International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)
https://ijcjournal.org/index.php/InternationalJournalOfComputer/index

Safe Observability: A Framework for Automated PII Redaction from LLM Prompts in OpenTelemetry Pipelines

Serhii Melnyk*

NAVEX, Charlotte, NC, USA
Email: 4melnyk@gmail.com

Abstract

The proliferation of Large Language Models (LLMs) within enterprise applications has introduced a critical conflict between the goals of modern observability and the mandates of data privacy. While observability platforms provide essential, deep visibility into complex distributed systems, they inadvertently become repositories for Personally Identifiable Information (PII) when they ingest the unstructured, information-rich prompts users submit to LLMs. This leakage of sensitive data into telemetry pipelines constitutes a significant security liability and a compliance risk under regulations such as GDPR and CCPA. This paper introduces the concept of "Safe Observability," a paradigm that reconciles the need for comprehensive system insight with robust privacy protection. I propose a novel framework for achieving this through the automated redaction of PII within the OpenTelemetry (OTel) ecosystem. The core of this framework is a custom, configurable PII-Redaction Processor for the OpenTelemetry Collector, designed to act as a strategic control plane for sanitizing telemetry data in-transit. The architecture employs a hybrid PII detection methodology, combining the speed of regular expressions with the contextual accuracy of Named Entity Recognition (NER) models, implemented as a decoupled microservice. This paper details the architectural design, provides a comprehensive implementation guide for building and deploying the custom processor using the OpenTelemetry Collector Builder (OCB) and Go, and presents a rigorous evaluation of its efficacy and performance impact. The findings demonstrate that this approach offers a practically viable and architecturally sound solution for preventing PII leakage, enabling organizations to leverage the power of observability and LLMs without compromising user privacy or regulatory compliance.

Keywords: Large Language Models; OpenTelemetry; Safe Observability; PII-Redaction.

Received: 10/9/2025
Accepted: 12/9/2025
Published: 12/19/2025

----

* *Corresponding author.*

1. The Imperative for Safe Observability

Modern engineering has shifted from traditional monitoring to observability, defined as the ability to infer a system's internal states from its external outputs — namely, its telemetry data (metrics, events, logs, and traces). This allows teams to debug "unknown unknowns" in complex, distributed systems, which is critical for reliability and performance [1].

However, this deep visibility creates a new liability. The rich telemetry data required for observability can inadvertently capture and aggregate sensitive user data, particularly Personally Identifiable Information (PII) [2].

The integration of Large Language Models (LLMs) has magnified this risk. User interactions with LLMs occur through free-form, natural language prompts, which frequently contain sensitive data like names, addresses, financial details, or protected health information (PHI) [3]. Even if the LLM's response is secure, the user's input prompt is often logged in its entirety for debugging, auditing, or fine-tuning purposes [4]. This action copies sensitive data from a secure application environment into observability backends, which often have broader access controls, creating a high-value target for attackers and a compliance nightmare [4].

This practice is in direct conflict with data privacy regulations like the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA), which mandate principles of data minimization and purpose limitation [5].The inadvertent logging of PII can be considered a data breach, leading to severe financial and reputational damage [6].This necessitates a new paradigm: **Safe Observability**, which embeds privacy protection directly into the telemetry pipeline, ensuring that the act of observing a system does not introduce more risk than it mitigates.

1.2. Related Work

Prior research on observability and privacy has addressed PII leakage in logs and telemetry, but rarely integrates automated redaction into pipelines like OpenTelemetry. Mudryi, Chaklosh and Wójcik [7] highlights the risks of spilling PII to logs, advocating data minimization, yet focuses on manual audits rather than in-transit automation — a gap this framework fills by embedding redaction as a 'Telemetry Firewall'.

In PII detection, Gamage [8] explores NER for sanitizing LLM inputs, achieving high accuracy with *spaCy,* but without addressing performance in high-throughput systems. Similarly, Asthana and his colleagues [9] propose adaptive mitigation for LLMs, reporting 0.92 F1-scores, but rely on LLM APIs, introducing recursive privacy issues that this decoupled microservice avoids. Sadafal[10] discusses OTel-specific masking, but limits to regex, sacrificing recall compared to this hybrid approach.

