# CnuasGPU, Datasheet

Virtual PCIe AI and ML accelerator, SIMT compute and device memory model

**Document** DS-CNU-004 • **Revision** G • **Issued** 21 August 2026 • **Status** Published

| Item | Value |
|---|---|
| Part | `cnuasgpu` |
| Type | PCIe Gen5 x16 AI and ML accelerator, SIMT, usable as a Soft-GPU on the host or as a QEMU PCIe device |
| Version | `v0.2.0-25-gc54883d` |
| Repo | PacketFive/CnuasGPU |

## 1. Overview

**CnuasGPU** is the emulated AI and ML accelerator of the Cnuas platform. In its emulated form it is a **PCIe Gen5 x16 endpoint**, enumerated by a guest kernel like any other discrete accelerator. It offers a SIMT (single-instruction, multiple-thread) execution model, a portable **CnuasIR** instruction set, and a CUDA-style runtime (`libcnuasrt`). Kernels are dispatched to the best available host SIMD backend (AVX-512, AVX2, or scalar) selected at runtime, giving accelerator-style compute on commodity CPUs.

Two further attachments are planned and are not present today. A **CnuasGPU FPGA accelerator** on a RISC-V host, and **UALink** as an accelerator to accelerator fabric alongside PCIe. Both are described in section 5.

It is available in **two independent forms**, described under *Deployment modes* below. As a **Soft-GPU** it runs in an ordinary host process with no hypervisor, no guest and no kernel module. As a **QEMU PCIe device** it is enumerated by a guest kernel, which is the form used inside a Cnuas rack.

### Deployment modes

| Mode | What provides the device | Needs a guest or kernel module | Typical use |
|---|---|---|---|
| **Soft-GPU (host process)** | `libcnuasdev` arena backend in the calling process | No | Linking the runtime or the ggml backend into a host application, including an unmodified local inference server |
| **QEMU PCIe device** | Emulated `cnuasgpu` PCI function plus `cnuasgpu.ko` in the guest | Yes | Rack deployments, driver development, guest-side software stacks |

Both modes execute the same kernels through the same compute backends, so a result does not depend on which one is in use. Mode selection is automatic and is documented in CnuasDev; it can be forced with ‘CNUAS_DEVICE_BACKEND‘.

The Soft-GPU exposes no `/dev/cnuasgpuN` node and consumes no PCI device ID. The node is created during PCI probe, so it exists only in the emulated form, and a PCI ID identifies a function on a bus rather than a host library. A host character device is possible and would still need no PCI ID; see CnuasGPU Soft-GPU section 1.1.

The Soft-GPU form includes a **ggml backend** (`libggml-cnuas.so`), which lets CnuasGPU be loaded by ggml-based applications. It has been run under an unmodified **Ollama** server, which loaded the backend and executed the prompt-phase matrix multiplications of a published model on the accelerator. See CnuasGPU Soft-GPU.

### Key features

- PCIe Gen5 x16 endpoint in the emulated form, no host bus as a Soft-GPU.
- SIMT execution model with 32-thread warps.
- Runs as a Soft-GPU in a host process, or as an emulated PCIe device in a guest.
- ggml backend for ggml-based applications, exercised under an unmodified Ollama.
- CnuasIR portable ISA (RVV 1.0 subset plus Cnuas tensor opcodes).
- Multi-architecture compute backends with runtime dispatch and env-var override.
- CUDA-style runtime (`libcnuasrt`) over a low-level driver shim (`libcnuasdev`).
- One TTU (Tile Tensor Unit) per SM for matrix work, implicit inside the matrix multiply rather than separately addressable. See section 4.
- TF32 and BF16 training acceleration formats and 2:4 structured sparsity are specified for the TTU and reserved in the backend ABI. No backend advertises them. See sections 4.4.1 and 4.4.2.
- Native CnuasLink endpoint for GPU-to-GPU fabric traffic.
- `cnuassmi` / `cnuas-smi` inventory and telemetry.

