The Probabilistic Paradox

Architecting Deterministic Safety into Enterprise AI Agents

Navigating the tension between Generative AI fluidity and Enterprise Rigidity

Aniruddha Mitra

Data and Analytics Specialist adn Evangelist,
Google Cloud Platform

Table of Contents

Executive Summary

The modern enterprise stands at a precipice defined by a fundamental contradiction: the **Probabilistic Paradox**. On one side lies the foundational bedrock of corporate operations—deterministic systems. These systems, comprising general ledgers, compliance engines, and Service Level Agreements (SLAs), operate on the binary principles of exactitude. In this world, a transaction is either valid or invalid; a balance sheet balances to the penny; and regulatory compliance is a pass/fail state. There is no "hallucination" in double-entry bookkeeping.

On the other side surges the transformative wave of Generative AI and Agentic systems. These engines are fundamentally probabilistic. Large Language Models (LLMs) do not "know" facts in the database sense; they predict the next token based on statistical likelihood distributions. They are stochastic, creative, and inherently uncertain. When an enterprise integrates an AI agent, it is attempting to embed a probabilistic engine—which might do the right thing—into a deterministic framework that demands it *must* do the right thing.

This report argues that building enterprise-grade agents is not merely an engineering challenge of prompt optimization; it is the structural challenge of **wrapping a probabilistic engine in a deterministic safety harness**. We posit that the solution lies in a "Defense in Depth" architecture where the LLM is treated not as the decision-maker, but as an untrusted reasoning component within a strictly governed computational graph.

This document provides an exhaustive analysis of this paradox and details a reference architecture using Google Cloud Platform (GCP) to resolve it. By leveraging services such as **Vertex AI**, **Cloud Workflows**, **Model Armor**, and **BigQuery**, we outline how to construct a "safety harness" that enforces binary constraints on stochastic outputs, ensuring that the immense power of AI agents can be safely tethered to the rigid requirements of the enterprise.

Who Is This For?

This research is critical for three primary groups operating within regulated or high-stakes environments:

Enterprise Architects and CTOs: Those responsible for moving AI from "innovation labs" to production environments where failure has financial or legal consequences. It addresses the architectural patterns required to integrate non-deterministic AI components with legacy deterministic systems (ERPs, Core Banking, EHRs).

Compliance and Risk Officers (CISOs): Stakeholders who must audit AI behaviors. This report provides the vocabulary and structural frameworks (such as "The Guardian Agent" and "Deterministic Harnesses") needed to approve AI deployments in sectors like Finance, Healthcare, and Legal Services.

AI Platform Engineers: Technical practitioners building on Google Cloud who need concrete implementation details for safety layers, specifically using tools like Vertex AI Agent Builder, Cloud Workflows, and BigQuery for auditability.

Why Is This Important?

As enterprises graduate from simple chatbots to agentic workflows that can take action (e.g., process refunds, approve loans, triage patients), the cost of error shifts from social embarrassment to operational liability.

The "Stochastic Drift" Risk: Without a safety harness, a multi-step agentic workflow can drift significantly from its intended logic path. A 1% error rate in step 1 of a 5-step process can compound into a catastrophic failure by step 5.

Regulatory Survival: Emerging frameworks like the EU AI Act and industry standards (e.g., HIPAA, SOX) demand explainability and reproducibility. A "black box" probabilistic decision is often legally indefensible. This report outlines how to wrap that black box in transparent, audit-ready code.

Operationalizing Trust: The "Probabilistic Paradox" is currently the primary blocker to enterprise AI adoption. Solving it—by guaranteeing that an agent will either be correct or safely abort to a human—is the only way to unlock the trillion-dollar potential of agentic AI in the Global 2000.

Part 1: The Anatomy of the Paradox

1.1 The Deterministic Foundations of the Enterprise

To understand the friction introduced by AI, one must first appreciate the rigidity of the enterprise environment. Corporate existence is predicated on predictability. This is not a preference; it is a legal and operational mandate.

