Whitepaper | Benchmark

RNGD meets Backend.AI: AI Inference Infrastructure for the Era of Sovereign AI

Author:
Lablup | Jinho Heo, Kyujin Cho, Joongi Kim, Jeongkyu Shin
FuriosaAI | Hyunsik Choi, Jeehoon Kang, Jongwook Kim

July 2026 v1.0

Index

Executive Summary

For organizations that operate generative AI services in-house, selecting the right model is only part of the problem. Deciding which infrastructure will serve that model reliably is an equally critical design challenge. Inference is an always-on workload that persists for the entire lifetime of a service, and its operational cost depends not only on compute resources but also on power consumption and datacenter facility constraints. Some recent GPUs require new cooling methods and draw substantial power, which can force additional capital investment or changes to facility operations. When evaluating inference infrastructure, power density and deployment compatibility deserve as much attention as raw compute performance.

This whitepaper examines a configuration in which FuriosaAI's datacenter inference accelerator, RNGD, is operated through Lablup's AI infrastructure operating platform, Backend.AI. RNGD is an inference-only accelerator designed for air-cooled datacenter deployment at 180 W TDP per card. Backend.AI is a software layer that allocates, isolates, and serves heterogeneous AI accelerators from a single platform.

The combination of RNGD and Backend.AI also carries relevance from a Sovereign AI perspective. Pairing an inference chip designed domestically with an open-source-based platform developed domestically addresses several practical requirements of AI sovereignty: air-gapped operation where data does not cross national borders, supply chain optionality free from foreign vendor lock-in, code-level transparency for auditing, and operational self-reliance.

Lablup and FuriosaAI conducted a controlled benchmark serving the Qwen3-32B model at FP8 precision on RNGD and a comparably specified accelerator. At 256 concurrent requests, four RNGD cards achieved approximately 95% of the throughput of four NVIDIA RTX PRO 6000 Blackwell Server Edition cards (2,336 vs. 2,467 tok/s), while total accelerator power consumption remained at 56-70% of the comparison configuration. Converted to throughput per watt, RNGD was approximately 1.3-1.5x higher across all measured concurrency levels, and time to first token (TTFT) was shorter for RNGD across the entire range.

Based on these benchmark results, this whitepaper evaluates which organizations and use cases are best suited to an infrastructure configuration that combines a dedicated inference accelerator with a heterogeneous orchestration platform, considering performance, power efficiency, and operational factors.

1. Introduction: Infrastructure Constraints in the Age of Inference

1.1 LLM Inference as an Always-on Workload

Inference has become the central variable determining the economics of AI infrastructure. Enterprise AI investment has long been concentrated on model training, but as LLM-powered services spread into routine business functions such as internal assistants, customer-facing interactions, and document processing, the operational center of gravity is shifting toward inference. Training and inference differ not only in workload characteristics but also in cost structure. Training resembles capital expenditure (CapEx): it concludes when a model development project ends. Inference, by contrast, is operational expenditure (OpEx) that scales with service volume. The competitiveness of AI infrastructure ultimately depends on how consistently an organization can achieve per-unit efficiency and scalability at the inference stage, rather than on model performance alone.

1.2 Power and Cooling Constraints

The most straightforward way to expand inference capacity is to add more AI accelerators, but in practice, power and cooling constraints make this difficult. Much of the hardware released recently assumes liquid cooling for high-density configurations. Deploying such equipment in facilities designed for air cooling requires retrofit work, which introduces both additional lead time and cost. Given that a significant share of enterprise and public-sector server rooms in Korea are air-cooled, hardware that requires liquid cooling is not an easy option for organizations planning inference infrastructure. Recent high-performance GPU servers demand anywhere from 7 kW to 15 kW of peak power per server. In existing facilities with limited power headroom, these requirements can become a bottleneck for accelerator expansion.

Offloading AI infrastructure to the cloud instead of owning it is an alternative, but the limitations are clear. In regulated industries such as finance, government, and healthcare organizations, sending data to external environments is often prohibited outright, making air-gapped on-premises infrastructure effectively the only option. More recently, the discussion has expanded beyond data residency to Sovereign AI, which concerns control over the entire infrastructure and technology stack. Sovereign AI is discussed further in Section 1.4.

1.3 Inference-only Accelerators and the Need for Operational Software

Inference-only accelerators are hardware designed specifically to address these constraints. Rather than performing general-purpose GPU computation, they are purpose-built to process the inference stage with greater speed and efficiency. By excluding the broad computational scope required for training, such as backpropagation and optimizer state management, these accelerators optimize their design for inference-stage latency, throughput, and power efficiency.

RNGD follows this design philosophy. By shedding general-purpose capability, it can handle the same serving workload at lower power consumption. In environments where large-scale inference services run continuously, this difference in power efficiency translates directly into cumulative cost savings.

However, even if an inference-only accelerator enables efficient serving, the hardware's efficiency cannot be realized at the operational level unless the accelerator integrates naturally with the organization's existing infrastructure. The practical adoption of inference-only accelerators therefore depends on whether software exists that can manage accelerators in the same way it manages existing ones.

1.4 Sovereign AI: From Option to Requirement

Sovereign AI is an approach that places the core components of an AI system, including data, models, infrastructure, and operations, under the control of a nation or organization. As dependency on specific foreign supply chains increases on both the accelerator hardware and operational software sides, risks such as supply instability, export restrictions, and price volatility grow in proportion. With global supply chain uncertainty increasing and geopolitical considerations around security gaining weight, organizations are moving to take direct control over a wide range of factors: data storage location, model deployment and updates, access permissions, operational logs, incident response, and supply chain selection.

In this context, when an organization adopts inference infrastructure, whether that infrastructure can be operated stably under its own control becomes an evaluation criterion alongside performance and cost. Backend.AI and RNGD are, respectively, an operational platform developed in Korea and an inference accelerator designed in Korea. Their combination enables an infrastructure configuration that addresses the practical requirements of Sovereign AI: domestic supply chain availability, air-gapped operation, and code-level auditability. This whitepaper examines how this combination integrates with existing GPU infrastructure and what performance and efficiency characteristics it demonstrates under real inference workloads, supported by benchmark data.

2. Solution Overview

To address the power and facility constraints and the lack of operational software discussed in Section 1, Lablup and FuriosaAI present RNGD and Backend.AI as a unified configuration. On the NXT RNGD Server equipped with eight RNGD cards, users can compile models using FuriosaAI's compiler and serve them through Furiosa-LLM. Adding Backend.AI, an enterprise AI operations platform, provides organization-level operational capabilities including resource allocation, scheduling, usage metering, and model serving endpoint management.

