signal65

NVIDIA DGX SPARK CASE STUDY:

RAG Inference by Day, Fine-tuning by Night

AUTHORS

Ryan Shrout
President & GM | Signal65

Ken Addison
Client Performance Director | Signal65

IN PARTNERSHIP WITH

NVIDIA

JUNE 2026

Introduction

The rapid maturation of large language models has created a new class of infrastructure challenge for small and mid-size organizations. Cloud-hosted inference APIs offer convenience but raise concerns around data sovereignty, recurring costs, and latency. Meanwhile, conventional desktop GPUs can lack the memory and compute to run production-quality models at the scale these organizations need. The NVIDIA DGX Spark™ occupies a deliberate middle ground as a desktop-form-factor AI supercomputer built on the Grace-Blackwell architecture with 128 GB of unified memory, designed to bring data-center-class AI capability into an office environment.

The DGX Spark addresses these challenges in a single desktop system a small team can own and operate. The dual-use model is central to that case. The same hardware that serves the team through the workday is put to fine-tuning overnight, so an expensive asset earns its keep around the clock instead of sitting idle outside business hours. For a four-to-eight-person group that cannot justify a dedicated MLOps function or a rack of discrete GPUs, DGX Spark is best understood not as a smaller version of rack-scale infrastructure but as an owned local AI computer sized for small-team deployment.

Key Takeaways

- One DGX Spark serves up to 8 users on a 30B model by day and fine-tunes a model by night, no data leaving the building.
- Support up to 4 users at 7.9s TTFT at ~39 tokens/s.
- Overnight LoRA training held ~425 tokens/s, finishing a 500-step run in about 45 minutes.
- Spread across 8 users, DGX Spark works out to <$600 per seat, one-time fixed cost.

This paper evaluates the DGX Spark as a dual-use AI workstation through a practical workflow we call the day/night cycle. During business hours, the system serves as a retrieval-augmented generation (RAG) chatbot, fielding concurrent queries from a small team against an ingested knowledge base. Overnight and on weekends, the same hardware pivots to fine-tuning a model with LoRA, incorporating new organizational knowledge and improving response quality over time. The two workloads never overlap; they share the same 128 GB memory envelope but use it for fundamentally different purposes.

Our goal is to characterize end-to-end performance at each stage of this cycle: how many concurrent users can the system serve with acceptable latency during RAG inference, and how quickly can it complete a meaningful fine-tuning run overnight? We present concrete benchmarks on both workloads using the Nemotron-3-Nano-30B-A3B model, document the practical setup requirements (including several non-obvious sharp edges), and assess the viability of the DGX Spark as a self-contained AI platform for teams of roughly four to eight people.

Hardware and Platform Overview

The NVIDIA DGX Spark is powered by the NVIDIA GB10 Grace Blackwell Superchip, which pairs a Grace ARM64 CPU with a Blackwell-architecture GPU in a unified memory configuration. The system has 128 GB of coherent, unified memory accessible to both processor and accelerator, eliminating the PCIe bottleneck that constrains discrete GPU configurations.

The desktop form factor is a defining characteristic. Unlike rack-mounted DGX systems, DGX Spark is designed for office deployment with standard power and cooling requirements. This positions it between cloud GPU instances (which offer elastic scaling but limited data locality) and traditional workstation GPUs (which typically provide smaller local memory pools). The 128 GB unified memory envelope is the key differentiator as it is large enough to serve a 30-billion-parameter model in NVFP4 for inference while retaining substantial headroom, and large enough to fine-tune that same model with LoRA in bf16 without running out of memory.

Specification	Value
SoC	NVIDIA GB10 (Grace-Blackwell)
CPU	ARM64 (Grace)
GPU Compute Capability	sm_121
Unified Memory	128 GB
Form Factor	Desktop
GPU Driver	580.159.03

Table 1: *DGX Spark hardware specifications*

Model Selection

Choosing the right model for this project required balancing several constraints. The model had to be large enough to deliver useful inference quality for RAG-augmented chat, but small enough to fine-tune on-device with LoRA.

