

Generative AI on a General-Purpose Neural Processor (GPNPU)

Nigel Drego, Yoshihiro Horie
Quadric.ai
{nigel, yoshihiro.horie}@quadric.ai

Abstract—Generative-AI workloads evolve on a quarterly cadence, with novel attention variants, quantization formats, and routing schemes appearing faster than fixed-function neural-processor silicon can be respun. GPUs offer the programmability to keep pace but cannot meet the power and area envelope required for edge deployment. This paper presents the General-Purpose Neural Processor (GPNPU): a single programmable processor architecture, delivered as configurable IP, that unifies scalar, vector, and matrix execution in one instruction-set architecture (ISA) and one programming model accessible from standard C++ and Python. The architecture spans roughly three orders of magnitude in compute capability with the same kernels and toolchain across the range. As a concrete demonstration that data-center-class kernel sophistication can be brought to edge silicon, we describe a fused attention kernel implemented in approximately 80 lines of standard C++, against which Qwen3-8B runs end-to-end at 21 tokens per second on a four-core configuration — approximately 90% of the external-memory bandwidth roofline at INT4 weight precision.

I. INTRODUCTION AND MOTIVATION

A. Generative AI workloads are moving targets

In the twelve months ending mid-2026, every major open-source large-language-model family — Llama [1], Qwen [2], Mistral, Phi — has shipped architectural changes that materially affect inference-time computation: new attention variants (grouped-query [3], sliding-window, multi-latent), new position-encoding schemes, and new quantization formats (W4A8 per-channel, MXFP4). The picture is similar in adjacent generative-AI domains such as automatic speech recognition and vision-language modeling. For an inference platform, this is not a one-time porting cost; it is a continuous engineering load that scales with the rate of model innovation, not with the size of the user base.

B. Why fixed-function NPUs and GPUs both fall short

Fixed-function neural processors implement a closed operator set directly in hardware, of which the Tensor Processing Unit [4] is a canonical example. When a model introduces an operator outside that set — a new attention variant, a new quantization format, a non-standard activation — that operator must be evaluated on a host CPU coprocessor paired with the NPU. This coprocessor fallback is the structural failure mode: the accelerator’s on-chip memory and the host CPU’s memory hierarchy are separate address spaces communicating through external DRAM, so each unsupported operator forces the intermediate tensor out of the accelerator’s local memory, across the system bus to DRAM, into the host’s memory, back

to DRAM after the CPU finishes, and back into accelerator memory for the next stage — a synchronization point between the two pipelines and an evaluation on a compute substrate one to two orders of magnitude less power-efficient than the NPU itself. Because the unsupported operators sit inside the model’s inner loop — attention runs once per layer per token; novel quantization scales apply per matmul — even a single fallback per layer can collapse end-to-end perf/W to a small fraction of the headline NPU specification. Coarse-grained reconfigurable architectures (CGRAs) and dataflow accelerators can absorb new operators in compiled form in principle, but the maturity of their custom compiler stacks lags the model-release cadence.

GPUs avoid the coverage problem but fall outside the edge power and cost envelope: CUDA and ROCm let a data-center GPU absorb new operators in days, but the power and bill-of-materials cost that pays for that flexibility puts a GPU out of reach for mobile, automotive, and industrial embedded deployment. The market needs an architecture that absorbs novel operators *in hardware* — not on a coprocessor — while staying inside the edge envelope.

C. Our contribution

We define the General-Purpose Neural Processor (GPNPU): a single processor architecture that combines scalar control, vector/SIMD compute, and matrix/tensor compute in a unified instruction-set architecture (ISA), programmed in standard C++ or Python via the Quadric® Chimera™ Graph Compiler (CGC), and delivered as configurable IP. The remainder of this paper presents the GPNPU architecture (Section II), demonstrates its applicability to generative AI through worked examples (Section III), and concludes (Section IV).