3. Backend.AI: Scalable Token Serving on Heterogeneous Accelerators

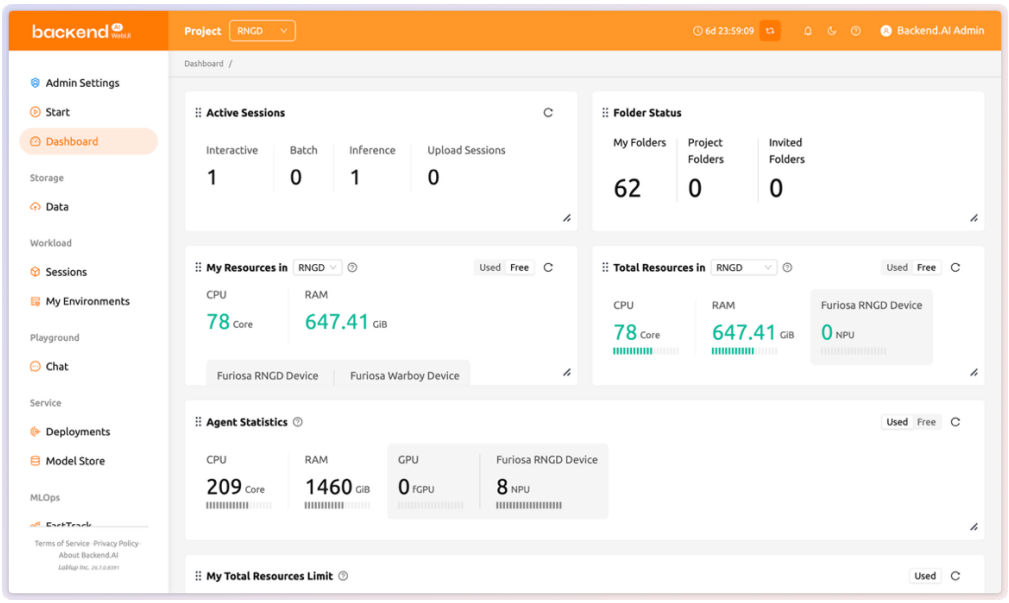


Figure 1. Backend.AI WebUI Dashboard

Backend.AI is an enterprise AI infrastructure operations platform designed to serve LLM tokens at scale across heterogeneous accelerators, covering both development and production serving. It is designed to manage heterogeneous AI accelerators of different types within a single operational framework. Backend.AI provides an execution foundation for efficiently distributing and controlling resources in multi-user environments. By abstracting the complex elements of AI infrastructure at the operational

layer, such as hardware configuration, framework dependencies, per-user resource allocation, and policy enforcement, Backend.AI provides an environment where developers and data scientists can focus on model development and execution. It is not tied to any specific framework, supporting a range of programming languages including Python, R, C/C++, Java, and Rust, as well as AI libraries such as PyTorch, Megatron-LM, Furiosa-LLM, and vLLM. With support for diverse deployment environments, Backend.AI can be used across on-premises, cloud, and hybrid configurations with a consistent experience.

This section walks through the core components of Backend.AI in sequence: sessions as the unit of workload execution, the Sokovan orchestrator for resource management, resource abstraction for uniform handling of heterogeneous accelerators, multi-tenancy for organization-scale operations, the plugin architecture for accelerator extensibility, and model serving and operations capabilities.

3.1. Workload Unit Abstraction: Sessions

In Backend.AI, every computational task is represented as a unit called a session. A session is an execution request that bundles a resource specification (number of accelerators, CPU cores, memory), a container image, and data to mount. Backend.AI materializes this request as a container. Sessions are classified into three types based on their purpose.

Sesson Type	Purpose	Lifecycle Characteristic
Interactive	Development / Experiment (Jupyter, IDE, Terminal)	User connects directly; subject to resource reclamation based on idle detection
Batch	Training / batch jobs	Session terminates on script completion; resources returned automatically
Inference	Model Serving	Exposes an externally accessible endpoint; scales based on accelerator utilization and inference metric

Table 1. Session types supported by Backend.AI

Even in clusters configured exclusively for inference, production environments often run interactive sessions for model updates or periodic batch jobs alongside serving workloads. The platform must apply distinct lifecycle policies per session type, such as idle resource reclamation, automatic termination, and persistent resource reservation, to operate accelerators efficiently.

Backend.AI manages the full session lifecycle in discrete stages, recording state transitions from resource allocation pending (PENDING) through image preparation, container creation, workload execution (RUNNING), and resource release (TERMINATED). Operators can track which session is at which stage and why it remains there. For multi-node sessions, multiple kernels are created according to the resource requirements and connected through a dedicated isolated network, allowing distributed inference or distributed training to be managed as a single unit of work.

3.2 Sokovan Scheduling Pipeline and Resource Reservation

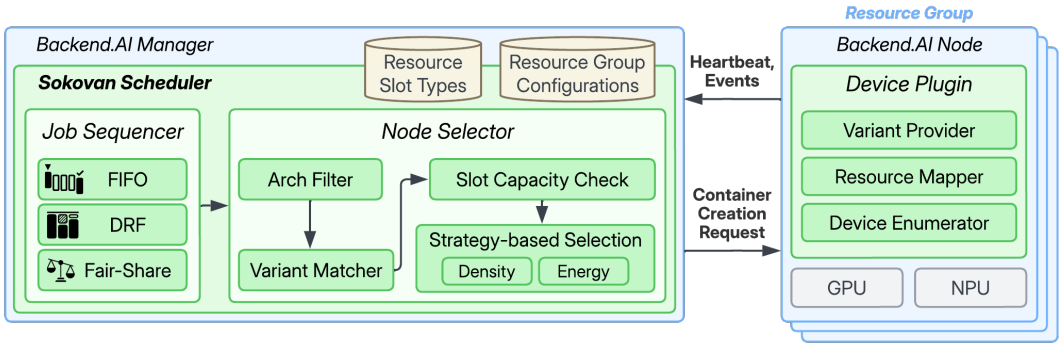


Figure 2. Sokovan Orchestrator of Backend.AI

Session placement onto nodes is handled by the Sokovan orchestrator. Sokovan is a container orchestrator independently designed by Lablup for large-scale AI and GPU workloads. It regulates job density and priority through a two-tier scheduler operating at both the cluster level and the node level. Because it supports multi-node, multi-tenant-aware scheduling, Sokovan focuses on improving resource utilization and execution efficiency in environments where multiple users and jobs coexist in parallel.

Sokovan provides scheduling that is aware of accelerator and workload characteristics. When multiple session execution requests enter the queue, Sokovan determines the order in which they are processed. Operators can select the scheduling policy that fits their environment. A suitable method can be selected among the FIFO (First-In-First-Out) method, which follows the first-in, first-out principle; the DRF (Dominant Resource Fairness) method, which assigns lower priority to users with higher resource occupancy; and the Fair Share method, which applies time decay based on past usage.

Policy	Behavior
FIFO (First-In-First-Out)	Processes requests in arrival order
DRF (Dominant Resource Fairness)	Deprioritizes users with the highest per-resource occupancy to prevent resource monopolization
Fair Share	Applies time decay to cumulative past usage, prioritizing users who have consumed relatively fewer resources

Table 2. Scheduling policy types

In addition, a preemption policy can optionally be applied: when a high-priority session is waiting, resources from a running lower-priority session are reclaimed and reassigned. This guarantees that urgent workloads or higher-priority jobs can run without queuing even under resource scarcity, which is advantageous in environments such as inference services where uninterrupted resource availability is required.

Sokovan's scheduling policies are applied per resource group, a logical partition that divides the cluster into independent policy units (see Section 6.1). Before placing a session, Sokovan's Validator checks tenant concurrent session limits, quotas, inter-session dependencies, and queue depth limits, filtering out requests that cannot be placed immediately. For requests that pass validation, Sokovan searches for eligible nodes and selects one according to the placement strategy assigned to each resource group. Two strategies are available per resource group: bin-packing, which consolidates workloads onto fewer nodes to reduce fragmentation, and spreading, which distributes load evenly across nodes.

Resource reservations in Sokovan are processed as integrity-guaranteed updates, so resources allocated to a user are used exclusively by that session. Double-assignment of the same accelerator to two sessions (overcommit) is prevented by design.

3.3 Resource Abstraction and Container Execution

Backend.AI's resource abstraction layer is what enables accelerators from different vendors to be handled in a uniform manner. The platform represents all resources as units called slots, which are divided into count-based slot types (CPU cores, number of accelerators) and byte-based slot types (memory).

