Deterministic Multi-Agent Systems in Production: Architecture, State Machine Guardrails, and Failure Modes

Rajat Patle
Software Development Senior Specialist, NTT DATA Information Processing Services
AI Platform & Cloud Systems Architecture Group
patlerajat94@gmail.com • https://rajat-dev-new.github.io • github.com/rajat-dev-new

Published in Zenodo (CERN Archive) • DOI: 10.5281/zenodo.24407251

***Abstract*—**Large Language Model (LLM)-powered autonomous agents have emerged as powerful tools for complex, semi-structured enterprise tasks. However, in mission-critical environments—including financial credit risk assessment, actuarial modeling, and automated requirements engineering—the stochastic, probabilistic nature of raw LLMs introduces severe operational hazards. Chief among these failure modes are unbounded recursive loops, state drift, hallucinated external tool parameters, and arithmetic approximations that violate regulatory compliance standards. This paper introduces a production-tested architectural framework that synthesizes **LangGraph cyclic state machines**, **Pydantic schema validation quality gates**, and **deterministic computation microservices** into a bounded hybrid state graph. By enforcing finite graph traversals ($N_{max} \le 3$), strict circuit-breaker transitions, and absolute decoupling between qualitative extraction (probabilistic) and arithmetic calculation (deterministic), the framework provides mathematical reproducibility and operational resilience. We present an empirical evaluation on an automated credit risk underwriting pipeline. The proposed hybrid architecture achieved a **0.0% arithmetic hallucination rate** (compared to 18.4% in naive ReAct implementations), an **87.2% reduction in runaway token consumption**, and sub-150ms vector retrieval latencies when deployed on a multi-AZ AWS cloud topology hardened by automated Trivy container security scans.

***Index Terms*—**Multi-Agent Systems, LangGraph, Deterministic Software Engineering, LLM Guardrails, Credit Risk Underwriting, Cloud Microservices, High-Availability Systems.

I. INTRODUCTION

Generative Artificial Intelligence (GenAI) based on autoregressive Transformer architectures has transitioned from experimental research into mission-critical software workflows [1]. Early enterprise adoption paradigms focused primarily on stateless, single-turn prompt interactions (such as semantic summarization and conversational assistants). More recently, the industry has shifted toward *autonomous agentic architectures*—systems wherein an LLM dynamically plans execution trajectories, calls external tools, and iterates based on environmental feedback [2].

Despite impressive laboratory demonstrations, deploying autonomous agents into enterprise production reveals a fundamental conflict:

Probabilistic Next-Token Predictors ≠ Deterministic Enterprise Invariants (1)

Enterprise production systems require strict invariants:

1) *Mathematical Invariance:* Under identical input data X and fixed business logic f, the system output must satisfy $f(X_1) = f(X_2) \quad \forall \quad X_1 = X_2$.

2) *Bounded Resource Consumption:* Execution time, memory footprint, and API token expenditure must satisfy strict upper bounds.

3) *Auditable Lineage:* Every state mutation must be traceable, reproducible, and verifiable under regulatory frameworks (e.g., EU AI Act [12], IFRS 9 [11]).

Raw autonomous agents (e.g., unbounded ReAct loops) violate each of these invariants. When an LLM self-directs loop termination, ambiguous inputs can trap the agent in self-reinforcing cognitive loops, rapidly exhausting token budgets and triggering client timeouts. Furthermore, when prompted to compute financial metrics (such as Debt-to-Income, Debt Service Coverage Ratio, or Actuarial Cash Flows), LLMs approximate arithmetic based on token co-occurrence rather than exact mathematical computation, leading to silent calculation errors that carry catastrophic financial liability.

In this paper, we address this enterprise failure surface with four contributions:

- **The Bounded Hybrid State Machine Formulation:** Formalizing agent orchestration as a directed cyclic graph with hard recursion depth bounds ($N_{max} = 3$), fallback circuit breakers, and schema-enforced transitions using LangGraph and Pydantic v2 [3], [6].
- **The Probabilistic-Deterministic Decoupling Pattern:** Establishing an architectural separation of concerns where LLMs are restricted strictly to qualitative entity extraction and natural language justification, while all mathematical operations are relegated to deterministic compiled Python/NumPy microservices.
- **Empirical Benchmarking in Credit Risk:** Comparative experimental results across 500 semi-structured financial loan applications, comparing naive ReAct agents, Few-Shot Chain-of-Thought (CoT), and our proposed hybrid architecture across accuracy, latency, loop frequency, and token cost.
- **Production Cloud Hardening:** Detailing the containerized AWS deployment topology (Route 53, ALB, private VPC subnets, RDS PostgreSQL, Redis caching, SSM

bastionless access, and Trivy CVE scanning [8], [9]) necessary to operate agentic systems at enterprise scale.