II. GPNPU ARCHITECTURE

A. A single instruction stream over a 2D SIMD PE array

The Chimera™ General-Purpose Neural Processor is organized around a single instruction stream per core (Fig. 1). A unified fetch and decode feeds two execution paths interleaved within that stream: a *scalar unit* handling uniform control flow, loop bookkeeping, and address arithmetic, and a 2D *SIMD PE array* — a square grid of *processing elements* (PEs) — executing vector and matrix operations in lockstep. Because scalar and SIMD instructions share the same fetch/decode pipeline, no compiler boundary separates control from compute. Each PE contains a 32-bit ALU, a configurable bank

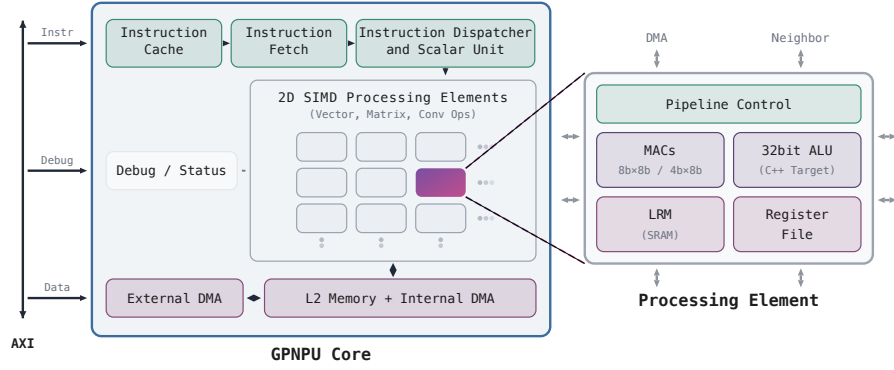


Fig. 1. Chimera™ GPNPU core block diagram. A single instruction stream feeds the scalar unit and the 2D SIMD PE array; per-PE detail (right) shows the 32-bit ALU, configurable MAC bank, local register-file SRAM (LRM), and neighbor-communication path. All on-chip memory (LRM and L2) is software-managed.

of MAC units, a local register-file SRAM, and a neighbor-communication path to adjacent PEs. The dispatcher broadcasts each SIMD instruction to every PE; per-PE conditional execution is expressed through predication rather than independent program counters, while uniform control flow executes on the scalar unit and bypasses the PE array. Matrix and convolution operations are realized through SIMD execution combined with first-class neighbor exchange, implementing the systolic dataflow of dense linear algebra without a separate fixed-function matrix engine. Because scalar control, vector arithmetic, matrix multiplication, and quantization all execute from the same instruction stream, no operator requires off-core fallback to a host CPU — eliminating the accelerator–host boundary identified in Section I-B as the dominant failure mode of fixed-function NPU architectures.

B. Memory hierarchy aligned to genAI working sets

Each PE has a private register-file SRAM — the *local register memory* (LRM) — holding tile-local operands during kernel execution. PEs within a core share a larger on-chip L2, sized at IP-configuration time to match target-workload working sets; L2 holds layer-scope tensors (activations, KV-cache slices, normalization and rotary-position-embedding tables) while weights and KV entries exceeding the on-chip budget are streamed from external memory through an integrator-selected interface. All on-chip memory is fully software-managed: there is no implicit cache between LRM, L2, and external memory; DMA engines move data between levels under software control. The hierarchy is exposed to kernel code through typed tensor accessors and explicit data-flow primitives, giving the compiler and kernel author predictable control over data movement.

C. Programming model

GPNPU kernels are authored in standard C++ — parameterized by templated tensor types that encode shape, element type, memory location, and tiling discipline at compile time — or in the *Quadric Kernel Language* (QKL), a Python DSL

that emits the same instruction stream. Kernels can also be composed at the tensor level through the ChiPy® framework, a higher-level Python layer. The Chimera Graph Compiler (CGC) lowers PyTorch and ONNX models to GPNPU instructions, mapping graph-level operators to a combination of pre-existing kernels (matmul, convolution, attention, activation, quantization) and any user-written kernel registered against the toolchain. New operators — novel attention variants, new quantization formats, alternative position encodings — are added by writing the kernel in C++ or QKL and registering it; the compiler lowers model-graph nodes through the existing operator-lowering infrastructure. There is no vendor-SDK gate between operator availability and recompilation: recompile, do not respin.

