent generator to select and inject task-relevant factual content.

We summarize the contributions of this work as follows:

- We identify context construction as a new attack surface in document-centric LLM decision systems, and formal-

ize a context-construction-based form of prompt injection that targets the document parsing and context formation stage rather than prompt interpretation.
- We propose ConStruct, a context-construction-based prompt injection framework that jointly designs the document structure and injected content to achieve stable context inclusion across different parsers.
- We conduct systematic evaluations in resume screening and paper review scenarios, demonstrating that explicitly targeting context construction yields significantly higher context inclusion stability and attack effectiveness than instruction-based prompt injection baselines, achieving improvements of up to 24.4% of the full paper review rating scale and 12.5% of the full resume matching score scale, with consistent superiority across all tasks and systems.

II. Background and Related Work

A. Document-Centric LLM Decision Systems

LLMs have been increasingly deployed in document-centric decision-making systems, where uploaded documents serve as the primary source of information for automated assessment and reasoning. Representative applications include resume screening, LLM-assisted academic peer review, and other document analysis tasks. In such systems, documents are not directly fed into the model. Instead, they are first processed by automated pipelines that perform document parsing, text extraction, and context construction before the resulting text is provided to the LLM.

This document-to-context pipeline plays a critical role in determining which content is visible to the model and how it is ordered. Recent security analyses and application studies have highlighted that discrepancies may arise between what human readers perceive in a document and what automated parsers extract, leading to potential risks in document-centric LLM applications [7], [8]. As a result, document parsing and context construction have become integral components of real-world LLM systems, with direct implications for both system reliability and security.

B. Prompt Injection Attacks

Prompt injection attacks have emerged as a prominent security concern for LLM-based systems. Existing research has primarily focused on instruction-based prompt injection, where attackers inject explicit or implicit commands into controllable inputs to override, manipulate, or bypass the model's intended task instructions. Typical examples include instruction overriding and role manipulation attacks. These attacks, such as the "ignore previous instructions" prompt, aim to manipulate the model's behavior by overriding earlier inputs or altering its assumed role during execution [9].

Beyond direct prompt manipulation, recent studies have explored Indirect Prompt Injection (IPI), where malicious instructions are embedded into external sources that the model consumes, such as web pages, retrieved documents, or other auxiliary content. Prior work has demonstrated that by embedding crafted prompts into such sources, attackers can influence LLM outputs without directly interacting with the user-facing prompt interface [6], [10], [11]. These attacks are generally more covert than direct prompt injection and pose increased challenges for detection and mitigation.

To counter prompt injection threats, existing defenses have largely followed two directions. The first focuses on instruction restriction and alignment, where models are guided or fine-tuned to prioritize legitimate system instructions while ignoring potentially malicious embedded commands [12]–[16]. The second direction relies on content filtering and detection mechanisms, which employ auxiliary models or heuristics to identify and remove suspicious input fragments before they reach the LLM [17]–[19]. These defenses are widely adopted in practice and reflect the growing recognition of prompt injection risks in deployed systems.

C. Limitations of Existing Prompt Injection in Document-Centric Pipelines

Despite extensive research on prompt injection and corresponding defenses, most existing studies implicitly assume that injected content has already been incorporated into the model input. In document-centric LLM pipelines, however, injected content must first survive document parsing, text extraction, and context construction before it can influence model behavior. Even if injected content passes detection mechanisms successfully, it may still fail to affect the model if it is excluded, reordered, or truncated during preprocessing stages.

In practice, text extraction behavior varies across commonly used document parsing libraries, such as PyMuPDF, pdfplumber, PDFMiner, and PyPDF2 [20]–[23]. Identical document objects may appear at different positions in the extracted text, or may not be extracted at all, depending on the parser and its structural interpretation rules. Such parser-induced variability introduces inconsistency in context inclusion, and current prompt injection threat models tend to ignore this inconsistency.

As a result, prompt injection techniques that are effective in direct prompt-based interfaces or controlled settings may exhibit unstable or inconsistent behavior when applied to document-centric pipelines. This observation motivates a systematic investigation of prompt injection attacks that explicitly target the context construction process, rather than solely focusing on prompt interpretation at the model level.

