Guardrails-as-a-Service

Pradeep Annepu, Akula Rajesh, Amarendra Chakravarthi, Anirudh Reddy
Email: {M25AI1109, M25AI1048, M25AI1082, M25AI1131}@iitj.ac.in

***Abstract*—We present the architecture and planned evaluation of a Guardrails-as-a-Service platform. The system is designed to scale: it runs as stateless microservices connected by asynchronous events.It's easy to extend because new handlers can be plugged in without changing the core. Every decision was logged in event driven way to scale independently. We trianed a gemma model with custom domain language to act as a guard model. We define measurable quality attributes (latency, throughput) using a load testing.**

***Index Terms*—Policy enforcement, microservices, vector similarity, audit logging, observability, gemma, guardrails.**

I. Introduction

Enterprises increasingly rely on the agentic systems and exposing varous tools and capabilities to their customers. The need for guarding the tools and capabilities is also increasing with these new systems. Training a large language model from the scratch can be a expensive affair for the many enterprises and many make use of the open source/weight llm models available in the market. These open models have trained in general and may not be alighed with enterprises needs and their domain knowledge. Hence the need for guardrails to these models is required for enterpises to avoid any legal and compliance issues. It is also very important to save their inference cost by avoiding any unwanted/unnecessary calls to the llm models. For example, if an enterprise of payment gateway is exposing an agentic systems to their customers to help with their support queries, but hackers may try to exploit the system to get the sensitive information or even play with it general discussion which may not be related to the customer business, which can lead to huge inference cost to the company.

It is not a one time job for these enterprises to set these guardrails and forget about it, as the business needs and complaince requirements keep changing, these guardrails need to be updated frequently. Hence the need for a policy service which can help enterprises to create, update and manage these guardrails easily.We address these gaps by designing a scalable system for the guardrails evaluation and continuous policy updates and management in this study.

II. Contributions

The work provides: (1) A hybrid evaluation architecture combining deterministic rule filtering and vector similarity; (2) An extensible handler interface enabling semantic and emerging policy types with low change cost; (3) A hash-linked audit log for tamper-evidence without blockchain overhead; (4) A metrics-driven evaluation plan aligning architectural tactics with verifiable quality attributes.

III. System Architecture

Fig. 1. System Architecture: Hybrid policy evaluation with event-driven decoupling

Refer Fig. 1 For System Architecture
Core components:

- API Gateway: Authentication, authorization, rate limiting.
- Policy Service: CRUD, versioning, vector indexing (pgvector [4]).
- Evaluation Plane: Cache-first retrieval, rule + semantic evaluation, decision event emission.
- Audit Logger: Consumes decision events; persists hash-chained records.
- Redis: Low-latency hot-set and embedding cache.
- Broker: Decouples evaluation from persistence; supports resilience tests.
- Observability Stack: Metrics (Prometheus [3]), logs, planned traces (OpenTelemetry [5]).

IV. Design Methodology

Microservice decomposition isolates policy mutation from evaluation throughput. Event-driven processing reduces synchronous coupling. Vector similarity augments metadata filtering, allowing semantically proximate policy selection. Hash chaining supplies integrity assurance with linear verification.

