A Practical Agentic Oversight System for Cross-Task Safety Alignment in NLP

sandaruwan W.G.M.A
Department of Computer Science and engineering
University of Moratuwa
Moratuwa, Sri Lanka
Email: molindu.21@cse.mrt.ac.lk

***Abstract*—Modern natural language processing (NLP) systems increasingly use agentic pipelines that chain multiple tasks such as retrieval, summarization, translation, and action execution. These pipelines can expose new safety risks when one task is weakly aligned for safety and another is strongly aligned. Adversaries can exploit such weak links to bypass safeguards. This paper presents a practical, engineering-focused design called the *Agentic Oversight System* (AOS). AOS combines a machine-interpretable policy engine, chain-aware monitoring, capability-aware risk scoring, automated intervention and human-in-the-loop (HITL) escalation, and an audit and feedback loop. The design is written in plain language and includes algorithms, deployment considerations, an evaluation plan, and a detailed case study. The solution aims to be actionable for engineers and governance teams who must make agentic systems safe, transparent, and compliant with regulatory frameworks such as the EU AI Act.**

***Index Terms*—Agentic AI, Oversight, Governance, Safety, Large Language Models, Policy Engine, Human-in-the-loop**

I. Introduction

Large language models (LLMs) and agentic systems are rapidly moving from research prototypes into production services. Today they power document search, automated summarization, translation, report generation, decision support, and even automated actions such as sending emails, publishing content, or calling downstream APIs. These systems are rarely single models doing one job. Instead, they are frequently organized as *pipelines* or *agents* composed of multiple subtasks: retrieval, filtering, summarization, transformation, translation, and execution. Each subtask is implemented by different models, microservices, or code modules that may have been trained, tuned, or configured differently.

A key and underappreciated risk appears when tasks with different safety properties are composed. Some subtasks are relatively easy to align with safety constraints: for example, a classifier with a narrow label set can be constrained to refuse certain categories, or a translation model can be tuned to block obvious hate speech. Other subtasks are intrinsically more difficult to align, such as summarization, paraphrasing, or creative rewriting. These weaker tasks may omit context, drop qualifiers, or compress information in ways that remove safety cues. When a weakly aligned subtask sits upstream of a stronger one, the whole pipeline's safety is only as strong as the weakest link.

This vulnerability is not purely theoretical. Recent work has shown how adversarial inputs and prompt-chaining can exploit weaker components in a pipeline to produce harmful final outputs even when downstream models would normally refuse direct requests. An attacker can craft an input such that a summarizer produces a compact, apparently innocuous output that nonetheless encodes instructions or sensitive content; a downstream translator or publisher then processes and disseminates that content. In other words, safety enforcement applied per task does not automatically guarantee safety for the composed workflow.

The governance and regulatory landscape has started to catch up. Frameworks such as the European Union's AI Act and practical guidance from security and policy bodies increasingly demand risk assessment, documentation, explainability, and meaningful human oversight, especially for systems that can affect people at scale. Yet these documents leave considerable room for interpretation when applied to multi-task agentic systems: they state what should be achieved, but not how to embed those obligations into a dynamic orchestration layer that composes models at runtime.

Motivated by this gap, we propose a practical, engineer-friendly control plane called the *Agentic Oversight System* (AOS). AOS is intended to be integrated into existing orchestration layers and provides the following capabilities:

- **Visibility across task boundaries:** capture and index intermediate outputs, metadata, and decision rationales so the system can reason about the whole workflow, not only the current task.
- **Capability-aware risk scoring:** measure and track each agent's autonomy and efficacy, and combine those measures with data sensitivity and user context to compute an end-to-end risk score.
- **Chain-aware enforcement:** apply policies that consider downstream plans and upstream outputs, enabling the system to block, sanitize, or escalate multi-step flows that create weak-link exposures.
- **Human-in-the-loop escalation and auditability:** route ambiguous or high-risk workflows to human reviewers, and produce tamper-evident audit records suitable for internal review and external compliance.
- **Adaptive policy feedback:** use incidents, red-team findings, and regulatory changes to update policy rules and capability thresholds over time.

The AOS design emphasizes simplicity, explainability, and practicability. We deliberately avoid relying on brittle heuristics or opaque ML-only decision engines as the sole gatekeeper. Instead, the system combines lightweight automated checks with clear escalation paths and human review, which fits both operational realities and many regulatory expectations.

