Information & Computer Science Department, King Fahd University of Petroleum and Minerals

COE 426: Data Privacy

Project Final Report

PrivPrompt: Private AI Session Firewall

December 6, 2025

ID #	Name
[redacted]	ALI MASHNI
[redacted]	TURKI ALGETHAMI
[redacted]	ABDULAZIZ HAKAMI
[redacted]	WALEED ALRAJHI
[redacted]	MOHAMMED ALAMRI

Introduction

LLM chat UIs routinely mix user text with files and images; this creates a high-impact privacy risk because sensitive data can slip through either in captions, free-text, or JSON that declares/embeds non-text (e.g., type: "input_image", image_url, data:image/...;base64, ...). Our project tackles this problem with a **privacy-aware browser extension + proxy** that inspects every ChatGPT request and enforces a mode-driven policy while keeping the user experience intact. Throughout the preparation and development phases, the system has successfully achieved all planned objectives. The proxy server reliably intercepts outgoing requests and, when necessary, sanitizes user messages by removing potential privacy threats before they reach the ChatGPT server. To give users flexible control, the accompanying browser extension offers three operational modes: Warn, Strict, and Blocked. Both Warn and Strict modes apply the same privacy-manipulation mechanisms, with the distinction that Strict mode prevents the transmission of images and files. Blocked mode provides the highest level of protection by rejecting any request that contains detected privacy risks.

In addition to real-time message filtering, the system includes a comprehensive dashboard that processes request logs to generate meaningful analytics. These visualizations include detection statistics by type, latency measurements, individual request histories, status codes, and detailed records of privacy concerns identified by the algorithms. The sections that follow provide demonstrations, evidence, and screenshots that illustrate the functionality and effectiveness of the system.

Why this is challenging? We must:

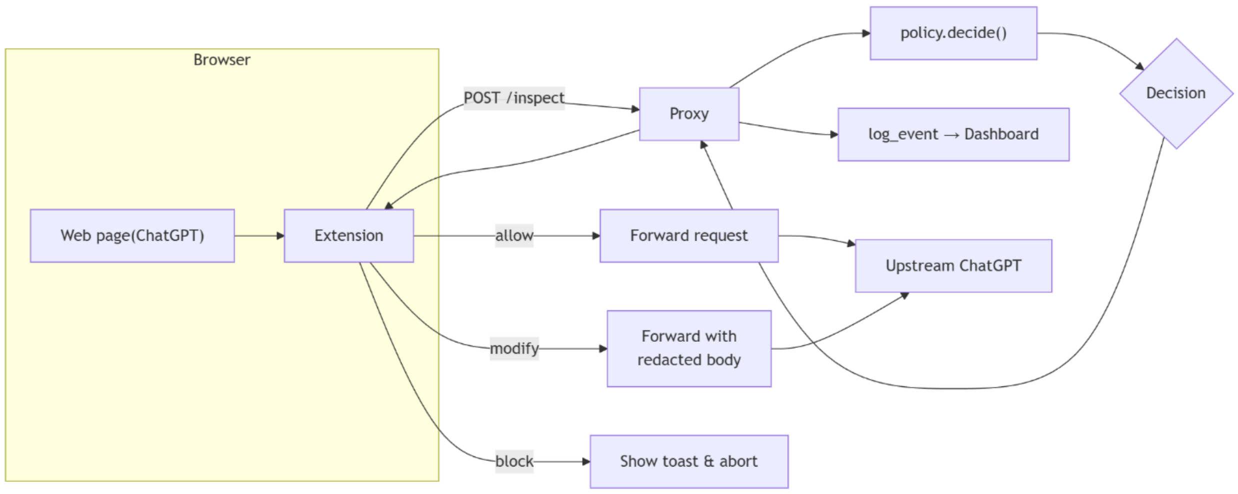
(1) differentiate plain text from non-text even when assets are only declared in JSON

(2) redact only user-text fields without corrupting structural fields (e.g., file_id, image_url)

(3) avoid touching base64/data-URLs and tokens (binary-like guards)

(4) preserve responsiveness (single JSON walk; linear time).

