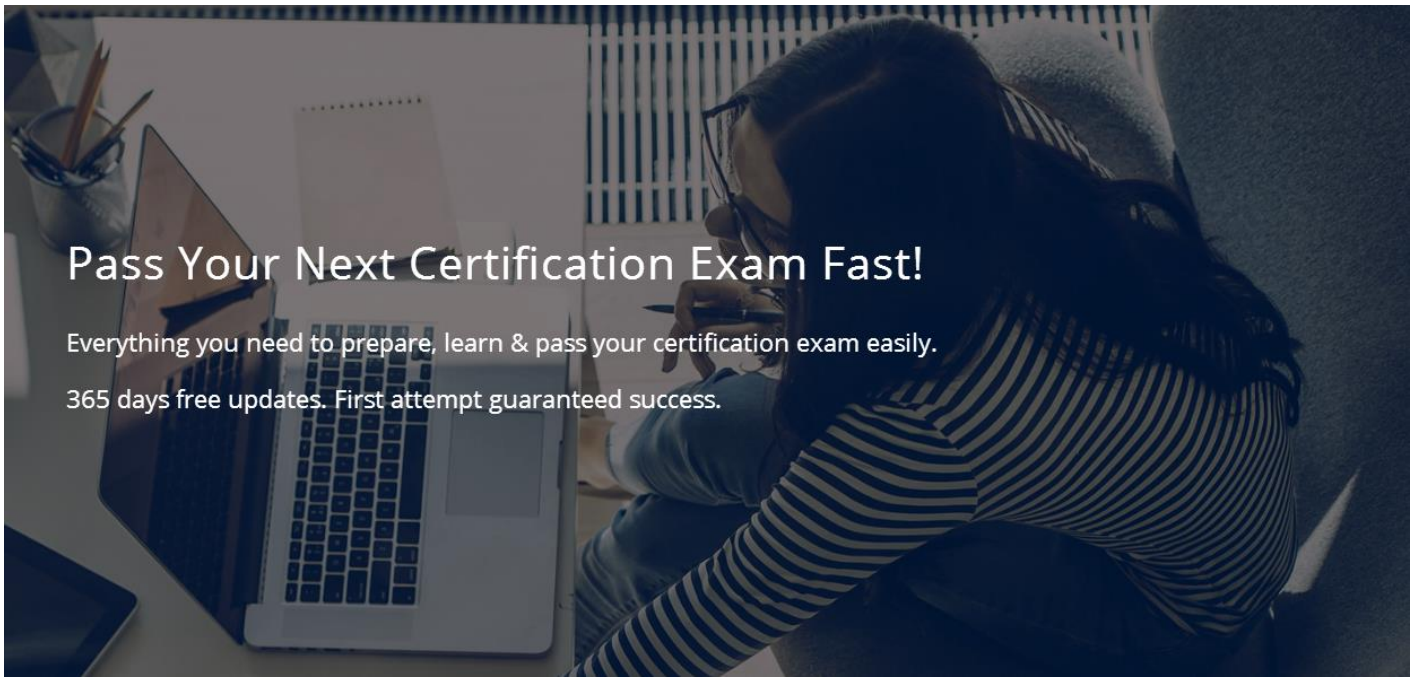
UpdateDumps



Choose the version that fits your needs	PDF Version	Desktop Test Engine	Online Test Engine
Latest and Up-to-Date exam dumps with real exam questions answers.	✓	✓	✓
Get 12-Months free updates without any extra charges.	✓	✓	✓
Experience same exam environment before appearing in the certification exam.	✗	✓	✓
100% exam passing guarante in the first attempt.	✓	✓	✓
20% discount on more than one license and 30% discount on 5+ license purchases.	✗	✓	✓
100% secure purchase on SSL.	✓	✓	✓
Completely private purchase without sharing your personal info with anyone.	✓	✓	✓

Exam : **NCP-AAI**

Title : Agentic AI

Vendor : NVIDIA

Version : DEMO

NO.1 A team is designing an AI assistant that helps users with travel planning. The assistant should remember user preferences, build personalized itineraries, and update plans when users provide new requirements.
Which approach best equips the AI assistant to provide personalized and adaptive travel recommendations?