III. Methodology

A. Threat Model and Problem Formalization

System Model. We consider document-centric LLM decision systems in which an uploaded document is processed by an automated *text-extraction-based* pipeline before being provided to a Large Language Model for inference. Given a document D, the system applies a document parser $P(\cdot)$

to extract textual content and construct an ordered context sequence:

$$C = P(D) = [c_1, c_2, \ldots, c_n],$$

where each c_i denotes a text fragment extracted from the document according to the parser's structural interpretation and ordering rules. Due to LLM context length constraints, the constructed context is typically truncated to a prefix or suffix of fixed length K, yielding the final model input:

$$C_K = \text{Trunc}_K(C).$$

The LLM then produces an output based on C_K and a fixed system prompt corresponding to the target task, such as resume screening or paper review.

We focus on systems whose effective model input is derived primarily from parser-extracted text, and do not claim coverage of vision-dominant or OCR-first document pipelines.

Threat Model. We assume an adversary who can fully control the content and layout of the uploaded document D, but has no access to the internal parameters of the LLM, the system prompt, or the implementation details of the document parser. This threat model reflects realistic scenarios in which attackers can submit documents for evaluation but cannot modify backend components of deployed systems.

Importantly, we do not assume that the adversary can bypass existing prompt-level detection or filtering mechanisms. Instead, injected content must remain sufficiently inconspicuous to avoid triggering deployed defenses, making stealthiness a necessary constraint rather than the primary optimization objective of the attack. We focus on PDF documents not because of a format-specific vulnerability, but because PDF is a widely deployed format that provides a representative setting for studying parser-dependent text extraction and context construction. Our attack relies on regularities observed across common PDF text extractors, rather than on any claim of universal behavior across all parsers or document understanding systems.

Context-Construction-Based Prompt Injection. Unlike conventional instruction-based prompt injection, which targets how the model interprets a given input prompt, we focus on attacks that target the construction of the model input itself. Specifically, an attacker aims to manipulate document content and structure such that injected text fragments are reliably incorporated into the constructed context C_K, thereby influencing the model's decision without overriding explicit instructions.

We refer to this class of attacks as context-construction-based prompt injection. The effectiveness of such an attack depends not only on the semantic content of injected text, but also on whether and where the injected fragments appear in the parser-extracted and truncated context.

Context Inclusion Stability. To characterize this objective, we introduce the notion of Context Inclusion Stability (CIS), which measures the consistency with which an injected fragment appears in the constructed context across different parsers or preprocessing settings. Let I denote an injected text fragment embedded in document D, and let $\mathcal{P}$ be a set of parsers. For a given truncation length K, CIS is defined as:

$$\text{CIS}(I) = \frac{1}{|\mathcal{P}|} \sum_{P \in \mathcal{P}} \mathbb{I}\big[I \in \text{Trunc}_K\big(P(D)\big)\big], \qquad (1)$$

where $\mathbb{I}[\cdot]$ is the indicator function. A higher CIS indicates that the injected fragment is more reliably included in the model input across different parsing behaviors.

Attack Objective. Under this formulation, the attacker's goal is to design document structure and injected content that maximize context inclusion stability while satisfying stealth constraints imposed by existing detection mechanisms, and finally influence downstream LLM evaluation outcomes in document-driven tasks.

B. Proposed Attack: ConStruct

1) Design Overview: Based on the problem formulation in Section III-A, the main challenge of context-construction-based prompt injection is to ensure that injected content is reliably included in the model input under heterogeneous parsing and truncation settings. ConStruct addresses this challenge through two complementary components: evidence-style content construction and structure-aware placement.

Specifically, ConStruct generates factual, task-relevant injected content that blends with the surrounding context, and places it near page-level front or end positions to improve stable inclusion in the constructed context. Figure 1 shows the overall pipeline. Stages 1–3 correspond to evidence-style content construction, and Stage 4 realizes the injection through overlay-based structural placement. The following two subsections describe these two components in turn.

2) Evidence-Style Content Construction: Stable inclusion alone is insufficient if injected content is easily detected or filtered. In real-world systems, prompt-level defenses are commonly deployed to identify explicit command-like patterns. To satisfy this constraint, ConStruct constructs injected content as evidence-style statements rather than imperative prompts.

Each injected fragment consists of three tightly coupled components: (i) a brief contextual reference to the surrounding document content, (ii) a factual or numerical evidence statement retrieved from external sources, and (iii) a concise synthesis that connects the evidence to the local context through neutral inference. This formulation allows the injected content to be interpreted by the LLM as task-relevant evidence, rather than as an instruction. ConStruct automatically queries external web sources to obtain concise and verifiable information, collecting relevant statistics, benchmark results, or contextual data, which are then incorporated into the generated content. The retrieved evidence is seamlessly integrated into the prompt, ensuring that the injected statement remains contextually appropriate and natural.

