# Architecting the SME-Plug: A Blueprint for Deterministic, Hot-Swappable AI Agents in Enterprise Ecosystems

## Introduction: The Evolution of Compound AI Systems and the Agentic Imperative

The global artificial intelligence landscape is currently undergoing a profound architectural and economic transition, shifting away from a reliance on monolithic, generalized Large Language Models (LLMs) toward highly specialized, multi-component frameworks known as compound AI systems.[1] Foundational LLMs, while demonstrating exceptional linguistic capabilities and the capacity to generate highly convincing human-like text, are fundamentally constrained by their static nature post-training.[1] They lack intrinsic contextual awareness, cannot natively interact dynamically with external systems, and operate purely on probabilistic token prediction.[1] In contrast, compound AI systems represent an architectural maturation, integrating non-differentiable components—such as deterministic search engines, programmatic code interpreters, and sophisticated vector databases—to create resilient, verifiable, and highly accurate workflows.[2] This evolution mirrors the historical progression of software engineering, where monolithic architectures inevitably yielded to microservices and modular integration patterns.

This paradigm shift has catalyzed the development of Agentic AI, a technological leap where systems transcend the role of passive text generators to become autonomous entities capable of executing complex, multi-step actions in pursuit of strictly defined operational goals.[1] Agentic systems leverage foundational language models embedded within dynamic feedback loops, allowing them to plan, reflect, act, and continuously adapt to external stimuli.[4] Economic modeling of the enterprise AI sector heavily supports this architectural pivot. Market research projections indicate that the deployment of multilingual, agentic workflows represents a significant consolidation event within the industry.[5] As the exorbitant computational costs of training proprietary, generalized LLMs become increasingly unsustainable for all but a few hyperscalers, the market is shifting.[5] Industry analysts predict a 75% reduction in GenAI vendor fragmentation by 2029, mirroring the historical consolidation of local electricity generators into standardized utility grids.[5] Consequently, enterprise value generation is pivoting toward the orchestration of specialized agents that utilize generic foundation models, augmented by hyper-specific, locally hosted logic layers and culturally relevant Small Language Models (SLMs) tailored to specific global regions.[5]

The primary objective of this exhaustive research report is to formalize the architecture for a "Subject Matter Expert Plugin" (SME-Plug). The SME-Plug is conceptualized as an encapsulated,

highly modular AI architecture capable of instantaneously transforming a baseline, generalized LLM into a deterministic, highly specialized domain expert. By orchestrating dynamic system prompts, real-time hot-swappable tool injection, rigorous Retrieval-Augmented Generation (RAG) pipelines, and uncompromising citation guardrails, the SME-Plug serves as the foundational unit for next-generation enterprise AI orchestration. To facilitate the development and rapid prototyping of this architecture, this document outlines a comprehensive hackathon problem statement, followed by an exhaustive technical blueprint detailing the underlying systems required to engineer the SME-Plug Minimum Viable Product (MVP).

# The Hackathon Problem Statement: Engineering Deterministic Expertise

To drive innovation in the realm of compound AI systems and address the critical limitations of current enterprise AI deployments, the following problem statement has been designed for an advanced, enterprise-level engineering hackathon. The challenge deliberately forces engineering teams to move beyond rudimentary chatbot interfaces and develop a robust, production-ready, highly constrained AI architecture capable of operating in high-stakes environments.

## Challenge Title: The SME-Plug MVP: Engineering Deterministic, Hot-Swappable AI Architectures

**Background:**
Enterprise AI adoption is severely hampered by the inherently probabilistic nature of Large Language Models. In critical sectors—ranging from legal compliance and medical documentation to supply chain logistics and financial auditing—LLMs frequently generate unverified claims, hallucinate citations, and lack strict contextual boundaries. Furthermore, deploying separate, fine-tuned monolithic models for every discrete enterprise domain is computationally prohibitive, impossible to maintain, and architecturally inefficient. The modern enterprise requires a singular, highly orchestrated AI routing layer capable of dynamically "plugging in" subject matter expertise on demand, effectively hot-swapping the persona, constraints, and tool access of an underlying foundation model in milliseconds.

**The Objective:**
Participants are tasked with building the Minimum Viable Product (MVP) of an 'SME-Plug' (Subject Matter Expert Plugin). The SME-Plug must function as a modular, state-aware architectural component that transforms a generic LLM into a hyper-specialized, deterministic expert capable of executing multi-step enterprise workflows without hallucination or deviation from corporate policy.

**Core Engineering Requirements:**

1. **Dynamic Context Engineering and Prompt Overrides:** Implement a middleware orchestration layer that modifies the LLM's system prompts, personas, and behavioral constraints in real-time. This modification must be driven by transient conversation state, persistent user store data, and static runtime context variables (e.g., user authorization

roles).

2. **Hot-Swappable Tool Injection:** Utilize advanced dependency injection patterns to dynamically mount and unmount RAG-based retrieval tools and simulated enterprise API endpoints (such as Salesforce or SAP connectors) directly into the agent's context window. The agent must only possess awareness of tools it is explicitly authorized to use for a given transaction, and these tools must be injected without requiring a system restart.
3. **Multi-Agent Routing and Handoff Protocols:** Design a multi-agent control plane capable of intelligently routing complex, cross-domain queries. The system must demonstrate the ability to execute seamless handoffs between discrete SME-Plugs (e.g., transferring control and conversational memory from an "HR Agent" to an "IT Provisioning Agent") while optimizing for token consumption and maintaining strict state compartmentalization.
4. **Deterministic Citation Guardrails and RAG Evaluation:** The MVP must actively prevent generative hallucinations. All factual outputs must be backed by a verified RAG retrieval process, accompanied by explicit source citations. The architecture must incorporate both rule-based and model-based middleware guardrails that intercept, verify, and filter ungrounded claims or malicious prompt injections before the final response payload is delivered to the end-user. The success of this component must be measurable using standardized retrieval and generation metrics.

**Evaluation Focus:**
Submissions will not be evaluated on the parameter count or native intelligence of the underlying language model utilized. Instead, judging will focus entirely on the rigor, stability, and determinism of the orchestration layer. Winning solutions will demonstrate exceptionally high context precision, seamless and secure tool handoffs, zero tolerance for ungrounded factual hallucinations, and a modular, easily extensible codebase that allows new SME-Plugs to be registered within the enterprise ecosystem with minimal configuration.

