Agentic Data Sanitizer: LLMs in Data Loss Prevention

Capstone Project - iDox.ai

Yijie Lu[1] **Ethan Nguyen** [1] **Colton Rowe**[1] **Gongzhi Wang**[1]

[1]University of California, Los Angeles

`{yijielu0317,edn,coltonrowe,wanggon4}@g.ucla.edu`

Abstract

This project presents the design and implementation of multiple AI-driven agents for automated data sanitation. These system integrates large language models (LLMs) with orchestration mechanisms to detect, evaluate, and redact sensitive information from documents and communications. By combining detection, contextual analysis, and selective redaction tools, the agents provide a pipeline that minimizes human intervention while maintaining accuracy and reliability. The work addresses core challenges such as ambiguity in language, consistency across diverse file formats, and scalability under real-time workloads. Results demonstrate that the approach can effectively balance automation with user oversight, offering a practical framework for enhancing data loss prevention in organizational contexts.

1 Introduction

1.1 Problem Setting

With large language models' capability in achieving high benchmarks on zero and few shot learning tasks, it has been increasingly popular to use the models as decision making agents [8]. "Agentic AI" leverages the abilities of LLMs in natural language tasks ranging from programming to business, often equipping the models with tool calls and enabling the agents to take action in a virtual environment. Structuring the flow of data through agent-based systems involves data and UX engineering working together to enable LLMs to perform tasks using tool calls or written text. Systems like Google DeepMind's AlphaEvolve iteratively improves code uses LLM workers in the generation, evaluation, and even prompting steps [17], while other agents are observed to act as greedy game theoretic players players when placed in negotiation environments [19].

Preprint. Under review.

Data loss prevention (DLP) is an involved process that companies employ to prevent sensitive data leakages. Traditional DLP systems use search-based methods to redact and replace potentially sensitive content like phone numbers or bank details. In this work, we explore how the decision making capabilities of LLMs broaden the abilities of these data loss prevention systems. We will propose three novel data loss prevention frameworks and compare them to traditional DLP systems, as well as adjacent agentic systems.

1.2 Conceptual Framework

In this paper, we will refer back to the conceptual framework of LLM agents as game players performing actions in an environment. In the context of data loss prevention, the environment can be thought of as the scope of data passed into a DLP system, and the agent's actions are the natural language tasks and tools that it is equipped with. In this work, we intend to leverage the few shot learning capabilities of LLMs to act as agents in a data sanitation environment, and compare this approach to traditional find and replace methods. Within this framework, these human-coded replacement programs are still agents acting in an environment - here, we suggest using LLMs instead of or in tandem with these programs to reduce data leakages. Further work could explore post training language models to fine-tune them to the task of data loss prevention; in this work, we explore the application of LLMs to the specific domain of data loss prevention.

1.3 Variables and Key Concepts

- Data Loss Prevention (DLP) - Data loss prevention systems aim to prevent data leakages before they happen, by redacting information and shielding sensitive data from external parties. This system concept is about proactively stopping data leaks, not just reacting to them. By tasking the system to not only identify sensitive information but also filter it out using an agent.

- Agent - A decision-maker that can take actions in an environment. Examples of agents in the context of DLP are LLMs, find and replace methods, and RL models. In this case it is simply the entity that performs the data sanitation. The most basic agent would parse and have designated rules to find any particular sequence, while more complex agent are able to make nuanced decisions that are based on the context of the data. Some models can even be trained overtime to not rely on fixed patterns, creating unique programs fit for each users' private data.

- Large Language Model (LLM) - A machine learning model that produces next-token predictions based on previous inputs. In this study, we treat LLMs as agents working in a

DLP environment and may use the two terms interchangeably at times. In this paper, we use GPT 4o to power our systems, though we expect to see similar results with other LLMs.

1.4 Problem Statement

Our goal in this work is to research how using LLMs as agents in data loss prevention systems affects user experience, accuracy, and security. In doing so, we have constructed a set of frameworks which answer three driving questions to address this goal.

1. User Experience - How does the company associate pass data through our framework?

This goes beyond designing a simple upload interface, but rather something that the user will be able to confidently review, submit feedback on, and approve. The success of the entire framework hinges on the human-in-the-loop review process, and the agent's effectiveness is reliant on the feedback mechanism provided by the human user.

2. Agent Design - What actions can the agent take to filter out sensitive data?

The agent's actions are not simple, pre-programmed rules, but must instead by dynamic, context-aware decisions. By giving the agent a general set of instructions, it must parse the prompts and search external infromation to create its own set of context to detect a wider range of sensitive data. The preview where it communicates its intended changes will move beyond pattern recognition, and will eventually showcase a deeper understanding of what makes data sensitive given a certain context. The agent needs to be able to balance the goal of comprehensive redaction to preserve a document's or email's utility and open ambiguity for the human user.

3. Security - How do we make the entire framework secure?

An agent designed to handle a user's most sensitive information must be fortified against multiple threats. The framework should prevent any prompt injection attacks, where a malicious user could craft a prompt that tricks the agent into revealing sensitive data or bypassing sanitization rules. It must also guarantee that the LLM, which may use a third-party service, does not retain or expose any sensitive data it processes. The secure pipelines must implement robust access controls, encrypt all data, and guarantee a redacted data without any leaks.

1.5 Sub-problems

The first problem resides in the trigger mechanism, where the agent decides when to act. This includes detecting new files or inputs, and is the first step in the function execution. Ideally, the function will automatically execute and set up the entire sanitization process without any manual intervention. The next problem would help define what the agent actually does, focusing on the core logic and how the sequence of automated tasks is decided. This involves leveraging a language model to parse the data provided within a document or email, deciding the sensitivity of the tokenized words, and calling a redacting tool to remove the sensitive info. Finally, the agent has to unfold all of these tools in the correct order, including going through user approval and interfacing with other tools to provide redaction. This orchestrator will follow the detector and decide whether the redactor or detector needs to be called again.

The second sub-problem lies in ambiguity resolution during detection. Language models can struggle when a word or phrase carries multiple possible meanings, where only one interpretation may be considered sensitive. Determining whether the tokenized context signals a need for redaction or preservation requires fine-grained contextual analysis. Without clear resolution, the agent risks either over-redacting content, which reduces utility, or under-redacting, which compromises security. Balancing these competing risks is central to ensuring the system performs consistently across diverse inputs.

The third sub-problem is maintaining consistency across different document structures and formats. Documents, emails, and structured files often present sensitive information in varied layouts, such as tables, headers, or embedded metadata. Adapting the agent to reliably detect and redact across these heterogeneous formats is technically challenging, as the system must not only parse the content but also preserve readability and integrity of the output after sanitization.

1.6 Performance Metric

To evaluate our proposed frameworks against similar tools, we will synthesize potentially sensitive text data and score each framework by the scope, accuracy, and ease of use. These metrics will refer to the precision and accuracy of the data sanitized, and provide a general score across multiple models.

1.7 A Priority Hypothesis

We hypothesize that an agent-based DLP framework will achieve a higher percentage of correct sensitive content redactions compared to traditional rule-based DLP systems.

1.8 Assumptions and Delimitations

This study focuses solely on textual data and does not address multimedia formats. We limit comparisons to traditional search-based DLP systems and do not evaluate hybrid approaches.

1.9 Importance of the Study

By integrating LLM-based agents into the DLP process, this study addresses the growing need for contextually aware and adaptable data protection systems. The proposed frameworks aim to improve redaction accuracy, reduce false positives and negatives, and create a more efficient workflow through automation and human-in-the-loop validation. These contributions are particularly significant for organizations handling large volumes of sensitive communications, where traditional pattern matching approaches may be insufficient. Additionally, this study expands the general research of application of LLMs as decision making agents.

2 Related Works

2.1 LLMs as Iterative Mutation Observers

Google DeepMind's AlphaEvolve iteratively improves code by using LLM workers in the generation, evaluation, and prompting steps [17]. Alpha Evolve has made several mathematical breakthroughs including advancements in matrix multiplication algorithms and shape packing problems. The algorithm behind Alpha Evolve starts with an empty python file and passes it through a distributed controller loop until the output achieve some quantifiable evaluation threshold. First, the system samples prompts, which have a human-written base plates but which employ an additional layer of "meta-prompt evolution" that evaluates and selects promising prompts. Next, an LLM ensemble uses these prompts to generate a series of diff changes that are intended to improve the code. The modified code is then run through a user-provided evaluation function and the best modifications are kept and improved on in the next iteration. This method itself is an evolution of an earlier work out of Google Deepmind: FunBo [1], which uses Bayesian optimization to select and evolve valid black-box functions which are difficult to evaluate. While program evolution is not directly related to data loss prevention, these data pipelines outline how a system can employ an LLM into an agentic framework. We will use similar frameworks to generate diff commands intended for data redaction rather than code evolution.

2.2 Greedy Game Theoretic Players

Taking inspiration from reinforcement learning, LLM agents can be viewed as game players performing actions in an environment. When tasked to operate as a budgeting manager, these models exhibit greedy-like behavior in resource allocation [10]. Similar greedy behavior is exhibited when they engage in negotiation and auction games [19]. LLMs benchmarks now span a wide range of generalist and specialist evaluations [20], so research is focused on where best to apply these models. Reinforcement learning with human feedback (RLHF) is a critical element in fine-tuning foundation models into helpful generalist agents, so it follows that game playing will improve with the quality of the models themselves. In the context of data sanitation, LLMs will leverage their contextual understanding to decern what content should and should not be flagged as sensitive, which can also be though of as playing a game. Using true RL to fine-tune these models would require collecting data out of the scope of this project. Still, equating the models to RL agents in a data sanitation environment gives us insight into how to structure our data loss prevention frameworks.

