CnuasDev, Datasheet

Device access library and the pluggable compute backend family

Document DS-CNU-012 • **Revision** K • **Issued** 23 August 2026 • **Status** Published

Item	Value
Part	`libcnuasdev.so`
Type	Device access library
SONAME	`libcnuasdev.so.0`, current build `libcnuasdev.so.0.1.0`
Package	`cnuas-libcnuasrt`
Version	`v0.2.0-29-gf3e5f47`
Repo	PacketFive/CnuasGPU

1. Overview

CnuasDev is the thin device layer beneath CnuasRT. In QEMU PCIe mode it opens the `/dev/cnuasgpuN` character devices created by `cnuasgpu.ko` and turns their ioctl and mmap protocol into a small C interface. In Soft-GPU mode it provides the same allocation/copy interface from an arena in the calling process and opens no Linux device node. The runtime therefore never issues a raw ioctl and tools that need device facts without the full runtime have somewhere lighter to link against.

A third `hostdev` backend opens `/dev/cnuasgpu_hostN`, provided by `cnuasgpu_host.ko` from host pages without QEMU or PCI. It adds capability querying, pinned allocation, DMA-BUF export, zero-copy mappings and ordered asynchronous batches while retaining the same ordinary allocation/copy API.

This datasheet also specifies the **compute backend ABI**, because the backend shared objects are built and shipped alongside the device library and share its release cadence.

The in-process backend is expected to have lower control-path overhead than a future host character device when both use the same CPU compute backend. The character-device value is a common kernel ABI, isolation, shared ownership, DMA-BUF interoperability and system-wide management—not automatic arithmetic acceleration. The complete comparison is in CnuasGPU as a Soft-GPU.

Key features

- Three backends: in-process `host`, host character device `hostdev`, and QEMU PCI character device `device`.
- Device enumeration, backend identity and capability information without opening the runtime.
- Explicit allocate, free, and copy against device memory.
- Direct memory mapping of device memory into the host address space.
- A published backend ABI so a new instruction set is a new shared object rather than a rebuild of the runtime.
- Backend selection at first use with an environment variable override.
- A CnuasLink transport that speaks frames rather than raw ioctls, with the wire format available as a standalone unit that needs no device.

2. Application programming interface

Every symbol below is exported by `libcnuasdev.so.0.1.0`.

Group	Function	Purpose
Discovery	`cnuasdev_get_device_count`	Count the `/dev/cnuasgpuN` nodes present
Handle	`cnuasdev_open`	Open a device by index and return a handle
Handle	`cnuasdev_close`	Release a handle and any allocations tied to it
Information	`cnuasdev_query_info`	Identity, geometry, and memory figures for the device
Memory	`cnuasdev_alloc`	Reserve a device memory range
Memory	`cnuasdev_free`	Release a device memory range
Memory	`cnuasdev_memcpy`	Copy between host and device memory
Mapping	`cnuasdev_mmap`	Map device memory into the calling address space
Mapping	`cnuasdev_munmap`	Undo a mapping
Diagnostics	`cnuasdev_status_str`	Text for a `cnuasdev_status_t` code
Backend	`cnuasdev_backend` / `cnuasdev_backend_name`	Identify the selected arena, host character device, or PCI character device
Capabilities	`cnuasdev_query_caps`	Zero-copy, pinned, DMA-BUF and async bits plus limits and live pinned use
Memory	`cnuasdev_alloc_flags`	Allocate with explicit flags, including pinned backing
Sharing	`cnuasdev_export_dmabuf`	Export a page-aligned pinned range as an independently owned fd
Async	`cnuasdev_async_submit`	Submit an ordered batch and report accepted count and first sequence
Async	`cnuasdev_async_poll` / `cnuasdev_async_wait`	Reap structured completions without or with a timeout
Async	`cnuasdev_synchronize`	Submit/wait for a barrier and surface prior failures
Async	`cnuasdev_async_fd`	Pollable completion descriptor
CnuasLink	`cnuasdev_link_id`	Local fabric identifier, cached after first use
CnuasLink	`cnuasdev_link_status`	Whether the switch is reachable and a frame is staged
CnuasLink	`cnuasdev_link_send`	Encode and send a frame, filling in the local source
CnuasLink	`cnuasdev_link_recv`	Receive one frame and decode it
CnuasLink	`cnuasdev_link_send_raw`	Send bytes that are already a frame
CnuasLink	`cnuasdev_link_recv_raw`	Receive bytes without decoding
CnuasLink	`cnuaslink_frame_encode`	Build a frame in a caller buffer
CnuasLink	`cnuaslink_frame_decode`	Validate and parse a received frame
CnuasLink	`cnuaslink_frame_type_str`	Text for a frame type

