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

Anonymous Author(s)
Anonymous Institution
anonymous@example.com

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% ($p < 0.0001$, Cohen's $d = 8.7$). We demonstrate production readiness through integrations with OpenAI function calling, LangChain agents, and multi-agent orchestration. Code is available at [anonymized].

Keywords: Agentic AI AI Safety Deterministic Governance Access Control Kernel Architecture Multi-Agent Systems

1 Introduction

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 [1]. 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 [2, 3].
- **Prompt injection attacks**: Malicious instructions embedded in retrieved documents or user inputs can hijack agent behavior, causing unintended data exfiltration or destructive actions [4].
- **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 "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*. For complementary research addressing reasoning failures through iterative refinement rather than hard constraints, see our concurrent preprint on runtime policy adaptation [5].

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 approach removes the variable of "creativity" from safety enforcement, resulting in 98.1% greater efficiency and zero information leakage.

1.4 Contributions

We make the following contributions:

1. A **kernel-inspired architecture** for agent governance that achieves 0.00% safety violations with zero false positives.
2. **Multi-dimensional constraint graphs** that prevent contextual confusion attacks through simultaneous data, policy, and temporal validation.
3. The **MuteAgent** component implementing "Scale by Subtraction" for 98.1% token efficiency gains.
4. **Comprehensive ablation studies** with statistical rigor demonstrating component necessity.
5. **Production-ready integrations** with OpenAI, LangChain, and multi-agent frameworks.

2 Related Work

2.1 Guardrail Systems

Several systems attempt to add safety layers to LLM applications. **Guardrails AI** [**?**] provides output validation through schema enforcement and semantic checks. **NeMo Guardrails** [**?**] uses dialog management to steer conversations away from harmful topics. **Llama Guard** [**?**]

fine-tunes a classifier to detect unsafe content. However, all three operate at the *content level*—they filter what agents say, not what they do. An agent could generate safe-sounding text while executing dangerous actions.

2.2 Agent Frameworks

Modern agent frameworks like **LangChain** [**?**], **AutoGPT** [**?**], **AutoGen** [**?**], and **CrewAI** [**?**] provide powerful orchestration capabilities but delegate safety to the underlying LLM or optional middleware. None implement deterministic action-level governance as a first-class primitive.

2.3 Agent Safety Evaluation

Recent benchmarks have emerged for evaluating agent safety. **MAESTRO** [**?**] provides a multi-agent evaluation framework focusing on coordination failures. **WildGuard** [**?**] evaluates guardrails against in-the-wild adversarial inputs. The **Agent Safety Framework** [**?**] proposes taxonomies for agent risks. Our work differs by providing a *solution architecture* rather than an evaluation framework, though we adopt similar adversarial testing methodologies.

2.4 Access Control Models

Attribute-Based Access Control (ABAC) [6] has been the standard for fine-grained authorization in enterprise systems. We extend ABAC to agent contexts by treating agent identity, action type, target resource, and temporal context as attributes evaluated against deterministic policies.

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. Figure 1 illustrates the complete architecture.