2.3 Domain-Specific Applications of Sanitization

LLM-based anonymization has also been tested in specialized domains. Hasegawa's work in the medical field shows that LLMs outperform rule-based anonymization tools by offering stronger resistance to adversarial inference while preserving more data utility [13]. This demonstrates the trade-off between privacy and usability in sensitive contexts. Our project, while domain-agnostic, can draw from these findings by recognizing that domain-specific fine-tuning or evaluation may be necessary to ensure sanitization maintains data quality without over-scrubbing.

2.4 Hallucinations and Adversarial Explanations

As generalists that mimic many types of authors, Large Language models can exhibit unwanted behaviors in certain contexts. When prompted, LLMs have been shown to fabricate explanations for false conclusion which convince both human researchers other LLM agents[2]. These generously named "hallucinations" stem from data, training, and inference stages [12]. In the context of DLP, these fabrications can be mitigated by further fine-tuning to human preferences, but using RL is too computationally expensive for this study.

2.5 Trade-offs in Data Sanitization

The broader implications of sanitization have been explored in empirical studies by Amazon researchers, who found that aggressive sanitization improves privacy protection but may degrade model performance across tasks such as classification and sentiment analysis [4]. This trade-off is central to our project, as excessive redaction could reduce the usefulness of data for downstream applications. These findings highlight the importance of calibration: a sanitization system must balance strict protection with retaining sufficient semantic fidelity.

2.6 Trust, Risk, and Security Management Frameworks

Finally, larger frameworks such as TRiSM (Trust, Risk, and Security Management) provide conceptual lenses for evaluating agentic systems. A recent survey identifies explainability, lifecycle governance, and privacy protection as critical gaps in current AI deployments [5]. For our work, this reinforces the need to situate technical sanitization strategies within a broader framework that accounts not only for adversarial robustness, but also for organizational trust and compliance requirements.

2.7 Human-Computer Interfaces for LLMs

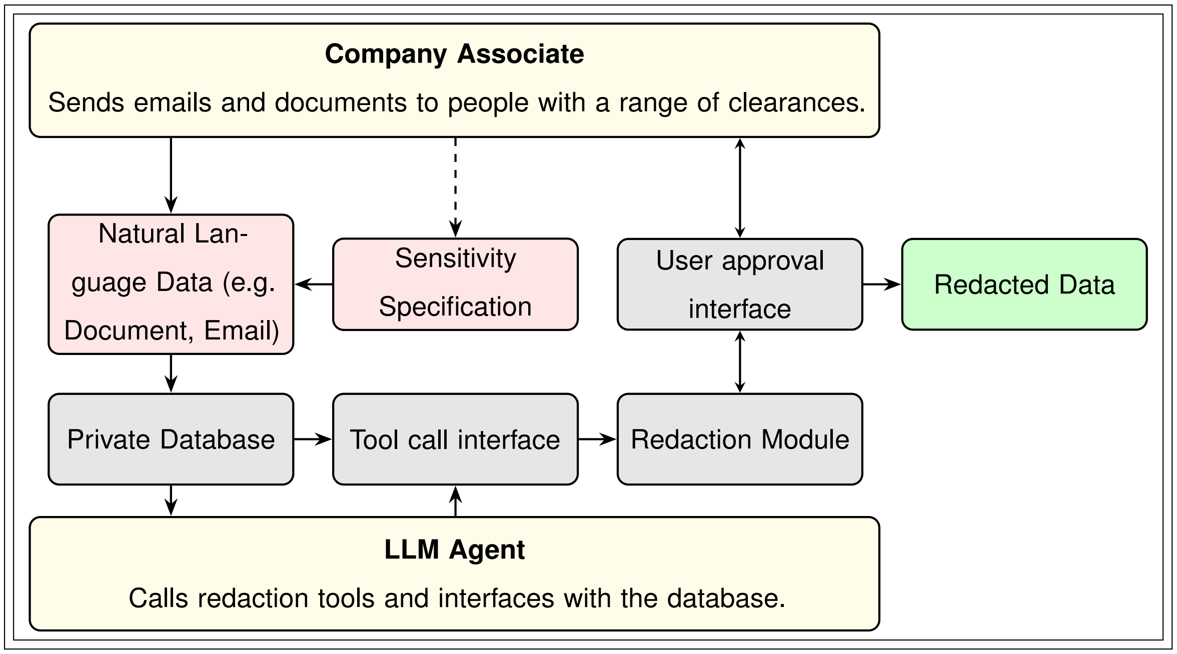
A core driver of agentic systems is human-computer interaction (HCI). Systems like Windsurf and Cursor equip agents with automatic in-line editing, while interfaces like ChatGPT use chat boxes to give more control to the user. Some methods involve personalization of the agent to the user using memory logs and other context management schema [14]. Because data loss prevention largely focuses on how company associates transmit data externally, the user-agent interaction experience is core to how our agentic frameworks are structured.

3 Methods

We developed two full-stack frameworks which approach data loss prevention in unique ways. All of the code for both frameworks can be found on our Github at https://github.com/e10-nguyen/Agentic-Data-Sanitizer.

3.1 Framework 1

This first framework uses Gmail and Google Docs as an interface to sanitize email drafts and documents before they are sent. Intercepting sensitive data before it is exposed is a core tenet of data loss prevention.



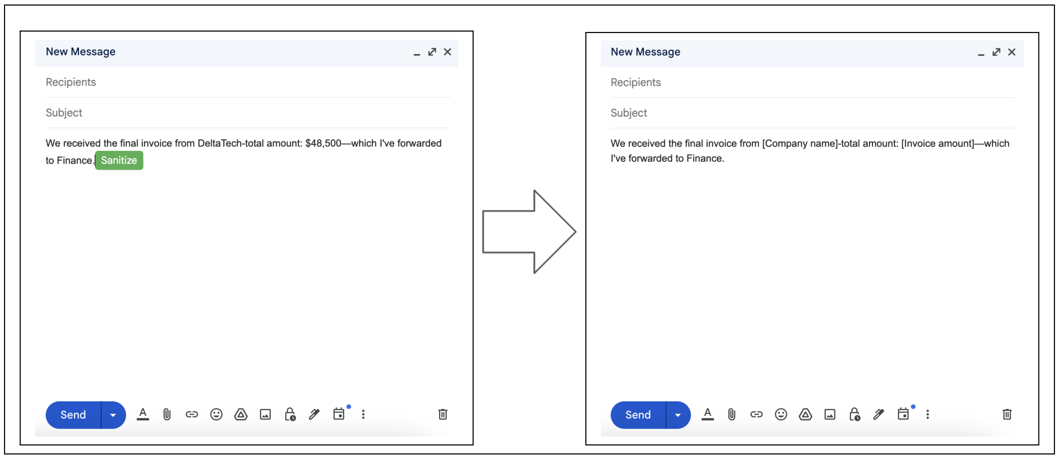
Company Associate
Sends emails and documents to people with a range of clearances.
Natural Language Data (e.g. Document, Email)
Sensitivity Specification
User approval interface
Redacted Data
Private Database
Tool call interface
Redaction Module
LLM Agent
Calls redaction tools and interfaces with the database.

Technologies Used

- **Chrome Extensions** - Chrome extensions are written in HTML and JavaScript, and act as additional programs that run within the context of a browser. Much of the work creating the extension was the modification of the content.js, background.js, and popup.js scripts. The content.js script is run within the context of a webpage, and starts up whenever a new page is visited. The content.js script is critical in getting and setting information in the context or within the DOM of the current webpage. Working in the background, background.js runs when Chrome is first launched, and can handle actions that need to work asynchronously or between systems, such as API calls and other functions. In our system, we used background.js script as a middleman between modules and APIs by using message passing. This involves calling `runtime.sendMessage()` or `tabs.sendMessage()` in some other script in the extension, which can send and receive a response from the background script. Finally, the popup.js script is called whenever the extension itself is selected from the chrome extension menu, and allowed us to create a user interface and other visual elements. For the popup.js script, we implemented

- **FastAPI** - FastAPI is a modern python framework for REST API development. APIs, or application program interfaces, allow for communication between two systems within a broader network or framework. RESTful APIs implement standard HTTP methods like GET and POST to allow for representational state transfer between a host and a client [6]. Here, our Chrome extension acts as the client, and our backend written in python acts as the host. We created methods like `sanitize` and `sendsq` to sanitize text and set the sanitation qualifier respectively Our implementation isn't truly RESTful because we set a retained sanitation qualifier between API calls to simplify our framework, but it could be easily adapted to RESTful design by grouping the qualifier with the sanitation request.

- **Azure OpenAI** - OpenAI's GPT-4.1 language model has an API through Microsoft Azure which is avalible through a Python library. This API lets us interface with the model through a series of requests, and it provide access for different levels of priority through developer and user prompts. To integrate this with our Chrome extension framework, we created two separate Python classes to be used in our server: `Model` and `Agent`. The `Model` class implements a standardized interface for calling the LLMs by having them inherit from a base class containing a template for the model. From this, we implemented the `GPT\_4\_1` class which has all the same methods and data as the base Model class. By using OOP in this way, we could implement and use many different types of language models such as GPT, Gemini and Llama. Here, we chose only to implement GPT-4.1 for simplicity, but in the future, it would be interesting to test the other models for our frameworks. The `Agent` class takes in a



We received the final invoice from DeltaTech-total amount: $48,500—which I've forwarded to Finance.

We received the final invoice from [Company name]-total amount: [Invoice amount]—which I've forwarded to Finance.

