MockCert

Study guide | NVIDIA | NCP-AAI

NVIDIA-Certified Professional: Agentic AI

Notes for every domain of the NVIDIA-Certified Professional: Agentic AI exam, organized by the official exam guide's domains and weights.

By MockCert about 155 min read Updated November 16, 2026

CONTENTS

DOMAIN 1 · 15%
Agent Architecture and Design 2

DOMAIN 2 · 15%
Agent Development 18

DOMAIN 3 · 13%
Evaluation and Tuning 32

DOMAIN 4 · 5%
Deployment and Scaling 44

DOMAIN 5 · 10%
Cognition, Planning, and Memory 56

DOMAIN 6 · 10%
Knowledge Integration, and Data Handling 67

DOMAIN 7 · 7%
NVIDIA Platform Implementation 78

DOMAIN 8 · 7%
Run, Monitor, and Maintain 90

DOMAIN 9 · 5%
Safety, Ethics, and Compliance 101

DOMAIN 10 · 5%
Human-AI Interaction and Oversight 111

MockCert is an independent certification preparation platform. It is not affiliated with, endorsed by or operated by NVIDIA. Exam domains and weights follow the official exam guide. These notes were last updated on November 16, 2026; the notes at mockcert.com are kept up to date as the exam changes.

Domain 1 | 15% of the exam | 22 min read

Agent Architecture and Design

Overview

Every point below comes from NVIDIA's official documentation; each section lists its sources.

Design user interfaces for intuitive human-agent interaction.

Start here

THINK FIRST

How does an agent keep a dialog natural, ask the user for input mid-run, and know which user it is serving?

Why it matters: Questions describe a human-agent interaction and ask which flow, callback or identity rule makes it behave as intended.

Intuitive human-agent interaction rests on three pieces. Colang 2.0 flows shape the dialog, `UserInteractionManager` asks the user for input at runtime, and identity resolution maps every request to a user.

Key terms

Term	Meaning
user silent event	A pre-defined Colang 2.0 event for time-driven interaction, when the user says nothing.
UserInteractionManager	The NeMo Agent Toolkit component that requests user input while a workflow runs.
user_input_callback	The callback in `ContextState` that does the actual prompting; `SessionManager` typically sets it.
user_id	The stable UUID v5 a credential resolves to, on HTTP or WebSocket alike.
Per-user workflow	A workflow whose state is isolated by the resolved `user_id`, one instance per user.

Conversation loops in Colang

Build an ongoing dialog with a `while` loop. Inside the loop, the bot can act on its own when the user is silent. The pre-defined `user silent` event is how Colang 2.0 models this time-driven interaction.

Branching on what the user does

`when` / `or when` **branches a flow on several paths.** At the branching point, the flow monitors every branch at once and continues as soon as one matches. If the LLM produces an intent the script does not handle, the `unhandled user intent` flow is triggered.

EXAM TRAP

A `when` / `or when` block does not test branches one after another; it monitors all of them and moves on as soon as any branch matches.

Asking the user at runtime

`prompt_user_input` **asks a question and waits for the answer.** `UserInteractionManager` hands the prompting to the callback in `ContextState.user_input_callback`, usually set by `SessionManager`. The default callback is a no-op: it just returns the input text from the Interaction Content object.

EXAM TRAP

The default callback does not show the user a prompt; it is a no-op that returns the input text it receives.

Resolving who the user is

Every HTTP request and WebSocket connection gets an identity. Each credential maps to a stable UUID v5 `user_id`, whatever the transport or credential format. If several credential types arrive, the server uses the first one it finds. With no credential, the request proceeds anonymously.

EXAM TRAP

Sending several credential types does not combine them; the server uses the first one it finds and ignores the rest.

JWT claims and WebSocket sessions

For a JWT, the identity key is the `sub` claim. If `sub` is absent or empty, the resolver falls back to `email`, then `preferred_username`. On a WebSocket, a `user_id` resolved from an `auth_message` lasts for the session and covers every later workflow request on that connection.

Identity is not authentication

Identity resolution maps users; it does not verify them. JWTs are decoded with `verify_signature=False`, so the server trusts whatever credential arrives. In a per-user workflow, the resolved `user_id` isolates state, so each user gets their own workflow instance.

EXAM TRAP

A resolved `user_id` does not prove who the user is: JWT signatures are not verified, and any credential that arrives is trusted.

Key takeaways

- In Colang 2.0, `while` builds an interaction loop and `user silent` lets the bot act when the user says nothing.
- `UserInteractionManager` asks for input through `ContextState.user_input_callback`.
- A JWT identity comes from `sub`, then `email`, then `preferred_username`.
- Identity resolution is a mapping step, not authentication: JWT signatures are not verified.

Sources: Interaction Loop | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · nat.builder.user_interaction_manager — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · User Identity Resolution — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Implement reasoning and action frameworks (e.g., ReAct).

Start here

THINK FIRST

When should an agent think between its actions, and when should it plan everything before acting?

Why it matters: Questions describe an agent task and ask which reasoning-and-action pattern or tool calling setup fits it.

A reasoning and action framework mixes thinking with tool calls. A ReAct agent reasons between steps, while the toolkit's reasoning agent plans ahead and then invokes a function. Either way, tool calling lets the agent act during execution.

Key terms

Term	Meaning
ReAct agent	An agent that reasons between its steps rather than planning everything up front.
Reasoning agent	A toolkit agent that plans ahead of time and directly invokes an underlying function, reasoning on top of it.
Tool calling	Using a framework's own tool calling mechanism so an agent can invoke functions and tools while it runs.
Execution trace	A profiler view of a workflow's intermediate steps, LLM calls and tool calls.

Two ways to reason

ReAct reasons between steps; the reasoning agent plans ahead. Choose by when the thinking should happen. The reasoning agent wraps an underlying function and does its reasoning on top of that call.

ReAct agent and reasoning agent compared

Feature	ReAct agent	Reasoning agent
When it reasons	Between steps	Ahead of time, through planning
How it acts	Reasons again between each step it takes	Directly invokes an underlying function, reasoning on top

Setting up the reasoning agent

The reasoning agent needs an LLM that supports reasoning. Its prompts can be customized in the YAML config file to suit your use case.

EXAM TRAP

Not every LLM can back a reasoning agent. You must pick one that supports reasoning.

Tool calling across frameworks

Four frameworks plug in through their own tool interfaces. The toolkit fully supports tool calling for LangChain, LlamaIndex, Google ADK and Strands through these interfaces, so agents can call tools during execution.

Framework tool interfaces the toolkit fully supports

Framework	Tool interface	Support
LangChain	StructuredTool	Fully supported
LlamaIndex	FunctionTool	Fully supported
Google ADK	ADK FunctionTool	Fully supported
Strands	AgentTool	Fully supported

EXAM TRAP

LangChain uses StructuredTool, not FunctionTool. FunctionTool is the interface for LlamaIndex and Google ADK.

Adding LangChain support

LangChain/LangGraph is one of the integrated frameworks. Install it with `uv pip install "nvidia-nat[langchain]"` before using LangChain tools in a workflow.

Watching the loop run

The profiler can show workflow execution traces. Those traces include intermediate steps, LLM calls and tool calls, all viewed within the toolkit's profiler.

Key takeaways

- A ReAct agent reasons between steps; the reasoning agent reasons ahead of time by planning.
- The reasoning agent requires an LLM that supports reasoning.
- LangChain tools use StructuredTool, LlamaIndex uses FunctionTool, and Strands uses AgentTool.
- Profiler traces show intermediate steps, LLM calls and tool calls.

Sources: About Reasoning Agent — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Framework Integrations — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Configure agent-to-agent communication protocols for collaboration.

Start here

THINK FIRST

How should agent components call other services so that pooling, retries, security and tracing stay consistent?

Why it matters: Questions describe how an agent component talks to another service and ask which client, retry or security setting fits.

Agents use tools and interact with external systems. In NeMo Guardrails, actions and rails send those calls through the shared `nemoguardrails.http` boundary. That way one shared policy covers retries, testing and observability.

Key terms

Term	Meaning
nemoguardrails.http	The boundary to use for outbound HTTP requests from actions and library rails.
RetryPolicy	The client setting that enables retries; its `max_attempts` counts the initial request.
HTTPTLSConfig	TLS settings for a client, including a client certificate and key for mutual TLS.
Shared pooled client	One client that IORails owns and passes to every rail action that declares `http_client`.
Defense in depth	Layering controls; injection detection is meant to strengthen the other security controls.

One boundary for outbound calls

Send outbound HTTP through `nemoguardrails.http`. Never construct an `aiohttp`, `httpx`, `requests` or `urllib3` client inside an action. A private client bypasses the shared retry, testing and observability policy.

EXAM TRAP

Building an `httpx` client inside an action is not a harmless shortcut; it bypasses shared retry, testing and observability policy.

Default client settings

The default client pools up to 100 connections. Up to **20** of them are keep-alive connections. A client from `create_http_client` uses a **30**-second total timeout. By default, `http_call` raises `HTTPStatusError` for any status code of **400** or higher.

Retrying failed requests

Nothing is retried unless the client has a `RetryPolicy`. Its `max_attempts` includes the initial request. The safe default method set excludes POST, so add POST only for retry-safe or idempotent operations.

EXAM TRAP

`max_attempts` is not a count of retries alone; the initial request counts as one of the attempts.

Shared clients, TLS and tracing

IORails shares one pooled client across rail actions. It passes the client to each rail action that declares `http_client` and closes it when the runtime stops. `HTTPTLSConfig` supports a client certificate and key for mutual TLS. Tracing emits a CLIENT span named `HTTP {METHOD}`.

Securing agents that use tools

Agentic security guards agents that call tools and external systems. Injection detection strengthens other security controls as part of defense in depth. When an injection rule triggers, rejecting the output is the safest action for production. The context bloat rail checks user input and retrieved knowledge-base chunks, the text an agent does not write.

EXAM TRAP

Injection detection is not meant to be your only control; it adds to other security controls in a defense-in-depth strategy.

Key takeaways

- Route outbound HTTP from actions and rails through `nemoguardrails.http`, never a private client.
- The default client pools **100** connections, **20** of them keep-alive, with a **30**-second total timeout.
- Retries need a `RetryPolicy`; `max_attempts` includes the first request, and POST is excluded by default.
- In production, reject the output when an injection rule triggers.

Sources: Agentic Security | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · Outbound HTTP in Actions | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab)

Manage short-term and long-term memory for context retention.

Start here

THINK FIRST

How do you make sure an agent saves and recalls each user's memory on every turn, without mixing users up?

Why it matters: Questions describe an agent's memory needs and ask which wrapper, provider or identity setting keeps its context correct.

The NeMo Agent Toolkit Memory subsystem stores a user's conversation history, preferences and other long-term memory. You can give the agent memory tools, or wrap it with `auto_memory_agent` so capture and retrieval happen automatically.

Key terms

Term	Meaning
Long-term memory	A user's conversation history, preferences and similar data that the Memory subsystem keeps.
auto_memory_agent	A wrapper that adds automatic memory capture and retrieval to any agent.
Provider	The toolkit's name for a custom memory back-end.
user_id_resolver	Import path of a trusted zero-argument callable that returns the authenticated user's stable ID.
conversation_id	Lets conversation-aware back-ends keep one user's separate conversations apart.

Tool-based versus automatic memory

Memory tools rely on the LLM; the wrapper does not. With tool-based memory, the LLM may forget to call the tools. `auto_memory_agent` guarantees memory operations on every interaction and works with any agent type, such as ReAct, ReWOO, Tool Calling and Reasoning.

Memory tools and the automatic wrapper compared

Feature	Memory tools	auto_memory_agent
How memory runs	Through `add_memory`, `get_memory` and `delete_memory` tools	Automatically, with no explicit memory tool calls
Consistency	The LLM may forget to call the tools	Guaranteed on every interaction

EXAM TRAP

The wrapper does not wait for the LLM to decide to use memory; it runs memory operations on every interaction.

What the wrapper does each turn

Relevant memory context is injected before each agent call. Feature flags such as `save_user_messages_to_memory` and `retrieve_memory_for_every_response` all default to true. Its `search_params`, such as `mode` and `top_k`, are passed to `memory_editor.search()`.

Binding memory to a user

Every memory tool operation is bound to an identity. Set `user_id` in the tool configuration for a fixed service or single-user identity. For a multi-user application, use `user_id_resolver` instead.

EXAM TRAP

A fixed `user_id` does not suit a multi-user app; there, `user_id_resolver` returns each authenticated user's stable ID.

Missing identity and separate conversations

Automatic memory operations fail closed without an identity. Conversation-aware memory back-ends can also use `conversation_id` to isolate separate conversations for the same user.

EXAM TRAP

A missing identity does not lead to shared or anonymous memory; automatic memory operations fail closed instead.

Memory providers

The toolkit includes 4 memory providers, all as plugins. Mem0 comes from the `nvidia-nat-mem0ai` plugin. MemMachine comes from `nvidia-nat-memmachine` and is experimental.

EXAM TRAP

MemMachine is not a production choice: it is experimental and not recommended for production use.

Key takeaways

- The Memory subsystem keeps a user's conversation history, preferences and other long-term memory.
- `auto_memory_agent` captures and retrieves memory on every interaction, for any agent type.
- Use `user_id` for one fixed identity and `user_id_resolver` for multi-user apps.
- Automatic memory fails closed when no user identity is available.

Sources: Memory in NVIDIA NeMo Agent Toolkit — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · About Automatic Memory Wrapper Agent — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Orchestrate multi-agent workflows and coordination.

Start here

THINK FIRST

When should a multi-agent workflow use a fixed pipeline, a router, or a call to a remote agent over A2A?

Why it matters: Questions describe a coordination need and ask which control flow component or A2A setting fits it.

NeMo Agent Toolkit coordinates work in two ways. Control flow components such as the Router Agent and Sequential Executor steer one workflow from inside. The A2A protocol lets separate agents find and call each other.

Key terms

Term	Meaning
Sequential Executor	A control flow component that runs functions in a fixed order, passing each result to the next.
Router Agent	A control flow component that reads each request and sends it to the best-fitting branch.
A2A Protocol	An open Linux Foundation standard for how agents discover skills, hand off tasks and share data.
Agent Card	The description an A2A agent publishes; clients read the transport and address to use from it.
A2A skill	A capability an A2A agent offers; each tool in a published workflow becomes one.

Workflows and control flow

A workflow is usually an agent, but any function can serve as one. To decide how that workflow runs, the toolkit offers control flow components. The two to know are the Router Agent and the Sequential Executor.

Fixed pipeline or router

The Sequential Executor follows a set order; the Router Agent chooses a path. Pick the executor when the steps never change. Pick the router when each request may need a different branch.

Sequential Executor and Router Agent compared

Feature	Sequential Executor	Router Agent
How the path is set	Predetermined linear order	Chosen per request by analyzing it
What moves between steps	Each function's output feeds the next	The Router Node picks one branch to run

EXAM TRAP

The Sequential Executor does not need an LLM or an agent to decide the order. The order is fixed before the run starts.

How the router decides

The Router Node does the choosing in the routing phase. The router agent is built as a graph of **two** nodes. In the routing phase, the Router Node looks at the request and selects the branch that should handle it.

Publishing a workflow over A2A

`nat a2a serve` **listens on localhost port 10000 by default.** Once published, every tool in the workflow shows up as an A2A skill. The server runs at most **8** workflow executions at once unless you raise `max_concurrency`.

EXAM TRAP

Inside Kubernetes, the bind host and port are often not what clients reach. Set `public_base_url` so the Agent Card advertises the external address.

Calling a remote agent

The client uses the Agent Card's transport; the default is JSON-RPC over HTTP. For most cases, use the high-level `call()` API. It gives LLM-based agents a natural language way to talk to the remote agent.

Per-user workflows for A2A clients

Workflows that use A2A clients must be registered per user. Register them with `@register_per_user_function`, or use a builtin per-user workflow such as `per_user_react_agent`.

EXAM TRAP

A shared workflow like `react_agent` cannot use A2A client function groups directly. Switch to a per-user workflow instead.

Key takeaways

- Use the Sequential Executor for a fixed chain of functions and the Router Agent when each request needs its own branch.
- A published workflow exposes each tool as an A2A skill, served on port **10000** by default.
- The A2A server's `max_concurrency` defaults to **8** concurrent workflow executions.
- A2A client workflows must be per-user; shared ones such as `react_agent` cannot use the client function groups directly.

Sources: About Building NVIDIA NeMo Agent Toolkit Workflows — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · NVIDIA NeMo Agent Toolkit Workflow as an A2A Client — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · About Router Agent — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · NVIDIA NeMo Agent

Toolkit Workflow as an A2A Server — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Apply logic trees, prompt chains, and stateful orchestration for multi-step reasoning.

Start here

THINK FIRST

How does an agent break a multi-step problem into choices, chained prompts and tracked state so each step builds on the last?

Why it matters: Questions describe a multi-step task and ask which routing rule, prompt task or state-tracking feature fits it.

Multi-step reasoning splits one job into smaller LLM calls. A router picks the path, a chain of task prompts does the work, and a step manager keeps track of where the run is.

Key terms

Term	Meaning
Branch	One path a router agent can send a request down; it picks exactly one per request.
Task prompt	The prompt for one NeMo Guardrails LLM call, each of which performs a specific task.
Canonical user message	A short label for what the user meant, such as "express greeting" for "Hello there".
Intermediate step	A tracked unit of work in a toolkit run, started and later ended, managed by IntermediateStepManager.

Routing as a logic tree

The router agent selects exactly one branch per request. Its system prompt lists the branch names to pick from. If several branches could fit, it is told to choose the most specific one. The user prompt passes the earlier conversation history along with the request.

EXAM TRAP

The router does not split a request across several branches. It picks one, favoring the most specialized fit.

Planning ahead or step by step

A reasoning agent plans up front; a ReAct agent reasons between steps. Pick the agent type by when the thinking should happen. The reasoning agent also needs an LLM that supports reasoning.

EXAM TRAP

Any LLM will not do for a reasoning agent. It must be paired with an LLM that supports reasoning.

Guardrails as a prompt chain

Each NeMo Guardrails LLM call performs one specific task. For example, `generate_user_intent` turns a raw utterance into a canonical user message. The `general` task writes the next bot message from history when no dialog rails are defined.

Templates that carry state

Guardrails prompt templates fill in variables with Jinja2. Context variables that flows define can also be used inside these templates. Custom task prompts are rendered in an action with LLMTaskManager.

EXAM TRAP

A prompt whose mode does not match the configured `prompting_mode` is not used. When no prompt matches, the standard prompt is used.

Tracking state in a run

IntermediateStepManager tracks steps that have started but not ended. `get_outstanding_step_count` returns how many are still open. `push_intermediate_steps` injects steps into the event stream without touching the internal stack, for example to replay a remote workflow.

Key takeaways

- The router agent chooses exactly one branch, the most specific when several fit.
- A reasoning agent plans ahead of time and needs a reasoning-capable LLM.
- NeMo Guardrails chains task-specific LLM calls, with Jinja2 templates that can read flow context variables.
- If no prompt matches `prompting_mode`, Guardrails falls back to the standard prompt.

Sources: nat.plugins.langchain.control_flow.router_agent.prompt — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Prompt Configuration for the NVIDIA NeMo Guardrails Library | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · nat.builder.intermediate_step_manager — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · About Reasoning Agent — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Integrate knowledge graphs to enable relational reasoning.

Start here

THINK FIRST

How could you connect an agent to an external knowledge store so it can reason over related facts?

Why it matters: Questions describe a knowledge source for an agent and ask how to connect it or which agent should reason over it.

This objective is about connecting knowledge so agents can reason over relations. In NeMo Guardrails, documents feed bot responses through RAG, and a knowledge base can be set up three ways. A reasoning agent then plans ahead before acting.

Key terms

Term	Meaning
RAG	Retrieval-Augmented Generation, which grounds bot responses in a set of documents.
kb folder	The simplest knowledge base option, for static collections of Markdown documents.
EmbeddingSearchProvider	A custom embedding search provider you implement for advanced knowledge base use cases.
Similarity search	Comparing the query's embedding with indexed document chunks to find relevant ones.

Three ways to connect knowledge

NeMo Guardrails offers three knowledge base options. You can use the kb folder, a custom `retrieve_relevant_chunks` action or a custom `EmbeddingSearchProvider`. Each one feeds documents into bot responses through RAG.

Knowledge base options compared

Feature	kb folder	retrieve_relevant_chunks	EmbeddingSearchProvider
What it is	Markdown files placed in a folder	A custom action	A custom embedding search provider
Best fit	Static document collections	Chunks from external sources	Advanced use cases

EXAM TRAP

The kb folder does not accept every file type. It supports Markdown documents only.

How retrieval finds facts

Retrieval embeds the query and runs a similarity search. The query embedding is compared with the indexed document chunks to find the closest matches. You can also pass context directly in a generate call, using a message with the `context` role.

EXAM TRAP

Retrieval is not the only way to give the bot context. You can pass it straight into a generate call with a `context` role message.

Reasoning over what you find

A reasoning agent plans ahead rather than between steps. Unlike a ReAct agent, it reasons up front, then invokes an underlying function directly while reasoning on top. It requires an LLM that supports reasoning.

EXAM TRAP

Not every LLM suits a reasoning agent. You must choose one that supports reasoning.

Key takeaways

- A NeMo Guardrails knowledge base uses the kb folder, a custom `retrieve_relevant_chunks` action or a custom `EmbeddingSearchProvider`.
- The kb folder is the simplest option and accepts only Markdown.
- A reasoning agent plans ahead and needs an LLM that supports reasoning.

Sources: NVIDIA-Certified Professional: Agentic AI Exam Study Guide (opens in a new tab) · Knowledge Base Configuration | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · About Reasoning Agent — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Ensure adaptability and scalability of the agent's architecture.

Start here

THINK FIRST

How do you build an agent platform that can take on new tools and more load without rewriting its core?

Why it matters: Questions describe a growing agent system and ask which plugin, packaging or deployment feature keeps it adaptable and scalable.

The toolkit stays adaptable because all its parts, its own and yours, are plugins that slot in without core changes. On the deployment side, NIM gives orchestrators health signals and a clean reschedule point, and evaluation jobs can mix GPU and CPU nodes.