Observability-focused works, like Risi[11] on OTel's Go performance, quantify instrumentation overhead (up to 20% CPU), aligning with our benchmarks but not extending to privacy processors. Nightfall [4] examines sensitive data in observability platforms, proposing backend scrubbing, which complements but does not prevent upstream leaks as this solution does.

Overall, while these studies provide foundational insights, they lack a comprehensive, vendor-agnostic framework for LLM prompt redaction in OTel pipelines, underscoring the novelty of this paper's contribution.

2. Architectural Foundation: The OpenTelemetry Collector

The OpenTelemetry (OTel) framework provides a standardized, vendor-agnostic way to generate, collect, and export telemetry data [12]. (which currently stands as the dominant vendor-neutral standard in the industry). At the heart of this framework is the OpenTelemetry Collector, a standalone proxy that acts as a flexible data processing pipeline [13] .A Collector pipeline consists of three main components [14]:

- **Receivers**: Entry points that ingest data from applications in various formats (e.g., OTLP, Jaeger).
- **Processors**: Components that transform or act upon data in-flight, such as batching, filtering, or adding attributes.
- **Exporters**: Exit points that send the processed data to one or more backend systems.

The processor component is the architecturally optimal point to implement automated PII redaction. Placing the logic here provides several advantages over embedding it in each application or redacting it post-ingestion:

1) **Centralization and Decoupling**: Redaction logic is managed in one place, ensuring consistent policy enforcement across all services without burdening application developers [10].
2) **Performance Offloading**: Computationally expensive PII detection is moved from the application to the dedicated Collector layer, minimizing performance impact on core business logic [15]. The main application stays fast because it isn't using its own CPU and memory to scan for PII. If the scanning becomes a bottleneck, you can add more copies of this service without having to scale the main application.
3) **Vendor and Language Agnostic**: A single processor can sanitize telemetry from any application sending OTel data, regardless of its programming language [14].

By leveraging a custom processor, the Collector is transformed from a simple data forwarder into an active, intelligent control plane for telemetry — a "Telemetry Firewall" that inspects and sanitizes data according to centrally managed privacy rules.

3. PII Detection Methodologies: A Hybrid Approach

The effectiveness of the redaction processor hinges on its PII detection engine. There is a fundamental trade-off between performance, accuracy, and cost when choosing a detection method.

- **Regular Expressions** (Regex): This technique is extremely fast and works well for structured PII with predictable formats, such as email addresses, phone numbers, and Social Security Numbers.18 However, Regex lacks contextual understanding, leading to high rates of both false positives and false negatives for unstructured data like names [9].

- **Named Entity Recognition** (NER): NER is a machine learning technique that identifies and classifies entities (e.g., PERSON, LOCATION) based on linguistic context [8]. It is far more accurate for unstructured PII than Regex. Open-source libraries like spaCy and Microsoft Presidio provide powerful, pre-trained NER models [16]. The main drawback is higher computational overhead compared to Regex Reference [17].
- **Large Language Models** (LLMs): Fine-tuned LLMs can achieve state-of-the-art PII detection accuracy, outperforming traditional NER models by leveraging their deep contextual understanding [9]. However, this approach has the highest computational cost and latency, making it less suitable for high-throughput, real-time pipelines. Using a third-party LLM API also creates a recursive privacy problem, as it requires sending unredacted data to an external service [18].

Given these trade-offs, a hybrid, tiered approach offers the most practical and effective solution. The proposed PII Detection Service first uses Regex as a fast-pass filter for well-structured PII. Any remaining text is then passed to a more accurate — but more computationally expensive — NER model for contextual analysis. This balances the competing demands of performance and accuracy. For a detailed review of these NER approaches, see **Appendix A**

4. System Design: The PII-Redaction Processor

The proposed solution uses a decoupled, microservice-based architecture to optimize for performance and maintainability.

1) **The PII redaction Processor** (Go):

A custom processor built using the Go programming language to integrate seamlessly with the OpenTelemetry Collector. Its sole responsibilities are to intercept telemetry, identify target attributes for scanning based on its configuration, make a gRPC call to the detection service, and apply the redaction policy to the data.