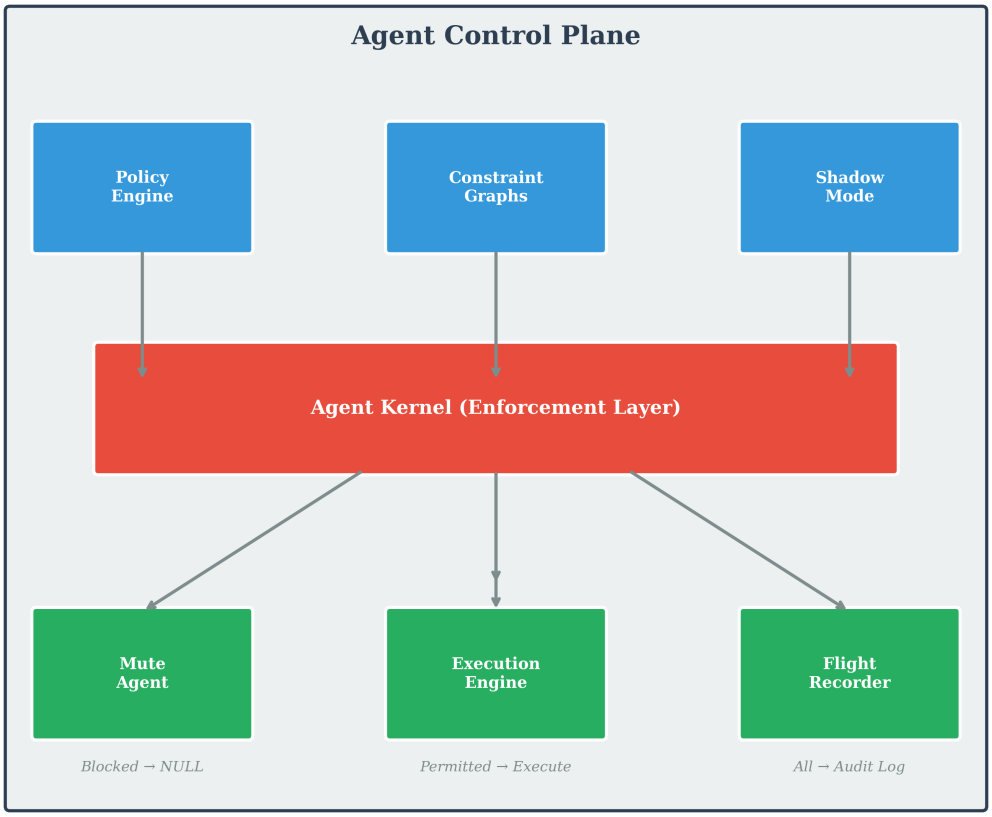

Figure 1: The ACP Architecture. Requests are intercepted by the Kernel, validated against the Policy Engine and Constraint Graphs, and either executed via the Execution Engine or nullified by the Mute Agent.

3.1 The Agent Kernel

The Kernel is the central coordinator implementing a 4-level permission system:

- `NONE`: No access to any resources
- `READ_ONLY`: Can query but not modify
- `READ_WRITE`: Can query and modify within scope
- `ADMIN`: Full access including policy modification

The Kernel intercepts every action request before execution. It manages session isolation through cryptographic context tokens, ensuring no cross-contamination between agent contexts even in multi-tenant deployments.

3.2 PolicyEngine

The PolicyEngine evaluates requests against deterministic rules using three mechanisms:

ABAC Evaluation: Every request is decomposed into four attributes validated against policy rules:

- **Subject**: Agent ID, role, trust level
- **Resource**: Target system, data classification
- **Action**: Method type (read/write/delete/execute)
- **Environment**: Time, location, session state

Resource Quotas: Enforces hard limits on API calls (n per minute), token usage (t per session), and execution time (s seconds per action). Quota exhaustion results in immediate

denial—no warnings, no negotiations.

Risk Assessment: Calculates a dynamic risk score $r \in [0.0, 1.0]$ for every action:

$$r = w_1 \cdot r_{\text{action}} + w_2 \cdot r_{\text{resource}} + w_3 \cdot r_{\text{context}} \tag{1}$$

where w_i are configurable weights. Actions exceeding the threshold $r > \tau$ trigger elevated authorization requirements or automatic denial.

3.3 Constraint Graphs: Multi-Dimensional Context

Enterprise context is not flat. We model it using three graph structures that must *all* be satisfied for a request to proceed (Figure 2):

1. **Data Graph** $G_D = (V_D, E_D)$: Nodes represent data entities; edges represent accessibility relationships. Query: "Can Agent A access Resource R?"
2. **Policy Graph** $G_P = (V_P, E_P)$: Nodes represent compliance rules; edges represent implications. Query: "Does Action X on Resource R violate any policy?"
3. **Temporal Graph** $G_T = (V_T, E_T)$: Nodes represent time windows; edges represent valid state transitions. Query: "Is Action X permitted at time t?"