The initial consideration was to use the full 120-billion-parameter Nemotron-3 variant, which would comfortably fit in 128 GB for inference. However, full and LoRA fine-tuning of a model of that size would exceed the available memory. While qLoRA would theoretically fit, Transformers does not currently support qLoRA fine-tuning for MoE models yet. At the other end of the spectrum, smaller models (sub-10B) would fine-tune quickly but would deliver noticeably weaker chat quality, undermining the value proposition of an on-premises AI workstation.

We settled on **Nemotron-3-Nano-30B-A3B**, a 31.8-billion-parameter hybrid architecture combining Mamba2 selective state-space layers with Transformer attention blocks in a Mixture-of-Experts (MoE) configuration with 128 experts. For inference, the model runs in NVFP4 quantization via vLLM, consuming approximately 18 GB of memory for weights alone, leaving substantial headroom for KV cache and larger context windows. For fine-tuning, we use bf16 LoRA (rank 16) targeting attention, MoE expert FFN, and Mamba SSM projections, which activates 441.9 million trainable parameters and peak at roughly 73 GB of memory, well within the 128 GB envelope.

This configuration leaves meaningful headroom in both modes. During inference, the remaining ~103 GB can be allocated to KV cache, supporting extended context windows that improve RAG retrieval quality. During fine-tuning, the ~48 GB of unused memory provides a buffer for larger batch sizes or longer sequence lengths if needed.Choosing the right model for this project required balancing several constraints. The model had to be large enough to deliver useful inference quality for RAG-augmented chat, but small enough to fine-tune on-device with LoRA.

Daytime Workload: RAG-Augmented Inference

Architecture

The inference stack consists of three components. vLLM serves the Nemotron-3-Nano-30B-A3B model in NVFP4 quantization, handling tokenization, batching, KV cache management, and the core forward pass.

Open WebUI provides the chat interface and RAG pipeline, including document ingestion, chunking, embedding, and vector retrieval. The two communicate over a local HTTP API, with Open WebUI prepending retrieved context to each user prompt before forwarding it to vLLM for generation.

For the knowledge base, we ingested 30 PDF documents of AI research papers sourced from arXiv into the built-in vector store in Open WebUI. These are real, full-length academic papers covering LLM and agent research, chosen to represent a realistic knowledge-intensive workload for a research or engineering team.

The choice of vLLM over alternatives was driven by concurrency requirements. Llama.cpp, while at times simpler to set up, can exhibit sharp performance degradation beyond three to four concurrent users.

Test Methodology

We conducted a concurrency sweep at four levels: 1, 2, 4, and 8 simultaneous users. Each concurrency level ran three iterations of the threaded launch (N × 3 samples per row), with the vLLM server warm-started once at the

beginning of the run. A warm-up call is issued before the user-count loop; its measurements are discarded. To avoid prefix-cache artifacts, the test script rotates through 12 distinct RAG queries targeting different papers in the collection, ensuring vLLM performs a fresh prefill on every request.

The test harness issues concurrent HTTP requests to the Open WebUI API with RAG enabled and streaming turned on (stream_options.include_usage=true). All timing metrics are measured client-side: TTFT is the interval from HTTP POST to the first streamed SSE chunk, and per-user throughput is computed as completion_tokens divided by decode time. Token counts are extracted from the final SSE usage payload emitted by vLLM. Results are aggregated into average, median (P50), and 99th percentile (P99) statistics across all samples at each concurrency level.

Results: Latency

Time-to-first-token is the primary latency metric for interactive chat because it determines how long a user waits before seeing any response. The following chart and table summarize TTFT across concurrency levels.