## 2. Device block diagram

*Diagram available in the online documentation at `https://github.com/PacketFive/cnuas`.*

## 3. Functional specifications

| Parameter | Value |
|---|---|
| PCI vendor : device ID | `0x1AF4` : `0x10F1` |
| Host bus (emulated device) | PCIe Gen5 x16, 32 GT/s per lane |
| Execution model | SIMT, 32-thread warps |
| ISA | CnuasIR (RVV 1.0 subset + tensor opcodes) |
| Streaming Multiprocessors (SM) | 16 (default configuration) |
| Vector ALU lanes / SM | 32 |
| TTU (Tile Tensor Unit) / SM | 1, 16 by 16 tile edge (specified, see the note below) |
| Warp schedulers / SM | 1 |
| Special function units / SM | 1 |
| Register file / SM | 64 KB (~1 cycle) |
| L1 / shared memory / SM | 128 KB (~30 cycles) |
| L2 cache (chip-wide) | 8 MB (~200 cycles) |
| Device memory (HBM model) | Configurable; 256 MiB default in the emulated device, 512 MiB default as a Soft-GPU (~500 cycles) |
| CnuasLink endpoint | 4 lanes (default) |
| Char device | `/dev/cnuasgpu0`, `/dev/cnuasgpu1`, ... |

The values above are the architectural specification, in the sense used by the CnuasIR datasheet section 3. One of them is not yet reachable by a caller and is called out here so the table is not read as a statement of what software can issue today.

The TTU is one of these. It is present only implicitly, and section 4.6 sets out what that means for a caller.

### 3.1 Configuring device memory

Device memory is sized at start and differs by deployment mode.

| Mode | Setting | Default | Range |
|---|---|---|---|
| Emulated device | `-device cnuasgpu,devmem_size=8G` | 256 MiB | 16 MiB to 1 TiB, power of two |
| Soft-GPU | `CNUAS_HOST_ARENA_MB` | 512 MiB | 1 to 1048576 MiB |

The emulated device carries device memory on BAR1, which is 64-bit prefetchable. A 32-bit BAR cannot describe a region of 4 GiB or more, so the 64-bit form is what allows sizes useful for model weights to be mapped at all. A size outside the range, or one that is not a power of two, fails at device realise rather than producing a device the guest cannot map.

BAR1 is backed by host RAM, so a guest with an 8 GiB device needs a host that can carry the guest RAM and the device memory at the same time. Sizing guidance is in the Minimum System Requirements.

The Soft-GPU arena is mapped with `MAP_NORESERVE`, so it is address space rather than committed memory and pages are faulted in as buffers are written, in the same way the emulated device's BAR is. A value outside the accepted range, or one that does not parse, prints a notice on stderr and the default is used, so a rejected setting is reported rather than silently ignored.

## 4. Compute and math units

### 4.1 Architecture and characteristics

Default configuration. Every quantity marked configurable is a QEMU device property, so the shape of the part is set at start rather than fixed.

| Property | Value |
|---|---|
| ISA | CnuasIR, an RVV 1.0 subset plus Cnuas tensor opcodes |
| Host ISA | x86-64. AArch64 and RISC-V are ABI slots with no backend |
| Implementation | Software emulation. No silicon process, transistor count or die |
| Execution model | SIMT, 32-thread warps over a 32-lane vector array |
| Compute | 16 tiles by 32 FP32 TVP lanes, 512 lanes total, configurable. One TTU per tile. FP32 is the only format any primitive accepts |
| Register file | 64 KB per SM, ~1 cycle |
| On-chip memory | 128 KB L1 and shared per SM, ~30 cycles. 8 MB L2 chip-wide, ~200 cycles |
| Device memory | HBM model, configurable. 256 MiB default as a PCIe device, 512 MiB as a Soft-GPU, ~500 cycles. BAR1 window 16 MiB to 1 TiB |
| Host bus | PCIe Gen5 x16, 32 GT/s per lane. Absent as a Soft-GPU |
| Accelerator fabric | CnuasLink endpoint, 4 lanes, to the CnuasLink switch |
| Power envelope | Not modelled. The accelerator contributes no figure to rack or facility power |
| Clock and latency | Cycle counts above are a latency model applied by QEMU clock dilation, not wall time |

Cycle and capacity figures are the architectural specification in the sense used by the CnuasIR datasheet section 3. They describe the modelled part. They are not a measurement of the host that executes it, and no performance figure is claimed from them.