Project Design

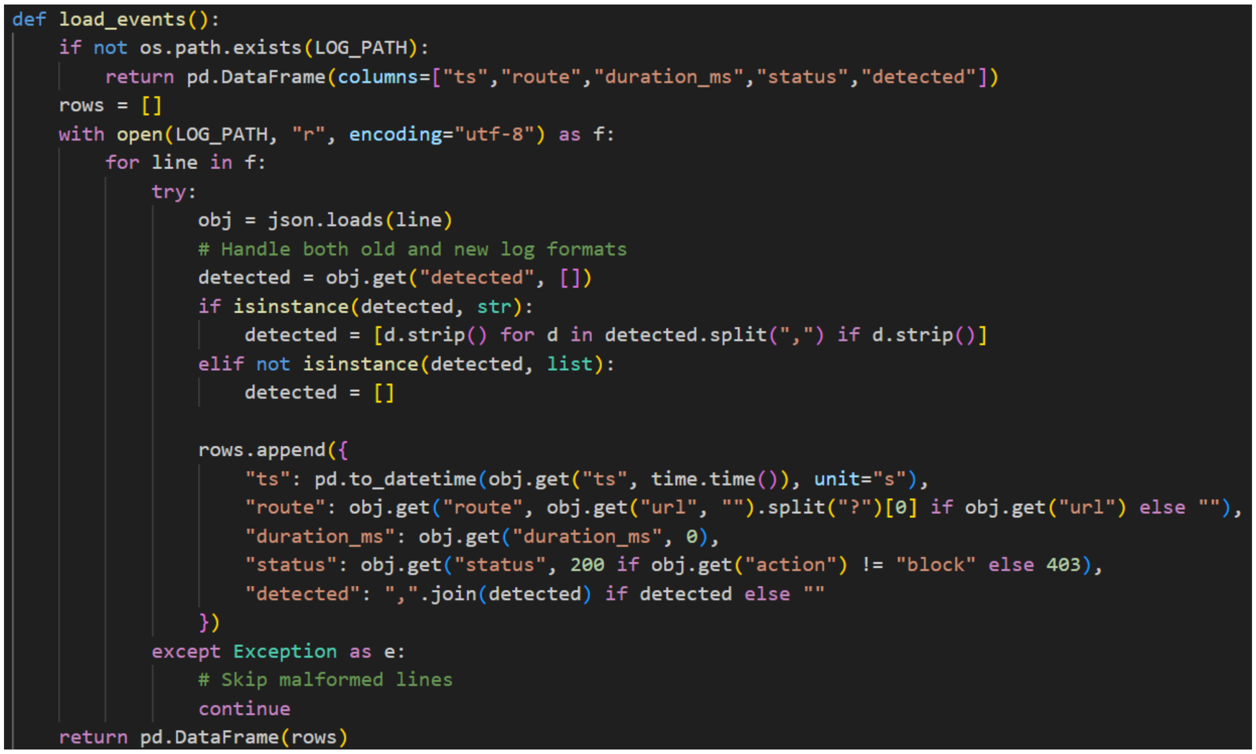


The flow of our project starts from the browser, where we specify ChatGPT as the website we want to operate on. Assuming the user has enabled the extension, all requests will first go to the extension, which will then forward them to our proxy for inspection. In the proxy, all algorithms are executed: detection, transformation, and the decision-making process for the request. Based on these steps, a decision is produced by our policy to determine the next action taken by the proxy.

After the proxy generates the new request, whether unchanged or modified, and receives the policy's decision, it sends the result back to the extension. The extension will then either forward the request without changes, modify the request because the proxy detected privacy concerns, or block the request entirely. The proxy mode explained in the introduction plays a major role in the decision policy, since depending on the selected mode, a message may be modified or blocked.

Implementation

Our project consists of three main folders: dashboard, extension, and proxy. The organization of these folders facilitates the implementation of the project, as each folder contains code related to a specific task. The dashboard is implemented using Python and presents the necessary data for future analytics and algorithm improvements. The extension is implemented using JavaScript. The proxy is implemented using Python and is divided into multiple components such as detectors, transformers, and policy. In the following screenshots, the most important parts of our code are shared for illustration. **For the full code, you can check it from here: https://github.com/Ali-Mashni/PrivPrompt.**