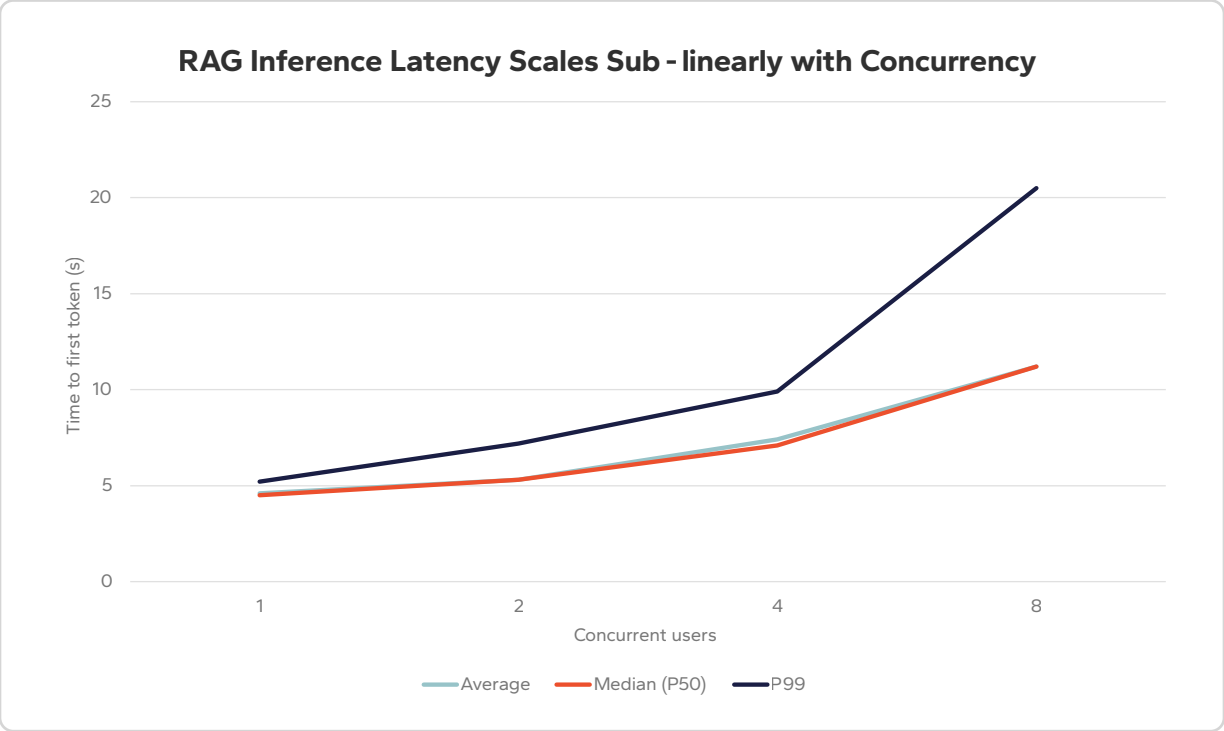

Figure 1: *Time-to-first-token scaling with concurrent RAG users*

Concurrent Users	Avg TTFT (s)	Median TTFT (s)	P99 TTFT (s)
1	4.6	4.5	5.2
2	5.3	5.3	7.2
4	7.4	7.1	9.9
8	11.2	11.2	20.5

Table 2: *Time-to-first-token scaling with concurrent RAG users*

Single-user TTFT of 4.6 seconds reflects the baseline cost of RAG retrieval, context assembly, and prompt prefill for a 30-billion-parameter model. As concurrency increases, TTFT grows sub-linearly; with only a modest increase from 1 to 2 users (5.3 s), scaling to 7.4 seconds at 4 users and 11.2 seconds at 8. This pattern suggests orderly request queuing rather than catastrophic resource contention. At 8 concurrent users, the average wait is about 11 seconds with a P99 tail of 20.5 seconds, which remains within acceptable bounds for a knowledge-retrieval chatbot where users expect multi-sentence responses.

Results: Throughput

Per-user generation throughput measures how quickly each individual receives tokens once generation begins.