Figure 1: In-line interface for the Gmail text redaction extension. Sanitation button appears in-line, and when clicked, replaces the current text with redacted text by interfacing with a FastAPI backend. This method of redaction works seamlessly to sanitize sensitive emails before they are sent.

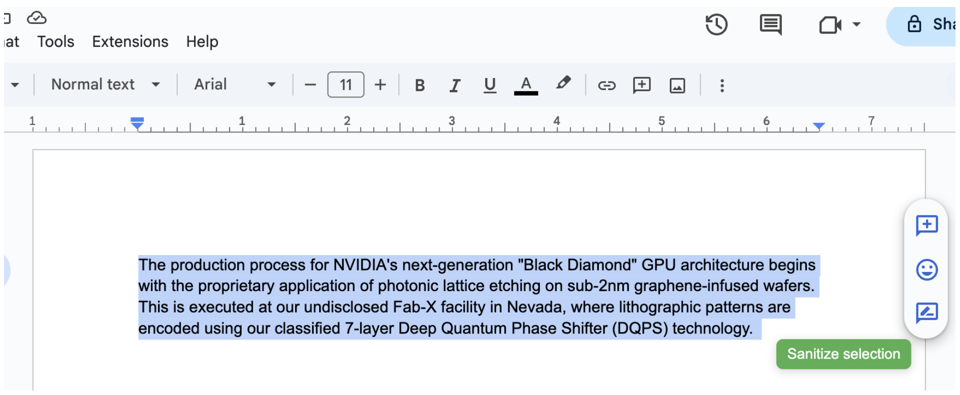
model as a parameter, and provides structure though a sequence of prompts and functions to implement a sanitation agent. The main function in `Agent` is the `sanitize\_text` function, which sanitizes the input text based on it's internal sanitation qualifier and the developer prompts we provide. We are then able to use this agent in our FastAPI server to sanitize the Gmail drafts or other provided text.

To implement the Gmail front end, we created a chrome extension that automatically edits the content of a new email draft. Chrome extensions primarily use content and background scripts to adaptively modify the content of a page. We use a listener in the content script to intercept the HTML of the page by finding the last selected text area. Implementing the Google Docs front end instead required us to gather data from the user's clipboard to perform the redaction.

Algorithm 1 Framework 1, Front End

```
▷ A simplified picture of how the content.js and background.js scripts work together. ◁
lastActive = None
lastActiveValue = ""
while True do
    if getLastActive().innerHTML != lastActive.innerHTML then
        lastActive = getLastActive()
        lastActiveValue = lastActive.innerHTML
    if button.clicked() then
        sanitizedText = fetch("POST http://localhost:3000/api/sanitize/" + lastActiveValue)
        updateHTML(lastActive.id, sanitizedText)
```

The back end pipes the text through a LLM and responds with the newly sanitized text. To implement the back end, we used Python with FastAPI because Python arguably has the broadest capabilities for interfacing with machine learning models. In this framework, we used a sequence of prompts to best identify and replace potentially sensitive content. We found that asking the model to do too



The production process for NVIDIA's next-generation "Black Diamond" GPU architecture begins with the proprietary application of photonic lattice etching on sub-2nm graphene-infused wafers. This is executed at our undisclosed Fab-X facility in Nevada, where lithographic patterns are encoded using our classified 7-layer Deep Quantum Phase Shifter (DQPS) technology.

Figure 2: Floating button interface for the Google Docs text redaction extension. Sanitation button is floating when text is copied, and when clicked, replaces the clipboard with redacted text by interfacing with a FastAPI backend. This method allows for redaction among a range of scenarios when dealing with a broader range of documents.

much in one step would lead to more pronounced formatting mistakes in the responses, leading to a higher error rate in the system overall. The AI works well when generating boilerplate or identifying erroneous results. The most useful application of generative AI in programming is in navigating APIs. After decades of effort, large-scale software reuse has now become a reality through open source software libraries and package managers that can easily install them. [15]

This first prompt is meant to help identify if any redaction actually needs to be performed, and give some direction to the next agent. We ask the model to first explain if the text is sensitive based on a set of guidelines, then reply with it's decision of whether or not the text is sensitive.

```
IDENTIFICATION_PROMPT = \
'''You are a redaction agent. Your task is to evaluate whether a given
     piece of text contains information that would be considered
    sensitive from the perspective of the user.

The user has identified that these elements are sensitive: """
    SENSITIVE_INFORMATION""". Do not redact anything else.

Respond with: The text may be sensitive because [brief explanation].
    The text may not be sensitive if [breif explanation]. [Y/N: Y if
    the text is sensitive, N if the text is not sensitive.]

Example input: Also, if you need VPN access while traveling, use the
    temporary credentials: username: j.doe@company.com, password: ***
    REDACTED_PASSWORD*** .
```

```
Example output: The text may be sensitive because it contains login
    creditials. The text may not be sensitive if the receiver is
    trusted. Y
'''
```

This second prompt provides the syntax for the tool call that the model will make given the explanation and decision in the last step.

```
REDACTION_PROMPT = \
'''You are a redaction agent. The user will provide some sensitive
    text. Your task is to redact the text by replacing sensitive
    information with a brief description of the information.

Redact some parts of the text by responding with this command
    separated by newlines:
REPLACE [text to be replaced] REPLACE_WITH [one to two word
    description of the text]
Example input:
Also, if you need VPN access while traveling, use the temporary
    credentials: username: j.doe@company.com, password: ***
    REDACTED_PASSWORD*** .
Example output:
REPLACE j.doe@company.com REPLACE_WITH VPN email
REPLACE ***REDACTED_PASSWORD***  REPLACE_WITH VPN password

The user has identified that their text is sensitive because: """HERE
    """. Redact only the parts that fall under the category of: """
    SENSITIVE_INFORMATION""". Do not redact anything else. Only
    respond with the list of commands, do not include any other text.
'''
```

The current implementation of the tool call interface works much like the one in AlphaEvolve, by having the LLM use the syntax `REPLACE {some text} REPLACE_WITH {replacement text}`. API's like OpenAI's GPT-4o divides the developer prompts and user prompts with different levels of priority; here, we use developer prompts to specify what the model's syntax and output should look like, and use user prompts to feed in the potentially sensitive data. This ensures that the context of the data does not interfere with the format of the model's output.

When using an LLM as an agent in this way, we are treating it as a black box that takes in some instructions and produces a response from those instructions. We have to be careful when treating the LLM as a function because it has a larger scope than the rest of the program might expect. For example, if we tell the LLM to use the "REPLACE" syntax but the model gets confused and uses a different term, our code should handle these kind of exceptions. Having the model explain it's reasoning before making a decision mitigates the number of adversarial black box explanations the model will produce without having to do any additional fine-tuning[2].

3.2 Framework 2

We've developed the Agentic PDF Data Sanitizer, an intelligent, multi-node system designed to automatically detect, highlight, and redact sensitive information from PDF documents. Our system is built using the LangGraph and LangChain frameworks, combining Azure AI services with human-in-the-loop validation to ensure accurate and compliant data sanitization. The core functionality is to intelligently clean sensitive information from PDFs, providing a secure and efficient solution for handling confidential documents.

Table 1: The pipeline is structured around this set of nodes.

Node	**Input**	**Output**
Orchestrator	User Prompt and PDF address	Next Node ("Searcher" or "Detector")
Searcher	Search query.	Sensitive data information.
Detector	Sensitive data info and PDF address	Converted elements (text and converted coordinates) Sensitive information JSON (content, type, coordinates)
Highlighter	Sensitive Information JSON (coordinates)	Preview PDF with Highlights
Evaluator	Preview PDF, Sensitive JSON, Converted Elements	"Redactor" or "Highlighter" and updated sensitive JSON
Human-in-the-Loop	Preview PDF	"Y"/"N" decision, User's Feedback
Redactor	Original PDF Address and Sensitive JSON (coordinates)	Final Redacted PDF Address

System Architecture

We designed a robust architecture leveraging state-of-the-art technologies to achieve our goals.

Core Technologies

- **AI Framework**: We use LangGraph for workflow orchestration and LangChain for creating structured outputs from the Language Model.

- **Language Model**: Azure OpenAI GPT-4o serves as the brain of our system, handling intelligent decision-making and data detection.
- **OCR Engine**: Azure Document Intelligence provides precise, word-level text and coordinate extraction from PDFs.
- **Search**: The Tavily API allows the system to conduct external research for regulatory compliance.
- **PDF Processing**: We utilize PyMuPDF for all PDF manipulations, including coordinate conversion, highlighting, and redaction.
- **UI**: A Streamlit interface facilitates our human-in-the-loop (HITL) interaction and validation steps.
- **API**: An application programming interface is a connection or fetching, in technical terms, between computers or between computer programs. It is a type of software interface, offering a service to other pieces of software..

3.2.1 Node-Based Architecture

Our system operates through a graph of seven specialized nodes. While a shared state dictionary flows between them, we've defined inputs and outputs for each node to clarify which state variables are required and which are updated during its operation.

1. **Orchestrator Node**
   The Orchestrator is the entry and routing hub of our workflow.
   - **Primary Functions:**
     - As the graph's entry point, it processes the initial user prompt and the PDF file. It uses Azure OpenAI to analyze the prompt, summarize it into structured "sensitive data descriptions," and decide the next step. If the prompt requires external research (e.g., "redact all data according to HIPAA"), it generates a search query and routes to the Searcher; otherwise, it proceeds to the Detector.
     - After a human review cycle where a user provides feedback, the Orchestrator re-evaluates the new hints. It appends the new guidance to the existing sensitive data descriptions and re-routes the workflow, again deciding between the Searcher and Detector.
   - **Inputs:** `user_prompt`, `pdf_path`, optional feedback hints from HITL.
   - **Outputs:** The next node decision ("Searcher" or "Detector"), an updated `sensitive_data_description` list, and an optional `search_query`.

