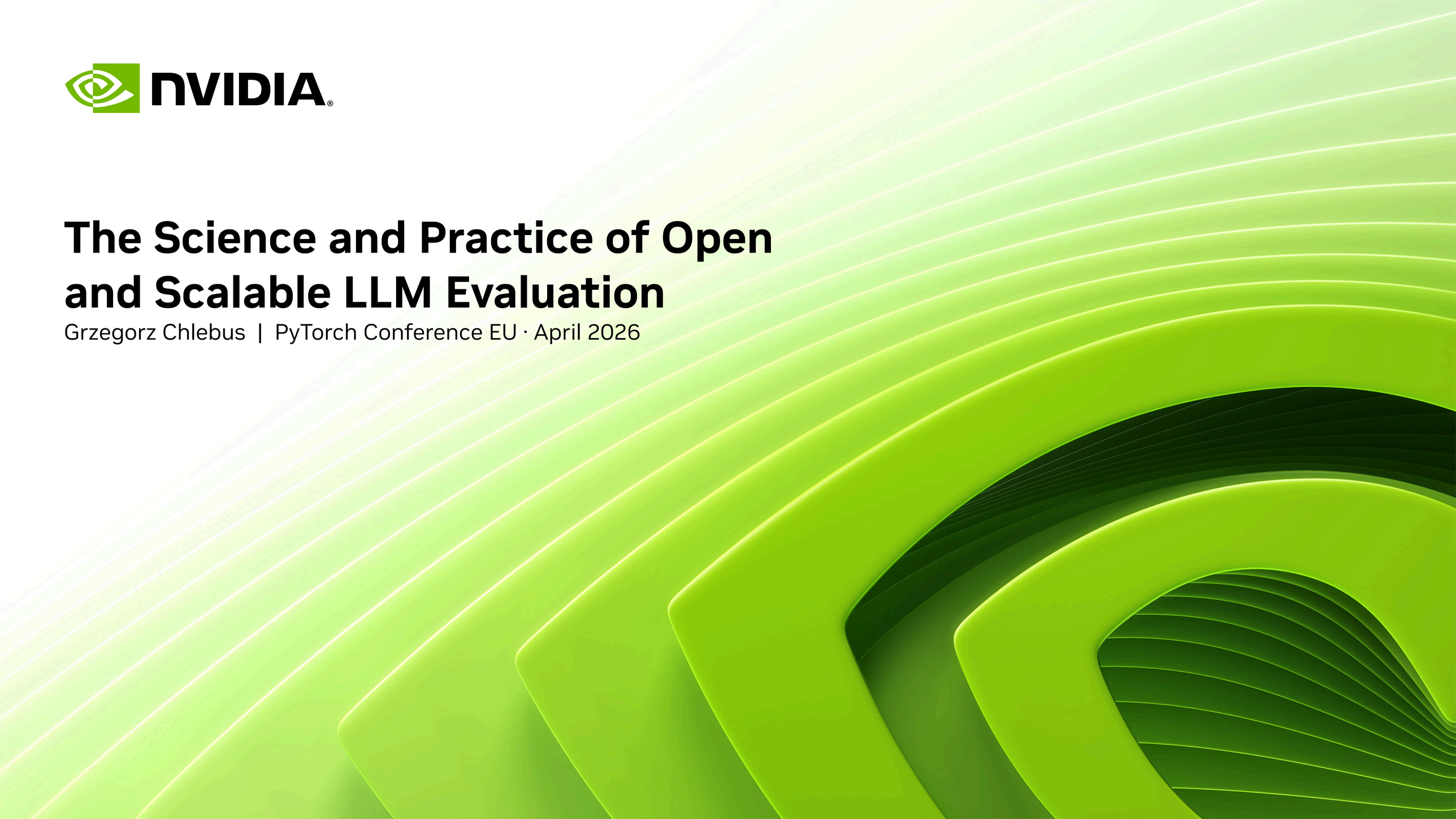


The Science and Practice of Open and Scalable LLM Evaluation

Grzegorz Chlebus | PyTorch Conference EU · April 2026

Why Reported Numbers Often Fail

The Reproducibility Gap

Every major model release claims benchmark wins - but can you **reproduce** them?