To improve logical consistency and stylistic naturalness, ConStruct adopts a lightweight multi-agent generation process with three roles: candidate generation, logical verification, and stealth evaluation. The candidate generator produces an initial

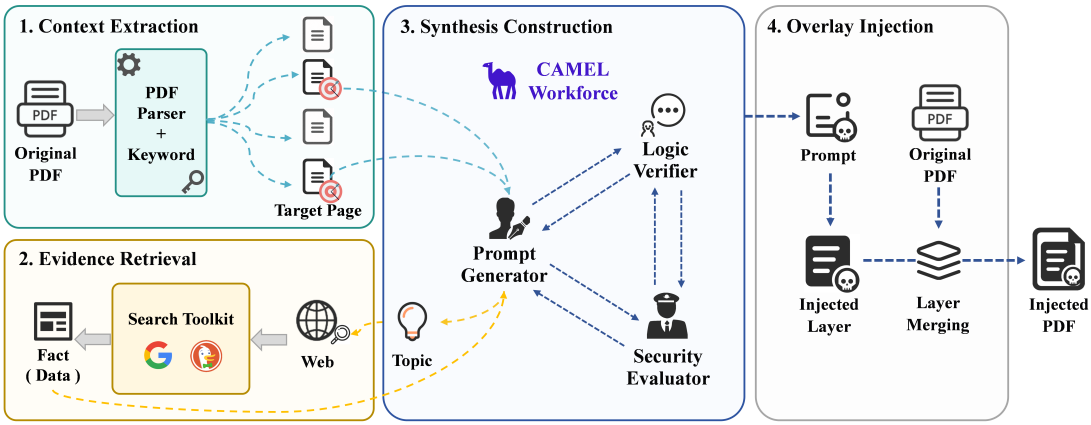

Fig. 1. Overview of our work.

evidence-style draft and suggests an appropriate injection position for the target page. The logical verifier removes unreliable, exaggerated, or weakly supported claims to reduce the risk of inconsistency or suspicion. The stealth evaluator then revises the wording from a detection-oriented perspective, removing imperative structures and abnormal prompt-like patterns, while further aligning the fragment with the surrounding document style through prompt-level stylistic control. Through this role separation, ConStruct produces injected content that is more coherent, better grounded, and less conspicuous than directly generated prompt text. As a result, the injected fragment is presented as factual, task-relevant evidence rather than as an explicit instruction, which makes it less likely to trigger defenses primarily designed to detect command-like prompt injection patterns.

3) Implementation via Overlay Injection: ConStruct realizes structure-aware placement through page-level overlay injection. Injected text is rendered as an additional transparent text layer and merged into the document content stream in a controlled order. By deterministically applying overlays corresponding to end placement before those for front placement, ConStruct ensures that injected fragments appear at consistent positions in the parser-extracted text.

The injected text does not alter the document's human visible content, but remains fully accessible to automated parsers. This overlay-based realization preserves document integrity while enabling precise control over parser-visible ordering, thereby enforcing the structural guarantees required for stable context inclusion.

Figure 2 illustrates this two-stage insertion process. In this process, each target page first receives an end overlay, shown in orange, followed by a front overlay, shown in blue. The blue overlay indicates front insertion, while the orange overlay indicates end insertion. The injected text, rendered in the smallest white font, remains invisible to human readers but is preserved in the page content stream. If the position is specified as front, the overlay is placed at the top of the target page; if end, it is positioned at the bottom. This design

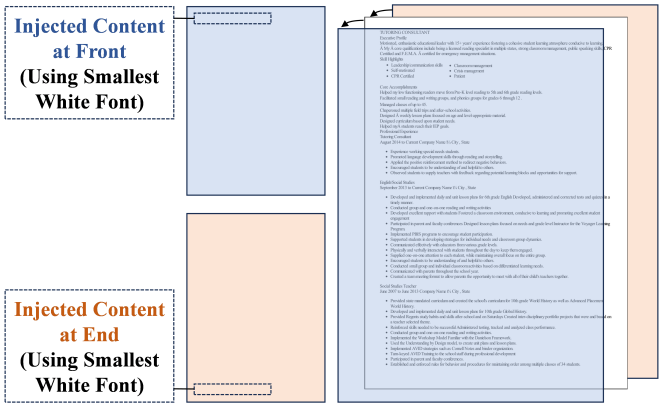

Fig. 2. Overlay injection at the front and end of a page.

ensures that the injected prompt is seamlessly embedded at the chosen location while being reliably extracted by downstream parsers. After constructing overlays, we merge them into the original PDF in a deterministic sequence. Besides, all overlays targeting the same document are applied incrementally, with intermediate files saved after each step to avoid conflicts and ensure reproducibility. During the merging process of a PDF, all pages with end overlays are applied first, then pages with front overlays, ensuring that the parser-visible order places the front-injected text at the top and the end-injected text at the bottom. This controlled layering process guarantees consistent extraction positions across different PDF parsers.