### 4.2 Functional blocks

The accelerator is organised as a cluster of tiles. A tile is the unit that holds arithmetic, and a cluster is the unit that is given work. The four tile blocks are named the way the wider accelerator field names them, because the same four appear in most inference parts and the vocabulary carries across.

| Block | Per | Function | Status |
|---|---|---|---|
| TTU (Tile Tensor Unit) | 1 per tile | Matrix by matrix multiply with direct accumulation, 16 by 16 tile edge | Architectural. Reached only from inside the matrix multiply, see 4.6 |
| TVP (Tile Vector Processor) | 32 lanes per tile | Activation functions and elementwise operations, and the array every primitive in 4.3 runs on | Implemented |
| TCP (Tile Control Processor) | 1 per tile | Pointer arithmetic, address generation and loop control | Partial. Present as scalar registers, address generation and branches in the CnuasIR ISA, not as a block a caller addresses |
| LSU (Load Store Unit) | 1 per tile | Moves operands between device memory and the tile | Implemented, inside the primitives |
| CCP (Cluster Control Processor) | 1 per cluster | Higher-level control. Reads work submission queues from host memory and hands descriptors to tiles | Implemented as the command processor, reached through the doorbell pages |

A tile is what the datasheet elsewhere calls a Streaming Multiprocessor, and the cluster is the device. Both namings are kept because the SIMT vocabulary is what the runtime and the CUDA-style API use, while the tile vocabulary is what the fabric implementation in section 5 is built from.

Chip-wide blocks outside the tile hierarchy.

| Block | Function | Status |
|---|---|---|
| Warp scheduler | Issues 32-thread warps, one per tile | Implemented |
| SFU (Special Function Unit) | Transcendentals, `sin`, `cos`, `sqrt`, `rcp` | Architectural. The transcendentals inside the activations run on the TVP lanes |
| Copy engines | DMA, host to device, device to host, device to device | Implemented, 2 engines |
| MMU | Per-context page tables, discrete addressing | Implemented |
| Performance counters | Per tile and chip-wide, for `cnuassmi` | Implemented |
| CnuasLink controller | Accelerator to accelerator traffic | Implemented |

Blocks marked architectural or partial are described in the specification and are not separately addressable by a caller. Making the TTU and the TCP addressable is carried in the Cnuas Roadmap, and it is the same work as giving them RTL on fabric, because a block that no caller can name has no interface to implement.

### 4.3 The math surface

These are the primitives a caller can issue. Each is a `cnuas_compute_ops` entry point, implemented once per backend, and reached through `libcnuasrt` on device pointers. `libcnuasblas` layers a BLAS-style surface on top of the same entry points.

| Class | Primitives | Runtime entry points |
|---|---|---|
| Vector, level 1 | Add, scale, dot product | `cnuasVectorAddF32`, `cnuasVectorScaleF32`, `cnuasVectorDotF32`, or `cnuasblasSaxpy`, `cnuasblasSscal`, `cnuasblasSdot` |
| Matrix vector, level 2 | General matrix vector product, expressed on the matrix multiply. No transpose, unit increments only | `cnuasblasSgemv` |
| Matrix, level 3 | `C = alpha·AB + beta·C` | `cnuasSgemm` |
| Matrix, level 3, transposed | `C = alpha·AB' + beta·C`, the shape a weight matrix by a batch of activations actually takes | `cnuasSgemmNT` |
| Reduction | Sum reduction underneath the dot product and the softmax maximum | Implicit |
| Activation | ReLU, GELU, SiLU, sigmoid, tanh | `cnuasReluF32`, `cnuasGeluF32`, `cnuasSiluF32`, `cnuasSigmoidF32`, `cnuasTanhF32` |
| Normalisation | Row-wise softmax, computed with the maximum subtracted so a large input cannot overflow | `cnuasSoftmaxF32` |
| Program execution | A CnuasIR kernel object, interpreted | `cnuasLaunchKernel` |

