



The science behind the report:

Expand your capacity for agentic AI with Dell PowerEdge XE9780LAP servers featuring Intel Xeon 6 processors

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report [Expand your capacity for agentic AI with Dell PowerEdge XE9780LAP servers featuring Intel Xeon 6 processors](#).

We concluded our hands-on testing on August 6, 2026. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on July 29, 2026, or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

System configuration information

Table 1: Detailed information on the system we tested.

System configuration information	Dell™ PowerEdge™ XE9780LAP
BIOS name and version	Dell 1.2.9
Non-default BIOS settings	SR-IOV Global Enable = Enabled PCIe® Resizable BAR = Enabled
Operating system name and version/build number	Ubuntu 24.04.4 LTS (kernel 6.8.0-134-generic)
Date of last OS updates/patches applied	2026-07-29
Power management policy	BIOS System Profile: Custom (Workload Profile = Low Latency Optimized); OS governor: schedutil on intel_cpufreq
Processor	
Number of processors	2
Vendor and model	Intel® Xeon® 6979P
Core count (per processor)	120
Core frequency (GHz)	2.1
Stepping	Model 173 Stepping 1

System configuration information		Dell™ PowerEdge™ XE9780LAP
Memory module(s)		
Total memory in system (GB)		2,304
Number of memory modules		24
Vendor and model		SK hynix HMCGM4MHBRB498N
Size (GB)		96
Type		DDR5
Speed (MT/s)		6,400
Speed running in the server (MT/s)		6,400
Storage controller		
Vendor and model		Dell BOSS-N1 Modular DC-MHS (OS), direct-attached NVMe (data)
Cache size (GB)		N/A
Firmware version		2.2.13.2038
Driver version		In-tree nvme (kernel 6.8.0-134-generic)
Local storage (OS)		
Number of drives		1
Drive vendor and model		Micron 7450
Drive size (GB)		480
Drive information (speed, interface, type)		PCIe Gen4 M.2 NVMe SSD
Local storage (data)		
Number of drives		1
Drive vendor and model		Samsung PM9D3a
Drive size (GB)		3,840
Drive information (speed, interface, type)		PCIe Gen5 NVMe SSD
Network adapter		
Vendor and model		Broadcom BCM57414 2x25G OCP Ethernet NIC
Number and type of ports		2x SFP28 (10/25GbE)
Driver version		bnxt_en, in-tree (kernel 6.8.0-134-generic); NIC firmware 233.0.195.0 (pkg 233.1.181.0)

System configuration information		Dell™ PowerEdge™ XE9780LAP
Accelerators		
Vendor and model		NVIDIA® Blackwell Ultra B300 SXM6
Number		8
Memory type		HBM3e with ECC
Memory size (GB)		288
Memory bandwidth (GB/s)		8,000
Power cap limit (W)		1,100
Connection		PCIe Gen6 x16, NVLink 5 interconnects
Driver		595.71.05
Cooling fans		
Vendor and model		Dell Platinum
Number of cooling fans		12

How we tested

We developed a proprietary agentic AI benchmark framework that runs AI agents completing representative tasks in the manufacturing, financial services, and coding domains. The tasks are real-world; the data they reference is synthetic and generated by the benchmark framework.

Tables 2, 3, and 4 offer details on each of the 32 tasks we used in testing: 8 manufacturing tasks, 8 financial services tasks, and 16 coding tasks.

Table 2: The manufacturing tasks in our agentic AI benchmark.