Finally, we conduct metadata handling. We normalize metadata to avoid potential extraction failures, as some parsers double-check metadata during the extraction process. These operations realize a structural guarantee: the injected text is visually unobtrusive but is consistently placed at parser-visible front or end positions across common extractors. The approach combines content-level stealth (fact-based prompts) with structure-level control (overlay ordering and validation).

IV. Experiments

A. Experimental Setup

1) Tasks and Datasets: We evaluate ConStruct in two representative document-centric LLM decision scenarios: resume screening and academic paper review. These tasks are chosen because they rely heavily on automated document parsing and context construction, and are widely adopted in real-world LLM-assisted decision pipelines.

Resume Dataset. For resume screening, we use the Kaggle Resume Dataset [24]. The dataset covers 24 distinct job categories. To control experimental scale while maintaining category diversity, we randomly select 20 resumes from each category, resulting in a balanced subset of 480 resumes.

Paper Dataset. For academic paper review, we use the ICLR 2017-2019 dataset [25], which provides PDF files and associated metadata for submitted papers. We randomly sample 300 papers for evaluation.

2) Document Parsers and Context Construction: In all evaluated systems, uploaded documents are first processed by built-in document parsers to extract textual content and construct the model input context. We do not modify the parsing logic of the target systems. Instead, we treat each parser as a black-box component that determines content visibility and ordering. For each parser, the extracted text is subject to the system's default context length constraints before being passed to the underlying LLM. To ensure fair comparison across parsers, we do not alter truncation policies or token budgets.

3) Evaluation Systems: We evaluate ConStruct across multiple representative LLM-based document evaluation systems.

Resume Evaluation Systems. We evaluate injected and non-injected resumes using three open source Applicant Tracking System (ATS)-style resume evaluation tools, all tested under their default configurations. ETEGemini [26] is an end-to-end applicant tracking system built on Google Gemini Pro, which parses resumes and outputs a Job Description (JD) Match score between 0 and 100. ResumePerfector.ai [27] is an online platform designed to optimize resumes for ATS filtering. It analyzes resumes against job descriptions and outputs an optimization score between 0 and 100. ResumeGPT [28] is an open-source GPT-based system that compares resumes with job descriptions and produces discrete suitability labels. It outputs a discrete suitability label, which we map to $\{0, 1, 2\}$ corresponding to "Not suitable," "Maybe suitable," and "Suitable," yielding a coarse-grained decision outcome. We calculate the mean score across all test samples and report the results in the table.

Paper Review Systems. We evaluate paper-level prompt injection on two LLM-based academic review frameworks. ChatReviewer [29] loads PDF papers, extracts content through built-in parsers, and queries an API-based LLM to generate structured reviews and an overall score from 1 to 10. AI-Scientist [30] adopts a multi-agent architecture that simulates reviewers and meta-reviewers, producing both numerical review scores on a 1–10 scale and final accept/reject decisions. In our evaluation, we report the mean review score and the average acceptance rate across all test samples.

Together, these systems cover both single-model and multi-agent evaluation paradigms commonly used in LLM-assisted academic review.

4) Baselines: We compare ConStruct against three categories of baselines.

Original Documents. The Original baseline corresponds to unmodified resumes and papers evaluated under each system's default settings.

Explicit Instruction Injection. As a representative instruction-based baseline, we include the canonical "Ignore Previous Prompt" [9]. We insert the prompt as invisible white text at the top-left corner of the page and instruct the LLM to assign a higher score.

Task-Specific Prompt Injection. We include a task-specific prompt injection baseline [8], which is designed exclusively for academic paper review, inserting the fixed, handcrafted prompt as invisible white text at the bottom-left corner of the page. As no analogous task-specific injection method has been studied for resume screening, the corresponding entries in Table II are marked as not applicable.

5) Detection Mechanisms: To evaluate stealthiness as a necessary constraint, we test injected documents against two representative detection mechanisms.