# The Hallucination Crisis: The Imperative for Deterministic Constraints

The absolute necessity for the constrained SME-Plug architecture is underscored by the severe, systemic vulnerabilities inherent in unprotected, generalized LLMs. When deployed in high-stakes operational environments, the probabilistic token-generation mechanisms of LLMs frequently lead to "hallucinations"—instances where the model confidently fabricates information, citations, or data structures that deviate entirely from factual reality or established organizational protocols.[6]

## High-Stakes Failures in Law, Medicine, and Logistics

The legal sector provides a stark, highly publicized illustration of these catastrophic risks. In a widely analyzed incident, a practicing attorney submitted a legal brief to a federal court that had been generated entirely by an unconstrained LLM.[7] The presiding judge discovered that

the brief was populated entirely with fabricated judicial decisions and fictitious internal citations, characterizing the event as an unprecedented circumstance driven by bogus LLM outputs.[7] This event highlighted a fundamental architectural flaw: LLMs are designed to optimize for linguistic coherence and statistical likelihood, not for factual accuracy or adherence to strict legal precedent.[7] Quantitative evaluations of popular models performing legal tasks—such as describing well-established rules or reasoning through multistate bar exam questions—reveal pervasive, deeply disturbing errors that necessitate external validation frameworks.[7]

Similar structural vulnerabilities plague the healthcare and clinical administrative sectors. Clinical note generation is a prime target for AI automation to reduce the cognitive load and burnout associated with prolonged interaction with electronic health records.[9] However, this requires absolute fidelity between the LLM output and the ground truth of the patient encounter.[9] Rigorous research assessing LLM-driven clinical document generation across 18 distinct experimental configurations and 12,999 clinician-annotated sentences identified base hallucination rates of 1.47% alongside critical omission rates of 3.45%.[9] In medicine, an omission—where an LLM simply drops relevant diagnostic data from a summary—can be as lethal as a fabricated claim, demanding the implementation of rigorous error taxonomies and clinical safety frameworks to evaluate the harm of AI-generated errors.[9]

In the realm of logistics and supply chain management, the impacts of unconstrained AI generation are uniquely disruptive. A demand forecasting model experiencing a hallucination might invent a completely fictitious sales trend or economic indicator.[10] Acting upon this hallucinated data inevitably causes catastrophic overstocking or severe supply stockouts, directly impacting corporate revenue.[10] Furthermore, AI systems tasked with transportation routing have been observed generating "optimal" delivery routes that violate physical world constraints or do not actually exist, leading to severe operational delays.[10] Semantic drift detection is required to spot when an agent's language pivots from routine logistics fields to sensationalized rhetoric, and entity-verification scoring must be utilized to ensure matches between named individuals and trusted operational watchlists.[11] When users receive fabricated outputs, trust in the automated system evaporates, resulting in a regression to manual verification that completely negates the theoretical efficiency gains of the AI deployment.[10]

## The Threat of Package Hallucination and "Slopsquatting"

Beyond operational inefficiencies and professional errors, the probabilistic nature of LLMs introduces highly sophisticated, novel cybersecurity attack vectors. One of the most insidious emerging threats within software supply chain security is "package hallucination," which facilitates a devastating exploit colloquially termed "slopsquatting".[12] During code generation or software architecture tasks, an LLM might reference a software library, dependency module, or framework that does not exist in reality.[13]

Malicious threat actors proactively monitor the outputs of popular AI coding assistants to identify these frequently hallucinated package names. Once identified, the attackers register the fake packages in legitimate open-source repositories and embed highly sophisticated

malicious code within them.[12] If a developer blindly trusts the LLM's architectural recommendation and installs the hallucinated package, the malware is introduced directly into the enterprise software supply chain.[12] This malware can operate silently to steal credentials, establish remote backdoors, or beacon to a command and control server to update the package with malicious payloads at a later date, rendering initial security scans useless.[12]
Studies conducted by researchers evaluating code generated by models across Python and JavaScript found that nearly 20% of the 756,000 code samples tested contained hallucinated package names, creating an massive, highly exploitable attack surface.[12] Given that over 97% of developers currently utilize AI coding tools, the risk of dependency hallucination necessitates strict isolation boundaries.[12]
The pervasive nature of these failures dictates a fundamental shift in how enterprises approach generative AI. Generative models must be treated as flawed cognitive reasoning engines rather than authoritative databases or secure execution environments.[14] For enterprise adoption to scale securely, the underlying architecture must tightly constrain the model, forcing it to operate strictly within verifiable, deterministic boundaries governed by external logic. This architectural containment is the core function of the SME-Plug.

# Architectural Blueprint: Dynamic Context Engineering

To architect an AI agent that operates with the precision, reliability, and security of a subject matter expert, developers must master the discipline of dynamic context engineering. Context engineering is the programmatic practice of supplying an LLM with the precise information, behavioral rules, and tools required for a specific operational state, while simultaneously withholding irrelevant data that could dilute its focus, trigger policy violations, or overwhelm its limited context window.[15]
The SME-Plug relies heavily on an orchestration framework, such as LangChain or LangGraph, to manage this context dynamically. The architecture utilizes a highly sophisticated middleware API that intercepts the communication flow between the user application and the LLM, programmatically altering the execution state on the fly.[15]

## Orchestrating Context Data Sources

Dynamic context engineering relies on synthesizing data from three distinct temporal scopes, ensuring the agent possesses both immediate situational awareness and long-term behavioral consistency:

1. **State (Short-Term Memory):** This layer represents the immediate, transient context of the current conversational interaction.[15] The orchestration middleware continuously monitors state variables, such as the current message count, the specific sequence of prior tool invocations, or the emotional sentiment of the most recent user prompt. For example, if an internal loop detects that a conversation has exceeded a specific length threshold, the middleware can dynamically inject a system prompt overriding the agent's verbosity, instructing the model to provide highly concise answers to optimize token usage and reduce latency.[15]

2. **Store (Long-Term Memory):** This layer provides persistent context across multiple sessions, ensuring the SME-Plug retains knowledge of historical interactions.[15] The framework accesses secure database records associated with a specific user identifier to retrieve historical preferences, past operational insights, or authorized communication styles.[15] By pulling an individual's email writing style from the store, the middleware can inject this data directly into the system prompt to guide drafting tasks seamlessly.[15]
3. **Runtime Context (Static Configuration):** This layer represents the immutable parameters of the current execution environment, acting as the ultimate authority on agent permissions.[15] It includes data such as the user's specific administrative role, active feature flag statuses, or the deployment environment classification.[15] This allows for critical overrides based on security boundaries, ensuring that an agent operating in a production environment behaves far more conservatively than one operating in a sandbox.