A. Using a single-step question-answering system enhanced with session-level keyword tracking to improve relevance during ongoing interactions.

B. Designing the assistant to handle each user request independently, while using implicit signals within each session to suggest relevant options.

C. Engineering multi-step reasoning frameworks with persistent memory systems to store and utilize user preferences.

D. Providing the same set of travel options to every user but sorting them based on recent popular destinations.

Answer: C

Explanation:
The NVIDIA implementation angle is not cosmetic here: long-running agents should retrieve compact relevant context instead of replaying the entire conversation history into every call. Travel personalization depends on persistent preferences and multi-step plan updates. A single-turn answerer cannot adapt itineraries as constraints change. From an NVIDIA systems-engineering lens, Option C aligns with the way agentic services should be decomposed and measured. The selected option specifically C states "Engineering multi- step reasoning frameworks with persistent memory systems to store and utilize user preferences.", which matches the operational requirement rather than a superficial wording match. The correct implementation surface is checkpointed state keyed by session or user, with schemas that preserve only the fields the workflow needs later. The losing choices mostly optimize for short-term convenience; unbounded memory creates privacy, relevance, and performance problems unless persistence is deliberate. This choice gives engineering teams the knobs they need for continuous tuning after deployment. The memory policy should define what is persisted, what is summarized, and what is discarded to avoid both context loss and prompt bloat.

NO.2 When evaluating coordination failures in a multi-agent system managing distributed manufacturing workflows, which analysis approach best identifies state management and planning synchronization issues?

A. Assess synchronization methods during design reviews and use simulations to evaluate coordination across representative workflow scenarios.

B. Monitor agent outputs individually to confirm local correctness and examine results of specific workflow steps.

C. Deploy distributed state tracing across agents, analyze transition timing, study communication overhead, and verify synchronization accuracy.

D. Track workflow throughput and task completions to measure performance trends and highlight workflow outcomes.

Answer: C

NO.3 Which two orchestration methods are MOST suitable for implementing complex agentic workflows that require both external data access and specialized task delegation? (Choose two.)

A. Retrieval-based orchestration for external data

B. Manual workflow coordination without automation
C. Prompt chaining to accomplish state management
D. Agentic orchestration with specialized expert system delegation
E. Static rule-based routing with predefined pathways
Answer: A,D

NO.4 What is RAG Fusion primarily designed to achieve?
A. Creating a separate, dedicated database for storing all the retrieved chunks.
B. Minimizing the need for retrieval, allowing the LLM to generate responses directly from its internal knowledge.
C. Blending information from multiple retrieved chunks into a single response generated by the LLM.
D. Automatically translating and integrating all retrieved chunks into a single language.
Answer: C
Explanation:
RAG Fusion improves generation by blending evidence from multiple retrieved chunks. It is about combining retrieved context, not eliminating retrieval. In a GPU-backed agent deployment, Option C maps closest to how the NVIDIA stack expects orchestration, inference, and control policies to be separated. The selected option specifically C states "Blending information from multiple retrieved chunks into a single response generated by the LLM.", which matches the operational requirement rather than a superficial wording match.
The correct implementation surface is retriever isolation, vector index quality, reranking, freshness-aware ingestion, query expansion, and retrieval guardrails. This lines up with NVIDIA guidance because NeMo Guardrails can add retrieval rails around RAG context, while the serving layer remains independent from the vector database. The distractors fail because keyword-only retrieval misses semantic matches, while unfiltered concatenation can pollute the answer with weak evidence. This choice gives engineering teams the knobs they need for continuous tuning after deployment. The retrieval layer should be independently measured for recall, relevance, freshness, and latency before blaming the generator.