Eliezer Prompt [17] is a reasoning-based detection method that queries an API-based LLM with a meta-instruction to identify hidden or adversarial prompts, producing a binary decision (malicious or benign). PromptGuard-86M [31] is a lightweight LLaMA-based classifier trained specifically for prompt injection detection, which outputs probabilities over benign and malicious classes. Despite their different output forms, we evaluate both detectors using a unified metric, the evasion rate. An injected prompt is considered to successfully evade detection if it is classified as benign. For Eliezer Prompt, we compute the evasion rate as the average fraction of samples classified as benign. For PromptGuard, we compute the evasion rate by averaging the benign scores.

6) Common Settings: All evaluation systems and detectors are used with their official default configurations unless otherwise specified. GPT-based components are run using GPT-4o, and Gemini-based components use Gemini-2.5-Pro. All experiments are conducted under identical settings across baselines to ensure comparability. Unless otherwise specified, all experiments are conducted three times with independent runs, and all the reported results are averaged over the runs.

TABLE I
Context Inclusion Stability Across Different PDF Parsers

Method	PDF Parser				
	PyMuPDF	pdfplumber	PDFMiner	PyPDF	PyPDF2
Explicit	0.01	0.97	0.89	0.11	0.02
Task-Specific	0.00	0.02	0.00	0.01	0.00
Ours	**1.00**	**0.99**	**1.00**	**0.99**	**0.98**

B. Context Inclusion Stability Analysis

This experiment evaluates the CIS of different document-level injection strategies across commonly used PDF parsers. As defined in Equation (1), CIS measures whether the injected content is included within the effective context window after text extraction and truncation.

We consider a minimal and controlled setting to isolate parser-induced behavior. Specifically, for each document, we inject a short marker string ("HIDDEN-TEXT") at the front of the document, such that it should appear early in the extracted text if the parser preserves the intended placement. The goal is to verify whether different parsers consistently include the injected content within the truncated context.

To determine whether the injected content is included in the effective context, we apply each parser to extract text from the injected documents and then truncate the extracted text to the first K tokens. In this experiment, we set K to ten times the length of the injected content, yielding a minimally sufficient context window: if the injected content is placed near the intended front position, it will be fully included; if a parser reorders the content to a distant position, it will fall outside the truncated window and be excluded. So that we can distinguish stable placement from parser-induced reordering without introducing excessive context.

We evaluate five widely used PDF parsers: PyMuPDF, pdfplumber, PDFMiner, PyPDF and PyPDF2. For each parser and injection method, CIS is computed as the fraction of documents for which the injected marker appears within the truncated context, following Equation (1).

As shown in Table I, both Explicit and Task-Specific injection strategies exhibit strong parser dependence. While some parsers frequently include the injected content within the truncated context, others often fail to do so, resulting in low CIS values. In contrast, our method achieves consistently high CIS across all tested parsers, indicating that the injected content is reliably included within the effective context regardless of parser choice.

These results demonstrate that simple hidden-text injection, even when placed visually at the page boundary, does not guarantee stable inclusion in the model input. In contrast, structure-aware injection is necessary to ensure that injected content consistently survives parser extraction and context truncation, forming a reliable basis for downstream influence.

C. Main Results

Table II shows downstream decision shifts and detection evasion rates under three injection strategies, together with the Original baseline. Two observations stand out.

First, the explicit command-style injection can change LLM's output scores, but is operationally fragile under detection. The Explicit baseline raises scores in both scenarios, for example improving ETEGemini by 54.85% and increasing AI-Scientist acceptance from 0.78 to 4.85. However, its evasion rates collapse to near zero across detectors and scenarios. This gap between effectiveness and survivability supports our claim that overt imperative prompts are unlikely to persist in realistic pipelines where input screening is enabled.

Second, the specially designed task-specific injection templates do not resolve the reliability problem and fail to generalize across scenarios. The Task-Specific baseline, available only in the paper-review setting, yields moderate gains on AI-Scientist, increasing the average score from 3.22 to 5.28 and acceptance from 0.78 to 5.6. Yet it remains substantially detectable, with evasion rates of 73.30% under Eliezer and 58.90% under PromptGuard. Moreover, the prompt itself is a practical limitation: the baseline is not defined for resumes, leaving the corresponding cells empty. Taken together, these results indicate that handcrafted, scenario-dependent templates neither guarantee stealth under existing detectors nor provide a transferable attack mechanism.