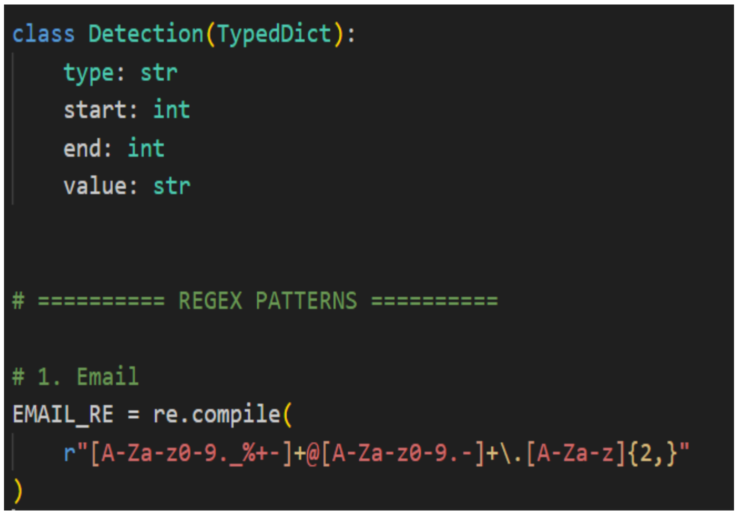


```python
def load_events():
    if not os.path.exists(LOG_PATH):
        return pd.DataFrame(columns=["ts","route","duration_ms","status","detected"])
    rows = []
    with open(LOG_PATH, "r", encoding="utf-8") as f:
        for line in f:
            try:
                obj = json.loads(line)
                # Handle both old and new log formats
                detected = obj.get("detected", [])
                if isinstance(detected, str):
                    detected = [d.strip() for d in detected.split(",") if d.strip()]
                elif not isinstance(detected, list):
                    detected = []

                rows.append({
                    "ts": pd.to_datetime(obj.get("ts", time.time()), unit="s"),
                    "route": obj.get("route", obj.get("url", "").split("?")[0] if obj.get("url") else ""),
                    "duration_ms": obj.get("duration_ms", 0),
                    "status": obj.get("status", 200 if obj.get("action") != "block" else 403),
                    "detected": ",".join(detected) if detected else ""
                })
            except Exception as e:
                # Skip malformed lines
                continue
    return pd.DataFrame(rows)
```

Figure 1 Dashboard Table Implementation

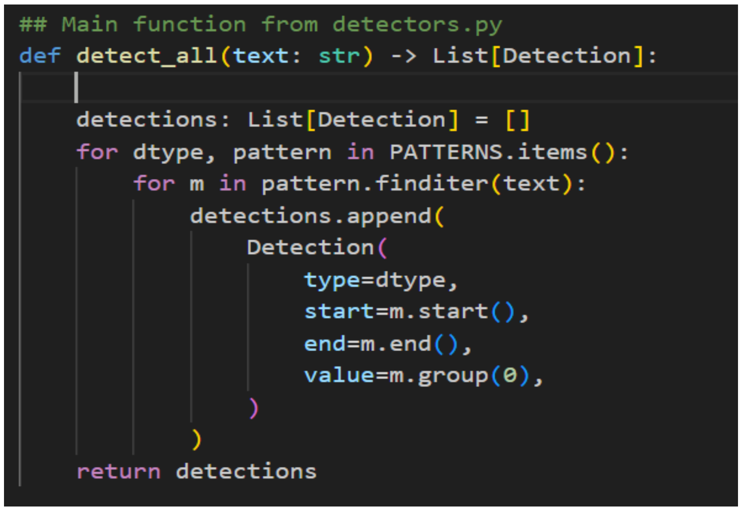
As shown in Figure 1, the implementation displays a table that contains all requests made by the user, along with additional features helpful for analysis and improvements, such as the status code, latency time, and all detected types in each request.


```python
class Detection(TypedDict):
    type: str
    start: int
    end: int
    value: str


# ========== REGEX PATTERNS ==========

# 1. Email
EMAIL_RE = re.compile(
    r"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}"
)
```

Figure 3 Detection Class and Regex Patterns


```python
## Main function from detectors.py
def detect_all(text: str) -> List[Detection]:

    detections: List[Detection] = []
    for dtype, pattern in PATTERNS.items():
        for m in pattern.finditer(text):
            detections.append(
                Detection(
                    type=dtype,
                    start=m.start(),
                    end=m.end(),
                    value=m.group(0),
                )
            )
    return detections
```

Figure 2 Main Detection Algorithm

As shown in Figures 2 and 3, these components are mainly used for the detection of privacy concerns in the project. We used regular expressions for different privacy concerns such as emails, phone numbers, company names, IP addresses, etc. These regular expressions are used in our main detection functions, which use the Detection class to create a list containing Detection elements. Each Detection element includes its type, whether email, phone number, or another category, its start index in the request message, its end index, and the value of the detected privacy concern. This implementation is used in the transformation code.

