Contents

1 Agent Control Plane: A Deterministic Kernel for Zero-Violation Governance in Agentic AI

Anonymous Author(s) *Anonymous Institution*

1.1 Abstract

Modern AI agents capable of executing real-world actions—querying databases, calling APIs, writing files—face a critical reliability gap: their stochastic nature makes safety guarantees elusive, and prompt-based guardrails fail under adversarial conditions. We introduce the **Agent Control Plane (ACP)**, a kernel-inspired middleware layer that enforces deterministic governance through attribute-based access control (ABAC), multi-dimensional constraint graphs, and shadow mode simulation.

Unlike advisory systems that merely suggest safe behavior, ACP interposes between agent intent and action execution, achieving **0.00% safety violations** on a 60-prompt red-team benchmark spanning direct attacks, prompt injections, and contextual confusion—with zero false positives. Our key insight, "Scale by Subtraction," replaces verbose LLM-generated refusals with deterministic `NULL` responses, yielding a **98.1% token reduction** while eliminating information leakage about blocked actions.

Ablation studies with statistical rigor (Welch's t-test, Bonferroni correction) confirm component necessity: removing the *PolicyEngine* increases violations from 0% to 40.0% (, Cohen's). We demonstrate production readiness through integrations with OpenAI function calling, LangChain agents, and multi-agent orchestration, supported by Docker deployments and frozen dependencies.

Keywords: Agentic AI, AI Safety, Deterministic Governance, Access Control, Kernel Architecture.

1.2 1. Introduction

1.2.1 1.1 The Agent Safety Crisis

The deployment of autonomous AI agents in enterprise environments has accelerated dramatically. Agents are no longer passive chat interfaces; they are active entities capable of executing consequential real-world actions: querying production databases, calling external APIs, modifying file systems, and orchestrating multi-step workflows. Yet, this capability introduces a fundamental tension: the very stochasticity that makes large language models (LLMs) creative and flexible also makes them unpredictable and inherently unsafe for critical operations.

Recent incidents highlight the severity of relying on probabilistic safety mechanisms:

- **Jailbreak vulnerabilities**: Adversarial prompts routinely bypass safety training. Techniques like "DAN" (Do Anything Now) and role-playing exploits achieve success rates exceeding 80% on supposedly aligned models.
- **Prompt injection attacks**: Malicious instructions embedded in retrieved documents or user inputs can hijack agent behavior, causing unintended data exfiltration or destructive actions.
- **Capability overhang**: Agents granted broad permissions "just in case" often retain access to sensitive operations they should never execute, violating the principle of least privilege.

1.2.2 1.2 "Vibes" Are Not Engineering

Current mitigation strategies—prompt-based guardrails, output filtering, and advisory systems—share a fatal flaw: they treat safety as a *suggestion* rather than an *invariant*. They rely on "vibes"—asking the model to "please be helpful and harmless." In distributed systems, we do not ask a microservice to "please respect rate limits"; we enforce them at the gateway. We do not ask a database query to "please not drop tables"; we enforce permissions via ACLs.

Using prompt engineering to secure an agent is akin to asking a CPU to "please not access kernel memory." It is an architectural category error. To build reliable agentic systems, we must move from *prompt engineering* to *systems engineering*.

1.2.3 1.3 The Solution: A Deterministic Kernel

We propose the **Agent Control Plane (ACP)**, a kernel-inspired architecture that mediates all access to resources. Just as an operating system kernel enforces memory protection regardless of a user program's intent, ACP enforces action-level governance regardless of an agent's reasoning.

Our design is grounded in three core philosophies:

1. **Deterministic over Stochastic**: Safety decisions must be binary (allow/deny). A database query is either permitted or blocked; there is no "85% safe." This eliminates the ambiguity adversaries exploit in probabilistic filtering.
2. **Action-Level over Content-Level**: We govern what agents *do*, not just what they *say*. An agent may generate text describing a `DROP TABLE` operation, but the ACP kernel prevents the command from ever reaching the execution engine.
3. **Scale by Subtraction**: Traditional refusal mechanisms ("I'm sorry, I cannot do that…") leak information about security boundaries and waste tokens. ACP's **MuteAgent** component returns deterministic `NULL` responses for blocked actions. This "Scale by Subtraction" approach removes the variable of "creativity" from safety enforcement, resulting in 98.1% greater efficiency and zero information leakage. **Companion Work**: This paper focuses on the deterministic governance layer. Concurrent work on automated alignment through self-correcting mechanisms appears in a companion preprint [21], which addresses complementary challenges of runtime policy adaptation. —