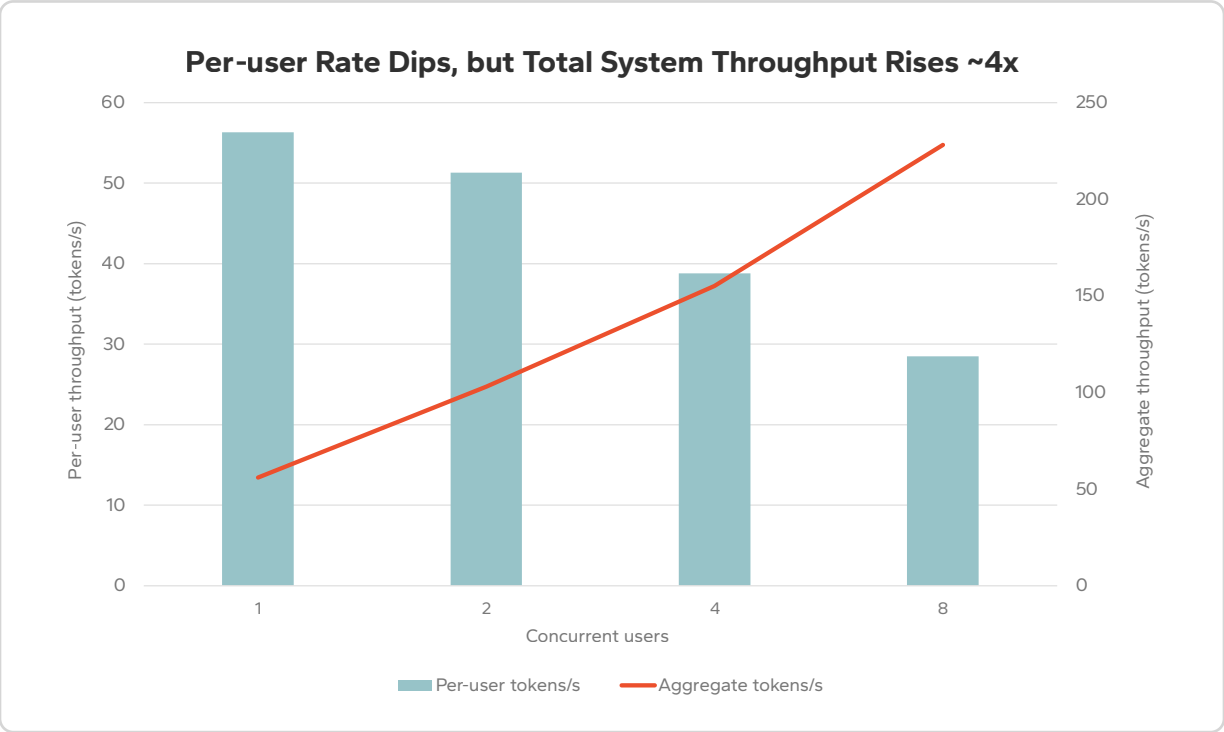

Figure 2: *Per-user token generation throughput with concurrent RAG users*

Concurrent Users	Avg tokens/s	Median tokens/s	P99 tokens/s
1	56.3	56.5	56.6
2	51.3	51.1	53.1
4	38.8	37.5	43.5
8	28.5	27.2	42.7

Table 3: *Per-user token generation throughput with concurrent RAG users*

At single-user load, the system generates 56.3 tokens per second, well above the threshold for comfortable real-time reading (roughly 10-15 tokens/s for displayed text). Throughput per user decreases as concurrency increases, as expected. The GPU's compute and memory bandwidth are shared across active generation

streams. At 4 concurrent users, each still receives roughly 39 tokens per second, which is fast enough for responsive chat. At 8 users, per-user throughput drops to about 28.5 tokens/s but remains comfortably above the readability threshold.

Notably, aggregate throughput (per-user rate multiplied by the number of users) increases with concurrency, reaching approximately 56 tokens/s at 1 user, 103 tokens/s at 2, 155 tokens/s at 4, and 228 tokens/s at 8. The system is doing substantially more total work under load, even as each individual user's experience degrades modestly. This is consistent with the continuous batching architecture in vLLM, which efficiently interleaves multiple generation streams on the GPU.

Discussion

The practical concurrency ceiling for a small-office deployment appears to be in the range of four to eight users. At four users, both latency and throughput remain strong at 7.4-second average TTFT and nearly 39 tokens/s per user deliver a responsive conversational experience. At eight users, the system remains functional but the 11-second average TTFT and 20.5-second P99 tail begin to push the boundaries of what feels interactive for a chatbot.