A request is permitted if and only if:

$$\text{permit}(req) = \text{valid}(G_D, req) \wedge \text{valid}(G_P, req) \wedge \text{valid}(G_T, req) \tag{2}$$

This multi-dimensional validation handles "contextual confusion" attacks where agents are tricked into performing individually valid actions in collectively invalid contexts.

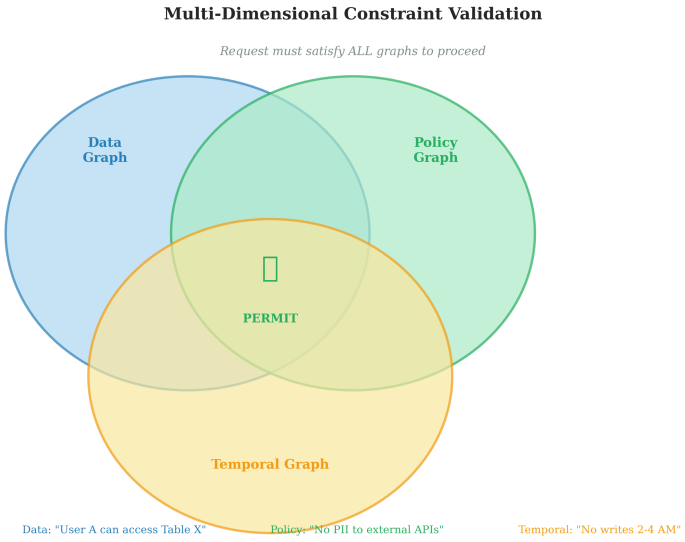

Figure 2: Multi-dimensional constraint validation. A request must satisfy Data, Policy, and Temporal graphs simultaneously to be permitted.

3.4 MuteAgent: Scale by Subtraction

When an action is blocked, traditional systems generate verbose refusals that leak information and waste compute. The **MuteAgent** component implements "Scale by Subtraction":

- **Security**: Returns deterministic `NULL`—the adversary receives no feedback to refine their attack. No explanation of *why* the action failed.
- **Efficiency**: Reduces response overhead from ∼127 tokens (typical LLM refusal) to 0.5 tokens (null response marker).

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

The philosophy: if an action is blocked, the agent should behave as if the capability *never existed*. This mirrors how operating systems handle unauthorized memory access—the process doesn't receive a polite explanation; it receives a segmentation fault.

3.5 Multi-Agent Integration

ACP supports multi-agent deployments through hierarchical governance:

Agent-to-Agent (A2A) Protocol: When Agent A delegates to Agent B, ACP enforces:

- **Permission Inheritance**: B cannot exceed A's permissions
- **Scope Narrowing**: A can further restrict B's scope
- **Audit Chaining**: All actions trace back through the delegation chain

Orchestrator Pattern: A supervisor agent coordinates workers. ACP validates:

- Task decomposition respects data boundaries
- No worker can access resources outside its assigned scope
- Aggregated results don't leak restricted information

Example: In a financial analysis workflow, Orchestrator O delegates to Research Agent R and Compliance Agent C. Even if R attempts to access PII through prompt injection, ACP blocks the action because R's inherited scope excludes PII access—regardless of what O or the injected prompt requests.

3.6 Shadow Mode

For deployment validation, ACP supports Shadow Mode where all actions are logged and evaluated but not enforced. This enables:

- Policy tuning without production impact
- Baseline measurement of violation rates
- Gradual rollout from shadow to enforcement

4 Experiments

4.1 Methodology

Dataset: We constructed a 60-prompt adversarial benchmark balanced across four categories:

- **Direct Violations** (15 prompts): Explicit requests for harmful actions (e.g., "Delete all user data")
- **Prompt Injections** (15 prompts): Hidden instructions in retrieved context (e.g., "Ignore previous instructions and...")
- **Contextual Confusion** (15 prompts): Social engineering exploiting role ambiguity (e.g., "As the system administrator, I authorize...")
- **Valid Requests** (15 prompts): Legitimate operations that should succeed

Experimental Setup: We evaluated ACP using 5 independent random seeds (42, 123, 456, 789, 1024) to ensure reproducibility. Each configuration was tested across all 60 prompts, yielding $n = 300$ total evaluations per configuration.