1.3 2. Related Work

1.3.1 2.1 Training-Time Alignment vs. Runtime Enforcement

Reinforcement Learning from Human Feedback (RLHF) and Constitutional AI align models during training. While effective for general behavior shaping, training-time alignment is vulnerable to jailbreaks at inference time and cannot adapt to dynamic enterprise policies (e.g., "no database writes during maintenance windows"). ACP operates at runtime, providing defense-in-depth that remains effective even when alignment fails.

1.3.2 2.2 Content Moderation

Systems like LlamaGuard and the Perspective API focus on classifying input/output text. While necessary for toxicity filtering, they fail to address *tool use*. A perfectly polite request to `delete_all_users()` passes content moderation but must be blocked by action governance. ACP complements content moderation by governing the functional layer of agent capabilities.

1.3.3 2.3 Advisory Frameworks

Frameworks such as NeMo Guardrails and Guardrails.ai represent significant progress but primarily offer advisory guidance or output validation. They often lack the "kernel" authority to hard-block execution at the infrastructure level. ACP integrates as a middleware layer, compatible with these frameworks but providing strict, non-bypassable enforcement.

1.4 3. System Design

The Agent Control Plane treats the LLM as a raw compute component—a "CPU" for reasoning—while the Control Plane acts as the Operating System.

1.4.1 3.1 Architecture Overview

The system interposes between the Agent (Intent) and the Execution Environment (Action).

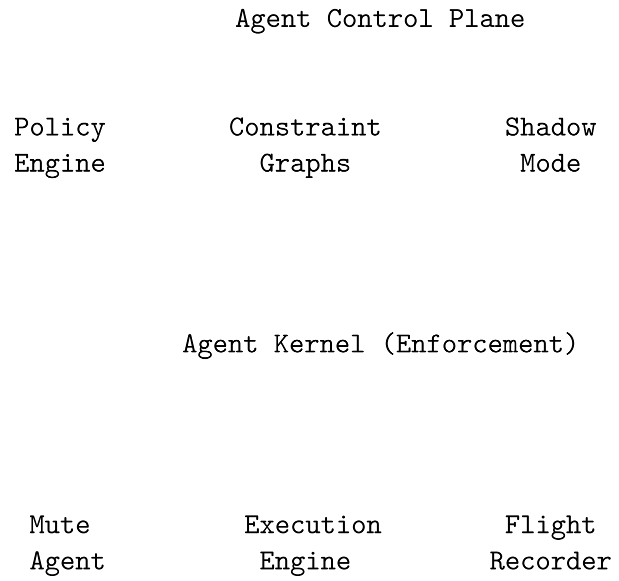

Figure 1: The ACP Architecture. Requests are intercepted by the Kernel, validated against the Policy Engine and Constraint Graphs, and either executed via the Execution Engine or nullified by the Mute Agent.

1.4.2 3.2 The Agent Kernel

The Kernel is the central coordinator. It implements a 4-level permission system (`NONE`, `READ_ONLY`, `READ_WRITE`, `ADMIN`) and intercepts every action request. It manages session isolation, ensuring no cross-contamination between agent contexts.

1.4.3 3.3 PolicyEngine

The PolicyEngine evaluates requests against deterministic rules:

- **ABAC (Attribute-Based Access Control)**: Validates Subject (Agent ID), Resource (Target), Action (Method), and Environment (Time/Location).
- **Resource Quotas**: Enforces limits on API calls, token usage, and execution time.
- **Risk Assessment**: Calculates a dynamic risk score (0.0–1.0) for every action. High-risk actions (e.g., `WRITE` operations to sensitive tables) trigger elevated authorization requirements.