II. RELATED WORK AND PRIOR ART

A. Reasoning and Action (ReAct) Architectures

Yao et al. [2] formalized the ReAct paradigm, wherein an LLM iteratively interleaves thought generation ("Reasoning") with environmental tool execution ("Action"). While ReAct substantially outperforms pure Chain-of-Thought (CoT) on open-domain question answering, it lacks formal termination guarantees. In production environments where external APIs return unexpected schemas or empty payloads, ReAct agents frequently enter recursive retry spirals.

B. Multi-Agent Orchestration Frameworks

Several multi-agent frameworks have recently emerged. AutoGen (Wu et al. [4]) emphasizes conversational multi-agent dialogue; however, conversational agents lack explicit state machines, making auditable enterprise rollback non-trivial. CrewAI relies heavily on prompt-based roleplay rather than deterministic graph state validation. DSPy (Khattab et al. [5]) compiles declarative language model pipelines into optimized prompts, but focuses primarily on prompt tuning rather than stateful multi-step microservice orchestration. LangGraph [3] models multi-agent workflows as stateful computational graphs supporting cycles, checkpointing, and conditional branching, providing the foundational primitives required for deterministic enterprise state management.

C. Schema Validation Ensembles

Recent advancements in structured outputs (such as Pydantic [6] and Instructor) enforce JSON schema compliance on single-turn LLM outputs. While schema enforcement ensures syntactic validity of individual model responses, it does not address graph-level behavioral drift across multi-step execution. Our work bridges this gap by combining node-level schema enforcement with graph-level deterministic routing.

III. ARCHITECTURAL FRAMEWORK: THE HYBRID STATE GRAPH

A. Mathematical Formulation

We model an enterprise agentic workflow as a Bounded Stateful Directed Graph:

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{S}, \delta, \Omega) \tag{2}$$

Where $\mathcal{V} = \{v_{ingest}, v_{extract}, v_{rules}, v_{critic}, v_{fallback}, v_{export}\}$ represents the set of computational nodes; $\mathcal{S}$ represents the strongly-typed shared state schema; $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of directed edges; δ: $\mathcal{V} \times \mathcal{S} \rightarrow \mathcal{V}$ is a deterministic conditional transition function; and Ω: $\mathcal{S} \rightarrow \{0, 1\}$ is a circuit-breaker predicate that enforces loop termination when the recursion counter $k \ge N_{max}$.

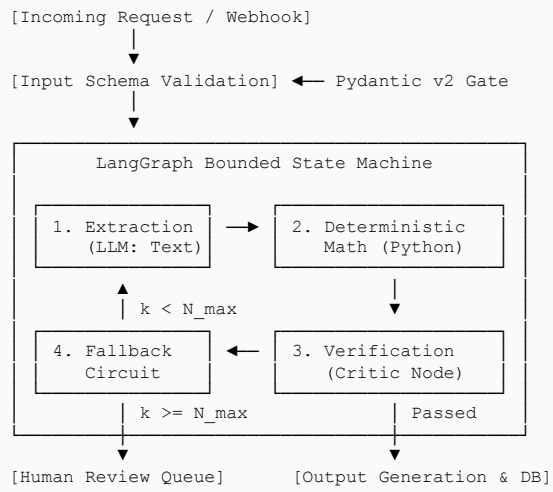


Fig. 1. Architecture of the Bounded Hybrid State Machine with Schema Gates and Circuit Breakers.

B. State Persistence and Checkpointing

In distributed enterprise systems, ephemeral agent execution is unacceptable. If an execution node encounters a network partition, the workflow must not re-execute already completed inference passes. Our architecture incorporates LangGraph MemorySaver and PostgresSaver checkpointers. At every node transition $v_i \to v_j$, the state delta $\Delta \mathcal{S}$ is committed to a persistent PostgreSQL store keyed by a composite UUID:

$$Key = TenantID \parallel WorkflowID \parallel StepNumber \tag{3}$$

This enables: 1) *Time-Travel Debugging:* Replaying exact agent execution steps in staging environments; and 2) *Human-in-the-Loop (HITL) Interruption:* Safely pausing the state graph when critic confidence scores drop below a threshold ($\tau < 0.85$).