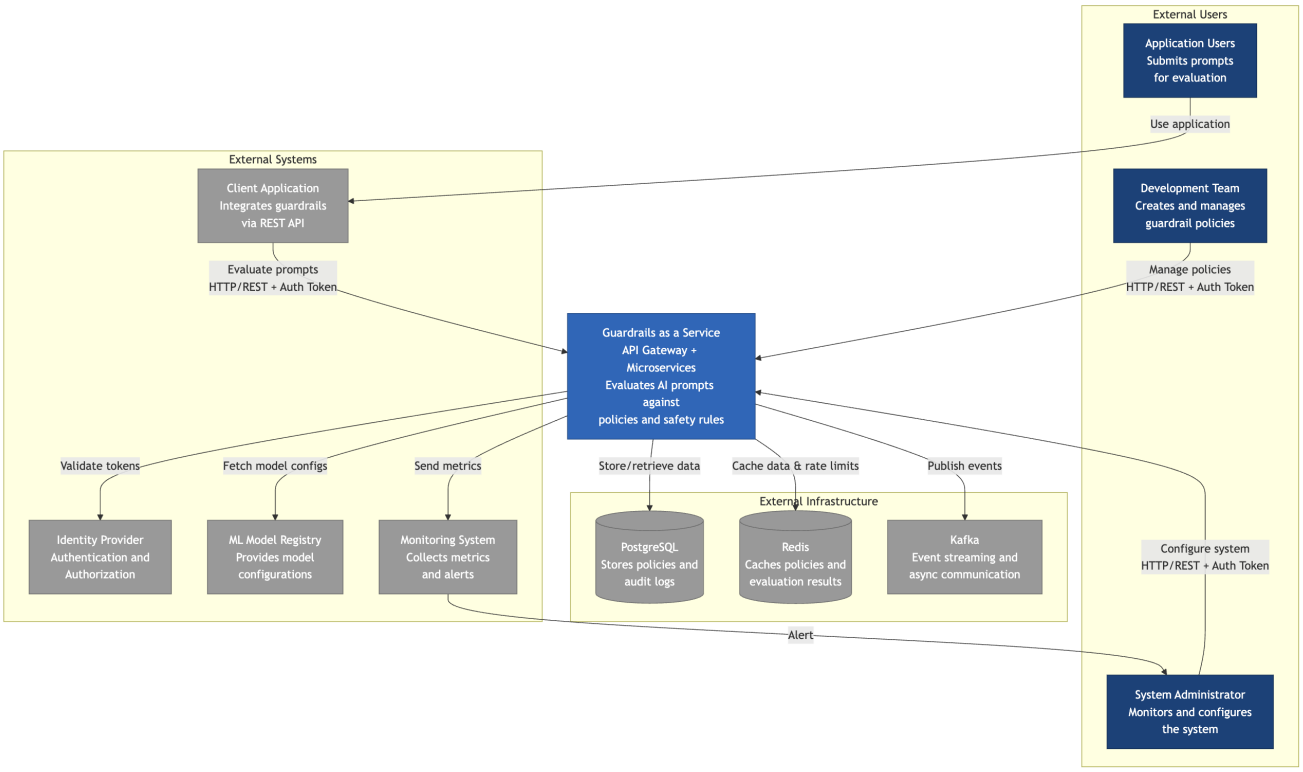

Fig. 2. System Architecture: Hybrid policy evaluation with event-driven decoupling

V. Quality Attributes

A. Scalability

Tactics: Stateless pods, horizontal autoscaling, Redis caching, asynchronous buffering. Metrics: P95 latency $<$ 150 ms under 2000 requests/sec; cache hit ratio correlated with latency reduction.

B. Extensibility

A `PolicyHandler` interface:

```
export interface PolicyHandler {
  supports(policyType: string): boolean;
  evaluate(context: any, policy: Policy): Promises
}
```

New handlers (e.g., advanced semantic constraint) integrate via dependency injection without core modification. Verification: Diff locality; unchanged regression suite; cyclomatic complexity ¡ 10.

C. Observability & Auditability

Correlation IDs propagate through logs and metrics. Audit integrity uses chained hashes:

```
prev_hash = "GENESIS"
for event in decision_events:
    serialized = canonical_json(event)
    curr_hash = sha256(prev_hash + serialized)
    store({...event, prev_hash, curr_hash})
    prev_hash = curr_hash
```

Periodic verification recomputes sequence and alerts on mismatch; target audit lag ¡ 5 s at sustained load.

D. Maintainability

Modular boundaries and standardized evaluation signatures reduce change effort. KPIs: Mean Time To Change for routine policy extension ¡ 0.5 developer-day.

VI. Implementation Overview

Policies stored in PostgreSQL with pgvector embeddings; evaluation pipeline performs: (1) Cache lookup; (2) Vector similarity search (threshold-based); (3) Rule evaluation; (4) Decision event emission; (5) Audit service hash-link persistence.

VII. Evaluation Plan

Phases: (1) Functional tests (unit, integration); (2) Load tests (k6 ramp to 2000 req/sec); (3) Resilience (broker partition throttling). Metrics scraped every 5 s; Grafana dashboards visualize throughput, latency, audit lag. Success: No message loss; hash chain validates end-to-end; latency targets met.

VIII. Security Considerations

Current: JWT-based client auth; least-privilege DB roles; rate limiting. Planned: mTLS inter-service; policy signature verification for provenance.