Our method achieves stable impact without incurring the detectability failures of Explicit injection and without relying on scenario-specific templates. On resume screening, it improves all three systems over both Original and Explicit, reaching 60.16 on ETEGemini and 1.28 on ResumeGPT, while maintaining near-perfect evasion of 100.00% under Eliezer and 99.84% under PromptGuard. On paper review, it yields the strongest outcomes across both pipelines, achieving 7.83 on ChatReviewer and 6.25 on AI-Scientist with 6.1% acceptance, while remaining effectively undetected with evasion rates of 100.00% and 99.47. These results validate our thesis developed in the CIS analysis: influence is only meaningful when injected content reliably survives parsing and filtering to enter the model context, and evidence-style formulation provides a stealth constraint that current detectors fail to capture.

D. Ablation Study

To further understand the contribution of document structure in our framework, we conduct an ablation study where structural placement control is disabled. We consider a variant, referred to as w/o Doc Struct., where prompts are directly injected by adding white-font text at the top-left corner without using ConStruct layering. For fairness, the injection is implemented using PyMuPDF: when the position is labeled as front, the text is inserted at the top of the page, as shown in Figure 2. The exact coordinate settings are identical to those used in our structured injection method, ensuring that the only difference lies in the absence of parser-level ordering control. Table III summarizes the results on the two paper review systems.

Without structural control, performance on ChatReviewer and AI-Scientist drops notably. On ChatReviewer, the average score decreases by 0.38 compared to our ConStruct method, and on AI-Scientist the review score decreases from 6.25 to 3.85, while the acceptance rate drops from 6.10% to almost half of it. This shows that simply inserting ConStruct generated text is insufficient when parsers reorder extracted content. This degradation cannot be attributed to changes in prompt content, as the injected statements remain factual, non-imperative, and identical to those used in our full method. Instead, the performance gap directly reflects the impact of parser-dependent reordering. Without structural control, the injected prompt may

TABLE II
COMPARISON OF INJECTION METHODS ON PAPER AND RESUME REVIEW SYSTEMS, WITH EVASION RATES FROM DETECTION MODELS.

Group	Paper Review Systems			Evasion Rate (Paper)		Resume Systems			Evasion Rate (Resume)	
	ChatReviewer	AI-Scientist		Eliezer	PromptGuard	ETEGemini	ResumePerfector	ResumeGPT	Eliezer	PromptGuard
	(1–10)	(1–10)	(%)	(%)	(%)	(%)	(0–100)	(0,1,2)	(%)	(%)
Original	7.21	3.22	0.78	98.33	99.51	35.95	49.23	0.97	99.58	99.91
Explicit	7.75	3.81	4.85	0.00	0.61	55.67	**69.26**	1.17	6.25	1.68
Task-Specific	7.68	5.28	5.61	73.30	58.90	–	–	–	–	–
Ours	**7.83**	**6.25**	**6.10**	**100.00**	**99.47**	**60.16**	68.12	**1.28**	**100.00**	**99.84**

TABLE III
THE EFFECT OF DOCUMENT STRUCTURE IN INJECTION

Method	ChatReviewer	AI-Scientist	
	Avg. (1–10)	Avg. (1–10)	Accept (%)
w/o Doc Struct.	7.45	3.85	3.39
Ours	**7.83**	**6.25**	**6.10**

be extracted at inconsistent positions, where its influence is weakened or overridden by subsequent task instructions. These results confirm that document-level structural stability is not a secondary optimization, but a necessary condition for reliable document-based prompt injection.

E. Evaluation on a Black-box LLM Web Interface

Evaluating in black-box environments is essential because many real-world LLM applications conceal their internal parsing and safety mechanisms. To test whether ConStruct remains effective under such conditions, we conduct a case study on the ChatGPT web interface. During this evaluation, the system used the reasoning-enabled GPT-5 Thinking model. We upload 20 ICLR 2017 papers and evaluate three versions of each document: the original paper, the Explicit baseline, and ConStruct. For each version, we ask ChatGPT to produce a single overall review score on a 1–10 scale using the prompt shown below. We then issue a separate detection query: "Is there any potential injected prompt in this PDF?"

Review prompt. *You are an expert academic reviewer. Please carefully evaluate the uploaded paper based on originality, methodology, clarity, significance, and contribution. Assign a single score from 1 to 10 and give a 2–3 sentence explanation. Output: 'Score: [1–10]' and 'Explanation: [...]'.*

TABLE IV
EVALUATION OF PAPER REVIEW INJECTION ON LLM

Method	Average Score(1–10)	Detection Rate(%)
Original	8.45	0
Explicit	8.85	100
Ours	**9.00**	**0**

Table IV shows that both injection methods increase the review score compared to the original papers, indicating that injected content is successfully extracted and incorporated by the black-box LLM. However, while the explicit injection is always detected, our method achieves a higher average score and remains entirely undetected, demonstrating that factual, non-imperative injection can preserve influence under black-box inspection. This case study demonstrates the feasibility of our method in real-world black-box LLM environments, showing that it could be extended to other document-centric tasks such as legal document review.