### Middleware Implementation and Dynamic Model Routing

The actual transformation of a generic model into an SME-Plug is executed via programmatic system prompt overrides. By leveraging Python decorators such as @dynamic_prompt within the agent's middleware stack, developers can algorithmically construct the model's base instructions just milliseconds before the API payload is dispatched to the LLM provider.[15]
For instance, an SME-Plug tasked with drafting external corporate communications can query the "Store" to extract a user's historical writing style, injecting these parameters directly into the prompt matrix.[15] Concurrently, the middleware queries the "Runtime Context" to determine if the invoking user holds a managerial role; if they do not, strict compliance guardrails are appended to the system instructions, explicitly forbidding the model from executing unauthorized commands or accessing sensitive financial data.[15] This complex combination of transient and persistent context modification ensures that the LLM is perpetually aligned with the exact operational requirements of the moment, creating the highly convincing illusion of a hot-swappable expert persona.[15]
Furthermore, advanced middleware patterns allow for highly optimized dynamic model selection.[15] Using decorators like @wrap_model_call, an incoming query can be algorithmically evaluated for cognitive complexity. Simple informational requests or basic routing tasks are automatically directed to smaller, highly cost-effective models. Conversely, if the middleware detects a computationally heavy analytical query or a conversation that has grown exceedingly complex, it automatically triggers a swap to a high-parameter frontier model.[16] This dynamic routing capability optimizes both system latency and financial expenditure without requiring manual intervention from the developer.[16]

## Hot-Swappable Tool Injection and API Orchestration

While dynamic prompts effectively define the SME-Plug's behavioral persona and conversational boundaries, its actual enterprise utility is derived entirely from the external tools and datasets it can access. To maintain the agility and security of the architecture, tools must

never be statically hardcoded into the agent's foundational definition. Instead, the system must utilize dynamic tool injection, adapting the available computational toolset based on real-time authentication states, user permissions, and the specific functional stage of a complex workflow.[15]

## FastAPI Dependency Injection Patterns

In modern enterprise Python architectures, dynamic tool injection is highly optimized through the implementation of web frameworks like FastAPI, which features a robust, deeply integrated Dependency Injection (DI) system.[17] Dependency injection is a software design pattern that deliberately decouples a class or path operation function from the instantiation of its underlying dependencies, allowing the FastAPI system to dynamically provide the required components at runtime.[18] This architectural approach heavily enhances code reusability, modularity, and the strict isolation of execution contexts.[19]

Within the SME-Plug framework, when a user submits a natural language request to a FastAPI endpoint, the dependency injection system instantaneously evaluates the user's authentication headers. Using Python libraries such as contextvars, the system securely manages context-specific state data, automatically injecting required authorization tokens directly into the agent's tool execution environment before the agent is even aware the tool exists.[20] This ensures that when the AI agent attempts to invoke a tool—such as querying a proprietary database or initiating a REST API call to a third-party service—it acts strictly within the permission boundaries of the invoking user, preventing unauthorized data exfiltration.[15]

FastAPI's advanced dependency handling mechanisms also allow for highly sophisticated resource lifecycle management using the yield keyword.[21] Dependencies structured with yield can initialize a secure connection to a database, pass that active connection object to the AI agent for the duration of the request processing, and then safely execute teardown logic immediately after the path operation completes.[21] This architectural pattern guarantees that tools are cleanly mounted and unmounted, preventing catastrophic resource leaks during extended, multi-turn AI interactions.[21] Furthermore, dependency functions can be programmatically merged to construct dynamic lists of capabilities, meaning the LLM only "sees" the specific tools it is explicitly authorized to use for that exact transaction.[15]

## Enterprise API Orchestration and System Connectors

The ultimate operational value of the SME-Plug lies in its seamless integration with core enterprise systems, transforming isolated generative capabilities into automated business processes.[23] The architecture requires a highly structured API layer to facilitate secure, deterministic communication between the agentic framework and external enterprise platforms.[23] AI agent integration platforms rely on comprehensive API specifications—detailing required HTTP methods, complex authentication parameters, highly structured JSON payload schemas, and explicit error response handling—to allow agents to interact with proprietary environments predictably.[23]

Integrating an AI agent with mission-critical systems such as an SAP ERP or a Salesforce CRM

requires high-fidelity, bidirectional connectors capable of handling both real-time data exchange and batch processing modes.[24] The architectural boundary between the deterministic enterprise application and the non-deterministic AI agent must be strictly enforced.[27] Embedding probabilistic agentic functionality directly inside legacy application architectures introduces unacceptable risk and unnecessary complexity.[27] Instead, agents must exist in an external orchestration layer, executing state changes via secure API endpoints utilizing standardized data mapping protocols.[24]

Through advanced integration platforms like MuleSoft or custom API management gateways, organizations can expose highly specific backend functionalities as encapsulated "agent-ready" actions.[25] This abstraction allows the SME-Plug to function as a collaborative digital worker, capable of triggering multi-step supply chain automations, updating Master/Transactional SAP data, or cross-referencing Salesforce inventory tables without ever possessing direct, uncontrolled access to the underlying database architecture.[24] This separation of concerns ensures that the AI agent merely acts as an intelligent router of API payloads, preserving the integrity of the core enterprise systems.