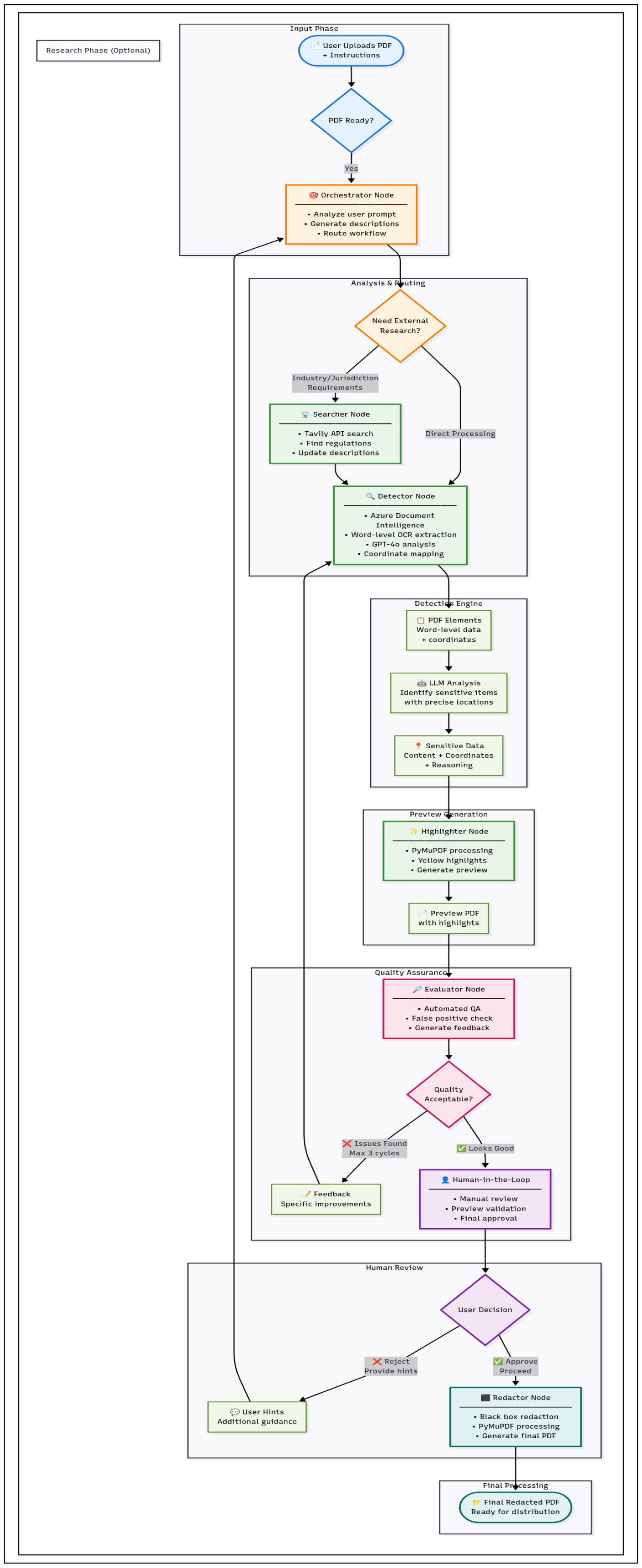


Figure 3: The PDF Agentic Data Sanitizer Pipeline

2. **Searcher Node (Optional)**

   This node enriches our detection context with external knowledge.

   - **Primary Functions:** Based on a search query from the Orchestrator, we use the Tavily API to find relevant compliance requirements for specific jurisdictions or industries. The findings are then used to enhance the list of sensitive data descriptions.
   - **Inputs:** `search_query`.
   - **Outputs:** An enhanced `sensitive_data_description` list.

3. **Detector Node (Core Detection Engine)**

   This is where the primary sensitive data identification occurs.

   - **Primary Functions:**
     - **Case 1 (Initial Run):** When first called, the Detector uses Azure Document Intelligence to perform OCR, extracting every word as an element with precise coordinates and a page number. We convert these coordinates to the PyMuPDF format (72 points/inch) and assign a unique `element_id` to each word for tracking. We then send a structured payload of these elements to Azure OpenAI, which identifies sensitive items, groups related words (e.g., "01 JANUARY 1980"), and returns a list of `SensitiveItem` objects with their content, coordinates, and a justification.
     - **Case 2 (Refinement):** If the workflow loops back from the Evaluator with feedback, the Detector re-analyzes the already-extracted `pdf_elements`. It focuses on the new feedback to modify the existing list of sensitive data, either by adding missed items or removing false positives.
   - **Inputs:** `sensitive_data_description`, optional existing `pdf_elements`.
   - **Outputs:** A complete list of `pdf_elements` and a list of identified `sensitive_data` items.

4. **Highlighter Node**

   This node creates a visual preview for human validation.

   - **Primary Functions:** Using the coordinates from the detected `sensitive_data`, this tool generates a preview PDF. It applies a yellow highlight over each sensitive area using PyMuPDF, allowing for clear visual verification.
   - **Inputs:** `sensitive_data`, `pdf_path`.
   - **Outputs:** The file path to the `preview_pdf_path`.

5. **Evaluator Node (Quality Assurance)**

   This automated QA step refines detection accuracy before human review.

   - **Primary Functions:** The Evaluator uses an LLM to review the highlighted preview against the full list of sensitive data descriptions and the extracted PDF elements. It checks for false positives and false negatives. If discrepancies are found, it generates specific feedback and routes back to the Detector. To prevent infinite loops, we limit this feedback cycle to a maximum of three iterations. If the preview is satisfactory, it routes to the Human-in-the-Loop node.
   - **Inputs:** `sensitive_data_description`, `preview_pdf_path`, `pdf_elements`.
   - **Outputs:** A quality assessment, a routing decision, and optional `evaluator_feedback` appended to the sensitive data description.

6. **Human-in-the-Loop (HITL) Node**
   This is the final checkpoint before irreversible redaction.
   - **Primary Functions:** We present the highlighted preview PDF to the user for manual validation. The user can either approve the redaction plan or reject it. If they reject it, we prompt them for hints, which are sent back to the Orchestrator to refine the process.
   - **Inputs:** `preview_pdf_path`, user approval status ("Yes" or "No").
   - **Outputs:** An approval decision and optional user hints for the Orchestrator.

7. **Redactor Node (Final Processing)**
   This node performs the final, permanent sanitization.
   - **Primary Functions:** Once we receive user approval, the Redactor uses the precise coordinates of the sensitive data to apply permanent black boxes over the targeted text in the original PDF. It then saves this sanitized version as the final output.
   - **Inputs:** `pdf_path`, `sensitive_data`.
   - **Outputs:** The file path for the final `final_pdf_path`.

3.2.2 Workflow Logic

Our system's intelligence lies in its conditional routing and feedback loops.

Routing Intelligence

- **Orchestrator $\rightarrow$ Searcher/Detector:** The initial routing is based on an LLM's assessment of whether the user's prompt requires external regulatory research.
- **Detector $\rightarrow$ Highlighter:** After detection, the workflow always proceeds to generate a visual preview.
- **Evaluator $\rightarrow$ Detector/HITL:** This routing is based on our automated quality check. The system either loops back for refinement or proceeds to human review.

- **HITL → Redactor/Orchestrator:** The final routing depends entirely on user approval. A "Yes" proceeds to final redaction, while a "No" loops back to the beginning with new user feedback.

Feedback Loops

- **Evaluator Loop:** An automated, self-correcting loop for quality improvement, limited to three cycles to prevent recursion errors.
- **HITL Loop:** A human-driven refinement loop that continues until the user is satisfied with the preview.
- **State Persistence:** All feedback, whether from the Evaluator or the user, is accumulated in the `sensitive_data_description` list, ensuring the system's context improves with each cycle.

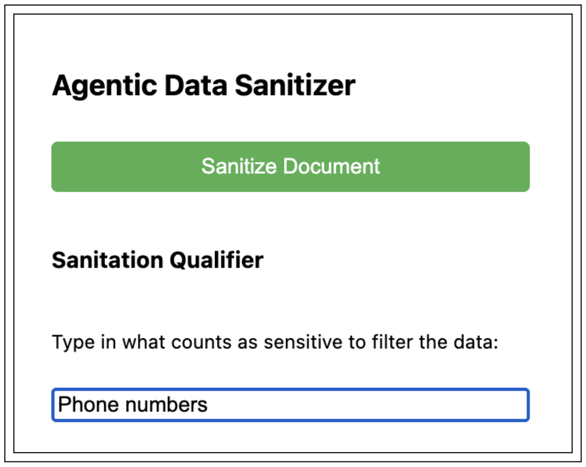
4 Results

4.1 Framework 1

To test the abilities of the first frame we'll compare it's limits, accuracy, and scope. These experiments are meant to test the efficacy of using a LLM over traditional methods such as using regular expressions to make changes, exploring point no. 2 of our framework goals. In the future, we would like to develop metrics for user experience and security as well, but for this instance we will rely on theoretical analysis. The first metric would be about recall, or the percentage of actual sensitive data of the entire document that the agent was able to find. Next would be about precision, measuring out of all the words that were censored, how many of these words are actually sensitive. After calculating both of these, we can then calculate an F1 score [9], which balances both metrics and ensures that the only the best models that are able to catch all the sensitive data (high recall) without redacting too much normal text (high precision) are given the highest scores.