1.1.1 The Binary Nature of Compliance

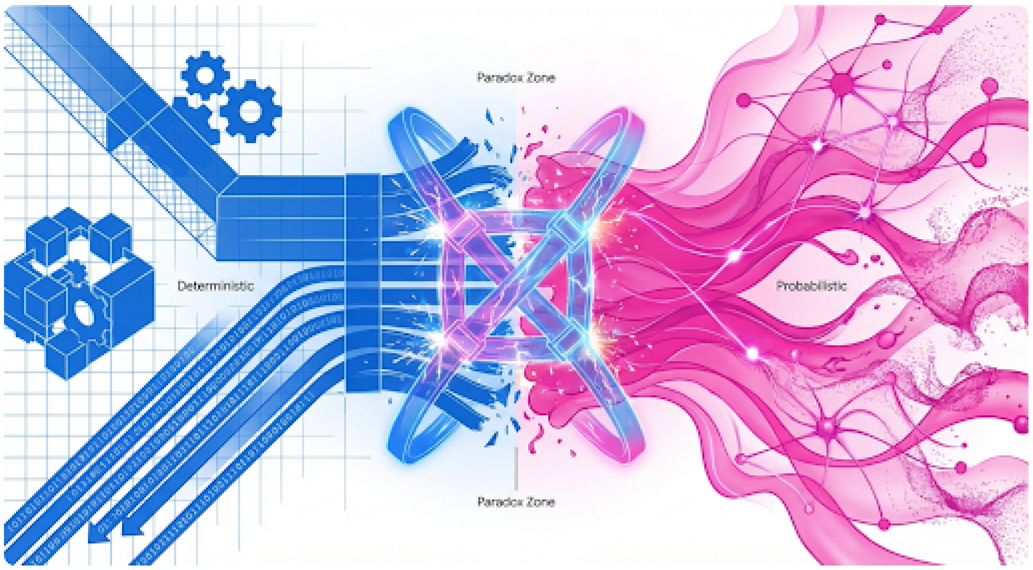
Regulatory frameworks, such as the General Data Protection Regulation (GDPR), the Health Insurance Portability and Accountability Act (HIPAA), and the emerging EU AI Act, are deterministic. A piece of data is either Personally Identifiable Information (PII) or it is not. A system either has the user's consent or it does not. There is no "80% probability of compliance."

In financial services, the "Probabilistic Paradox" is most acute. The snippets reviewed highlight that financial audits require exact explainability. If a loan application is denied, the institution must point to the specific deterministic rule (e.g., "Debt-to-Income ratio > 40%") that triggered the denial. An AI agent that denies a loan because "the vector embedding of the applicant's profile effectively mapped to a high-risk cluster in latent space" offers no legal defense. The enterprise requires **causality**, while the AI offers **correlation**.

1.1.2 The SLA Imperative

Service Level Agreements (SLAs) are the contractual manifestation of determinism. They guarantee specific outcomes within specific timeframes (e.g., "99.9% uptime," "response within 200ms"). Probabilistic models, by their nature, introduce variability. The "time to first token" or the logical path an agent takes can vary based on the stochastic seed or the complexity of the input.

The Probabilistic Paradox: A Collision of Systems



The enterprise is built on a foundation of rigid, binary logic (left), while AI agents operate on fluid, statistical probabilities (right). The 'Paradox Zone' is where these systems interact, creating risks of hallucination, non-compliance, and unpredictability.

1.2 The Probabilistic Engine: Understanding the Beast

At the core of the agent is the Large Language Model (LLM). As noted in source , LLMs are "probabilistic beasts" that predict the next word in a sequence. They do not retrieve facts; they reconstruct them based on statistical weights.

1.2.1 The Mechanism of Hallucination

"Hallucination" is a misnomer; the model is simply doing what it was trained to do—generate a plausible continuation. In an enterprise context, "plausible" is dangerous. If an accounting agent is asked to sum a column of numbers, a probabilistic model might generate a number that *looks* like a total (e.g., it ends in.00 and is larger than the addends) but is mathematically incorrect.

This is the **Stochastic Drift**. In a multi-step agentic workflow (Chain of Thought), a minor probabilistic error in step 1 (e.g., misinterpreting a tax code) compounds in step 2 and 3, leading to a confidently wrong conclusion. The enterprise cannot tolerate this drift in critical functions.

1.2.2 The "Creativity" vs. "Accuracy" Trade-off

Enterprises often want it both ways: they want the creative reasoning of an agent to solve complex customer service issues , but the rigid accuracy of a database for transaction processing. The research highlights that probabilistic AI excels in ambiguity (unstructured text, intent recognition) but fails in exactitude (math, strict logic enforcement). The architectural challenge, therefore, is not to force the LLM to be a calculator, but to prevent it from *trying* to be one.