Key terms

Term	Meaning
Entry point	A Python packaging hook the toolkit uses to find plugins in any installed distribution package.

Term	Meaning
Function group	A set of related tools from one external service, exposed as `instance_name__function_name`.
Readiness endpoint	A NIM health probe that tells an orchestrator the container can serve inference, not just that it runs.
Model profile	The NIM configuration chosen to match the GPU hardware found at startup.

Everything is a plugin

First-party and external plugins are treated exactly alike. The toolkit builds its own components with the same plugin system it offers you. That keeps the library modular and extendable by default.

How plugins are found

Plugins are discovered through Python entry points. The toolkit scans the `nat.plugins` group and still loads `nat.components` for older packages. Every plugin decorator is an async context manager, so a plugin can run setup and teardown code.

EXAM TRAP

New external plugin packages should register under `nat.plugins`. The `nat.components` group is loaded only for backward compatibility.

Grouping related tools

Use a function group when one service offers several related tools. It is the preferred pattern for that case. Each tool is exposed under the group's instance name, joined to the function name by a double underscore.

Stable surfaces and telemetry

Breaking changes to stable `nat.plugin_api` symbols need a major release. External packages can rely on their documented behavior across minor and patch releases. A decorator's `wrapper_type` names the compatible LLM framework and adds telemetry hooks for observability, evaluation and profiling.

NIM in an orchestrator

NIM gives orchestrators clear health signals and a clean reschedule point. Separate probes show when a container is ready, and fail-fast supervision lets the orchestrator reschedule it. The table also shows NIM's default ports and how it picks a model profile.

NIM deployment features that help it scale

Feature	What it does	Key detail
Health probes	Separate liveness and readiness endpoints	Tells a running container from a ready one

Feature	What it does	Key detail
Fail-fast supervision	Shuts the container down if the engine or proxy exits	Lets the orchestrator reschedule it
Two-port design	Proxy handles inference and management traffic; the backend listens on its own port	Ports **8000** and **8001**
Profile selection	Picks a model profile that matches the detected GPUs	Override with NIM_MODEL_PROFILE

EXAM TRAP

A container that is running is not always ready. Use the readiness endpoint, not liveness, to decide when to send inference traffic.

Scaling evaluation

SLURM `node_pools` let one job mix GPU and CPU nodes. For example, GPU nodes can serve the model while CPU nodes run sandboxes. Adding an executor such as Kubernetes takes just a new class, a metadata convention and a registry entry. In a suite, a failing benchmark is logged and skipped.

Key takeaways

- Plugins are found through Python entry points; new packages should use the `nat.plugins` group.
- Function groups are the preferred way to expose several related tools from one service.
- NIM's separate liveness and readiness endpoints tell a running container apart from a ready one.
- By default, the NIM proxy listens on port **8000** and the backend on **8001**.

Sources: Plugin System in NVIDIA NeMo Agent Toolkit — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Architecture — NVIDIA NIM for LLM and VLM (opens in a new tab) · Architecture | NVIDIA NeMo Evaluator (opens in a new tab) · NVIDIA NeMo Agent Toolkit — Public Plugin API — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Domain 2 | 15% of the exam | 18 min read

Agent Development

Overview

Every point below comes from NVIDIA's official documentation; each section lists its sources.

Engineer prompts and dynamic prompt chains for reliable performance.

Start here

THINK FIRST

How do you control which prompt each LLM task gets, what goes into it, and which version runs?

Why it matters: Questions describe a prompt setup and ask which file, attribute, variable or mode controls what the LLM receives.

In NeMo Guardrails every LLM call does one specific task, and each task has its own prompt. You customize those prompts in YAML, fill them with Jinja2 variables at render time, and keep several versions with modes.

Key terms

Term	Meaning
Task prompt	The prompt sent for one LLM task; each LLM call performs exactly one specific task.
Jinja2	The template language prompts use to substitute variables when they are rendered.
prompting_mode	Top-level setting that a prompt's `mode` must match; with no match, the standard prompt is used.
Prompt context variable	A custom template variable you register with `LLMRails.register_prompt_context()`.
GA prompt optimizer	A NeMo Agent Toolkit tool that evolves prompts with a genetic algorithm.

One task per LLM call

Every LLM call performs one specific task. The `prompts` key lets you customize the prompts used for these various LLM tasks. Inside an action, you can render a custom task prompt with `LLMTaskManager`.

Where prompts live

Define prompts under the `prompts` key. That key can sit in the main `config.yml` or in a separate `prompts.yml` file. To load prompts from outside the config, set the `PROMPTS_DIR` environment variable to a directory of `.yml` prompt files.

EXAM TRAP

A `PROMPTS_DIR` folder is not read for any file type: it must contain `.yml` files with prompt definitions.

Prompt attributes

A prompt uses `content` or `messages`, never both. The two attributes are mutually exclusive. The `max_length` attribute caps the prompt length in characters and defaults to **16000**.

Three prompt attributes compared

Attribute	What it sets	Rule to remember
`content`	The prompt as one string	Cannot be combined with `messages`
`max_length`	Longest allowed prompt, in characters	Defaults to **16000**
`mode`	Which version of a task's prompt this is	Must match the top-level `prompting_mode`

EXAM TRAP

Setting both `content` and `messages` on one prompt is not a way to add context: the two attributes are mutually exclusive.

Dynamic values with Jinja2

Templates fill in variables with Jinja2. Register your own variable with `LLMRails.register_prompt_context()`. If you pass a function, its value is computed again for each rendering. Filters such as `first_turns(n)` keep only the first n turns of a Colang history.

EXAM TRAP

A function registered as prompt context is not evaluated once and cached; it runs again every time the prompt is rendered.

Prompt modes

Modes give one task and model several prompt versions. For example, you can keep a compact prompt for lower latency next to the standard one. The prompt's `mode` must match the top-level `prompting_mode`. If no prompt matches, the standard prompt is used.

EXAM TRAP

A `prompting_mode` with no matching prompt does not stop the app; the standard prompt is used instead.

Parsing output and token limits

The `user_intent` parser strips a prefix; `max_tokens` caps the output. The `user_intent` output parser removes a "User intent:" prefix when present. Self-check and content safety actions fall back to **1024** `max_tokens` if unset. Reasoning models need it set explicitly so the reasoning trace and the verdict both fit.

EXAM TRAP

Leaving `max_tokens` unset for a reasoning model is risky: the **1024** fallback may not fit both its reasoning and its verdict.

Optimizing prompts automatically

The GA optimizer evolves prompts with a genetic algorithm. It belongs to NeMo Agent Toolkit. When it finishes, it writes the final `optimized_prompts.json` and `ga_history_prompts.csv` files.

Key takeaways

- Each LLM call performs one task, so you customize prompts task by task.
- Prompts go under `prompts` in `config.yml` or `prompts.yml`, or in `.yml` files under `PROMPTS_DIR`.
- `content` and `messages` are mutually exclusive, and `max_length` defaults to **16000** characters.
- With no prompt matching `prompting_mode`, the standard prompt is used.

Sources: Prompt Configuration for the NVIDIA NeMo Guardrails Library | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · nat.plugins.config_optimizer.prompts.ga_prompt_optimizer — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Integrate generative and multimodal models (text, vision, audio).

Start here

THINK FIRST

How would you send text, images, audio or video to a model served by NIM, and what decides which inputs work?

Why it matters: Questions describe a multimodal request or a NIM setup and ask which endpoint, content part or container image fits.

NIM serves multimodal models through OpenAI-compatible Chat Completions requests whose content mixes text and media. What works depends on the model and the engine: a text-only model cannot take media, and video needs vLLM.

Key terms

Term	Meaning
Content array	The list in a Chat Completions request that mixes text and media items.
audio_url	Content part for audio input, an extension to the Chat Completions schema both engines accept.
Profile prefix	The vllm- or sglang- start of a profile name, naming the one engine an image embeds.
Data URL	An inline image as base64 in image_url.url, such as ..., with a matching MIME type.

One request format

Send media in an OpenAI-compatible Chat Completions request. The message content array mixes text and media items. `POST /v1/chat/completions` keeps multi-turn history and supports streaming and tool calling.

Passing images and audio

Inline images go in `image_url.url` as a base64 data URL. Use the MIME type of the file, for example `image/png` for a PNG. Audio uses the `audio_url` part, and both engines also accept OpenAI's `input_audio` part. Media URLs must be reachable from the NIM container.

EXAM TRAP

A media URL that only your own machine can reach will not work. The NIM container itself must be able to reach it.

Engines and modalities

Video needs a vLLM image; image and audio work on both. Each NIM image embeds one engine, vLLM or SGLang, shown by the `vllm-` or `sglang-` profile prefix. SGLang images lack the video decoder that `video_url` requests need.

Input support by NIM engine

Feature	vLLM image	SGLang image
Image input	Yes	Yes
Audio input	Yes	Yes
Video input (video_url)	Yes	No, it lacks the video decoder

EXAM TRAP

An SGLang image cannot serve `video_url` requests because it has no video decoder. Use a vLLM image for video.

The model must match

A text-only model cannot take image, audio or video input. Serving it through NIM does not change that. The NIM Support Matrix groups models into LLMs, VLMs and Model-Free NIM.

EXAM TRAP

NIM does not give a text-only model the ability to read images or audio. The model itself must handle that input.

Anthropic-style clients

NIM also offers an Anthropic-compatible `/v1/messages` endpoint. Requests pass through the nginx proxy with bodies left unchanged. NIM does not check `x-api-key`, so any non-empty key satisfies Anthropic SDK clients.

OCI provider

The toolkit's `oci` provider uses OCI SDK authentication. Its region defaults to `us-chicago-1`. The current release does not enable the OpenAI Responses API for it.

Key takeaways

- Multimodal requests use OpenAI-compatible Chat Completions with a mixed content array.
- Video input needs a vLLM image; image and audio work on both vLLM and SGLang.
- A text-only model cannot process image, audio or video, even on NIM.
- Media URLs must be reachable from the NIM container.

Sources: Image, Audio, and Video Input — NVIDIA NIM for LLM and VLM (opens in a new tab) · Support Matrix for NIMs — NVIDIA NIM for LLM and VLM (opens in a new tab) · API Reference — NVIDIA NIM for LLM and VLM (opens in a new tab) · NVIDIA NeMo Agent Toolkit OCI Integration — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Build and connect custom tools, APIs, and functions for external system interaction.

Start here

THINK FIRST

How do you turn your own code into a tool an agent can call, and share it with other agents or clients?

Why it matters: Questions describe a custom tool or API call and ask how to register, configure, invoke or

expose it.

In NeMo Agent Toolkit, tools and agents are both functions: reusable pieces that call APIs, search or calculate. You register a function so configuration files can use it, and you can publish it to MCP clients.

Key terms

Term	Meaning
Function	A reusable component that does one job, such as a web search, an API call or a calculation.
FunctionBaseConfig	The base class that every function's configuration object inherits from.
Function group	Related functions that share one configuration object and runtime context.
FunctionRef	A `str`-like annotation marking a configuration field as a reference to another function.
MCP server worker	The default worker that publishes workflow functions as MCP tools.

Functions are the building block

Tools and agents are both functions. So one agent can serve as a tool for another, which is how multi-agent workflows are built. A function can run a web search, call an API or do a calculation.

Registering a function

Register with `@nat.plugin_api.register_function`. A configuration file can use a function only after it is registered. Its configuration object inherits from `FunctionBaseConfig` and needs a `name` attribute, which identifies the function in the configuration file.

EXAM TRAP

Writing the function in Python is not enough for configuration files. It must be registered with the decorator first.

Inputs and outputs

A function takes exactly one input argument. Input and output types can be anything. Their schemas are Pydantic BaseModel classes that validate data at runtime. Pass a multi-argument callable to `from_fn()` and it is wrapped in a single-argument lambda.

EXAM TRAP

`from_fn()` does not reject a multi-argument callable. It wraps it in a lambda that takes one argument and passes it on.

Invoking a function

Every function call is asynchronous. Use `ainvoke()` for a single output and `astream()` for streaming output. Picking the wrong method for the function's output type results in an error.

The two ways to invoke a function

Method	Use for	Errors when
`ainvoke()`	A single output	The function only has a streaming output
`astream()`	Streaming output	The function only has a single output

EXAM TRAP

`ainvoke()` is not a universal fallback. Calling it on a streaming-only function results in an error.

Function groups

Groups share one configuration and runtime context. That lets related functions reuse connections, caches and other resources. Each function gets a full name with a double underscore, such as `math__add`. Pick functions with `include` or `exclude`, never both.

EXAM TRAP

`include` and `exclude` cannot both be set on one group. They are mutually exclusive, so use one or the other.

Referencing other functions

Annotate function names with `FunctionRef`. It behaves like `str` but marks the value as a reference to another function. That lets the toolkit make sure the referenced function is registered first.

Exposing tools over MCP

The default MCP server worker publishes functions as tools. Your workflow functions then appear as MCP tools. For custom behavior, inherit from `MCPFrontEndPluginWorker` and put the worker's fully qualified class name in `runner_class`.

Key takeaways

- Agents and tools are both functions, so an agent can be another agent's tool.
- Register functions with `@nat.plugin_api.register_function` so configuration files can use them.
- A function takes **one** input argument, and every call is asynchronous.
- Grouped functions are named `group__function`; `include` and `exclude` are mutually exclusive.

Sources: Functions — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Writing Custom Functions — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Adding a Custom MCP Server Worker — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Function Groups — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Implement error handling (retry logic, graceful failure recovery).

Start here

THINK FIRST

How should an agent decide when a failed outbound call is worth retrying, and how long to wait before trying again?

Why it matters: Questions describe a failing call and ask which retry setting, default or error type explains the behavior.

Retries are governed by a policy, not by guesswork. The policy sets how many attempts, how long to wait, and which methods are safe to repeat. Errors that survive the retries reach the caller with a record of what was tried.

Key terms

Term	Meaning
RetryPolicy	The NeMo Guardrails settings for attempts, delays, retryable methods and Retry-After handling.
Exponential backoff	The fallback wait strategy when a server's Retry-After value cannot be used.
RetryMixin	A NeMo Agent Toolkit Pydantic mixin that adds retry settings to a component's config.
retry_count	How many retries completed before an error or response came back to the caller.
Clamping	A RetryPolicy option, off unless explicitly enabled, that lifts the rule that Retry-After counts only within `max_retry_after`.

Counting attempts

`max_attempts` **counts the first request too.** A value that looks like the number of retries is really the total number of tries. The first call uses up one of them.

EXAM TRAP

Reading `max_attempts` as retries on top of the first call overstates how often the request is sent.

Default delays

Waits start at 0.5 seconds and are capped at 8.0 seconds. These are the RetryPolicy defaults for `initial_delay` and `max_delay`.

Which methods are retried

POST is left out of the safe default set on purpose. Method names are also normalized to uppercase.

EXAM TRAP

Do not assume every failed request is retried by default. POST is deliberately excluded from the safe default methods.

Honoring server hints

A Retry-After value counts only inside `max_retry_after`**, unless clamping is on.** An invalid or out-of-policy value is dropped, and the client falls back to exponential backoff. Vendor override headers count only when you turn on `honor_retry_override_header`.

How RetryPolicy treats server wait hints

Hint	Default handling	Changed by
Retry-After within max_retry_after	Honored	Not needed
Retry-After outside max_retry_after	Not honored	Explicitly enabling clamping
Vendor retry-override header	Ignored	Enabling honor_retry_override_header

Errors after retries

`HTTPClientError` **is the base class for outbound HTTP failures.** `HTTPConnectionError` means a connection could not be opened or kept. `HTTPTimeoutError` means the request went past its total timeout. Each error carries `retry_count`, so you can see how hard the client tried.

Retry settings in the toolkit

RetryMixin adds retry fields to a NeMo Agent Toolkit config. Use `do_auto_retry` to switch automatic retries on or off. List the HTTP status codes that should trigger a retry in `retry_on_status_codes`.

Seeing retries on success

Successful responses also report `retry_count`**.** RetryingHTTPClient puts the count in the response extensions. Transport errors expose their completed count before they are raised again.

Key takeaways

- `max_attempts` includes the initial request, so it is a total, not a retry count.
- RetryPolicy defaults: `initial_delay` of **0.5** seconds and `max_delay` of **8.0** seconds.
- POST is not retried by default because it is missing from the safe method set.
- An unusable Retry-After value means the client uses exponential backoff instead.

Sources: nemoguardrails.http.retry | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · nemoguardrails.http.errors | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · nat.data_models.retry_mixin — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Develop dynamic conversation flows with real-time streaming and feedback mechanisms.

Start here

THINK FIRST

When you stream an agent's answer, how do you keep it fast without letting unsafe text slip through?

Why it matters: Questions describe a streaming setup and ask which output rail setting balances speed against safety.

NeMo Guardrails streams with stream_async() and checks output in chunks of tokens. The chunk settings and stream_first decide how soon users see text, and whether they can see text a rail later blocks.

Key terms

Term	Meaning
stream_async()	The NeMo Guardrails method that streams responses; basic streaming needs no configuration.
chunk_size	Tokens per chunk that output rails process; the default is **200**.
context_size	Tokens carried from one chunk into the next for continuity; the default is **50**.
stream_first	Whether the client gets each chunk before output rails check it; the default is True.

Streaming out of the box

Basic streaming works with `stream_async()` alone. No configuration is needed. The old top-level `streaming: True` field is deprecated; call `stream_async()` directly instead.

Turn on output rail streaming

Output rails need `rails.output.streaming.enabled` set to True. The setting defaults to False. With output rails configured, enable it explicitly so they process tokens in chunked mode.

EXAM TRAP

Calling `stream_async()` alone is not enough once output rails are configured. Until `enabled` is True, you cannot use `stream_async()` with output rails.

How chunks are checked

Tokens are buffered until `chunk_size` is reached. The last `context_size` tokens of each chunk are prepended to the next chunk's context. Rails can then judge a sentence that spans two chunks as a whole.

Output rail streaming settings and their defaults

Setting	Default	What it controls
enabled	False	Needs True before output rails can stream
chunk_size	200	Tokens in each chunk the output rails process
context_size	50	Tokens carried over from one chunk to the next
stream_first	True	Client gets tokens before rails check the chunk

Choosing chunk_size

Smaller chunks reach users sooner; larger ones run rails less. Choose by what matters more for your users: seeing text early, or fewer rail executions per response.

Smaller and larger chunk_size compared

Feature	Smaller chunk_size	Larger chunk_size
Rail executions	More	Fewer
Time to first output	Faster	Slower

Stream first or check first

`stream_first` **decides who sees a chunk first.** With the default True, users get text sooner. With False, rails check every chunk before the client sees it.

stream_first set to True and to False

Feature	True (default)	False
Time-to-first-token	Faster	Higher by the rail time per chunk
Blocked content	User may already have received it	User never sees it

EXAM TRAP

With `stream_first` True, a block does not take text back. The user already has those tokens when the stream ends with a JSON error.

When a rail blocks

A block ends the stream with a JSON error object. If any output rail blocks content, the stream yields a JSON object with an `error` key and stops at once.

Key takeaways

- Call `stream_async()` directly; the deprecated top-level `streaming: True` field is not needed.
- Set `rails.output.streaming.enabled` to True to stream with output rails.
- `chunk_size` defaults to **200** tokens and `context_size` to **50**.
- `stream_first` True is faster, but users may see text a rail later blocks.

Sources: Streaming Configuration | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · Output Rail Streaming Configuration | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab)

Evaluate and refine agent decision-making strategies.

Start here

THINK FIRST

How would you judge whether an agent took a sensible path to its answer, and then improve the way it plans?

Why it matters: Questions describe an agent evaluation and ask which evaluator, judge setting or planner refinement fits it.

You evaluate an agent by running its workflow on curated test data and scoring the results. The trajectory evaluator judges the path the agent took. Repeated runs and plan refinement then help you improve its decisions.

Key terms

Term	Meaning
Trajectory	The path an agent takes, read from the intermediate steps its workflow generated.
Judge LLM	A model that scores the workflow's trajectory or output, taking the available tools into account.
Intermediate steps	Steps the workflow records; workflow_output.json keeps LLM_END and TOOL_END by default.
Tunable RAG Evaluator	An evaluator that scores answers against an expected answer description, not a ground truth.

Scoring the path

The trajectory evaluator judges the agent's path. It reads the intermediate steps the workflow generated. A judge LLM scores that path while considering the tools available during execution. The evaluator builds on LangChain/LangGraph agent trajectory evaluation.

EXAM TRAP

The trajectory evaluator does not look only at the final answer. It scores the path, using the workflow's intermediate steps.

Reading the score

Trajectory scores run from 0 to 1. A score of **1.0** means a perfect trajectory. Give the trajectory judge LLM a `max_tokens` of **1024** so it has enough context to evaluate.

Ragas versus Tunable RAG

Ragas prompts are fixed; Tunable RAG uses an answer description. The Ragas metrics' judge prompt cannot be changed, and their judge LLM should use **8** tokens. The Tunable RAG Evaluator scores against an expected answer description instead of a ground-truth answer.

EXAM TRAP

You cannot edit the Ragas judge prompt. If you need your own prompt, switch to the Tunable RAG Evaluator or a custom evaluator.

LLM-as-judge requirements

`langsmith_judge` **needs a judge with structured output.** Its judge LLM must support JSON schema mode. Check that your chosen judge model offers it before you run the evaluation.

Repeating runs

`--reps` **repeats an evaluation several times.** Repeated runs let you average the score and see how much outputs vary. By default, `workflow_output.json` keeps only `LLM_END` and `TOOL_END` intermediate steps.

EXAM TRAP

By default, `workflow_output.json` does not keep every intermediate step. It keeps only `LLM_END` and `TOOL_END`.

Refining the plan

IterativePlanRefinementEditor reworks a plan with the same LLM. This experimental planner writes an initial plan, then improves it several times. Each planning item is refined in parallel, **3** times by default.

Key takeaways

- The trajectory evaluator scores the agent's path from its intermediate steps, from **0** to **1**.
- Set the trajectory judge LLM's `max_tokens` to **1024**.
- Ragas judge prompts are not configurable; use the Tunable RAG Evaluator or a custom evaluator instead.
- `--reps` lets you average scores across runs and measure variation.