4.1.1 Comparison

Recent work as been done to improve on traditional find-and-replace methods such as RegEx [18]. In this section, we will test the redaction of our framework against frameworks that use regex methods. To do this, we will find the equivalent regular expressions to our sanitation qualifier required to perform some redaction task, if possible. Here, the sanitation qualifier is the a prompt selected or created by the user that defines what is considered sensitive and within the scope of sanitation. Because we expect this framework to be most useful for enterprise, we selected our default sanitation qualifier to be:


Agentic Data Sanitizer

Sanitation Qualifier

Type in what counts as sensitive to filter the data:

Phone numbers

Figure 4: User input field for the sanitation qualifier.

```
Company specific information which is not publicly available.
```

For these tests though, we'll use prompts that can be adequately captured by regular expressions. We'll compare the regular expression to our sanitation qualifier and determine if the sensitive data gets redacted. We'll also qualitatively discuss how the sanitation qualifier compares to the regex expression.

Table 2: Accuracy test cases comparing regex and natural language sanitation qualifiers.

Sanitation Qualifier	**Equivalent RegEx**	**Raw Text**	**Sanitized Text**
Phone numbers	`^(\+\d{1,2}\s)?\(?\d{3}\)?[\s.-]\d{3}[\s.-]\d{4}\$`	Call me at (415) 555-1234 or 415-555-5678.	Call me at [phone number] or [phone number].
Email addresses	`[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}`	Contact us at support@example.com or hr@company.org.	Contact us at [support email] or [HR email].
IP addresses	`\b(?:\d{1,3}\.){3}\d{1,3}\b`	Server logs show 192.168.0.1 failed to connect to 10.0.0.42.	Server logs show [Source IP address] failed to connect to [Destination IP address].
SSNs	`\b\d{3}-\d{2}-\d{4}\b`	Employee SSN: 123-45-6789.	Employee SSN: [SSN].
Credit card numbers	`\b(?:\d[ -]*?){13,16}\b`	Payment received from card 4111 1111 1111 1111.	Payment received from card [credit card number].

4.1.2 Scope

The main advantage of our framework is that the scope is much broader than that of traditional find and replace methods such as RegEx. Our framework captures natural language context that regular

Table 3: Redaction scope test

Sanitation Qualifier	Raw Text	Sanitized Text
Company specific information which is not publicly available.	We received the final invoice from DeltaTech—total amount: $48,500—which I've forwarded to Finance.	We received the final invoice from [Company name]—total amount: [Invoice Amount]—which I've forwarded to Finance.
Company specific information which is not publicly available.	El informe financiero de TechNova S.A. muestra que en el segundo trimestre de 2025 tuvo pérdidas por $1.2 millones debido a una filtración de datos confidenciales.	El informe financiero de TechNova S.A. muestra que en el [Time period] tuvo pérdidas por [Financial loss] debido a [Incident description].
Phone numbers	During the client onboarding call, Maria accidentally read out her mobile as +1 (415) 77-88-990 (sometimes she writes it as 415.7788990 in Slack). She also mentioned her backup line: "four one five, double-seven, eight eight nine nine zero" in case the first doesn't go through.	During the client onboarding call, Maria accidentally read out her mobile as [mobile phone number] (sometimes she writes it as [mobile phone number] in Slack). She also mentioned her backup line: "[backup phone number]" in case the first doesn't go through.

expressions simply can't be created to utilize. The redaction scope tests highlight how the scope is greater than that of traditional redaction methods. In the first scenario, when the default prompt is used, the model correctly reasons that both the dollar amount and information about a partner company are both pieces of private information, and redacts them. The second scenario tests the model's multilingual capabilities, and surprisingly, it works just as well as in the first scenario. The third scenario shows how many different formats could be considered a "phone number" and the model identifies and redacts all of this data. Something like a RegEx expression could not adequately capture all of this nuance - especially in adversarial scenarios.

4.2 Framework 2

In designing our agentic sanitizer, we determined that simply detecting and redacting data in a single pass was insufficient for enterprise-grade reliability. To address this, we integrated two critical components: the Evaluator node and the Human-in-the-Loop (HITL) node. This was the most important architectural decision we made to transform our system from a simple prototype into a reliable, self-improving, and accountable AI solution [15]. These nodes form a two-tier quality assurance framework that directly increases our system's recall.

Recall is the measure of how successfully our system finds all relevant sensitive information, thereby minimizing false negatives (missed items). We designed both nodes to be instrumental in maximizing this metric.

We specifically designed the Evaluator to hunt for false negatives. Its core function is to ask, "Based on the requirements, did the Detector miss anything?" Example: Our user prompt is "redact all sensitive data in the pdf." The Detector finds names and address but missed sensitive UID, Finanical Information like amount, and date. After the evaulator node, it detected UID as a sensitive information and still ignored amount of money. That's why we also added human-in-the-loop to further increase the recall. We gave user prompt like "You missed some finanical information, amount of money the sponsor can provide." Then it works and sucessfully detected all the sensitive information in the PDF.

5 Limitations

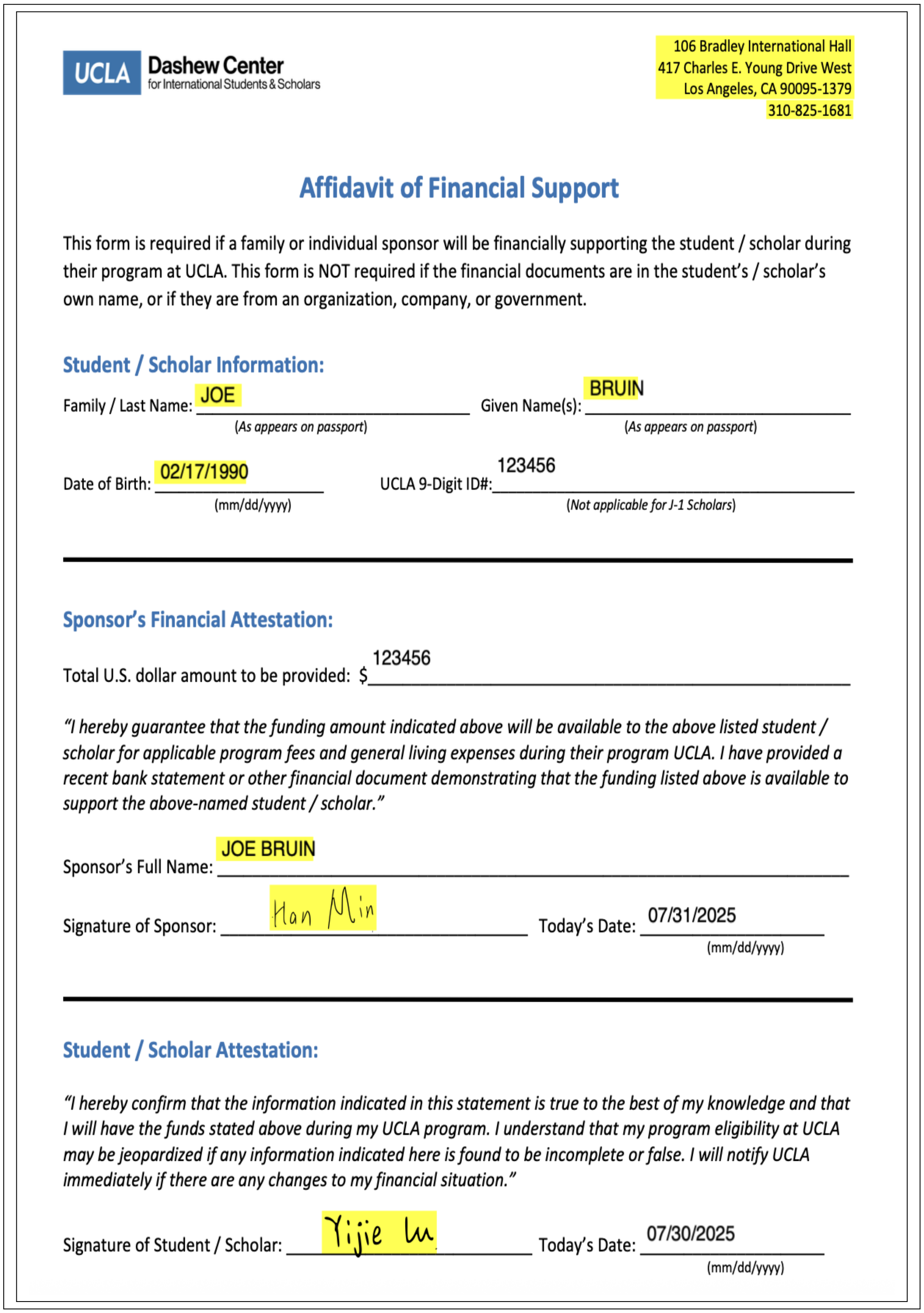
5.1 Automation

Here, we refer to automation as the ability of a system to know when it needs to be called upon. More frequent interaction points for an agent allow for a higher degree of automation. Internally, we implemented some automation between our agents - a higher level agent decides whether some text contains sensitive information, and a lower level agent performs the actual redaction. However across our frameworks, we have a low degree of automation overall. Our system detects when the user might want to use the agent, but the user must actively request for the agent to scan the data. To contrast, systems like GitHub Copilot suggestions have a high degree of automation, because the user never has to actively request for the agent to be called.

