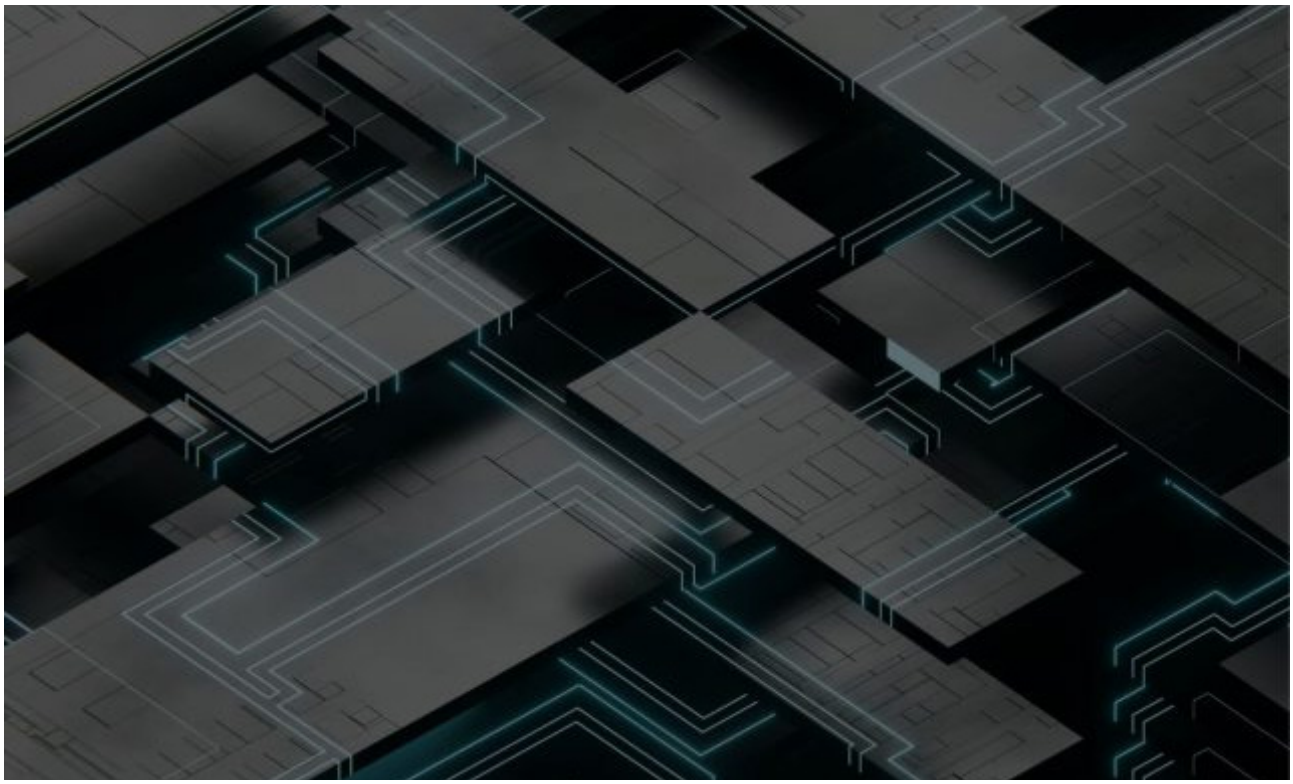
rebellions_

LLM Serving with NPU:

Re-engineered, Built for Scale and Efficiency

Aug 24, 2025



Introduction

The era of large language models (LLMs) has shifted focus from simply running models to serving them efficiently, reliably, and at scale. Success now lies in system-level design: decomposing complex inference workflows and reconstructing them into hardware-optimized execution paths with minimal overhead.