1.3 The Failure Modes of Unbounded Agents

When a probabilistic engine is deployed without a safety harness, specific failure modes emerge that are incompatible with enterprise risk appetites:

1. **The "Confident Lie" in Advisory Services:** An agent effectively "invents" a policy clause to satisfy a user's request for a refund. In a deterministic system, a NULL result is returned if no policy exists. An LLM often fills the void with a fabrication.
2. **Non-Deterministic Compliance:** Running the same compliance check on the same document ten times might yield ten slightly different justifications. For an auditor , this lack of reproducibility is a failure of internal controls.
3. **Prompt Injection and Jailbreaking:** Deterministic code paths are hard to "trick" unless there is a bug. Probabilistic models, however, are susceptible to semantic hacking—using language to bypass safety filters.

Part 2: The Theory of Containment — Wrapping the Engine

To resolve the paradox, we must adopt a new architectural philosophy. We do not try to make the AI deterministic; that is impossible. Instead, we **wrap** the probabilistic component in a deterministic harness.

2.1 The Harness Concept

The "Harness" is a set of deterministic software components (code, logic, databases) that surround the LLM. It acts as a filter for inputs and a validator for outputs.

- **Input Harness (Pre-Computation):** Deterministic rules that sanitize data, enforce identity, and select the specific tools the agent is *allowed* to see.
- **Execution Harness (The Rails):** Rigid orchestration logic (State Machines) that dictate the *process* the agent must follow, even if the *content* of the steps is generated by AI.

- **Output Harness (Post-Computation):** Deterministic validation of the agent's output. This includes schema validation (JSON compliance), grounding checks (fact verification), and safety scans (DLP).

2.2 Grounding: The Anchor of Truth

Grounding is the primary mechanism for tethering the agent to reality. It transforms the agent from a "creative writer" to a "librarian."

2.2.1 RAG as a Deterministic Constraint

Retrieval-Augmented Generation (RAG) is often viewed as a way to give the model "knowledge." More importantly, it is a constraint mechanism. By forcing the model to answer *only* using retrieved context , we drastically reduce the probability space of the output.

However, RAG alone is insufficient. The retrieval itself is probabilistic (vector similarity). To make it a "safety harness," we must introduce **Confidence Scoring**. As detailed in snippet , the *Check Grounding API* provides a support score (0.0 to 1.0).

- **The Deterministic Rule:** IF Grounding_Score < 0.95 THEN Fallback_To_Human.
- This simple logic gate converts a probabilistic output into a deterministic workflow decision.

2.3 The "Sandwich" Architecture

The most effective pattern for wrapping the engine is the **Sandwich Architecture**.

1. **Top Bun (Deterministic):** The user's request is received. Identity is verified (IAM). A deterministic router decides if this request *needs* AI or if it's a simple lookup.
2. **The Meat (Probabilistic):** The AI Agent reasons, plans, and generates a response or tool call.
3. **Bottom Bun (Deterministic):** The output is intercepted.
   - *Syntax Check:* Is it valid JSON?
   - *Semantic Check:* Does it violate safety policies? (Model Armor).
   - *Grounding Check:* Is it supported by facts?
   - *Action:* Only if all checks pass is the action executed.

This architecture ensures that no probabilistic "thought" ever becomes a real-world "action" without passing through a deterministic checkpoint.

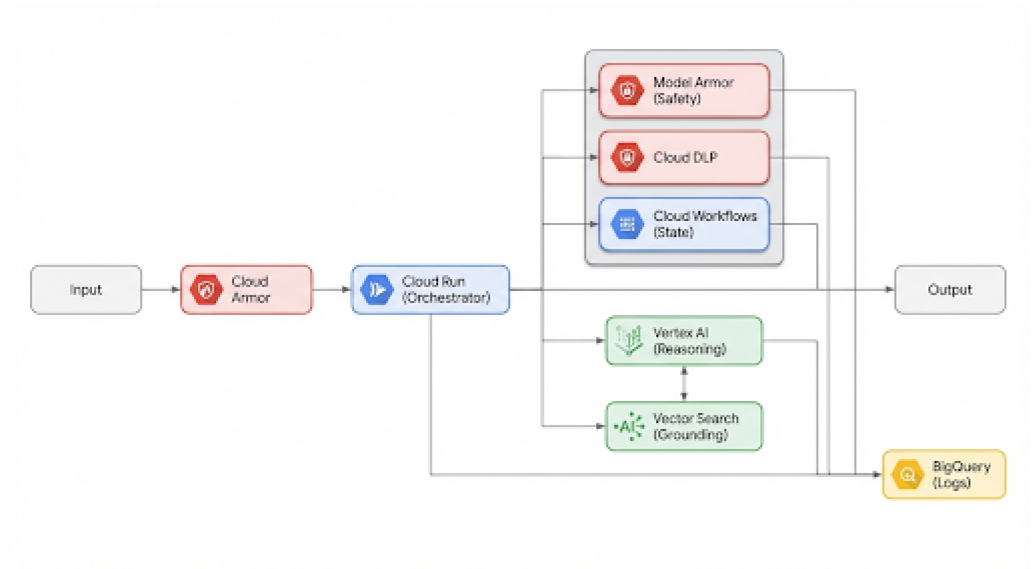
Part 3: Detailed Architecture on GCP — Wrapping the Engine

This section details how to implement the "Safety Harness" using Google Cloud Platform's specific services. We will move beyond high-level concepts to specific component interactions.

