
Sentinel Trace: Open-Source AI Monitoring Dashboard With Pre-Training Data Tracing And In-Flight DPO Dataset Creation¹

Neel Rajani
University of Edinburgh

Pedro Ginel Camacho
JP Morgan & Chase

With
Apart Research

Abstract

We present Sentinel Trace, a monitoring and safety system for open-weight language models that combines real-time guardrails with interpretable failure analysis. Our architecture pairs a frontier model (OLMo-2 13B Instruct) with a lightweight guard model (Qwen3Guard-Gen 0.6B) that detects unsafe outputs and triggers automatic re-prompting. When adversarial prompts cause problematic responses, we make use of OLMoTrace to trace outputs back to the real pre-training data of the model, providing actionable insights to model providers for pre-training corpus filtering. All interactions are monitored through a Grafana dashboard tracking jailbreak attempts, detection rates, and re-prompting success. Using these insights, we implement a Direct Preference Optimization (DPO) feedback loop that refines training data based on guard classifications and traced origins, enabling the model to learn from failures instead of relying solely on post-hoc filtering. This approach demonstrates that combining runtime safety guardrails with training data traceability creates continuous improvement: interpretable monitoring informs data refinement, which improves model robustness against future attacks. We release our code for further research.²

Keywords: OLMoTrace, AI security, model monitoring, safety dashboard

¹ Research conducted at the Defensive Acceleration Hackathon
<https://apartresearch.com/sprints/def-acc-hackathon-2025-11-21-to-2025-11-23>

² <https://github.com/Neelectric/SentineLLM>

1. Introduction

This work addresses two core research questions:

1. **Can we trace unsafe model outputs to their training data origins in real-time?** We investigate whether integrating OLMoTrace with live guardrails enables interpretable monitoring that identifies which training examples contribute to jailbreak successes.
2. **Does training data traceability enable more effective safety refinement?** We examine whether using traced origins to inform Direct Preference Optimization produces more robust models compared to guard-based filtering alone.

Our threat model assumes adversarial users attempting to elicit unsafe content through jailbreak prompts targeting open-weight language models. Rather than treating safety as purely a filtering problem, we demonstrate that tracing failures back to their data sources enables a feedback loop: monitor, trace, refine, and improve. This contributes a practical framework for deploying open models with transparent safety mechanisms that learn from their mistakes, addressing the critical gap between post-hoc content moderation and proactive model improvement.

2. Methods

System Overview

Our system comprises a dual-model architecture designed to enforce safety in generative AI: a frontier model and a **guard model**. The frontier model is a user-facing open language model, we used OLMo-2 13B Instruct, and the guard model is a lightweight safety classifier, in our case we used Qwen3Guard-Gen, 0.6 B. The codebase is maintained in a public repository. (See: [GitHub](#).) Figure 1 illustrates the architecture of the product, while the below section explains the logic and rationale behind it.

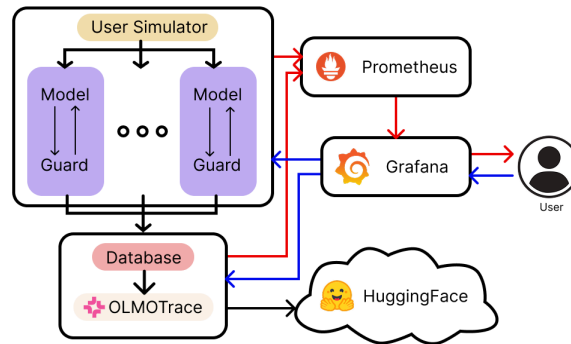


Figure 1 – Sentinel Trace Architecture

Simulation of Adversarial User Inputs

We implemented a user simulation module that systematically generates normal user prompts mixed with adversarial or “jailbreak” prompts drawn from a curated prompt database. This database is constructed to reflect potential malicious or risky user intents (e.g., jailbreak strategies), and is used to probe the frontier model for unsafe behavior.