Sources: Agent Evaluation in NVIDIA NeMo Agent Toolkit — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.experimental.test_time_compute.editing.iterative_plan_refinement_editor — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Domain 3 | 13% of the exam | 16 min read

Evaluation and Tuning

Overview

Every point below comes from NVIDIA's official documentation; each section lists its sources.

Implement evaluation pipelines and task benchmarks to measure performance.

Start here

THINK FIRST

What makes an agent benchmark score trustworthy enough to compare two versions of the same agent?

Why it matters: Questions describe an evaluation setup and ask which scorer, statistic or tracing step gives a reliable result.

An evaluation pipeline pairs each benchmark with a scorer, then turns many scored samples into statistics you can trust. After an evaluation, LangSmith tracing can link runs to dataset examples and attach the evaluator scores as feedback.

Key terms

Term	Meaning
Benchmark	A task set in NeMo Evaluator, defined with decorators such as @benchmark plus @scorer, or @register.
Scorer	Set for each benchmark with the @scorer decorator; the scoring/ package also offers reusable scorers.
LLM-as-judge	Scoring where another model grades the answer in a post-processing step.
Experiment	The LangSmith experiment project an evaluation run's OTEL traces land in; each run gets its own.

Benchmarks and where they come from

NeMo Evaluator ships 17 built-in benchmarks. More environment types can be pulled in through URI schemes and namespace prefixes, such as `lm-eval://` and `gym://`. Each benchmark brings its own scorer, set with the `@scorer` decorator.

Choosing a scorer

Each reusable scorer checks answers in its own way. The table shows what each one checks. `code_sandbox` pulls code out of markdown fences before running it, and `needs_judge` hands the sample to a judge.

Reusable scorers compared

Scorer	What it checks	Typical use
exact_match	Normalized strings: lowercase, no punctuation or extra spaces, articles collapsed	Text answers compared after normalization
numeric_match	The last number found in the response	Responses scored on the last number they contain
code_sandbox	Runs extracted code inside an isolated Docker container with memory limits and timeouts	HumanEval code tasks
needs_judge	Marks the sample for an LLM judge to score later	Samples that require LLM-as-judge scoring

EXAM TRAP

A `needs_judge` sample first comes back with `correct` set to False. That is a placeholder; the judge post-processor in `scoring/judge.py` gives the real score.

Removing bias in the tasks

GPQA Diamond shuffles answer choices with a seeded RNG. Shuffling stops a model from scoring well just by favoring one position.

Turning scores into statistics

Confidence intervals are 95%, from 10,000 bootstrap resamples. With repeats, rewards are averaged per problem before analysis. When problems carry category metadata, per-category accuracy is computed for you.

Gating on results

`nel gate` **finds result bundles named** `eval-.json` **in a folder.** Name your bundles to match that pattern, or the gate step will not pick them up.

Why linking needs retries

OTEL traces arrive in asynchronous batches, so matching retries. Runs are matched to items when one input text contains the other. LangSmith can also return **200** OK before a run is indexed, silently losing the link. Eager linking avoids that delay by mapping span IDs straight to run IDs.

EXAM TRAP

If runs cannot be queried, every item is treated as unlinked, so feedback can still be recorded at the experiment level.

Key takeaways

- Each NeMo Evaluator benchmark defines its own scorer with the `@scorer` decorator.
- Reported confidence intervals are **95%**, built from **10,000** bootstrap iterations.
- Judge-scored samples get their final score later, from the post-processing judge in `scoring/judge.py`.
- LangSmithEvaluationCallback links runs to dataset examples after the evaluation and attaches scores as feedback.

Sources: Scoring | NVIDIA NeMo Evaluator (opens in a new tab) · nat.plugins.langchain.langsmith.langsmith_evaluation_callback — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Built-in Benchmarks | NVIDIA NeMo Evaluator (opens in a new tab) · Evaluation Result Format | NVIDIA NeMo Evaluator (opens in a new tab)

Compare agent performance across tasks and datasets.

Start here

THINK FIRST

How do you tell whether one agent run truly beats another, rather than winning by chance?

Why it matters: Questions describe two evaluation runs and ask whether the comparison is fair, what a verdict means or which setting reduces noise.

A fair comparison holds everything constant except the change you are testing. Repeated runs and paired statistics then separate a real difference from sampling noise.

Key terms

Term	Meaning
Paired test	A test that compares two runs problem by problem; nel compare treats a comparison this way.
Discordant pair	A problem on which the two models disagree; McNemar's test looks only at these.
reps_per_param_set	Optimizer setting for how many times the workflow runs per parameter set; the default is 3.
Pareto optimal	A trial where improving one metric would make another metric worse.

Match the runs first

Compare only runs that were set up the same way. Both runs need the same benchmark, prompt template, dataset slice, max problems and repeat count. Otherwise a difference may come from the setup instead of the agent.

EXAM TRAP

Small benchmarks are noisy. HumanEval has **164** items and can move several percentage points between runs from sampling noise alone.

Paired statistics

`nel compare` **runs a paired statistical test.** It applies McNemar's exact binomial test to the discordant pairs, asking whether the regression count is bigger than chance would explain. The default practical threshold, `--max-drop`, is a **5%** drop, written **0.05** on a 0-1 scale.

Read the verdict

INCONCLUSIVE means too little paired data. Re-run with more problems or switch to a larger benchmark. For CI, `--strict` (off by default) makes the command exit non-zero on BLOCK, WARN or INCONCLUSIVE. By default it saves `regression_report.md` inside the candidate's result folder.

EXAM TRAP

`nel compare` diagnoses; it is not the release gate. `nel gate` is the tool that enforces policy across benchmarks.

Repeat to tame noise

Repeat runs to average scores and see variation. Use **3-5** reps for deterministic workflows and **10-20** for highly stochastic outputs. If metrics still vary a lot, raise `reps_per_param_set` and keep evaluation conditions consistent.

The two repetition settings compared

Feature	nat eval --reps	reps_per_param_set
Used by	nat eval	The optimizer, for each parameter set
Default	1	3

Choose the dataset

`--dataset` **swaps in a different JSON dataset.** The file holds questions with ground truth answers and overrides the dataset path in the config file. With `--skip_workflow`, the workflow does not run; the dataset is scored as given and must hold the `generated_` columns.

Keep runs apart

Give every run its own output folder. Set `append_job_id_to_output_dir` to true to add a `job_{UUID}` folder per run, so earlier results survive. In the Weave dashboard, keep one project name and tell runs apart with `workflow_alias`.

Compare in Weave

Select the runs, then click Compare. Pick the runs in the Weave dashboard and press the Compare button. Plots of metrics for single dataset entries appear only when exactly **2** runs are compared.

EXAM TRAP

Per-entry metric plots do not appear when you compare three or more runs in Weave. They work only across two runs.

Weigh several metrics

The optimizer folds many metrics into one score. `multi_objective_combination_mode` supports harmonic (the default), sum and chebyshev. Sum allows compensation, so a strong metric can offset a weak one. Each metric's weight defaults to **1.0** and is normalized.

EXAM TRAP

Do not expect a 2D Pareto front plot for three metrics. `pareto_front_2d.png` appears only when exactly **2** metrics are optimized.

Key takeaways

- Match benchmark, slice, template, repeats and max problems before comparing runs.
- `nel compare` applies McNemar's exact binomial test to discordant pairs.
- INCONCLUSIVE means re-run with more problems or a larger benchmark.
- Repeat noisy evaluations: `--reps` defaults to **1** and `reps_per_param_set` to **3**.

Sources: Comparing Evaluation Runs | NVIDIA NeMo Evaluator (opens in a new tab) · Agent Evaluation in NVIDIA NeMo Agent Toolkit — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · NVIDIA NeMo Agent Toolkit Optimizer — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Command Line Interface — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Collect and integrate structured user feedback for iterative improvements.

Start here

THINK FIRST

How do you turn user reactions and evaluation failures into concrete improvements to an agent?

Why it matters: Questions describe a feedback need and ask which endpoint, mode or pipeline captures it and puts it to use.

Feedback comes from two places: users rating responses and evaluations finding failures. NeMo Agent Toolkit links user ratings to traces, feeds failure reasoning back into prompt optimization, and exports traces to improve models.

Key terms

Term	Meaning
/feedback endpoint	A FastAPI endpoint for thumbs-up or thumbs-down ratings that are linked to Weave traces.
Oracle feedback	Failure reasoning from evaluations, injected into the prompt optimization genetic algorithm.
Data Flywheel Blueprint	NVIDIA blueprint that uses exported traces to fine-tune and evaluate smaller replacement models.
workload_id	Data Flywheel label on a trace, mapped by default to the parent registered function.

Collect ratings from users

Weave plus FastAPI can expose a `/feedback` endpoint. Users give thumbs-up or thumbs-down on agent responses, and each rating is linked to a trace in your Weave project. The `WeaveFastAPIPluginWorker` registers the endpoint when Weave telemetry is configured.

Who gave the feedback

Weave attributes feedback by name, then email, then ID. It takes the first available of `trace_user_name`, `trace_user_email` and `trace_user_id`, falling back to anonymous. You opt in to attribution by setting these `trace_` fields explicitly.

> **EXAM TRAP**
> Setting a general user name on the context metadata does not attribute Weave feedback. Only the explicit `trace_` fields are used.

Stable user identity

Each credential maps to a stable UUID v5 `user_id`. The same credential gives the same `user_id` over HTTP or WebSocket. With HTTP Basic Auth the password is part of the key. A new password gives a new `user_id`, and prior workflow state becomes unreachable.

> **EXAM TRAP**
> Identity resolution is not authentication. It only maps identities, and JWTs are decoded without signature verification.

Feedback from evaluation failures

Oracle feedback learns from specific evaluation failures. It puts failure reasoning into the prompt optimization genetic algorithm. `extract_worst_reasoning` takes reasoning from the worst-performing items, and `build_oracle_feedback` returns None when no reasoning exists.

Choosing a feedback mode

Oracle feedback has 4 modes. The modes are `never`, `always`, `failing_only` and `adaptive`. In `failing_only`, `fitness_threshold` decides whether feedback is injected. `adaptive` can trigger after `stagnation_generations` without improvement.

Scores and prompts in LangSmith

LangSmith adds evaluation metrics to logging and tracing. With `nat eval`, each example's run links to its dataset example, with evaluator scores attached as feedback. In prompt optimization, each trial's prompt gets a commit tag, and the best trial's prompt is tagged `best`.

Close the loop with Data Flywheel

Exported traces can train smaller, faster models. The Data Flywheel Blueprint fine-tunes and evaluates smaller models from the traces to cut latency. It picks training data by `client_id` plus `workload_id`. Use `@track_unregistered_function` for custom workload scopes.

Ask during a run

`UserInteractionManager` **asks the user at runtime.** Its `prompt_user_input` method asks a question and waits, delegating to the callback in `ContextState.user_input_callback`. SessionManager typically sets that callback.

Key takeaways

- The Weave `/feedback` endpoint links thumbs-up or thumbs-down ratings to traces.
- Weave attributes feedback by `trace_user_name`, then email, then ID, else anonymous.
- Oracle feedback injects failure reasoning into the prompt optimization genetic algorithm.
- Data Flywheel uses traces to fine-tune smaller models that reduce latency.

Sources: Observe Workflows — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.plugins.config_optimizer.prompts.oracle_feedback — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · User Identity Resolution — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.builder.user_interaction_manager — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Tune model parameters (e.g., accuracy, latency-efficiency trade-offs).

Start here

THINK FIRST

How do you trade a little latency for throughput, or memory for speed, when tuning a served model?

Why it matters: Questions describe a deployment goal and ask which NIM setting, override or benchmark result reaches it.

A NIM starts from a profile, and several setting sources can override it in a fixed order. A concurrency sweep shows the throughput and latency trade-off, and acceptable accuracy comes before any cost figure.

Key terms

Term	Meaning
Profile	A NIM configuration whose description may name a GPU, a throughput or latency variant, or a minimum memory.
Tensor parallelism	Splitting model layers across several GPUs; set with NIM_TENSOR_PARALLEL_SIZE.
Speculative decoding	NGRAM, MTP or EAGLE3 decoding, switched on or off at runtime rather than chosen by profile.
Pareto curve	The throughput and latency pairs that a concurrency sweep produces.

Accuracy before cost

Measure cost only once accuracy is acceptable. The application's use case defines what acceptable means. Then benchmark the model itself, its throughput, latency and token-level metrics, rather than simulating traffic.

Who wins an override

Direct container arguments beat every other setting. Once every override is applied, NIM fetches the profile for the final resolved setup. Downloaded model files then always fit the launch configuration.

Settings that override a NIM profile, highest priority first

Rank	Setting	Example
1	Direct container arguments	--tensor-parallel-size 4
2	NIM_PASSTHROUGH_ARGS	--block-size 32 --max-num-seqs 16
3	NIM environment variables mapped to backend arguments	NIM_TENSOR_PARALLEL_SIZE=4
4	runtime_config.json	A profile's spec config

EXAM TRAP

A profile's `tp=2` does not win over your setting. With `NIM_TENSOR_PARALLEL_SIZE=4` set, TP=4 is what the backend uses.

Passing extra backend arguments

`NIM_PASSTHROUGH_ARGS` **carries backend CLI flags.** If the image already supplies a flag such as `--block-size 32`, keep it and append yours. For example, set the variable to `--block-size 32 --max-num-seqs 16`.

EXAM TRAP

Do not replace the image's `--block-size 32` with your new flag alone. Keep the image value and append your own after it.

Memory, context and speed

The profile selector estimates GPU memory needs. It works from model weights, KV cache, activations and overhead. The table lists variables that override parallelism, the context window, CUDA graphs and warmup.

NIM tuning variables and what to watch for

Variable	Effect	Watch for
NIM_TENSOR_PARALLEL_SIZE	Splits layers across that many GPUs	It overrides the profile's own TP degree
NIM_MAX_MODEL_LEN	Overrides the context window	Values above the trained maximum may cause errors
NIM_DISABLE_CUDA_GRAPH=true	Turns off CUDA graph optimization	May save GPU memory but lowers throughput
NIM_STARTUP_WARMUP	Sends one small chat request before ready	Spares the first client the kernel compilation cost

Speculative decoding

Speculative decoding is a runtime toggle, not a profile. Set `NIM_SPECDEC_ENABLE=0` to force it off for every profile. For its settings, CLI `--speculative-*` flags and `NIM_PASSTHROUGH_ARGS` beat `NIM_SPECDEC_ARGS`, which beats the profile's spec block.

Read the trade-off curve

Sweep concurrency to get a Pareto curve. Put output token throughput on one axis and inter-token latency on the other, then choose the concurrency with the highest throughput inside your latency budget. Record the engine and resolved arguments before you collect data.

EXAM TRAP

At the default log level you cannot tell whether an override took effect, because the `Resolved configuration:` line is suppressed.

Benchmark hygiene

Keep runs comparable and inputs honest. A synthetic workload and a replayed trace of real requests answer different questions, so never compare a synthetic point with a traced one. Use a fresh `cache_salt` per run. The completions endpoint has no system role, so a shared system prefix vanishes. An input near a tenth of the expected length means checking `--endpoint-type`.

Automate the search

`optimize_parameters` **tunes non-prompt hyperparameters.** It persists the best configuration it finds. `create_pareto_visualization` plots results from an Optuna Study, a file path or a pandas DataFrame.

Key takeaways

- Direct container arguments outrank `NIM_PASSTHROUGH_ARGS`, mapped variables and `runtime_config.json`.
- `NIM_DISABLE_CUDA_GRAPH=true` may save GPU memory but costs throughput.
- `NIM_SPECDEC_ENABLE=0` turns speculative decoding off for every profile.
- Pick the concurrency with the highest throughput inside your latency budget.

Sources: Model Profiles and Selection — NVIDIA NIM for LLM and VLM (opens in a new tab) · Benchmarking — NVIDIA NIM for LLM and VLM (opens in a new tab) · Environment Variables — NVIDIA NIM for LLM and VLM (opens in a new tab) · nat.plugins.config_optimizer.parameters.optimizer — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.plugins.config_optimizer.parameters.pareto_visualizer — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Analyze evaluation results to guide targeted optimization.

Start here

THINK FIRST

After an evaluation run, how do you tell whether a low score points to the answers, the output format or the model errors?

Why it matters: Questions show evaluation results and ask which breakdown, failure category or tool points to the right fix.

NeMo Evaluator can break evaluation results down in three ways. The ArtifactCollector sorts failures into categories, category metadata turns on per-category accuracy, and bootstrap resampling gives **95%** confidence intervals.

Key terms

Term	Meaning
Failure analysis	NeMo Evaluator's automatic sorting of failed samples into categories, written to failure_analysis.json.
pass@k	The standard Codex-style metric, computed from n attempts per problem of which c are correct.
Eval bundle	A results file read by `nel compare` and `nel gate`, named to match `eval-*.json`.
Bootstrap CI	A **95%** confidence interval built from **10,000** bootstrap resamples.

Read the failure categories first

The ArtifactCollector sorts failures into categories for you. The output goes to `failure_analysis.json`. Check which categories dominate before tuning anything.

Failure categories and what each one is detected from

Category	Detected when	Checked in
format_error	Response is not empty, but no answer was extracted	The model's response
refusal	Response contains phrases like "I cannot" or "I'm sorry"	The model's response
rate_limit	Model error contains "429" or "rate"	The model error

EXAM TRAP

A `rate_limit` failure is read from the model error, which contains **429** or "rate". It is not judged from the answer the model gave.

Break results down by category

Category metadata turns on per-category accuracy. Add a `metadata.category` label such as algebra or geometry to each result. Regression reports then show a per-category breakdown and clustered confidence intervals.

Account for uncertainty

Intervals are 95%, from 10,000 bootstrap iterations. With repeats, rewards are averaged per problem before analysis. For multiple attempts, pass@k uses n attempts per problem with c correct.

EXAM TRAP

Keep `problem_idx` the same for a problem in every run. If it changes, runs cannot be compared problem by problem.

Compare and gate runs

`nel compare` and `nel gate` read results.jsonl and eval bundles. `nel gate` finds bundles named `eval-*.json` in a folder. Each bundle's `benchmark.name` has to equal the name used in the gate policy YAML.

Find runtime bottlenecks

The toolkit profiler has modules for analyzing evaluation traces. Its `inference_optimization` package covers prompt caching, token uniqueness and workflow runtimes. Bottleneck analysis comes in nested stack and simple stack variants.

Track optimizer trials in LangSmith

Each optimizer trial gets its own experiment on a shared dataset. LangSmithOptimizationCallback creates that dataset before any trial runs. Without a set prompt format, it auto-detects one and falls back to "f-string" when it finds no curly-brace templating.

> **EXAM TRAP**
>
> Above **5,000** items per trial, the **600**-second retry window may run out. Some runs then look unlinked in the UI, although every trace was delivered.

Key takeaways

- Check `failure_analysis.json` first: it sorts failures into format errors, refusals and rate limits.
- Add category metadata to get per-category accuracy and clustered confidence intervals.
- NeMo Evaluator reports **95%** confidence intervals from **10,000** bootstrap iterations.
- For `nel gate`, name bundles `eval-*.json` and match `benchmark.name` to the policy YAML.

Sources: Scoring | NVIDIA NeMo Evaluator (opens in a new tab) · Evaluation Result Format | NVIDIA NeMo Evaluator (opens in a new tab) · nat.plugins.profiler.inference_optimization — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.plugins.profiler.inference_optimization.bottleneck_analysis — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.plugins.langchain.langsmith.langsmith_optimization_callback — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Domain 4 | 5% of the exam | 15 min read

Deployment and Scaling

Overview

Every point below comes from NVIDIA's official documentation; each section lists its sources.

Deploy and orchestrate multi-agent systems at production scale.

Start here

THINK FIRST

What does it take to run a model too large for one node, and keep that deployment healthy at production scale?

Why it matters: Questions describe a large-model deployment on Kubernetes and ask which parallelism, network, storage or operator setting it needs.

When one node's GPUs are not enough, NIM spreads a very large model across several nodes. Leader and worker pods form a Ray cluster, and vLLM or SGLang runs the model across it.

Key terms

Term	Meaning
Leader/worker model	Leader and worker pods run the same NIM image; a command injected at deployment sets each role.
LeaderWorkerSet	A Kubernetes-native CRD for managing leader/worker pod groups.
Tensor-parallel size	Commonly set to the number of GPUs per node in a multi-node deployment.
Pipeline-parallel size	Commonly set to the number of nodes in a multi-node deployment.
NIMService	The NIM Operator custom resource that automates multi-node deployment.

Deployment options

NIM runs on Kubernetes with Helm charts or the NIM Operator. Other orchestration platforms, such as KServe, OpenShift and Run:ai, work too. When a single node's GPU capacity is not enough for a very large model, such as DeepSeek R1, use multi-node deployment.

Leaders, workers and Ray

One NIM image serves both leader and worker pods. The command injected at deployment time sets each pod's role. Install the LeaderWorkerSet CRD to manage these pod groups. Ray forms the cluster, and vLLM or SGLang runs the model across it.

EXAM TRAP

The distributed backend is not a separate choice you make; vLLM or SGLang is selected automatically based on the NIM image.

Sizing tensor and pipeline parallelism

Commonly, TP matches GPUs per node and PP matches node count. The Helm chart needs both sizes greater than **0**, or `model.profile` set instead. With neither, the chart fails to render.

The most common multi-node parallelism settings

Setting	Common value	Two nodes with 8 GPUs each
Tensor-parallel size	GPUs per node	**8**
Pipeline-parallel size	Number of nodes	**2**

EXAM TRAP

Leaving both parallel sizes unset without `model.profile` does not fall back to a default; the Helm chart fails to render.

Networking between nodes

Multi-node TP needs a high-bandwidth, low-latency interconnect. InfiniBand with RDMA is one example. InfiniBand or RoCE is recommended for optimal NCCL performance. Workers join the Ray cluster on port **6379** by default, so that port must be reachable between pods.

EXAM TRAP

Pod-to-pod traffic is not optional: workers join Ray on port **6379** by default, so network policies must let it through.

Storage and logging settings

Use `ReadWriteMany` when persistence is enabled. A `ReadWriteOnce` PVC cannot be mounted on multiple nodes. Also set `model.jsonLogging=false` in Helm values, because the JSON log formatter breaks Ray worker initialization.