The extended KV cache is a key lever for RAG quality. With the model consuming roughly 18 GB for weights in NVFP4, the remaining memory can be allocated to KV cache, supporting longer context windows. Longer contexts allow the RAG pipeline to inject more retrieved chunks per query, improving answer quality at the cost of increased TTFT (since the model must prefill more tokens before generating). For deployments prioritizing answer quality over response speed, this trade-off is favorable since the DGX Spark memory envelope makes it possible without external trade-offs.

Put in practical terms, the daytime results describe a system a small team would find usable rather than merely functional. At four concurrent users, answers begin in about seven seconds and then stream faster than most can read them, which is the responsiveness people now expect from a hosted assistant, delivered here with no data leaving the building and no per-query meter running. Because latency and throughput taper gradually rather than collapsing under load, a team can grow into the appliance and sense the limits approaching well before they turn disruptive, which is the behavior that makes a fixed, owned device practical to plan around.

Nighttime Workload: LoRA Fine-Tuning

Training Configuration

Fine-tuning uses the **Unsloth** framework running inside the NVIDIA PyTorch container (nvcr.io/nvidia/pytorch:26.04-py3) on the DGX Spark. We chose Unsloth over raw PyTorch for its optimized LoRA dispatch and padding-free packing, and over NeMo Curator/Customizer for simplicity.

The training run uses bf16 LoRA at rank 16. The full hyperparameter configuration is summarized below:

Parameter	Value
Framework / optimizer	Unsloth; adamw_torch_fused
Method	bf16 LoRA, rank 16
Target modules	Attention (q/k/v/o); MoE expert FFNs (up/down_proj, experts + shared); Mamba SSM in/out projections
Trainable parameters	441.9M (1.38% of 31.8B)
Base model	nvidia/NVIDIA-Nemotron-3-Nano-30B-A3B-BF16
Load flags	trust_remote_code=True; attn_implementation="eager"; unsloth_force_compile=True
Memory optimizations	Gradient checkpointing (Unsloth mode); padding_free=True (FlashAttention varlen)
Dataset	yahma/alpaca-cleaned (51,760 examples)
Max sequence length	2,048 tokens
Batch size	8 per-device, no gradient accumulation (effective 8)
Token counting	trainer.state.num_input_tokens_seen (real, non-padded)

Table 4: *Fine-tuning configuration*

Token throughput is measured from real (non-padded) token counts to avoid overstating utilization on a short-sequence dataset.

Performance Results

We ran a 100-step benchmark to characterize throughput and training dynamics:

Metric	100 Steps (Including Warmup)	Steady-State (Steps 11+)
Tokens/sec (real)	376.07	~425
Step time	4.26 s	~3.77 s
Samples/sec	1.877	~2.12
Peak Memory	68.706 GB	68.706 GB
Training loss	1.05 -> 0.88	Plateau in 0.83-0.99 range

Table 5: *Per-user token generation throughput with concurrent RAG users*

Total wall clock for the 100-step run was 7.10 minutes of training plus 7.1 minutes of one-time model loading. Tokens per second is measured using real token counts from trainer.state.num_input_tokens_seen (160,266 total tokens across 100 steps). Prior runs reported a padded ceiling (steps × batch × seq_len / runtime); on Alpaca the real count is lower, making the two numbers not directly comparable.

Step-Time Warmup Profile

The Nemotron-3-Nano-30B-A3B hybrid Mamba2 architecture requires JIT compilation of selective scan kernels and Triton operators on first use. This creates a pronounced warmup phase.

Figure 3: *Step-time warmup profile showing JIT compilation amortization*

Step	Time (s)	Notes
1-5	~13.6	Triton JIT compile (cold cache)
6-10	~3.62	JIT amortized; steady state begins
11-15	~3.69	Steady state
16-20	~3.94	Steady state
50	~3.72	Steady state
100	~3.72	Steady state

Table 6: *Step-time warmup profile showing JIT compilation amortization*

The step-time profile exhibits a single JIT plateau followed by steady state. The first 5 steps average ~13.6 s/step as triton kernels compile for the Mamba SSM paths, then converge to a steady state of ~3.77 s/step for the remaining 95 steps. The 376 tokens/s mean over 100 steps includes this warmup penalty; steady-state throughput after step 5 is approximately 425 tokens/s. Runs shorter than ~10 steps will significantly underreport steady-state capability.