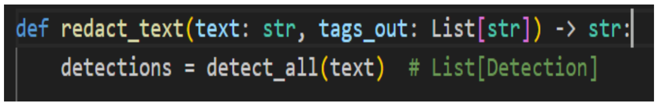

```python
def redact_text(text: str, tags_out: List[str]) -> str:
    detections = detect_all(text)  # List[Detection]
```

Figure 4 Redact function

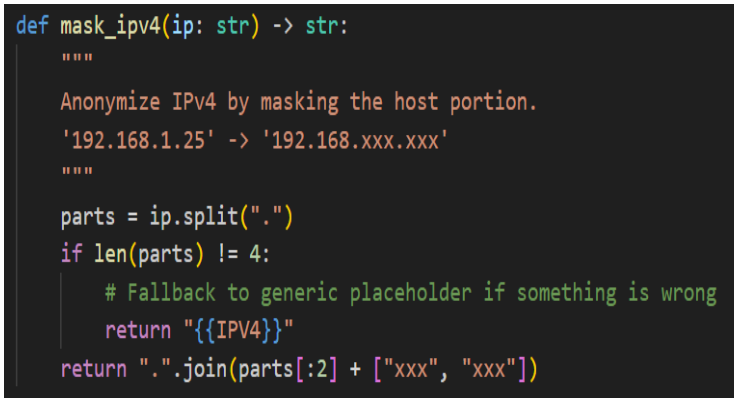

```python
def mask_ipv4(ip: str) -> str:
    """
    Anonymize IPv4 by masking the host portion.
    '192.168.1.25' -> '192.168.xxx.xxx'
    """
    parts = ip.split(".")
    if len(parts) != 4:
        # Fallback to generic placeholder if something is wrong
        return "{{IPV4}}"
    return ".".join(parts[:2] + ["xxx", "xxx"])
```

Figure 5 Transformation Example

As shown in Figures 4 and 5, these components are used for the transformation of the text using the detect_all algorithm from the detectors section. The transformation technique applies a specific transformation for each detected type, where the detected content is either partially or fully masked depending on its category. The redact_text function is not fully shown in the screenshot because it is long, but it uses all the transformation functions, such as the mask IP address function shown in Figure 5, to modify the text from the request when needed.

Application of Privacy Concepts

Privacy by design & default. The proxy enforces safe defaults. In *strict/block* modes, any non-text or JSON that declares/embeds non-text (images, PDFs, archives, audio/video, multipart) is blocked by default. Text is only forwarded after running the redaction pipeline.

Data minimization. We never persist request bodies. Logs store only what's necessary for oversight: timestamp, route, action, and the **types** of PII detected (e.g., ["email","phone"]) ,never the raw values.

Purpose limitation. Inspection is used solely to decide **allow / modify / block** and to sanitize text. No profiling or secondary analytics. Policy decisions are centralized in policy.decide(); the app just orchestrates.

Pseudonymization. Sensitive strings are transformed before leaving the browser:

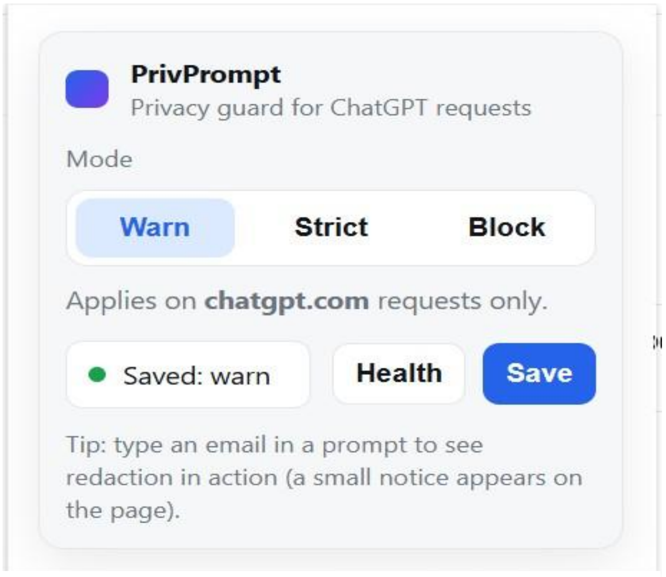
- Emails/JWTs/companies → placeholders {{EMAIL}}, {{JWT}}, {{COMPANY}}.
- Phones/API keys/National IDs → keep last 4 characters only.
- IPv4 → mask host portion (192.168.xxx.xxx).
  This preserves utility while severing direct identifiability.

Least privilege. Only the policy module makes decisions. Redaction operates **only** on likely user-text fields (text, content, prompt, etc.). Structural/transport fields (e.g., file_id, image_url, mime_type) are never altered.

Results

In this section we will provide screenshots for illustrations that's hard to be illustrate using just our words. Additionally, test cases will be shown.

As shown in Figure 6, the interface displays the modes that the user can select, along with the status of the proxy server below. It will indicate that the proxy is not running if a problem occurs.



PrivPrompt

Privacy guard for ChatGPT requests

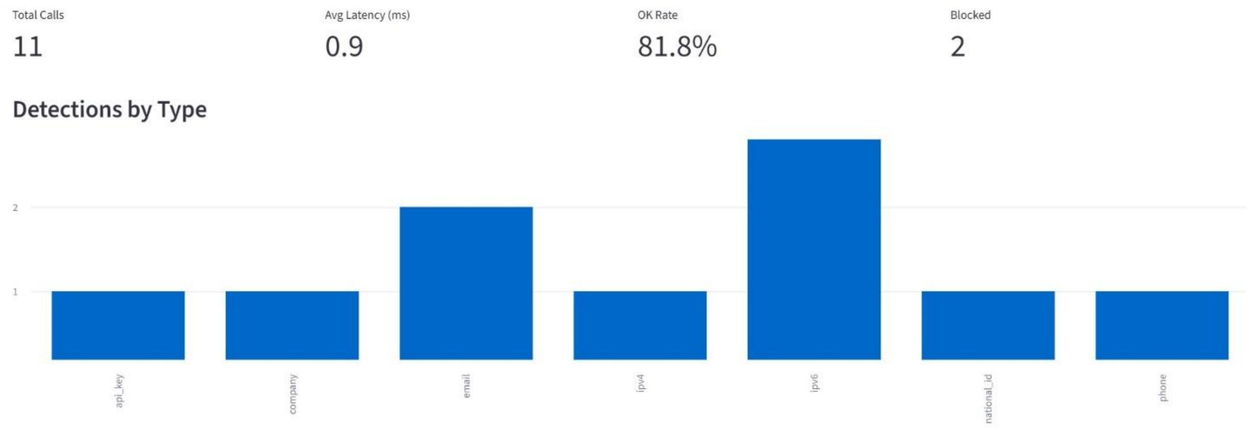
Mode

Applies on **chatgpt.com** requests only.

● Saved: warn

Tip: type an email in a prompt to see redaction in action (a small notice appears on the page).

Figure 6 Proxy Modes



Total Calls	Avg Latency (ms)	OK Rate	Blocked
11	0.9	81.8%	2

Figure 7 Dashboard Statistics

As shown in Figure 7, this is displayed in the dashboard and presents statistics that can be used for analysis and future enhancements, particularly in determining which privacy concerns require more attention. For example, if a certain type shows many detections, it may need stronger handling, or if it appears frequently, it may require less manipulation to increase system utility. This analysis can be somewhat subjective, as there are many possible interpretations.