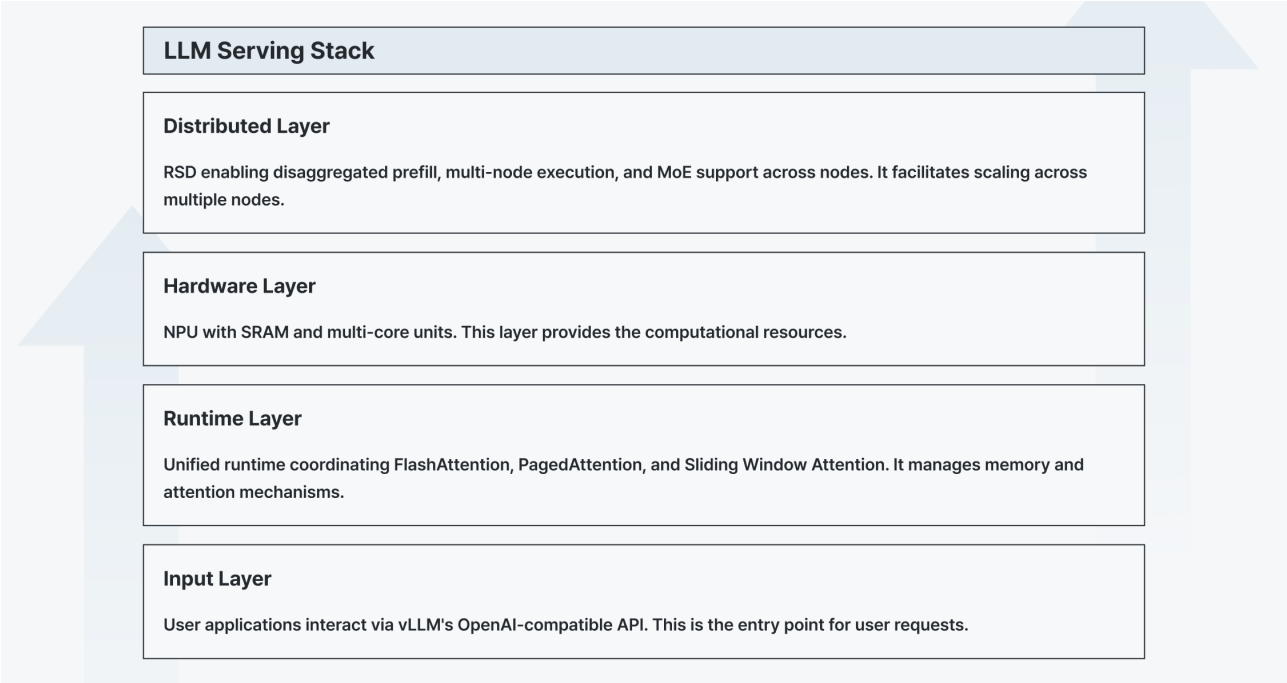
Modern LLMs rely on advanced attention mechanisms such as FlashAttention, PagedAttention, and Sliding Window Attention to achieve high throughput, low latency, and long context support. Frameworks like vLLM abstract these mechanisms for GPU-based serving, leveraging CUDA for fast memory access and parallel computation. However, reliance on GPUs increases power and costs, limiting portability to other architectures such as NPUs.

Rebellions tackled this challenge by optimizing these mechanisms specific to our NPU architecture. We adapted FlashAttention, PagedAttention, and Sliding Window Attention to Rebellions NPU's memory hierarchy and compute model, integrating them into a unified runtime for seamless operation. Through vLLM RBLN, a vLLM plugin for RBLN NPUs, we enable applications to leverage these optimizations without modifying existing vLLM-based workflows, delivering NPU-native performance and efficiency.

Optimizing LLM Serving for Rebellions NPU

Rebellions extends vLLM's capabilities by re-engineering core attention mechanisms to run natively on our NPU. FlashAttention and PagedAttention were co-designed with the hardware and integrated into a unified runtime. Causal masking is handled at the kernel level, and scaled-dot product attention (SDPA) is supported through optimized paths. This architecture enables consistent execution across attention types, tailored for performance on Rebellions hardware.

The Rebellions NPU remains fully compatible with vLLM's API, enabling users to deploy models with minimal changes while transparently benefiting from hardware-level optimizations in memory access, scheduling, and kernel execution.



[Figure 1. LLM Serving Stack]