EXAM TRAP

`ReadWriteOnce` storage does not work for multi-node persistence; such a PVC cannot be mounted on multiple nodes.

Automating with the NIM Operator

The `NIMService` resource automates multi-node deployment. The operator generates the leader and worker pod specs and injects the Ray startup commands. It handles PVCs through NIMCache and manages probes, networking and secrets.

Sharded evaluation on Kubernetes

NeMo Evaluator shards work with an Indexed Job. Each pod's `JOB_COMPLETION_INDEX` is mapped to `NEL_SHARD_IDX`, giving every pod its own shard index.

Key takeaways

- Multi-node NIM runs one image for leader and worker pods, with Ray forming the cluster.
- Common sizing: tensor-parallel size equals GPUs per node, pipeline-parallel size equals the number of nodes.
- Set both parallel sizes above **0** or set `model.profile`, or the chart will not render.
- With persistence enabled, multi-node needs `ReadWriteMany` storage; also set `model.jsonLogging=false`.

Sources: Multi-Node Deployment — NVIDIA NIM for LLM and VLM (opens in a new tab) · Kubernetes Deployment — NVIDIA NIM for LLM and VLM (opens in a new tab) · NeMo Evaluator: Deployment Patterns | NVIDIA NeMo Evaluator (opens in a new tab)

Apply MLOps practices for continuous integration and continuous delivery (CI/CD) workflows, monitoring, and governance.

Start here

THINK FIRST

How would you keep an agent's evaluations repeatable and its behavior observable as it moves through a delivery pipeline?

Why it matters: Questions describe an evaluation or monitoring setup and ask which nat eval option or telemetry setting fits it.

Evaluation runs a workflow on curated test data and scores it with metrics such as accuracy, reliability and latency. Saved configurations keep each run reproducible, and telemetry set in `general.telemetry` covers monitoring.

Key terms

Term	Meaning
nat eval	The NeMo Agent Toolkit command that evaluates a workflow, locally or remotely with `--endpoint`.

Term	Meaning
config_effective.yml	The saved final configuration with every command-line override applied.
general.telemetry	The section of the workflow configuration file where observability is set up.
Telemetry exporter	A component that handles events asynchronously in background tasks, off the hot path.

Evaluation in the pipeline

nat eval can test a local or a remote workflow. Add `--endpoint` to evaluate a workflow running on a remote server. The toolkit server also offers evaluation over REST at `/evaluate`. Dataset entries run in parallel, and `eval.general.max_concurrency` defaults to **8**.

EXAM TRAP

Remote evaluation does not need a separate tool. It is still `nat eval`, run with the `--endpoint` flag.

Reproducible runs

`config_effective.yml` **records the configuration actually used.** It is the final configuration after all command-line overrides. Use `--reps` to repeat an evaluation, average the score and see how much the outputs vary.

EXAM TRAP

Your original config file is not the record of a run once you pass overrides. `config_effective.yml` holds the configuration after them.

Managing job output

`append_job_id_to_output_dir` **gives each run its own directory.** When it is true, each run adds a unique `job_{UUID}` directory to the output path. `max_jobs` caps how many job directories are kept. Its default of **0** means no limit.

Two settings for evaluation job directories

Feature	append_job_id_to_output_dir	max_jobs
Purpose	A separate directory for each evaluation run	How many job directories to keep
Behavior	When true, adds job_{UUID} to the output path	Evicts the oldest directories past the limit; 0 means no limit

EXAM TRAP

A `max_jobs` value of 0 does not delete every job directory. Zero means no limit at all.

Telemetry and logging

Configure observability in `general.telemetry`. Console and file logging come by default. OpenTelemetry and Phoenix tracing need the optional telemetry extras. Exporters run asynchronously in background tasks, away from the hot path. A log level records that level and every level above it.

EXAM TRAP

Phoenix tracing is not included out of the box. Install the optional telemetry extras before you use it.

Improving from traces

Traces can feed the Data Flywheel Blueprint. The blueprint uses traces to fine-tune and evaluate smaller models. Those models can replace the original one to reduce latency. For numerical settings, the Optimizer uses Optuna.

Key takeaways

- Use `nat eval` with `--endpoint` to evaluate a workflow on a remote server.
- `config_effective.yml` is the configuration after all command-line overrides.
- `eval.general.max_concurrency` defaults to **8**, and a `max_jobs` of **0** means no limit.
- OpenTelemetry and Phoenix tracing need the optional telemetry extras.

Sources: Agent Evaluation in NVIDIA NeMo Agent Toolkit — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Observe Workflows — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · NVIDIA NeMo Agent Toolkit Optimizer — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Profile performance and reliability under distributed system loads.

Start here

THINK FIRST

How do you find where an agent workflow slows down, and how do you spread a large evaluation across many nodes?

Why it matters: Questions describe a slow or large workload and ask which profiler option or sharding setup to use.

The NeMo Agent Toolkit profiler measures and forecasts workflow performance, then points to bottlenecks and caching chances. NeMo Evaluator splits large evaluations into shards across nodes and merges the results.

Key terms

Term	Meaning
Profiler	The NeMo Agent Toolkit module for profiling and forecasting, shipped as `nvidia-nat-profiler`.
Nested stack analysis	Bottleneck analysis that also catches tool calls running inside other tool calls.
Concurrency spike	A moment when at least `spike_threshold` functions are running at once.
Shard	One slice of an evaluation; its results land in a `shard_N` directory for merging.
Latency sensitivity	A score for how much a call's speed matters; manual annotations override it.

Setting up the profiler

Install `nvidia-nat-profiler` for profiling and forecasting. The profiler hooks into agent frameworks through callback handlers. Supported ones are CrewAI, LangChain/LangGraph, LlamaIndex, Semantic Kernel and Google ADK.

Tracking custom functions

`@track_function` **profiles your own functions.** It records when a function starts and ends, plus its input arguments and output. It also works on async functions, generators and class methods.

Example: Marks a custom function so the profiler records its timing, inputs and output.

```
@track_function
def my_tool(my_input):
    return my_input
```

- `@track_function`: the decorator that adds profiling

Reading the results

`inference_optimization.json` **holds the workflow metrics.** It reports **90%**, **95%** and **99%** confidence intervals for three measures: latency, throughput and workflow runtime.

Finding bottlenecks

Nested stack analysis finds bottlenecks inside bottlenecks. It can spot a slow tool call running inside another tool call. Choose `simple_stack` when a simpler analysis is enough. Other options flag concurrency spikes and repeated prompt prefixes.

Profiler analysis options compared

Option	What it finds	Note
Nested stack	Nested bottlenecks, such as tool calls inside tool calls	The more detailed analysis
`simple_stack`	Bottlenecks without the nesting detail	The simpler analysis
`spike_threshold`	Moments of high concurrency	At **7**, flags **7** or more functions running at once

Option	What it finds	Note
`prompt_caching_prefixes`	Commonly repeated prompt prefixes	Candidates to pre-populate in KV caches

EXAM TRAP

A `spike_threshold` of **7** does not mean more than 7. It flags any spike with **7** or more functions running at once.

Latency sensitivity

Manual annotations always beat computed scores. The critical path weight, `w_critical`, defaults to **0.5** . It asks what share of the workflow's wall-clock time a call uses. A `@latency_sensitive` annotation overrides any computed value.

EXAM TRAP

An auto-computed sensitivity does not win when both exist. A manual `@latency_sensitive` annotation always takes priority.

Scaling evaluations

NeMo Evaluator scales with SLURM, Kubernetes, Ray or manual shards. On Kubernetes, an Indexed Job gives each pod a `JOB_COMPLETION_INDEX`, mapped to `NEL_SHARD_IDX`. Afterward, `nel eval merge` finds every `shard_N` directory, drops duplicate results and builds merged bundles with combined scores.

EXAM TRAP

Sharding and `auto_resume` cannot be used together. For per-task retries in SLURM array mode, use `--requeue` instead.

Experimental integrations

The profiler's Dynamo integration is experimental. It needs Dynamo **1.1.0** or later, so check the installed version before you rely on it.

Key takeaways

- The profiler ships as `nvidia-nat-profiler` and covers both profiling and forecasting.
- Nested stack analysis finds tool calls hidden inside other tool calls.
- Manual `@latency_sensitive` annotations always override auto-computed values.
- Evaluator shards cannot use `auto_resume`; combine them with `nel eval merge`.

Sources: Profiling and Performance Monitoring of NVIDIA NeMo Agent Toolkit Workflows — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Distributed Evaluation | NVIDIA NeMo Evaluator (opens in a new tab)

Scale deployments using containerization (Docker, Kubernetes) with load balancing.

Start here

THINK FIRST

How would you run NIM on Kubernetes so it scales with inference load and survives restarts and upgrades?

Why it matters: Questions describe a NIM deployment on Kubernetes and ask which chart setting, operator resource or scaling metric fits.

NIM runs on Kubernetes through Helm charts or the NIM Operator. The operator handles deployment, scaling, model caching and health monitoring. Scaling works on inference metrics from Prometheus, not on CPU or memory.

Key terms

Term	Meaning
NIM Operator	A Kubernetes operator that manages NIM deployment, scaling, model caching and health monitoring.
NIMService	Custom resource for pod creation, health probes, service exposure and GPU scheduling.
NIMCache	Custom resource that keeps model files on persistent storage for reuse after restarts and scaling.
HPA	Horizontal pod autoscaling; for NIM it should use inference-specific Prometheus metrics.
PriorityClass	A pod priority setting; without one, NIM pods are lowest priority and can be preempted.

Helm chart basics

The nim-llm chart needs Kubernetes v1.26 or later with GPU nodes. By default the chart deploys a StatefulSet. Set `statefulSet.enabled` to false to get a Deployment. Set `multiNode.enabled=true` to get LeaderWorkerSet resources.

Workload types the nim-llm chart creates

Feature	StatefulSet	Deployment	LeaderWorkerSet
Chart setting	statefulSet.enabled: true	statefulSet.enabled: false	multiNode.enabled=true
When the chart creates it	By default	When statefulSet.enabled is false	When multiNode.enabled=true

EXAM TRAP

The chart does not create a Deployment by default. It creates a StatefulSet unless `statefulSet.enabled` is false.

Operator resources

NIMService runs the model; NIMCache stores its files. NIMService handles pod creation, health probes, service exposure and GPU scheduling. NIMCache keeps model artifacts on persistent storage. The model downloads once, so later pods reuse it after restarts or scaling.

Health and startup

A healthy pod returns HTTP 200 from `/v1/health/ready`. The operator's startup probe on that endpoint uses `failureThreshold` **120** and `periodSeconds` **10**. So a model may take up to **20** minutes to load before Kubernetes kills the pod.

EXAM TRAP

A slow first start is not automatically a failure. The default startup probe allows up to 20 minutes for the model to load.

Autoscaling on the right metric

Scale NIM on inference metrics, not CPU or memory. NIM exposes the backend's Prometheus metrics at `/v1/metrics`. To drive HPA from them, install the Prometheus Adapter so they appear as Kubernetes custom metrics.

EXAM TRAP

CPU and memory targets are not useful for scaling NIM. Point HPA at inference-specific custom metrics instead.

Upgrades and priority

Use `Recreate` when no spare GPU pod fits during upgrades. The default RollingUpdate strategy starts the new pod first, so the cluster needs room for one extra. Give NIM pods a PriorityClass too.

EXAM TRAP

Leaving out a PriorityClass is not neutral. NIM pods then get the lowest priority, and other workloads can preempt them.

Evaluator container

The NeMo Evaluator image runs `nel` as its entrypoint. Its Dockerfile installs the package with the `[scoring]` extras, then sets the container entrypoint to `nel`.

Key takeaways

- The nim-llm chart deploys a StatefulSet by default; `statefulSet.enabled: false` gives a Deployment.
- NIMCache keeps model artifacts on persistent storage for reuse after restarts and scaling.
- Autoscale NIM on custom Prometheus metrics, exposed through the Prometheus Adapter.
- The default startup probe allows up to **20** minutes for the model to load.

Sources: Kubernetes Deployment — NVIDIA NIM for LLM and VLM (opens in a new tab) · NIM Operator Deployment — NVIDIA NIM for LLM and VLM (opens in a new tab) · Helm and Kubernetes — NVIDIA NIM for LLM and VLM (opens in a new tab) · Docker Deployment | NVIDIA NeMo Evaluator (opens in a new tab)

Optimize deployment costs while ensuring high availability.

Start here

THINK FIRST

How would you cut the GPU and tuning costs of an agent deployment without losing the accuracy and availability it needs?

Why it matters: Questions describe a NIM or agent deployment and ask which profile, setting or trade-off lowers cost.

On the serving side, NIM picks a model profile that fits your hardware, and the profile fixes the engine, precision and parallelism. On the agent side, the NeMo Agent Toolkit Optimizer searches for settings that maximize quality while minimizing response time and cost.

Key terms

Term	Meaning
Model profile	A pre-validated inference setup chosen at container startup, tagged by engine, precision, TP, PP and LoRA support.
NIM_MODEL_PROFILE	Variable that pins a profile by ID or description, replacing the deprecated 1.x selectors.
--max-model-len	Flag that caps input plus output tokens per request, which lowers a profile's memory need.
Optimizer	NeMo Agent Toolkit tool that tunes numbers with Optuna and prompts with a genetic algorithm.

How NIM picks a profile

Without `NIM_MODEL_PROFILE`, NIM picks the best compatible profile. It weighs the GPU device, free VRAM, estimated memory needs and parallelism limits. Within each memory category, NVIDIA-optimized profiles rank ahead of generic ones.

EXAM TRAP

The custom and tag-based selectors from NIM LLM 1.x are deprecated. Pin a profile with `NIM_MODEL_PROFILE` instead.

Checking what fits

`list-model-profiles` **shows compatible profiles and their VRAM needs.** Each profile carries an estimated per-GPU VRAM figure. Profiles are tagged by backend engine, precision, tensor parallelism, pipeline parallelism and LoRA support. Precisions include **bf16**, **fp8**, **mxfp4** and **nvfp4**.

EXAM TRAP

The per-GPU VRAM figure is an estimate, not a guarantee. NVIDIA describes memory estimates as advisory.

When a profile will not fit

Raise tensor parallelism or choose a quantized precision. Both help a profile marked incompatible because it exceeds GPU memory. Passing `--max-model-len`, for example **4096**, also lowers the memory need by capping input plus output tokens per request.

Multi-node downloads

Give multi-node NIM shared storage for the model. Without it, every node downloads the model separately. That still works, but it wastes time and network bandwidth. On AWS, NIM for LLM and VLM runs on either Amazon EKS or SageMaker.

Tuning the agent

The Optimizer trades quality against response time and cost. It uses Optuna for numerical parameters and a genetic algorithm for prompts. `multi_objective_combination_mode` merges several objectives into one score: `harmonic` (the default), `sum` or `chebyshev`.

EXAM TRAP

`sum` is not the default combination mode. The Optimizer uses `harmonic` unless you set another mode.

Choosing a result

When latency is critical, take the fastest point that meets accuracy. Among the Pareto-optimal points, choose the fastest one whose accuracy is still good enough. Keep tuning budgets in mind too: for genetic prompt optimization, cost grows linearly with population size.

Key takeaways

- If `NIM_MODEL_PROFILE` is unset, NIM selects the best compatible profile for the hardware.
- A profile too big for GPU memory may fit with higher tensor parallelism or a quantized precision.
- `--max-model-len` caps tokens per request and lowers a profile's memory need.

- The Optimizer combines objectives with `harmonic` by default.

Sources: Model Profiles and Selection — NVIDIA NIM for LLM and VLM (opens in a new tab) · NVIDIA NeMo Agent Toolkit Optimizer — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Multi-Node Deployment — NVIDIA NIM for LLM and VLM (opens in a new tab) · Cloud Service Provider (CSP) Deployment — NVIDIA NIM for LLM and VLM (opens in a new tab)

Domain 5 | 10% of the exam | 13 min read

Cognition, Planning, and Memory

Overview

Every point below comes from NVIDIA's official documentation; each section lists its sources.

Implement memory mechanisms for short- and long-term context retention.

Start here

THINK FIRST

How do you give an agent memory that keeps user context across steps, and when would you build your own backend?

Why it matters: Questions describe a memory need and ask which wrapper, interface, provider or config setting delivers it.

NeMo Agent Toolkit memory lets a stateful app recall user-specific data across multiple steps. You configure a memory backend and reach it through a memory client, or wrap an agent so history is stored and retrieved automatically.

Key terms

Term	Meaning
MemoryItem	The main object that represents one piece of memory.
MemoryEditor	The low-level abstract interface for adding, searching and removing memory items.
MemoryManager	The abstract interface for higher-level operations such as summarization or reflection.
MemoryBaseConfig	The Pydantic base class that every memory config class must extend.
Auto-memory wrapper	Wraps an agent so conversation history is stored and retrieved for it automatically.

Why agents need memory

Memory lets an agent recall user-specific data across steps. That recall is what makes an LLM application stateful. Your code or tools get a memory instance by calling `builder.get_memory_client()` with the

name of the memory config.

Example: Gets the memory instance configured under the name user_store.

```
my_memory = builder.get_memory_client("user_store")
```

- `"user_store"`: the memory config's name

The memory interfaces

Three abstractions split the memory work. A `MemoryItem` is the data. `MemoryEditor` handles item-level changes, while `MemoryManager` handles higher-level work when you need it.

Memory abstractions compared

Abstraction	Level	What it handles
`MemoryItem`	Main data object	One piece of memory
`MemoryEditor`	Low-level abstract interface	Adding, searching and removing items
`MemoryManager`	Higher-level abstract interface	Summarization or reflection

EXAM TRAP

Summarization is not a `MemoryEditor` job; higher-level operations such as summarization or reflection belong to `MemoryManager`.

Automatic memory for an agent

The auto-memory wrapper stores and retrieves history for you. It captures both user messages and agent responses, so your workflow makes no manual memory calls. The `auto_memory_agent` requires `inner_agent_name`, which names the agent to wrap.

Keeping memory per user

A `user_id_resolver` callable runs for every memory operation. The callable can be synchronous or asynchronous. For `nat run`, the console front end's `user_id` defaults to `nat_run_user_id`.

EXAM TRAP

The resolver is not called once per session and cached; it is invoked for every memory operation.

Plugin backends

Zep and Redis memory come as plugins. Zep is provided by the `nvidia-nat-zep-cloud` plugin. Redis is provided by the Redis-maintained `nemo-agent-toolkit-redis` plugin.

Building a custom provider

Extend `MemoryBaseConfig` and implement a `MemoryEditor`. The editor must handle `add_items`, `search` and `remove_items` calls. Declaring `name="my_custom_memory"` on the config class lets the

toolkit recognize `_type: my_custom_memory` in the memory config.

EXAM TRAP

A custom memory config is not just any class; it must extend the Pydantic base class `MemoryBaseConfig`.

Key takeaways

- Wrap an agent with auto-memory to store user messages and agent responses automatically.
- `MemoryEditor` adds, searches and removes items; `MemoryManager` summarizes or reflects.
- Every memory config class extends `MemoryBaseConfig`, and its `name` is what `_type` refers to.
- Get a memory instance with `builder.get_memory_client()` and the memory config's name.

Sources: Memory in NVIDIA NeMo Agent Toolkit — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Adding a Memory Provider — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Apply reasoning frameworks (chain-of-thought, task decomposition).

Start here

THINK FIRST

When should an agent plan its reasoning up front, and when should it reason between each action?

Why it matters: Questions describe an agent task and ask which reasoning approach or agent type fits it, and what that agent requires.

Reasoning frameworks such as chain-of-thought and task decomposition shape how an agent works toward a goal. In NeMo Agent Toolkit, the reasoning agent plans ahead of time, while a ReAct agent reasons between steps.

Key terms

Term	Meaning
Reasoning agent	A toolkit agent that plans ahead and calls an underlying function directly, reasoning on top of it.
ReAct agent	A toolkit agent that reasons between steps; its output_parser reads the LLM's output.
Reasoning LLM	A model that supports reasoning; the reasoning agent needs one.
ReWOO agent	A LangChain-based toolkit agent with agent, prompt and register submodules.

Plan first or reason per step

The reasoning agent plans ahead; ReAct reasons between steps. Both are NeMo Agent Toolkit agents. The reasoning agent works out a plan before acting, then invokes the underlying function directly. A ReAct agent does its reasoning between steps instead.

Reasoning agent and ReAct agent compared

Feature	Reasoning agent	ReAct agent
When it reasons	Ahead of time, through planning	Between steps
Notable parts	Invokes an underlying function directly	output_parser and register_per_user_agent submodules

EXAM TRAP

The reasoning agent does not reason between steps the way ReAct does. It reasons ahead of time through planning.

Choosing the model

A reasoning agent needs an LLM that supports reasoning. NIM lists supported profiles for reasoning models such as `nvidia/nemotron-3-nano-omni-30b-a3b-reasoning`. The `meta/muse-glimmer` model reasons before answering and needs a cache-permission step during deployment.

EXAM TRAP

Not every LLM will work with the reasoning agent. You must choose one that supports reasoning.

Shaping the reasoning

Customize the reasoning agent's prompts in YAML. The workflow's YAML config file holds prompts you can tailor to your task. The exam expects you to excel at goal-oriented reasoning and chain-of-thought prompt engineering.

Other agent styles

The toolkit also ships a LangChain-based ReWOO agent. Its package has agent, prompt and register submodules. Like the ReAct agent, it sits in the toolkit's LangChain plugin.

Key takeaways

- The reasoning agent reasons ahead of time through planning; ReAct reasons between steps.
- A reasoning agent needs an LLM that supports reasoning.
- You can customize the reasoning agent's prompts in the YAML config file.

Sources: NVIDIA-Certified Professional: Agentic AI Exam Study Guide (opens in a new tab) · About Reasoning Agent — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.plugins.langchain.agent.react_agent — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.plugins.langchain.agent.rewoo_agent — NVIDIA

NeMo Agent Toolkit (1.9) (opens in a new tab) · Support Matrix for NIMs — NVIDIA NIM for LLM and VLM (opens in a new tab)

Engineer planning strategies for sequential and multi-step decision-making.