Loss Curve

Training loss drops cleanly from 1.05 at step 5 to approximately 0.85 by step 15, confirming that the LoRA path is actively training and not running on a frozen graph. Loss plateaus in the 0.83-0.99 range for the remainder of the run, ending at 0.88 at step 100 with a full-run average of 0.91. A longer 500-step run (estimated at approximately 45 minutes) would provide a deeper loss curve without meaningfully changing steady-state throughput.

Optimization Headroom

The current configuration still leaves room for improvement. Three specific optimizations could push throughput higher without any hardware changes: explicit sequence packing (packing=True) on top of padding-free mode to fill the token budget more densely; and increasing per-device batch size further (the 69 GB peak against 121 GB available leaves significant headroom).

The Compounding Value of Nightly Fine-Tuning

The throughput numbers describe what an overnight run costs but its value lies in what the model knows the next morning. Each night of fine-tuning folds the organization's own material, new documents, corrected answers, internal terminology, and the patterns of how the team actually phrases its questions, into the weights that serve the next business day. Over weeks, that steadily turns a capable general model into one fluent in the specifics of a particular business.

The expected gains are in quality rather than speed. Fine-tuning does not change the model architecture or its parameter count, so the daytime throughput and latency measured above hold regardless of how many nights of adaptation have accumulated. What improves is the relevance and accuracy of the responses. The system makes fewer missteps on in-house jargon, produces more consistent formatting and tone, and handles the recurring question types a given team brings to it more reliably. For a RAG deployment the two halves reinforce each other, with retrieval supplying current facts and the adapted model better able to interpret and present them. Because the cycle is incremental and reversible, a night that improves the model is kept and one that degrades it is rolled back to the prior adapter at no cost beyond the electricity.

The Day/Night Transition

The practical workflow for switching between modes is straightforward. At the end of the business day, the vLLM inference server is stopped, freeing the GPU memory. The fine-tuning container is then launched, loads the model for training, and runs for the desired number of steps. Upon completion, the fine-tuned checkpoint is

exported (or merged with the base model weights), and vLLM is restarted with the updated model for the next business day.

The two workloads have non-overlapping memory profiles. Inference uses approximately 18 GB for the NVFP4 model weights plus KV cache, while fine-tuning peaks at 68.7 GB including optimizer states and activations. There is no need (or benefit) to run them simultaneously. The transition overhead is dominated by model loading, approximately 7.1 minutes for fine-tuning setup and a similar duration for vLLM startup, which is negligible relative to an overnight training window.

This workflow is amenable to automation via cron jobs or systemd timers that stop the inference service at a scheduled time, launch the training container, and restart inference when training completes (or at a fixed morning time). Checkpoint management and rollback (reverting to the previous LoRA adapter if the new one degrades quality) can be handled with simple file versioning.

Conclusions

The DGX Spark viably serves a dual role for small teams. During business hours, it delivers responsive RAG-augmented chat for four to eight concurrent users using a 30-billion-parameter model, with per-user throughput remaining above 28 tokens per second even at the higher end of that range. Overnight, the same hardware achieves approximately 425 real tokens per second of steady-state fine-tuning throughput with LoRA, enabling meaningful model adaptation on real datasets within a single overnight window.

The 128 GB unified memory envelope is the key enabler for this dual-use workflow. It provides enough capacity to serve the model in NVFP4 with generous KV cache allocation during the day, and enough to fine-tune with bf16 LoRA (peaking at 69 GB) at night, without any configuration changes to the model itself. Only the serving and training frameworks swap in and out.

The concurrency sweet spot for interactive use is four users, where average TTFT is 7.4 seconds and per-user generation runs at nearly 39 tokens/s. Eight users remain functional, but push latency into a territory where the experience feels less conversational. For teams larger than eight, either a second Spark or a move to a rack-mounted configuration would be warranted.