3.1 The Architecture Overview

The proposed architecture utilizes **Vertex AI** as the core reasoning engine, but heavily relies on **Cloud Run**, **Cloud Workflows**, and **BigQuery** to provide the deterministic wrapper.

GCP Reference Architecture: The Deterministic Safety Harness



The architecture separates the 'Probabilistic Core' (Vertex AI) from the 'Deterministic Wrapper' (Cloud Run & Workflows). All inputs and outputs pass through safety layers (Model Armor, DLP) before execution.

3.2 Component 1: The Orchestrator (The Wrapper)

Service: Cloud Run running a custom Python/Go application using the **Agent Development**

Kit (ADK).

While **Vertex AI Agent Builder** offers a low-code path, true enterprise control usually requires the flexibility of a custom container. Cloud Run acts as the "Body" of the agent. It holds the session state, manages the API connections, and crucially, executes the *logic* that evaluates the LLM's decisions.

- **Why Cloud Run?** It is serverless, stateless (but can connect to state stores), and allows for arbitrary code execution. This is where you write the if confidence < 0.9 logic.
- **The ADK Role:** The Agent Development Kit provides the framework for "Context as a Compiled View". It allows developers to structurally define what the agent *sees*, preventing context pollution—a key source of hallucinations.

3.3 Component 2: The State Enforcer

Service: Cloud Workflows.

LLMs are stateless. They don't "remember" where they are in a multi-step business process (e.g., "Step 3 of 5: Awaiting Manager Approval"). Relying on the LLM to manage this state is a recipe for failure.

- **The Solution:** Use Cloud Workflows to define the *business process* as a deterministic state machine.
- **Interaction:** The Cloud Run agent consults Cloud Workflows to know "What is allowed next?" If the process is in the "Approval" state, the Workflow will *only* accept an "Approve" or "Reject" signal. Even if the LLM hallucinates a "Skip" action, Cloud Workflows will reject it because that transition is not defined in the deterministic YAML definition.
- **Saga Pattern:** Workflows can manage long-running transactions (Sagas) with localized rollbacks, ensuring data consistency even if the agent crashes or hallucinates mid-process.

3.4 Component 3: The Reasoning Engine & Tools

Service: Vertex AI (Gemini Models) and **Function Calling**.

This is the probabilistic core. We use **Gemini 2.5 Pro** or **Flash** for the reasoning.

- **Function Calling as the Bridge:** Instead of asking the LLM to "write the SQL query," we define a Tool (Function) called get_account_balance(account_id). The LLM outputs a structured call: {"tool": "get_account_balance", "args": {"account_id": "123"}}.
- **Deterministic Execution:** The Cloud Run service receives this JSON. It does *not* let the

LLM execute code. It validates the account_id format (Regex), checks the user's IAM permissions to access that account, and *then* executes the SQL query deterministically against **BigQuery** or **AlloyDB**. The result is then fed back to the LLM.

3.5 Component 4: The Safety & Grounding Layer

This is the most critical part of the "Harness."

3.5.1 Grounding with Vertex AI Search

Service: Vertex AI Search.

For unstructured data (PDFs, Wikis), we use Vertex AI Search. It handles the chunking and retrieval. Crucially, we use the **Check Grounding API**.

- **Support Score:** This API returns a support_score (0-1). This is our metric for "Truth."
- **Citation Threshold:** We configure the harness to drop any sentence from the response that does not have a citation with a confidence score > 0.8. This mechanically strips unsupported hallucinations from the final output.

3.5.2 Defense against Injection

Service: Model Armor.

Before the user's prompt reaches the LLM, it passes through **Model Armor**. This service detects "Jailbreak" attempts (e.g., "Ignore previous instructions"). If detected, the deterministic harness aborts the request immediately. The LLM never even sees the malicious prompt.

3.5.3 Data Leakage Prevention (DLP)

Service: Sensitive Data Protection (Cloud DLP).