Model	Benchmarks	Compared Against
Nemotron 3 Nano	26	2 models
Nemotron 3 Super	33	2 models
GPT-OSS-120B	20	3 models
Qwen 3.5-122B	82	5 models
Gemma 4-31B	15	5 models
Kimi K2.5	15	3 models
DeepSeek V3	27	6 models

Hundreds of numbers published - but prompts, configs, harnesses, scoring rules, and inference params are **rarely shared**. Without full methodology, these numbers are **not comparable**.

The Reproducibility Gap

Same model + different setup = different numbers

🔧 SWE-Bench Verified

Which coding agent harness? All 500 tasks or a subset?
Reasoning / thinking mode on or off?

🔍 HLE with Tools (search agent)

Which search tools and APIs?
Max tool calls per task - 10? 50? Unlimited?
Context window management strategy?

🔪 Long-Context Reasoning (e.g. AA-LCR)

Max completion tokens? How many samples hit the limit?
Logic for truncated / empty responses - scored 0 or dropped?
Allocation window: did all samples get equal compute?

The Reproducibility Gap

Same model + different setup = different numbers

🔧 SWE-Bench Verified

Which coding agent harness? All 500 tasks or a subset?
Reasoning / thinking mode on or off?

🔍 HLE with Tools (search agent)

Which search tools and APIs?
Max tool calls per task - 10? 50? Unlimited?
Context window management strategy?

🔪 Long-Context Reasoning (e.g. AA-LCR)

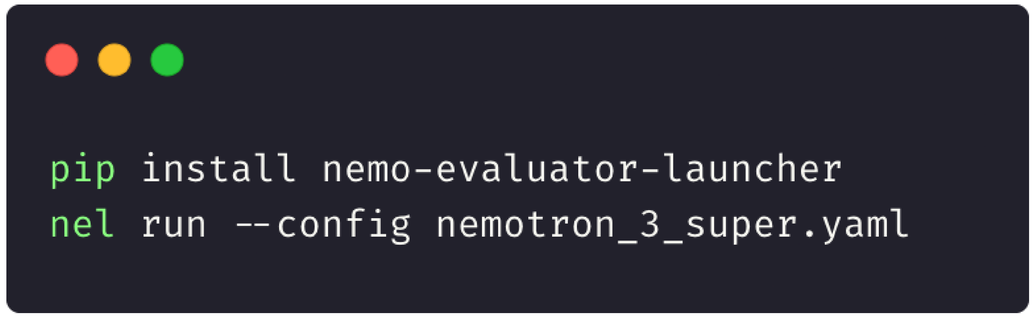
Max completion tokens? How many samples hit the limit?
Logic for truncated / empty responses - scored 0 or dropped?
Allocation window: did all samples get equal compute?

Our response: open evaluation recipes

Starting with Nemotron 3 Nano, NVIDIA publishes **full eval configs** alongside every model release.

- **Exact YAML configs** - prompts, sampling, scoring
- **Reproducibility tutorials** with step-by-step guides

Nemotron 3 Nano recipe ↗ | Nemotron 3 Super recipe ↗



Powered by NeMo Evaluator SDK

Patterns vs. Anti-Patterns

🚫 Anti-Patterns

- Cherry-picking prompt strategies
- Reporting best run, not average
- Accuracy without CIs, stderr
- Single run on stochastic benchmarks
- No reproducible configs
- No contamination checks

Patterns vs. Anti-Patterns

🚫 Anti-Patterns

- Cherry-picking prompt strategies
- Reporting best run, not average
- Accuracy without CIs, stderr
- Single run on stochastic benchmarks
- No reproducible configs
- No contamination checks