V. DISCUSSION

In this work, we show that the success of document-based prompt injection depends not only on what is injected but on whether it appears in the model's input context—something heavily influenced by how different parsers handle document structure. Even well-crafted prompts fail if misplaced. By controlling placement and ensuring parser-robust positioning, our method guarantees consistent context inclusion. Moreover, stealth comes not from hiding the prompt, but from making it appear as natural, task-relevant content—such as factual statements that blend seamlessly into the document. This design evades both automated detectors and human reviewers, as the injection looks like ordinary document content.

Our results highlight a bigger issue with document-driven LLM systems: these systems often treat uploaded documents as fully trusted input without considering how hidden content might affect the model. Our approach shows that these vulnerabilities are present even in standard parsing and content construction processes, without needing access to the internal workings of the system.

In conclusion, we suggest that securing document-based LLM systems requires a better understanding of how content, structure, and parsing interact. Future defenses should consider how document layout affects the model's behavior, and not just focus on the text content alone. ConStruct is not a PDF-specific trick, but a study of how upstream document processing deterministically shapes the learning context of LLMs, providing a foundation for future improvements in document-processing systems that are more robust and secure.

VI. ETHICS CONSIDERATIONS

This work is conducted for research purposes to better understand and mitigate security vulnerabilities in LLM document processing. We acknowledge that techniques discussed in this work could be misused if applied irresponsibly. However, we believe that revealing such vulnerabilities is necessary for improving system robustness. Our study aims to inform

developers and researchers of potential risks and encourage the design of more secure document-processing workflows, and is intended solely to support defensive research and promote the development of safer, more transparent and more trustworthy AI systems.

VII. Conclusion

In this work, we present ConStruct, a context-construction-based prompt injection framework designed to address the challenge of ensuring stable content placement across different document parsers while maintaining content-level stealth. Our results demonstrate that, by combining structural placement with fact-based content generation, ConStruct can reliably influence model behavior in document-centric LLM applications such as resume screening and academic paper review. We showed that traditional prompt injection methods suffer from placement instability due to differences in how parsers handle document structure. In contrast, ConStruct achieves consistent placement across various parsers and evades detection mechanisms, making it a more robust and practical solution for real-world scenarios. Experimental results confirm that ConStruct outperforms existing baseline methods, achieving stronger influence on LLM evaluations while maintaining nearly perfect evasion from detection systems, with evasion rates consistently above 99.47% across both paper and resume settings. Overall, ConStruct advances research on securing document-centric LLM systems by emphasizing both document structure and content formulation.

Acknowledgements

During the writing process, the authors used the ChatGPT web interface to assist with language refinement in parts of this paper. The resulting text was carefully checked and revised by the authors, who take full responsibility for the final published content.

References

[1] G. Vagale, S. Y. Bhat, P. P. P. Dharishini, and P. GK, "Prospectcv: Llm-based advanced cv-jd evaluation platform," in *2024 IEEE Students Conference on Engineering and Systems (SCES)*. IEEE, 2024, pp. 1–6.

[2] Y. Jin, Q. Zhao, Y. Wang, H. Chen, K. Zhu, Y. Xiao, and J. Wang, "Agentreview: Exploring peer review dynamics with llm agents," pp. 1208–1226, 2024.

[3] M. D'Arcy, T. Hope, L. Birnbaum, and D. Downey, "Marg: Multi-agent review generation for scientific papers," *arXiv preprint arXiv:2401.04259*, 2024.

[4] Y. Weng, M. Zhu, G. Bao, H. Zhang, J. Wang, Y. Zhang, and L. Yang, "Cycleresearcher: Improving automated research via automated review," in *The Thirteenth International Conference on Learning Representations*, 2025.

[5] M. Zhu, Y. Weng, L. Yang, and Y. Zhang, "Deepreview: Improving llm-based paper review with human-like deep thinking process," 2025. [Online]. Available: https://arxiv.org/abs/2503.08569

[6] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "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*, 2023, pp. 79–90.