Allocating abstracted resources to isolated containers is the responsibility of the Backend.AI Agent, which resides on each accelerator node. The Agent manages the lifecycle of each accelerator and its associated containers, and collects metrics such as utilization, power draw, and temperature, reporting them to the control plane. When assigning AI accelerators to containers, the Agent takes NUMA topology into account. For training and inference workloads that use multiple accelerators simultaneously, grouping accelerators within the same NUMA node has a direct impact on performance, and the Agent applies this optimization by default.

When creating a session, users can select a framework and version, such as Furiosa-LLM or PyTorch, as a container image. Because environments are isolated at the session level, different teams can run different version combinations on the same cluster in parallel. Backend.AI injects the runtime components needed for container execution as read-only mounts, so there is no need to bundle Backend.AI components inside the image in advance. This means official SDK images distributed by vendors can be used as session images without modification.

3.4 Multi-tenancy

Backend.AI manages resource usage through a multi-layered tenant hierarchy composed of domains, projects, and users. This structure makes it possible to implement complex resource management policies across multiple organizational levels. Independent resource limits can be set at each layer, and organization-level and individual-level policies can be applied in combination. For example, an operator can allocate 16 RNGD cards to Department A while restricting each individual within that department to a maximum of 4 cards concurrently. When combined with the scheduling policies described in Section 3.2, such as FIFO, DRF, and Fair Share, time-based imbalances can also be prevented. For instance, a GPU resource group might use FIFO to suit long-running training jobs, while an RNGD resource group might use Fair Share for situations where inference sessions from multiple teams compete for resources. Policies can be tailored per resource group to match accelerator characteristics and workload profiles. The configuration and operation of resource groups is covered in Section 6.1.

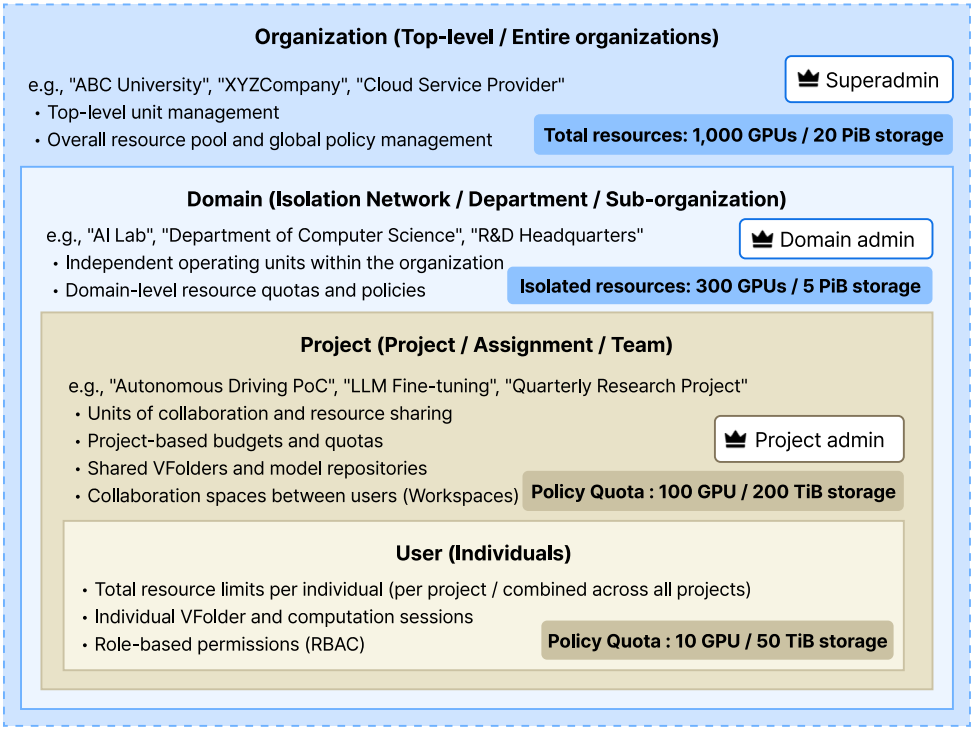


Figure 3. Tenant layer of Backend.AI

Tenant Layer	Corresponding Organizational Unit	Representative Policy Items
Domain	Company / Organizations (top-level isolation boundary)	Domain-level resource policies
Project	Team / Department	Project quotas, total storage capacity
User	Individual Account	Per-user quotas, folder count limits, concurrent session limits

Table 3. Tenant layer of Backend.AI

Backend.AI also provides mechanisms for terminating or reclaiming allocated resources based on usage patterns. Resources are reclaimed by the scheduler when a policy-defined period elapses after the last detected traffic, when CPU, memory, or accelerator utilization remains below a configured threshold, or when a session exceeds its maximum allowed execution time.

Idle Detection Criterion	Behavior
No network activity detected	Automatic reclamation after a configured period since last traffic
Low utilization	Automatic reclamation when CPU, memory, or accelerator utilization stays below threshold
Session lifetime exceeded	Automatic reclamation when the per-type maximum execution time is reached

Table 4. Backend.AI idle activity detection criteria

Safeguard logic is also in place to prevent false positives. Sessions without allocated accelerators are excluded from accelerator utilization checks. Newly created sessions are given a grace period, so they are not reclaimed during initial startup. Reclamation thresholds can be managed individually through per-user resource policies, enabling differentiated operation by pricing tier or team.

3.5 Accelerator Plugin Architecture

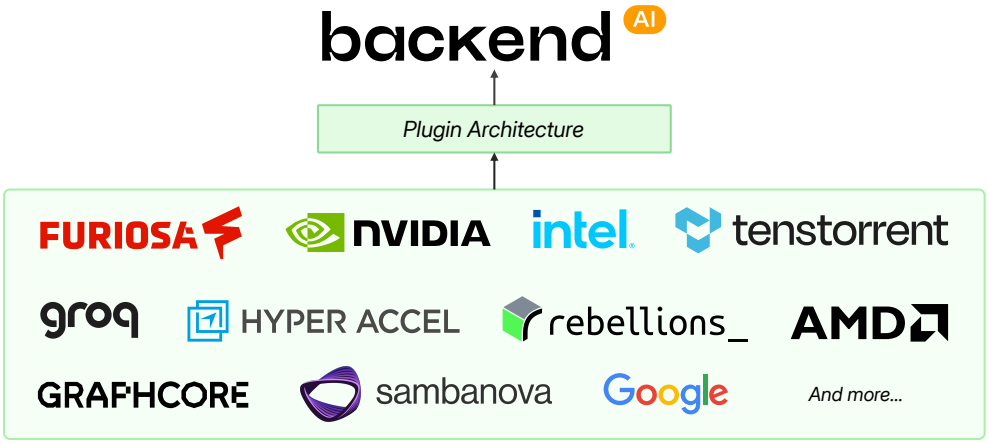


Figure 4. Heterogeneous accelerators supported by plugin architecture

Backend.AI adopts a plugin architecture for heterogeneous accelerator support. By separating accelerator support into plugins rather than adding it directly to the core, new accelerators can be onboarded quickly regardless of the product release cycle, and new accelerators can be added without affecting existing accelerator environments already in operation.

Through this architecture, Backend.AI supports not only the FuriosaAI RNGD but also the first-generation FuriosaAI Warboy Vision NPU, along with AI accelerators from vendors including NVIDIA, Intel, AMD, Google, Graphcore, Tenstorrent, Rebellions, and HyperAccel.