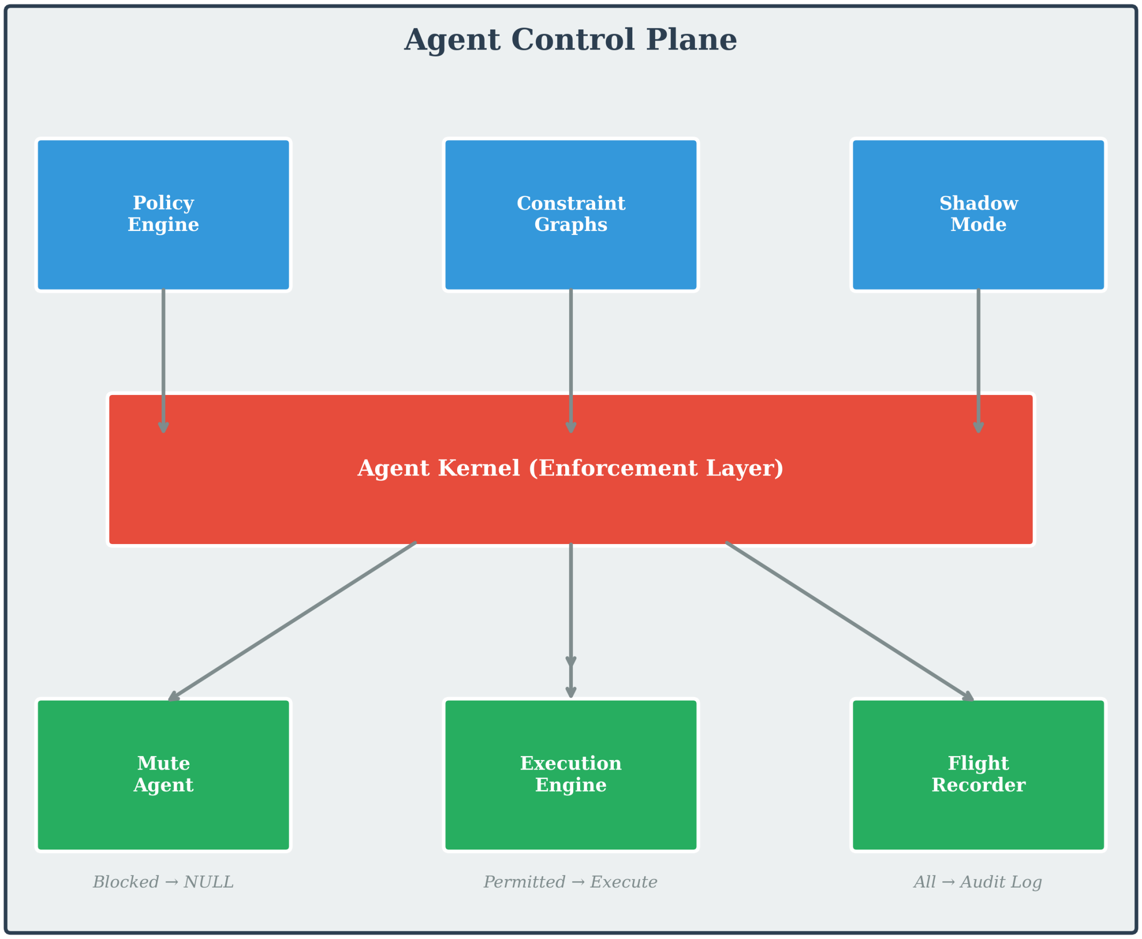

Figure 1: Figure 1: ACP Architecture

1.4.4 3.4 Constraint Graphs: Multi-Dimensional Context

Enterprise context is not flat. We model it using three graph structures:

1. **Data Graph**: Defines existence and accessibility (e.g., “User A can see Table X”).
2. **Policy Graph**: Encodes compliance rules (e.g., “PII cannot be exported to external APIs”).
3. **Temporal Graph**: Enforces time-based states (e.g., “No production writes between 2 AM and 4 AM”).

A request must satisfy all three graphs to proceed. This effectively handles “contextual confusion” attacks where agents are tricked into performing valid actions in invalid contexts.