2) **The PII Detection Service** (Python):

A separate microservice that encapsulates the hybrid PII detection logic. Python is chosen for its mature ecosystem of NLP and machine learning libraries (e.g., spaCy, Presidio).

This architectural separation allows each component to be scaled and updated independently. The data flows as follows: an instrumented application sends a trace containing an LLM prompt to the Collector. The PII redaction processor intercepts this trace, extracts the prompt text, and sends it to the PII Detection Service via a gRPC call. The service analyzes the text and returns a list of detected PII entities. The Go processor then redacts this PII from the prompt string before forwarding the sanitized trace to the observability backend.

The processor is highly configurable via the Collector's config file, allowing operators to define parameters such as the gRPC endpoint of the detection service, a list of attributes to scan (e.g., llm.prompt), the redaction_policy (e.g., mask with * or replace with entity type like <PERSON>), and a failure policy (fail open or fail closed) in case the detection service is unavailable.

5. Implementation and Deployment

The solution is deployed by first building a custom OpenTelemetry Collector binary that includes the new PII redaction processor. This is accomplished using the OpenTelemetry Collector Builder (OCB), a command-line tool that automates the compilation process based on a manifest file [19].The manifest declaratively lists all the receivers, processors, and exporters to be included in the build, pointing to their Go module paths.The Go processor itself is developed by implementing the *processor.Traces* interface from the OTel framework [17].The core logic resides in the *ConsumeTraces* method, which iterates through spans, identifies attributes targeted for scanning, and calls the external Python service.

The Python service is built using **grpcio** to implement the gRPC server and a library like Microsoft Presidio to orchestrate the hybrid detection engine [16]. A Protobuf file defines the simple service contract for sending text and receiving a list of detected PII entities.

Finally, the custom Collector is configured and deployed. The *config.yaml* file defines the full pipeline, ensuring the *pii_redaction* processor is placed after the batch processor for efficiency but before any sampling processors to ensure all data is scanned.

6. Performance Evaluation and Benchmarking

A PII redaction solution is only viable if it is both effective and does not impose an unacceptable performance penalty. The framework was evaluated for both efficacy and overhead.

6.1. Efficacy Analysis

The accuracy of the PII Detection Service was evaluated against the Cleaned Repository of Annotated Personally Identifiable Information (**CRAPII**), a standard public benchmark dataset containing over 22,000 student essays with annotated PII [20]. Performance was measured using standard classification metrics:

- **Precision**: The percentage of identified PII that was correct (measures false positives).
- **Recall**: The percentage of actual PII that was successfully identified (measures false negatives). For PII detection, high recall is critical, as a single false negative represents a data leak.
- **F1-Score**: The harmonic mean of precision and recall.

6.2. Performance Overhead Analysis

The performance impact of the PII redaction processor was measured in a controlled load test. A sample application generated a constant load of 10,000 trace spans per second, which were sent to an OTel Collector. The Collector's resource consumption (CPU, memory) and pipeline latency were measured with and without the redaction processor enabled.

The results, summarized in **Table 1**, provide a quantitative picture of the trade-offs.

Table 1: Performance Benchmark Results

Collector Configuration	**Throughput (spans/sec)**	**CPU Overhead (%)**	**Memory Overhead (MB)**	**P99 Latency Increase (ms)**	**Precision**	**Recall**	**F1-Score**
PII Redaction (Regex only)	10,000	+5-8%	+10-15	+2-4ms	0.78	0.65	0.71
PII Redaction (Hybrid NER)	10,000	+30-40%	+100-150	+40-60ms	0.92	0.96	0.94

Note: Efficacy metrics are based on evaluation against the CRAPII dataset. Performance overhead metrics are relative to the Baseline Collector configuration under sustained load.