3.6 Model Serving and Operations

Inference sessions are managed as units within a model service endpoint. Each endpoint distributes traffic across multiple Inference sessions (replicas), issues API tokens, and automatically adjusts the replica count based on metrics such as average response time and queue length. The model service endpoint exposes the HTTP APIs provided by Inference sessions to external clients, enabling direct use of OpenAI- and Anthropic-compatible APIs offered by inference libraries such as Furiosa-LLM, vLLM, and SGLang. This allows organizations to build LLM services in air-gapped environments using standard LLM client libraries, with application-side integration requiring only an endpoint address change.

Three deployment strategies are available for model version updates: rolling, blue/green, and canary.

Deployment Strategy	Description
Rolling Deployment	Sequential replacement of replicas
Blue/Green Deployment	Simultaneous operation of old and new versions, followed by a full cutover
Canary Deployment	Routing a subset of traffic to the new version for pre-production validation

Table 5. Deployment strategies available in Backend.AI

As of July 2026, rolling deployment is available, with blue/green and canary deployment strategies planned for release within 2026. For rolling deployment, parameters such as rollout ratio can be freely adjusted

3.6.1 VFolder

Backend.AI integrates diverse storage backends, including NFS, CephFS, VAST Data, and WekaFS, through an abstraction layer called VFolder, which is mounted into sessions. Storage quotas are managed at the user and project level. On the security side, user code is isolated through container sandboxing based on system call whitelisting, and containers run without privileged mode by default. All features operate identically in air-gapped environments with no external network connectivity.

3.6.2 Backend.AI Agent and all-smi

The Backend.AI Agent collects metrics required for serving operations, including utilization, memory, power, and temperature at the session, node, and accelerator level. Collected metrics can be viewed in

real time per tenant through the dashboard. When used together with all-smi, an open-source metric exporter developed by Lablup, metrics from mixed heterogeneous accelerator environments including RNGD can be queried in a unified format regardless of vendor.

3.6.3 Continuum Router and Continuum Hub

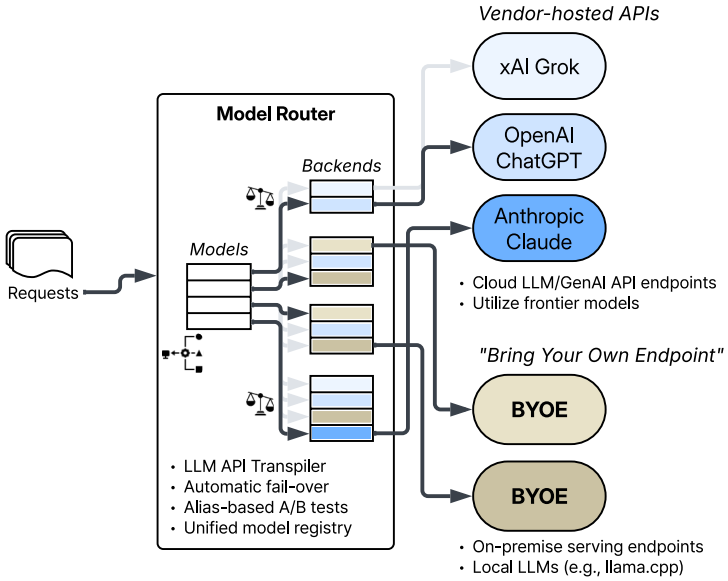


Figure 5. Continuum Router Architecture

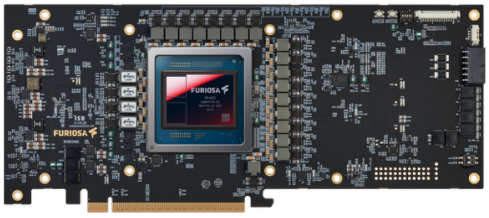
Continuum Router is an AI routing layer that sits in front of multiple serving endpoints and inference engines. It distributes incoming requests to appropriate targets and provides smart routing, guardrails, and failover capabilities in an integrated manner. Continuum Router supports real-time conversion between API formats. Depending on user configuration, open models and self-hosted model services can be exposed in OpenAI Responses API or Anthropic Messages API format, enabling coding agents such as Claude Code and Codex to be connected to the model of choice.

Continuum Hub is an upper management layer that oversees multiple Continuum Router instances. It tracks which endpoints and engines each user request passed through and how requests were distributed and enables construction of a token factory system where per-department and per-user token consumption and costs can be monitored from a single control plane. For detailed Continuum Router functionality, refer to the following manual.

4. FuriosaAI RNGD: An Accelerator Designed for Inference

RNGD (Renegade) is FuriosaAI's second-generation AI accelerator, designed for inference of large language models (LLMs) and multimodal models, and entered mass production in 2026. An NPU (Neural Processing Unit) is a dedicated accelerator specialized for neural network computation. Unlike CPUs, which handle general-purpose computation, or GPUs, which cover a broad range of parallel workloads, an NPU focuses on executing the tensor operations required for AI model inference with high power efficiency. RNGD is manufactured on the TSMC 5nm process and is equipped with 48 GB of HBM3 memory.

4.1 RNGD Product Type and Specifications



Architecture	Tensor Contraction Processor
Process Node	TSMC 5nm
Operating Clock	1.0 GHz
BF16	256 TFLOPS
FP8	512 TFLOPS
INT8	512 TOPS
INT4	1,024 TOPS
Memory	HBM3 48 GB, 1.5 TB/s bandwidth
On-chip SRAM	256 MB
Card Power	180 W
Form Factor	PCIe Gen5 x16, passive cooling
Security	Secure Boot with Root of Trust, ECC Memory Support

Table 6. RNGD Specifications

4.1.1 NXT RNGD Server

Figure 6. FuriosaAI NXT RNGD Server

The NXT RNGD Server is a datacenter inference server housing eight RNGD cards in a 4U chassis. It provides a total of 384 GB HBM3 memory with an aggregate memory bandwidth of 12 TB/s, and delivers up to 4 PFLOPS of compute performance at FP8 precision within a system power envelope of approximately 3 kW. The NXT RNGD Server is designed for air cooling, allowing inference clusters to be deployed in existing facilities without liquid cooling infrastructure. This combination of low power requirements and air-cooled design enables organizations to adopt the NXT RNGD Server with minimal changes to their existing datacenter environment and server installation setup.

4.2 Tensor Contraction Processor Architecture

The compute core of RNGD is a proprietary architecture that FuriosaAI calls the TCP (Tensor Contraction Processor). Tensor contraction refers to the generalization of matrix multiplication to higher-dimensional tensors, and modern transformer-based LLMs rely heavily on this class of operations.

Many existing accelerators implement fixed-size matrix multiplication units in hardware, with the compiler partitioning and mapping a model's tensor operations to fit these unit dimensions. This is efficient when the operation shape matches the unit size exactly, but in LLM serving environments where batch size and sequence length vary per request, underutilized fragments increase and hardware utilization drops. TCP addresses this problem by treating the tensor contraction operation itself as the

fundamental hardware primitive, with the compiler explicitly planning tensor partitioning and data movement to map operations onto the compute units. Because only the mapping plan needs to change when the operation shape varies, this approach is better suited to maintaining utilization under variable serving loads.

4.3 Memory Subsystem and Multi-card Configuration

RNGD's memory subsystem is designed for LLM inference workloads. External to the chip, two HBM3 modules are integrated via CoWoS-S packaging, providing 48 GB of capacity and 1.5 TB/s of bandwidth. Inside the chip, 256 MB of SRAM allows frequently reused tensor fragments to be kept on-chip, reducing external memory accesses and increasing compute throughput. The 48 GB memory capacity per card is sufficient to load a 30B-class model quantized to FP8 on a single card. Furiosa-LLM, FuriosaAI's serving framework, supports tensor parallelism across up to eight cards, so larger models can be distributed across multiple cards. Recent Sovereign AI models such as Solar-Open and K-EXAONE can also be run in this configuration.