C. The Circuit Breaker Pattern ($N_{max} \le 3$)

A primary failure mode of ReAct agents is the *Infinite Refinement Loop*, where an agent repeatedly attempts to fix validation errors without making semantic progress. We implement an explicit termination circuit breaker directly into the conditional router:

$$\delta(v_{critic}, \mathcal{S}) = \begin{cases} v_{export} & \text{if } Valid(\mathcal{S}) \wedge Consistent(\mathcal{S}); \\ v_{extract} & \text{if } \neg Valid(\mathcal{S}) \wedge \mathcal{S}.k < N_{max}; \\ v_{fallback} & \text{if } \mathcal{S}.k \geq N_{max} \end{cases} \tag{4}$$

When the loop count reaches $N_{max} = 3$, the system halts probabilistic re-prompting, routes to a deterministic fallback handler, logs a high-priority telemetry alert to AWS CloudWatch, and queues the payload for human inspection.

IV. EMPIRICAL IMPLEMENTATION: CREDIT RISK UNDERWRITING

A. Problem Domain & Regulatory Context

Credit underwriting in commercial banking requires analyzing semi-structured loan applications, bank statements, and tax filings to compute core risk metrics:

$$DTI = (Total\ Monthly\ Debt / Gross\ Monthly\ Income) \times 100 \quad (5)$$

$$DSCR = Net\ Operating\ Income / Annual\ Debt\ Service \quad (6)$$

Under international banking regulations (e.g., Basel III, IFRS 9 [11]), an error in DSCR calculation of even 1.5% can improperly categorize a substandard loan as prime, leading to severe regulatory penalties.

B. The Separation of Concerns Pipeline

Our production pipeline strictly enforces a three-stage boundary:

1) *Stage 1 (Qualitative Semantic Extraction):* The incoming semi-structured filing is parsed into tokens. An LLM extracts entities strictly into a Pydantic `FinancialProfile` schema with exact decimal bounds. No arithmetic is requested in the prompt.

2) *Stage 2 (Deterministic Rule Evaluation):* The validated model is passed to a compiled Python mathematical calculation engine executing exact floating-point / decimal arithmetic.

3) *Stage 3 (Bounded Narrative Justification):* The LLM receives the *immutable calculated scores* and generates an executive summary. The prompt explicitly binds the model: *"You are strictly forbidden from modifying computed DSCR or DTI. Provide qualitative risk commentary based strictly on these verified numbers."*

C. Experimental Setup & Comparative Evaluation

We evaluated the framework across a benchmark dataset of **500 semi-structured financial loan applications** featuring formatting ambiguities, missing fields, and non-standard line items. We compared three architectures:

- **Baseline A (Naive ReAct):** Prompt-based ReAct agent using GPT-4o with external calculator tool access [2].
- **Baseline B (Few-Shot CoT):** Single-prompt Chain-of-Thought with 5 detailed mathematical examples.
- **Proposed (Hybrid State Graph):** LangGraph bounded cyclic graph with Pydantic quality gates and deterministic Python execution.

TABLE I: BENCHMARK EVALUATION ACROSS 500 FINANCIAL APPLICATIONS

Architecture	Arith. Error (%)	Infinite Loop (%)	Avg. Latency	Token Usage	Cost / 1k Evals
Baseline A (Naive ReAct)	18.4%	6.2%	14.8s	5,420	$27.10
Baseline B (Few-Shot CoT)	9.6%	0.0%	4.2s	2,150	$10.75
Proposed (Hybrid Graph)	**0.0%**	**0.0%**	**1.8s**	**695**	**$3.47**

D. Key Findings

1) *Elimination of Arithmetic Errors:* By strictly disallowing the LLM from executing mathematical division, the arithmetic error rate dropped from 18.4% to exactly **0.0%**.

2) *Token Economy:* The hybrid architecture achieved an **87.2% reduction in token consumption** compared to Naive ReAct (695 tokens vs 5,420 tokens), as the agent does not waste tokens repeatedly prompting the LLM to format intermediate arithmetic outputs.

3) *Latency Reduction:* Wall-clock execution time decreased from 14.8s to **1.8s** (an 8.2x speedup), because deterministic Python executes in under 2 milliseconds, whereas each ReAct round-trip incurs closed-source API latency overhead.