D. Architecture configurability and SKU range

The GPNPU is delivered as configurable IP. Two largely independent axes are set at configuration (hardware-build) time: the *core tier* selects the PE-array dimension within a core (along with the accompanying MAC count per PE, LRM size, and L2 size), and the *core count* selects how many such cores are instantiated on the die. Three named tiers anchor the range — QC-Nano (64 PEs/core, roughly 1 TOPS, for the most cost- and power-constrained inference), QC-Perform (mid-range, used in the Section III worked example), and QC-Ultra (~1,024 PEs/core, for the most compute-intensive workloads) — and any tier may appear in single- or multi-core form. The largest deployed systems combine many QC-Ultra cores for compute densities exceeding a PetaOP (over 1,000 TOPS), targeting automotive perception, advanced driver assistance, and industrial embedded workloads where edge inference at scale is current production reality. The same C++ and Python kernel sources and the same CGC toolchain target every point in this two-dimensional design space.

III. GENERATIVE AI ON GPNPU

A. GenAI workload coverage

Our canonical worked example is **Qwen3-8B**, an 8-billion-parameter decoder-only transformer (embedding dimension 4,096; 32 query heads, 8 key/value heads under grouped-query attention with factor 4; head dimension 128; 36 layers). We deploy it at W4A8 with INT8 KV cache on a QC-Perform four-core configuration: 4 KB local SRAM per PE, 2 MB L2 per core, 16 MAC units per PE, 1 GHz clock, 96 GB/s aggregate external-memory bandwidth over a 256-bit AXI. End-to-end inference reaches **21 tokens per second** in autoregressive decode at batch size 1, measured by the Chimera Instruction Set Simulator against the full integrated model. The same compiled binary serves both prefill and decode paths. This is the configuration and workload that anchor Sections III-B and III-C.

Three additional workloads illustrate that the GPNPU is not specialized to a single model family. **DeepSeek-R1-Distill-Qwen-1.5B** [6], a 1.5-billion-parameter reasoning distillation of Qwen, runs at **4.4 tokens per second** on a single-core QC-Nano (4 MB on-chip memory, 8 MAC units per PE) — the same kernel base targeting a smaller core *tier*, not merely a smaller core count. The same infrastructure also supports earlier Qwen and Llama-family architectures through compile-time template parameterization. **Whisper-tiny** [5] runs end-to-end as a full encoder-decoder, extending the architecture beyond decoder-only LLMs to workloads with cross-attention between encoder hidden states and decoder queries. **Pi 0.5** ($\pi_{0.5}$) [7], a Vision-Language-Action model for robotics from Physical Intelligence, extends coverage further to fused vision, language, and action prediction. A multimodal Qwen2.5-VL deployment is in late-stage integration on the same kernel and compiler infrastructure. Together these workloads exercise the GPNPU range from a single-core QC-Nano to a multi-core QC-Perform (Section II-D), with the same kernel sources and compiler binary path across.

B. Architectural payoff: fused attention with in-place online softmax

The attention block is the right place to make the architectural claim concrete. While the feed-forward network dominates weight-streaming bandwidth in a transformer layer, attention is where intermediate-tensor bandwidth scales with context length, where novel architectural variants appear most often (sliding-window, grouped-query, multi-latent), and where the dominant share of per-target hand-tuning effort is spent across the generative-AI ecosystem.

Conventional inference stacks split scaled dot-product attention into three sequential kernels — $QK^\top$ matrix multiplication, row-wise softmax, and output quantization — and each materializes the per-row attention-score tensor in shared memory between phases. Unlike layer outputs, this intermediate is purely a transient byproduct of the softmax reduction; nothing else in the model needs to observe it. On the GPNPU, the entire pipeline is expressed as a single C++

kernel: the MAC array streams the INT8 query against the INT8 K tiles and accumulates each row of $QK^\top$ into a 32-bit integer register-file accumulator; the result is rescaled into 32-bit fixed-point form, over which Milakov and Gimelshein’s single-pass online softmax [8] runs with its running maximum and denominator also kept register-resident; and a fused INT8 quantization step writes one INT8 row to L2 memory. The softmax reduction, normalization, activation, and requantization arithmetic all execute in fixed-point — natively efficient on the PE-array hardware and accurate at the precisions LLM inference requires, without invoking floating-point hardware. The MAC units do support FP16 matmul where workloads require higher precision; the examples in this paper did not. The attention-score tensor never exists as a memory-resident object.