4.4 Software Stack: Furiosa SDK and Furiosa-LLM

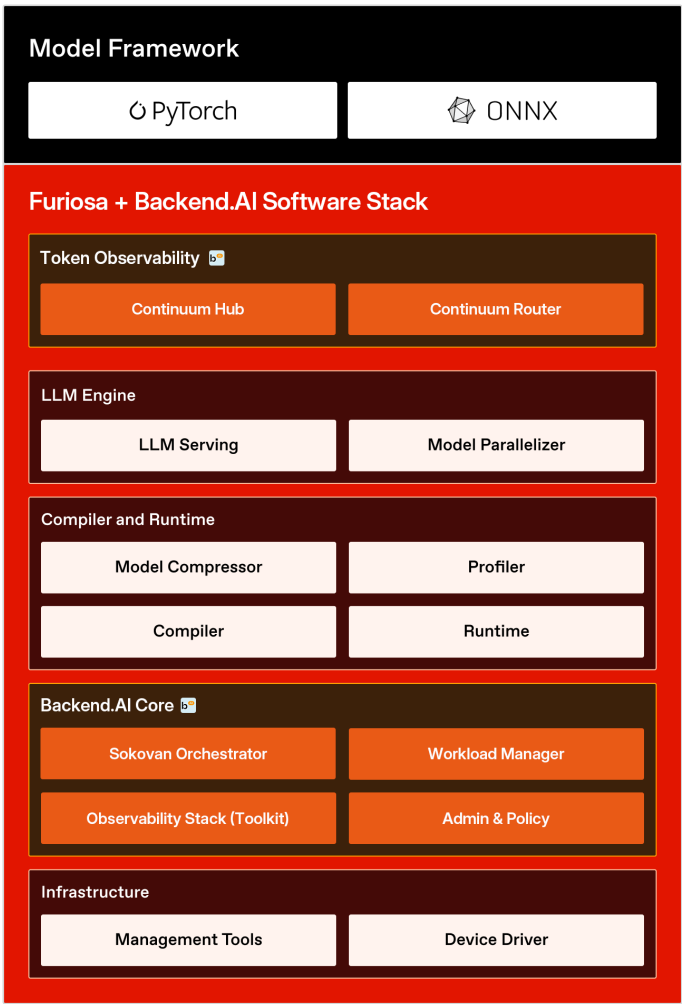


Figure 7. Software stack combination of Furiosa and Backend.AI

FuriosaAI provides a dedicated software stack for deploying RNGD across a range of environments. Furiosa SDK is the integrated software stack for model deployment and serving on RNGD, designed with both portability and operational efficiency in mind.

Furiosa Compiler

Furiosa Compiler optimizes model graphs and generates execution binaries for the NPU. Depending on the model architecture and application requirements, a single model can be compiled into multiple executables. Existing PyTorch code can be converted into an RNGD execution plan without significant modification, and models published on Hugging Face Hub can be built directly without a separate conversion step.

Furiosa-LLM

Furiosa-LLM is a high-performance inference engine for large language models (LLMs) and multimodal models. It supports vLLM-compatible APIs, text and multimodal serving, generative and pooling models, efficient KV cache management with PagedAttention, radix-tree prefix caching for shared prompt prefix reuse, data/tensor/pipeline parallelism across multiple NPUs, various decoding algorithms, a Prometheus metrics endpoint for observability, and integration with Hugging Face models and Hub. Furiosa-LLM natively supports model quantization to reduce memory footprint, inference latency, and power consumption. Supported post-training quantization (PTQ) formats include BF16 (W16A16), FP8 (W8A8), MXFP4, and NVFP4.

Furiosa-LLM uses TCL (Tensor Contraction Language) as its kernel framework for model support. TCL is a declarative language that, like the RNGD TCP (Tensor Contraction Processor) architecture itself, treats tensor contraction as its fundamental operation. It expresses hardware-native operations directly at a high level while allowing explicit optimization hints at the model implementation stage. Because kernels are modularized into building blocks shared across models, new models can be ported efficiently by combining existing blocks. All models added after the Furiosa-LLM 2026.2.0 release are implemented on TCL.

* In the second half of 2026, FuriosaAI plans to release TCL alongside Furiosa-LLM, enabling users to add models independently.

4.5 Extending RNGD Model Serving Capabilities with Lablup's MLxcel

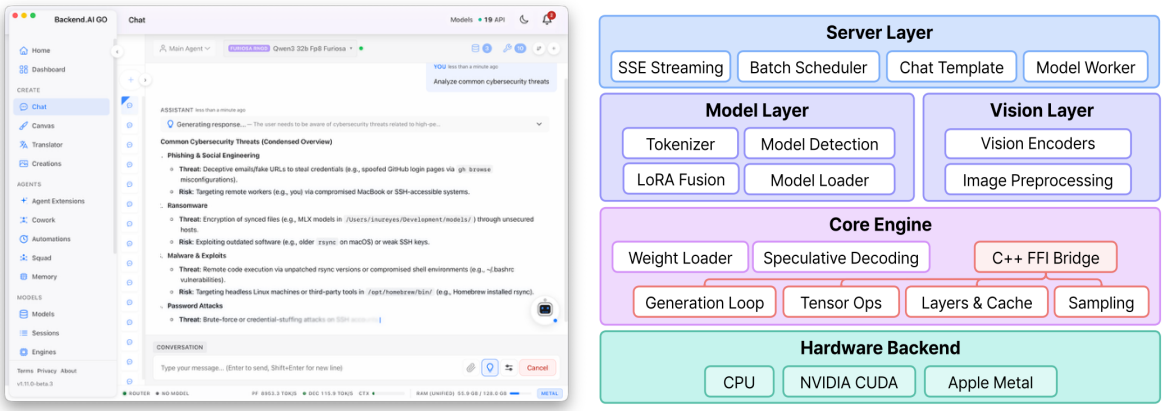


Figure 8. Qwen 32B FP8 model running on Lablup's AI:GO via RNGD & MLxcel architecture

Lablup's inference engine MLxcel (pronounced "ML-accel") is also preparing RNGD support. MLxcel is a unified inference engine that provides inference, batch scheduling, KV cache management, and multi-node large-scale model serving for a variety of model types, including LLM, VLM, dLLM, and STT, through an OpenAI-compatible API. By separating the per-hardware compute layer in its architecture, MLxcel adds OpenXLA-RNGD support, making the model ecosystem already available in MLxcel accessible on RNGD as well.

* RNGD support in MLxcel is scheduled for Q3 2026.

5. Integrated Architecture: Operating RNGD on Backend.AI

5.1 Lifecycle Management

Training and inference demand different hardware characteristics, and environments that mix accelerator pools are becoming more common — for example, separating prefill and decode phases onto different accelerators within the inference stage itself. Backend.AI manages these environments in a consistent manner through the resource group concept. Operators register nodes to a resource group, and from that point Backend.AI manages the entire lifecycle, from hardware resources through session creation and termination.