✅ Patterns

- Versioned eval recipes with config hash
- n-repeats + confidence intervals
- Per-category breakdowns
- Significance tests between checkpoints
- Open configs, prompts, scoring rules
- Contamination detection in pipeline (CoDeC, ICLR 2026)

The Evaluator Architecture

From Benchmarks to Environments

GEN 1

Static

Prompt + answer

Score over answers

GEN 2

Harnesses

Config + runner + scorer

Standardized execution

From Benchmarks to Environments

GEN 1

Static

Prompt + answer

Score over answers

GEN 2

Harnesses

Config + runner + scorer

Standardized execution

GEN 3

Agent

Task + environment + loop

Multi-turn, tool use, sessions

GEN 4

Platforms

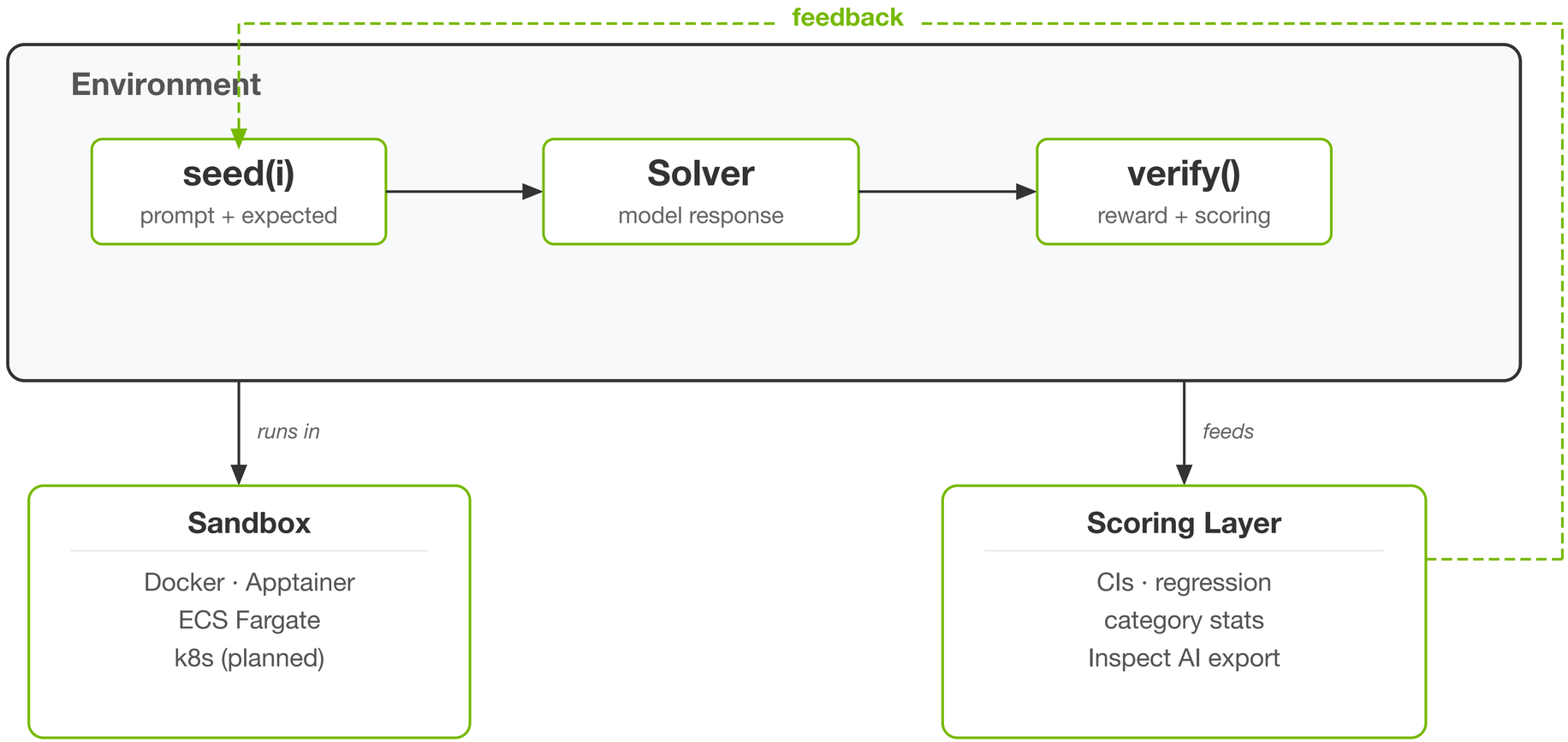
Portable env package

Stateful agents, cross-session memory, agentic OS (e.g., OpenClaw)

Most tooling is built for **Gen 2**. **Frontier models** already need **Gen 4**.

The Core Loop

seed → solve → verify → score → decide



Evaluator owns the loop. Scoring, aggregation, reporting — all inside NeMo Evaluator.

Environments

One interface - extensible to any provider

```
env = get_environment("mmlu")                          # built-in envs
env = get_environment("lm-eval://aime25")              # envs from 3rd party harnesses
env = get_environment("harbor://terminal-bench@2.0")   # Harbor agents
env = get_environment("my_benchmark.py")               # envs via BYOB
```

All share the same contract:

```
class EvalEnvironment:
    def seed(self, idx) -> SeedResult          # Get the problem
    def verify(self, response, expected)       # Judge the answer
        -> VerifyResult
```

Adding a new env provider is a **thin adapter** - CIs, regression, traces, logs, and result export work automatically.

Decision-Grade Evaluation

Not decision-grade

Scoring reward avg is 0.723

Comparison Model B got higher

Granularity 72% overall

Regression *(nothing)*

Reproducibility *(not tracked)*

Decision-Grade Evaluation