# Multi-Agent Orchestration Topologies: Routing, Handoffs, and Skills

As enterprise AI deployments scale from isolated proofs-of-concept to mission-critical infrastructure, single-agent architectures inevitably fail. When an individual LLM is burdened with an extensive system prompt covering dozens of disparate domains and is provided with hundreds of potential tools, the model suffers from severe context degradation. This results in unacceptable latency spikes, catastrophic token consumption, and a high failure rate in tool selection logic.[30] To resolve this fundamental limitation, the SME-Plug framework utilizes advanced multi-agent orchestration topologies, decomposing complex operational objectives into discrete, highly verifiable subtasks executed by specialized agents.[30]

The design of a multi-agent control plane typically falls into one of several architectural patterns, each offering highly distinct operational advantages in terms of token efficiency, parallelization capabilities, and context isolation.[30]

| Orchestration Pattern | Execution Mechanism | Latency and Token Consumption Profile | Optimal Enterprise Use Case |
|---|---|---|---|
| **Subagents** | A primary supervisor agent orchestrates highly specialized subagents as tools, managing all routing logic and final output synthesis. | High model calls (~8 per multi-turn request), strong isolation, consistent but high cost. | Highly secure environments requiring absolute boundary separation and strict hierarchical control between agent tasks.[30] |
| **Router** | A centralized routing | Medium model calls | Multi-domain queries |

| | | | |
|---|---|---|---|
| | step utilizes intent classification to dispatch queries to specific domain agents in parallel. | (~6), facilitates highly efficient concurrent execution. | that can be processed simultaneously before the final results are synthesized.[30] |
| **Handoffs** | State-driven transitions where execution control and conversational context are explicitly transferred between specialized agents via tool calls. | Low model calls (~5), sequential execution required, saves up to 40% of calls on repeat requests. | Multi-turn conversational workflows requiring deep, unbroken context preservation across shifting domain boundaries.[30] |
| **Skills** | A single coordinating agent dynamically loads specialized prompts and tools on-demand using a progressive disclosure architecture. | Low model calls (~5), but high token consumption due to constant context accumulation in the primary window. | Low-latency applications where the agent must adapt to thousands of potential schemas dynamically without breaking its core state.[30] |

## Deep Dive: Analyzing Router and Handoff Patterns

The **Router** pattern is highly effective for applications requiring aggressive horizontal scaling. An incoming enterprise query is intercepted by a lightweight intent classifier, which rapidly determines the required operational domains. If a user submits a highly complex query regarding both financial forecasting models and HR compliance regulations, the router triggers the 'Finance SME-Plug' and the 'HR SME-Plug' simultaneously in parallel.[32] This drastically reduces overall system latency and ensures that neither agent's context window is polluted by the instructions of the other. However, routers are inherently stateless; they struggle significantly with multi-turn, organic conversations where the user's intent shifts unpredictably over time.

In stark contrast, the **Handoff** pattern accurately mimics human organizational behavior and departmental collaboration.[34] Agents are organized in a swarm configuration, connected via a sophisticated parent-child architecture, and granted explicit "transfer" tools.[35] If the currently active agent reaches the boundary of its defined system prompt, it utilizes a standard tool call to invoke a specific Command object.[34] This command dynamically rewrites the underlying graph state, simultaneously determining the next node to run and transferring execution control to a more relevant domain agent.[34] Crucially, this transfer mechanism includes the full conversational context and shared state variables, ensuring the user is never forced to repeat previously established information.[35] Empirical testing of multi-agent systems demonstrates that stateful patterns like Handoffs reduce redundant model calls by up to 40%, generating massive cost savings in high-volume environments.[32] However, because execution must occur

strictly sequentially as state is passed from node to node, handoffs cannot leverage parallel processing.[32]

Advanced implementation of these patterns in frameworks like LangGraph also requires sophisticated checkpointing mechanisms to manage state interrupts.[36] When wrapping multi-turn conversational agents as tools within a supervisor architecture, managing the pause and resumption of execution requires highly robust, database-backed checkpointers (such as PostgreSQL integration via LangGraph Studio) to prevent state corruption during complex user interactions.[36]

### The Skills Pattern for Progressive Disclosure Architecture

For enterprise environments burdened by massive, highly complex data schemas—such as an internal SQL assistant that must interface with thousands of distinct database tables or data lakes—the **Skills** pattern offers the most resilient architecture.[33] Instead of transferring control between entirely distinct AI personas, a single, highly capable agent maintains continuous control while dynamically fetching required "skills" from an external tool registry.[33]

Utilizing progressive disclosure, the agent initially loads into memory with a minimal, lightweight system prompt. Upon analyzing the user's initial request, it executes a preliminary tool call to query a metadata registry, retrieving the specific database schema, access permissions, or operational logic required for that exact query.[33] While this design pattern incurs an initial latency penalty to perform the retrieval step, it fundamentally prevents the model's context window from being overwhelmed. This progressive loading enables the SME-Plug to operate effectively across an infinitely scaling, highly complex enterprise infrastructure, dynamically integrating various specialized models based on the immediate context.[33]

## Ensuring Determinism: RAG Evaluation and Grounding Metrics

The dynamic injection of tools and the sophisticated routing of multi-agent swarms provide the SME-Plug with immense operational capability, but its core objective remains absolute deterministic accuracy. To achieve this, the architecture relies heavily on advanced Retrieval-Augmented Generation (RAG) pipelines. RAG architectures deliberately bypass the LLM's inherently flawed internal parametric memory by forcing the model to generate responses strictly based on semantic chunks retrieved from a verified, continuously updated external enterprise database.[38]

However, merely bolting a vector database onto an LLM does not inherently solve the hallucination crisis. RAG pipelines are highly susceptible to retrieving irrelevant noise, failing to retrieve critical data, or failing to accurately synthesize the fetched text into a coherent response.[38] Transitioning an AI agent from a fragile prototype to a production-grade SME-Plug requires rigorous, highly automated evaluation using standardized algorithmic frameworks such as RAGAS (Retrieval-Augmented Generation Assessment Suite).[40] Traditional Natural Language Processing evaluation metrics, such as BLEU and ROUGE, are entirely insufficient for this task, as they evaluate surface-level n-gram text overlaps rather than assessing factual

accuracy, context relevance, or semantic correctness.[41]

## RAGAS Evaluation Metrics and Quantitative Analysis

The RAGAS framework fundamentally decomposes the evaluation of the RAG pipeline into two distinct analytical vectors: the mechanical performance of the context retriever, and the generative performance of the language model.[38]

**1. Context Retriever Evaluation: Precision and Recall Mechanics**

If the underlying retriever fails to surface the correct documents from the vector store, the LLM will inevitably fail to generate a correct response. This critical function is measured using highly specific metrics:

- **Context Precision:** This metric quantitatively evaluates the signal-to-noise ratio of the retrieval process.[42] It calculates the precise proportion of retrieved data chunks that are genuinely relevant to the user query. Critically, it utilizes rank-aware mathematical formulations, such as $Precision@k$ and Normalized Discounted Cumulative Gain ( $NDCG@k$ ), which heavily penalize systems that retrieve relevant documents but bury them at the bottom of the retrieval stack, out of sight of the LLM's primary attention mechanism.[42]
- **Context Recall:** This metric measures the absolute completeness of the retrieval system.[42] It compares the retrieved documents against a pre-annotated ground-truth dataset to determine what exact percentage of the necessary information was successfully fetched.[42] High context recall ensures that the LLM is not starved of the critical data required to form a complete, accurate conclusion.[42]
- **Context Entities Recall:** This specialized metric evaluates the system's ability to retrieve specific entities (names, dates, product codes) by calculating the recall based on the number of entities present in both the ground truth and the retrieved context relative to the ground truth alone.[43]

**2. Generator Evaluation: Response Groundedness and Faithfulness**

Once the pristine context is retrieved, the LLM must synthesize the information without fabricating external details or reverting to its pre-trained biases.