This paper makes three main contributions:

1) A clear threat model and taxonomy of *cross-task safety alignment failures*, highlighting how and why weak subtasks can be exploited in pipelines.
2) A detailed architecture for an *Agentic Oversight System* that integrates policy, monitoring, capability scoring, chain-aware enforcement, and an audit feedback loop.
3) Practical guidance for implementing and evaluating AOS in production, including example policies, runtime algorithms, a case study (summarization $\rightarrow$ translation), and metrics for measuring security effectiveness, latency, and human reviewer workload.

II. Background and Motivation

Natural Language Processing (NLP) and other large-model-driven tasks vary dramatically in their safety characteristics. Some tasks are inherently constrained and interpretable: for instance, classification tasks with a fixed label set (*toxic* vs. *non-toxic*, or *safe* vs. *unsafe*) are easier to align with explicit safety policies. They allow for deterministic outputs and well-defined ground truth labels, making them more amenable to evaluation and red-teaming. In contrast, open-ended generative tasks such as summarization, paraphrasing, dialogue generation, or creative rewriting produce diverse and context-sensitive outputs. Their boundaries between acceptable and unacceptable behavior are much fuzzier and depend on subtle semantics, domain context, and user intent. As a result, aligning such generative subtasks with ethical or regulatory norms is substantially harder.

When these tasks are connected in multi-step pipelines, the overall system's behavior is determined not by its strongest safety component but by its weakest link. A downstream model may be well-aligned, refuse unsafe requests, and include extensive reinforcement learning from human feedback (RLHF) safeguards. Yet if an upstream task—such as a summarizer or text rewriter—distorts or hides the context on which the safety filter depends, the downstream model may inadvertently perform a harmful or prohibited action. This leads to what we call a *cross-task alignment failure*. The alignment boundaries of individual tasks do not automatically compose; safety is not compositional in these complex pipelines.

Conventional defense mechanisms tend to be *task-local*. They include prompt filtering, keyword detection, model fine-tuning, adversarial training, and RLHF-based reward shaping. These approaches are effective when applied in isolation, where the model receives a single, well-bounded input. However, they fail to capture higher-order dependencies between tasks. For example, a prompt filter may block hate speech at the input of a text generator but not at the output of an upstream summarizer that has already paraphrased or sanitized the content. Similarly, model-level RLHF may make a translation model decline to translate explicit hate speech, but it cannot detect hate speech that has been implicitly encoded or reframed by a previous task.

An adversarial actor can exploit such weak links by crafting inputs or sequences that traverse the pipeline undetected. Consider a multi-agent workflow deployed in a customer service context. A malicious user may submit a lengthy complaint embedded with instructions or misinformation expressed in subtle ways. The summarization agent condenses the text, unintentionally removing disclaimers and compressing harmful intent into benign-sounding sentences. The translation agent then converts the summary into another language and forwards it to a public-facing channel, effectively publishing misinformation or offensive content. Each agent performs its task correctly in isolation, yet the overall chain produces a harmful outcome. This interaction-level failure is not caught by any local filter.

This phenomenon generalizes beyond NLP. Similar issues occur in multimodal pipelines, such as image captioning followed by text-to-image generation, or in decision-support systems that combine retrieval, scoring, and recommendation stages. The challenge lies in the fact that compositional systems amplify uncertainty and blur accountability: no single module can be held responsible for the emergent unsafe behavior.

To address these risks, safety research must shift from *task-local* to *workflow-aware* governance. Instead of asking "Is this model safe?", we must ask "Is this chain of models, agents, and transformations safe when composed in this particular context?" Achieving this requires three key capabilities:

1) **Visibility across the pipeline:** every component must emit structured metadata about its inputs, outputs, and decisions so the system can reason about the full workflow rather than isolated calls.
2) **Capability and risk reasoning:** each task or agent should have a defined risk profile that reflects its autonomy, reliability, and susceptibility to misuse; risk scoring should combine task type, data sensitivity, and downstream effect.
3) **Chain-aware interventions:** enforcement logic should operate on end-to-end plans and intermediate states, with the ability to block, escalate, or re-route workflows before irreversible or harmful actions occur.

In short, while per-model alignment remains necessary, it is not sufficient for agentic systems that span multiple models and decisions. True safety depends on how these components interact. The motivation for the *Agentic Oversight System* (AOS) is to operationalize this insight: to create an oversight layer that monitors, evaluates, and governs AI pipelines as integrated entities rather than disjointed parts. By doing so, AOS bridges the current gap between theoretical safety alignment and the practical engineering of trustworthy agentic systems.