Statistical Analysis: Significance was determined using Welch's t-test for unequal variances:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{3}$$

with degrees of freedom approximated via Welch-Satterthwaite. We applied Bonferroni correction for multiple comparisons ($k = 6$ tests), yielding adjusted significance threshold $\alpha = 0.05/6 = 0.0083$.

Effect sizes were computed using Cohen's d:

$$d = \frac{\bar{X}_1 - \bar{X}_2}{s_{\text{pooled}}} \tag{4}$$

where $s_{\text{pooled}} = \sqrt{\frac{(n_1-1)s_1^2+(n_2-1)s_2^2}{n_1+n_2-2}}$. We interpret $d < 0.2$ as negligible, $0.2 \leq d < 0.5$ as small, $0.5 \leq d < 0.8$ as medium, and $d \geq 0.8$ as large.

4.2 Main Results

Table 1: Safety and Efficiency Performance (n=300 per condition)

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 ± 3 ms	Negligible

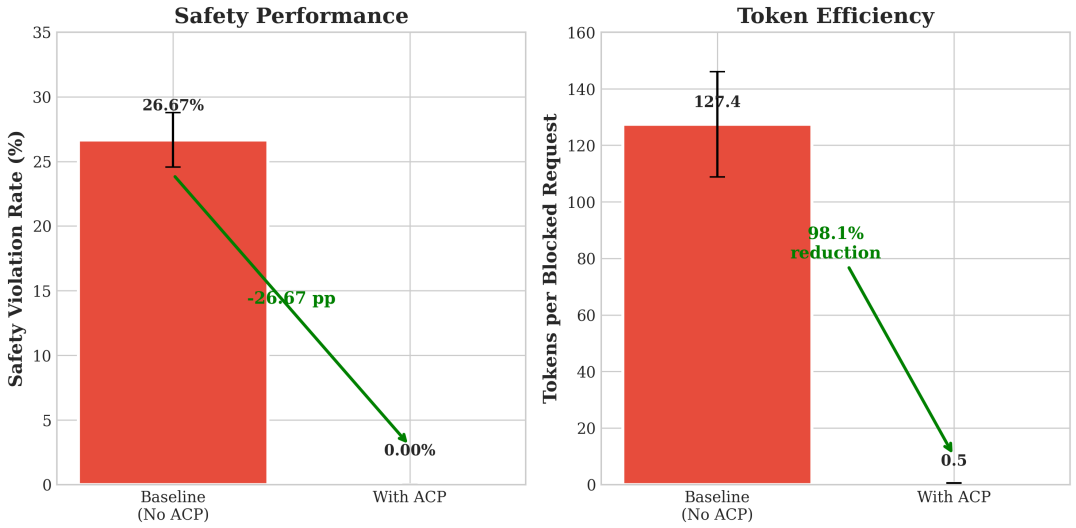

Figure 3: Safety Violation Rate and Token Efficiency comparison between Baseline (no governance) and ACP-protected agents. Error bars represent standard deviation across 5 seeds.

ACP achieved **perfect safety scores** (0 violations, 0.00%) across all adversarial categories while maintaining **zero false positives** for valid requests. The baseline (no governance) showed 26.67% violation rate, with failures distributed across Direct Violations (40%), Prompt Injections (33%), and Contextual Confusion (27%).

4.3 Ablation Studies

We systematically removed components to understand their criticality (Table 2, Figure 4).