Activations permit the destination and the source to be the same buffer, because activation in place is the normal case in an inference graph.

### 4.4 Numeric formats

| Format | Width | Status |
|---|---|---|
| FP32 | 32 bit | Every primitive above |
| TF32 | 19 bit significand in a 32 bit container | Specified for the TTU; no primitive accepts it today |
| BF16 | 16 bit | Specified for the TTU; no primitive accepts it today |
| FP16 | 16 bit | Specified for the TTU; no primitive accepts it today |
| INT8, FP8 | 8 bit | Named in the architectural specification of the tensor unit; no primitive accepts them today |

The CnuasIR object header carries an ISA level for this: `0x0001` is the current subset at SEW 32, and `0x0002` adds f16 and bf16. See the CnuasIR datasheet.

#### 4.4.1 Training acceleration formats

TF32 and BF16 are the two operand formats the TTU is specified to take for training work, and they differ in what the caller has to change.

| | TF32 | BF16 |
|---|---|---|
| Exponent, significand | 8, 10 explicit | 8, 7 explicit |
| Dynamic range | FP32 range | FP32 range |
| Caller side buffers | Stay FP32, the narrowing happens inside the tile unit | Become BF16, a separate entry point and a conversion |

| | TF32 | BF16 |
|---|---|---|
| Accumulation | FP32 | FP32 |
| Intended use | A drop-in path for existing FP32 matrix multiply call sites | Operand bandwidth and footprint reduction where the caller can hold BF16 |

Both accumulate in FP32, so the accumulator format does not change with the operand format. Neither is implemented. The capability bits `CNUAS_CCAP_GEMM_TF32` and `CNUAS_CCAP_GEMM_BF16` are reserved in the backend ABI and advertised by no backend, so a caller that asks for either receives `cnuasErrorNotSupported`.

### 4.4.2 Structured sparsity

The specified scheme is 2:4 structured sparsity: within every aligned group of four weights along the reduction axis, two are zero. The operand is stored as the two surviving values per group plus two bit indices naming their positions, which halves the stored weight matrix and lets the tile unit skip the pruned lanes rather than multiplying by zero.

This is a property of the weight operand, so it composes with the formats above rather than replacing them, and it requires a caller that has pruned and retrained to the 2:4 pattern. The capability bit `CNUAS_CCAP_SPARSE_2_4` is reserved and advertised by no backend. No pruning tool, metadata layout or sparse entry point exists.

### 4.5 Capability reporting

A backend advertises what it implements in a capability bitmap, and a caller that issues an unimplemented primitive receives `cnuasErrorNotSupported` rather than a wrong answer.

| Capability | Meaning | Advertised by |
|---|---|---|
| `VEC_F32` | Vector level 1 | Every backend |
| `GEMM_F32` | Matrix multiply | Every backend |
| `GEMM_NT_F32` | Transposed matrix multiply | scalar, AVX2, AVX-512 |
| `REDUCE` | Reductions | Every backend |
| `ACTIVATION_F32` | Activations and softmax | scalar, AVX2, AVX-512 |
| `CNUASIR_EXEC` | Interprets a CnuasIR program | CnuasIR backend |
| `TENSOR_CORE` | A directly addressable tile primitive | No backend |
| `GEMM_TF32` | Matrix multiply with TF32 operands and FP32 accumulation | No backend |
| `SPARSE_2_4` | 2:4 structured sparse weight operands with index metadata | No backend |
| `GEMM_F16`, `GEMM_BF16`, `CONV2D_F32`, `FFT_C32` | Reserved in the ABI | No backend |

The ISA hint bits `X86_AVX2`, `X86_AVX512`, `ARM_NEON`, `ARM_SVE`, `RISCV_RVV` and `FPGA` are informational and describe how a backend reaches the host, not what it can compute.