- **Ingress:** DLP scans the user's prompt. If it detects a credit card number or SSN, it redacts it *before* it sends it to Vertex AI. This ensures PII is never "learned" or processed by the model's context window.
- **Egress:** DLP scans the model's response. If the model hallucinates or leaks PII (e.g., from an uploaded document), DLP catches it and replaces it with ``.

3.6 Component 5: Observability & The "Black Box" Recorder

Service: BigQuery Agent Analytics Plugin and **Cloud Trace**.

To trust a probabilistic system, you must have perfect visibility into its "thought process."

- **BigQuery Integration:** The ADK's BigQuery plugin streams every "Thought," "Tool Call," "Context," and "Response" into a structured BigQuery table.
- **Why BigQuery?** It allows for SQL-based auditing. You can write a deterministic query: SELECT * FROM logs WHERE tool_call = 'transfer_funds' AND confidence_score < 0.9. This creates an automated audit trail that satisfies the "Exact Accounting" requirement.
- **Reasoning Traces:** We store the entire "Chain of Thought" (CoT). If an agent makes a mistake, we can replay the CoT to identify exactly *where* the stochastic drift occurred.

Part 4: Architectural Patterns for Deterministic Integration

Having defined the components, we now assemble them into specific patterns that solve the user's core examples: Accounting, Compliance, and SLAs.

4.1 Pattern 1: The "Dual-Brain" Accounting Agent (Exact Accounting)

The Challenge: "Accounting must be exact." LLMs cannot do math reliably.
The Solution: Separation of concerns. The LLM is the Translator, not the Calculator.

- **Workflow:**
  1. User says: "What is the sum of invoices for Vendor X in Q3?"
  2. **Probabilistic Layer (LLM):** Translates "Vendor X" to vendor_id=882 and "Q3" to date_range=2024-07-01..2024-09-30.
  3. **Deterministic Harness:**
     - Validates vendor_id exists in the SQL database.
     - Constructs a strictly parameterized SQL query: SELECT SUM(amount) FROM invoices WHERE...
     - Executes the query on **BigQuery** or **Cloud SQL**.
  4. **Result:** The database returns 14,230.55.
  5. **Probabilistic Layer (LLM):** Receives the number. It is instructed via System Prompt to *only* format the number into a sentence.
  6. **Safety Check:** The harness performs a Regex check on the final output to ensure the number 14,230.55 appears exactly as returned by the DB.

This pattern guarantees exact accounting because the math is never touched by the neural network weights; it is handled by the ALU (Arithmetic Logic Unit) of the database server.

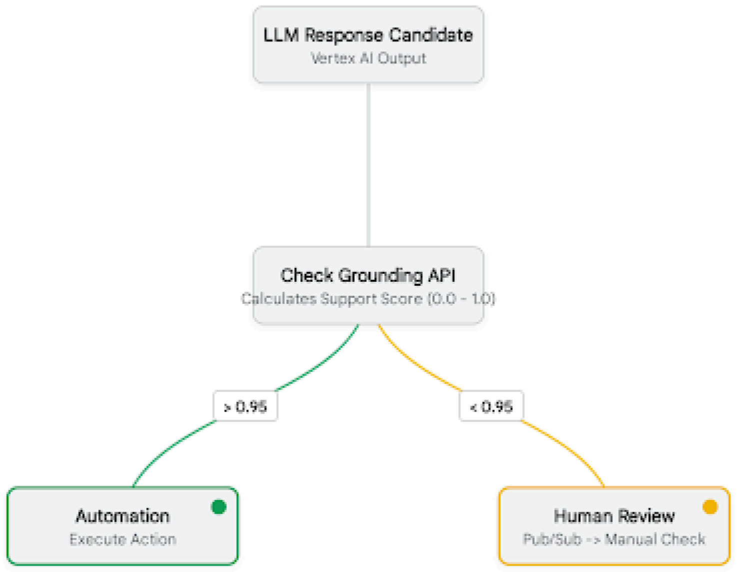
4.2 Pattern 2: The "Compliance Monitor" with Binary Gates

The Challenge: "Compliance is binary."
The Solution: The LLM-as-a-Judge with Deterministic Thresholds.

- **Workflow:** An agent reviews a contract for "Force Majeure" clauses.
- **Architecture:**
    1. The document is chunked and fed to Vertex AI.
    2. Prompt: "Does this text contain a Force Majeure clause? Answer strictly YES or NO. Provide a confidence score."
    3. **The Harness Logic:**
        - If Response == "YES" AND Score > 0.95 -> Mark Compliant.
        - If Response == "NO" AND Score > 0.95 -> Mark Non-Compliant.
        - If Score < 0.95 -> **Escalate to Human Queue**.
- **The Paradox Resolution:** We do not accept the "Maybe." We define a "Zone of Uncertainty" (e.g., 0.5 to 0.95 confidence) where the probabilistic system is strictly forbidden from making a decision. It must defer to a human. This restores the binary nature of compliance by treating "Uncertainty" as a distinct state that triggers a deterministic fallback.