Not decision-grade

Scoring reward avg is 0.723

Comparison Model B got higher

Granularity 72% overall

Regression *(nothing)*

Reproducibility *(not tracked)*

Decision-grade

Scoring 72.3% ± 1.8% (95% CI, n=8)

Comparison Model B is better, $p < 0.01$

Granularity 87% algebra, 34% geometry

Regression Instruction following dropped 8%

Reproducibility Config hash, SDK versions, timestamps

When a number **moves**, you should **know whether to trust it**.

nel gate

`nel gate` answers **can we ship this** across a benchmark suite.

```
nel gate ./results/baseline ./results/candidate --policy gate_policy.yaml --strict

NO-GO
Policy:    gate_policy.yaml
Baseline:  ./results/baseline
Candidate: ./results/candidate

VERDICT REASONS
  - BREACH: gpqa [critical], mmlu_pro [critical]

BENCHMARKS
  name                tier         status                metric       delta
  ──────────────────────────────────────────────────────────────────────────
  gpqa                critical     BREACH                mean_reward  -0.0150
    - Absolute drop 0.0150 exceeds threshold 0.0100
  humaneval           supporting   PASS                  pass@1       -0.0050
    - 95% CI on damage [0.0010, 0.0080] is within threshold 0.0150
  mmlu_pro            critical     BREACH                mean_reward  -0.0120
    - 95% CI on damage [0.0105, 0.0142] exceeds threshold 0.0100
  triviaqa            advisory     PASS                  mean_reward  +0.0120
```

Agentic Eval & When Things Broke

What Makes Agentic Eval Different

- 🔒 **Sandboxing**
  - Per-problem isolation: Docker, Apptainer, ECS Fargate, SLURM
  - Untrusted model code = non-negotiable

What Makes Agentic Eval Different

- 🔒 **Sandboxing**
  - Per-problem isolation: Docker, Apptainer, ECS Fargate, SLURM
  - Untrusted model code = non-negotiable
- 🔄 **Multi-Turn**
  - Each sample is a conversation, not a prompt-response pair
  - State across turns, timeouts, context window management