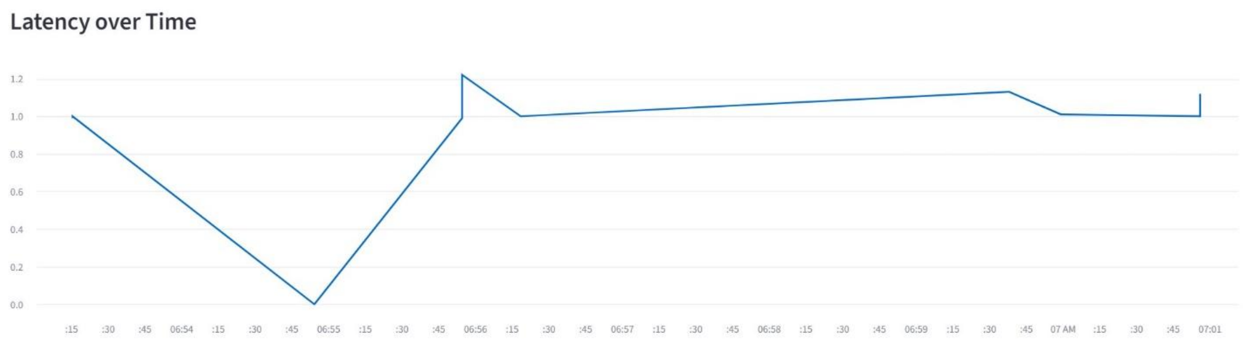
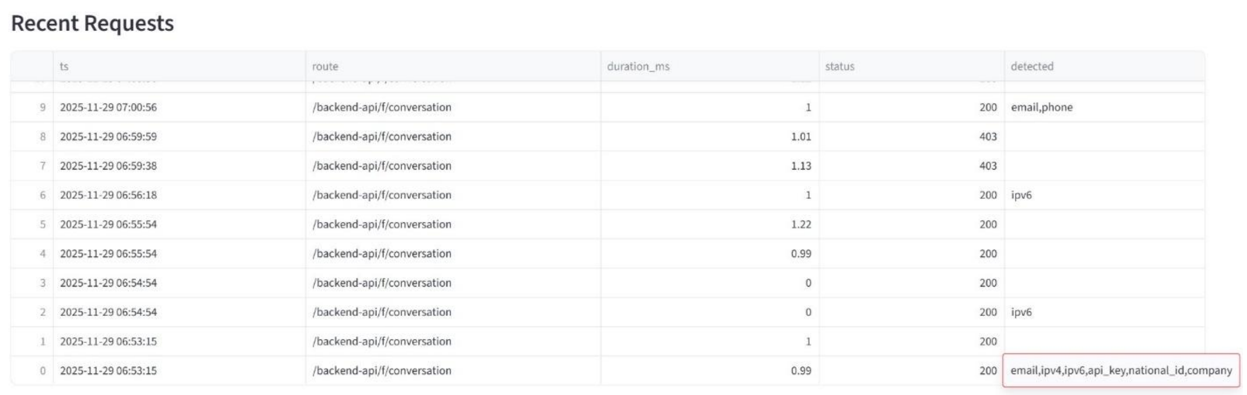


Figure 8 Dashboard Latency Graph

In Figure 8, a latency graph is shown for all requests that have been forwarded and manipulated. This can be used to study the performance of detection and transformation algorithms. For illustration, it shows how much a particular message was delayed while being detected, transformed, and transmitted again to ChatGPT. There is also an average latency of all requests displayed in the statistics in Figure 7.

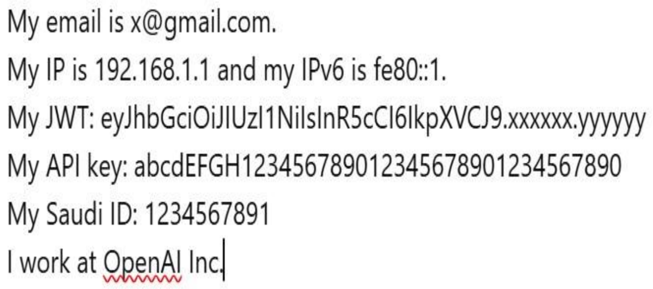


Recent Requests

	ts	route	duration_ms	status	detected
9	2025-11-29 07:00:56	/backend-api/f/conversation	1	200	email,phone
8	2025-11-29 06:59:59	/backend-api/f/conversation	1.01	403	
7	2025-11-29 06:59:38	/backend-api/f/conversation	1.13	403	
6	2025-11-29 06:56:18	/backend-api/f/conversation	1	200	ipv6
5	2025-11-29 06:55:54	/backend-api/f/conversation	1.22	200	
4	2025-11-29 06:55:54	/backend-api/f/conversation	0.99	200	
3	2025-11-29 06:54:54	/backend-api/f/conversation	0	200	
2	2025-11-29 06:54:54	/backend-api/f/conversation	0	200	ipv6
1	2025-11-29 06:53:15	/backend-api/f/conversation	1	200	
0	2025-11-29 06:53:15	/backend-api/f/conversation	0.99	200	email,ipv4,ipv6,api_key,national_id,company