- **Faithfulness (Response Groundedness):** This is the ultimate, most critical measure of hallucination prevention within the SME-Plug.[40] It quantifies the absolute degree to which the generated content is derived strictly from the retrieved context.[43] A perfect faithfulness score indicates that the model relies entirely on external knowledge rather than its pre-trained parametric memory, effectively operating as a deterministic logic engine.[46]
- **Answer Correctness and Semantic Similarity:** This end-to-end evaluation metric gauges the overall accuracy of the generated answer by assessing the semantic resemblance between the generated response and the ideal, human-annotated ground truth.[45]

## Citation Architecture and Algorithmic Post-Processing

A defining, non-negotiable characteristic of an enterprise SME-Plug is its ability to provide explicit, verifiable source attribution. Users must be able to instantly audit the AI's reasoning. Industry studies consistently reveal that out-of-the-box RAG systems, without explicit attribution training, achieve citation accuracy rates of only 65% to 74%.[46] To bridge this critical reliability gap, the SME-Plug architecture must implement highly efficient post-processing algorithms.[47]

By utilizing sophisticated keyword matching, deep semantic similarity checks, and lightweight LLM-based verification techniques (such as fine-tuned models utilizing BERTScore), the system can algorithmically cross-check generated citations against the retrieved documents before returning the payload to the user.[47] Experimental implementations of this post-processing citation validation have demonstrated up to a 15.46% relative improvement in overall accuracy metrics.[47] This significant enhancement allows developers to achieve highly deterministic outputs while safely utilizing smaller language models that are approximately 12x more cost-effective and 3x faster in inference time, fundamentally altering the economics of enterprise AI deployment.[47]

# Guardrails and Compliance: Enforcing Architectural Boundaries

While highly optimized RAG pipelines provide the factual grounding for the SME-Plug, guardrails provide the structural safety net required for enterprise deployment. An AI guardrail is an invisible, multi-layered safety system—composed of explicit programmatic logic, strict behavioral policies, and advanced model-based classifiers working in concert—designed to intercept inputs and outputs, ensuring the agent remains strictly within defined operational boundaries.[48]

Without robust guardrails, an agentic system is highly vulnerable to malicious prompt injection attacks, catastrophic sensitive data leakage, and the unauthorized execution of destructive enterprise workflows.[49] Guardrails operate continuously across the entire lifecycle of the interaction, preventing models from leaking personally identifiable information (PII), hallucinating dangerous financial advice, or executing API calls that violate corporate governance.[48]

## Dual-Layer Guardrail Implementation Topology

Effective, production-grade SME-Plug design requires a hybrid approach, inextricably combining fast deterministic rules with nuanced model-based constraints.[49]

**1. Deterministic (Rule-Based) Guardrails:** Deterministic guardrails rely entirely on explicit programming logic, such as tightly defined Regular Expressions (regex) and hardcoded keyword blocklists.[49] These filters execute in mere milliseconds and provide an auditable, unyielding, and highly predictable layer of security.[49] They are optimally deployed for critical

tasks such as identifying and redacting social security numbers (PII filters), enforcing strict JSON formatting constraints on API payloads, or blocking predefined toxic language before it reaches the model.[50] Because they bypass the LLM entirely, they are highly cost-effective and latency-free, though they lack the nuance to detect sophisticated, multi-turn semantic violations.[49]

**2. Model-Based (Semantic) Guardrails:** To enforce highly complex behavioral constraints, the architecture utilizes secondary, highly specialized LLMs or fine-tuned classifiers.[49] These guardrails act as intelligent semantic routers and validators. For example, a customer support SME-Plug might be governed by a system prompt instructing it to "never give medical advice".[52] If a malicious user attempts to bypass this restriction via a complex, multi-turn hypothetical scenario, a rule-based regex system will almost certainly fail. A model-based guardrail, however, deeply analyzes the semantic intent of the user's query and the agent's proposed response. Advanced frameworks like NeMo Guardrails utilize a sophisticated four-part mechanism to achieve this: a *Checker* that scans the generated output for nuanced policy violations, a *Corrector* that refines or completely redacts the response, a *Rail* that manages the complex interaction workflow, and the *Guard* interface itself.[52]

### Active Safety Enforcement in Regulated Environments

The critical distinction between a passive filter and an active guardrail is paramount for high-stakes compliance tasks, such as assisting in complex ISO 9001 quality management audits or ISO 42001 AI management system reviews.[54] While an intelligent agent might proactively parse highly technical corporate documentation, flag deviations from established safety baselines, and automate compliance evidence collection [54], it must be actively and cryptographically constrained from altering compliance records autonomously.[55] Guardrails enforce a mandatory "human-in-the-loop" middleware circuit for any potentially destructive or legally binding action.[49] By implementing explicit safety enforcement mechanisms at runtime, rather than relying solely on flawed pre-execution risk assessments, the SME-Plug ensures that the agent cannot deviate from established workflows, ensuring a state of continuous, verifiable audit readiness.[51]

## Observability and the Data Infrastructure Layer

The ultimate differentiator between a theoretical proof-of-concept agent and a production-grade, highly autonomous SME-Plug is the underlying data infrastructure and observability layer. Despite rapid advancements in agentic frameworks, enterprise adoption statistics are stark: more than 40% of agentic AI initiatives are currently projected to be canceled by 2027 due to rising operational costs, unclear return on investment, and critically insufficient risk controls.[57] Furthermore, more than 80% of enterprise AI projects fail—a rate double that of traditional technology deployments.[57]