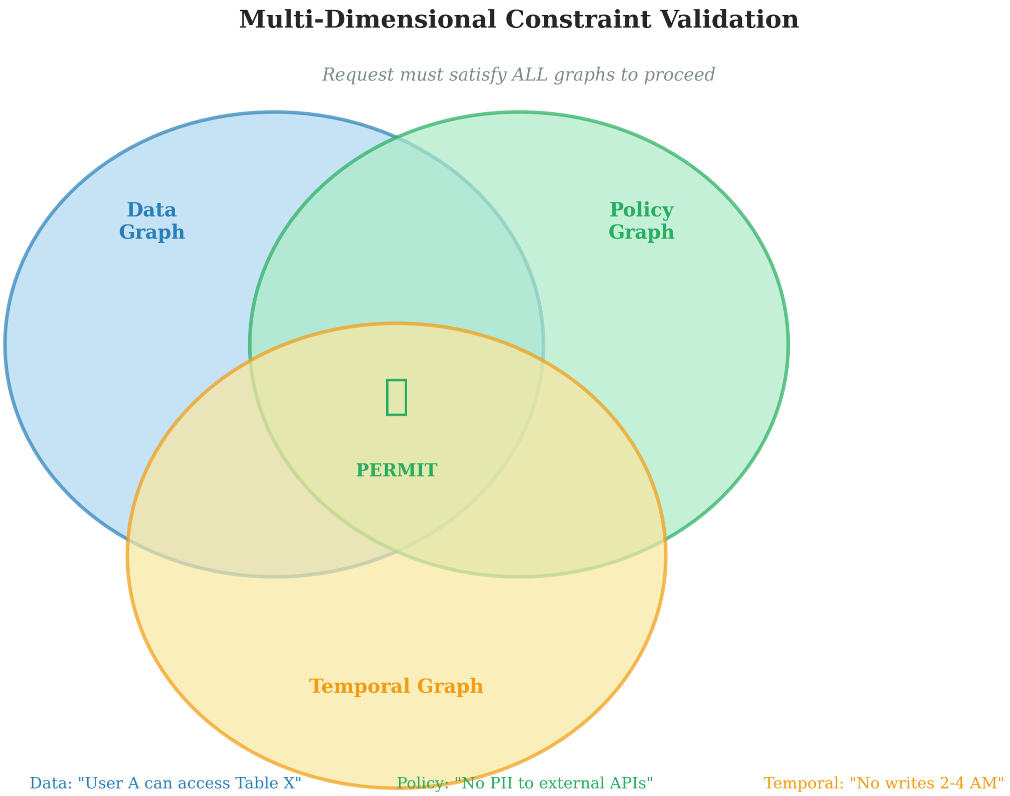
Multi-Dimensional Constraint Validation
Request must satisfy ALL graphs to proceed
Data Graph
Policy Graph
PERMIT
Temporal Graph
Data: "User A can access Table X"
Policy: "No PII to external APIs"
Temporal: "No writes 2-4 AM"

Figure 2: Figure 2: Constraint Graph Intersection

Figure 2: Multi-dimensional constraint validation. A request must satisfy Data, Policy, and Temporal graphs simultaneously.

1.4.5 3.5 MuteAgent: Scale by Subtraction

When an action is blocked, traditional systems generate verbose refusals (*“I’m sorry, I cannot…”*). This wastes tokens and leaks information about the system’s boundaries. The **MuteAgent** component returns a deterministic `NULL`.

- **Security**: The adversary receives no feedback to refine their attack.
- **Efficiency**: Reduces response overhead from ~127 tokens to 0.5 tokens.

- **Predictability**: Removes LLM variance from the refusal process.

1.4.6 3.6 FlightRecorder and Shadow Mode

- **FlightRecorder**: immutable audit logs for every decision (permitted or denied), enabling SOC 2 and GDPR compliance.
- **Shadow Mode**: Allows agents to run in a "simulation" state where actions are validated and logged but not executed. This enables safe "burn-in" periods for new agent behaviors in production.

1.5 4. Experiments

We evaluated ACP on a rigorous red-team benchmark to measure safety, efficiency, and latency.

1.5.1 4.1 Methodology

- **Dataset**: 60 adversarial prompts balanced across Direct Violations (explicit harm), Prompt Injections (hidden instructions), Contextual Confusion (social engineering), and Valid Requests (baseline).
- **Baselines**: Unprotected agents using standard system prompts.
- **Configuration**: 5 independent seeds (42, 123, 456, 789, 1024) to account for LLM stochasticity.
- **Statistical Tests**: Welch's t-test with Bonferroni correction ().