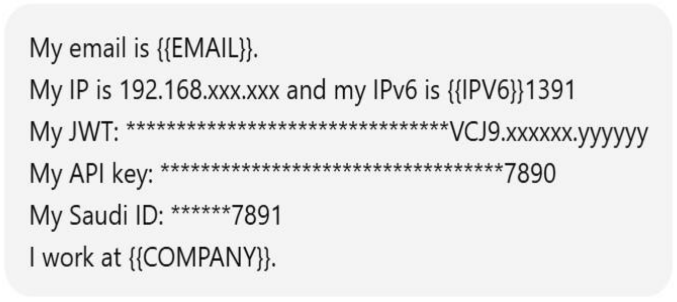
Figure 9 Dashboard Data Visualization as a Table

The last component of the dashboard is shown in Figure 9, which displays all the requests with more detailed information. It shows all the detections for a particular request, its latency time, its status code, and the time it was sent. This table can be exported for further analysis and enhancements, and it was helpful during the testing phase.



```
My email is x@gmail.com.
My IP is 192.168.1.1 and my IPv6 is fe80::1.
My JWT: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.xxxxxx.yyyyyy
My API key: abcdEFGH123456789012345678901234567890
My Saudi ID: 1234567891
I work at OpenAI Inc.
```

Figure 10 Test Case (Before Manipulation)



```
My email is {{EMAIL}}.
My IP is 192.168.xxx.xxx and my IPv6 is {{IPV6}}1391
My JWT: ********************************VCJ9.xxxxxx.yyyyyy
My API key: **********************************7890
My Saudi ID: ******7891
I work at {{COMPANY}}.
```

Figure 11 Test Case (After Manipulation)

As shown in Figures 10 and 11, test cases are a major part of the project, where we review and check the completeness and correctness of the implementation. The test case shown includes a combination of different privacy concerns that were sent, along with their transformed form after being forwarded by our proxy server. Manipulation mainly involves masking or some type of replacement, which protects user privacy.

Discussion

Our discussion is mainly about the privacy and utility of the project, since these aspects are the major focus of our course. In terms of privacy and utility measurements, there is no single metric or standardized way we can fully rely on for evaluation. However, we can discuss different points that provide insight into these aspects. One of the features we have is latency measurement, which can be linked to the utility of our system, if the latency increases, the utility decreases. Conversely, privacy depends on the number of privacy concerns we have regular expressions for. The more regular expressions we include in our project, the more privacy we can achieve.

Many might ask whether adding more regular expressions will reduce the system's utility. Under normal usage, it could reduce utility, but the user can retain some utility by disabling the extension after receiving the modified message. By doing this, the user can keep whatever private data masking they consider necessary and rewrite anything they are comfortable keeping visible. In this scenario, utility may still be affected if ChatGPT's response depends on a privacy concern the user could not keep visible.

Additionally, in our implementation, we took the typing ping feature into consideration, where ChatGPT can detect what the user is typing in real time before the request is sent. Keeping this feature enabled would lead to zero privacy for our project, so we disabled it. As a tradeoff, disabling typing pings prevents ChatGPT from providing suggestions while the user is writing their request.

Challenges Faced

Overall, the implementation process was smooth, especially since most of the project was developed in Python. All team members are familiar with Python, which made debugging and resolving errors relatively easy. The main challenge arose during the development of the browser extension, as it involved new concepts and tools that we had not used before. Despite this, it was a valuable experience that strengthened our knowledge and deepened our understanding of the field by allowing us to overcome unfamiliar technical obstacles.

GenAI Statement: All the work in this project was completed by the team. Generative AI was only used for code assistance, check grammar and improve clarity of writing.