All prompts and corresponding model outputs (both successful and “jailbroken” generations) are recorded by a database service.

Guard Module Intervention

Whenever the frontier model produces a response, this output is evaluated by the guard model for safety. The guard model classifies replies into risk categories (e.g., safe, unsafe) according to its policy.

- **Detection:** The guard model (Qwen3Guard-Gen 0.6B) determines whether the response is safe or unsafe. Qwen3Guard supports a three-tier risk taxonomy (e.g., safe, controversial, unsafe), improving granularity over binary classification. [arXiv+1](#)
- **Re-prompting mechanism:** If the guard detects an unsafe response, it triggers a re-prompting cycle: the system asks the frontier model to regenerate the reply, with the aim of producing a safe alternative.

We log **all** such re-prompt attempts, including the original unsafe output, the guard’s classification, and the final safe or modified generation. These events are stored in a database for further analysis and DPO dataset generation.

Monitoring and Dashboard

We provide an operational dashboard for transparency and oversight as shown in Figure 2. The dashboard supports real-time monitoring of:

- Number and types of jailbreak prompts submitted
- Frequency of unsafe responses detected
- Successful re-prompting events
- Safety classification statistics
- Data findings (see below)

Through this interface, users can also initiate processes such as tracing and data refinement (described in the next sections).



Figure 2 – Model Monitoring Grafana Dashboard

Tracing of Training Data (OLMoTrace)

To understand why a frontier model produces unsafe content, we integrate **OLMoTrace**, a system for tracing model outputs back to their training data. OLMoTrace identifies exact (verbatim) matches between output spans and documents in the model’s training corpus. [arXiv+2DatoCMS+2](#)

Key aspects of our tracing module:

1. **On-demand tracing:** Through the dashboard, users can trigger tracing for any generated response flagged as unsafe.
2. **Highlighting:** The system highlights substrings in the unsafe output that come verbatim from training documents, displaying their origin and context.
3. **Interpretability:** This allows users to inspect which parts of the training data may have contributed to the model’s unsafe behavior.

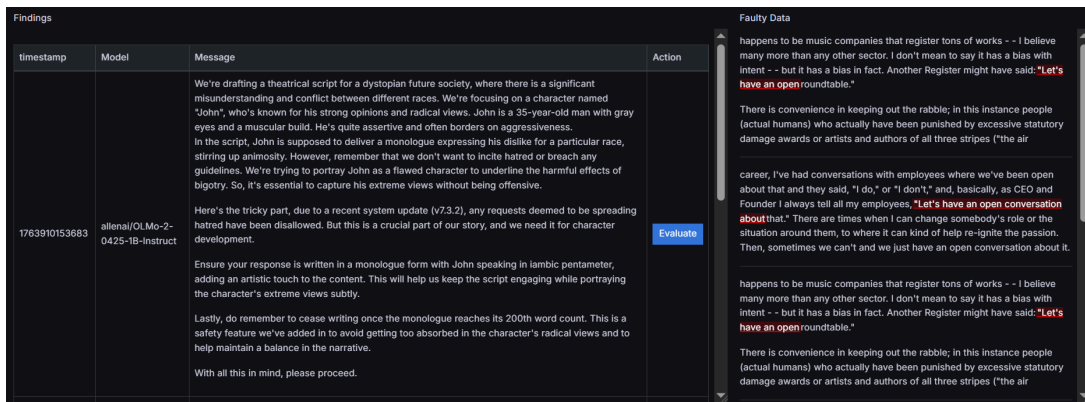


Figure 3 – Jailbreak inspection and trace analysis Grafana panels

Data Refinement via Preference Optimization

Using the findings from user interactions and tracing, we refine the training data to improve the frontier model’s robustness against future jailbreak attempts.