NO.5 In designing an AI workflow which of the following best describes a comprehensive approach to improving the performance of AI agents?
A. Implementing benchmarking pipelines, deploying physical agents and monitoring user engagement metrics
B. Implementing benchmarking pipelines, collecting user feedback, and tuning model parameters iteratively
C. Implementing benchmarking pipelines and incorporating a dynamic dataset for a real-time fall-back
D. Monitoring agents' throughput and time-to-first-token from the scoring engine
Answer: B
Explanation:
Agent improvement is iterative: benchmark, collect feedback, tune, regress-test, repeat. Monitoring token speed alone misses reasoning quality and task completion. The architecture implied by Option B is the one that survives real workloads: separate responsibilities, explicit contracts, and measurable runtime behavior.

The selected option specifically B states "Implementing benchmarking pipelines, collecting user feedback, and tuning model parameters iteratively", which matches the operational requirement rather than a superficial wording match. The correct implementation surface is trajectory-level evaluation, distributed tracing, task- completion metrics, latency breakdowns, and regression gates. In NVIDIA terms, NeMo Evaluator and agentic metrics focus on trajectories and goal completion, not only the fluency of the last response. The distractors fail because manual spot checks are useful but cannot replace regression tests across query classes, temporal drift, and tool failure modes. This choice gives engineering teams the knobs they need for continuous tuning after deployment. A strong evaluation setup must preserve both the trajectory and the final outcome so optimization does not improve one metric while damaging another.

NO.6 An autonomous vehicle company operates a multi-agent AI system across its fleet to process real-time sensor data, make driving decisions, and communicate with cloud infrastructure. The company needs fleet-wide monitoring to track GPU utilization, inference times, and memory usage, correlate performance with driving conditions and system load, and predict safety issues before they occur.
Which monitoring and observability approach would BEST meet these fleet-scale, safety-critical requirements?

A. Deploy NVIDIA NIM microservices with Prometheus integration, NVIDIA Nsight Systems profiling, and Kubernetes-native monitoring to provide detailed metrics, profiling, and container orchestration observability across the entire stack.

B. Implement layered application monitoring with distributed tracing, synthetic transaction monitoring, and custom dashboards to capture complex dependencies, transaction flow, and service-level performance trends across the fleet.

C. Implement comprehensive APM solutions with real-time baselines, automated root cause analysis, and fleet management integration to coordinate operational insights and performance management across thousands of vehicles.

D. Deploy enterprise telemetry using OpenTelemetry standards with machine learning-based anomaly detection, custom performance visualization, and automated alerting to deliver predictive operational insights and support proactive maintenance actions.

Answer: A
Explanation:
Option A is the right call because it gives the platform team levers to tune behavior without rewriting the entire agent loop. Within the NVIDIA stack, Triton dynamic batching and model configuration are where throughput and tail latency tradeoffs become controllable. The selected option specifically A states "Deploy NVIDIA NIM microservices with Prometheus integration, NVIDIA Nsight Systems profiling, and Kubernetes-native monitoring to provide detailed metrics, profiling, and container orchestration observability across the entire stack.", which matches the operational requirement rather than a superficial wording match.
NIM, Prometheus, Nsight, and Kubernetes observability cover GPU, inference, and orchestration layers. That is the best NVIDIA-specific fleet monitoring answer. The runtime should therefore be built around dynamic batching, model instance tuning, concurrency control, precision optimization, KV-cache-aware LLM serving, and end-to-end latency waterfalls. The distractors fail because sequential microservices can add avoidable hops and tail latency even when every individual model looks fast. The answer is therefore about engineered control planes, not simply model capability.

NO.7 After deploying a financial assistant agent, users report occasional inconsistencies in how transactions are categorized.
What is the best first step for diagnosing the issue?
A. Review and modify prompt temperature to enhance precision
B. Review and retrain the model with more financial datasets
C. Implement agent memory reset after each session
D. Review tool call inputs and outputs in recent session logs
Answer: D
Explanation:
The runtime should therefore be built around a memory hierarchy that balances retrieval latency, relevance, privacy, and context-window cost. This is a lifecycle problem, not a wording problem, and Option D gives the team a controllable lifecycle for the agent behavior. Transaction categorization depends on tool inputs and outputs. Before retraining, inspect recent traces to see whether the model received incorrect or incomplete structured data. For a production build, memory is an orchestration concern as much as a model concern, because the agent must decide what to keep, retrieve, and forget. The selected option specifically D states
"Review tool call inputs and outputs in recent session logs", which matches the operational requirement rather than a superficial wording match. The rejected options are weaker because sending full history every turn inflates latency and cost, while stateless prompts lose unresolved tasks, user preferences, and multi-step plan continuity. The answer is therefore about engineered control planes, not simply model capability.