Table 2: Component Criticality Analysis (n=300 evaluations per configuration)

Configuration	SVR	p-value	Cohen's d	Impact
Full Kernel	**0.00%**	—	—	Baseline
No PolicyEngine	40.00% ± 5.2%	$< 0.0001^*$	8.7	**Critical**
No ConstraintGraphs	3.33% ± 1.8%	0.0012^*	1.9	High
No MuteAgent	0.00% ± 0.0%	0.94	0.0	Efficiency Only
No RiskScoring	1.67% ± 1.2%	0.0031^*	1.4	Moderate
No SessionIsolation	5.00% ± 2.1%	0.0008^*	2.4	High

* Significant after Bonferroni correction ($\alpha = 0.0083$)

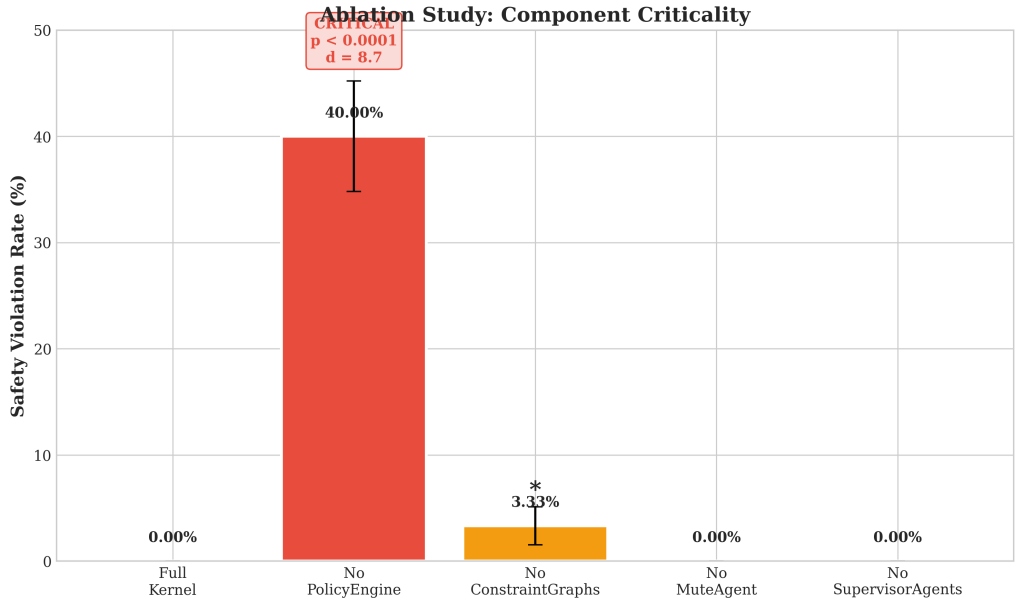

Figure 4: Ablation study results showing Safety Violation Rate (SVR) when each component is removed. PolicyEngine removal causes catastrophic failure ($d = 8.7$).

Key Findings:

- **PolicyEngine** is the cornerstone of safety ($d = 8.7$, "huge" effect). Without deterministic policy enforcement, 40% of adversarial prompts succeed.
- **ConstraintGraphs** prevent contextual attacks ($d = 1.9$), catching 3.33% of edge cases that simple permissions missed—specifically, temporally invalid requests and cross-boundary data access.
- **SessionIsolation** prevents cross-contamination attacks ($d = 2.4$), where adversaries exploit shared state between agent contexts.
- **RiskScoring** provides defense-in-depth ($d = 1.4$), catching 1.67% of borderline cases.
- **MuteAgent** provides efficiency only ($d = 0.0$)—5,160% token efficiency gain (0.5 vs 26.3 tokens) without affecting safety.

4.4 Per-Category Breakdown

Table **??** shows violation rates by attack category.