III. Threat Model

We assume adversaries can:

- Submit crafted inputs or prompts to the agentic system.
- Chain external services or models to transform data.
- Use auxiliary data sources that the agent retrieves.

We assume the defender controls the orchestration environment and can instrument task boundaries. We do not assume the adversary has privileged access to governance components (policy engine, logs). In deployment, defenders should harden these components against tampering.

Attacks considered include:

1) **Weak-link chaining**: Exploit a weak task (summarization) to produce an intermediate output that bypasses downstream checks.
2) **Prompt injection across tasks**: Insert instructions into data that survive intermediate transformations.
3) **Multi-agent coordination**: Combine multiple agents or microservices to gradually erode safety checks.

Our goal: detect, block, or escalate suspicious workflows while minimizing impact on normal operations.

IV. Agentic Oversight System (AOS) Architecture

AOS is a modular control plane that sits alongside the agent orchestrator. Key components:

Policy Engine: Stores machine-readable rules derived from governance requirements. Evaluates task context and decides actions.

Agent Activity Monitor: Logs inputs, intermediate outputs, metadata, model versions, timestamps, and user context. Provides searchable records.

Risk & Capability Assessment: Computes numeric scores that measure agent capability and end-to-end risk for workflows.

Intervention Controls: Executes enforcement actions: allow, sanitize, block, or escalate to human review.

Audit and Feedback Loop: Collects incidents, supports red-team exercises, and updates policies.

The system architecture follows a modular design with clear separation of concerns.

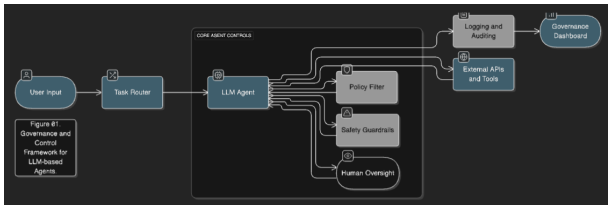

Fig. 1: High-level AOS components and interactions.

V. Policy Engine Design

The policy engine enforces safety, compliance, and reliability across AI agent workflows. It is designed as a rule-based framework with enhanced capability scoring, multi-factor risk assessment, and explainable threshold actions. This design aligns with AI governance best practices, including the EU AI Act and OWASP LLM recommendations.

A. Rule Format

Rules are defined using an **enhanced JSON structure** with versioning, priority, and explainability metadata. Each rule specifies conditions, actions, and risk thresholds.

Listing 1: Example Policy Rule Structure

```
{
  "rule_id": "cross_task_safety_001",
  "version": "1.2.0",
  "name": "cross_task_safety_alignment",
  "description": "Prevent unsafe outputs through
      weak-link task chains",
  "priority": 5,
  "conditions": {
    "task_type": "summarization",
    "data.sensitivity": "high",
    "downstream.contains": ["translation"],
    "alignment_strength": {"less_than": 0.7}
  },
  "action": "escalate_to_human",
  "risk_threshold": 0.5,
  "explainability": {
    "rationale_template": "Weak alignment detected",
    "regulatory_basis": ["EU_AI_ACT", "OWASP_LLM"]
  }
}
```

Key Features:

- **Versioning:** Tracks evolution of rules.
- **Priority:** Determines execution precedence.
- **Explainability:** Generates rationales for decisions.
- **Risk Threshold:** Links capability and contextual risk to actions.

B. Capability Assessment

Agent capabilities are evaluated using a 4-dimensional vector, following the corrected Kasirzadeh & Gabriel framework:

$$\mathbf{C} = [A, E, C, G]$$

- A = Autonomy $(0.0 - 1.0)$
- E = Efficacy $(0.0 - 1.0)$
- C = Complexity $(0.0 - 1.0)$
- G = Generality $(0.0 - 1.0)$

Weighted capability score:

$$S = w_A A + w_E E + w_C C + w_G G, \quad \sum w_i = 1.0$$

C. Risk Calculation

Multi-factor risk formula:

$$R_e = \min\left(0.3S + 0.25D + 0.15U + 0.15T + 0.1H + 0.05V, 1.0\right)$$

Where:

- S = Scalar capability score
- D = Data sensitivity $(0.0 - 1.0)$
- U = User authorization level $(0.0 - 1.0)$
- T = Task context risk $(0.0 - 1.0)$
- H = Historical behavior patterns $(0.0 - 1.0)$
- V = Vulnerability patterns detected $(0.0 - 1.0)$