Start here

THINK FIRST

When should an agent follow a fixed chain of steps, and when should it generate and compare several plans?

Why it matters: Questions describe a multi-step task and ask whether a sequential executor or a multi-plan planner fits, and how to set it up.

A sequential executor chains functions in a fixed order, passing each output to the next function, with no LLM orchestration. The experimental multi-plan planners instead produce several candidate plans, either in a single shot or by combining plans from multiple LLMs.

Key terms

Term	Meaning
Sequential executor	A control flow component that chains functions, feeding each output into the next one.
tool_list	The SequentialExecutorConfig field that lists the FunctionRef entries to run in order.
SequentialExecutorExit	The exception a tool raises to leave the chain early with a custom message.
Test-time compute (TTC)	The experimental search module that holds the multi-plan planners.

Fixed chains of functions

A sequential executor runs functions in a set order. Each function's output becomes the next function's input. The order is decided in advance, so no LLM or agent has to orchestrate the steps.

EXAM TRAP

A sequential executor does not ask an LLM which tool to call next. It follows a predetermined order.

Configuring the chain

List the functions to run, in order, in `tool_list`. SequentialExecutorConfig adds options for type mismatches and for exceptions. ToolExecutionConfig sets options for each individual tool in the chain.

Fields that shape a sequential chain

Field	Belongs to	What it controls
tool_list	SequentialExecutorConfig	The functions to run, in order
raise_type_incompatibility	SequentialExecutorConfig	Handling of type mismatches between chained functions
return_error_on_exception	SequentialExecutorConfig	How exceptions in the chain are handled
use_streaming	ToolExecutionConfig	Streaming for one tool in the chain

Type checks and early exit

You can validate types between chained functions. The check confirms one function's output type fits the next function's input. A tool that must stop the chain raises `SequentialExecutorExit` with a custom message. Streaming and compatibility validation can be customized in YAML config files.

EXAM TRAP

Type validation between chained functions is something you opt in to. It is not described as always on.

Generating several plans

Multi-plan planners produce several candidate plans. SingleShotMultiPlanPlanner generates multiple plans in one shot. MultiLLMPlanner queries several LLMs and combines every plan they return. It binds each LLMRef with the LANGCHAIN framework wrapper.

Two multi-plan planners compared

Feature	SingleShotMultiPlanPlanner	MultiLLMPlanner
How plans are made	Several plans in a single shot	Each bound LLM makes plans_per_llm plans
What you get back	Multiple plans	All plans from every LLM, combined

EXAM TRAP

These planners are experimental. They live in the test-time compute (TTC) search module, not in the sequential executor.

Key takeaways

- A sequential executor passes each function's output to the next, in a predetermined order.
- List the chained functions in `tool_list`.
- Raise `SequentialExecutorExit` to leave the chain early with a custom message.
- MultiLLMPlanner combines the `plans_per_llm` plans from each bound LLM.

Sources: About the Sequential Executor — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.experimental.test_time_compute.search.single_shot_multi_plan_planner — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.experimental.test_time_compute.search.multi_llm_planner — NVIDIA NeMo

Agent Toolkit (1.9) (opens in a new tab) · nat.plugins.langchain.control_flow.sequential_executor — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Manage stateful orchestration to coordinate complex tasks and knowledge retention.

Start here

THINK FIRST

How do you coordinate many functions in one task while keeping step history and knowledge available throughout?

Why it matters: Questions describe a multi-step workflow and ask which orchestration, step-tracking or knowledge base feature keeps it coordinated.

Stateful orchestration has three parts here. A TTC orchestration function coordinates multiple functions, an intermediate step manager tracks the steps in the event stream, and a knowledge base keeps documents ready as context.

Key terms

Term	Meaning
TTCToolOrchestrationFunction	A NeMo Agent Toolkit function used to orchestrate multiple functions.
TTCItem	The form each `ToolUselist` item is converted to inside the orchestration function.
IntermediateStepManager	Manages updates to the NAT Event Stream for intermediate steps.
kb folder	The folder inside the configuration folder that holds knowledge base documents.
$relevant_chunks	The context variable that holds the retrieved knowledge base chunks.

Orchestrating multiple tools

The TTC orchestration function coordinates multiple functions. You configure it with `TTCToolOrchestrationFunctionConfig`. It instantiates every relevant strategy: search, editing, scoring and selection.

The orchestration pipeline

Items move from a `ToolUselist` through optional strategies. Some stages always apply and some only when configured. The function returns a new `ToolUselist` with each output set.

Stages of TTC tool orchestration

Stage	What it does	Detail
Convert	Turns each item into a `TTCItem`	Input is a `ToolUselist`
Search and edit	Runs search and editing strategies	Optional
Call	Runs the matching augmented function per item	Matched by tool name
Score and select	Runs scoring and selection on the result	Only if configured

EXAM TRAP

Scoring and selection are not stages that always run; the function applies them to the result only if configured.

Tracking intermediate steps

`IntermediateStepManager` **updates the NAT Event Stream.** `get_outstanding_step_count` returns how many steps have started but not yet ended. `push_intermediate_steps` injects steps into the event stream without updating the manager's internal stack.

EXAM TRAP

`push_intermediate_steps` does not add the steps to the manager's internal stack; it only injects them into the event stream.

Replaying remote workflow steps

Replay remote steps so the full trace tree is visible. Pass the remote workflow's steps to `push_intermediate_steps`. Invoke the remote with the right `workflow-parent-id` and `workflow-parent-name`, so the replayed root step gets the correct parent.

EXAM TRAP

Replayed steps do not find their parent on their own; the remote must be invoked with `workflow-parent-id` and `workflow-parent-name`.

Knowledge retention with a knowledge base

Put documents in the `kb` **folder inside the configuration folder.** The knowledge base turns on automatically when documents are there. Customize it in the `knowledge_base` section of `config.yml`. Currently only Markdown documents are supported.

EXAM TRAP

Placing files of other formats in `kb` does not extend the knowledge base; currently only the Markdown format is supported.

Chunking and retrieval

Documents are split into topic chunks at markdown headers. Large chunks are split again at blank lines to stay within a maximum chunk size. Retrieved chunks are then available as `$relevant_chunks` in the context.

Key takeaways

- The TTC orchestration function converts items, calls tools by name, and can search, edit, score and select.
- `get_outstanding_step_count` counts steps that have started but not yet ended.
- `push_intermediate_steps` replays steps into the event stream without updating the internal stack.
- Knowledge base documents go in `kb`, must be Markdown, and reach the context as `$relevant_chunks`.

Sources: nat.experimental.test_time_compute.functions.ttc_tool_orchestration_function — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.builder.intermediate_step_manager — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Knowledge Base Configuration | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab)

Adapt reasoning strategies based on prior experiences and feedback.

Start here

THINK FIRST

How can an agent's plans, prompts and model improve from earlier results and evaluation feedback?

Why it matters: Questions describe an agent that should learn from past results and ask which planning, feedback or finetuning mechanism fits.

NeMo Agent Toolkit adapts reasoning at three levels. A reasoning agent plans ahead, oracle feedback feeds evaluation failures back into prompt evolution, and finetuning sends trajectories away to update model weights.

Key terms

Term	Meaning
Reasoning agent	An agent that reasons ahead of time through planning, while a ReAct agent reasons between steps.
Oracle feedback	Failure reasoning from evaluation results, injected into the prompt optimization genetic algorithm.
Worst items	The worst-performing evaluation items, whose reasoning `extract_worst_reasoning` pulls out.

Term	Meaning
TrainerAdapter	Sends trajectories to a remote training cluster for weight updates.
LLMBasedPlanScorer	An experimental test-time compute scorer that uses an LLM to score each planning item.

Planning before acting

A reasoning agent plans ahead instead of between steps. That is its key difference from a ReAct agent. You must pick an LLM that supports reasoning, and you can customize its prompts in the YAML config file.

EXAM TRAP

A reasoning agent does not think between tool calls like ReAct; it reasons ahead of time through planning.

Scoring plans with an LLM

`LLMBasedPlanScorer` **uses an LLM to score planning items.** It is an experimental part of test-time compute. Its `ainvoke` method scores a list of planning items, given the original prompt and the agent context.

Learning from evaluation failures

Oracle feedback grounds prompt evolution in real failures. It extracts, formats and injects failure reasoning into the prompt optimization genetic algorithm. `extract_worst_reasoning` takes reasoning from the worst items, using each evaluator's direction: maximize or minimize. `build_oracle_feedback` turns it into a string capped by `max_chars`.

When feedback is injected

Feedback injection has 4 modes. They are `never`, `always`, `failing_only` and `adaptive`. The `failing_only` mode uses a `fitness_threshold`, while `adaptive` fires on the triggers below.

The adaptive mode's three triggers

Trigger	Setting	What it watches
Stagnation	`stagnation_generations`	Generations without improvement
Collapse	`fitness_variance_threshold`	Fitness variance, to detect collapse
Low diversity	`diversity_threshold`	Ratio of duplicated prompts

EXAM TRAP

`fitness_threshold` belongs to `failing_only` mode; the adaptive triggers are stagnation, fitness variance and prompt duplication.

Finetuning from trajectories

A `TrainerAdapter` sends trajectories for weight updates. Its `submit` method sends them to the remote training backend and returns a `TrainingJobRef`. Then `wait_until_complete` polls that job, waiting `poll_interval` seconds between status checks.

> **EXAM TRAP**
> Oracle feedback changes prompts, not model weights; weight updates come from trajectories sent through a `TrainerAdapter`.

Key takeaways

- Reasoning agents plan ahead; ReAct agents reason between steps.
- Oracle feedback injects failure reasoning from evaluations into the prompt optimization genetic algorithm.
- Feedback modes are `never`, `always`, `failing_only` and `adaptive`.
- A `TrainerAdapter` sends trajectories to a remote cluster for weight updates.

Sources: About Reasoning Agent — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.plugins.config_optimizer.prompts.oracle_feedback — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.finetuning.interfaces.trainer_adapter — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.experimental.test_time_compute.scoring.llm_based_plan_scorer — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Domain 6 | 10% of the exam | 14 min read

Knowledge Integration, and Data Handling

Overview

Every point below comes from NVIDIA's official documentation; each section lists its sources.

Implement retrieval pipelines (RAG, embedded search, hybrid approaches).

Start here

THINK FIRST

When should NeMo Guardrails manage retrieval for a RAG app, and when should the app retrieve chunks itself?

Why it matters: Questions describe a RAG setup and ask which mode, knowledge base option or embedding search setting fits it.

NeMo Guardrails uses embedding search for both its rails and its knowledge base. In a RAG app, you either let it manage retrieval from a configured knowledge base, or retrieve chunks yourself and pass them in.

Key terms

Term	Meaning
Knowledge Base mode	You configure a knowledge base in the guardrails config, and NeMo Guardrails manages retrieval.
Relevant Chunks mode	Your app performs retrieval and passes the chunks directly to the `generate` method.
EmbeddingsIndex	The class you subclass to build a custom embedding search provider.
Embedding cache	Stores computed embeddings so they are not recomputed; disabled by default.
NVIDIARAGToolConfig	The NeMo Agent Toolkit tool config for searching the NVIDIA Developer database.

Two ways to plug in RAG

First decide who performs the retrieval. Embedding search, also known as a vector database, powers both the guardrails process and the knowledge base. The modes differ in who retrieves and where chunks come from.

Knowledge Base and Relevant Chunks modes compared

Feature	Knowledge Base mode	Relevant Chunks mode
Who retrieves	NeMo Guardrails	Your application
Where chunks come from	A knowledge base in the guardrails config	Passed directly to the `generate` method

EXAM TRAP

In Relevant Chunks mode, NeMo Guardrails does not retrieve anything for you; your app retrieves the chunks and passes them in.

Passing chunks yourself

Send chunks in a message with the role `context`. The message content holds a `relevant_chunks` field with your retrieved text. You pass that message to `rails.generate`.

Example: A context message that hands your own retrieved chunks to the generate call.

```
{
  "role": "context",
  "content": {
    "relevant_chunks": "..."
  }
}
```

- `"relevant_chunks"`: holds the text your app retrieved

Configuring a knowledge base

A knowledge base can be configured in 3 ways. Use the `kb` folder, a custom `retrieve_relevant_chunks` action, or a custom `EmbeddingSearchProvider`. The `kb` folder goes inside the config folder.

EXAM TRAP

The `kb` folder does not accept any document type; currently only the Markdown format is supported.

The default embedding search

By default, FastEmbed computes embeddings with `all-MiniLM-L6-v2`. An exact NumPy index then handles the similarity search. The batch feature starts embedding generation after a predefined latency of **10** milliseconds.

Caching embeddings

Embedding caching is disabled by default. When enabled, it stores computed embeddings to avoid repeated computation and speed up search. The default cache configuration uses the `sha256` key generator and the filesystem store.

EXAM TRAP

Embedding caching is not on out of the box; by default the caching mechanism is disabled.

Custom embedding search providers

Subclass `EmbeddingsIndex` to build your own provider. Its `search` method takes the query text, `max_results` and an optional `threshold`, and returns a list of `IndexItem` objects. Register the provider in the `init` function of `config.py`.

Example: Registers a custom embedding search provider under the name simple.

```
def init(app: LLMRails):
    app.register_embedding_search_provider(
        "simple", SimpleEmbeddingSearchProvider
    )
```

- `"simple"`: the name the provider is registered under

The Agent Toolkit RAG tool

`NVIDIARAGToolConfig` **searches the NVIDIA Developer database.** It finds documents across many NVIDIA asset types. Its settings include `top_k` and `collection_name`, alongside `base_url`, `timeout`, `document_separator` and `document_prompt`.

Key takeaways

- Knowledge Base mode lets NeMo Guardrails retrieve; Relevant Chunks mode passes your own chunks to `generate`.
- Pass your chunks as a `context` role message with a `relevant_chunks` field.
- Default search uses FastEmbed with `all-MiniLM-L6-v2` and an exact NumPy index.
- Embedding caching is off by default; its default configuration uses `sha256` keys and a filesystem store.

Sources: Embedding Search Providers | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · Retrieval-Augmented Generation | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · nat.tool.nvidia_rag — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Configure and optimize vector databases for fast retrieval.

Start here

THINK FIRST

Which retrieval and caching settings would you change first to make a guardrailed RAG application answer faster?

Why it matters: Questions describe a retrieval or latency problem and ask which embedding, knowledge base or cache setting addresses it.

Retrieval runs as a vector similarity search over a knowledge base of Markdown documents, using a configurable embedding search provider. Caching then cuts repeated work, either by reusing whole LLM responses or by reusing the system prompt's computation inside NIM.

Key terms

Term	Meaning
Embedding search provider	The component that runs vector similarity search, using FastEmbed, OpenAI or a custom implementation.
Knowledge base	A folder of Markdown documents that NeMo Guardrails uses for RAG-based responses.
In-memory model cache	A cache that returns stored LLM responses for repeated prompts, evicting entries by LFU.
KV cache reuse	A NIM-level optimization so each NemoGuard NIM call skips recomputing its system prompt.

Pick an embedding engine

FastEmbed is the default embedding engine. The other supported engines are `nim` for NVIDIA NIM embeddings and `openai` for OpenAI embeddings. An embedding model is declared in the models list with `type: embeddings`, for example FastEmbed with `all-MiniLM-L6-v2`.

EXAM TRAP

There are three engines, not two. Besides FastEmbed and OpenAI, the `nim` engine serves NVIDIA NIM embeddings.

Set up the knowledge base

The `knowledge_base` section in config.yml points to the documents folder. It also names the `embedding_search_provider` and holds a cache setting. For example, the folder can be `kb`, the provider `default`, and the cache `enabled: false`.

Two ways to cache

Guardrails offers two caching strategies to reduce inference latency. You can turn on either one, or both, independently. They save work at different layers.

In-memory model cache and KV cache reuse compared

Feature	In-memory model cache	KV cache reuse
What it reuses	Stored LLM responses for identical prompts	The system prompt's computation
Where it works	In the Guardrails library	At the NIM level, on NemoGuard NIM calls
Notable detail	Evicts entries using LFU	Avoids redoing the system prompt each call

Model cache defaults

A model's cache is off by default. Set `models.cache.enabled` to true to use it. Its `maxsize` defaults to **50000** entries. Cache statistics tracking is also off until you enable `models.cache.stats.enabled`.

EXAM TRAP

Caching is not on out of the box. Both `models.cache.enabled` and `models.cache.stats.enabled` default to false.

Key takeaways

- FastEmbed is the default embedding engine; `nim` and `openai` are the alternatives.
- The knowledge base is a folder of Markdown documents used for RAG-based responses.
- The model cache reuses whole responses with LFU eviction; KV cache reuse saves system prompt computation in NIM.
- The model cache is disabled by default and holds up to **50000** entries when enabled.

Sources: Other Configurations | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · Caching Instructions and Prompts | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · Configuration YAML Schema Reference | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab)

Build extract, transform, and load (ETL) pipelines to integrate enterprise or client data sources.

Start here

THINK FIRST

How would you connect an agent workflow to outside data sources and tools so their data flows into it reliably?

Why it matters: Questions describe an enterprise data source or service and ask which client, transport or setting brings it into a workflow.

The toolkit brings outside data in through function groups. An MCP client turns a server's tools into functions, and a RAG client adds search and generation. Dataset loaders hand data over as a DataFrame.

Key terms

Term	Meaning
MCP host	A workflow that creates MCP clients, which connect to MCP servers so their tools work as functions.
mcp_client	A function group that connects to an MCP server, discovers its tools and registers them as toolkit functions.
Agent Card	What an `a2a_client` fetches to discover a remote agent's skills, by default from `/.well-known/agent-card.json`.
load_fn	The callable in a DatasetLoaderInfo that returns a pandas DataFrame.

Workflows as MCP hosts

A workflow that uses MCP tools acts as an MCP host. It creates MCP clients that connect to MCP servers. You can use several `mcp_client` function groups in one workflow to pull tools from several servers. The feature needs the nvidia-nat-mcp package.

Shaping the tool list

`mcp_client` **discovers a server's tools and registers them as functions.** Use `include` and `exclude` to filter which tools come in. Use `tool_overrides` to change a tool's name or description from what the server defines.

Choosing an MCP transport

streamable-http is the default and recommended for new deployments. Both sse and streamable-http need `server.url`. Protected servers need an `auth_provider`, and that only works with streamable-http.

> **EXAM TRAP**
>
> An `auth_provider` for a protected MCP server is not supported on every transport. It works only with streamable-http.

MCP client limits

Know the `mcp_client` defaults for timeouts, sessions and reconnects. These settings decide how long a call may take and how the client recovers.

mcp_client settings and their defaults

Setting	What it controls	Default
tool_call_timeout	Timeout for MCP tool calls	**60** seconds
max_sessions	Maximum concurrent session clients	**100**
reconnect_max_attempts	Maximum reconnect attempts	**2**

RAG and remote agents

The RAG client plugin exposes search and generate tools on one shared client. To reach a remote agent instead, use `a2a_client`. It reads the agent's Agent Card to discover its skills, and its default transport is JSON-RPC over HTTP.

EXAM TRAP

A2A clients are per-user function groups. A shared workflow like `react_agent` cannot use them directly.

Loading datasets

A dataset loader's `load_fn` returns a pandas DataFrame. The loader hands the toolkit its data as a DataFrame through this callable in DatasetLoaderInfo.

Key takeaways

- A workflow using MCP tools is an MCP host, and it can combine several `mcp_client` groups.
- streamable-http is the default MCP transport and the only one that supports `auth_provider`.
- `mcp_client` defaults: **60**-second tool call timeout, **100** sessions and **2** reconnect attempts.
- The RAG client plugin's search and generate tools share a single RAG client.

Sources: NVIDIA NeMo Agent Toolkit as an MCP Client — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.plugins.rag.client — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.builder.dataset_loader — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · NVIDIA NeMo Agent Toolkit Workflow as an A2A Client — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Conduct data quality checks, augmentation, and preprocessing.

Start here

THINK FIRST

How would you set up a release gate so that only real regressions on the benchmarks you care about block a model?

Why it matters: Questions describe gate results or a policy file and ask which verdict nel gate returns or which setting to change.

nel gate compares baseline and candidate results against a policy YAML that lists benchmarks, tiers and thresholds. It judges each benchmark on paired per-item data, then rolls the tiers up into one release verdict.

Key terms

Term	Meaning
Policy YAML	The file that names the required benchmarks, the tier of each one and the thresholds to apply.
Tier	Critical, supporting or advisory. Critical and supporting both block a release; advisory never does.
max_drop	The largest absolute regression tolerated on a 0-1 scale; 0.01 is one percentage point.
INSUFFICIENT_EVIDENCE	The result when there are fewer than 10 paired items or results.jsonl is missing.

What nel gate needs

Two result directories and a policy YAML. You pass the baseline and candidate result directories. The policy says which benchmarks are required, the tier of each and its thresholds. Benchmark names must match exactly, because matching is case-sensitive. A Duplicate Benchmark Error usually means stale results from earlier runs are still in the directory.

EXAM TRAP

A benchmark spelled `MMLU` in the policy will not match `mmlu` in the results. Name matching is case-sensitive.

Paired per-item data

Each benchmark is judged on paired per-item results. The gate loads `results.jsonl`, works out the primary-metric difference per pair and puts a **95%** confidence interval around it. With fewer than **10** paired items, or no `results.jsonl`, the benchmark gets INSUFFICIENT_EVIDENCE.

EXAM TRAP

Summary scores alone are not enough. Without per-problem `results.jsonl` data, the gate can only report INSUFFICIENT_EVIDENCE.

Benchmark tiers

Only advisory benchmarks never block a release. Critical and supporting benchmarks both have hard caps. They differ only in their threshold value. Advisory benchmarks show in the report but never change the aggregate verdict.

How each tier affects the release verdict

Feature	Critical	Supporting	Advisory
Can block a release	Yes, hard cap	Yes, hard cap	No

Feature	Critical	Supporting	Advisory
BREACH or MISSING leads to	NO-GO	NO-GO	No effect on the verdict

EXAM TRAP

Supporting does not mean non-blocking. It differs from critical only in its threshold; use advisory for a benchmark that must never block.