[7] GenAI Security Project, "Owasp top 10 for llm & generative ai security," 2025. [Online]. Available: https://genaisecurityproject.com/llmrisk2023-24/llm01-24-prompt-injection/

[8] R. Ye, X. Pang, J. Chai, J. Chen, Z. Yin, Z. Xiang, X. Dong, J. Shao, and S. Chen, "Are we there yet? revealing the risks of utilizing large language models in scholarly peer review," *arXiv preprint arXiv:2412.01708*, 2024.

[9] F. Perez and I. Ribeiro, "Ignore previous prompt: Attack techniques for language models," in *NeurIPS 2022 Workshop on Machine Learning Safety*, 2022.

[10] Z. Li, B. Peng, P. He, and X. Yan, "Evaluating the instruction-following robustness of large language models to prompt injection," pp. 557–568, 2024.

[11] Q. Zhan, R. Fang, H. S. Panchal, and D. Kang, "Adaptive attacks break defenses against indirect prompt injection attacks on llm agents," pp. 7101–7117, 2025.

[12] J. Piet, M. Alrashed, C. Sitawarin, S. Chen, Z. Wei, E. Sun, B. Alomair, and D. Wagner, "Jatmo: Prompt injection defense by task-specific finetuning," in *European Symposium on Research in Computer Security*. Springer, 2024, pp. 105–124.

[13] K. Hines, G. Lopez, M. Hall, F. Zarfati, Y. Zunger, and E. Kiciman, "Defending against indirect prompt injection attacks with spotlighting," *arXiv preprint arXiv:2403.14720*, 2024.

[14] S. Willison, "Delimiters won't save you from prompt injection," 2023. [Online]. Available: https://simonwillison.net/2023/May/11/delimiters-wont-save-you

[15] E. Wallace, K. Xiao, R. Leike, L. Weng, J. Heidecke, and A. Beutel, "The instruction hierarchy: Training llms to prioritize privileged instructions," *arXiv preprint arXiv:2404.13208*, 2024.

[16] S. Chen, A. Zharmagambetov, S. Mahloujifar, K. Chaudhuri, and C. Guo, "Secalign: Defending against prompt injection with preference optimization," *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:273227822

[17] S. Armstrong, "Using gpt eliezer prompt against chatgpt jailbreaking," 2022. [Online]. Available: https://www.lesswrong.com/posts/pNcFYZnPdXyL2RfgA

[18] S. Kokkula, G. Divya *et al.*, "Palisade–prompt injection detection framework," *arXiv preprint arXiv:2410.21146*, 2024.

[19] Y. Chen, H. Li, Y. Sui, Y. He, Y. Liu, Y. Song, and B. Hooi, "Can indirect prompt injection attacks be detected and removed?" in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025, pp. 18 189–18 206.

[20] PyMuPDF Developers, "Pymupdf," 2021. [Online]. Available: https://github.com/pymupdf/PyMuPDF

[21] pdfplumber Developers, "pdfplumber," 2020. [Online]. Available: https://github.com/jsvine/pdfplumber

[22] Y. Shinyama, "Pdfminer," 2013. [Online]. Available: https://github.com/euske/pdfminer

[23] PyPDF2 Developers, "Pypdf2," 2020. [Online]. Available: https://github.com/mstamy2/PyPDF2

[24] S. Bhawal, "Resume dataset," 2021. [Online]. Available: https://www.kaggle.com/datasets/snehaanbhawal/resume-dataset

[25] F. Chen, "Iclr papers dataset (2017–2019)," 2019. [Online]. Available: https://www.kaggle.com/datasets/chenfanchao/iclr-papers-datasets-20172019

[26] praj, "End-to-end resume ats tracking llm project with google gemini pro (etegemini)," 2024. [Online]. Available: https://github.com/praj2408/End-To-End-Resume-ATS-Tracking-LLM-Project-With-Google-Gemini-Pro

[27] S. Prajapati, "Resumeperfector.ai," 2023. [Online]. Available: https://github.com/shubhamprajapati7748/resume-perfector.ai

[28] A. Aryan, "Resumegpt," 2023. [Online]. Available: https://github.com/aliaryan/ResumeGPT

[29] ShiwenNi, "Chatreviewer: Llm-based paper review system," 2024. [Online]. Available: https://github.com/nishiwen1214/ChatReviewer

[30] C. Lu, C. Lu, R. T. Lange, J. Foerster, J. Clune, and D. Ha, "The ai scientist: Towards fully automated open-ended scientific discovery," *arXiv preprint arXiv:2408.06292*, 2024.

[31] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv e-prints*, pp. arXiv–2407, 2024.