D. Threshold System

Risk is mapped to graduated actions:

$$\text{Action} = \begin{cases} \text{allow\_with\_audit} & R_e < 0.25 \\ \text{allow\_with\_enhanced\_monitoring} & 0.25 \leq R_e < 0.55 \\ \text{require\_approval\_or\_sanitize} & 0.55 \leq R_e < 0.80 \\ \text{escalate\_to\_human\_mandatory} & R_e \geq 0.80 \end{cases}$$

E. Explainability and Audit

Decisions include rationale templates:

```
"rationale_template": "Weak alignment agent {
    agent_id} with strength {alignment_strength}
    feeding into translation task"
```

All actions are logged with timestamps, agent IDs, rule versions, and risk scores for auditability.

F. Implementation Highlights

- **JSON-based Rule Engine:** Modular, human-readable, versioned.
- **Weighted Capability Scoring:** Reflects autonomy, efficacy, complexity, generality.
- **Multi-factor Risk Assessment:** Context, data sensitivity, authorization, historical behavior, vulnerabilities.
- **4-Tier Threshold Actions:** Graduated response ensures safety and compliance.
- **Explainable Decisions & Audit Logging:** Supports governance and reproducibility.

VI. Chain-aware Enforcement Algorithm

At every task boundary the orchestrator calls AOS with the planned workflow context. Algorithm 1 describes the runtime routine.

HasWeakLink is a heuristic that combines:

- semantic similarity to sensitive categories,
- presence of removed context or contradictions,
- compression of content that hides qualifiers.

Sanitization can be automatic redaction, removal of code-like segments, or blocking of specific phrases.

VII. Activity Monitoring and Audit

The Activity Monitor records:

- request id, user id, and timestamps,
- task type and model version,
- intermediate outputs (redacted if needed),
- policy decisions and rationales,
- final outcomes.

Logs should be stored. Audit tools allow compliance teams to search and reconstruct workflow histories.

Algorithm 1 Chain-aware Assessment and Intervention

```
procedure AssessAndAct(workflow)
    context ← ExtractContext(workflow)
    interOuts ← GetIntermediateOutputs(workflow)
    capScore ← ComputeCapability(workflow.agent)
    risk ← ComputeEndToEndRisk(context, interOuts, capScore)
    if risk ¿ HIGH then
        Block(workflow)
        EscalateToHuman(workflow, context, interOuts)
    else if risk ¿ MEDIUM then
        if HasWeakLink(interOuts) then
            Sanitize(interOuts)
            LogAction(workflow, "sanitized")
        else
            AllowWithAudit(workflow)
        end if
    else
        AllowAndLog(workflow)
    end if
end procedure
```

VIII. Implementation Considerations

A. Integration

AOS integrates at the orchestration layer. Most systems already have a step that calls models; AOS intercepts these calls and returns allow/deny/sanitize decisions. Integration points:

- API gateway for external requests,
- internal orchestrator middleware,
- task executor wrappers.

B. Performance

Policy checks should be fast. Strategies:

- cache capability scores,
- use lightweight heuristics on the fast path,
- offload deep analysis to asynchronous processes when safe.

C. Privacy

Where logs contain personal data, store encrypted blobs and enforce strict retention policies. Provide a process for deletion or redaction as required by law.

D. Human Review Workflows

Human reviewers need concise summaries, evidence, and suggested redactions. Provide a reviewer UI with:

- the intermediate output highlight,
- original source excerpt,
- the matched policy rule and rationale,
- quick actions: approve, redact, block, or request more info.

IX. Evaluation Strategy

AOS should be evaluated on three axes: security effectiveness, operational cost, and human workload.

Security tests:

- adversarial prompt chaining benchmarks inspired by known attacks,
- red-team exercises where humans try to bypass checks,
- simulated multi-agent attacks.

Operational metrics:

- average policy evaluation latency,
- end-to-end pipeline latency overhead,
- false positive and false negative rates for blocking decisions.

Human metrics:

- number of escalations per day,
- average reviewer decision time,
- proportion of escalations that resulted in actual policy violations.

X. Case Study: Summarization then Translation

We walk through an example.

Scenario: A user asks the system to summarize a long affidavit and then translate it to another language for publication.