The primary catalyst for this massive failure rate is data degradation. With 91% of AI models experiencing severe quality degradation over time due to interaction with stale, incomplete, or highly fragmented data sources, the make-or-break factor is the data infrastructure pipeline

feeding the agent.[57] Successful AI engineering requires treating data infrastructure as a first-class architectural component, ensuring agents possess unified, strictly governed access to real-time data across diverse CRM, ERP, and API ecosystems.[57]

### Instrumentation and Framework Agnosticism

Compounding the data quality issue is the severe lack of visibility into highly nonlinear agentic workflows. Agentic systems frequently involve orchestrator agents delegating complex tasks to specialized sub-agents, executing operations in parallel, and following complex fan-in/fan-out execution patterns.[58] Traditional observability tools struggle to render these workflows understandable, as internal memory states and intermediate fallback chains are rarely exposed in standard application logs.[58] Without consistent methodologies to capture these behaviors, developers are forced to resort to tedious, error-prone manual instrumentation, wrapping agent logic and injecting trace metadata by hand.[58]
To resolve this, modern SME-Plug architectures increasingly leverage advanced GenAI frameworks like Pydantic AI, which tightly integrates with OpenTelemetry observability platforms such as Logfire.[59] Built upon the foundational Pydantic validation layer heavily utilized by the OpenAI SDK, LangChain, and LlamaIndex, this approach guarantees framework-agnostic real-time debugging and evals-based performance monitoring.[59] By enforcing strict data validation at the core framework level, developers can confidently scale multi-agent architectures, ensuring that complex outputs generated by varying models conform absolutely to expected schemas, bridging the gap between experimental AI and enterprise-grade reliability.[59]

## Hackathon Evaluation Rubric and Utility Plugin Blueprints

To effectively and rigorously evaluate the submissions for the SME-Plug MVP hackathon, technical judges must look far beyond superficial conversational abilities and assess the architectural resilience of the underlying data pipelines and middleware logic.[31] Organizations conducting highly successful internal AI hackathons frequently discover that the highest-value projects are not conversational chatbots, but sophisticated "utility plugins"—knowledge-aware workflow assistants that execute deep, highly contextual system integrations.[60]

### Real-World Utility Examples for Hackathon Ideation

Successful AI hackathons hosted by enterprise organizations highlight the incredibly diverse applications of the SME-Plug concept when properly executed [60]:

- **Cybersecurity and Vulnerability Agents:** An SME-Plug that continuously ingests real-time security advisories (such as Cisco's OpenVuln), cross-references them against an enterprise's live hardware and software inventory state via CLI commands, and algorithmically maps the severity of vulnerabilities, automating days of manual spreadsheet analysis and generating structured audit reports.[61]

- **Human Resources Logistics Assistants:** A deterministic agent designed to handle the time-consuming hiring process. It securely reads resumes, assesses job fit based on predefined skill matrices, executes data extraction directly to Airtable, and formulates strictly standardized interview questions while adhering to non-negotiable anti-bias guardrails.[62]
- **Emergency Response Multi-Agent Hubs:** A sophisticated multi-agent framework where an orchestrator dynamically routes queries between specialized agents (e.g., a Flooding SME, a Public Health SME, and a Winter Storm SME), dynamically coordinating real-world resources like municipal pumps, vaccines, and evacuation logistics during a rapidly evolving public crisis.[63]
- **Travel Planning and Experience Agents:** Complex planners capable of searching and booking flights using natural language queries across multiple disparate APIs. Crucially, these agents demonstrate extreme resilience by maintaining a local fallback database to ensure operational continuity in the event of upstream API outages, preserving the conversational flow and contextual memory across tasks.[64]

## Architectural Scoring Rubric

To ensure the submitted SME-Plugs meet the strict, unyielding requirements of enterprise determinism, the following rubric must be strictly applied during the hackathon evaluation phase:

| Evaluation Category | Key Metrics and Measurement Criteria | Architectural Verification Method |
|---|---|---|
| **Contextual Agility** | Measurement of the speed and accuracy of system prompt swapping and capability transitions based on state changes. | Deep inspection of LangChain/FastAPI middleware layers; empirical verification that context boundaries remain strictly isolated during intense, adversarial multi-turn testing.[15] |
| **Retrieval Precision** | Quantitative analysis of the underlying RAG mechanism's accuracy (Context Precision, Context Recall, Entity Recall). | Assessment using the RAGAS framework against a hidden golden dataset; require mathematical Precision@k scores strictly exceeding 0.90.[42] |
| **Response Groundedness** | Verification of the complete elimination of hallucinated facts and fictitious data generation. | Measurement of "Faithfulness" via automated LLM-as-a-judge scoring; automatic penalization or disqualification for any output not directly supported by the provided retrieval context.[43] |

| | | |
|---|---|---|
| **Citation Validity** | Assessment of source attribution accuracy and the efficiency of real-time post-processing verification algorithms. | Verification that every factual claim includes an inline citation; simulated execution of a "package hallucination" code-generation attack to test system resilience and filtering.[13] |
| **Guardrail Enforcement** | The demonstrable capability of the system to deflect out-of-bounds queries, prevent prompt injection, and redact PII. | Execution of adversarial testing (red-teaming) targeting restricted APIs; evaluation of the seamless integration of both deterministic regex and semantic LLM checkers within the pipeline.[49] |
| **Tool Orchestration** | Stability of dynamic tool loading via dependency injection and state preservation. | Comprehensive audit of the FastAPI Depends and yield implementations; verification of the seamless execution and proper teardown of simulated SAP/Salesforce API calls without memory leaks or state corruption.[17] |

# Conclusion

The era of relying solely on monolithic, generalized Large Language Models is rapidly yielding to the absolute necessity of compound, agentic workflows. As enterprises increasingly face the severe, escalating risks of generative AI—ranging from lethal medical omissions and highly embarrassing legal fabrications to sophisticated, malware-laden software supply chain exploits—the demand for highly constrained, strictly deterministic architectures has become paramount.

The Subject Matter Expert Plugin (SME-Plug) represents the definitive architectural solution to this ongoing crisis. By treating the LLM merely as a flawed but highly capable cognitive reasoning engine, and systematically surrounding it with dynamic context engineering, FastAPI-driven dependency injection, sophisticated RAG evaluation metrics, and multi-layered, active guardrails, organizations can safely extract the immense economic value of generative AI without exposing themselves to unacceptable operational risk.