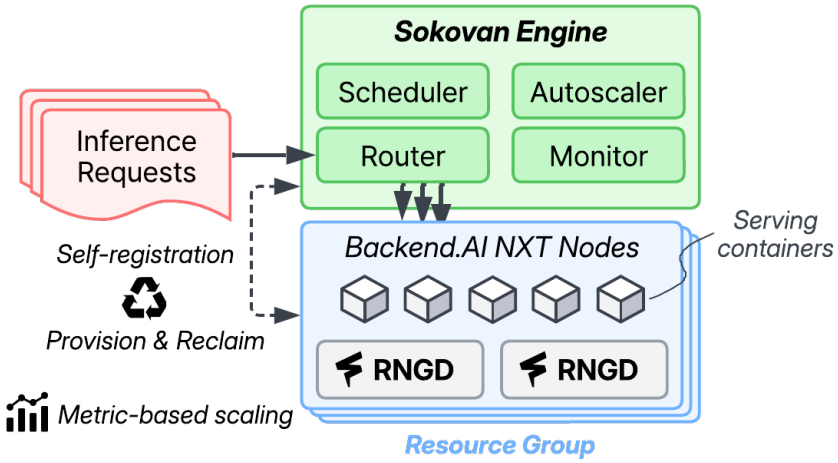


Figure 9. Resource lifecycle at Backend.AI Sokovan

- **Resource registration:** On each NXT RNGD Server node, the Backend.AI Agent detects RNGD devices through the FuriosaAI plugin. Detected devices are registered with the Manager as NPU slots and are then treated identically to other accelerator resources in the dashboard and scheduler.
- **Session request:** Users create a session by specifying the number of accelerators needed, along with CPU, memory, and a container image. Request is passed to the scheduler as a single execution unit.
- **Scheduling:** Sokovan selects a placeable node based on tenant quotas and current resource availability. NUMA placement is also considered during this process. Once resources are confirmed, the session is created as a container with only the assigned RNGD devices exposed. Other accelerators on the same node are assigned independently to separate sessions.

- **Serving:** Inside the container, Furiosa-LLM loads the model and performs inference using tensor parallelism. Backend.AI creates an endpoint through which the session can be reached, and external requests are routed to the session via this endpoint. Traffic distribution and autoscaling are configured through endpoint settings.

- **Operations, monitoring, and reclamation:** Resource usage and power metrics for running sessions are collected through Backend.AI and aggregated per tenant. When a session terminates, its allocated resources are returned immediately and made available for other requests.

5.2 Host Configuration: Preparation That Affects Performance

To reliably achieve full accelerator performance, host-level settings must also be reviewed. The same accelerator can yield different measured performance depending on host configuration, so the following settings are recommended for production environments. The content may change with future updates, so for the most up-to-date information, please refer to the following Furiosa documentation.

Settings	Description	Expected Effect
Disable PCIe ACS	Configures card-to-card P2P communication to bypass the root complex	Reduced communication latency in multi-accelerator tensor parallel environments
Enable hugepages	Reduces page management overhead when mapping large memory regions	Stabilized performance for inference workloads with frequent memory access
NUMA Alignment	Assigns CPU cores to the same NUMA node as the accelerator through Backend.AI's NUMA-aware placement	Reduced memory access latency

Table 7. Host settings for efficient operation

6. Benchmark: Operating RNGD on Backend.AI

6.1 Benchmark Overview and Test Environment

Accelerator	FuriosaAI RNGD x4 (NXT RNGD Server)	NVIDIA RTX PRO 6000 Blackwell Server Edition x4
Model	furiosa-ai/qwen3-32b-fp8 (Furiosa-optimized)	Qwen/Qwen3-32B-FP8
Serving Engine	Furiosa-LLM 2026.3.0, Tensor parallelism 4	vLLM 0.22.0 (26.05.29 release), Tensor parallelism 4
Platform	Backend.AI 26.4.6	Bare-metal
Workload	1,024 Input Token / 1,024 Output Token / Concurrency 1~256	

Table 8. Benchmark test environment and workload

Four FuriosaAI RNGD accelerators and four NVIDIA RTX PRO 6000 Blackwell accelerators were used for the benchmark. The model selected was the 32B variant of Qwen3, an open model widely used in production serving environments. Both systems were measured serving the same model at FP8 precision with tensor parallelism 4. Because enabling radix-tree prefix caching would cause results to vary depending on the prompt sharing ratio, prefix caching was disabled on both sides to compare baseline accelerator performance. Benchmark was performed in June 2026.

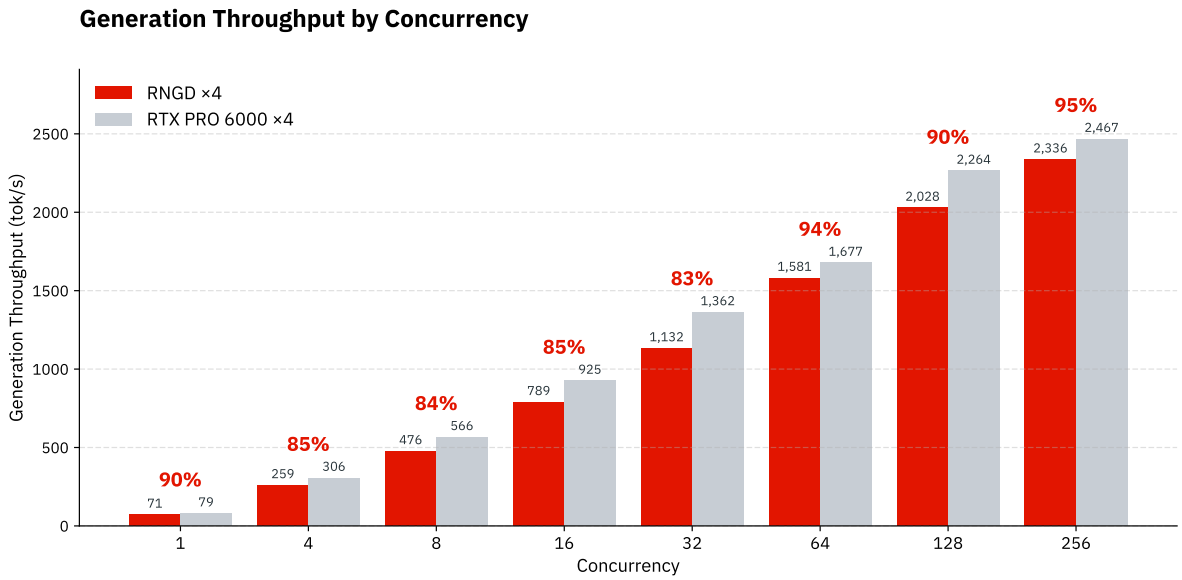
Metric	Definition	Significance
Throughput (tok/s)	Tokens generated per second	Overall system processing capability
TTFT (Time-To-First-Token)	Latency from request to first token generated	User-perceived responsiveness
TPOT (Time-Per-Output-Token)	Average interval between tokens after the first	Sustained generation speed

Table 9. Performance evaluation metrics

The performance metrics used in this evaluation are as follows. Throughput, measured in tokens per second (tok/s), represents the overall processing capacity of the system. TTFT (Time to First Token) is the

latency from request submission to the generation of the first token, and directly reflects user-perceived responsiveness. TPOT (Time Per Output Token) is the average interval between tokens after the first, used to evaluate sustained generation speed. Power consumption was measured as the aggregate draw across four accelerator cards; actual total power difference at the system level may be larger.