Task name	What it exercises	How it stresses hardware
mfg_spc_analysis	Loads multi-series production telemetry (think: 50 sensors, each producing one reading per second for a week), runs the full SPC battery on every series — control chart limits, Cp/Cpk capability indices, Western Electric rule scans, CUSUM and EWMA change detection, normality tests — and asks the LLM to identify which parameters are out of control and what the likely root causes are.	Heavy CPU compute, mostly NumPy/SciPy.
mfg_oeo_calculation	Simulates a production event log for a multi-line factory across a multi-week window (machine starts, stops, downtime reasons, output counts, scrap counts), rolls up per-line-per-shift OEE — the industry-standard metric defined as Availability x Performance x Quality — identifies the worst-performing lines, and explains the dominant loss drivers (changeover time, micro-stops, scrap rate).	Pandas roll-ups across event logs, then a small LLM call. CPU + memory bandwidth on big event-log volumes; stresses analytical pipelines more than LLM inference.
mfg_batch_genealogy	Constructs a real directed graph of a supply chain (component supplier to component lot to batch to shipment to customer), simulates a recall on a specific component lot, traces every downstream batch and customer that received material from the contaminated lot using BFS, and recommends a containment action.	NetworkX graph operations on a directed acyclic graph that can grow into the millions of edges. Stresses CPU and graph-algorithm libraries; LLM call is small.
mfg_predictive_maintenance	Generates synthetic vibration-sensor data for a fleet of rotating machinery (motors, pumps, gearboxes), runs the diagnostic battery — rolling RMS, rolling kurtosis, FFT band energies, two-state Kalman-style projection filter — on every channel, and asks the LLM to classify each machine's risk level and estimate remaining useful life (RUL) in days.	Heavy DSP compute (FFT, rolling-window stats) on a fleet of synthetic time series. CPU + memory bandwidth dominant. The narrative LLM step is small.
mfg_defect_trend_investigation	Simulates daily defect counts split by (defect type x production line) across a multi-month window, runs the full trend-analysis battery (rolling stats, linear-regression slope, autocorrelation at lags 1/7/14 to detect weekly patterns, weekend-vs-weekday ratios, CUSUM change-point detection), then runs a multi-turn ReAct loop where the LLM investigates specific spikes by querying for related context and proposing root causes.	Statistics-heavy CPU compute, then a multi-turn LLM loop (3–8 turns typical). Mixed CPU + LLM-latency profile.
mfg_quality_inspection	The simplest of the three vision scenarios. Receives a production image (rendered from a synthetic 3D scene of a product on a conveyor), asks a vision-language model to determine pass/fail and identify any defect category visible in the image. One round trip per part.	Vision-language model inference. Stresses the dedicated vision GPU(s) more than the text GPUs.

Task name	What it exercises	How it stresses hardware
mfg_quality_analysis	The previous scenario plus a second stage. After the vision model classifies a defect, the agent pulls the relevant production telemetry from the time window of that part's manufacture and asks the text LLM to correlate the defect with telemetry signals — was a temperature out of spec, did a tool wear into a known pattern, was a machine ramping back up after a changeover?	Mixed vision + text. Stresses the multi-instance topology, with vision running on its own GPU group while the text LLM works on a different group.
mfg_quality_full_pipeline	The most ambitious scenario in the suite. Four-stage pipeline: (1) vision inspection, (2) telemetry root-cause correlation, (3) disposition decision (quarantine? rework? continue?), (4) report generation including suggested process-engineering follow-up. Multiple round trips between vision model and text LLM with intermediate analytical steps.	The most demanding workload in the suite. Vision GPU + text GPU + CPU all engaged simultaneously. Best for showcasing dual-NUMA, multi-GPU topology benefits.

Table 3: The financial services tasks in our agentic AI benchmark.