Setup friction remains the primary practical challenge. The Mamba kernel source builds are a hard requirement for acceptable fine-tuning performance (without them, step time increases roughly fivefold), and several version conflicts between container-bundled libraries and current framework releases require manual workarounds. These issues are specific to the May 2026 software stack and are expected to be resolved as the ecosystem matures.

The day/night pattern examined here is only one way to put the appliance to work. The same 128 GB envelope supports a range of other small-team configurations. It could run as an always-available local endpoint serving several models at once, a development sandbox for prototyping against frontier-class open weights before any cloud commitment, an overnight batch engine for embeddings and large-scale document processing, or an on-prem inference node for latency-sensitive and air-gapped settings where data cannot leave the premises. The dual-use workflow is simply the case that most clearly shows what pairing large unified memory with a desktop power budget makes possible.

For a small organization, the value of this configuration is significant. A four-to-eight-person team can own a single device that handles both halves of a practical AI deployment, customer-facing inference during the day and model improvement at night, without a cloud subscription, a server room, or a dedicated infrastructure hire. The data never leaves the building, the cost is a one-time purchase rather than a metered bill that grows with usage, and the same hardware earns its keep around the clock instead of sitting idle outside business hours. That combination did not have a clean answer before a desktop appliance with this much unified memory existed. For the many teams that have wanted production-grade AI on their own terms but could not justify the cloud spend or the operational overhead, the dual-use DGX Spark is a credible and genuinely new starting point.

Appendix A: Software Stack and Ecosystem

Component	Version
Container	nvcr.io/nvidia/pytorch:26.04-py3
Python	3.12.3
PyTorch	2.12.0a0+0291f960b6.nv26.04
CUDA Runtime	13.2
Triton	3.5.0
Transformers	5.9.0
TRL	0.26.1
PEFT	0.19.1
Unsloth	2026.5.5
bitsandbytes	0.49.2
mamba-ssm	2.3.2.post1 (source-built)
causal-conv1d	1.6.2.post1 (source-built)
vLLM	(NVFP4 serving, container 26.04)
Open WebUI	(RAG + chat interface)

Table 7: *Complete software stack*

The stack spans multiple partner ecosystems, including the NVIDIA container registry and model zoo, the Unsloth optimization framework, the vLLM serving engine, and Open WebUI for the user-facing RAG interface. This breadth is both a strength (it demonstrates real-world interoperability across the ecosystem) and a current friction point (version conflicts between components require manual patching, as documented in Section: Optimization Headroom).

The primary maturity gap as of May 2026 is ARM64/sm_121 wheel availability. Several critical libraries (mamba-ssm, causal-conv1d) require source builds on this platform, and the container's bundled library versions do not always align with the latest releases of frameworks like PEFT. These are expected to improve as the DGX Spark ecosystem matures and more prebuilt binaries target the Grace-Blackwell platform.

Important Information About this Report

CONTRIBUTORS

Ryan Shrout
President and GM | Signal65

Ken Addison
Client Performance Director | Signal65

PUBLISHER

Ryan Shrout
President and GM | Signal65

INQUIRIES

Contact us if you would like to discuss this report and Signal65 will respond promptly.

CITATIONS

This paper can be cited by accredited press and analysts, but must be cited in-context, displaying author's name, author's title, and "Signal65." Non-press and non-analysts must receive prior written permission by Signal65 for any citations.

LICENSING

DISCLOSURES

Signal65 provides research, analysis, advising, and consulting to many high-tech companies, including those mentioned in this paper. No employees at the firm hold any equity positions with any companies cited in this document.

IN PARTNERSHIP WITH

NVIDIA

ABOUT SIGNAL65

Signal65 is an independent research, analysis, and advisory firm, focused on digital innovation and market-disrupting technologies and trends. Every day our analysts, researchers, and advisors help business leaders from around the world anticipate tectonic shifts in their industries and leverage disruptive innovation to either gain or maintain a competitive advantage in their markets.

signal65

signal65

CONTACT INFORMATION
Signal65 I signal65.com