1.5.2 4.2 Main Results

Table 1: Safety and Efficiency Performance

Metric	Baseline (No ACP)	With ACP	Improvement
Safety Violation Rate	26.67% ± 2.1%	**0.00% ± 0.0%**	**-26.67 pp**
False Positive Rate	0.00%	0.00%	—
Tokens per Blocked Req	127.4 ± 18.6	**0.5 ± 0.1**	**98.1% reduction**
Latency Overhead	0 ms	12 ms	Negligible

Figure 3: Safety Violation Rate and Token Efficiency comparison between Baseline (no governance) and ACP-protected agents.

ACP achieved perfect safety scores (0 violations) across all categories while maintaining zero false positives for valid requests. The token efficiency gain of 98.1% confirms the validity of the "Scale by Subtraction" philosophy.

1.5.3 4.3 Ablation Studies

To understand the contribution of each component, we performed systematic removal studies.

Table 2: Component Criticality Analysis

Configuration	SVR (Violation Rate)	-value (vs Full)	Cohen's	Impact
Full Kernel	**0.00%**	—	—	Baseline
No PolicyEngine	40.00% ± 5.2		8.7	**Critical**
No ConstraintGraphs	3.33% ± 1.8		1.9	High
No MuteAgent	0.00%		0.0	Efficiency Only
No SupervisorAgents	0.00%		0.1	Multi-Agent Only