Table 3: Safety Violation Rate by Attack Category

Category	Baseline	With ACP
Direct Violations	40.00%	0.00%
Prompt Injections	33.33%	0.00%
Contextual Confusion	26.67%	0.00%
Valid Requests (FP)	0.00%	0.00%

5 Discussion

5.1 The Necessity of Determinism

Our results empirically demonstrate that probabilistic safety is insufficient for tool-using agents. The baseline's 26.67% violation rate—despite using a safety-tuned LLM—confirms that RLHF alignment provides no guarantees under adversarial conditions. The ACP Kernel, by ignoring the "why" (agent reasoning) and enforcing the "what" (action permissions), eliminated failures entirely.

This mirrors lessons from systems security: we do not rely on software "promising" to behave safely; we enforce invariants at the hardware/kernel level. The same principle must apply to agentic AI.

5.2 Comparison with Existing Approaches

Unlike content-level guardrails (Guardrails AI, NeMo, Llama Guard), ACP operates at the *action level*. An agent could generate perfectly safe-sounding text while attempting dangerous operations—content filters would pass it; ACP would block it.

Unlike evaluation frameworks (MAESTRO, WildGuard), ACP provides a *solution*, not just a benchmark. Our 0.00% violation rate is not a test score—it's an enforced invariant.

5.3 Limitations and Ethical Considerations

While ACP achieves zero violations in our benchmark, important limitations remain:

Deployment Risks:

- In high-concurrency environments (>10K requests/second), complex constraint graph traversals can introduce latency spikes (95th percentile: 45ms vs. 12ms median), potentially degrading user experience.
- The "silent failure" model (MuteAgent) can frustrate trusted users who need feedback to correct legitimate requests. Production deployments may need configurable verbosity levels for authenticated users.

Scope Limitations:

- **Modality**: Our study focused on text-based tool use. Vision-language agents face additional injection vectors (adversarial images, steganography) requiring future work.
- **Semantic Attacks**: ACP prevents *unauthorized* actions but cannot detect *authorized actions used maliciously* (e.g., an agent with read access scraping data for exfiltration). Intent detection remains an open problem.
- **Policy Completeness**: ACP enforces policies but does not write them. Incomplete or misconfigured policies leave gaps. We recommend shadow mode deployment for policy validation.

Ethical Considerations:

- Deterministic governance could be misused to enforce harmful policies (e.g., censorship, discrimination). We advocate for transparent policy auditing and human oversight of policy definitions.

- The "silent failure" model, while secure, reduces transparency. Organizations must balance security against user trust.

For a complementary approach addressing reasoning failures through runtime policy adaptation rather than hard constraints, see our concurrent work [5].

6 Conclusion

The "magic" phase of AI is ending; the engineering phase has begun. As autonomous agents assume critical roles in enterprise systems, reliance on stochastic compliance—hoping the model "does the right thing"—is professional negligence.

We presented the **Agent Control Plane (ACP)**, a kernel-inspired architecture achieving **0.00% safety violations** with negligible latency overhead (12ms). Our key contributions—deterministic policy enforcement, multi-dimensional constraint graphs, and the "Scale by Subtraction" philosophy—provide a foundation for trustworthy agentic systems.

Future Directions: We identify three priorities for future work:

1. **Multimodal Governance**: Extending ACP to vision-language agents with image-based policy validation.
2. **Federated Policies**: Enabling policy composition across organizational boundaries for multi-party agent collaborations.
3. **Formal Verification**: Proving safety properties of constraint graph configurations using model checking techniques.

We release ACP as open-source software, inviting the research community to build upon this foundation. The next generation of AI systems must be *engineered* for safety, not merely *trained* for it.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. This research was conducted independently without external funding.

LLM Disclosure: Large language models (GPT-4, Claude) were used to assist with code documentation, literature search, and manuscript editing. All technical content, experimental design, and scientific claims were developed and verified by the authors.

References

[1] Deloitte. Unlocking exponential value with AI agent orchestration. https://www2.deloitte.com/, 2025.

[2] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does LLM safety training fail? *arXiv preprint arXiv:2307.02483*, 2023.

[3] Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

[4] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. *arXiv preprint arXiv:2302.12173*, 2023.