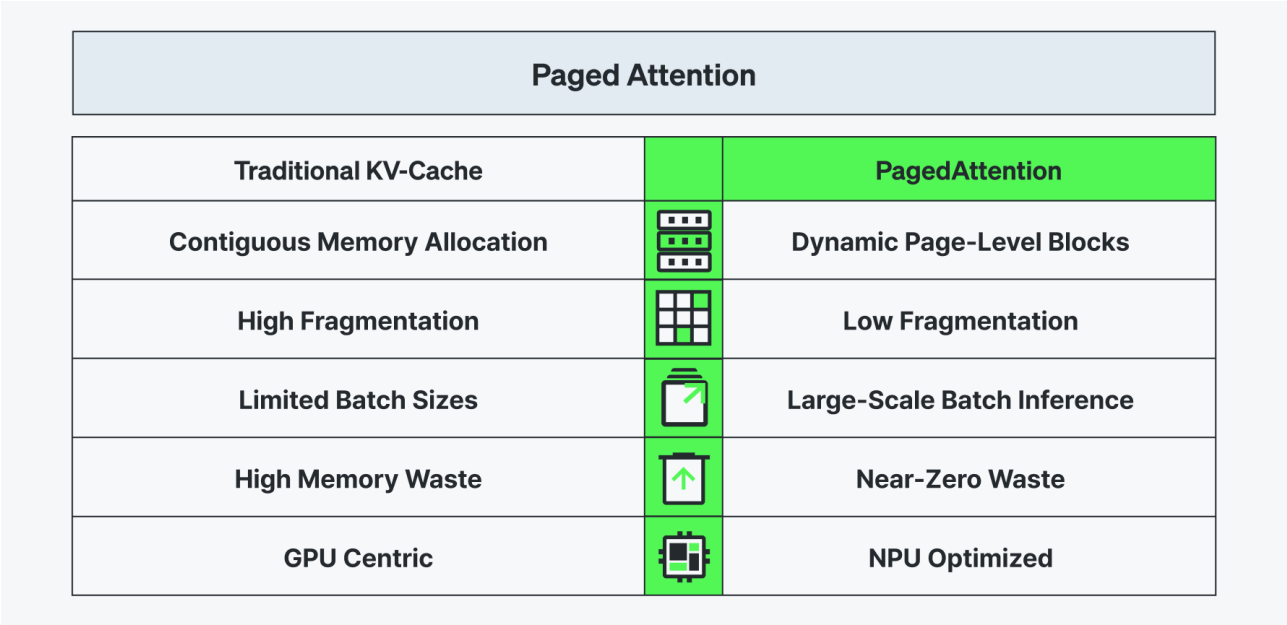
This architecture forms the foundation of RSD (Rebellions Scalable Design), a distributed system framework that extends LLM serving beyond a single device. RSD supports multi-node deployments, disaggregated prefill, and Mixture of Experts (MoE) routing, enabling scalable inference while preserving performance.

Unified Execution for LLM Serving

FlashAttention

Rebellions' FlashAttention is implemented as a tile-based kernel optimized for the NPU's local SRAM size. Blockwise softmax and matrix multiplication are executed entirely within shared memory, reducing DRAM access and improving compute efficiency.

The kernel operates on a partition size provided at runtime, either set explicitly by the user or selected through default settings in our `optimum-rbln` library, which connects HuggingFace models to Rebellions NPUs and handles model compilation and execution. Fused primitives combine normalization and accumulation steps to reduce memory traffic across DRAM and SHM.



[Figure 2. PagedAttention]

PagedAttention

PagedAttention enables scalable LLM inference by managing KV cache in logical blocks, allowing for efficient memory usage across long sequences and multi-session batches. Unlike eager attention, it avoids memory fragmentation by efficiently evicting/reusing cached KV blocks during decoding.

Rebellions implements PagedAttention with kernel-level support for memory management based on KV blocks. Partition size is configurable, with default values optimized via optimum-rbln to balance performance and memory footprint.

Our runtime is fully compatible with vLLM's block table structure. During inference, the block table is passed directly into the kernel. Using dynamic DMA, the CP evaluates addresses on the fly and accesses arbitrary DRAM locations without relying on fixed memory addresses. This dynamic block addressing is made possible by compiler-level support for runtime address resolution.

All memory mapping, block translation, and alignment are handled internally in the kernel, enabling high-throughput inference even under irregular workloads without requiring preprocessing or static allocation.