The efficacy results highlight the superiority of the hybrid NER approach, achieving a precision of 0.92, recall of 0.96, and F1-score of 0.94 on the CRAPII dataset. Precision measures the proportion of detected PII entities that are correctly identified, minimizing false positives that could unnecessarily obscure non-sensitive data and reduce telemetry utility. Recall, critically important in privacy contexts, quantifies the fraction of actual PII captured, where a value of 0.96 indicates that only 4% of sensitive information might escape redaction—a substantial improvement over the regex-only baseline's 0.65 recall, which risks leaking up to 35% of PII. The F1-score, as the harmonic mean, balances these metrics and confirms the hybrid method's robustness for unstructured LLM prompts, where contextual cues (e.g., linguistic patterns around names or locations) are essential.

Comparatively, these results align with state-of-the-art benchmarks in PII detection literature. For instance, Asthana and his colleagues [9] reported F1-scores of 0.92-0.95 for LLM-adapted NER models on similar essay-based datasets, but this hybrid approach achieves higher recall without fine-tuning on domain-specific data, demonstrating its generalizability to telemetry pipelines. The regex-only mode, while faster, exhibits lower efficacy due to its inability to handle contextual variations, such as misspelled names or embedded PII in sentences—issues mitigated by the NER tier.

On the performance side, the hybrid configuration introduces a 30-40% CPU overhead and 40-60ms P99 latency increase under a sustained 10,000 spans/sec load, which was selected to simulate high-traffic enterprise environments (e.g., LLM-integrated applications processing user queries at scale). This overhead stems primarily from the NER model's computational demands, including tokenization and inference on spaCy/Presidio libraries. However, statistical analysis (e.g., t-tests on repeated runs) showed no significant throughput degradation ($p > 0.05$), indicating scalability via horizontal Collector replication. In real-world deployments, this trade-off is justifiable: the added latency is negligible compared to typical LLM response times (hundreds of ms), and the resource cost (100-150 MB memory) is offset by preventing compliance fines, which can exceed millions under GDPR for PII breaches.

These findings clarify that while the regex-only mode suits low-risk, structured data scenarios, the hybrid approach is recommended for LLM prompts to prioritize privacy without crippling observability.

7. Conclusion and Future Work

The rapid integration of Generative AI has made the traditional approach to observability untenable. The risk of leaking sensitive user data into telemetry pipelines is too great to ignore. This paper has introduced Safe Observability as a necessary evolution, embedding privacy protection directly into the data collection pipeline.

I have demonstrated that the OpenTelemetry Collector serves as the ideal strategic control plane for this task. The proposed PII-Redaction Processor, integrated with a hybrid detection service, provides a concrete, architecturally sound blueprint for automatically sanitizing telemetry in-transit. My evaluation confirms that while there is a measurable performance cost, it is a manageable trade-off for the dramatic improvement in an organization's privacy posture.

Future work could enhance this framework with more advanced redaction techniques, such as tokenization, which replaces PII with consistent, non-identifiable tokens to preserve analytical utility [6]. Another promising direction is a self-adapting, regulation-aware policy engine that could dynamically apply different redaction rules based on the data's geographic origin and the corresponding legal framework (e.g., GDPR vs. CCPA) [9].

As AI becomes ever more pervasive, the principles of Safe Observability and the practice of automated telemetry sanitization will become a fundamental requirement for building secure, compliant, and trustworthy systems.

8. Limitations

While this framework advances Safe Observability, several constraints inherent to the study warrant acknowledgment. First, the evaluation relies on the **CRAPII** dataset, which consists primarily of English-language student essays. This may introduce biases toward academic text styles, potentially limiting generalizability to diverse LLM prompts in enterprise settings, such as multilingual queries or domain-specific jargon (e.g., medical or financial terms). Future validations on broader datasets, like those including non-English PII, could address this.

Second, the hybrid detection methodology, while effective, is not immune to NER model limitations. Pre-trained models like *spaCy* and Presidio may produce false negatives for rare or obfuscated PII (e.g., encoded addresses), achieving only 96% recall—meaning a small risk of leaks persists in high-stakes environments. Additionally, the approach assumes structured telemetry attributes (e.g., 'llm.prompt'); unstructured or encrypted data might require custom extensions.

Performance-wise, the benchmarks were conducted in a controlled lab setup with a single Collector instance, not accounting for distributed, production-scale deployments where network latency or variable loads could amplify the 10-20% CPU overhead. Resource-constrained environments, such as edge devices, may find the hybrid mode impractical without optimizations like model distillation.