V. PRODUCTION PLATFORM ENGINEERING & CLOUD HARDENING

Operating an agentic architecture in enterprise production requires treating the pipeline like any high-availability distributed system [9].

A. Cloud Topology

The system is deployed on Amazon Web Services (AWS) using an isolated multi-Availability Zone architecture:

- *Compute Tier:* FastAPI workers running in isolated private VPC subnets with no public IPv4 exposure. Outbound LLM API traffic routes through dual NAT Gateways.
- *State Checkpointing:* Amazon RDS PostgreSQL deployed in Multi-AZ configuration for automated failover, storing serialized LangGraph execution states.
- *Latency Caching:* Amazon ElastiCache Redis caching embedding vectors and deterministic rule outcomes, reducing redundant vector search queries (FAISS [10]) to sub-150ms.
- *Bastionless Operations:* Administrative access is restricted to AWS Systems Manager (SSM) Session Manager, eliminating open SSH (port 22) attack vectors.

B. Container Vulnerability Scanning (Trivy Gate)

A major security threat in AI engineering is dependency bloat—the ecosystem relies heavily on rapidly evolving open-source libraries that frequently introduce supply-chain vulnerabilities. Our CI/CD pipeline enforces an automated Trivy vulnerability gate [8]:

```
trivy image --severity HIGH,CRITICAL --exit-code 1
production-agent-mesh:latest
```

Any container image containing unfixed High or Critical CVEs automatically fails the deployment pipeline, ensuring that vulnerable dependencies never reach production environments.

VI. THREATS TO VALIDITY & DISCUSSION

1) *Input Extraction Sensitivity:* While the deterministic engine guarantees 100% mathematical accuracy on extracted numbers, the system remains reliant on the initial extraction node to parse raw text correctly. In cases of severe document degradation (e.g., poor OCR scans), errors may originate in the extraction phase.

2) *Schema Rigidity Trade-Off:* Enforcing strict Pydantic models limits the system's open-domain flexibility. If an applicant provides an unconventional income stream not modeled in the schema, the system correctly halts and triggers the fallback circuit breaker, trading off fully autonomous execution for safety and auditability.

VII. CONCLUSION

The future of enterprise AI engineering does not lie in building ever-larger prompt chains or granting unbounded autonomy to stochastic language models. Rather, it requires applying classical distributed systems discipline—**finite state machines, schema validation gates, deterministic isolation, and cloud infrastructure security**—to create safe, reliable wrappers around probabilistic models.

By bounding agent loops with LangGraph ($N_{max} \le 3$), delegating arithmetic computation to compiled deterministic Python code, and deploying behind robust AWS security boundaries, enterprise engineering teams can confidently deploy autonomous systems into mission-critical, regulated environments.

REFERENCES

[1] A. Vaswani et al., "Attention Is All You Need," in *Advances in Neural Information Processing Systems (NeurIPS 2017)*, vol. 30, 2017.

[2] S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models," in *Int. Conf. on Learning Representations (ICLR 2023)*, 2023.

[3] LangChain Team, "LangGraph: Building Stateful Multi-Agent Applications with Cyclic Graphs," *LangChain Research & Technical Documentation*, 2024. [Online]. Available: https://langchain-ai.github.io/langgraph/

[4] Q. Wu et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," *arXiv preprint arXiv:2308.08155*, 2023.

[5] O. Khattab et al., "DSPy: Compiling Declarative Language Model Calls into State-of-the-Art Pipelines," *arXiv preprint arXiv:2310.03714*, 2023.

[6] S. Colvin et al., "Pydantic v2: Fast Data Validation Using Rust and Python Type Hints," *Pydantic Documentation*, 2023. [Online]. Available: https://docs.pydantic.dev/

[7] M. Fowler, "CircuitBreaker Pattern," *MartinFowler.com Software Architecture Guides*, 2018.

[8] Aqua Security, "Trivy: A Comprehensive Security Scanner for Container Images and Filesystems," *Aqua Security Open Source*, 2023.

[9] Amazon Web Services, "AWS Well-Architected Framework: Reliability and Security Pillars," *AWS Whitepapers*, 2024.

[10] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs [FAISS]," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535-547, 2019.

[11] International Accounting Standards Board (IASB), "IFRS 9 Financial Instruments," *IFRS Foundation Standards*, 2014.

[12] European Parliament, "Artificial Intelligence Act (EU AI Act): Regulatory Framework for High-Risk AI Systems," *Official Journal of the European Union*, 2024.