The challenge laid out in this report demands the seamless synthesis of these highly complex, disparate technologies into a singular, hot-swappable MVP framework. The engineering teams that successfully master this orchestration layer will not merely build a highly capable software application; they will definitively outline the structural foundation for the autonomous, highly verifiable, and ultimately secure enterprise AI ecosystems of the future.

**Works cited**


1. From LLMs to Agentic AI: The Evolution of AI Decisioning Platforms - Sapiens, accessed February 28, 2026, https://sapiens.com/resources/blog/from-llms-to-agentic-ai-the-evolution-of-decision-management/
2. The Shift from Models to Compound AI Systems - Berkeley AI Research, accessed February 28, 2026, https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/
3. The Rise of Agentic AI: A Review of Definitions, Frameworks, Architectures, Applications, Evaluation Metrics, and Challenges - MDPI, accessed February 28, 2026, https://www.mdpi.com/1999-5903/17/9/404
4. AI Agents vs. Agentic AI: A Conceptual Taxonomy, Applications and Challenges - arXiv, accessed February 28, 2026, https://arxiv.org/html/2505.10468v1
5. The AI consolidation is coming: Gartner's latest reports validate the shift from LLM-Building to agentic workflows - Pangeanic Blog, accessed February 28, 2026, https://blog.pangeanic.com/the-ai-consolidation-is-coming-gartner-latest-reports-validate-the-shift-from-llm-building-to-agentic-workflows
6. LLM hallucinations and failures: lessons from 5 examples - Evidently AI, accessed February 28, 2026, https://www.evidentlyai.com/blog/llm-hallucination-examples
7. Hallucinating Law: Legal Mistakes with Large Language Models are Pervasive, accessed February 28, 2026, https://hai.stanford.edu/news/hallucinating-law-legal-mistakes-large-language-models-are-pervasive
8. Hallucination-Free? Assessing the Reliability of Leading AI Legal Research Tools - Daniel E. Ho, accessed February 28, 2026, https://dho.stanford.edu/wp-content/uploads/Legal_RAG_Hallucinations.pdf
9. A framework to assess clinical safety and hallucination rates of LLMs for medical text summarisation - PMC, accessed February 28, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC12075489/
10. Why Gen AI Hallucinations Can Be a Nightmare for Logistics—And How to Fix Them, accessed February 28, 2026, https://runink.org/blog/gen-ai-hallucinations-logistics-solutions/
11. 10 AI Hallucinations Every Company Must Avoid | Galileo, accessed February 28, 2026, https://galileo.ai/blog/ai-hallucination-examples
12. AI-Driven Hallucinations in Cyber Supply Chain Lead to New Threat: Slopsquatting, accessed February 28, 2026, https://www.captechu.edu/blog/ai-driven-threats-in-software-supply-chains
13. Package Hallucination: The Latest, Greatest Software Supply Chain Security Threat? - IDC, accessed February 28, 2026, https://www.idc.com/resource-center/blog/package-hallucination-the-latest-greatest-software-supply-chain-security-threat/
14. AI Hallucinations Could Cause Nightmares for Your Business: 10 Steps You Can Take to Safeguard Your GenAI Use - Fisher Phillips, accessed February 28, 2026, https://www.fisherphillips.com/en/insights/insights/ai-hallucinations-could-cause-

nightmares-for-your-business
15. Context engineering in agents - Docs by LangChain, accessed February 28, 2026, https://docs.langchain.com/oss/python/langchain/context-engineering
16. Agents - Docs by LangChain, accessed February 28, 2026, https://docs.langchain.com/oss/python/langchain/agents
17. Dependencies - FastAPI, accessed February 28, 2026, https://fastapi.tiangolo.com/tutorial/dependencies/
18. Mastering Dependency Injection in FastAPI: Clean, Scalable, and Testable APIs | by Aziz Marzouki | Medium, accessed February 28, 2026, https://medium.com/@azizmarzouki/mastering-dependency-injection-in-fastapi-clean-scalable-and-testable-apis-5f78099c3362
19. Dependency Injection in FastAPI - GeeksforGeeks, accessed February 28, 2026, https://www.geeksforgeeks.org/python/dependency-injection-in-fastapi/
20. Passing an Authorization Token to the tool & avoiding the LLM. · langchain-ai langserve · Discussion #534 - GitHub, accessed February 28, 2026, https://github.com/langchain-ai/langserve/discussions/534
21. Advanced Dependencies - FastAPI, accessed February 28, 2026, https://fastapi.tiangolo.com/advanced/advanced-dependencies/
22. Dynamic Dependencies / programmatically trigger dependency injection in FastAPI, accessed February 28, 2026, https://stackoverflow.com/questions/70168960/dynamic-dependencies-programmatically-trigger-dependency-injection-in-fastapi
23. Agent Orchestration for API Driven Workflows - Boomi, accessed February 28, 2026, https://boomi.com/blog/agent-orchestration-for-ai-workflows/
24. AI Agent Integrations: Unlock Efficiency (2026) - Salesforce, accessed February 28, 2026, https://www.salesforce.com/agentforce/ai-agent-integrations/
25. Salesforce-SAP Connector, accessed February 28, 2026, https://appexchange.salesforce.com/appxListingDetail?listingId=a0N4V00000FpUgaUAF
26. Build AI Agent with External APIs in Salesforce Agentforce | Real-Time Integration Tutorial, accessed February 28, 2026, https://www.youtube.com/watch?v=lxhWFIdERbo
27. The Agentic Enterprise - The IT Architecture for the AI-Powered Future | Agentforce | Fundamentals - Architects | Salesforce, accessed February 28, 2026, https://architect.salesforce.com/docs/architect/fundamentals/guide/agentic-enterprise-it-architecture
28. Difference Between APIs, Connectors and Integration Applications - MuleSoft, accessed February 28, 2026, https://www.mulesoft.com/api/management/difference-between-apis-connectors-integration-applications
29. AI Agent Orchestration for Enterprise Workflow Efficiency - Moveworks, accessed February 28, 2026, https://www.moveworks.com/us/en/resources/blog/improve-workflow-efficiency-with-ai-agent-orchestration
30. Multi-agent - Docs by LangChain, accessed February 28, 2026,