Finally, the framework focuses on in-transit redaction and does not address post-ingestion privacy in backends or end-to-end encryption, nor does it incorporate user consent mechanisms required by evolving regulations. These limitations highlight areas for refinement, ensuring the solution's robustness across varied operational contexts.

9.Appendix A. Named Entity Recognition Algorithms and Approaches

Rule-Based Approaches: This is the simplest method, relying on hand-crafted linguistic rules, patterns, and gazetteers (dictionaries or lists of known entities).

- **How it works**:
  - Pattern-based rules: Use regular expressions (regex) to match specific formats (e.g., phone numbers, email addresses, or patterns like "Capitalized Word, then Capitalized Word" to find names).
  - Context-based rules: Look at surrounding words. For example, a word following a title like "Mr." or a location pre-position like "in" might be tagged as a Person or Location, respectively.
- **Pros/Cons**: Highly accurate in specific, narrow domains but difficult to scale, update, and generalize to new text styles.

Machine Learning-Based Approaches: These models are trained on large amounts of text that have been manually labeled with entity tags.

- **Conditional Random Fields** (CRF):
  - **How it works**: CRF is a classic statistical modeling technique used for structured prediction tasks like sequence labeling. It models the conditional probability of a sequence of output labels (the entity tags) given a sequence of input observations (the words/tokens). It considers the context of surrounding words and their predicted labels, making it more robust than simpler models.
- **Support Vector Machines (SVM) & Hidden Markov Models (HMM)**: Older ML models that have also been applied to NER, typically using handcrafted feature engineering (e.g., word capitalization, part-of-speech tag, suffixes) to help the model learn.

Deep Learning-Based Approaches: Modern state-of-the-art NER is dominated by deep learning, particularly recurrent and transformer-based neural networks.

- **Recurrent Neural Networks (RNN)/LSTMs (Long Short-Term Memory)**:
   - **How it works**: RNNs, especially LSTMs, are designed to process sequences of data. They read the text word-by-word, and the information from previous words (context) is carried forward to help classify the current word. For NER, a popular setup is a Bi-directional LSTM (BiLSTM) with a CRF layer, which processes the text in both forward and backward directions and then uses the CRF to apply sequence-level constraints to ensure a coherent sequence of entity tags.
- **Transformer Models (e.g., BERT, ROBERTa)**:
   - **How it works**: These models use an attention mechanism to weigh the importance of all other words in the sentence when processing a single word. This allows them to capture extremely complex and long-range dependencies in the text, leading to the highest performance in most modern NER tasks. They are pre-trained on massive text corpora and then fine-tuned for the specific NER task.

10.Appendix B. Protobuf Service Definition for Pii-Redaction Processor

This file defines the gRPC service contract between the Go OTel Collector processor and the external Python PII Detection Service. The Go processor acts as the client, and the Python service acts as the server. This contract enables the Go processor to send raw text and receive a structured list of detected PII entities, which it then uses to apply the configured redaction policy.

```
syntax = "proto3";

package pii_detection;

option go_package = "./pb";

// The PII detection service definition.

service PiiDetectionService {

  // Sends text to be analyzed for PII

  rpc DetectPii (PiiRequest) returns (PiiResponse) {}

}

// The request message contains the text to scan.

message PiiRequest {

  string text = 1;

}
```

```
// Defines a single detected PII entity.

// The Go processor will use this information to apply redaction.

message PiiEntity {

   // The type of PII (e.g., "PERSON", "EMAIL", "SSN")

  string type = 1;

  // The starting byte offset of the PII in the original text

  int32 start_pos = 2;

  // The ending byte offset of the PII in the original text

  int32 end_pos = 3;

}

// The response message contains a list of all detected entities.

message PiiResponse {

  repeated PiiEntity entities = 1;

}
```

References

[1] J. Turner. “What is Observability? Beyond Logs, Metrics, and Traces.” Internet: https://www.strongdm.com/observability, Oct. 23, 2025 [accessed Oct. 2, 2025].

[2] “Logging Sensitive Information - PII.” Internet: https://docs.guidewire.com/security/secure-coding-guidance/logging-sensitive-information-PII, Oct. 3, 2025 [accessed Oct. 3, 2025].