6.2 Throughput Approaching the Comparison Under Equivalent Conditions

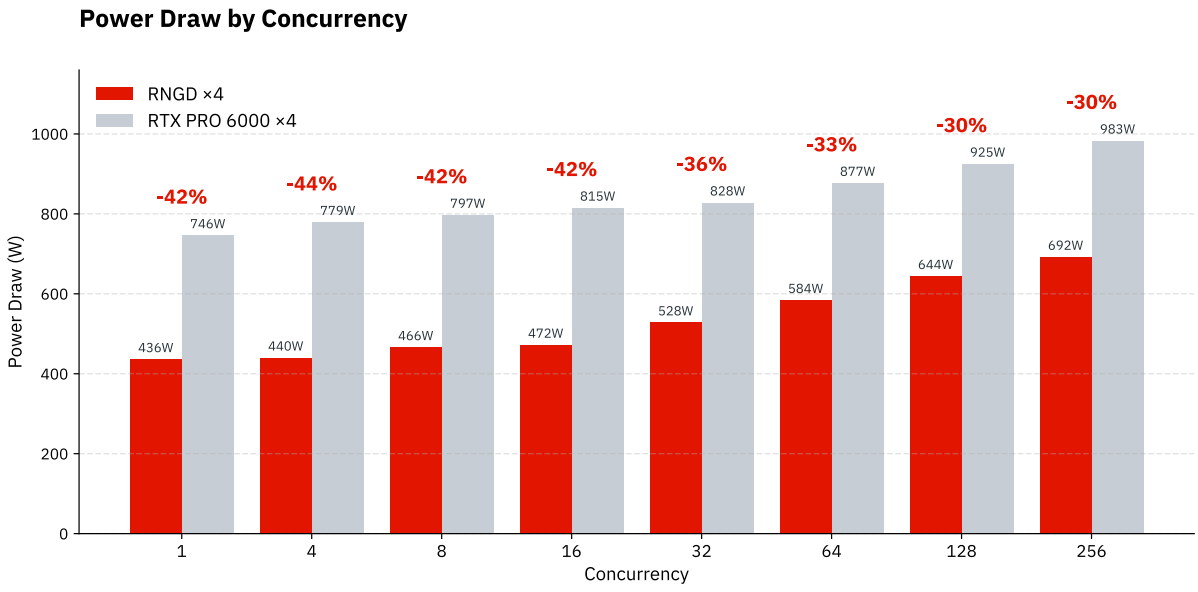


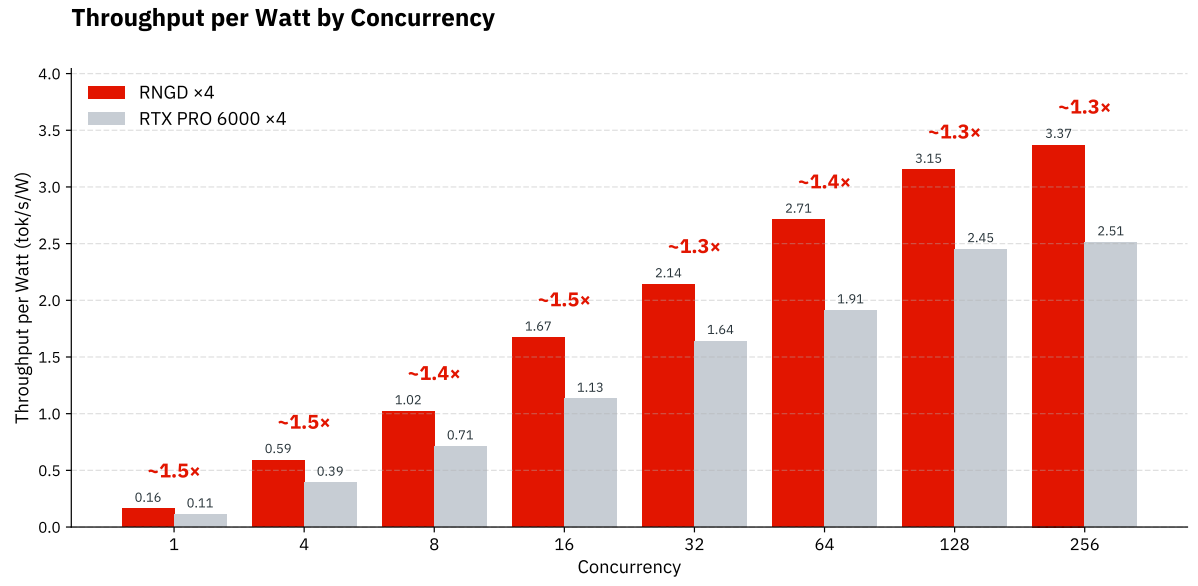
Concurrency	RNGD x4	RTX PRO 6000 x4	Ratio
1	71 tok/s	79 tok/s	90%
4	259 tok/s	306 tok/s	85%
8	476 tok/s	566 tok/s	84%
16	789 tok/s	925 tok/s	85%
32	1,132 tok/s	1,362 tok/s	83%
64	1,581 tok/s	1,677 tok/s	94%
128	2,028 tok/s	2,264 tok/s	90%
256	2,336 tok/s	2,467 tok/s	95%

Table 10. Throughput comparison by concurrency level

At 256 concurrent requests, four RNGD cards recorded 2,336 tok/s, reaching approximately 95% of the throughput of four RTX PRO 6000 cards (2,467 tok/s). Across other concurrency levels, the ratio remained within the 83–95% range. Looking at the progression by concurrency band, RNGD started at 90% at concurrency 1, the gap widened to 83–85% in the mid-load range, then narrowed again to 90–95% at high concurrency levels of 64 and above. From concurrency 1 to 256, RNGD throughput scaled 32.8x, while RTX PRO 6000 scaled 31.2x. In the highest band, RNGD throughput increased by 15.2% compared to 9.0% for RTX PRO 6000, indicating that RNGD retains relatively more headroom under heavy load.

6.3 Power Efficiency: Approximately 1.3–1.5x Throughput per Watt





Concurrency	Power (RNGD / RTX PRO 6000)	Throughput per Watt (RNGD / RTX PRO 6000)	Multiple
1	436W / 746W	0.16 / 0.11 tok/s/W	~1.5x
4	440W / 779W	0.59 / 0.39 tok/s/W	~1.5x
8	466W / 797W	1.02 / 0.71 tok/s/W	~1.4x
16	472W / 815W	1.67 / 1.13 tok/s/W	~1.5x
32	528W / 828W	2.14 / 1.64 tok/s/W	~1.3x
64	584W / 877W	2.71 / 1.91 tok/s/W	~1.4x
128	644W / 925W	3.15 / 2.45 tok/s/W	~1.3x
256	692W / 983W	3.37 / 2.51 tok/s/W	~1.3x

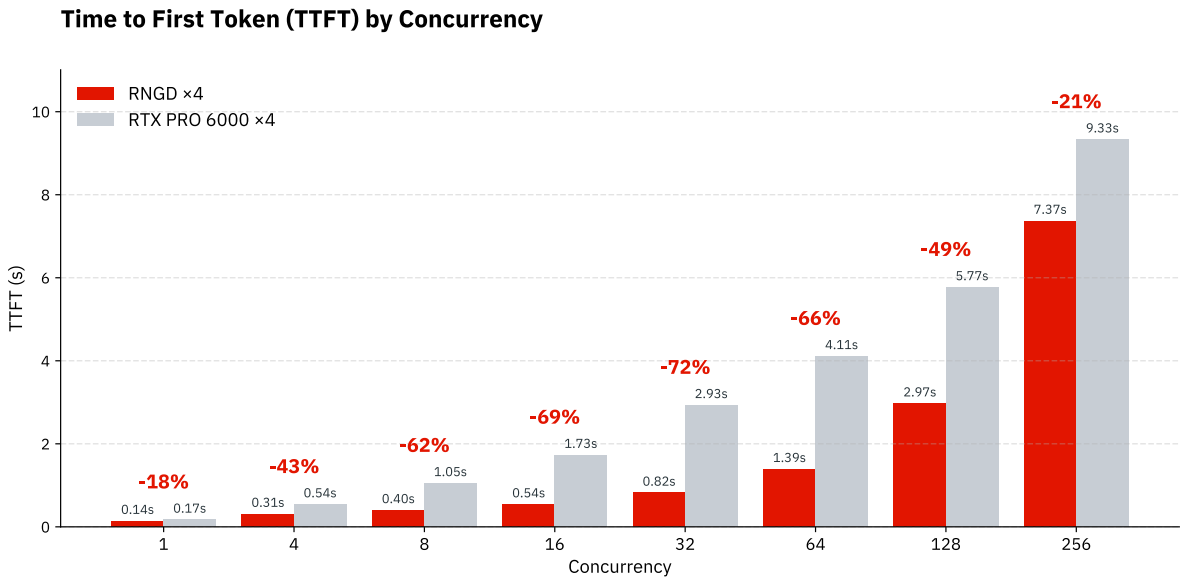
Table 11. Power consumption comparison by concurrency level