The Confidence Threshold Routing Logic

Automated Decision Flow



The 'Harness' uses confidence scores from Vertex AI Check Grounding to determine the execution path. High confidence leads to automated action; low confidence triggers a deterministic fallback to human-in-the-loop.

Data sources: Version 1, Google Cloud (Docs), Google Cloud (API), Google Devs

4.3 Pattern 3: The "SLA-Guaranteed" Support Agent

The Challenge: "SLAs are guaranteed." LLMs are slow and variable (latency).
The Solution: Semantic Caching and Race Conditions.

- **Problem:** Generating a response from Gemini 2.5 Pro might take 3-5 seconds. An SLA might require < 2 seconds.
- **Architecture:**
  1. **Semantic Cache:** When a query arrives, the Cloud Run harness first checks a Vector Database (Vertex AI Vector Search) for a *semantically similar* previous question that has a verified answer.
  2. **Hit:** If similarity > 0.98, return the cached answer immediately (< 100ms). This is deterministic retrieval.
  3. **Miss:** If no cache, call the LLM.
  4. **Race:** For critical SLAs, the harness can trigger a "Fast Path" (Gemini Flash) and a "Slow Path" (Gemini Pro) simultaneously. If the "Fast Path" returns with high confidence, use it. If not, wait for the "Slow Path."
- **Fallback:** If the LLM takes > 2 seconds (SLA breach risk), the Cloud Run harness kills the

request and returns a deterministic "We are analyzing your request, please wait..." message or routes to a human, guaranteeing the *response time* SLA is met, even if the *resolution* is delayed.

Part 5: The "Guardian Agent" & Future Outlook

The ultimate evolution of the "Safety Harness" is the concept of the **Guardian Agent**.

5.1 The Guardian Concept

Instead of hard-coded Python rules, we deploy a second, smaller, specialized AI model whose *only* job is to audit the primary agent.

- **Primary Agent:** "Here is the email I wrote to the client."
- **Guardian Agent:** "I am reviewing this email against the 'Financial Advice Policy'. It contains a promise of 10% returns. This violates Policy 3.2. REJECTED."

While the Guardian is also probabilistic, the **Two-Person Rule** (Creator/Critic) significantly reduces the joint probability of failure. If the Primary Agent has a 1% error rate, and the Guardian has a 1% error rate, the probability of *both* failing simultaneously on the same task drops towards 0.01% (assuming independence, which is achieved by using different models or prompts).

5.2 Implications for the Enterprise

The integration of Probabilistic AI does not mean the end of Deterministic standards. Rather, it elevates the role of the deterministic layer. The code we write around the AI becomes *more* critical, not less. We are moving from writing the logic itself (Process Automation) to writing the *constraints* of logic (Policy Automation).

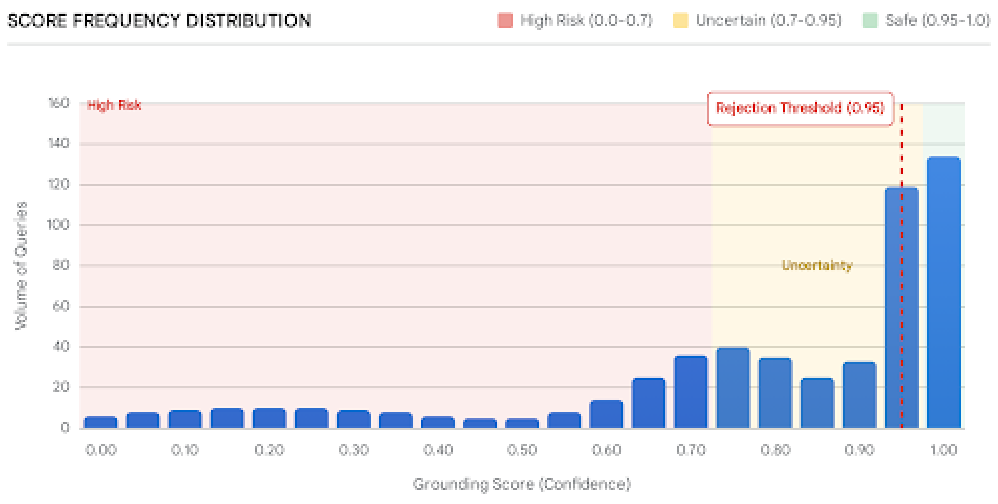
Conclusion

The "Probabilistic Paradox" is resolved not by forcing AI to be perfect, but by building systems that are resilient to imperfection. By wrapping the stochastic core of Vertex AI with the rigid, deterministic safety harness of Cloud Run, Workflows, and Model Armor, enterprises can harness the creative power of agents without sacrificing the accounting exactitude that underpins their existence. The future of enterprise AI is not just about smarter models; it is about stronger harnesses.