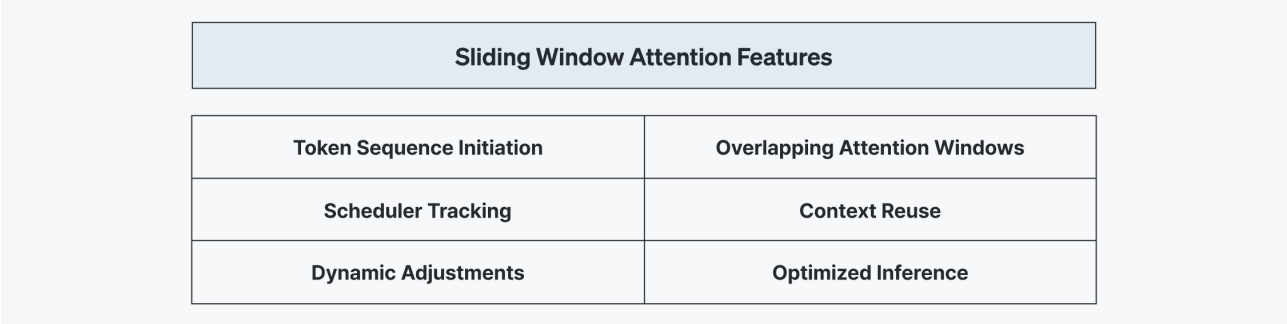
Causal Mask

Causal masking is automatically handled inside the compute primitives. There is no need to explicitly pass or create a separate attention mask during runtime. This simplification reduces setup overhead and enables native support for models that require causal attention, such as autoregressive decoders.

Scaled Dot-Product Attention (SDPA)

Rebellions fully supports scaled dot-product attention via the `torch.nn.functional.scaled_dot_product_attention` interface. FlashAttention is applied adaptively for large sequences, using partition sizes optimized for memory and compute efficiency. Multiple variants, such as causal and masked attention with `float`, `bool`, or `None` types, are also internally optimized for execution on the NPU.

Sliding Window Attention Features

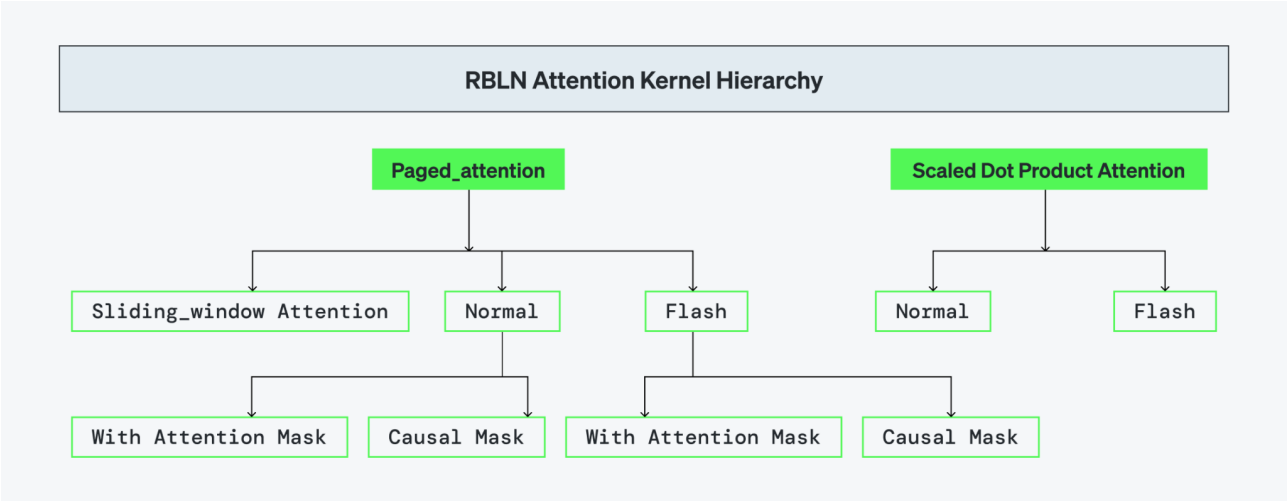


[Figure 3. Sliding Window Attention]

Sliding Window Attention enables long-context and streaming inference by restricting attention to a fixed-size window. Instead of storing the full history, it maintains only the most recent tokens needed for the current step, significantly reducing memory pressure.

Rebellions implements this with efficient KV cache window management. Only the active window is stored, and as the context advances, KV entries are rotated in-place without reallocating memory.

The runtime manages index rotation and window tracking internally, ensuring that models like Gemma3 can operate efficiently with minimal overhead. This allows Rebellions NPUs to support streaming use cases with consistent memory utilization and high token throughput.



[Figure 4. RBLN Attention Kernal Hierarchy]

vLLM RBLN Plugin

Our vLLM RBLN plugin integrates attention mechanisms into a unified execution path, serving as the interface between user applications and the NPU runtime. Specifically, FlashAttention and PagedAttention are deeply embedded in the runtime, sharing a consistent computation graph and memory model. The plugin enables vLLM-compatible applications to run on Rebellions NPUs without code changes, while leveraging NPU-specific optimizations for high throughput and low latency.