### 4.6 What is specified but not separately addressable

The TTU is present only implicitly. `sgemm` computes `alpha·AB + beta·C`, so accumulation into a destination tile does happen, and the AVX-512 backend blocks internally with a 16 by 16 `C` tile. What does not exist is an exposed tile abstraction. No caller can address a tile register file or issue an accumulate-into-tile primitive, and the `tmma` opcode named in the CnuasIR specification has no implementation. The emulated device reports a `tensor_size` of 16, which describes the specified tile edge rather than a unit that can be driven directly; the host backend reports 0, and `TENSOR_CORE` is set by no backend.

Convolution, FFT and batched matrix multiply have reserved slots in the backend ABI and no implementation. The training acceleration formats of 4.4.1 and the structured sparsity of 4.4.2 depend on that same exposed tile primitive, because there is no unit to route a narrowed operand to until the TTU is addressable. Exposing the tile primitive, the TF32, BF16 and 2:4 sparse paths, and the quantised paths are carried in the Cnuas Roadmap.

## 5. Planned attachments

Neither of the following is present. They are recorded here because they change how the accelerator attaches, and the interfaces in section 10 describe only what exists today.

| Attachment | What it is | Where it is tracked |
|---|---|---|
| CnuasGPU FPGA IP | The tile blocks of section 4.2 as synthesisable RTL on real fabric, with a RISC-V host. See 5.1 | Cnuas Roadmap, Epic 2 |
| UALink | An open accelerator to accelerator interconnect, alongside PCIe rather than replacing it. Today that role is filled by CnuasLink, whose endpoint is described in section 10 | Cnuas Roadmap, Epic 2 |

### 5.1 CnuasGPU FPGA IP

The reference target is the Microchip PolarFire SoC on the Icicle Kit, chosen because it carries PCIe hard IP and four Linux-capable RISC-V cores on the same die, so the board is a whole system rather than a fabric that needs a host beside it.

| Resource | Icicle Kit value | What it carries |
|---|---|---|
| FPGA fabric | MPFS250T, 254 K logic elements | The tile blocks |
| Math blocks | 784 by 18 by 18 MACC | The TVP lanes, about 64 of them for an 8-lane v0.1 vector, and the TTU once it has a caller-visible interface |
| LSRAM | 17.6 Mb | The vector register file and the device memory window |
| Hard processors | 4 by SiFive U54 RV64GC, 1 by E51 monitor | The CCP and the TCP, as a firmware daemon under Linux |
| Host bus | PCIe Gen2 x4 hard IP | The endpoint, in place of the emulated Gen5 x16 |

The block names in section 4.2 are the reason this is tractable. Each is a module with an interface rather than a description, so the fabric build has a module list to implement and the emulated device becomes the reference model that the RTL is checked against. The blocks that are marked architectural today are the ones with no RTL and no caller-visible interface, and they are the same list in both places.

The v0.1 fabric parameters, the RTL module list, the descriptor format and the bring-up phases are in the CnuasGPU FPGA design note in the component repository. The software above `libcnuasdev` does not change, because the board is another device backend.

## 6. Kernel driver features (`cnuasgpu.ko`)

Single char-device driver (`src/cnuasgpu/driver/cnuasgpu.c`) exposing `/dev/cnuasgpuN`.