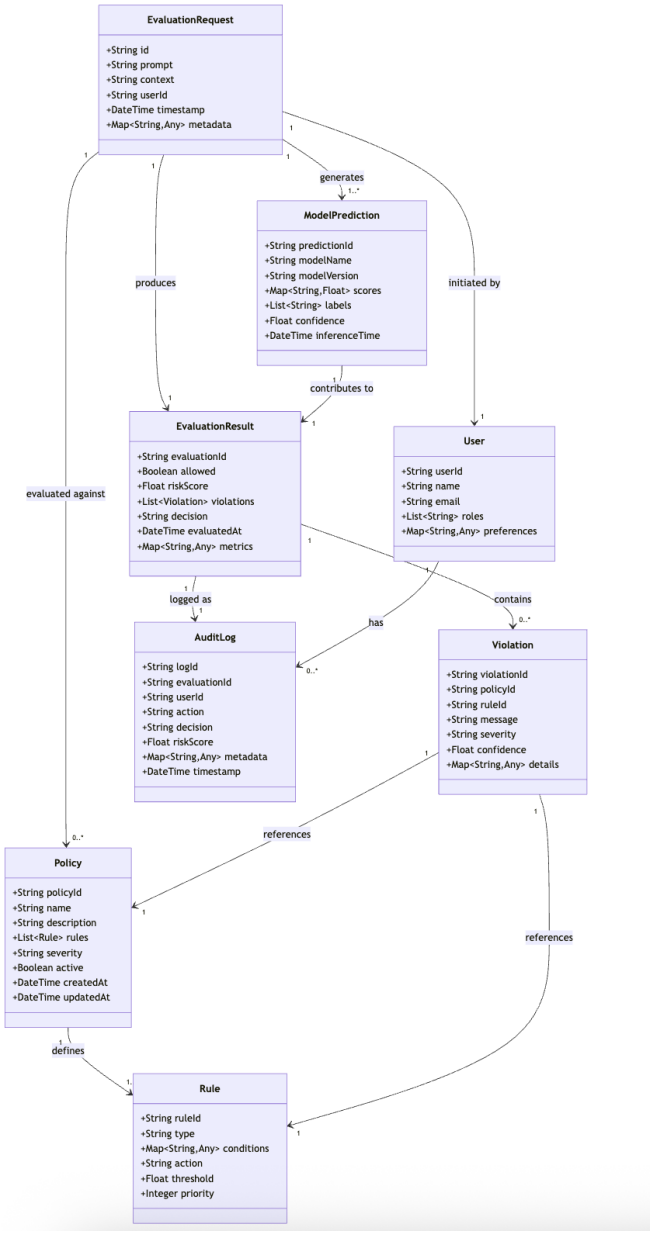

Fig. 3. System Architecture: Hybrid policy evaluation with event-driven decoupling

IX. Related Work

OPA [1] offers strong declarative evaluation but lacks semantic retrieval integration and native hash-chained audit logging. AWS Control Tower guardrails [2] provide opinionated cloud governance with limited cross-cloud extensibility. Our approach differentiates through hybrid retrieval, event-driven decoupling, and cryptographic audit integrity.

X. Conclusion

The proposed Guardrails-as-a-Service platform aligns architectural tactics with measurable quality attributes. Hybrid policy evaluation and hash-linked auditing together enhance governance fidelity and trust. Future work targets production hardening, expanded semantic handlers, and full distributed tracing.

Acknowledgment

The authors would like to thank the Indian Institute of Technology Jodhpur for supporting this study.

References

[1] Open Policy Agent Documentation. [Online]. Available: https://www.openpolicyagent.org
[2] AWS Control Tower Guardrails. [Online]. Available: https://docs.aws.amazon.com/controltower
[3] Prometheus Documentation. [Online]. Available: https://prometheus.io
[4] pgvector Extension Documentation. [Online]. Available: https://github.com/pgvector/pgvector
[5] OpenTelemetry Specification. [Online]. Available: https://opentelemetry.io