https://docs.langchain.com/oss/python/langchain/multi-agent
31. AI Agent Workflow Orchestration: A Complete Production Implementation Guide - Medium, accessed February 28, 2026, https://medium.com/@dougliles/ai-agent-workflow-orchestration-d49715b8b5e3
32. Choosing the Right Multi-Agent Architecture - LangChain Blog, accessed February 28, 2026, https://blog.langchain.com/choosing-the-right-multi-agent-architecture/
33. Build a SQL assistant with on-demand skills - Docs by LangChain, accessed February 28, 2026, https://docs.langchain.com/oss/python/langchain/multi-agent/skills-sql-assistant
34. How Agent Handoffs Work in Multi-Agent Systems | Towards Data Science, accessed February 28, 2026, https://towardsdatascience.com/how-agent-handoffs-work-in-multi-agent-systems/
35. Understanding multi-agent handoffs - YouTube, accessed February 28, 2026, https://www.youtube.com/watch?v=WTr6mHTw5cM
36. Multi-Agent Pattern: Tool Calling vs Handoffs for Multi Turn Conversations with Interrupts : r/LangChain - Reddit, accessed February 28, 2026, https://www.reddit.com/r/LangChain/comments/1p1ayyp/multiagent_pattern_tool_calling_vs_handoffs_for/
37. Enterprise Guide to Dynamic Tool Loading Agents - Sparkco, accessed February 28, 2026, https://sparkco.ai/blog/enterprise-guide-to-dynamic-tool-loading-agents
38. RAG Evaluation Metrics: Best Practices for Evaluating RAG Systems - Patronus AI, accessed February 28, 2026, https://www.patronus.ai/llm-testing/rag-evaluation-metrics
39. Building Production-Grade RAG Systems for Document AI: What It Actually Takes, accessed February 28, 2026, https://hackernoon.com/building-production-grade-rag-systems-for-document-ai-what-it-actually-takes
40. Best Practices in RAG Evaluation: A Comprehensive Guide - Qdrant, accessed February 28, 2026, https://qdrant.tech/blog/rag-evaluation-guide/
41. RAGAS for RAG in LLMs: A Comprehensive Guide to Evaluation Metrics. | by Karthikeyan Dhanakotti, accessed February 28, 2026, https://dkaarthick.medium.com/ragas-for-rag-in-llms-a-comprehensive-guide-to-evaluation-metrics-3aca142d6e38
42. How to Evaluate RAG: Metrics, Evals, and Best Practices - Agenta, accessed February 28, 2026, https://agenta.ai/blog/how-to-evaluate-rag-metrics-evals-and-best-practices
43. Metrics - Ragas, accessed February 28, 2026, https://docs.ragas.io/en/latest/concepts/metrics/index.html
44. A complete guide to RAG evaluation: metrics, testing and best practices - Evidently AI, accessed February 28, 2026, https://www.evidentlyai.com/llm-guide/rag-evaluation
45. A Beginner's Guide to Evaluating RAG Pipelines Using RAGAS - Analytics Vidhya,

accessed February 28, 2026, https://www.analyticsvidhya.com/blog/2024/05/a-beginners-guide-to-evaluating-rag-pipelines-using-ragas/
46. Complete Guide to RAG Evaluation: Metrics, Methods, and Best Practices for 2025, accessed February 28, 2026, https://www.getmaxim.ai/articles/complete-guide-to-rag-evaluation-metrics-methods-and-best-practices-for-2025/
47. CiteFix: Enhancing RAG Accuracy Through Post-Processing Citation Correction - arXiv, accessed February 28, 2026, https://arxiv.org/html/2504.15629v1
48. How to Build Guardrails for AI Applications | Galileo, accessed February 28, 2026, https://galileo.ai/blog/ai-guardrails-framework
49. Guardrails - Docs by LangChain, accessed February 28, 2026, https://docs.langchain.com/oss/python/langchain/guardrails
50. What is the difference between guardrails and filters in LLMs? - Milvus, accessed February 28, 2026, https://milvus.io/ai-quick-reference/what-is-the-difference-between-guardrails-and-filters-in-llms
51. \tool: Customizable Runtime Enforcement for Safe and Reliable LLM Agents - arXiv, accessed February 28, 2026, https://arxiv.org/html/2503.18666v1
52. Guardrails for AI Agents - UX Planet, accessed February 28, 2026, https://uxplanet.org/guardrails-for-ai-agents-24349b93caeb
53. What are AI guardrails? - McKinsey, accessed February 28, 2026, https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-are-ai-guardrails
54. How To Use Agentic AI To Support Integrated ISO Audits - Fieldguide, accessed February 28, 2026, https://www.fieldguide.io/resource-articles/how-to-use-ai-to-support-integrated-iso-audits
55. Will Agentic AI Replace ISO Certification Auditors? Here's the Truth! - CertBetter, accessed February 28, 2026, https://certbetter.com/blog/will-agentic-ai-burn-out-auditors-strategies-to-bridge-the-assurance-talent-gap
56. AI Agent Development Services by *instinctools, accessed February 28, 2026, https://www.instinctools.com/ai-agent-development-services/
57. Enterprise AI Agent Engineering & Data Infrastructure | Informatica, accessed February 28, 2026, https://www.informatica.com/resources/articles/enterprise-ai-agent-engineering.html
58. Monitor, troubleshoot, and improve AI agents with Datadog, accessed February 28, 2026, https://www.datadoghq.com/blog/monitor-ai-agents/
59. Pydantic AI - Pydantic AI, accessed February 28, 2026, https://ai.pydantic.dev/
60. From Ideas to Action: Highlights from Our AI Agents Hackathon | RealityMine, accessed February 28, 2026, https://www.realitymine.com/articles/from-ideas-to-action-highlights-from-our-ai-agents-hackathon

61. Itential AI Hackathon: 17 Agents. 31 Tool Bindings. 10+ Projects. One Very Big Signal., accessed February 28, 2026, https://www.itential.com/blog/itential-ai-hackathon-17-agents-31-tool-bindings-10-projects-one-very-big-signal/
62. Inside our hackathon: 10 creative AI Agent examples you can build with Make, accessed February 28, 2026, https://www.make.com/en/blog/ai-agents-hackathon-2025
63. New York AI Hackathon Use Cases - Complete implementation guides for building AI solutions using Microsoft technologies, accessed February 28, 2026, https://github.com/msftsean/ai-hackathon-use-cases
64. Hackathon Winners Bring Agentic AI to Life with the NVIDIA NeMo Agent Toolkit, accessed February 28, 2026, https://developer.nvidia.com/blog/hackathon-winners-bring-agentic-ai-to-life-with-the-nvidia-nemo-agent-toolkit/