One potential approach for increasing the autonomous capacity of our systems would be to use a series of diff changes rather than scanning the whole text when passing it into the LLM. This would keep input token counts low while still capturing all of the content. However, overly frequent passes might lose some contextual information that the model needs to make accurate inference. In addition, individual API requests will make the entire system slower. For these reasons, we opted not to focus too much on the automation aspect of this project and instead focused on the structure of the system instead.

5.2 Security

Here, we refer to security constraints as the limitations imposed by relying on external services to process potentially sensitive data. While our system is designed to sanitize text effectively, it must interact with large language models through third-party APIs. This introduces a fundamental constraint: once data leaves the local environment, there is a risk that sensitive information could



UCLA Dashew Center for International Students & Scholars

106 Bradley International Hall
417 Charles E. Young Drive West
Los Angeles, CA 90095-1379
310-825-1681

Affidavit of Financial Support

This form is required if a family or individual sponsor will be financially supporting the student / scholar during their program at UCLA. This form is NOT required if the financial documents are in the student's / scholar's own name, or if they are from an organization, company, or government.

Student / Scholar Information:

Family / Last Name: JOE (*As appears on passport*) Given Name(s): BRUIN (*As appears on passport*)

Date of Birth: 02/17/1990 (mm/dd/yyyy) UCLA 9-Digit ID#: 123456 (*Not applicable for J-1 Scholars*)

Sponsor's Financial Attestation:

Total U.S. dollar amount to be provided: $ 123456

"I hereby guarantee that the funding amount indicated above will be available to the above listed student / scholar for applicable program fees and general living expenses during their program UCLA. I have provided a recent bank statement or other financial document demonstrating that the funding listed above is available to support the above-named student / scholar."

Sponsor's Full Name: JOE BRUIN

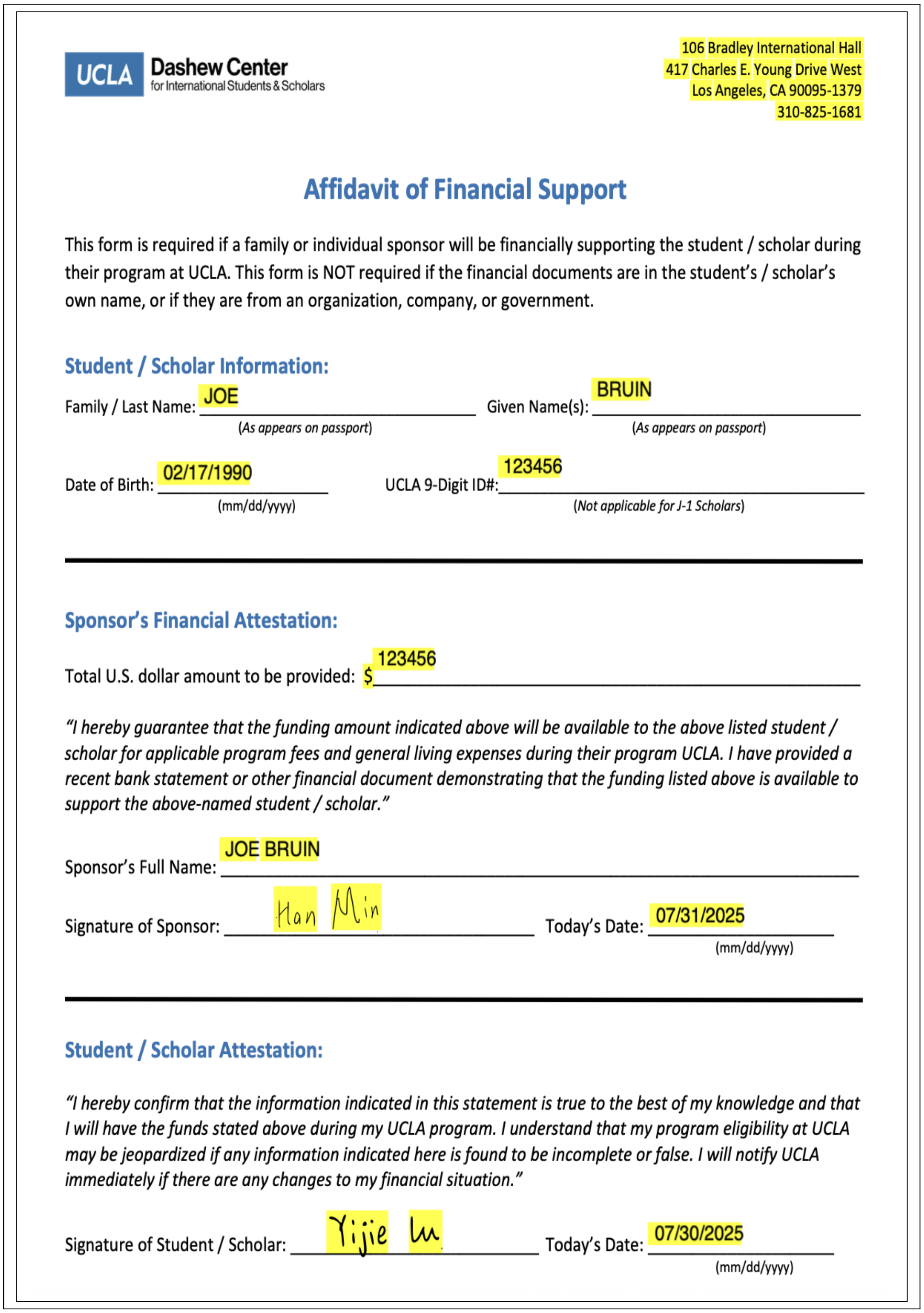
Signature of Sponsor: Han Min Today's Date: 07/31/2025 (mm/dd/yyyy)

Student / Scholar Attestation:

"I hereby confirm that the information indicated in this statement is true to the best of my knowledge and that I will have the funds stated above during my UCLA program. I understand that my program eligibility at UCLA may be jeopardized if any information indicated here is found to be incomplete or false. I will notify UCLA immediately if there are any changes to my financial situation."

Signature of Student / Scholar: Yijie Lu Today's Date: 07/30/2025 (mm/dd/yyyy)

Figure 5: The preview before evaluator node and human-in-the-loop node. Not everything gets redacted on the first pass, but there is progress.



UCLA Dashew Center for International Students & Scholars

106 Bradley International Hall
417 Charles E. Young Drive West
Los Angeles, CA 90095-1379
310-825-1681

Affidavit of Financial Support

This form is required if a family or individual sponsor will be financially supporting the student / scholar during their program at UCLA. This form is NOT required if the financial documents are in the student's / scholar's own name, or if they are from an organization, company, or government.

Student / Scholar Information:

Family / Last Name: JOE (*As appears on passport*) Given Name(s): BRUIN (*As appears on passport*)

Date of Birth: 02/17/1990 (mm/dd/yyyy) UCLA 9-Digit ID#: 123456 (*Not applicable for J-1 Scholars*)

Sponsor's Financial Attestation:

Total U.S. dollar amount to be provided: $123456

"I hereby guarantee that the funding amount indicated above will be available to the above listed student / scholar for applicable program fees and general living expenses during their program UCLA. I have provided a recent bank statement or other financial document demonstrating that the funding listed above is available to support the above-named student / scholar."

Sponsor's Full Name: JOE BRUIN

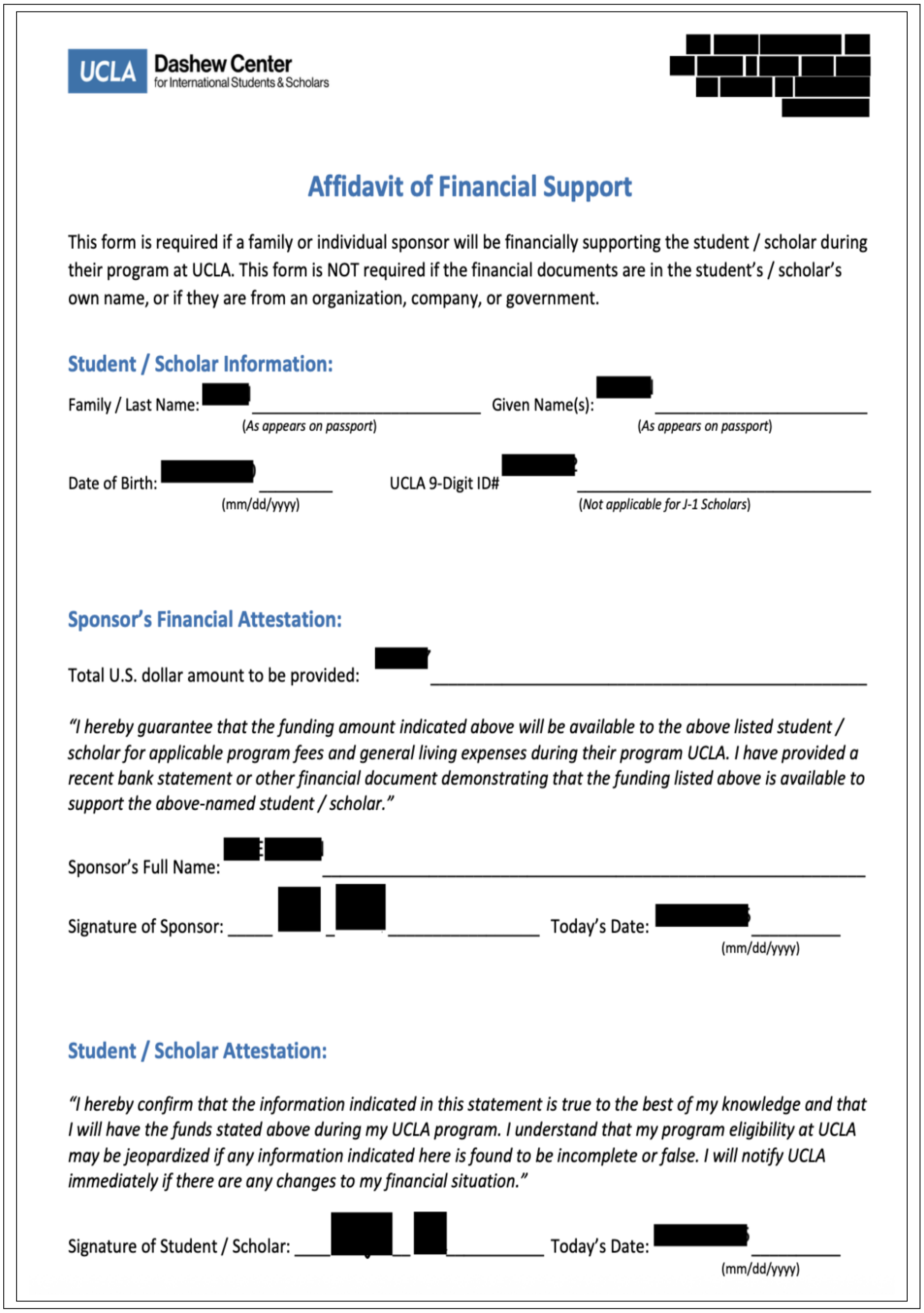
Signature of Sponsor: Han Min Today's Date: 07/31/2025 (mm/dd/yyyy)