### 6.1 ioctl interface

| ioctl | Purpose |
|---|---|
| `CNUASGPU_IOC_INFO` | Query device properties (SM count, memory, IDs) |
| `CNUASGPU_IOC_ALLOC` | Allocate device memory |
| `CNUASGPU_IOC_FREE` | Free device memory |
| `CNUASGPU_IOC_MEMCPY` | Host↔device / device↔device copy |
| `CNUASGPU_IOC_LINK_STATUS` | Query CnuasLink endpoint status |
| `CNUASGPU_IOC_LINK_SEND` | Send a CnuasLink frame to a peer GPU |
| `CNUASGPU_IOC_LINK_RECV` | Receive a CnuasLink frame from a peer GPU |

### 6.2 Memory / doorbell

- `mmap()` of BAR1 exposes the device-memory window to userspace.
- BAR2 doorbell pages signal work submission; BAR0 holds MMIO control registers.
- MSI-X interrupts deliver completion and CnuasLink events.

## 7. Runtime / userspace features

Two libraries, layered CUDA-style over the driver.

### 7.1 `libcnuasrt.so`, CUDA-Runtime-style API

| Function | Purpose |
|---|---|
| `cnuasInit` / `cnuasShutdown` | Runtime lifecycle |
| `cnuasGetDeviceCount` | Enumerate GPUs |
| `cnuasGetDevice` / `cnuasSetDevice` | Current device selection |
| `cnuasGetDeviceProperties` | Device capability query |
| `cnuasMalloc` / `cnuasFree` | Device memory management |
| `cnuasMemcpy` | Host↔device / device↔device transfers |
| `cnuasDeviceSynchronize` | Barrier / completion wait |
| `cnuasSgemm` | Single-precision GEMM primitive |
| `cnuasReluF32`, `cnuasGeluF32`, `cnuasSiluF32`, `cnuasSigmoidF32`, `cnuasTanhF32` | Elementwise activations, in place permitted |
| `cnuasSoftmaxF32` | Row wise softmax |
| `cnuasGetErrorString` | Error decoding |

### 7.2 `libcnuasdev.so`, low-level driver shim

| Function | Purpose |
|---|---|
| `cnuasdev_open` / `cnuasdev_close` | Open/close `/dev/cnuasgpuN` |
| `cnuasdev_get_device_count` | Device enumeration |
| `cnuasdev_query_info` | `CNUASGPU_IOC_INFO` wrapper |
| `cnuasdev_alloc` / `cnuasdev_free` | `CNUASGPU_IOC_ALLOC` / `FREE` wrappers |
| `cnuasdev_memcpy` | `CNUASGPU_IOC_MEMCPY` wrapper |

| Function | Purpose |
| --- | --- |
| `cnuasdev_mmap` / `cnuasdev_munmap` | BAR1 device-memory mapping |
| `cnuasdev_status_str` | Status decoding |

## 8. Compute backends & dispatch

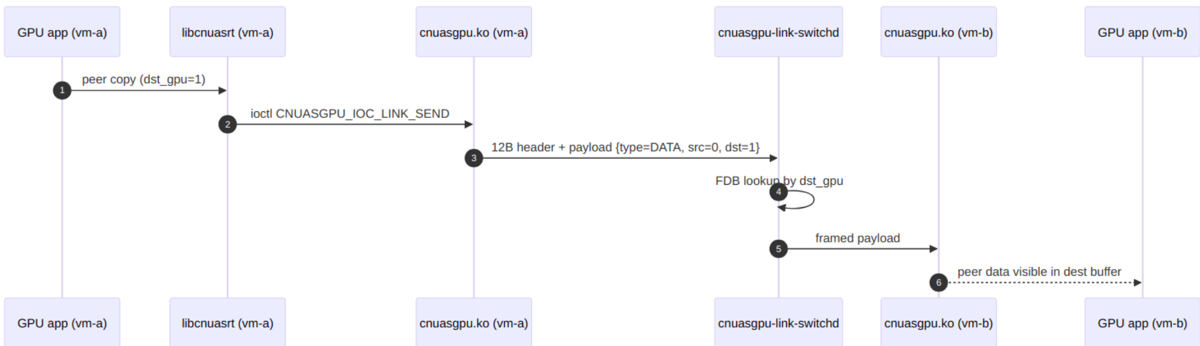
Kernels are implemented behind a `cnuas_compute_ops` vtable; the loader picks a backend at runtime.