Task name	What it exercises	How it stresses hardware
filing_discrepancy	Reads a synthetic 10-K filing as a PDF, pulls key financial figures (revenue, net income, EPS, segment results), cross-references those figures against an authoritative database of the same company's reported numbers, and writes a structured risk summary that flags any mismatches with materiality grading.	PDF text extraction stresses CPU, the cross-reference query hits PostgreSQL, and the final structured-summary generation is an LLM call against the long extracted text. Typical of “intelligent document processing” pitches.
txn_reconcile	At the end of a banking day, replays every posted transaction across 500 branches and 50,000 accounts, computes what the closing balance should be for each account, compares against the ledger's actual posted balance, flags accounts that don't reconcile, and writes a per-exception narrative for the operations team.	Heavy SQL aggregation against a 1M-row transaction table on PostgreSQL, then a relatively short LLM call to narrate flagged exceptions. Stresses database I/O, CPU concurrency, and storage subsystem.
portfolio_valuation	For a given investment portfolio, loads every position the portfolio holds, joins each position against the most recent market-close price, computes the portfolio's net asset value (NAV), ranks the top three sector exposures by percent of NAV, and writes a brief explanatory note about NAV movement.	Pandas/numpy compute over 100k positions × 30 historical snapshots = 3 million rows, joined against 1.26M market-data rows. CPU + RAM are the constraint; LLM is a small narrate-the-result step at the end.
reg_ratios	Loads a slice of bank balance-sheet inputs (capital, assets, liabilities, off-balance-sheet exposures), computes the regulatory ratios that bank regulators care about — Basel capital adequacy, Liquidity Coverage Ratio (LCR), Net Stable Funding Ratio (NSFR), leverage ratio, debt-service coverage — and flags any ratio that breaches its threshold.	Small structured-data pull, modest CPU compute, then an LLM call to explain breaches in plain English. Less compute-intensive than the other financial scenarios — useful for showing baseline LLM throughput on small inputs.
peer_olap	Picks a target ticker, finds every other publicly traded company in the same industry sector, computes trailing-window returns and volatility for the cohort using SQL window functions, ranks the target against its peers, and writes a sector-commentary paragraph that names the leaders and laggards.	SQL window functions over 1.26M market-data rows — heavy on database CPU and the analytical query optimizer. The LLM narration is small. Good for demonstrating analytical-database performance.
fraud_investigation	Scans a week of bank transactions for three classic fraud patterns (rapid-fire same-account activity, suspiciously round-number debits, and large rapid debits), surfaces 10–30 candidate accounts, then runs a multi-turn investigation loop where the AI pulls account history, runs additional rule checks, and decides which accounts to escalate to a human investigator with written justification.	The SQL pre-filter is database-heavy; the multi-turn investigation is the most LLM-intensive scenario in the suite (six to ten back-and-forth turns per investigation). Stresses LLM inference latency more than throughput — investigators want fast turns.

Task name	What it exercises	How it stresses hardware
filing_qa	Loads ten full synthetic 10-K filings into a single context window, takes a batch of ten questions about specific numbers in those filings (one question per filing — “what was Apex Data Group’s 2023 revenue in USD millions?”), and answers all ten in one LLM call.	Long-context LLM inference. Stresses model memory (KV cache) and compute together — the larger the context, the more the GPU has to hold and process. Demo of the benefits of VRAM and faster GPUs for context-heavy workloads.
board_pack_narrative	Takes a structured “facts table” for one company-quarter (revenue, net income, segment results, headline risks, forward outlook) and generates a five-section board pack: executive summary, revenue and earnings, segment results, key risks, and outlook. Each section has a length budget (80–400 words) and a list of must-mention keywords.	Longest single LLM generation in the suite — 5 sections × up to 400 words each is roughly 3,000 output tokens per pack. Stresses output throughput (tokens/second). Good demo for writing quarterly content at scale.

Table 4: The coding tasks in our agentic AI benchmark.