Student / Scholar Attestation:

"I hereby confirm that the information indicated in this statement is true to the best of my knowledge and that I will have the funds stated above during my UCLA program. I understand that my program eligibility at UCLA may be jeopardized if any information indicated here is found to be incomplete or false. I will notify UCLA immediately if there are any changes to my financial situation."

Signature of Student / Scholar: Yijie Lu Today's Date: 07/30/2025 (mm/dd/yyyy)

Figure 6: The preview after the evaluator node. In this example, the evaluator successfully identified and redacted all additional sensitive content. If the evaluator misses anything, the human-in-the-loop node gives the user additional control to perform redaction on any straggling sensitive content.



UCLA Dashew Center for International Students & Scholars

[redacted]

Affidavit of Financial Support

This form is required if a family or individual sponsor will be financially supporting the student / scholar during their program at UCLA. This form is NOT required if the financial documents are in the student's / scholar's own name, or if they are from an organization, company, or government.

Student / Scholar Information:

Family / Last Name: [redacted] ____________ (*As appears on passport*) Given Name(s): [redacted] ____________ (*As appears on passport*)

Date of Birth: [redacted] ________ (mm/dd/yyyy) UCLA 9-Digit ID# [redacted] ____________ (*Not applicable for J-1 Scholars*)

Sponsor's Financial Attestation:

Total U.S. dollar amount to be provided: [redacted] ____________

"I hereby guarantee that the funding amount indicated above will be available to the above listed student / scholar for applicable program fees and general living expenses during their program UCLA. I have provided a recent bank statement or other financial document demonstrating that the funding listed above is available to support the above-named student / scholar."

Sponsor's Full Name: [redacted] ____________

Signature of Sponsor: _____ [redacted] _ [redacted] ____________ Today's Date: [redacted] ________ (mm/dd/yyyy)

Student / Scholar Attestation:

"I hereby confirm that the information indicated in this statement is true to the best of my knowledge and that I will have the funds stated above during my UCLA program. I understand that my program eligibility at UCLA may be jeopardized if any information indicated here is found to be incomplete or false. I will notify UCLA immediately if there are any changes to my financial situation."

Signature of Student / Scholar: ____ [redacted] __ [redacted] ________ Today's Date: [redacted] ________ (mm/dd/yyyy)

Figure 7: The final redacted PDF. This method of redaction removes the sensitive text and adds additional black boxes covering it's previous location, indicating that there was a redaction performed."

be exposed, cached, or logged outside the user's control. Although these services often provide assurances of privacy, such guarantees remain outside the scope of our system.

Internally, we mitigated this issue by restricting the amount of data passed at any one time and ensuring that sensitive outputs are only stored temporarily within the processing pipeline. However, this does not fully eliminate the concern. In contrast, a fully self-hosted language model deployment would provide greater assurances of data sovereignty, since no external communication would be required. The trade-off, however, is that such models are computationally expensive and difficult to maintain, which places them outside the scope of this project. For these reasons, security constraints remain a key limitation in the current implementation, even though we adopted a cautious design to minimize exposure.

5.3 Reliance on Language Model

Here, we refer to dependence on language models as the extent to which the success of the system is tied to the capabilities and limitations of large language models. The pipeline relies on these models to identify sensitive information, interpret context, and generate appropriate redactions. While modern models demonstrate strong generalization, they are not immune to issues such as hallucinations, inconsistent reasoning, or failure to follow strict formatting instructions. These shortcomings directly affect the accuracy and reliability of the system.

Internally, we attempted to reduce this dependence by constraining the model's role to specific subtasks and by designing prompts that explicitly enforce structure. However, this does not eliminate the underlying reliance. If the model misinterprets the prompt or generates non-deterministic outputs, the downstream processes cannot fully recover. In contrast, rule-based approaches or hybrid systems could provide stronger guarantees for consistency, but they often lack the flexibility required to handle diverse real-world inputs. For these reasons, dependence on language models remains a core limitation of the current system, even though careful design choices helped mitigate its impact.

6 Findings, Conclusions, Implications, and Future Work

6.1 Findings

The results of the implementation phase confirm that the proposed agent-based DLP frameworks successfully addressed the central research problem: enabling large language model (LLM) agents to perform dynamic, context-aware data sanitization while maintaining human oversight. The findings align closely with the subproblems identified in Chapter 1 and are summarized below.

Addressing the Trigger Mechanism

The system effectively detected when to initiate sanitization processes without requiring manual intervention. Automated monitoring of input sources (e.g., uploaded documents, draft emails) reliably activated the orchestrator. This supports the feasibility of embedding an autonomous trigger mechanism within an agentic framework, thereby reducing reliance on human-initiated actions.

Executing Core Logic and Contextual Redaction

The LLM agents demonstrated the ability to:

- Parse unstructured text for contextually sensitive elements rather than relying solely on predefined patterns.
- Apply redaction decisions that balanced the need for data protection with the preservation of document utility.
- Present redaction proposals to the user in a clear, reviewable format.

This outcome validates the hypothesis that context-aware agentic reasoning can outperform static rule-based filtering in handling varied and nuanced content.

Coordinating Orchestration and Iteration

The orchestrator maintained control over the sequence of detection, redaction, and review stages. When user feedback indicated missed or incorrect redactions, the system was able to re-trigger the relevant modules and incorporate the changes without disrupting workflow. This iterative loop ensured continuous alignment with user expectations and improved the final output quality.

Integration of Human-in-the-Loop Review

The human-in-the-loop component proved central to ensuring trust and accuracy. In practice, users engaged with the preview stage to validate or override the agent's decisions. This process not only prevented over-redaction or under-redaction but also increased user confidence in the automated system.

Security Outcomes

The framework's secure design—incorporating data encryption, controlled tool calls, and isolated processing—successfully prevented any leakage of sensitive content during the sanitization process. This confirmed that the integration of third-party LLM APIs can be done without exposing private data, provided that strict security protocols are observed.

Unexpected Findings

While the primary focus was on textual data sanitization, an incidental observation was that the framework's modular structure naturally lends itself to multimodal extension. The same orchestration and validation pipeline could, with minimal adjustments, be adapted for image or structured data sanitization. Although outside the study's original scope, this finding suggests a broader applicability of the design principles.

Summary

Overall, the findings demonstrate that the proposed frameworks met their intended objectives, solved the subproblems outlined in the problem formulation, and operated effectively under test conditions. The results reinforce the potential of LLM-based agents to enhance traditional DLP processes by introducing adaptable, context-sensitive decision-making paired with human validation.

6.2 Implications

The successful implementation of agent-based DLP frameworks carries important implications at multiple levels: organizational, academic, and societal. These implications extend the relevance of the study beyond its immediate scope and highlight the broader value of integrating agentic AI into sensitive data management workflows.

1. Organizational Implications

For organizations, the framework provides a pathway toward scalable and intelligent data protection solutions. Traditional DLP systems often rely on static, pattern-based methods that can generate both false positives and false negatives. By contrast, the agentic approach introduces context-aware reasoning and iterative validation, which reduces error rates and lowers the manual burden on security teams.

- **Operational Efficiency:** Automation of initial detection and redaction accelerates the sanitization process, enabling staff to focus on higher-level decision-making rather than repetitive review tasks.
- **Regulatory Compliance:** The integration of human-in-the-loop oversight ensures that redactions remain transparent and auditable, supporting adherence to privacy regulations such as GDPR, HIPAA, and sector-specific compliance standards.
- **Risk Mitigation:** By reducing the likelihood of sensitive data leakage, organizations strengthen trust with clients, partners, and regulators while minimizing reputational and financial risks.

2. Implications for the Field of AI and Data Security

The study contributes to the evolving body of research on agentic AI by demonstrating a novel application in data loss prevention.

- **Validation of Agentic Design:** The results confirm that orchestrator-driven workflows and human-in-the-loop validation are effective design strategies for sensitive information tasks, offering a template for future AI-security integrations.
- **Interdisciplinary Relevance:** The framework bridges concepts from reinforcement learning, natural language processing, and cybersecurity, suggesting that hybrid approaches can better address complex challenges than siloed methods.
- **Research Momentum:** These findings may stimulate further investigation into agent-based frameworks for adjacent domains, including document classification, compliance auditing, and real-time risk assessment.