| Host ISA | Backend | Source | Vector width |
| --- | --- | --- | --- |
| x86-64 | AVX-512 | `lib/compute/x86_avx512/` | 512-bit |
| x86-64 | AVX2 | `lib/compute/x86_avx2/` | 256-bit |
| Portable | scalar | `lib/compute/scalar/` | reference/fallback |

The loader in `lib/compute/loader/loader.c` selects the best backend automatically, preferring AVX-512 over AVX2 over scalar. Four environment variables override that choice.

| Variable | Effect |
| --- | --- |
| `CNUAS_COMPUTE_BACKEND` | Force a specific backend |
| `CNUAS_COMPUTE_LIST` | List available backends |
| `CNUAS_COMPUTE_VERBOSE` | Verbose selection logging |

## 9. GPU-to-GPU peer copy flow




## 10. Interfaces

| Interface | Description |
| --- | --- |
| Host bus | QEMU PCIe device (`cnuasgpu`), a Gen5 x16 endpoint at 32 GT/s per lane. Absent in Soft-GPU mode |
| BAR0 | 64 KB MMIO control registers |
| BAR1 | Device memory window, 64-bit prefetchable, 16 MiB to 1 TiB, configurable |
| BAR2 | 4 KB per-context doorbell pages |
| Char device | `/dev/cnuasgpu0`, `/dev/cnuasgpu1`, … Absent in Soft-GPU mode, where `libcnuasdev` serves allocations from a host arena instead |
| ggml backend | `libggml-cnuas.so`, loaded by ggml-based applications through `GGML_BACKEND_PATH` (Soft-GPU mode) |

| Interface | Description |
|---|---|
| GPU fabric | CnuasLink endpoint (see CnuasLink) |

## 11. Software support

| Item | Value |
|---|---|
| Runtime | `libcnuasrt.so` (CUDA Runtime-style API) |
| Low-level driver | `libcnuasdev.so` |
| Compute primitives | Vector level 1, matrix vector, tiled SGEMM and SGEMM-NT, activations and softmax. See section 4 |
| Math library | `libcnuasblas.so`, BLAS-style entry points over the same primitives |
| Tools | `cnuassmi` / `cnuas-smi` (device inventory + telemetry) |
| Design reference | CnuasGPU Design |

## 12. Ordering / integration information

| Item | Value |
|---|---|
| Repo | `PacketFive/CnuasGPU` |
| Submodule path | `src/cnuasgpu` (in `PacketFive/cnuas`) |
| Version | v0.2.0 |
| Language | C (GNU C) + host SIMD intrinsics |

## 13. Revision history

| Revision | Date | Notes |
|---|---|---|
| A | 2026-07-05 | Initial datasheet |
| B | 2026-07-05 | Added block diagram, ioctl/runtime tables, compute-backend dispatch, peer-copy flow |
| C | 2026-08-12 | Recorded the Soft-GPU deployment mode, the ggml backend and the Ollama result alongside the QEMU device form |
| D | 2026-08-12 | Marked the tensor MAC unit as specified but not separately addressable, matching the CnuasIR datasheet convention |
| E | 2026-08-15 | Stated that the Soft-GPU exposes no device node and consumes no PCI device ID |
| F | 2026-08-15 | Recorded configurable device memory, the 64-bit prefetchable BAR1 and the host arena bounds |
| G | 2026-08-21 | Added the architecture and characteristics table and the functional block list, described the math primitives each backend implements, stated the PCIe generation and width, and recorded the planned FPGA and UALink attachments |
| H | 2026-08-21 | Specified the TF32 and BF16 training acceleration formats and 2:4 structured sparsity for the TTU, and reserved their capability bits in the backend ABI |

PacketFive, Packet Five Networks Ltd., Dublin, Ireland.