Thresholds and direction

`max_drop` **is an absolute drop on a 0-1 scale.** A `max_drop` of **0.01** tolerates one percentage point of regression. `direction` defaults to `higher_is_better`, so any drop counts as a regression. Pick `lower_is_better` for perplexity or loss. `relative_guard_below` turns on the relative guardrail only when the baseline is below its value.

EXAM TRAP

A `max_drop` of 0.01 is not one percent of the baseline score. It is an absolute drop of one percentage point on the 0-1 scale.

Verdicts in CI

Add `--strict` **to fail a pipeline on NO-GO or INCONCLUSIVE.** NO-GO means a critical or supporting benchmark has a BREACH or is MISSING. With `--strict`, the command exits **1** on NO-GO and **2** on INCONCLUSIVE, so a CI or release script can stop the release.

Gate versus compare

nel compare investigates; nel gate decides. nel compare runs McNemar's significance test to ask whether a regression is statistically real. nel gate applies thresholds from the policy to make the release decision.

EXAM TRAP

nel compare is not the release gate. Its McNemar test is an investigation tool, while nel gate makes the go or no-go call.

Key takeaways

- Critical and supporting benchmarks both block; only advisory benchmarks never affect the verdict.
- Without `results.jsonl`, or with fewer than **10** paired items, a benchmark gets INSUFFICIENT_EVIDENCE.
- A `max_drop` of **0.01** means one percentage point on the 0-1 scale.
- With `--strict`, nel gate exits **1** on NO-GO and **2** on INCONCLUSIVE.

Sources: Implementing Quality Gates | NVIDIA NeMo Evaluator (opens in a new tab)

Enable real-time access and reasoning over structured and unstructured knowledge.

Start here

THINK FIRST

How would you let an agent answer from your own documents in real time, and when is the simplest setup enough?

Why it matters: Questions describe a knowledge source and ask how NeMo Guardrails retrieves it or which retrieval option fits.

NeMo Guardrails can ground bot responses in documents through RAG. It embeds the user query and places the retrieved chunks in the context as `$relevant_chunks`. You supply knowledge through a kb folder, a custom action or direct context.

Key terms

Term	Meaning
kb folder	The folder inside the configuration folder that holds Markdown knowledge base documents.
$relevant_chunks	The context variable that holds the retrieved knowledge base chunks.
Topic chunk	A piece of a document split at Markdown headers, then at blank lines if it is large.
retrieve_relevant_chunks	A custom action that fetches chunks from external sources, such as a vector database.

The kb folder

Markdown files in the `kb` folder enable a knowledge base. The folder sits inside the configuration folder, and the feature turns on by itself once documents are there. Only Markdown is supported for now. It is the simplest option for a static set of documents.

EXAM TRAP

PDF or Word files are not an option for the kb folder. Only Markdown documents are supported.

How retrieval works

Documents are chunked, and the query is embedded. Documents split into topic chunks at Markdown headers, and large chunks split again at blank lines. At query time, the user query is embedded with the model set in the `models` section of `config.yml`. Retrieved chunks appear in the context as `$relevant_chunks`.

Beyond static files

Use a custom action to reach external sources. A custom `retrieve_relevant_chunks` action can pull chunks from an external vector database. You can also pass context straight into a generate call, using a message with the `context` role.

Ways to supply knowledge to NeMo Guardrails

Feature	kb folder	Custom action	Direct context
Source	Markdown files in the configuration folder	External sources such as a vector database	Context sent with the generate call
How it is used	Enabled automatically when documents are present	Implemented as retrieve_relevant_chunks	A message with the context role

EXAM TRAP

The kb folder is not the only route. A custom `retrieve_relevant_chunks` action can fetch chunks from an external vector database.

Reasoning over knowledge

NIM supports apps that reason over images, audio, video and text. Its inference engine, vLLM or SGLang, runs the model and offers OpenAI-compatible endpoints. In NeMo Agent Toolkit, a reasoning agent needs an LLM that supports reasoning, and it plans ahead instead of reasoning between steps.

Key takeaways

- Markdown files in the `kb` folder enable the knowledge base automatically.
- Retrieved chunks reach the context as `$relevant_chunks`.
- Queries are embedded with the embedding model from the `models` section of `config.yml`.
- Use a custom `retrieve_relevant_chunks` action for an external vector database.

Sources: Knowledge Base Configuration | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · NVIDIA NIM for LLM and VLM Documentation — NVIDIA NIM for LLM and VLM (opens in a new tab) · About Reasoning Agent — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Domain 7 | 7% of the exam | 19 min read

NVIDIA Platform Implementation

Overview

Every point below comes from NVIDIA's official documentation; each section lists its sources.

Integrate NVIDIA NeMo Guardrails for compliance and safety enforcement.

Start here

THINK FIRST

How do you decide which NeMo Guardrails rails a safety need calls for, and where does each one go in the configuration?

Why it matters: Questions describe a safety or compliance need and ask which rail, setting, or configuration file meets it.

A guardrails configuration lives in one config folder. The file `config.yml` names the models and switches rails on, while Colang flows and Python actions implement them. Each rail type runs at its own point in a conversation.

Key terms

Term	Meaning
config.yml	The core file: which LLMs to use, general instructions, a sample conversation, and which rails are active.
Colang flow	A rail written in a `.co` file; such files load from anywhere inside the config folder.
Custom action	A Python function marked with `@action()` that Colang flows can call while handling a request.
Input rail	A rail that runs when user input arrives, to validate, filter, or change it.

Files in the config folder

Every guardrails component sits in one config folder. The file `config.yml` holds core options such as the LLMs, general instructions, and active rails. Colang `.co` files load recursively from anywhere in the folder. The file `config.py` registers custom LLM and embedding providers, plus shared resources, at startup. With the files ready, the SDK classes `RailsConfig` and `LLMRails` run the guardrails.

When each rail type runs

Rail types differ by the moment they trigger. Catalog guardrails are configurable flows you place in the input, output, or retrieval section of `config.yml`. Pick the rail type by when the check must happen.

Rail types and when they run

Rail type	Runs when	Used to
Input rails	User input arrives	Validate, filter, or change the input
Retrieval rails	RAG retrieval has finished	Process the retrieved chunks
Execution rails	Before and after an action runs	Control tool and action calls
Tool rails (IORails engine)	A model emits a tool call, or a tool result returns	`tool_output` checks the call; `tool_input` checks the result

EXAM TRAP

The names mislead: `tool_output` rails do not check tool results. They check the tool calls the model emits, and `tool_input` rails check the results sent back to the model.

Defining models for rails

Declare every model in the `models` section first. Input and output flows can use a model only after it is defined there. A flow points to one with `$model=<type>`, and the runtime checks that each reference has a matching model. For content safety, `$model` names the model that performs the check. The type `jailbreak_detection` is reserved for NIM-based jailbreak detection.

Content safety rails

Content safety rails send input and output to a safety model. The input rail runs the `content_safety_check_input` action, and the output rail runs `content_safety_check_output`. Usable models include Nemotron Content Safety, Llama Guard **3**, and ShieldGemma. A missing prompt raises an exception when the configuration loads. When the prompt task sets no `max_tokens`, the action uses **1024** tokens.

Built-in jailbreak and injection checks

Each built-in check works in its own way. Self check input asks an LLM, and the jailbreak flows use a NIM model or heuristics. Injection detection checks LLM output, and its action can be set to `omit` instead of `reject`.

Built-in safety flows compared

Flow	Checks	How it works
self check input	User input	An LLM checks the input against a policy
jailbreak detection model	User input	A NIM model classifies it; endpoint path defaults to `classify`

Flow	Checks	How it works
jailbreak detection heuristics	User input	Heuristics instead of a model
injection detection	LLM output	Finds SQL, XSS, code, and template injection; action defaults to `reject`

EXAM TRAP

The heuristics flow does not call a NIM model. It uses heuristics instead; the jailbreak detection model flow is the NIM-based one.

Masking sensitive data

PII rails detect sensitive data and mask it. With Presidio, the default mask token is `*` and the default detection score threshold is **0.2**. You set both separately for input, output, and retrieval.

Parallel input and streaming output

Input rails do not run in parallel by default. The setting `rails.input.parallel` defaults to `false`. For output streaming, `stream_first` defaults to `true`, so tokens stream before output rails apply. Chunks default to **200** tokens, carrying **50** tokens of context from the previous chunk.

EXAM TRAP

Streamed output is not held until output rails finish by default. Because `stream_first` is `true`, tokens reach the client before the rails check them.

Key takeaways

- Define a model in the `models` section before any flow refers to it with `$model`.
- Input rails run on arrival, retrieval rails after RAG retrieval, and execution rails around actions.
- On the IORails engine, `tool_output` checks the model's tool calls and `tool_input` checks tool results.
- Output streaming sends tokens before output rails by default, in chunks of **200** tokens.

Sources: NeMo Guardrails Library Configuration Overview | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · Configuration YAML Schema Reference | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · About Configuring Guardrails | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · Guardrail Catalog | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · Content Safety | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab)

Deploy NVIDIA NIM microservices for high-performance inference.

Start here

THINK FIRST

What must you get right when deploying NIM on Kubernetes, from profile selection through multi-node scaling?

Why it matters: Questions describe a NIM deployment and ask which profile, Helm value, health check, or operator setting makes it work.

At startup NIM picks one model profile, which decides the files it downloads and how the engine launches. On Kubernetes you deploy it with the Helm chart or the NIM Operator. Multi-node deployments add Ray, tensor- and pipeline-parallel sizes, and ReadWriteMany storage when persistence is on.

Key terms

Term	Meaning
Model profile	A pre-validated inference configuration that NIM selects when the container starts.
NIMCache	Operator resource that caches model files on persistent storage, so restarts and scaling reuse them.
NIMService	Operator resource that runs the NIM deployment: pods, health probes, service exposure, and GPU scheduling.
Readiness endpoint	The path `/v1/health/ready`, which returns HTTP **200** only once the model is fully loaded.

How NIM picks a profile

NIM selects exactly one profile when the container starts. Without `NIM_MODEL_PROFILE`, NIM picks the best compatible profile for your GPU, free VRAM, memory needs, and parallelism limits. A profile description names the backend, `vllm` or `sglang`, and the precision: `bf16`, `fp8`, `mxfp4`, or `nvfp4`.

EXAM TRAP

A profile's parallelism can be overridden. Setting `NIM_TENSOR_PARALLEL_SIZE=4` on a profile with `tp=2` launches the backend with `TP=4`.

Checking a deployment

Use `/v1/health/ready`, not `/v1/health/live`, to judge readiness. The live endpoint answers at once, even while the model loads. The ready endpoint returns **200** only after loading finishes. The Operator's startup probe on it allows **120** failures **10** seconds apart, up to **20** minutes. A dry run selects the profile and downloads the model without starting the server.

EXAM TRAP

A **200** from `/v1/health/live` does not mean the model can serve. Only the ready endpoint waits for the model to finish loading.

Helm chart or NIM Operator

Both deploy NIM on Kubernetes; the Operator adds custom resources. The Operator uses NIMCache to cache model files and NIMService to run the deployment. A failed NIMCache download job retries up to **5** times, the default `backoffLimit`.

Helm chart and NIM Operator compared

Feature	Helm chart	NIM Operator
Minimum version	Kubernetes **v1.26** or later with GPU nodes	NIM Operator **3.0.2** or later
What it creates	A StatefulSet by default; a Deployment if `statefulSet.enabled` is false	NIMCache and NIMService custom resources
Extra arguments on multi-node	`customArgs` reach the leader container only	Avoid `spec.args`; it reaches neither leader nor workers

Multi-node deployments

Ray forms the cluster; vLLM or SGLang runs the model across it. The usual setup sets tensor-parallel size to the GPUs per node and pipeline-parallel size to the node count. Helm needs both sizes above **0**, or `model.profile`, or the chart fails to render. Ray's port, **6379** by default, must be reachable between pods. Turn off JSON logging with `model.jsonLogging=false`.

EXAM TRAP

ReadWriteOnce storage fails for multi-node NIM with persistence on. The volume must mount on several nodes, so use ReadWriteMany.

Scaling and scheduling

Autoscale on inference metrics, not CPU or memory. Plain CPU and memory figures are poor signals for scaling NIM, so point HPA at inference-specific metrics. Give NIM pods a priority class, or they rank lowest and can be preempted. When upgrades leave no room for an extra GPU pod, use Recreate instead of the default RollingUpdate.

Tuning memory, caching, and network

Lower `--max-model-len` to cap tokens per request. It limits the combined input and output tokens of each request. On the SGLang backend, prefix caching (radix cache) is already on, so you add no flag. Across nodes, tensor parallelism relies on network bandwidth, latency, and Remote Direct Memory Access (RDMA) availability.

Key takeaways

- Without `NIM_MODEL_PROFILE`, NIM picks the best compatible profile for the hardware.
- Only `/v1/health/ready` returning **200** shows the model is fully loaded.
- Multi-node NIM needs ReadWriteMany storage when persistence is on, an open Ray port, and JSON logging off.
- Scale NIM on inference-specific metrics rather than CPU or memory.

Sources: Model Profiles and Selection — NVIDIA NIM for LLM and VLM (opens in a new tab) · Kubernetes Deployment — NVIDIA NIM for LLM and VLM (opens in a new tab) · Multi-Node Deployment — NVIDIA NIM for LLM and VLM (opens in a new tab) · NIM Operator Deployment — NVIDIA NIM for LLM and VLM (opens in a new tab) · Helm and Kubernetes — NVIDIA NIM for LLM and VLM (opens in a new tab) · Multi-Node Deployment — NVIDIA NIM for LLM and VLM (opens in a new tab)

Optimize workflows with the NVIDIA NeMo Agent Toolkit.

Start here

THINK FIRST

When a NeMo Agent Toolkit workflow is slow or costly, how do you find the bottleneck and then tune it?

Why it matters: Questions describe a workflow to measure or tune and ask which profiler output, optimizer setting, or sizing step fits.

The toolkit gives you three tools for optimizing. The profiler reports latency and finds bottlenecks, and the optimizer searches parameters and prompts for better scores. The sizing calculator estimates the GPU cluster a target load needs.

Key terms

Term	Meaning
Profiler	The measuring tool from the `nvidia-nat-profiler` package, run through `nat eval`.
Optimizer	An optional tool, run with `nat optimize`, that tunes numbers with Optuna and prompts with a genetic algorithm.
Parameter set	One combination of values the optimizer evaluates, run **3** times by default for a stable score.
Sizing calculator	A tool that estimates GPU cluster size for a target number of users and response time.

Profiling a workflow

Run the profiler through `nat eval`. Callback handlers cover LangChain/LangGraph, LlamaIndex, CrewAI, Google ADK, and Semantic Kernel. For a registered function, list its frameworks in the `framework_wrappers` argument of `register_function`. For your own Python code, `@track_function` records start, end, inputs, and outputs, even for async functions and generators.

Reading profiler results

`inference_optimization.json` **holds confidence intervals.** It gives **90%**, **95%**, and **99%** intervals for latency, throughput, and workflow runtime. The `prompt_caching_prefixes` option finds repeated prompt prefixes worth pre-loading into KV caches. Nested stack bottleneck analysis finds tool calls inside other tool calls; `simple_stack` gives a simpler view.

Spotting concurrency spikes

A spike threshold counts functions running at the same time. With `spike_threshold` set to **7**, concurrency spike analysis flags every moment when **7** or more functions are running at once. Lower the value to catch smaller bursts of parallel work.

EXAM TRAP

A threshold of **7** also flags exactly **7** concurrent functions. The test is greater than or equal to the threshold, not strictly greater.

What the optimizer tunes

Optuna tunes numbers; a genetic algorithm tunes prompts. Install the optimizer first, for example with `pip install nvidia-nat[config-optimizer]`, then run `nat optimize`. Your configuration object must inherit from `OptimizableMixin` to get the `optimizable_params` field. GA prompt optimization stays off unless `prompt.enabled` is set.

Numeric and prompt optimization compared

Feature	Numeric values	Prompts
Method	Optuna	Custom genetic algorithm (GA)
Default effort	**20** trials (`numeric.n_trials`)	Population of **10** over **5** generations

EXAM TRAP

Prompts are not tuned unless you turn it on. The setting `prompt.enabled` defaults to `false`.

Scoring trials

Each metric is maximized or minimized, then scores are combined. Metric weights default to **1.0** and are normalized. Combination modes are harmonic, sum, and chebyshev, with harmonic as the default. Each parameter set runs **3** times by default to smooth noisy results. With the sampler unset or `bayesian`, Optuna uses TPE for one objective and NSGA-II for several; `grid` is the alternative.

Sizing the GPU cluster

Gather metrics first, then size the cluster offline. The calculator first collects metrics at several concurrency levels, then sizes the cluster in `offline_mode` from them. Aim for at least **10** concurrency values; fewer still work, but accuracy may drop. The dataset needs inputs only, with no ground truth answers.

EXAM TRAP

Sizing does not need labeled data. The calculator's dataset needs only the inputs, not ground truth answers.

Running evaluations smoothly

Lower `eval.general.max_concurrency` when `nat eval` hits rate limits. On **429** Too Many Requests errors, reduce the concurrency, for example to **1** with a command-line override. Each run saves `config_effective.yml`, the final configuration with every override applied. Out of the box, `nat eval` reads json, jsonl, csv, xls, and parquet datasets, or a custom type.

Key takeaways

- Profile with `nat eval`; tune with `nat optimize` after installing the optional optimizer.
- Optuna tunes numeric values and a custom genetic algorithm tunes prompts, which is off by default.
- Each parameter set runs **3** times by default, and harmonic is the default way to combine scores.
- The sizing calculator works best with **10** or more concurrency values and needs no ground truth answers.

Sources: Profiling and Performance Monitoring of NVIDIA NeMo Agent Toolkit Workflows — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Profiling and Performance Monitoring of NVIDIA NeMo Agent Toolkit Workflows — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · NVIDIA NeMo Agent Toolkit Optimizer — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Size a GPU Cluster With NVIDIA NeMo Agent Toolkit — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · NVIDIA NeMo Agent Toolkit Optimizer — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Size a GPU Cluster With NVIDIA NeMo Agent Toolkit — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Agent Evaluation in NVIDIA NeMo Agent Toolkit — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Leverage NVIDIA TensorRT-LLM and Triton Inference Server for latency reduction.

Start here

THINK FIRST

How can an agent workflow mark its latency-critical work and return results without making clients wait or time out?

Why it matters: Questions describe a latency or timeout problem and ask which decorator, endpoint, or server setting addresses it.

Cutting latency in an agent workflow happens at two levels. Inside the code, functions declare how latency-sensitive they are, so routing can send urgent requests to fast backends. At the server, the endpoint you call decides whether a client waits, streams, or polls.

Key terms

Term	Meaning
Latency sensitivity	An integer level, such as **1** for low up to **3** for high, marking how strict a function's latency needs are.
@latency_sensitive	A decorator that tags a function with an integer latency sensitivity level.
Async generate	An endpoint that runs a workflow in the background and returns a job identifier at once.
Dask	The library the toolkit server uses to manage asynchronous jobs.

Marking latency-sensitive functions

Tag a function with `@latency_sensitive` and an integer level. Levels are integers, for example **1** for low, **2** for medium, and **3** for high. While the function runs, its level sits on the context stack. Code reads the current value through `Context.get().latency_sensitivity`, with `Context` imported from `nat.builder.context`.

How nested levels combine

The highest level in the call stack wins. Sensitivity flows down nested calls with max-based merging. So a level-**1** function called from inside a level-**3** function still runs at level **3**. One use is LLM routing, which sends high-sensitivity requests to low-latency backends.

> **EXAM TRAP**
> An inner function cannot lower the level it inherits. The effective value is the maximum of all pushed levels, not the most recently pushed one.

Choosing a workflow endpoint

Pick the transaction by how long the client can wait. A running toolkit server offers **5** workflow transactions over HTTP or WebSocket. The three generate options differ in when the client gets its data.

Generate options compared

Feature	Non-streaming	Streaming	Async
Client receives	The result once all workflow data is ready	Chunks as the workflow produces them	A job identifier at once; results later
Route or requirement	`/v1/workflow`	`/v1/workflow/stream`	Needs the `async_endpoints` extra installed

Async jobs and timeouts

Async generate keeps long workflows from timing out clients. The call hands back a unique job identifier for checking status and results later. Its `sync_timeout` defaults to **0**, so it returns immediately; the maximum is **300** seconds. Jobs run on Dask, which starts a local cluster unless `scheduler_address` points to an existing scheduler.

EXAM TRAP

The `--max_concurrent_jobs` flag is not a direct job limit. It sets the number of Dask workers, **10** by default, and applies only when `scheduler_address` is unset.

Trimming the response

Set `filter_steps` to `none` to get only the final output. On the `/v1/workflow/full` route, `filter_steps` takes a comma-separated list of step types to filter by. The value `none` drops every intermediate step. Separately, `disable_legacy_routes` set to `true` drops the legacy endpoint paths entirely.

Serving tools over MCP

`nat mcp serve` **publishes workflow tools on port 9901.** By default it binds to localhost, publishes every workflow tool, and serves them at the `/mcp` path over streamable-http. A custom `base_path` needs streamable-http, since SSE cannot use custom base paths. SSE has no authentication either, so production servers should run streamable-http with authentication configured.

EXAM TRAP

SSE is a poor production choice for an MCP server. It supports neither authentication nor a custom base path.

Key takeaways

- The effective latency sensitivity is the highest level pushed on the context stack.
- Async generate returns a job identifier at once; `sync_timeout` defaults to **0** with a maximum of **300**.
- Streaming sends chunks as they appear, while non-streaming waits for the complete result.
- Use streamable-http, not SSE, for MCP servers that need authentication or a custom base path.

Sources: nat.plugins.profiler.decorators.latency — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · NVIDIA NeMo Agent Toolkit API Server Endpoints — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · NeMo Agent Toolkit as an MCP Server — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Manage and optimize multimodal input pipelines on NVIDIA hardware.

Start here

THINK FIRST

How do you get image, audio, or video into a model served by NIM, and when does a well-formed request still fail?

Why it matters: Questions describe a multimodal request or container and ask which engine, content part, or format makes it work.

NIM passes media to multimodal models through ordinary Chat Completions requests whose content mixes text and media parts. Whether a request works depends on the model and on the engine inside the image. vLLM handles image, audio, and video; SGLang handles image and audio only.