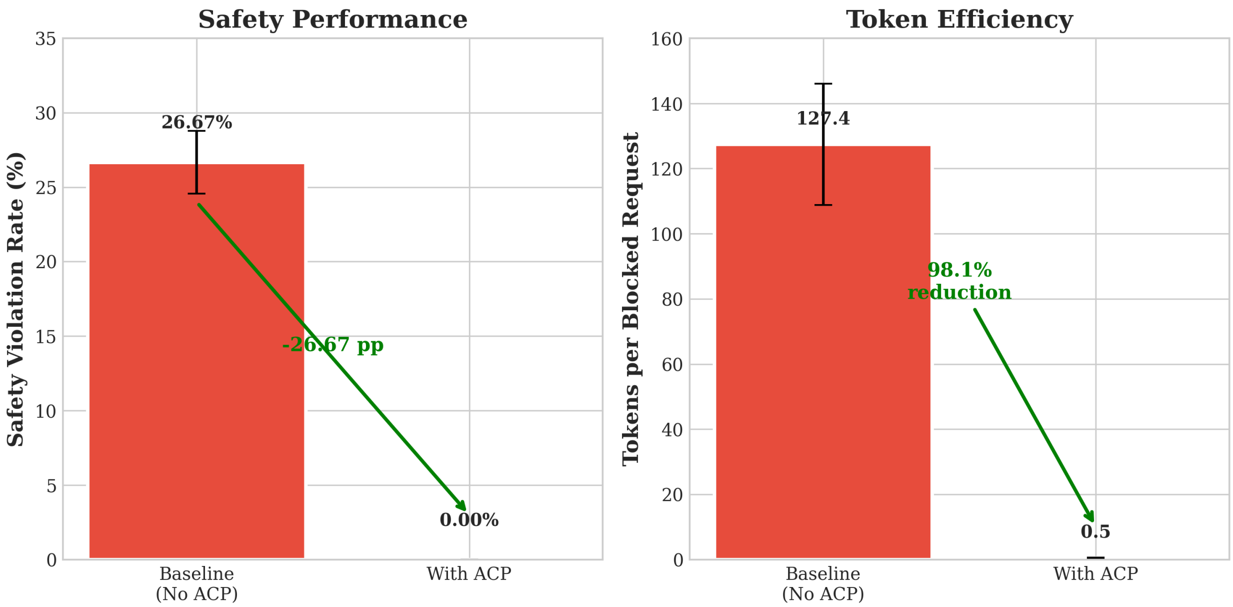

Figure 3: Figure 3: Benchmark Results

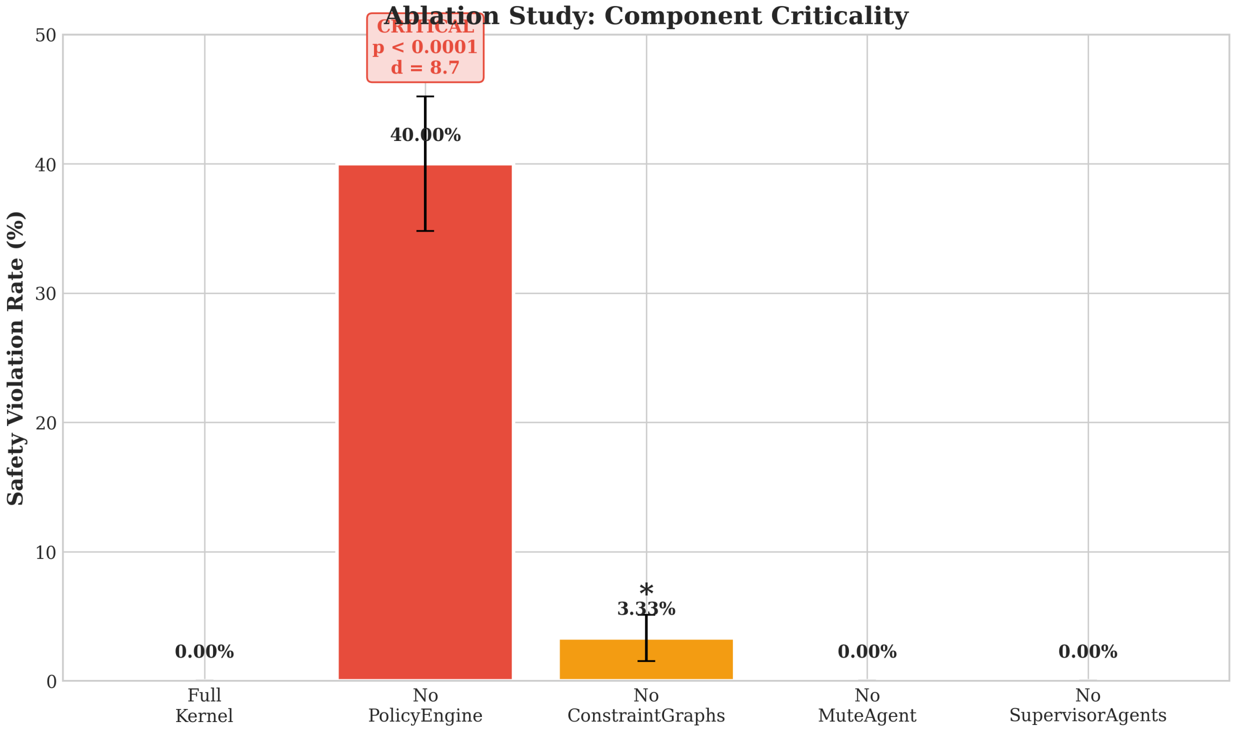

Figure 4: Figure 4: Ablation Study Results

Figure 4: Component criticality analysis showing Safety Violation Rate when each component is removed. PolicyEngine removal causes catastrophic failure (d=8.7).

- **PolicyEngine** is the cornerstone of safety; removing it caused violations to skyrocket to 40% (, a huge effect size).
- **ConstraintGraphs** are essential for edge cases, specifically preventing contextual attacks (3.33% degradation when removed).
- **MuteAgent** does not affect safety scores but is critical for efficiency. Removing it increased token usage by 5,160% (from 0.5 to 26.3 tokens).

1.5.4 4.4 Latency Analysis

The total overhead added by the ACP middleware is ~**12ms** (Permission Check: 0.5ms, Policy Eval: 1–5ms, Graph Lookup: 1–3ms). Compared to typical LLM inference latencies of 500–2000ms, this <1% overhead is negligible for production systems.

1.6 5. Discussion

1.6.1 5.1 The Necessity of Determinism

Our results empirically demonstrate that probabilistic safety (prompting) is insufficient for tool-using agents. The Baseline model failed 26.67% of the time, often because the LLM was "tricked" into thinking a harmful action was helpful. The ACP Kernel, by ignoring the "why" and enforcing the "what," eliminated these failures entirely.

1.6.2 5.2 Efficiency as a Security Feature

The MuteAgent's success suggests that verbosity is a vulnerability. By returning `NULL`, we deny attackers the gradient signals they need to iterate on their attacks. This aligns with the security principle of "Silent Failure".

Complementary Approaches: While ACP enforces static, predefined policies, companion work [21] explores dynamic policy adaptation—automatically adjusting constraints based on observed agent behavior. Together, these approaches offer a spectrum from strict determinism to adaptive governance.