[3] D. Samanta. “When Prompts Leak Secrets: The Hidden Risk in LLM Requests.” https://www.keysight.com/blogs/en/tech/nwvs/2025/08/04/pii-disclosure-in-user-request, Aug. 4, 2025 [Oct. 4, 2025].

[4] Nightfall AI. “How does sensitive information end up in observability platforms?” https://www.nightfall.ai/blog/how-does-sensitive-information-end-up-in-observability-platforms. [Oct. 8, 2025].

[5] Proofpoint. "AI and Data Protection: Strategies for LLM Compliance and Risk Mitigation." https://www.proofpoint.com/us/blog/dspm/ai-and-data-protection-strategies-for-llm-compliance-and-risk-mitigation [Oct. 13, 2025].

[6] S. Falconer. "How to Keep Sensitive Data Out of Your Logs: 9 Best Practices." https://www.skyflow.com/post/how-to-keep-sensitive-data-out-of-your-logs-nine-best-practices, Feb. 24, 2025 [Nov. 26, 2025].

[7] M. Mudryi, M. Chaklosh, G. M. Wójcik, et al. "The Hidden Dangers of Browsing AI Agents." https://arxiv.org/abs/2505.13076 [Nov. 28, 2025].

[8] L. P. Gamage. "Named Entity Recognition (NER) for sanitizing the PII and sensitive data for public LLMs." https://blog.stackademic.com/named-entity-recognition-ner-for-sanitizing-the-pii-and-sensitive-data-for-public-llms-2273912b7b90, Mar. 5, 2025 [Oct. 22, 2025].

[9] S. Asthana, R. Mahindru, B. Zhang, and J. Sanz. "Adaptive PII Mitigation Framework for Large Language Models." https://arxiv.org/html/2501.12465v1, Jan. 20, 2025 [Oct. 13, 2025].

[10] S. Sadafal. "Masking PII in OpenTelemetry: How to Keep Observability Secure and Compliant." https://medium.com/@sonal.sadafal/masking-pii-in-opentelemetry-how-to-keep-observability-secure-and-compliant-07baeac1a286, Jul. 29, 2025 [Oct. 15, 2025].

[11] C. Risi. "Report Shows OpenTelemetry's Impact on Go Performance." https://www.infoq.com/news/2025/06/opentelemetry-go-performance/, Jun. 2025 [Nov. 26, 2025].

[12] "OpenTelemetry Architecture." Internet: https://uptrace.dev/opentelemetry/architecture [Oct. 15, 2025].

[13] S. Ajay. "Mastering the OpenTelemetry Collector: Architecture and Core Components." https://medium.com/@siddhuajay001/mastering-the-opentelemetry-collector-architecture-deployment-and-core-components-3ebe7e1c9c39, Oct. 27, 2024 [Oct. 20, 2025].

[14] P. Sonpatki. "What is the OpenTelemetry Collector and How Does It Work?" https://last9.io/blog/what-is-opentelemetry-collector/, Jul. 17, 2024 [Oct. 20, 2025].

[15] "Collector." Internet: https://opentelemetry.io/docs/collector/ [Oct. 22, 2025].

[16] S. Curran. "A simple Python library to detect PII in structured data, powered by Microsoft Presidio and spaCy NER." https://github.com/shanecurran/piiscan [Oct. 22, 2025].

[17] "Using NLP and Pattern Matching to Detect, Assess, and Redact PII in Logs - Part 1." https://www.elastic.co/observability-labs/blog/pii-ner-regex-assess-redact-part-1, Oct. 24, 2025 [Oct. 24, 2025].

[18] A. Jones. “SOTA PII Redaction on Your Laptop.” https://openpipe.ai/blog/pii-redact, Mar. 26, 2025 [Oct. 20, 2025].

[19] “Building a custom collector.” https://opentelemetry.io/docs/collector/custom-collector/ [Oct. 10, 2025].

[20] “The Learning Agency Lab - PII Data Detection.” https://www.kaggle.com/c/pii-detection-removal-from-educational-data/data [Oct. 5, 2025].