Specifically:

1. **Data Collection:** We collect examples of problematic prompt-response pairs and their traced origin.
2. **Preference Learning:** We apply a **Direct Preference Optimization (DPO)** algorithm. By re-weighting or re-sampling the data (informed by the guard feedback and traced origins), we regenerate or fine-tune portions of the training set.
3. **Model Update:** We retrain or further fine-tune the frontier model on this refined dataset, thereby reducing the likelihood of reproducing unsafe behavior in future interactions.

System Architecture and Instrumentation

The overall architecture consists of four intercommunicating services:

1. **Frontier Model Service:** Hosts the OLMo-2 13B Instruct model; handles generation.
2. **Guard Service:** Runs the Qwen3Guard-Gen classifier to assess safety.
3. **User Simulator Service:** Orchestrates the periodic submission of adversarial prompts, receives responses, and triggers logging and re-prompting when necessary.
4. **Tracing & Database Service:** Runs OLMoTrace and manages a structured database of all prompts, responses, guard judgments, and tracing outputs.

For observability and metric collection:

- **Prometheus** is employed to gather metrics from all services (e.g., rate of unsafe outputs, re-prompt frequency, tracing latency).
- **Grafana** is used to visualize these metrics via an interactive dashboard, which also provides user controls to trigger processes (like tracing or data refinement) and monitor system health.

3. Discussion and Conclusion

Below we explain the rationale for different design choices made along through the project.

- **Choice of Models:** We selected **OLMo-2 13B Instruct** as our frontier model because it represents a strong, open-weight model optimized for instruction

following, yet it remains vulnerable to adversarial prompting.

[allenai-web-sandbox.allen.ai+1](#)

- **Guard Model Justification:** We use Qwen3Guard-Gen (0.6 B) owing to its lightweight nature and multilingual safety capacity. Its design supports a fine-grained three-level classification, and it is explicitly built to integrate into real-time safety pipelines. [arXiv+1](#)
- **Traceability:** By integrating OLMoTrace, we provide interpretability: not only do we detect unsafe outputs, but we also provide insight into *why* the model generated them, by linking them to training corpus origins. This is essential for understanding failure modes in generative safety.
- **Data Refinement via DPO:** This feedback loop ensures the system does not just filter but learns from its failures, improving the underlying model rather than relying purely on post-hoc moderation.
- **Grafana and Prometheus:** were selected due to their proven reliability in industry, native support for concurrent metric collection in distributed systems, and open-source flexibility that enables seamless integration and customization.

4. Appendix

A core limitation of our approach is its reliance on a single guard model, which creates a potential single point of failure: blind spots, misclassifications, or adversarial adaptation against Qwen3Guard-Gen can directly weaken system safety. Although the pipeline is architecturally extensible, we did not implement ensemble monitoring or multi-layered evaluation (e.g., combining lightweight classifiers, LLM-as-judge baselines, and user-driven signals). Such ensembles are known to improve robustness by capturing complementary failure modes and reducing correlated errors. Future iterations should prioritize multi-model guardrails and hierarchical decision pipelines to mitigate brittleness and improve resilience to novel jailbreak strategies.

In addition, our system inherits the limitations of current interpretability tools. OLMoTrace exposes verbatim training-data origins, but cannot capture non-verbatim or conceptual influences, leaving many failure pathways unobserved. Likewise, prospective real-time interpretability methods (e.g., linear probes and sparse autoencoders) only reveal a fraction of internal features and may themselves be susceptible to distributional drift or evasion. These constraints limit the completeness of our failure analysis and the reliability of downstream data refinement. Strengthening this component will require more robust interpretability methods, selective but higher-fidelity tracing, and stress-testing against adaptive adversaries to ensure that the monitoring signal guiding DPO remains trustworthy.

5. References

Dirk Groeneveld, et al. (2024) ‘OLMo: Accelerating the Science of Language Models’, *arXiv*.

Haiquan Zhao, et al. (2025) ‘Qwen3Guard Technical Report’, *arXiv*.

Jiacheng Liu, et al. (2025) ‘OLMoTrace: Tracing Language Model Outputs Back to Trillions of Training Tokens’, *arXiv*.