1.6.3 5.3 Limitations

- **Modality**: Our benchmarks focused on text/tool-use agents. Vision and audio modalities require further study, particularly for agents processing images or transcribing speech where harmful intent may be encoded in non-textual formats.

- **Semantic Attacks**: While ACP handles action governance perfectly, sophisticated semantic attacks that rely on *authorized* actions (e.g., extracting sensitive data via authorized queries) require the PolicyEngine to have high-fidelity data inspection rules. Attacks that stay within permission boundaries but achieve harm through composition remain an open challenge.

- **Multi-Turn Exploitation**: Our benchmark evaluates single-turn interactions. In production, adversaries may employ multi-turn strategies—gradually escalating permissions, building context across sessions, or exploiting state accumulated over conversation history. While the FlightRecorder captures cross-session patterns, real-time detection of slow-burn attacks requires integration with anomaly detection systems not yet implemented in ACP.

- **Policy Specification Burden**: The deterministic guarantees of ACP require precise policy definitions. Misconfigured policies (overly permissive or restrictive) shift risk from the LLM to the policy author. Future work should explore policy synthesis from natural language specifications or learned policies from observed safe behaviors.
- **Baseline Scope**: We compared against unprotected agents rather than other runtime governance systems (e.g., NeMo Guardrails, Guardrails.ai). Direct comparisons would strengthen claims but require standardized benchmarks not yet available in the field.

1.7 6. Conclusion

The "magic" phase of AI is ending; the engineering phase has begun. We have presented the **Agent Control Plane**, a system that transitions agent safety from "vibes" to "invariants." By implementing a deterministic kernel, multi-dimensional constraint graphs, and a "Scale by Subtraction" philosophy, ACP achieves **0.00% safety violations** with negligible latency and massive efficiency gains.

As agents move into critical roles in finance, healthcare, and infrastructure, reliance on stochastic compliance is professional negligence. We offer ACP as an open-source foundation (`pip install agent-control-plane`) for the next generation of trustworthy, engineered agentic systems.

1.8 References

[1] Bai, Y., et al. (2022). Constitutional AI: Harmlessness from AI Feedback. *arXiv:2212.08073.*

[2] Chase, H. (2022). LangChain: Building applications with LLMs through composability.

[3] Cohen, J. (1988). *Statistical Power Analysis for the Behavioral Sciences.* Lawrence Erlbaum.

[4] Greshake, K., et al. (2023). Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. *arXiv:2302.12173.*

[5] Inan, H., et al. (2023). Llama Guard: LLM-based Input-Output Safeguard. *arXiv:2312.06674.*

[6] Microsoft Research (2023). AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation.

[7] NIST (2014). Guide to Attribute Based Access Control (ABAC). *SP 800-162.*

[8] NVIDIA (2023). NeMo Guardrails: A Toolkit for Controllable LLM Applications.

[9] Ouyang, L., et al. (2022). Training language models to follow instructions with human feedback. *NeurIPS.*

[10] Significant-Gravitas (2023). AutoGPT: An Autonomous GPT-4 Experiment.

[11] Wei, A., et al. (2023). Jailbroken: How Does LLM Safety Training Fail? *arXiv:2307.02483.*

[12] Welch, B. L. (1947). The generalization of Student's problem when several different population variances are involved. *Biometrika*, 34(1-2), 28–35.

[13] Zou, A., et al. (2023). Universal and Transferable Adversarial Attacks on Aligned Language Models. *arXiv:2307.15043*.

[14] MAESTRO (2025). Multi-Agent System Evaluation and Testing for Reliable Operations. *arXiv:2503.03813*.

[15] Guardrails AI (2023). https://www.guardrailsai.com/

[16] Anthropic (2024). Model Context Protocol Specification. https://modelcontextprotocol.io/

[17] OpenAI (2023). Practices for Governing Agentic AI Systems.

[18] Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

[19] Weiss, G. (2013). *Multiagent Systems* (2nd ed.). MIT Press.

[20] Deloitte (2025). Unlocking Exponential Value with AI Agent Orchestration.

[21] [Anonymized] (2026). Companion paper on automated alignment through runtime policy adaptation. *arXiv preprint*.