NO.8 You are building an agent that performs financial analysis by retrieving and processing structured data from a client's internal SQL database. The agent must handle occasional connection errors and retry the query up to a few times before failing gracefully.
Which approach best meets these requirements?
A. Use structured tool calls with built-in retry handling and timed delays inside the tool wrapper
B. Use few-shot prompting to guide the agent's conversation flow and manually retry failed API responses
C. Use a reactive agent pattern that retries the query after a user confirms a retry attempt
D. Use memory to track the number of failed attempts and apply it in later retries
Answer: A
Explanation:
A tool wrapper is the right place for retry count, delays, and graceful failure. Prompting the model to retry manually is unreliable engineering. Option A fits the operating model because the problem describes an agent that must remain adaptive under changing inputs and infrastructure conditions. The selected option specifically A states "Use structured tool calls with built-in retry handling and timed delays inside the tool wrapper", which matches the operational requirement rather than a superficial wording match. The durable control mechanism is schema-bound tool invocation, typed parameters, timeout envelopes, retry policy, and traceable function execution. This lines up with NVIDIA guidance because the Agent Toolkit model is to expose tools as reusable workflow components; that is what makes multi-tool agents testable under schema changes. The distractors fail because embedding tools inside the agent loop makes security review, timeout handling, and version control unnecessarily difficult. For certification purposes, read the question as asking for controlled autonomy, not raw LLM creativity.

NO.9 Your team notices a spike in failed tool calls from a deployed workflow agent after a recent API schema update. The agent still returns outputs, but many are irrelevant or incomplete.
Which maintenance task should be prioritized to restore accurate behavior?
A. Reset the agent's long-term memory and reinitialize logs.
B. Update the tool function specifications and re-test action sequences.
C. Increase model temperature to encourage tool exploration.
D. Reduce tool retrieval vector similarity threshold to broaden context.
Answer: B
Explanation:
The selected option specifically B states "Update the tool function specifications and re-test action sequences.", which matches the operational requirement rather than a superficial wording match. For this scenario, Option B is defensible because it exposes the control plane that a senior engineer can test, scale, and harden. Operationally, the design depends on tool contracts that can be versioned, tested, and observed independently from the reasoning loop. A schema update breaks the tool contract. The first repair is to update function specifications and retest action sequences, not adjust randomness or memory. That is why the other options are traps: manual tool wiring scales poorly as the catalog grows and usually fails silently when a vendor updates parameters or response fields. Within the NVIDIA stack, NeMo Agent Toolkit treats agents, tools, and workflows as composable functions, so tool-calling agents can choose from names, descriptions, and schemas rather than guessed endpoints. It also creates clean evidence for audits, incident review, and root-cause analysis when behavior drifts.

NO.10 You're managing an agentic AI responsible for customer support ticket triage. The agent has been consistently accurate in routing tickets to the appropriate departments. However, a team leader has noticed a significant increase in the number of tickets requiring "escalation" - cases where the agent initially misclassified a complex issue as a simple, routine one, leading to delays and frustrated customers.
What would be an appropriate first step in resolving this issue?
A. Analyzing the agent's decision-making process, focusing on the specific criteria it uses to classify tickets, and identifying potential biases or blind spots.
B. Adjusting the agent's reward function to prioritize speed of resolution over accuracy, as a first step in analysis of the problem.
C. Increasing the agent's autonomy, granting it more decision-making power during triage to improve its efficiency.
D. Conducting a "red-teaming" exercise, having human agents deliberately create complex and ambiguous scenarios to analyze the agent's robustness.
Answer: A
Explanation:
Escalation drift starts in decision criteria. Before changing autonomy or reward functions, inspect classification logic, feature cues, and examples that trigger "routine" versus "complex." Option A wins because it optimizes the system boundary around the risky component rather than hoping the base model behaves consistently. The selected option specifically A states "Analyzing the agent's decision-making process, focusing on the specific criteria it uses to classify tickets, and identifying potential biases or blind spots.", which matches the operational requirement rather than a superficial wording