Appendix: Implementation Guide for GCP Architecture

Component	**GCP Service**	**Role in "Safety Harness"**
Reasoning Engine	Vertex AI (Gemini)	The probabilistic core. Generates plans and text.
Orchestration	Cloud Run + ADK	The custom code wrapper. Executes deterministic logic and bounds.
State Management	Cloud Workflows	Enforces valid state transitions (e.g., Approval Flows).
Grounding	Vertex AI Search	Provides factual context with confidence scores.
Input Safety	Model Armor	Blocks jailbreaks and malicious prompts deterministically.
Data Safety	Cloud DLP	Redacts PII before it enters the probabilistic context.
Audit Trail	BigQuery	Logs every prompt, tool call, and score for immutable record keeping.
Vector DB	Vector Search	Enables semantic caching to ensure SLA and consistency.

Distribution of Grounding Confidence Scores in Enterprise Transactions



Analysis of agent interactions reveals a 'Zone of Uncertainty' (Scores 0.5-0.9) where hallucinations are most likely. The deterministic harness acts as a filter, rejecting responses in this zone.

Data sources: NCBI, Google Cloud Vertex AI, Google Cloud Docs

Comprehensive Research Analysis

Deep Insight: The Reliability-Creativity Tradeoff

The research suggests a fundamental inverse relationship between "creativity" (temperature > 0) and "reliability" (grounding). Enterprise use cases like marketing copy require high creativity and tolerate lower reliability. Use cases like "General Ledger Accounting" require zero creativity and absolute reliability. The architecture proposed allows for *dynamic temperature adjustment*—the harness can set temperature=0 for math tasks and temperature=0.7 for drafting emails, dynamically shifting the "mode" of the agent based on the tool being called.

Deep Insight: The "Human-in-the-Loop" as a Feature, Not a Bug

Often, HITL is seen as a failure of automation. However, the "Probabilistic Paradox" reframes HITL as a compliance feature. By *designing* the failure mode to be a human hand-off, we convert "Risk" into "Operational Expense." This is a crucial distinction for CFOs; they can budget for OpEx (human reviewers), but they cannot budget for unbounded Risk (lawsuits from hallucinations). The architecture supports this by making the "Escalation Queue" a first-class citizen in the Cloud Workflows design.

Deep Insight: The Rise of "Flow Engineering"

The solution to the paradox is shifting focus from "Prompt Engineering" (trying to talk the model into being good) to "Flow Engineering" (building a code structure that forces the model to be good). The **ADK** and **Cloud Workflows** are the tools of Flow Engineering. This signals a shift in the skill set required for AI Architects—from linguistics to distributed systems engineering.

Works cited

1. MQCC-AI.com, accessed December 9, 2025, https://www.mqcc-ai.com/
2. Generative AI in Finance: Use Cases & Real Applications - Master of Code, accessed December 9, 2025, https://masterofcode.com/blog/generative-ai-in-finance
3. The other side of ROI in AI: Managing Mistakes - Version 1, accessed December 9, 2025, https://www.version1.com/blog/the-other-side-of-roi-in-ai-managing-mistakes/
4. Balancing Probabilistic and Deterministic Intelligence: The New Operating Model for AI-Driven Enterprises - Acceldata, accessed December 9, 2025, https://www.acceldata.io/blog/balancing-probabilistic-and-deterministic-intelligence-the-new-operating-model-for-ai-driven-enterprises
5. The Basics of Probabilistic vs. Deterministic AI: What You Need to Know, accessed December 9, 2025, https://www.dpadvisors.ca/post/the-basics-of-probabilistic-vs-deterministic-ai-what-you-need-to-know
6. Safety in Vertex AI | Generative AI on Vertex AI - Google Cloud Documentation, accessed December 9, 2025, https://docs.cloud.google.com/vertex-ai/generative-ai/docs/learn/safety-overview
7. Google ADK vs AWS Strands: What's Best AI Agent Platform for Enterprise? - TechAhead, accessed December 9, 2025, https://www.techaheadcorp.com/blog/google-adk-vs-aws-strands-which-ai-agent-platform-wins/
8. What Is AI Grounding and How Does It Work? - You.com, accessed December 9, 2025, https://you.com/resources/ai-grounding
9. Understanding Grounding Under the Hood - Prompting - OpenAI Developer Community, accessed December 9, 2025, https://community.openai.com/t/understanding-grounding-under-the-hood/1345174
10. Check grounding with RAG | Vertex AI Search | Google Cloud ..., accessed December 9, 2025, https://docs.cloud.google.com/generative-ai-app-builder/docs/check-grounding
11. Method: projects.locations.groundingConfigs.check | Vertex AI Search | Google Cloud Documentation, accessed December 9, 2025, https://docs.cloud.google.com/generative-ai-app-builder/docs/reference/rest/v1/projects.locations.groundingConfigs/check