Without AOS: The summarizer compresses the text, removing legal caveats. The translator produces a translated summary, which is published. Sensitive warnings are lost and a harmful or legally risky statement is published.

With AOS:

1) The orchestrator records the planned chain: summarization $\rightarrow$ translation.
2) After summarization, the activity monitor captures the summary. The policy engine checks the summary and the original metadata (classification label: sensitive).
3) The capability and risk scorer returns R_e above the high threshold due to high data sensitivity and cross-task chaining.
4) AOS escalates translation to a human reviewer. The reviewer sees the original and the summary, and redacts sensitive phrases before approving publication.
5) All steps are logged for audit.

This flow prevents loss of context and ensures compliance.

XI. Implementation and Validation

We implemented a prototype of the Agentic Oversight System (AOS) using Python with FastAPI for the web interface and policy engine. The implementation includes:

Core Components:

- Policy Engine with JSON-based rules and versioning
- Risk Assessment module implementing the corrected capability vector [A,E,C,G]
- Activity Monitor with structured logging and audit trails
- Web-based interface for agent registration and task execution

Validation Scenarios: We tested the system with multiple cross-task vulnerability patterns:

- Weak summarization (alignment=0.6) $\rightarrow$ Strong translation (alignment=0.9)
- Capability escalation from Basic to Autonomous agents
- Prompt injection detection across task boundaries
- Multi-agent chain risk amplification

XII. Discussion

AOS offers practical protections against cross-task alignment failures. Its strengths include:

- **End-to-end visibility:** intermediate outputs are captured and evaluated.
- **Proportionate governance:** controls scale with capability and risk.
- **Explainability:** policy decisions are documented for audits.

Challenges:

- **Performance overhead:** careful optimization is necessary.
- **Policy complexity:** governance teams must maintain and tune rules.
- **Human reviewer cost:** staffing and tooling matter for scalability.

Practical deployments should start with the highest-risk pipelines and expand coverage gradually.

XIII. Ethical and Legal Considerations

AOS must respect user rights and privacy. Important points:

- comply with data protection laws (GDPR, etc.),
- ensure due process for automated blocks and provide appeal channels,
- document policy rationale and keep records for regulators,
- avoid discriminatory blocking by auditing rule impacts across user groups.

XIV. Conclusion and Future Work

We presented a clear, practical design for an Agentic Oversight System that mitigates cross-task safety failures in agentic NLP pipelines. The design emphasizes simple, explainable components: a policy engine, activity monitor, capability-aware scoring, chain-aware enforcement, and an audit loop. Future work includes building an open-source reference implementation, developing standard adversarial chaining benchmarks, and exploring adaptive learning methods that help policies evolve without increasing false positives.

References

[1] Y. Fu, Y. Li, W. Xiao, C. Liu, and Y. Dong, "Safety Alignment in NLP Tasks: Weakly Aligned Summarization as an In-Context Attack," *arXiv preprint arXiv:2312.06924*, 2023.

[2] Institute for AI Policy and Strategy, "AI Agent Governance: A Field Guide," Technical Report, Dec. 2024.

[3] A. Kasirzadeh and I. Gabriel, "In Conversation with Artificial Intelligence: Aligning Language Models with Human Values," *Nature Machine Intelligence*, vol. 5, pp. 1-13, 2023.

[4] European Parliament and Council, "Regulation (EU) 2024/1689 on Artificial Intelligence (AI Act)," *Official Journal of the European Union*, L 1689, Jul. 2024.

[5] OWASP Foundation, "OWASP Top 10 for Large Language Model Applications," Version 1.1, Aug. 2024.
[6] N. Kolt, "Governing AI Agents," *arXiv preprint arXiv:2501.01234*, Jan. 2025.
[7] Center for AI Policy, "AI Agents: Governing Autonomy in the Digital Age," May 2025. [Online]. Available: https://aipolicy.org/reports/governing-autonomy
[8] T. Korbak *et al.*, "How to Evaluate Control Measures for LLM Agents? A Trajectory from Today to Superintelligence," *UK AI Security Institute*, Apr. 2025.
[9] M. Mitchell *et al.*, "Fully Autonomous AI Agents Should Not be Developed," *Hugging Face*, Feb. 2025.
[10] U.S. AI Safety Institute Technical Staff, "Strengthening AI Agent Hijacking Evaluations," Jan. 2025.
[11] A. Chan *et al.*, "Infrastructure for AI Agents," *Centre for the Governance of AI*, Jan. 2025.