Key terms

Term	Meaning
Content array	The list inside a chat message that mixes text parts with image, audio, or video parts.
Inference engine	The serving backend packed into a NIM image: either vLLM or SGLang.
Data URL	A string that carries base64-encoded media bytes inline, with the MIME type at the front.
Multimodal rail	A NeMo Guardrails rail that considers more than text, such as voice, gestures, posture, or images.

Requests carry mixed content

Media rides in a Chat Completions content array. NIM LLM and VLM can serve models that take image, audio, or video input. You reach them with OpenAI-compatible Chat Completions requests whose content array mixes text items with media items. Check the model first: a text-only model cannot process any of the three media types.

Which engine handles which media

Video needs vLLM; image and audio work on both engines. Every container image ships with one engine, vLLM or SGLang. The SGLang images also leave out the video decoder that `video_url` requests depend on.

Media support by inference engine

Feature	vLLM image	SGLang image
Image input	Supported	Supported
Audio input (`audio_url`)	Supported	Supported
Video input (`video_url`)	Supported	Not supported: no video decoder in the image

EXAM TRAP

Video on an SGLang image is not fixed by a request flag. The image has no video decoder, so video input needs a vLLM image.

Finding the engine in a container

Startup logs or the chosen profile's description name the engine. Before you send video, confirm which engine the running container uses. Read the startup logs, or look at the description of the profile it selected. That check stops you from sending `video_url` content to an SGLang image.

Audio and video content parts

Audio uses `audio_url` or `input_audio`; video uses `video_url`. The `audio_url` and `video_url` parts are extensions to the OpenAI Chat Completions schema. Both engines accept `audio_url`. Both also accept the standard OpenAI `input_audio` part for inline audio. A `video_url` part still needs a vLLM image.

Sending images inline

Put a base64 data URL in `image_url.url`. To send image bytes instead of a link, encode them as base64 and set `image_url.url` to a data URL, for example one starting ``. The MIME type in that URL should match the file, so a PNG file takes `image/png`.

EXAM TRAP

Labeling a PNG as `image/jpeg` in the data URL is a mismatch. The MIME type must name the file's real format.

Multimodal rails in NeMo Guardrails

Rails can act on voice, gestures, posture, and images too. Multimodal rails take several input and output modalities into account. The Colang Standard Library has an avatars module whose flows cover multimodal events and actions. For example, the user gestured flow waits for `GestureUserActionFinished` events, and the bot gesture flow triggers `StartGestureBotAction` events.

Key takeaways

- Video input needs a vLLM image, because SGLang images lack the video decoder.
- Media travels next to text in an OpenAI-compatible Chat Completions content array.
- Inline images use a base64 data URL in `image_url.url` with a MIME type that matches the file.
- A text-only model cannot take image, audio, or video, whatever engine serves it.

Sources: Image, Audio, and Video Input — NVIDIA NIM for LLM and VLM (opens in a new tab) · Multimodal Rails | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab)

Domain 8 | 7% of the exam | 14 min read

Run, Monitor, and Maintain

Overview

Every point below comes from NVIDIA's official documentation; each section lists its sources.

Define monitoring dashboards and reliability metrics.

Start here

THINK FIRST

Which signals would you put on a dashboard for an agent served by NIM, and where does each one come from?

Why it matters: Questions describe a monitoring need and ask which endpoint, log setting, header or metric class provides it.

NIM serves its engine's own Prometheus metrics at `/v1/metrics`, while logs and request headers fill in the story. NeMo Agent Toolkit adds per-user and per-call metrics for the agent layer.

Key terms

Term	Meaning
Prometheus metrics	Metrics in a Prometheus-compatible format, exposed by NIM at `/v1/metrics`.
Native engine metrics	The inference backend's own metrics, which NIM passes through unchanged.
JSONL logging	One JSON object per log line, turned on with `NIM_JSONL_LOGGING=true`.
Correlation ID	An `X-Request-Id` value you send; NIM returns it on the response and passes it to the backend.
traceparent	The trace header that NIM's proxy layer always forwards to the backend.

The metrics endpoint

NIM exposes Prometheus metrics at `/v1/metrics`. These are the inference backend's own metrics, passed through unchanged. They report latency, throughput, token counts, queue depth and GPU utilization.

Dashboards are engine-specific

Dashboards built for one engine do not transfer. NIM does not wrap or rename the engine's metrics. A dashboard or alert rule made for the matching engine works unchanged, but it will not fit the other engine, vLLM or SGLang.

EXAM TRAP

NIM does not give every engine one common metric set. Dashboards and alerting rules do not carry over when you switch engines.

Log level and format

NIM writes its logs to stderr. `NIM_LOG_LEVEL` sets verbosity with standard Python levels. For collectors such as Fluentd, Logstash or CloudWatch, set `NIM_JSONL_LOGGING=true` to get one JSON object per line.

Two environment variables that control NIM logs

Setting	What it controls	Values
`NIM_LOG_LEVEL`	Log verbosity	DEBUG, INFO, WARNING, ERROR or CRITICAL
`NIM_JSONL_LOGGING`	Log format for NIM and the backend	`true` gives one JSON object per line

EXAM TRAP

JSONL logging is not limited to NIM's own messages. It formats the vLLM or SGLang backend's logs the same way.

Tracing a request

Send `X-Request-Id` to correlate a request. NIM returns the ID on the response and passes it on to the backend. If you omit it, NIM adds none. NIM also always forwards `traceparent`, but spans appear only when the engine has tracing on.

EXAM TRAP

Forwarding traceparent is not enough on its own. The inference engine must have tracing enabled before it creates spans.

Metrics from LLM call logs

`LLMMetrics` **derives metrics from LLM call logs.** This NeMo Agent Toolkit class works on standardized call logs. Its `compute_profiling_metrics` method returns the same DataFrame with **six** NOVA- columns added. The ATIF `Metrics` model records one step's LLM operational and confidence data.

Per-user monitoring

Track latency and errors for each user. `PerUserMetricsCollector` gathers metrics from SessionManager's per-user builders, for one user or all users. `PerUserRequestMetrics` has fields such as `avg_latency_ms` and `error_count`. The `/monitor/users` endpoint returns `PerUserMonitorResponse`.

Key takeaways

- Dashboards read NIM's Prometheus metrics at `/v1/metrics`, passed through from the engine unchanged.
- Engine metrics keep their native names, so dashboards do not carry over between engines.
- Set `NIM_JSONL_LOGGING=true` when a production log collector ingests NIM logs.
- Spans need tracing enabled on the engine, even though NIM always forwards traceparent.

Sources: Logging and Observability — NVIDIA NIM for LLM and VLM (opens in a new tab) · nat.runtime.metrics — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.plugins.profiler.inference_optimization.llm_metrics — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.atif.metrics — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Track logs, errors, and anomalies for root cause diagnosis.

Start here

THINK FIRST

When an agent call fails, how do you find which part of the stack broke and why?

Why it matters: Questions describe a failing agent and ask which error, log or tracking tool points to the root cause.

Root cause diagnosis starts with a precise signal. NeMo Guardrails turns outbound HTTP failures into specific exception types. NeMo Agent Toolkit decorators record what each function call did.

Key terms

Term	Meaning
HTTPClientError	The base class every outbound HTTP failure in NeMo Guardrails inherits from.
retry_count	Attribute on HTTPClientError giving the retries that finished before the caller saw the error.
track_function	Agent Toolkit decorator that runs tracking logic around sync, async and generator functions.
NAT Event Stream	The Agent Toolkit stream that receives intermediate steps from the profiler.

One family of HTTP errors

All outbound HTTP failures share one base class. In NeMo Guardrails, `HTTPClientError` sits at the top. The exceptions are transport-neutral, so your handler does not depend on the HTTP library underneath. Catch the base class to log every failure in one place.

Match the error to the failure

Each subclass names a different failure point. Read the exception type first. It tells you whether the request never connected, ran out of time, got an error status or returned an unreadable body.

NeMo Guardrails HTTP exceptions and what each one means

Exception	Raised when	What failed
HTTPConnectionError	The transport cannot open or keep a connection	The link to the server
HTTPTimeoutError	The request runs past its total timeout	The time budget
HTTPStatusError	The server answers with an unsuccessful status	The request, as judged by the server
HTTPResponseDecodeError	The response body is not valid JSON	Parsing the reply

EXAM TRAP

A decode error does not mean the server was unreachable. A reply arrived; its body just could not be decoded as JSON.

Retries before the error

`retry_count` **shows how many retries finished first.** Log it together with the exception. A value above **0** means the caller saw the error only after retries had already run.

Logs from the LLM tracker

`llm_tracker` **logs through a standard Python logger.** The module creates it with `logging.getLogger(__name__)`, so it is an ordinary named Python logger. Your existing Python logging setup applies to it.

Tracking any function

`track_function` **wraps any function type with tracking logic.** It handles **4** kinds of function: sync, async, generator and async generator. The tracking logic runs around each call of the wrapped function.

Unregistered functions

`track_unregistered_function` **adds scope management and automatic tracking.** It relies on `Context.push_active_function`, the toolkit's built-in tracking, which avoids duplicate tracking entries. Pass `name` to replace `func.__name__` in the records, and `metadata` to attach extra data.

EXAM TRAP

The `name` argument does not rename the function. It only sets the name used for tracking in place of `func.__name__`.

Intermediate steps

`push_intermediate_step` **sends a step to the NAT Event Stream.** This is how the Agent Toolkit profiler records an intermediate step of a run, alongside the function tracking above.

Key takeaways

- Catch `HTTPClientError` to handle every outbound HTTP failure in NeMo Guardrails.
- Connection, timeout, status and decode errors each point to a different failure.
- `retry_count` tells you how many retries ran before the error reached the caller.
- `track_unregistered_function` uses built-in tracking to avoid duplicate entries.

Sources: nemoguardrails.http.errors | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · nat.plugins.profiler.decorators.function_tracking — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nemoguardrails.logging.llm_tracker | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab)

Continuously benchmark deployed agents against prior versions.

Start here

THINK FIRST

How do you prove a new agent version is really faster or cheaper than the version before it?

Why it matters: Questions describe a benchmark run and ask which setting, record or comparison makes the result trustworthy.

A throughput figure is meaningless unless you know the configuration that produced it. AIPerf measures the served model, and its output plus the server log show exactly what was measured.

Key terms

Term	Meaning
AIPerf	NVIDIA's preferred tool for benchmarking LLM applications served by NIM.
Resolved configuration line	Server log line listing the arguments NIM actually resolved; hidden at the default log level.
Trace	The sequence of requests a traced workload sends; sequential sampling walks it in order and wraps.
cache_salt	A per-run value that stops the prefix cache from carrying results from one run to the next.

Term	Meaning
Pareto curve	The throughput and latency pairs you get by sweeping concurrency.

Benchmarking versus load testing

AIPerf measures the model; load tools measure scaling. Load testing tools such as K6 send many concurrent requests to see how a deployment scales. Performance benchmarking with AIPerf measures the model's own throughput, latency and token-level metrics.

Load testing and performance benchmarking compared

Feature	Load testing	Performance benchmarking
Example tool	K6	AIPerf
What it tells you	How the deployment copes with many concurrent requests	Throughput, latency and token-level metrics of the model

Record the configuration first

A throughput number needs the configuration behind it. Before collecting data, record the inference engine and the resolved arguments. Start the NIM with `NIM_LOG_LEVEL=INFO`, because the default level hides the `Resolved configuration:` line.

EXAM TRAP

Do not rely on the default log level to show the server's arguments. It suppresses the `Resolved configuration:` line; set `NIM_LOG_LEVEL=INFO` instead.

Keep the evidence

Keep `input_config` and the server log line with your numbers. The `input_config` object in AIPerf's output shows the configuration AIPerf accepted. Paired with the `Resolved configuration:` line, together they are the authoritative record of the measured setup.

Match the model ID

The `-m` value must match the served model ID exactly. Query the server's `/v1/models` endpoint for the ID rather than typing the name from memory.

Same requests every run

Use `--dataset-sampling-strategy sequential` for repeatable runs. It walks the trace in order and wraps around, so each run at one concurrency sees the same request order. Add a fresh `cache_salt` per run so the prefix cache cannot inflate later sweep points.

EXAM TRAP

Do not use a base64 salt. It can emit `/` and `+`, so use a hex salt instead.

Never mix workload types

Never compare a synthetic point with a traced one. Synthetic and traced workloads are both valid, yet each answers a different question. With a traced workload, expect the mean input length you measure to sit about **10%** or less from the trace's mean.

Pick the concurrency

Sweep concurrency, then read the Pareto curve. Every concurrency level gets a separate artifact directory. Put `output_token_throughput` on one axis and `inter_token_latency` on the other. Choose the concurrency with the highest throughput that still meets your latency budget.

Accuracy comes first

Cost numbers count only at acceptable accuracy. The NIM cost benchmarking method does not measure accuracy. NeMo Evaluator has **17** runtime-registered built-in benchmarks, such as HumanEval, whose `code_sandbox` scorer runs code in Docker with network isolation.

Key takeaways

- Record the engine and resolved arguments, with `NIM_LOG_LEVEL=INFO`, before collecting data.
- Use sequential sampling and a fresh hex `cache_salt` so runs stay comparable.
- Compare synthetic with synthetic and traced with traced, never across the two.
- Pick the concurrency with the highest throughput inside your latency budget.

Sources: Benchmarking — NVIDIA NIM for LLM and VLM (opens in a new tab) · Benchmarking — NVIDIA NIM for LLM and VLM (opens in a new tab) · Built-in Benchmarks | NVIDIA NeMo Evaluator (opens in a new tab)

Implement automated tuning, retraining, and versioning in production.

Start here

THINK FIRST

How can an agent keep improving in production without a risky manual retraining cycle?

Why it matters: Questions describe a production agent and ask how the finetuning harness, its rewards or its deployment safeguards should be set up.

The NeMo Agent Toolkit finetuning harness trains an agent with the same workflow it runs in production. Evaluators score each run to produce rewards. You evaluate model quality before enabling automatic deployment, then track performance in production and retrain as needed.

Key terms

Term	Meaning
Finetuning harness	Experimental Agent Toolkit feature for in-situ training; breaking changes may come without notice.
Trajectory	Training data the harness collects by running evaluation; it is kept only when it carries log probabilities.
On-policy	Training that needs fresh trajectories after each update, as GRPO and PPO do.
KL penalty	DPO setting where higher values give more conservative, slower but more stable updates.

Train where you run

The harness trains with your production workflow. In-situ training means no move to a separate development environment. Start it with `nat finetune` and a configuration file. The feature is experimental, so later releases may break your setup.

Three pluggable parts

TrajectoryBuilder, TrainerAdapter and Trainer can be mixed and customized. Rewards come from the Agent Toolkit evaluator system. That means the metrics you already evaluate with can double as training signals.

Log probabilities are required

The inference endpoint must return log probabilities. Any trajectory lacking logprobs is dropped before training. Check that your endpoint supports them before you start a run.

> **EXAM TRAP**
> An endpoint that returns only text is not enough. Trajectories need log probabilities, or the harness filters them out.

The on-policy loop

GRPO and PPO need fresh trajectories after each update. That is why the harness runs evaluation at the start of every epoch: it collects new trajectories. With curriculum learning, difficulty comes from the average reward on each example group.

Preference tuning with DPO

DPO pairs come from Test-Time Compute during workflow runs. Install the NeMo Customizer plugin with `pip install nvidia-nat-nemo-customizer`. Higher KL penalty values, **0.2-0.5**, make updates more conservative: slower, but more stable.

Split inference and training

OpenPipe ART separates inference from training. Its async client-server design lets inference run anywhere while training runs on GPU infrastructure.

Deploy with care

Evaluate a finetuned model before automatic deployment. Once it is live, track its performance in production and retrain when needed. Retraining becomes a loop rather than a one-off event.

EXAM TRAP

Do not enable automatic deployment first and test later. Evaluate model quality before turning it on.

Key takeaways

- The harness trains in situ with `nat finetune` and your production workflow.
- Trajectories without log probabilities are dropped, so the endpoint must return them.
- On-policy methods like GRPO and PPO regenerate trajectories every epoch.
- Test model quality before auto-deploying, then monitor and retrain as needed.

Sources: NVIDIA NeMo Agent Toolkit Finetuning Harness for Reinforcement Learning — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Finetuning Harness: Concepts and Architecture — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · DPO with NeMo Customizer — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · GRPO with OpenPipe ART — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Ensure continuous uptime, transparency, and trust in live deployments.

Start here

THINK FIRST

What does it take to keep a live agent available, explainable and trusted after it ships?

Why it matters: Questions describe a live agent deployment and ask which practice keeps it available, transparent or accountable.

After deployment, agents need ongoing operation, monitoring and maintenance. Uptime rests on dashboards and reliability metrics, while log tracking serves root cause diagnosis. Trust rests on transparency, audit trails and human oversight.

Key terms

Term	Meaning
Reliability metrics	Measures you define, together with monitoring dashboards, to watch a live agent.
Decision traceability	A transparency mechanism, alongside explainable reasoning, for following an agent's decisions.
Audit trail	A record that is designed and enforced along with system security.

Term	Meaning
Human oversight	People's ability to oversee an agent and intervene, for accountability and trust.

Operations after launch

Running an agent is ongoing work after deployment. The Run, Monitor, and Maintain domain covers operating, monitoring and maintaining agentic systems once they are live. It carries **7%** of the exam weight.

Practices for uptime and trust in the study guide

Goal	Practice	Guide objective
Uptime	Monitoring dashboards and reliability metrics	8.1
Diagnosis	Track logs, errors and anomalies for root causes	8.2
Regression checks	Benchmark deployed agents against prior versions	8.3
Security	Design and enforce security and audit trails	9.1
Transparency	Explainable reasoning and decision traceability	10.3
Accountability	Human oversight and intervention	10.4

Watching for uptime

Dashboards and reliability metrics keep live agents up. Tracking logs, errors and anomalies is a separate practice, for diagnosing root causes. Benchmark each deployed agent against its prior versions on an ongoing basis.

EXAM TRAP

Monitoring is not only a dashboard. The guide also lists tracking logs, errors and anomalies to diagnose root causes.

Making decisions visible

Transparency means explainable reasoning and decision traceability. These mechanisms let people follow how an agent reached a decision. Pair them with system security and audit trails that are both designed and enforced.

Keeping humans in the loop

Human oversight and intervention support accountability and trust. People must be able to oversee an agent and step in. The professional role calls for building resilient, secure and trustworthy agentic solutions.

EXAM TRAP

Trust is not only a technical metric. The guide ties accountability and trust to human oversight and intervention.

Key takeaways

- Uptime starts with monitoring dashboards and reliability metrics.
- Benchmark deployed agents continuously against their prior versions.
- Transparency comes from explainable reasoning and decision traceability.
- Human oversight and intervention support accountability and trust.

Sources: NVIDIA-Certified Professional: Agentic AI Exam Study Guide (opens in a new tab)

Domain 9 | 5% of the exam | 14 min read

Safety, Ethics, and Compliance

Overview

Every point below comes from NVIDIA's official documentation; each section lists its sources.

Design and enforce system security and audit trails.

Start here

THINK FIRST

How do you keep an agent's logs useful for audits without turning them into a security risk?

Why it matters: Questions describe an agent deployment and ask which control protects the system, its tools or its audit trail.

Agent security works in layers: limit what the agent can touch, add injection detection to strengthen other controls, and log with care. Logs can reveal system architecture or security measures, so scrub sensitive data before logging.

Key terms

Term	Meaning
Audit trail exposure	The risk that logs reveal system architecture or security measures.
RBAC	Role Based Access Control: limiting agent permissions to specific resources.
YARA rule	A rule that lists string patterns to match plus a Boolean expression for its logic.
Context bloat detection	A rail against context manipulation that screens user input and retrieved knowledge-base chunks.

Contain the agent

Limit permissions and isolate execution. Use RBAC so the agent reaches only specific resources, which guards against tool abuse. Run it in a sandbox such as WebAssembly or a container to protect system integrity. Prefer signed models from trusted providers to cut supply chain risk.

Detect injection

Injection detection is one layer of defense in depth. NeMo Guardrails looks for SQL injection, code injection, template injection and cross-site scripting. It is meant to strengthen other security controls in

agentic systems, not replace them. Turn on the SQL injection rule when LLM output feeds a database query.

YARA rules and actions

YARA rules define what injection detection matches. Each rule lists text or binary patterns plus a Boolean expression for its logic. When a rule fires, rejecting the output is the safest action for production. The alternative is to omit the triggering text.

EXAM TRAP

Omitting the triggering text is not the safest choice for production. Rejecting the whole output is.

Context bloat

Context bloat detection screens text the agent did not write. It covers both user input and the knowledge-base chunks an agent retrieves. Its size cap, `max_chars`, defaults to **5000** characters, which stops oversized input meant to exhaust the token budget.

Traces as audit records

Guardrails traces go to `./.traces/trace.jsonl` by default. The FileSystemAdapter writes interaction traces there. Agent Toolkit observability can also record LLM prompts and responses, and that capture raises security considerations of its own.

Protect the logs

Logs can expose the system they record. They may reveal system architecture or security measures. Scrub sensitive data before it is logged, set log retention policies, and use a SIEM for secure log management.

EXAM TRAP

Keeping logs longer is not automatically safer for audits. Holding them longer than necessary increases exposure risk.

Key takeaways

- Use RBAC and sandboxing to limit and isolate what an agent can do.
- In production, reject the output rather than omit text when injection is detected.
- Context bloat detection caps input at **5000** characters by default.
- Sanitize logs, limit their retention and manage them with a SIEM.

Sources: Security Considerations — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Agentic Security | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · nemoguardrails.tracing.adapters.filesystem | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab)

Integrate compliance guardrails (privacy, enterprise policy).

Start here

THINK FIRST

How would you add privacy and company policy checks to an agent without rewriting the agent itself?

Why it matters: Questions describe a compliance need and ask which rail, setting or failure behavior meets it.

In NeMo Guardrails, compliance checks are rail flows you switch on in config.yml. PII detection protects privacy, while PolicyAI applies your own tagged policies to user messages and bot replies.