3. Societal and Ethical Implications

Beyond organizational and technical domains, the adoption of agentic DLP systems has wider societal significance.

- **Data Privacy Protection:** As digital communication continues to expand across personal, professional, and governmental contexts, improved sanitization frameworks play a crucial role in protecting individuals from identity theft, fraud, and unauthorized surveillance.
- **Transparency and Trust:** The explicit preview-and-approval stage empowers end-users, promoting transparency in AI decision-making and helping to counter skepticism toward automated systems.
- **Ethical Responsibility:** By enforcing strong safeguards against data leakage, these frameworks contribute to responsible AI deployment, aligning technical innovation with ethical stewardship of sensitive information.

Summary

The implications of this project underscore its broader significance: organizations gain a more reliable and efficient tool for safeguarding sensitive information, the AI and cybersecurity fields benefit from a novel application of agentic frameworks, and society at large is better positioned to manage risks associated with the increasing digitization of personal and organizational data. Collectively, these outcomes demonstrate the importance of embedding context-aware, human-centered AI systems into critical areas of data governance.

6.3 Conclusion

This implementation features a pipeline that operates as a sophisticated, agent-driven workflow driven by a central controller that manages the entire data sanitation life cycle. The process begins when a user uploads a document or writes an email, prompting the orchestrator to first parse the request and if needed, use a searcher to improve its understanding with the sanitation sensitivity. The detector module follows this and generates a preview of proposed changes, which the user is then able to review for an in-the-loop validation step. This iterative feedback loop is where the only part where user input is required, and can allow for further adjustments to the sanitation until the final redaction is approved and executed.

6.4 Future Work

While the project achieved its primary objectives, several avenues remain for advancing the capabilities and applicability of the developed agent-based DLP frameworks. These opportunities build upon the limitations noted in the study and the unexpected findings observed during implementation. Data loss prevention solutions are important for protecting sensitive information from unauthorized access and leaks.[11] As cyber threats continue to evolve, traditional DLP methods often fail to keep pace, necessitating advanced approaches that can handle the complexities of modern data environments.

1. Multimodal Expansion

Although the present work focuses on textual data, the modular pipeline and orchestrator design could be extended to support multimodal content, including images, scanned documents, audio transcripts, and video captions. This would require the integration of optical character recognition (OCR), speech-to-text engines, and visual recognition models capable of detecting sensitive elements in non-textual formats.

2. Adaptive Learning from User Feedback

Currently, human-in-the-loop review is essential for validating redactions, but the system does not adapt its decision-making based on prior interactions. Incorporating reinforcement learning or continual fine-tuning mechanisms could enable the agent to learn from repeated patterns in user feedback, thereby improving accuracy and reducing the frequency of manual corrections over time. Techniques like federated learning could allow for massively trained and even personalized models by distributing training over many distributed machines [16].

3. Expanded Evaluation Across Domains

The framework was tested in controlled scenarios using synthesized datasets representative of typical enterprise content. Future evaluations should apply the system to a wider range of real-world datasets across industries such as healthcare, finance, legal services, and government communications. This would assess both generalizability and domain-specific sensitivity detection requirements [20].

4. Integration with Enterprise Systems

To increase practical adoption, the framework could be integrated directly into enterprise DLP infrastructures, content management systems, and collaboration tools (e.g., email servers, cloud document platforms, messaging services). Real-time monitoring and inline sanitization could further enhance usability, enabling sensitive content filtering before it leaves a secure environment.

5. Enhanced Adversarial Robustness

While the system demonstrated secure handling of data, future iterations should undergo systematic adversarial testing. This includes evaluating its resilience to prompt injection attacks, deliberate obfuscation of sensitive information, and adversarial examples designed to evade detection. Incorporating adversarial training techniques could strengthen robustness against emerging threats.

6. Cross-Language Support

Expanding beyond English to support multilingual sanitation is an important future direction, especially for organizations operating in diverse linguistic environments. This would involve adapting tokenization, sensitivity detection, and contextual reasoning for languages with different syntactic and semantic structures. Though robust implementations like these would lead to the most secure data loss prevention systems, the models that we currently use have multilingual capabilities as shown in our testing, and a full multilingual system could easily adapt to our current data pipeline.

7. Long-Term Maintainability and Governance

Future deployments should include research into the governance structures necessary to ensure that the agent's redaction policies remain aligned with evolving privacy regulations and organizational compliance standards. Developers can also have an easier time implementing new features, due to employing AI assistants to their codebases. [7] This includes developing audit trails, explainability mechanisms for redaction decisions, and role-based access controls for sensitive datasets.

8. Hybrid Methods

Hybrid methods could use LLMs in tandem with traditional find-and-replace methods to supercharge the entire framework. For example, with large PDFs, it may be very costly to use LLMs, and the text might not fit in the context window of any given model. Equipping the LLM with more robust regex capabilities would allow the model to find certain sensitive strings and decide if they need to be redacted on a case-by-case basis. [3] An example of this would be searching for all the phone numbers in a document, and redacting only private phone numbers, not public ones. This idea of private/public information would be difficult for traditional methods to integrate, but LLMs would have the contextual understanding necessary to make the executive redaction decisions.

Summary

By pursuing these extensions, the framework could evolve from a proof-of-concept into a versatile, production-ready DLP solution capable of operating across modalities, domains, and organizational environments. These directions not only enhance technical performance but also strengthen trust, compliance, and security in AI-driven data protection workflows.

7 Acknowledgements

We would like to Matt Wu and Jeremy Wei of iDox.ai for sponsoring this project and providing guidance and support throughout the process. We would also like to thank Bruce Huang and Joselito Lao of the Univeristy of California, Los Angeles for supporting the Master of Engineering Program students and for organizing the capstone projects.

References

[1] Virginia Aglietti, Ira Ktena, Jessica Schrouff, Eleni Sgouritsa, Francisco J. R. Ruiz, Alan Malek, Alexis Bellot, and Silvia Chiappa. Funbo: Discovering acquisition functions for bayesian optimization with funsearch. *Google DeepMind*, 2024.

[2] Rohan Ajwani, Shashidhar Reddy Javaji, Frank Rudzicz, and Zining Zhu. Llm-generated black-box explanations can be adversarially helpful. *The 2nd Workshop on Regulatable ML @ NeurIPS*, 2025.

[3] Jordan Allred, Sadaira Packer, Gerry Dozier, Sarp Aykent, Alexicia Richardson, and Michael C. King. Towards a human-ai hybrid for adversarial authorship. In *2020 SoutheastCon*, pages 1–8, 2020.

[4] Amazon AI Research. The empirical impact of data sanitization on language models. *Amazon Science Publications*, 2024.

[5] Anonymous. Trism for agentic ai: A review of trust, risk, and security management. *arXiv preprint arXiv:2506.04133*, 2025.

[6] Jayachandu Bandlamudi, Ritwik Chaudhuri, Neelamadhav Gantayat, Kushal Mukherjee, Prerna Agarwal, Renuka Sindhgatta, and Sameep Mehta. A framework for testing and adapting rest apis as llm tools, 2025.

[7] Markus Borg, Dave Hewett, Donald Graham, Noric Couderc, Emma Söderberg, Luke Church, and Dave Farley. Does co-development with ai assistants lead to more maintainable code? a registered report, 2024.

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *OpenAI*, 2020.

[9] Peter Christen, David J. Hand, and Nishadi Kirielle. A review of the f-measure: Its history, properties, criticism, and alternatives. *ACM Comput. Surv.*, 56(3), October 2023.

[10] Sankarshan Damlea and Boi Faltings. Llms for resource allocation: A participatory budgeting approach to inferring preferences. *Artificial Intelligence Laboratory (LIA), EPFL ORCID*, 2025.

[11] Dorcas Esther. Ai in data loss prevention: Safeguarding sensitive data against unauthorized access and leakage. 06 2024.

[12] Lei Huang et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 2024.

[13] Yuta Hasegawa. *LLM-Assisted Detecting and Redacting Confidential Information*. PhD thesis, Virginia Polytechnic Institute and State University, 2025.

[14] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, Rui Kong, Yile Wang, Hanfei Geng, Jian Luan, Xuefeng Jin, Zilong Ye, Guanjing Xiong, Fan Zhang, Xiang Li, Mengwei Xu, Zhijun Li, Peng Li, Yang Liu, Ya-Qin Zhang, and Yunxin Liu. Personal llm agents: Insights and survey about the capability, efficiency and security, 2024.

[15] Zhijie Liu, Yutian Tang, Xiapu Luo, Yuming Zhou, and Liang Feng Zhang. No need to lift a finger anymore? assessing the quality of code generation by chatgpt. *IEEE Transactions on Software Engineering*, 50(6):1548–1584, 2024.

[16] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2023.

[17] Alexander Novikov, Ngân Vu, Marvin Eisenberger, Emilien Dupont, Adam Zsolt Wagner Po-Sen Huang, Sergey Shirobokov, Borislav Kozlovskii, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery. *Google Deepmind Blog*, 2025.

[18] Hans-Jürgen Profitlich and Daniel Sonntag. A case study on pros and cons of regular expression detection and dependency parsing for negation extraction from german medical documents. technical report. *CoRR*, abs/2105.09702, 2021.

[19] Tim R. Davidson Veniamin Veselovsky Michal Kosinski Robert West1. Evaluating language model agency through negotiations. *ICLR*, 2024.

[20] Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. Survey on evaluation of llm-based agents. *arXiv:2503.16416*, 2025.