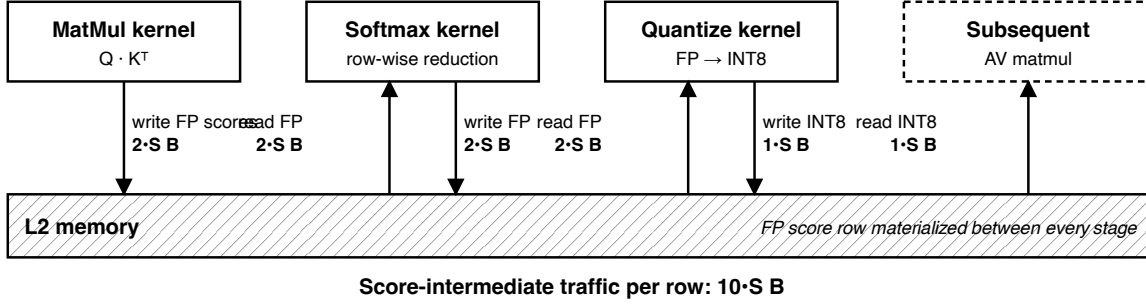
The implication, scoped carefully, is an approximately $5\times$ reduction in L2 memory traffic for this score-intermediate tensor relative to a split-kernel reference at FP16 intermediates and INT8 output (analytical; $10 \cdot S \rightarrow 2 \cdot S$ bytes per row). KV-cache traffic, which dominates total attention bandwidth at decode, is unchanged by fusion. On the four-core configuration of Section III-A, the 8 key/value heads of Qwen3-8B are split at 2 per core, giving a per-core per-layer per-token KV footprint of 512 bytes at INT8; per-layer KV residency in L2 is preserved up to approximately 3,600 tokens before external-memory streaming becomes necessary.

This is GPU-class kernel sophistication — the register-resident fusion of matmul, reduction, nonlinearity, and quantization exemplified by FlashAttention [9]–[11] — inside an edge power and area envelope, expressed in roughly 80 lines of standard C++ portable across the Quadric Chimera GPNPU product line. Figure 2 contrasts the conventional split-kernel attention dataflow with the GPNPU fused kernel. New model architectures extend this same kernel without an IP-core respin.

C. Quantitative results

On the configuration of Section III-A, Qwen3-8B reaches **21 tokens per second** in autoregressive decode at batch size 1, measured by the Chimera Instruction Set Simulator on the full integrated model. Decode is memory-bandwidth-bound on weight streaming: 8.2 billion parameters at INT4 occupy approximately 4.1 GB, so the 96 GB/s external interface caps theoretical throughput at about 23 tok/s. The measured 21 tok/s therefore corresponds to roughly **90 % of the external-memory bandwidth roofline** — the compound result of W4A8 quantization, the Section III-B fused attention kernel keeping intermediate-tensor traffic off the external bus, and the CGC compiler’s tile scheduling. DeepSeek-R1-Distill-Qwen-1.5B reaches 4.4 tok/s on its single-core QC-Nano configuration under the same kernel base. The Section III-B fused attention kernel further reduces on-chip memory traffic for the score-intermediate tensor by approximately $5\times$ relative to a split-kernel reference — a benefit that grows in absolute terms as model context lengths increase, since the score-intermediate