What Makes Agentic Eval Different

- 🔒 **Sandboxing**
  - Per-problem isolation: Docker, Apptainer, ECS Fargate, SLURM
  - Untrusted model code = non-negotiable
- 🔄 **Multi-Turn**
  - Each sample is a conversation, not a prompt-response pair
  - State across turns, timeouts, context window management
- 🛠️ **Tool Use**
  - Code execution, web search, file systems, APIs
  - Max tool use cap

What Makes Agentic Eval Different

- 🔒 **Sandboxing**
  - Per-problem isolation: Docker, Apptainer, ECS Fargate, SLURM
  - Untrusted model code = non-negotiable
- 🔄 **Multi-Turn**
  - Each sample is a conversation, not a prompt-response pair
  - State across turns, timeouts, context window management
- 🛠️ **Tool Use**
  - Code execution, web search, file systems, APIs
  - Max tool use cap
- 📈 **Scale**
  - 1k+ sandboxes × agent loop

What Makes Agentic Eval Different

- 🔒 **Sandboxing**
  - Per-problem isolation: Docker, Apptainer, ECS Fargate, SLURM
  - Untrusted model code = non-negotiable
- 🔄 **Multi-Turn**
  - Each sample is a conversation, not a prompt-response pair
  - State across turns, timeouts, context window management
- 🛠️ **Tool Use**
  - Code execution, web search, file systems, APIs
  - Max tool use cap
- 📈 **Scale**
  - 1k+ sandboxes × agent loop
- 📦 **Context Compaction**
  - Sliding window, summarization, hierarchical compression, RAG
  - Strategy choice changes scores - must be fixed and reported

What Makes Agentic Eval Different

- 🔒 **Sandboxing**
  - Per-problem isolation: Docker, Apptainer, ECS Fargate, SLURM
  - Untrusted model code = non-negotiable
- 🔄 **Multi-Turn**
  - Each sample is a conversation, not a prompt-response pair
  - State across turns, timeouts, context window management
- 🛠️ **Tool Use**
  - Code execution, web search, file systems, APIs
  - Max tool use cap
- 📈 **Scale**
  - 1k+ sandboxes × agent loop
- 📦 **Context Compaction**
  - Sliding window, summarization, hierarchical compression, RAG
  - Strategy choice changes scores - must be fixed and reported
- 🧠 **Memory & State Across Sessions**
  - Persistent memory: RAG, vector DBs, relational stores
  - Cross-session learning changes what “the same agent” means

When Things Broke: HLE with Tools

2,158 items · 2 stuck for 7 days

What happened:

- Raw Python sandbox - no search API, no browser
- Model ran `import requests` → HTTP GET to Google
- Data center IP → **anti-bot block page**
- 100K-token reasoning blocks about why it couldn't click
- DuckDuckGo, `pip install wikipedia`, BeautifulSoup - all dead ends

When Things Broke: HLE with Tools

2,158 items · 2 stuck for 7 days

What happened:

- Raw Python sandbox - no search API, no browser
- Model ran `import requests` → HTTP GET to Google
- Data center IP → **anti-bot block page**
- 100K-token reasoning blocks about why it couldn't click
- DuckDuckGo, `pip install wikipedia`, BeautifulSoup - all dead ends

The cost:

- **71 chain breaks** across 4-hour Slurm allocations
- **42,806 API requests**
- **580 GPU-hours wasted**

Lesson:

- Sandbox design choices have **massive cost implications**
- Curate the tool suite for the benchmark

When Things Broke: AA-LCR

Long-context reasoning · unreleased model · chained cluster jobs

What happened:

- Long-context reasoning eval across **chained Slurm jobs**
- Each job got a **fixed time window** on the cluster
- When a job timed out, the next one resumed - skipping finished samples
- Unfinished samples got **another attempt** in the next job
- Some samples had one shot; others had two or three
- Two runs of the same model on the same benchmark produced **different scores**

When Things Broke: AA-LCR

Long-context reasoning · unreleased model · chained cluster jobs

What happened:

- Long-context reasoning eval across **chained Slurm jobs**
- Each job got a **fixed time window** on the cluster
- When a job timed out, the next one resumed - skipping finished samples
- Unfinished samples got **another attempt** in the next job
- Some samples had one shot; others had two or three
- Two runs of the same model on the same benchmark produced **different scores**

Why it matters:

- Samples that finish fast complete on the first job
- Hard or long-running samples spill into later jobs
- Those samples get **extra attempts** the fast ones never needed
- The reported score silently depends on **how many chances each sample got**

Lesson:

- Scheduler boundaries can shape evaluation results
- Every sample should get **exactly one clean attempt**
- Infra details *are* evaluation details

Lessons from the Trenches

- **1. Real Tools, Not Just Sandboxes**
  - Curate the tool suite for the benchmark
  - Watch out for API rate limits (e.g., search providers)
- **2. Circuit Breakers**
  - Timeout per sample, not per job
  - Limits on tool calling
- **3. Logs Are Everything**
  - Per-turn rich traces
- **4. 🦞 Automated Monitoring**
  - AI agents monitoring evals 24/7

Lessons from the Trenches

- **1. Real Tools, Not Just Sandboxes**
  - Curate the tool suite for the benchmark
  - Watch out for API rate limits (e.g., search providers)
- **2. Circuit Breakers**
  - Timeout per sample, not per job
  - Limits on tool calling
- **3. Logs Are Everything**
  - Per-turn rich traces
- **4. 🦞 Automated Monitoring**
  - AI agents monitoring evals 24/7

Monitoring in practice

Stats from the last two weeks:

- **82 benchmark evaluations** monitored
- **150 Slurm chain jobs** managed
- **15 distinct failure modes** caught
- **8+ autonomous recoveries**

OOM, NCCL crashes, stale credentials, zombie chains, KV cache overflow

Est. 2,600 GPU-hours saved by early detection

Use NeMo Evaluator for Your Benchmarks

Bring Your Own Benchmark

What it takes:

```
@benchmark(
    name="my-bench",
    dataset="hf://my-org/data",
    prompt="Q: {question}\nA:",
    target_field="answer",
)
@scorer
def score(sample):
    return exact_match(sample)
```

What you get:

- Bootstrap CIs + pass@k
- Regression detection with p-values
- Per-category breakdowns
- Inspect AI log export
- NeMo Gym-compatible RL endpoint
- Container packaging

Even complex agentic evals like **PinchBench** use this path.

Call to Action

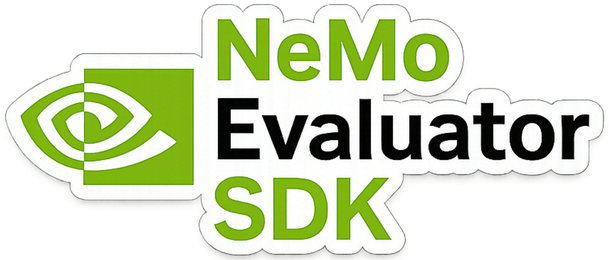
1. Start with **one** benchmark.

```
pip install "nemo-evaluator[all] @ git+https://github.com/NVIDIA-NeMo/Evaluator.git@dev/0.3.0"
nel eval run --bench mmlu --model-url "https://api.example.com/v1"
```

📦 Preview — `dev/0.3.0` branch available now

2. Make your evals **decision-grade.**

3. **Contribute** your benchmarks back.



Explore NeMo Evaluator on GitHub and ⭐ it if it earns a place in your workflow

🔗 **github.com/NVIDIA-NeMo/Evaluator**

NVIDIA Developer Champions Program

A select cohort for top community builders:

- ✓ Priority access to NVIDIA experts & resources
- ✓ Global visibility for your work
- ✓ Exclusive perks

Apply To Become a Champion
developer.nvidia.com/developer-champions-program

Contact: devchamps@nvidia.com

NeMo Evaluator - Open-source, decision-grade LLM evaluation

🔗 github.com/NVIDIA-NeMo/Evaluator

📧 gchlebus@nvidia.com