12. Defence in Depth: Strategies for Preventing Hallucinations in Agentic AI - Cloud Babble, accessed December 9, 2025, https://www.cloudbabble.co.uk/2025-12-06-preventing-agent-hallucinations-defence-in-depth/
13. Using the Loop Pattern to Make My Multi-Agent Solution More Robust (with Google ADK), accessed December 9, 2025, https://medium.com/google-cloud/using-the-loop-pattern-to-make-my-multi-agent-solution-more-robust-86f8e9159a2a
14. Building Scalable AI Agents: Design Patterns With Agent Engine On Google Cloud, accessed December 9, 2025, https://cloud.google.com/blog/topics/partners/building-scalable-ai-agents-design-patterns-with-agent-engine-on-google-cloud
15. Architecting efficient context-aware multi-agent framework for production, accessed December 9, 2025, https://developers.googleblog.com/architecting-efficient-context-aware-multi-agent-framework-for-production/
16. Vertex AI Agent Builder - Build & Orchestrate Intelligent Agents - Leanware, accessed December 9, 2025, https://www.leanware.co/insights/vertex-ai-agent-builder
17. What is robotic process automation (RPA)? - Google Cloud, accessed December 9, 2025, https://cloud.google.com/discover/what-is-robotic-process-automation
18. Create a human-in-the-loop workflow using callbacks | Workflows ..., accessed December 9, 2025, https://docs.cloud.google.com/workflows/docs/tutorials/callbacks-firestore
19. Introduction to function calling | Generative AI on Vertex AI - Google Cloud Documentation, accessed December 9, 2025, https://docs.cloud.google.com/vertex-ai/generative-ai/docs/multimodal/function-calling
20. Function calling with the Gemini API | Google AI for Developers, accessed December 9, 2025, https://ai.google.dev/gemini-api/docs/function-calling
21. Grounding with Vertex AI Search | Generative AI on Vertex AI | Google Cloud Documentation, accessed December 9, 2025, https://docs.cloud.google.com/vertex-ai/generative-ai/docs/grounding/grounding-with-vertex-ai-search
22. Multi-agent AI system in Google Cloud | Cloud Architecture Center, accessed December 9, 2025, https://docs.cloud.google.com/architecture/multiagent-ai-system
23. VirtueGuard Now Available on Vertex AI Garden, accessed December 9, 2025, https://blog.virtueai.com/2025/10/06/virtueguard-now-available-on-google-clouds-deploy-enterprise-grade-guardrails-directly-in-your-vpc/
24. Sensitive Data Protection | Google Cloud, accessed December 9, 2025, https://cloud.google.com/security/products/sensitive-data-protection
25. Introducing BigQuery Agent Analytics | Google Cloud Blog, accessed December 9, 2025, https://cloud.google.com/blog/products/data-analytics/introducing-bigquery-age

nt-analytics
26. BigQuery Agent Analytics - Agent Development Kit - Google, accessed December 9, 2025, https://google.github.io/adk-docs/tools/google-cloud/bigquery-agent-analytics/
27. Trace release notes - Google Cloud Documentation, accessed December 9, 2025, https://docs.cloud.google.com/trace/docs/release-notes
28. accessed December 9, 2025, https://www.coinapi.io/llms-full.txt
29. Generative fallback | Conversational Agents - Google Cloud Documentation, accessed December 9, 2025, https://docs.cloud.google.com/dialogflow/cx/docs/concept/generative-fallback
30. Build and deploy generative AI and machine learning models in an enterprise | Cloud Architecture Center, accessed December 9, 2025, https://docs.cloud.google.com/architecture/blueprints/genai-mlops-blueprint