(a) Conventional split-kernel attention



(b) GPNPU fused kernel — score row never leaves the register file

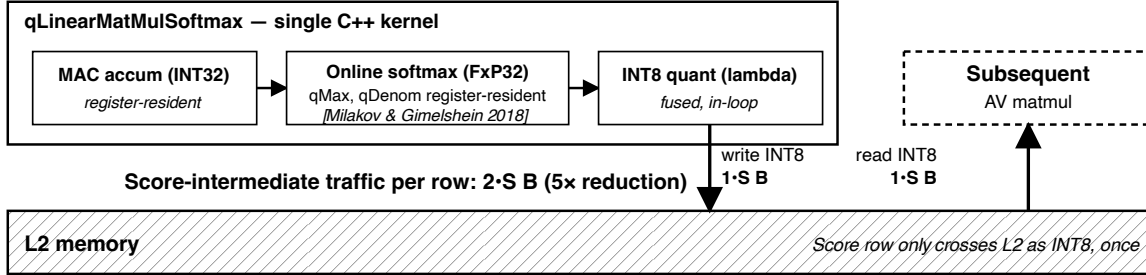


Fig. 2. Score-intermediate memory traffic in the attention block. (a) Conventional split into three kernels materializes the per-row score tensor in L2 between every stage, for a total of $10 \cdot S$ bytes per row at FP16-intermediate / INT8-output precision. (b) The GPNPU fused kernel keeps the row in register-resident state through the matmul accumulator, the online softmax (running maximum and denominator), and the fused INT8 quantization, writing only the final INT8 row to L2 ($2 \cdot S$ bytes per row, including the subsequent AV-matmul read).

tensor is the one component of attention-block bandwidth that scales linearly with sequence length.

IV. CONCLUSION

We have presented the General-Purpose Neural Processor: a single processor architecture that unifies scalar, vector, and matrix execution in one ISA and one C++ programming model, delivered as configurable IP across roughly three orders of magnitude in compute capability. The architecture’s central claim — that data-center-class kernel sophistication can be expressed on edge silicon, without surrendering edge-class power and area envelopes — is demonstrated by a fused attention kernel implemented in approximately 80 lines of standard C++, against which Qwen3-8B runs end-to-end at 21 tokens per second on a four-core configuration at roughly 90 % of the external-memory bandwidth roofline. Novel operators introduced by upstream generative-AI releases are absorbed as kernel and compiler-pass changes rather than IP-core respins or vendor-SDK upgrades, which is the architectural prerequisite for keeping pace with a quarterly model-release cadence. The same architecture and toolchain extend across the generative-AI workload spectrum demonstrated here — from sub-2-billion-parameter LLMs to 8-billion-parameter decoders, and from decoder-only models to full encoder-decoder speech recognition — and the configurability of the Quadric Chimera IP positions the same source base to address on-device multi-

billion-parameter agentic workloads and multimodal vision-language deployments as those become deployable on edge silicon.

REFERENCES

- [1] Llama Team, “The Llama 3 herd of models,” *arXiv:2407.21783*, 2024.
- [2] Qwen Team, “Qwen3 technical report,” *arXiv:2505.09388*, 2025.
- [3] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, “GQA: Training generalized multi-query transformer models from multi-head checkpoints,” in *Empirical Methods in Natural Language Processing (EMNLP)*, 2023.
- [4] N. P. Jouppi *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Int. Symp. Computer Architecture (ISCA)*, 2017.
- [5] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, “Robust speech recognition via large-scale weak supervision,” in *Int. Conf. Machine Learning (ICML)*, 2023.
- [6] DeepSeek-AI, “DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning,” *arXiv:2501.12948*, 2025.
- [7] K. Black *et al.*, “ π_0 : A vision-language-action flow model for general robot control,” *arXiv:2410.24164*, 2024.
- [8] M. Milakov and N. Gimelshein, “Online normalizer calculation for softmax,” *arXiv:1805.02867*, 2018.
- [9] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, “FlashAttention: Fast and memory-efficient exact attention with IO-awareness,” in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2022.
- [10] T. Dao, “FlashAttention-2: Faster attention with better parallelism and work partitioning,” *arXiv:2307.08691*, 2023.
- [11] J. Shah, G. Bikshandi, Y. Zhang, V. Thakkar, P. Ramani, and T. Dao, “FlashAttention-3: Fast and accurate attention with asynchrony and low-precision,” in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2024.



Nigel Dreago is Co-Founder and Chief Technology Officer of Quadric, where he focuses on the architecture, compiler stack, and programming model of the Chimera General-Purpose Neural Processor (GPNPU). He received the M.S. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA. His work spans the full computing stack, from ASIC design to high-level software. Prior to Quadric, he was Co-Founder and Chief Architect at 21 Inc., leading the design of custom silicon for high-throughput

cryptographic workloads, and earlier held research and development roles at PDF Solutions and Intel. His technical interests include programmable inference architectures, machine-learning compilers, and hardware–software co-design for edge AI.