Task name	What it exercises	How it stresses hardware
csv-validator	A self-contained task at the moderate tier, with a budget of 18 agent steps. The agent receives a single 49-line validator.py whose validate_row function fails 2 of its 7 tests (an empty value treated as a missing field and an unknown field type accepted silently), and edits the file until the hidden test suite passes.	GPU-decode dominated: up to 18 sequential turns of roughly 1,500 tokens each. There is no retrieval index, so the only CPU work is the pytest subprocess the agent spawns each time it runs the tests. A clean probe of LLM inference latency.
fix-pagination	The simplest task in the coding suite: a self-contained fix at the simple tier with a budget of 12 agent steps. A 12-line paginate(items, page, per_page) function carries a single off-by-one error in its 1-indexed page math, and 5 tests define correct behavior.	GPU decode across up to 12 short turns, with negligible CPU beyond pytest. Pure single-agent inference latency.
flask-blueprint-url-prefix-001	The first of the real-world tasks, which run against a vendored copy of Flask 3.0.3 (24 files, 9,024 lines of code) with a budget of 40 agent steps. A 13-line patch injects a doubled-slash bug into blueprint url_prefix handling, and 2 hidden tests define the fix. The agent works through a search_codebase tool backed by a roughly 684-chunk index of the repository.	A CPU and GPU hybrid: a one-time bge-small embedding pass on CPU (cached to disk), then cross-encoder reranking of up to 50 candidates per search plus pytest runs, all overlapping up to 40 GPU decode turns. The clearest demonstration of a CPU that retrieves and reranks while the GPU generates.
flask-config-namespace-001	A real-world task on the same Flask 3.0.3 tree under the same 40-step budget. A 13-line patch breaks key trimming in Config.get_namespace, and 2 tests define the fix.	The same hybrid profile: retrieval and reranking on CPU over the 684-chunk Flask index, running alongside GPU decode.
flask-config-namespace-002-UNFIXABLE	A deliberate trap: no patch is applied and no bug is injected. The shipped test is self-contradictory, so no code change can satisfy it. Correct behavior is for the agent to give up and cite the contradictory test, and the run scores as a true negative only when that citation is grounded in the repository.	The same hybrid profile, but it exercises the give-up and grounding path rather than a fix. This task measures refusal with evidence, not throughput.
flask-config-prefixed-env-001	A real-world task under the same 40-step budget. A 13-line patch breaks nested double-underscore handling in Config.from_prefixed_env, and 3 tests define the fix.	Hybrid: retrieval and reranking on CPU, plus GPU decode.
flask-double-bug-001	A real-world task with two coupled defects. A 20-line patch injects two regressions in JSON-tag byte round-tripping, and 5 tests in test_json_tag.py define the fix — the agent has to land more than one correct edit.	Hybrid and edit-heavier: more edit-and-retest cycles mean more pytest CPU and more decode turns against the same index.
flask-helpers-debug-flag-001	A real-world task under the same 40-step budget. A 13-line patch inverts the logic of get_debug_flag, and 3 tests define the fix.	Hybrid: retrieval and reranking on CPU, plus GPU decode.
flask-json-provider-default-001	A real-world task under the same 40-step budget. A 13-line patch breaks the JSON provider’s default encoding for dates and RFC 822 values, and 4 tests define the fix.	Hybrid: retrieval and reranking on CPU, plus GPU decode.

Task name	What it exercises	How it stresses hardware
flask-json-provider-sortkeys-001	A real-world task under the same 40-step budget. A 13-line patch flips the JSON provider's sort_keys default, and 2 tests define the fix.	Hybrid: retrieval and reranking on CPU, plus GPU decode.
flask-response-mimetype-001	A real-world task under the same 40-step budget. A 13-line patch changes the default mimetype on Response, and 2 tests define the fix.	Hybrid: retrieval and reranking on CPU, plus GPU decode.
flask-session-name-001-UNFIXABLE	The second deliberate trap, again with no patch applied. Here the shipped test encodes a wrong specification, so the correct behavior is a grounded decision to give up, scored as a true negative.	The same hybrid profile; exercises grounded refusal rather than a fix.
flask-tagbytes-decode-001	A real-world task under the same 40-step budget. An 11-line patch breaks bytes decoding in the JSON-tag path, and 5 tests in test_json_tag.py define the fix.	Hybrid: retrieval and reranking on CPU, plus GPU decode.
flask-triple-bug-001	The heaviest fix in the coding suite. A 29-line patch injects three JSON-tag regressions, and 5 tests in test_json_tag.py define the fix, giving the agent the longest edit chain of any task.	Hybrid, and the most edit-and-retest-intensive task in the set: the highest pytest CPU load and the most decode turns of any fixable coding scenario.
flask-views-methodview-001	A real-world task under the same 40-step budget. A 13-line patch breaks HTTP-verb dispatch in MethodView, and 2 tests define the fix.	Hybrid: retrieval and reranking on CPU, plus GPU decode.
ttl-cache	A self-contained task at the complex tier, with a budget of 30 agent steps. Starting from stubs, the agent implements a ttl_cache(maxsize, ttl_seconds) decorator that combines LRU eviction with time-to-live expiry — roughly 26 lines of target code — against 6 tests. A build-from-specification exercise rather than a bug hunt.	GPU-decode dominated across up to 30 turns, with no index or reranking, so CPU work is limited to pytest. The longest self-contained reasoning chain in the coding suite.

This benchmark is proprietary, and as such, we are not releasing its full code. We would be happy to discuss any questions you have. Please email info@principledtechnologies.com for more information.

Software versions we used

Table 5: Software versions we used in testing.