Key terms

Term	Meaning
Rail flow	A configurable guardrail you place in the input, output or retrieval part of config.yml.
config.yml	The only required file of a guardrails configuration; it names the LLMs and the active rails.
PolicyAI	Policy-based content moderation where you define custom policies and manage them with tags.
Policy tag	The label that groups PolicyAI policies, chosen with POLICYAI_TAG_NAME (default prod).

Where guardrails live

Every guardrail is a rail flow in `config.yml`. Add each rail to the input, output or retrieval section. You can mix NVIDIA safety models, community models, self-check prompts and managed APIs for defense in depth.

Parts of a configuration

Only `config.yml` is required. Everything else is optional. Colang files load recursively from anywhere in the config folder, and custom actions suit jobs such as external API calls or validation logic.

What each part of a NeMo Guardrails configuration holds

Part	Required	What it holds
config.yml	Yes	LLMs, general instructions, active rails and rail options
Colang .co files	No	The rail flows, from the config root or any subfolder
actions.py or actions/	No	Python functions marked @action() that flows can call

Privacy and topic rails

PII detection rails detect and mask sensitive data. Use them to protect user privacy. To hold an agent to approved subjects, topic control rails keep conversations inside predefined subject boundaries.

PolicyAI on input and output

PolicyAI checks both user messages and bot replies. Each result also carries a severity from **0** (safe) to **3** (high severity).

PolicyAI as an input rail and as an output rail

Feature	Input rail	Output rail
What it checks	The user's message	The bot's response, before the user sees it
When a policy fails	The message is blocked	The response is swapped for a refusal message

EXAM TRAP

One failing policy is enough. An input message is blocked if any policy attached to the tag returns UNSAFE, not only when most do.

Setting up PolicyAI

PolicyAI needs `POLICYAI_API_KEY` set. With the key in place it is ready to use on both input and output, with no extra code. `POLICYAI_TAG_NAME` is optional and defaults to `prod`; set it to use different tags for staging and production.

When PolicyAI is down

An API error makes the PolicyAI action raise an exception. To choose fail-open or fail-closed behavior, catch that exception with a try-catch block around the action in your own custom flow.

EXAM TRAP

Do not expect PolicyAI to fall back quietly when its API fails. The action raises an exception unless your flow catches it.

Key takeaways

- Compliance rails go in the input, output or retrieval section of `config.yml`.
- PII detection rails detect and mask sensitive data to protect privacy.
- PolicyAI blocks input or replaces output when any policy in the tag fails.
- `POLICYAI_TAG_NAME` defaults to `prod`; change it for each environment.

Sources: Guardrail Catalog | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · PolicyAI Integration | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · NeMo Guardrails Library

Configuration Overview | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab)

Mitigate bias and toxicity in outputs.

Start here

THINK FIRST

Where does bias and toxicity mitigation fit among the safeguards an agentic AI professional is expected to apply?

Why it matters: Questions ask which safeguard or NVIDIA component helps keep an agent's outputs responsible and safe.

Bias and toxicity mitigation belongs to the Safety, Ethics, and Compliance domain. It sits beside other ethical safeguards and layered safety frameworks. NeMo Guardrails is the NVIDIA component for enforcing safety.

Key terms

Term	Meaning
Ethical safeguards	Protections such as bias detection, privacy preservation and guardrails.
NeMo Guardrails	The NVIDIA component integrated for compliance and safety enforcement.
Layered safety framework	Safety built from several layers, such as filters and escalation protocols.
Safety, Ethics, and Compliance	The domain on running agentic AI responsibly and ethically; it carries **5%** of the exam weight.

Where this objective sits

Objective 9.3 belongs to Safety, Ethics, and Compliance. That domain covers the principles and practices that keep agentic AI systems responsible and ethical. It carries **5%** of the exam weight.

Safeguards to know

Bias detection is one of the expected ethical safeguards. The other named examples are privacy preservation and guardrails. They appear in the same list as workflows, monitoring and logging, and troubleshooting.

EXAM TRAP

Bias and toxicity mitigation is not the objective about filters and escalation protocols; those belong to objective **9.4**, layered safety frameworks.

Enforcing safety with NeMo Guardrails

NeMo Guardrails is NVIDIA's component for safety enforcement. Objective **7.1** integrates it for compliance and safety enforcement. Suggested readings pair it with NVIDIA NIM to secure generative AI

deployments, and with LangChain Templates to build safer LLM apps.

Layered safety in the same domain

The same domain covers layered safety frameworks. Objective **9.4** asks you to deploy them, with filters and escalation protocols as examples. Study it next to bias and toxicity mitigation, since both sit in one domain.

Reading on ethical design

Ethically Aligned Design is a suggested reading. Its subtitle states the goal: a vision for prioritizing human well-being with autonomous and intelligent systems.

Key takeaways

- Bias and toxicity mitigation is objective **9.3**, in a domain worth **5%** of the exam.
- Bias detection, privacy preservation and guardrails are the named ethical safeguards.
- NeMo Guardrails is NVIDIA's component for compliance and safety enforcement.

Sources: NVIDIA-Certified Professional: Agentic AI Exam Study Guide (opens in a new tab)

Deploy layered safety frameworks (filters, escalation protocols).

Start here

THINK FIRST

How do you layer content safety checks and refusals so unsafe text is caught both coming in and going out?

Why it matters: Questions describe a guardrails setup and ask which model, prompt, parser or fallback makes the safety layer behave as intended.

Content safety in NeMo Guardrails runs as an input rail and an output rail. Each rail asks a safety model about the text and parses its reply into a safe or unsafe verdict. A refusal can then answer in the user's own language.

Key terms

Term	Meaning
Content safety rail	An input or output rail that runs `content_safety_check_input` or `content_safety_check_output`.
Output parser	Reads the safety model's reply and judges the content safe or unsafe; `is_content_safe` is the built-in one.
$model parameter	Names the safety model a flow uses; it must be defined under `models` in `config.yml`.

Term	Meaning
Model type	A model's unique identifier, passed to the input and output rails as a parameter.
Multilingual refusal	A refusal message in the user's detected language, for **9** supported languages.

Choosing a safety model

Plug in one of several content safety models. Options include NVIDIA's Nemotron Content Safety model, Meta's Llama Guard 3 and Google's ShieldGemma. Content moderation with Meta's Llama Guard is supported out of the box.

Wiring the input and output rails

Each rail runs its own content safety action. The `$model` parameter names the safety model both flows use. That model must be defined in the `models` section of `config.yml`, where its `type` is its unique identifier.

The two content safety rails compared

Feature	Input rail	Output rail
Action it runs	`content_safety_check_input`	`content_safety_check_output`
Text it judges	The input text	The output text
Safety model	Named by `$model`	Named by `$model`

Prompts and output parsers

Every safety flow needs a prompt and an output parser. If the prompt is missing, an exception is raised when the configuration loads. For the parser, register your own or use the built-in `is_content_safe`. It treats a reply containing "unsafe" or "yes" as unsafe.

EXAM TRAP

A missing safety prompt is not skipped quietly at runtime; loading the configuration raises an exception.

What a check returns

Each check returns a boolean saying whether the text is safe. Depending on the safety model, the check can also return a set of policy violations alongside that verdict.

Token budget for reasoning guards

Unset `max_tokens` falls back to 1024 tokens. If `max_tokens` is too small, a reasoning guard model can spend it all on reasoning. It then returns empty content with `finish_reason="length"`, and the parser typically treats that as unsafe.

EXAM TRAP

An empty reply from a guard model is not a pass: after `finish_reason="length"`, the parser typically treats it as unsafe.

Refusing in the user's language

Multilingual refusals support 9 languages. The `detect_language` action uses the fast-langdetect library to find the input's language. Any other language falls back to English. With no custom message for a detected language, that language's built-in default is used.

EXAM TRAP

An unsupported input language does not break the refusal; English is used as the fallback.

Key takeaways

- Content safety runs as an input rail and an output rail, each with its own action.
- Every safety flow needs a defined prompt and an output parser such as `is_content_safe`.
- Unset `max_tokens` falls back to **1024**; a reasoning guard that runs out returns empty content, usually judged unsafe.
- Refusals cover **9** languages and fall back to English for any other.

Sources: Content Safety | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab)

Ensure compliance with licensing and regulatory standards.

Start here

THINK FIRST

What do you owe when you add, edit, or borrow code in an agent toolkit released under Apache v2.0?

Why it matters: Questions describe a code change and ask which license header, date or notice it needs.

NeMo Agent Toolkit is released under **Apache v2.0**, and its files carry a header saying so. What you owe depends on the change: a new file, an edited file, or code taken from another project.

Key terms

Term	Meaning
Apache v2.0	The license NVIDIA NeMo Agent Toolkit is released under.
SPDX identifier	A standard tag in the file header; for this toolkit it reads `Apache-2.0`.
Copyright date range	The years in a file's copyright line, moved to the current year on every edit.
Third-party code	Code copied into the toolkit's tree, as opposed to an external dependency.

The toolkit's license

The toolkit ships under Apache v2.0. New source files, including build scripts, must carry the same license in their header.

New files

Every new file gets the Apache v2.0 header. That includes CMake files and other build scripts, not only application code. The header pairs an NVIDIA CORPORATION & AFFILIATES copyright line with the SPDX tag `Apache-2.0`.

EXAM TRAP

Build scripts are not exempt. A new CMake file needs the same Apache v2.0 header as any other new file.

What the header says

The header disclaims all warranties. Unless law or a written agreement requires otherwise, the software is distributed **AS IS**. No warranties or conditions of any kind are given, express or implied.

Editing existing files

An edit moves the copyright years to now. When you change an existing file, update the date range in its copyright line so it ends in the current year. A new file and an edited file are handled differently.

Third-party code

Borrowed code must be Apache v2.0 compatible. This covers code copied into the tree, not external dependencies. Keep its original license and a URL to where it came from. If you modify it, add the Apache v2.0 license first, then the original license and the URL.

What each kind of code change needs

Situation	What to do	Detail
Add a new file	Add the Apache v2.0 header	Applies to CMake and other build scripts too
Edit an existing file	Update the copyright date range	End the range in the current year
Copy in third-party code	Keep its original license	Add a URL to the original; it must be Apache-compatible
Modify copied third-party code	Apache v2.0 license, then the original	Also keep the URL to the original code

EXAM TRAP

Modified third-party code does not drop its original license. It carries Apache v2.0 first, followed by the original license and its URL.

Where this sits in the exam

Licensing is part of a 5% domain. Safety, Ethics, and Compliance carries **5%** of the exam weight. It is about agentic systems that act responsibly, ethically and within legal rules. Objective **9.2** in the same domain adds privacy and enterprise policy guardrails.

Regulatory reading

Suggested reading covers AI regulation. Two items stand out. One is the EU proposal for harmonized AI rules, the Artificial Intelligence Act. The other covers AI and machine learning used as medical device software.

Key takeaways

- New files, including CMake and other build scripts, need the Apache v2.0 header.
- Editing a file means updating its copyright years to the current year.
- Modified third-party code lists Apache v2.0 first, then its original license and URL.
- Safety, Ethics, and Compliance is worth **5%** of the exam.

Sources: Licensing — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · NVIDIA-Certified Professional: Agentic AI Exam Study Guide (opens in a new tab)

Domain 10 | 5% of the exam | 10 min read

Human-AI Interaction and Oversight

Overview

Every point below comes from NVIDIA's official documentation; each section lists its sources.

Build intuitive UIs with user-in-the-loop interaction.

Start here

THINK FIRST

How does an agent pause to ask a person something, and keep the conversation moving when they go quiet?

Why it matters: Questions describe an interactive agent and ask which Colang construct or toolkit component handles the user's turn.

User-in-the-loop design has two layers. Colang 2.0 flows shape the conversation, including loops, branches and silence. In NeMo Agent Toolkit, UserInteractionManager asks the user at runtime through a callback, while a front end runs the workflow.

Key terms

Term	Meaning
user silent	Pre-defined Colang 2.0 event for time-driven interaction when the user says nothing.
UserInteractionManager	Agent Toolkit component that requests user input at runtime through a callback.
user_input_callback	The callback in ContextState that does the actual prompting; SessionManager typically sets it.
FrontEndBase	Abstract base class for Agent Toolkit front ends, built from the full NAT configuration.

Loop the conversation

A `while` construct keeps the interaction going. In Colang 2.0 you build a continuous interaction loop with `while`. Inside it, the bot can act on its own when the user stays silent.

React to silence

Use the pre-defined `user silent` event for time-driven turns. For example, the bot can prompt the user after **12** seconds without input. A pause becomes a cue for the bot to speak.

Branch on what happens

`when` / `or when` **watches several paths at once.** At a branching point the flow monitors every branch. It continues as soon as the first branch matches.

EXAM TRAP

A `when` / `or when` block does not wait for all of its branches. The interaction continues as soon as one branch is matched.

Understand free-form input

The LLM can map an unhandled utterance to a user intent. The flow named `generating user intent for unhandled user utterance`, from the avatars module, asks the LLM for the canonical form of the message. That canonical form is the user intent.

Try it in a terminal

Test the loop with `nemoguardrails chat`. Point the command at your configuration folder with `--config`, then talk to the bot interactively.

Ask at runtime

`UserInteractionManager` **requests user input during a run.** Its `prompt_user_input` method asks a question and waits. The prompting itself is delegated to the callback in `ContextState.user_input_callback`, which SessionManager typically sets. The answer comes back as a string.

EXAM TRAP

The default callback does not prompt anyone. It is a no-op that just hands back the text held in the Interaction Content object.

Front ends run the workflow

`FrontEndBase` **is the base for every front end.** It is initialized with the full NAT configuration, and `front_end_config` returns the front end part of it. The abstract async `run()` method keeps the workflow running until the front end finishes.

Guide the agent's choices

Tool descriptions and routers steer the agent. In a workflow config, each tool's `description` tells the LLM when and how to use it. A Router Agent analyzes each request and sends it to the most appropriate branch.

Key takeaways

- Build Colang 2.0 interaction loops with `while`, and react to quiet users with `user silent`.
- `when` / `or when` continues on the first branch that matches.
- `UserInteractionManager` prompts through `ContextState.user_input_callback` and returns a string.
- The default input callback is a no-op that only returns the input text.

Sources: Interaction Loop | NVIDIA NeMo Guardrails Library Developer Guide (opens in a new tab) · nat.builder.user_interaction_manager — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.builder.front_end — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · About Building NVIDIA NeMo Agent Toolkit Workflows — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Design structured feedback loops that guide iterative agent improvements.

Start here

THINK FIRST

How can an agent's evaluation results be turned into feedback that makes each new version better than the last?

Why it matters: Questions describe an optimization loop and ask which feedback mode or setting makes the next round of changes useful.

A feedback loop runs the agent, scores it, explains the low scores, and feeds that explanation into the next change. The NeMo Agent Toolkit optimizer does this with oracle feedback, so prompts evolve toward fixing known failures.

Key terms

Term	Meaning
Oracle feedback	Reasoning taken from poorly scoring evaluation items and fed back into prompt mutation.
Mutation	How the genetic algorithm evolves prompts; evaluator reasoning can guide prompt mutations toward the identified issues.
EvalOutputItem reasoning	The evaluator field that explains a score; oracle feedback needs it filled in.
Fitness threshold	The score line below which an individual counts as failing in `failing_only` mode.

Score every run

Each workflow run's output goes to the evaluators in `eval_metrics`. Each evaluator scores one objective, such as correctness or latency. The output gives a score and its reasoning for every dataset entry.

Two kinds of optimization

Optuna tunes numerical parameters; genetic algorithms evolve prompts. Oracle feedback belongs to the prompt side. It injects failure reasoning from evaluations into the genetic algorithm, so the optimizer learns why prompts failed.

Explain the failures

Evaluators must fill the `reasoning` field for oracle feedback to work. The reasoning should say why an item scored poorly, not just repeat the score. That explanation steers the next prompt mutations toward the problem.

EXAM TRAP

A bare score is not enough feedback. Without reasoning in `EvalOutputItem`, oracle feedback has nothing useful to pass on.

Choosing a feedback mode

`oracle_feedback_mode` **defaults to** `never`**.** Pick a mode by how much feedback each round should get. Feedback text is capped at **4000** characters by default to protect the context window.

Oracle feedback modes compared

Mode	Who gets feedback	When
never	No one	Default; original behavior
always	Every mutation, from the parent's worst items	Every round
failing_only	Only individuals below the fitness threshold	Whenever they fall below it
adaptive	No one at first	Once fitness stagnates or diversity collapses

EXAM TRAP

Oracle feedback is off unless you choose a mode. The default, `never`, injects no feedback at all.

Steady results and refined plans

`reps_per_param_set` **repeats each run 3 times by default.** Repeats make noisy evaluations more stable. Planning can loop too: IterativePlanRefinementEditor drafts a plan and refines it with the same LLM, each item **3** times in parallel by default.

Key takeaways

- Oracle feedback feeds the reasons for low scores back into prompt mutation.
- Evaluators must populate `reasoning` in `EvalOutputItem`, explaining why an item scored poorly.

- `oracle_feedback_mode` defaults to `never`; `adaptive` turns feedback on when fitness stagnates or diversity collapses.
- Run each parameter set more than once; `reps_per_param_set` defaults to **3**.

Sources: NVIDIA NeMo Agent Toolkit Optimizer — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · Agent Evaluation in NVIDIA NeMo Agent Toolkit — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.plugins.config_optimizer.prompts.oracle_feedback — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab) · nat.experimental.test_time_compute.editing.iterative_plan_refinement_editor — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Implement transparency mechanisms (explainable reasoning, decision traceability).

Start here

THINK FIRST

How can a user see why an agent chose its actions, and trace each decision afterward?

Why it matters: Questions describe an agent whose choices must be explained and ask which design makes its reasoning visible.

Transparency has two parts: explainable reasoning and decision traceability. A reasoning agent plans before it acts. Decision traceability is supported by audit trails, designed and enforced alongside system security.

Key terms

Term	Meaning
Explainable reasoning	One of the two transparency mechanisms in this objective, beside decision traceability.
Decision traceability	The other transparency mechanism, backed by audit trails that are designed and enforced.
Reasoning agent	An Agent Toolkit agent that invokes an underlying function directly and reasons on top of it.
ReAct agent	An agent that reasons between steps, in contrast to a reasoning agent's upfront plan.

Two transparency mechanisms

Transparency covers explainable reasoning and decision traceability. It sits beside human oversight and intervention, which support accountability and trust. Live deployments must keep transparency and trust up continuously. The Human-AI Interaction and Oversight domain carries **5%** of the exam weight.

Plan ahead

A reasoning agent plans before it acts. Unlike a ReAct agent, it does its thinking up front, as a plan, instead of between steps.

EXAM TRAP

A reasoning agent does not reason between steps the way a ReAct agent does. It plans ahead of time.

What a reasoning agent needs

Choose an LLM that supports reasoning. The agent invokes an underlying function directly and reasons on top of it. You can tailor its prompts in the workflow's YAML config file.

Trace decisions

Decision traceability rests on audit trails. Design and enforce audit trails as part of system security. For the explainable reasoning side, Chain-of-Thought prompting is suggested reading.

Key takeaways

- Transparency means explainable reasoning plus decision traceability.
- A reasoning agent plans ahead instead of reasoning between steps like ReAct.
- A reasoning agent needs an LLM that supports reasoning.

Sources: NVIDIA-Certified Professional: Agentic AI Exam Study Guide (opens in a new tab) · About Reasoning Agent — NVIDIA NeMo Agent Toolkit (1.9) (opens in a new tab)

Enable human oversight and intervention for accountability and trust.

Start here

THINK FIRST

What does a person need from an agentic system before they can oversee it, step in, and answer for its decisions?

Why it matters: Questions describe an agent deployment and ask which oversight or intervention mechanism builds accountability and trust.

Systems should be designed to make human oversight of agents effective. Transparency shows why an agent acted, and audit trails keep a record that supports accountability. The objective pairs this oversight with intervention, for accountability and trust.

Key terms

Term	Meaning
Human-in-the-loop	A design where a person takes part in the agent's work, for example through a user-in-the-loop UI.
Decision traceability	A transparency mechanism that lets people follow how the agent reached a decision.
Audit trail	A record the system keeps of its actions, enforced to support accountability.
Escalation protocol	A layer of a layered safety framework for agents, used alongside filters.

What the domain covers

Human-AI Interaction and Oversight is about designing for oversight. The domain covers the design and build of systems that let people oversee and interact with agents well. It carries a **5%** exam weight.

Seeing why the agent acted

Transparency comes from explainable reasoning and decision traceability. Both are listed as transparency mechanisms that support human oversight of an agent.

Mechanisms that support oversight

Each mechanism gives people a different handle on the agent. Some let people see, some keep a record, and some let them step in or shape later behavior.

Oversight mechanisms and what each gives a person

Mechanism	What it gives	Exam objective area
Explainable reasoning and decision traceability	Visibility into how decisions were made	Transparency mechanisms
Audit trails	A record that supports accountability	System security
Filters and escalation protocols	Layers of safety, with a path to escalate	Layered safety frameworks
User-in-the-loop UI	A direct way for people to take part	Intuitive interfaces
Structured feedback loops	Guidance for iterative agent improvement	Feedback design

EXAM TRAP

Oversight is not only an interface question. Audit trails also support accountability, and they are enforced as part of system security.

Trust in live deployments

Live deployments should stay up, transparent and trusted. Continuous uptime sits beside transparency and trust as a goal for any agent deployment that is live.

Key takeaways

- Transparency for oversight means explainable reasoning and decision traceability.
- Audit trails are enforced as part of system security to support accountability.
- Layered safety frameworks combine filters with escalation protocols.
- Structured feedback loops guide iterative agent improvements.

Sources: NVIDIA-Certified Professional: Agentic AI Exam Study Guide (opens in a new tab)

MockCert

Take a NVIDIA-Certified Professional: Agentic AI mock exam the way the real one feels

5 timed mock exams · 65 questions in 120 minutes each · one is free

- Free to start: study notes, explained quizzes, flashcards and the first mock exam
- Timed mock exams built to the official exam guide
- A verifiable Certificate of Assessment when you pass a premium mock

Explore NVIDIA-Certified Professional: Agentic AI on MockCert

Open it on your phone

mockcert.com/nvidia-ncp-aai/

Scan the code with your phone's camera.