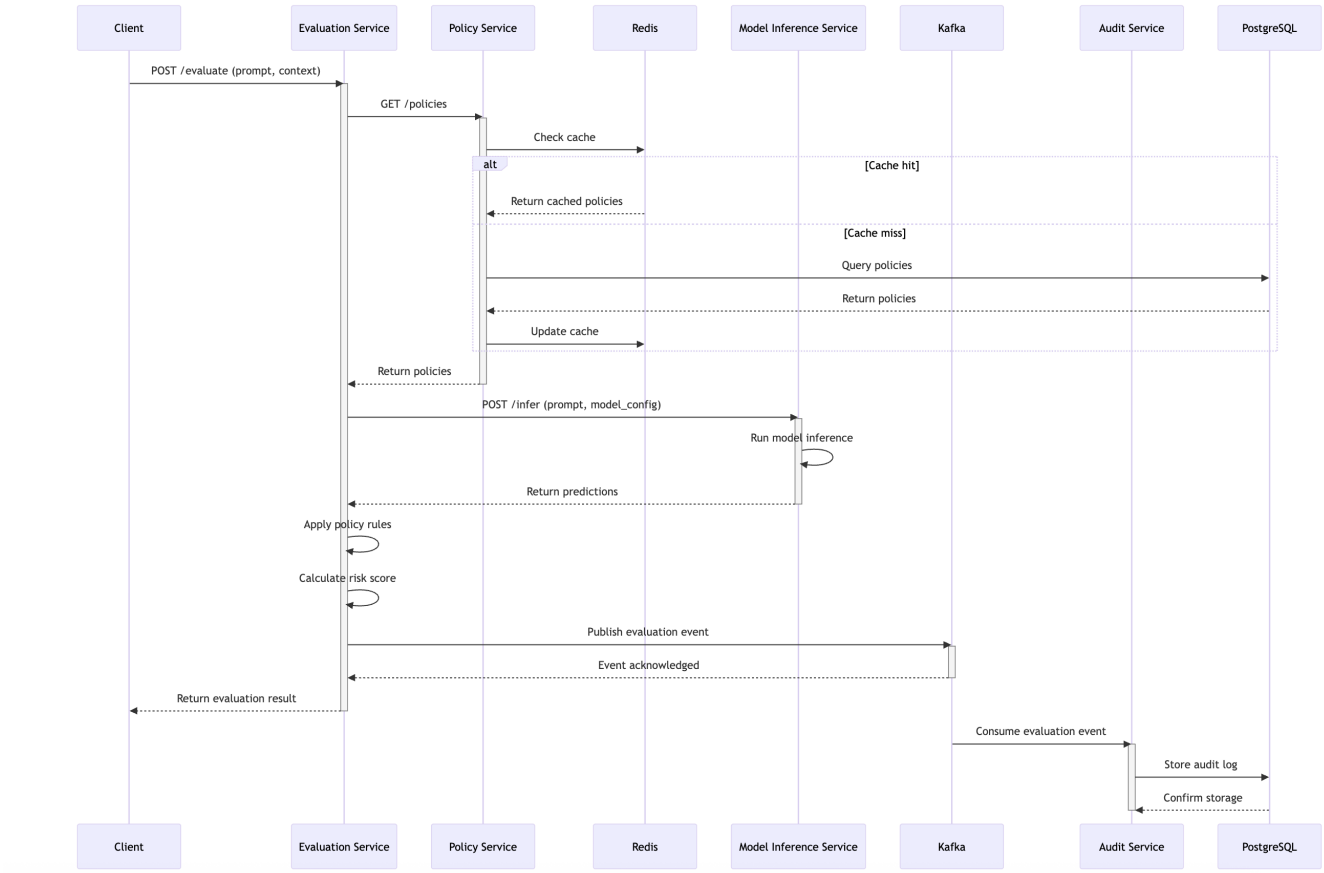

Fig. 4. System Architecture: Hybrid policy evaluation with event-driven decoupling

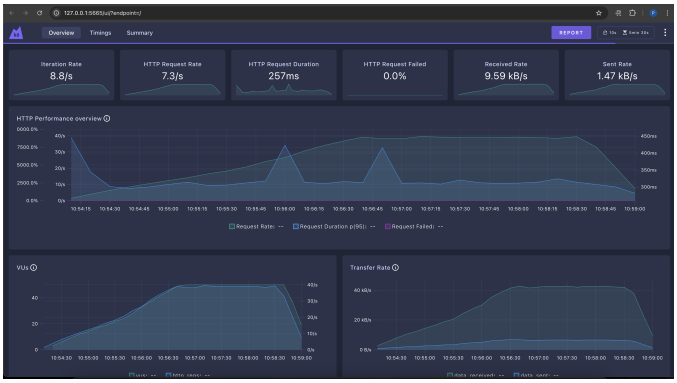

Fig. 5. System Architecture: Hybrid policy evaluation with event-driven decoupling