Component	Version
Operating system	Ubuntu 24.04.4 LTS (kernel 6.8.0-134-generic)
NVIDIA driver	595.71.05
CUDA	13.0.2
Container runtime	Docker 29.4.0
Python	3.11.15
Inference server	vLLM 0.23.0
Agent framework	LangGraph 1.2.6; LangChain 1.3.11 (langchain-core 1.4.8)
Benchmark tooling	AIPerf 0.7.0
Database	PostgreSQL 16.13
LLM (text inference)	Llama-3.3-70B-Instruct (BF16)
VLM (vision inference)	Qwen2.5-VL-3B-Instruct
LLM (CPU inference)	Qwen2.5-3B-Instruct (BF16 and FP32)

GPU topology

We used four GPUs per NUMA node for the eight-GPU standard configuration (two NUMA nodes). For the financial, financial MoE, and coding workloads, we ran two instances of Llama-3.3-70B-Instruct, each across four GPUs (one per NUMA node). For the manufacturing workloads, we ran three instances of Llama-3.3-70B-Instruct across two GPUs each and two instances of Qwen2.5-VL-3B-Instruct on one GPU each.

Setting up Ubuntu 24.04 LTS

1. Log into iDRAC.
2. Assign the Ubuntu Server 24.04 installation ISO to the virtual media.
3. Boot the system from the virtual ISO.
4. At the GRUB screen, select Try or Install Ubuntu Server, and press Enter.
5. Select English for the installation, and press Enter.
6. Choose English (US) for the keyboard layout, and click Done.
7. Select Ubuntu Server for the base installation, and click Done.
8. On the Network Connections screen, select the system's management NIC, and enter its IP address. Click Done.
9. On the Configure Proxy screen, click Done.
10. On the Configure Ubuntu archive mirror screen, click Done.
11. On the Guided storage configuration screen, select Use an entire disk, select the system's OS volume, select Set up this disk as an LVM group, and click Done.
12. On the Storage configuration screen, assign all free space to the default volume group, and click Done.
13. On the Confirm destructive action pop-up screen, click Continue.
14. On the Profile Setup screen, enter ubuntu for Your name, host-01 for Your server's name, ubuntu for Pick a username, and enter a password. Click Done.
15. On the Upgrade to Ubuntu Pro screen, select Skip for now, and click Continue.
16. On the SSH Setup screen, select Install OpenSSH server, and click Done.
17. On the Featured Server Snaps screen, click Done to start the installation.
18. When the installation completes, click Reboot Now.
19. After the system reboots, log in as user ubuntu.
20. Update the system packages:

```
sudo apt update
sudo apt upgrade
```

21. Install additional packages:

```
sudo apt install net-tools git
```

22. Reboot the server:

```
sudo shutdown -r now
```

This project was commissioned by Dell Technologies.

[Read the report ►](#)

Primary contributors

 **Tech:** Warren S. and Chris B.

 **Writing:** Jennifer V.

 **Design:** Cassandra O.

 **PM:** Greg C.

How we created this report

A PT team, which includes the contributors we've listed and others, created this report and performed the technical work behind it. In addition to our use of AI in testing, we used AI to draft and proofread sections of this report.



Facts matter.®

Principled Technologies is a registered trademark of Principled Technologies, Inc. All other product names are the trademarks of their respective owners. For additional information, review the science behind this report.

DISCLAIMER OF WARRANTIES; LIMITATION OF LIABILITY:

Principled Technologies, Inc. has made reasonable efforts to ensure the accuracy and validity of its testing, however, Principled Technologies, Inc. specifically disclaims any warranty, expressed or implied, relating to the test results and analysis, their accuracy, completeness or quality, including any implied warranty of fitness for any particular purpose. All persons or entities relying on the results of any testing do so at their own risk, and agree that Principled Technologies, Inc., its employees and its subcontractors shall have no liability whatsoever from any claim of loss or damage on account of any alleged error or defect in any testing procedure or result.

In no event shall Principled Technologies, Inc. be liable for indirect, special, incidental, or consequential damages in connection with its testing, even if advised of the possibility of such damages. In no event shall Principled Technologies, Inc.'s liability, including for direct damages, exceed the amounts paid in connection with Principled Technologies, Inc.'s testing. Customer's sole and exclusive remedies are as set forth herein.