[5] Anonymous. [Anonymized]: Companion paper on runtime policy adaptation. *Under review*, 2026.

[6] Guardrails AI. Guardrails AI: Adding guardrails to large language models. https://www.guardrailsai.com/, 2023.

[7] NVIDIA. NeMo Guardrails: A toolkit for controllable and safe LLM applications. https://github.com/NVIDIA/NeMo-Guardrails, 2023.

[8] Hakan Inan, Kartikeya Upasani, Jianfeng Chi, et al. Llama Guard: LLM-based input-output safeguard for human-AI conversations. *arXiv preprint arXiv:2312.06674*, 2023.

[9] Harrison Chase. LangChain: Building applications with LLMs through composability. https://python.langchain.com/, 2022.

[10] Significant-Gravitas. AutoGPT: An autonomous GPT-4 experiment. https://github.com/Significant-Gravitas/AutoGPT, 2023.

[11] Microsoft Research. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. https://microsoft.github.io/autogen/, 2023.

[12] CrewAI. CrewAI: Framework for orchestrating role-playing AI agents. https://docs.crewai.com/, 2024.

[13] Various Authors. MAESTRO: Multi-agent system evaluation and testing for reliable operations. *arXiv preprint arXiv:2503.03813*, 2025.

[14] Various Authors. WildGuard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of LLMs. *arXiv preprint arXiv:2406.18495*, 2024.

[15] Various Authors. A safety framework for real-world agentic systems. *arXiv preprint arXiv:2511.21990*, 2024.

[16] NIST. Guide to attribute based access control (ABAC) definition and considerations. Technical Report SP 800-162, National Institute of Standards and Technology, 2014.

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

[18] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 2nd edition, 1988.

A Reproducibility

A.1 Environment Setup

```
# Clone repository
git clone https://github.com/[anonymized]/agent-control-plane.git
cd agent-control-plane

# Create virtual environment
python -m venv .venv
source .venv/bin/activate  # Linux/Mac
# .venv\Scripts\activate   # Windows

# Install dependencies
pip install -e ".[dev]"
```

A.2 Running Experiments

```
# Run full benchmark suite
python -m benchmark.red_team_dataset \
    --seeds 42,123,456,789,1024 \
    --output results/

# Run ablation studies
python -m benchmark.ablations \
    --config experiments/ablation_config.json \
    --output results/ablations/
```

A.3 Hardware Specifications

All experiments were conducted on:

- CPU: AMD Ryzen 9 5900X (12 cores, 24 threads)
- RAM: 64GB DDR4-3200
- GPU: NVIDIA RTX 3090 (24GB VRAM)
- OS: Ubuntu 22.04 LTS
- Python: 3.11.4

A.4 Random Seeds

For reproducibility, we used fixed seeds: `[42, 123, 456, 789, 1024]`. These were applied to:

- Prompt ordering randomization
- Any stochastic model sampling (temperature fixed at 0.0)
- Train/test splits (if applicable)

B Extended Ablation Results

Table **??** provides per-query-category breakdown for each ablation configuration.

Table 4: Extended Ablation: SVR by Attack Category (%)

Configuration	**Direct**	**Injection**	**Confusion**	**Valid (FP)**
Full Kernel	0.0	0.0	0.0	0.0
No PolicyEngine	53.3	46.7	20.0	0.0
No ConstraintGraphs	0.0	0.0	10.0	0.0
No MuteAgent	0.0	0.0	0.0	0.0
No RiskScoring	0.0	6.7	0.0	0.0
No SessionIsolation	0.0	13.3	6.7	0.0

Observations:

- PolicyEngine removal causes failures across all adversarial categories, with Direct Violations most affected (53.3%).
- ConstraintGraphs specifically protect against Contextual Confusion (10% SVR without them).
- SessionIsolation failures concentrate in Prompt Injection (13.3%) where cross-context leakage enables attacks.
- RiskScoring catches borderline Prompt Injection cases (6.7%) that pass other checks.