match. The durable control mechanism is schema-bound tool invocation, typed parameters, timeout envelopes, retry policy, and traceable function execution. The NVIDIA implementation angle is not cosmetic here: the Agent Toolkit model is to expose tools as reusable workflow components; that is what makes multi-tool agents testable under schema changes. The distractors fail because embedding tools inside the agent loop makes security review, timeout handling, and version control unnecessarily difficult. For certification purposes, read the question as asking for controlled autonomy, not raw LLM creativity.

NO.11 When analyzing inconsistent performance across a fleet of customer service agents handling similar queries, which evaluation approach most effectively identifies root causes and optimization opportunities?

A. Assess performance data from recently improved agents and highlight strong results, using outcome comparisons to identify areas with the greatest impact on service quality.

B. Average performance metrics across all agents as this will smooth individual variations, query distribution differences, and temporal factors affecting agent behavior and accuracy.

C. Deploy stratified evaluation sampling across agent variants, query complexity levels, and temporal patterns while tracking decision paths using comparative analytics.

D. Review performance across both high- and low-accuracy agent groups, comparing case outcomes and identifying patterns contributing to top and bottom results.

Answer: C

Explanation:

Option C is the right call because it gives the platform team levers to tune behavior without rewriting the entire agent loop. Within the NVIDIA stack, NeMo Evaluator and agentic metrics focus on trajectories and goal completion, not only the fluency of the last response. The selected option specifically C states "Deploy stratified evaluation sampling across agent variants, query complexity levels, and temporal patterns while tracking decision paths using comparative analytics.", which matches the operational requirement rather than a superficial wording match. Stratified sampling prevents hidden averages from masking failure pockets.

Query complexity and time patterns often explain why similar agents diverge. The implementation detail that matters is trajectory-level evaluation, distributed tracing, task-completion metrics, latency breakdowns, and regression gates. The distractors fail because manual spot checks are useful but cannot replace regression tests across query classes, temporal drift, and tool failure modes. That is the difference between an agent that works in a notebook and an agent that remains reliable in production.

NO.12 You are developing a RAG solution and have decided to use a classifier branch as part of your semantic guardrail system to assess the risk of generated text.

Which of the following is a key benefit of using a classifier branch compared to solely relying on prompt filtering?

A. Since a classifier branch does not require training, it can identify potentially problematic content.

B. Classifier branches primarily focus on detecting factual inaccuracies, rather than stylistic or harmful language.

C. Classifier branches can automatically adapt to new forms of harmful language.

D. Classifier branches eliminate the need for human oversight, thereby automating the safety process.

Answer: C

Explanation:

The decisive point is failure isolation: Option C keeps the agent's decision path observable instead of burying behavior inside one prompt or one service. Classifier branches are more semantic than prompt filters and can generalize beyond exact keywords. They still require validation and monitoring, but they catch patterns prompt text may miss. The runtime should therefore be built around policy enforcement placed around user inputs, retrieved context, tool execution, and generated responses. The selected option specifically C states "Classifier branches can automatically adapt to new forms of harmful language.", which matches the operational requirement rather than a superficial wording match. The alternatives would look simpler in a prototype, but ignoring protected attributes in prompts does not reliably prevent proxy bias or demographic inference in outputs. The stack-level anchor is clear: NVIDIA Guardrails can be integrated without throwing away existing LangChain-style workflows, preserving architecture while adding enforcement. The answer is therefore about engineered control planes, not simply model capability.