Currently, the vLLM-RBLN plugin is designed to integrate with `optimum-rbln`. In this setup, models are compiled using `optimum-rbln`, and the resulting model directory is then referenced by vLLM through the model parameter. This workflow remains the default implementation, providing a stable and proven path for users. All online tutorials are currently based on this design.

Looking ahead, we are actively developing a new architecture that uses `torch.compile()` and natively integrates with vLLM's APIs and model zoo. This next-generation design removes the need for a separate compilation step, enabling a more seamless user experience through standard vLLM workflows. With **`torch.compile()`**, the first run triggers a cold start during which the model is compiled, and subsequent runs become warm starts, benefiting from cached and optimized artifacts.

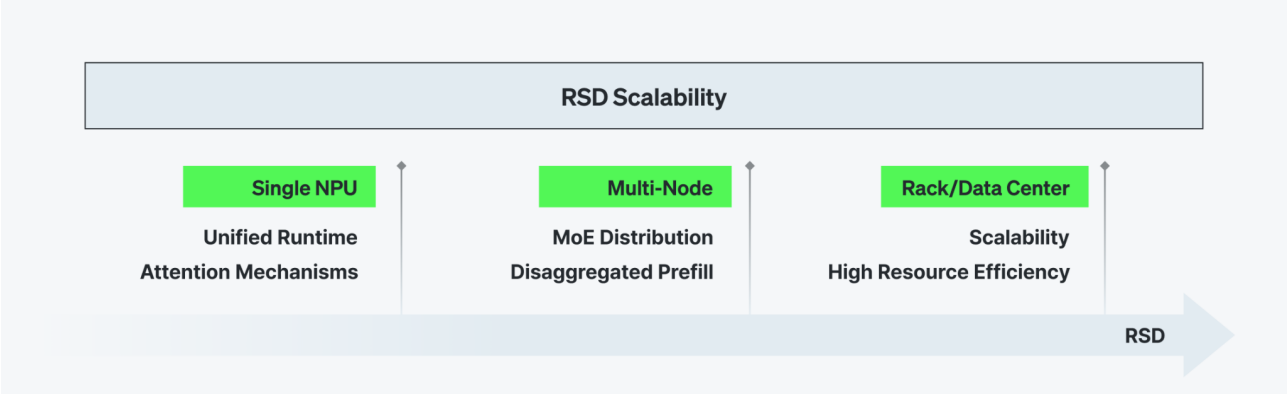
Execution Over Composition: A Cohesive System

Taken together, these constitute not a collection of features, but a fully executed system. The

Rebellions runtime manages LLM workloads end-to-end, supporting dynamic sequence batching, token-level parallelism, cache-aware scheduling, and memory compaction. These optimizations ensure production-scale serving across diverse workloads.

The runtime coordinates all operations across the NPU as a single executable graph, coordinating compute kernels, memory transfers, and cache states in a single control path. This unified design, tailored to the NPU's architecture from the outset, enables seamless interaction between attention mechanisms, avoiding the inefficiencies of layered extensions. This cohesive system unlocks production-grade LLM serving tailored to real-world demands, reinforcing system-level focus and impact.

Scaling with RSD



[Figure 5. RSD Scalability]

Production LLM serving demands a distributed architecture. RSD is a technical structure that extends LLM serving beyond a single device, scaling to racks and data center nodes. It supports:

- **Disaggregated Prefill:** Separates context-building from decoding to optimize resource use across nodes.
- **Multi-Node Execution:** Enables inference across multiple NPUs for scalable performance.
- **Mixture of Experts (MoE) Support:** Handles MoE models efficiently, distributing expert computations across devices.

These capabilities allow LLM instances to span multiple devices, scale throughput under memory constraints, and distribute workloads intelligently across compute resources.

Conclusion: Rebellions' AI Serving Infrastructure

LLM inference is defined by execution, not just speed. Serving complex models and diverse workloads requires a robust, scalable system that operates stably in production environments. Rebellions has built this system on our NPUs, with optimized FlashAttention, PagedAttention, and Sliding Window Attention, integrated via the vLLM RBLN plugin for easy use with vLLM applications.

RSD extends this to distributed environments with disaggregated prefill, multi-node execution, and Mixture of Experts support, positioning Rebellions as a provider of AI serving infrastructure, not just hardware accelerators. The future of AI will depend on who delivers executable, scalable serving infrastructure. Rebellions is executing that infrastructure now.