Aggregate power draw for four RNGD cards measured 56–70% of four RTX PRO 6000 cards across all concurrency levels. Converted to throughput per watt, RNGD was approximately 1.3–1.5x more efficient across all concurrency levels. The throughput gap between the two configurations was 5–17%, while the power gap was 30–43%, which is what produces the power efficiency advantage.

The power profile under increasing load is also worth examining. As concurrency rose from 1 to 256, aggregate RNGD power increased from 436 W to 692 W, corresponding to 109–173 W per card. Even at peak load, per-card consumption stayed within the 180 W TDP, meaning that using TDP as the upper bound for power planning is unlikely to underestimate actual draw. For RTX PRO 6000, aggregate power rose from 746 W to 983 W, with per-card draw ranging from 187 W to 246 W.

The power difference is especially pronounced at low load. In the 1–16 concurrency range, RNGD held between 436–472 W while RTX PRO 6000 drew 746–815 W. This characteristic translates into cumulative power cost differences for services with high traffic variability and extended low-load periods. Inference services that run continuously across multiple servers for high availability often spend a significant share of their operating time at low average load, making power efficiency at low concurrency an important consideration.

6.4 Fast Time to First Token (TTFT)



Concurrency	TTFT (RNGD / RTX PRO 6000)	TPOT (RNGD / RTX PRO 6000)
1	0.14s / 0.17s	13.9ms / 12.5ms
4	0.31s / 0.54s	15.1ms / 12.5ms
8	0.40s / 1.05s	16.4ms / 12.9ms
16	0.54s / 1.73s	19.7ms / 15.9ms
32	0.82s / 2.93s	27.4ms / 20.7ms
64	1.39s / 4.11s	38.9ms / 33.7ms
128	2.97s / 5.77s	59.3ms / 52.1ms
256	7.37s / 9.33s	99.2ms / 99.1ms

Table 12. TTFT & TPOT comparison

TTFT: RNGD recorded shorter TTFT across all concurrency levels. RTX PRO 6000 exceeded 1 second at concurrency 8 and rose to approximately 3 seconds at concurrency 32, while RNGD stayed under 1 second through concurrency 32. In the 8–64 concurrency range, the difference was roughly 2x. In production environments where many requests are processed simultaneously, TTFT is the metric users perceive most directly, so this gap can have a meaningful impact on service quality.

TPOT: At low concurrency levels, RTX PRO 6000 showed shorter TPOT, and this trend held through concurrency 128. At concurrency 256, both systems converged to approximately 99 ms. However, in the low-concurrency range where RTX PRO 6000 has the advantage, both systems generate tokens at 50–80 tokens per second, so the perceptible difference in a streaming user experience is limited.

7. Use Case Scenarios and Adoption Guide

7.1 Scenarios by Workload

Based on the benchmark results, the following are environments where RNGD can be put to practical use.

Scenario 1: Internal LLM assistant and chatbot services

In internal assistant environments used by tens to hundreds of concurrent users, time to first response (TTFT) and streaming stability can be primary evaluation criteria. Sampling the concurrency 8–64 range from the benchmark, RNGD recorded TTFT of 0.40–1.39 seconds, while the comparison configuration ranged from 1.05–4.11 seconds, a substantial gap.

Scenario 2: Bulk document processing and batch inference

Batch workloads such as document summarization, classification, and information extraction typically prioritize throughput, making higher-throughput accelerators advantageous. In the benchmark, RNGD achieved approximately 90–95% of the comparison configuration's throughput at the 128–256 concurrency range while consuming roughly 70% of the power. Deploying inference infrastructure on RNGD therefore increases the volume of work that can be processed within a given power budget. Because batch processing is governed by total completion time rather than individual request latency, TPOT differences are often not a binding constraint. The scaling characteristics at high concurrency can directly reduce batch processing time.

In Backend.AI, Batch and Inference sessions can be placed in the same resource group, enabling time-of-day resource sharing that improves workload operational efficiency.

Scenario 3: Adding an inference tier to an existing GPU cluster

In environments where an existing GPU cluster is already operated through Backend.AI, RNGD can be introduced by adding a resource group without separating onto a different platform. A common configuration keeps training and fine-tuning workloads in the existing GPU resource group while isolating inference services into a dedicated RNGD resource group. This reduces situations where training GPUs are occupied by always-on serving, producing a clear separation benefit in resource utilization. On the operational side, the same dashboard, account structure, and usage metering remain in place. From the user's perspective, the only change is an additional resource type available when creating a session, yet operational efficiency improves.

7.2 Recommended Adoption Environments

- Datacenters that need to expand LLM inference capacity within power and cooling constraints
- Organizations that want to run inference workloads alongside existing GPUs with separate placement
- Public sector and enterprise organizations evaluating domestic AI semiconductor adoption
- Organizations building LLM on air-gapped or on-premises to meet Sovereign AI requirements

8. Conclusion and Recommendations

Backend.AI is a platform designed to manage AI infrastructure, including heterogeneous accelerators, within a single operational framework. It can flexibly place resources across multi-node environments and provides tenant-level isolated execution environments. Its session-level recovery structure maintains operational consistency while allowing continued scaling.

RNGD is an inference-only accelerator that delivers performance and power efficiency advantages on top of this operational structure. As the benchmarks in this whitepaper confirmed, throughput per watt was approximately 1.3–1.5x higher across all concurrency levels, and TTFT was shorter across the entire range. For always-on inference services in particular, power efficiency differences are directly reflected in cumulative operating costs. With Backend.AI, RNGD accelerators can be placed and managed in the same way as existing GPU resources.

The key advantage of this combination is that existing infrastructure does not need to be rebuilt from scratch. Organizations can expand their operational scope by adding an accelerator type to a running environment, separate training and inference workloads through resource group isolation, and rebalance resource utilization across the cluster as a whole. The fact that both the accelerator and the platform are designed and developed domestically also makes this configuration a practical option for environments that require data sovereignty, supply chain autonomy, and operational control.

To evaluate an inference infrastructure built with Backend.AI and RNGD firsthand, contact Lablup and FuriosaAI.

9. About

Lablup Inc.

Lablup is a software company that develops Backend.AI, an AI infrastructure operations platform. Since its founding in 2015, Lablup has addressed core challenges in enterprise AI infrastructure, including GPU virtualization, unified management of heterogeneous accelerators, and multi-tenant resource operations. Backend.AI operates consistently across on-premises, cloud, and air-gapped environments, and supports more than 12 types of AI accelerators, including FuriosaAI, from a single platform. For more information, visit https://www.lablup.com/

FuriosaAI

FuriosaAI develops next-generation AI semiconductors for datacenters and enterprises. The company aims to make powerful AI technology accessible to more people by building more sustainable AI computing. For more information, visit http://furiosa.ai