2a. CnuasLink transport

The transport is a synchronous wrapper over the three `LINK` ioctls. It is the lowest layer that speaks frames rather than bytes and is the foundation CnuasCCL is expected to build on.

Parameter	Value
Header	`cnuaslink_frame.h`
Frame header size	12 bytes, fixed
Maximum frame	65536 bytes
Maximum payload	65524 bytes
Byte order	Little endian for every multi byte field
Identifier range	1 to 65534; 0 and 65535 are reserved
Receive timeout	0 polls, 4294967295 waits without limit, otherwise milliseconds

Two properties of the fabric shape what can be built on top and are stated here so callers do not have to rediscover them. Receive is not demultiplexed by peer, so a frame from any sender satisfies any waiting receive and a caller expecting a specific peer must check the source itself. Discovery and keepalive frames train the forwarding table but are never forwarded, so neither can carry a message to another rank. Both points are treated at length in the CnuasCCL bootstrap design.

CnuasCCL v0.2 consumes this API in production. The library plugs `cnuasdev_link_send` and `cnuasdev_link_recv` into its `cnuasccl_link_transport` vtable and wraps every collective payload frame in its own 40 byte envelope so the caller side properties above are addressed without changing the transport. The vtable is injectable, which is what makes the CnuasCCL receive path testable without a device: `lib/test/cnuasccl_link_smoke` plugs in a fake transport that counts every byte on both channels, so a regression that reached for bootstrap for payload cannot pass.

The wire format is defined normatively in the CnuasLink repository. The copy in `cnuaslink_frame.h` is a restatement, because the two live in separate repositories, and `lib/test/cnuaslink_frame_smoke` asserts the encoded bytes against literals so that a drift between them fails a test rather than producing frames the switch discards.

3. Device memory models

Backend	Backing	Default	Mapping and sharing
`host`	Process arena	512 MiB	Direct process mapping; no DMA-BUF
`hostdev`	Lazily populated kernel host pages	64 MiB	Configurable zero-copy `mmap`; pinned ranges can be exported as DMA-BUF
`device`	QEMU CnuasGPU BAR1	256 MiB	Write-combined BAR mapping; allocation ranges can be exported as peer-memory DMA-BUF

The allocator is deliberately simple

Allocation advances a pointer and does not coalesce freed ranges. A process that allocates and frees in a long loop will exhaust the device memory window even though its live footprint is small. Close and reopen the handle to reset. A free list allocator is roadmap work.

4. Character device interface

This section describes **QEMU PCIe mode**. A successfully bound `cnuasgpu.ko` instance creates `/dev/cnuasgpuN`; the Soft-GPU backend bypasses this interface.

Item	Value
Node	`/dev/cnuasgpuN`, one per device
Class	`/sys/class/cnuasgpu/`
Sysfs attributes	`gpu_id`, `sm_count`, `devmem_size`, `devmem_used`, `link_up`, `link_rx_ready`, `peer_rdma`, `export_align`, `exports`
Operations	`ioctl` for information, allocate, free, copy, DMA-BUF export/capabilities, and the three CnuasLink calls; `mmap` for direct access

4.1 Host character-device interface

Item	Value
Node	`/dev/cnuasgpu_hostN`, one per configured host device
Class	`/sys/class/cnuasgpu_host/`
Selection	`CNUAS_DEVICE_BACKEND=hostdev`
Operations	Common information/allocation/free/copy/export ioctls plus host capabilities, pinned allocation, batch submit/wait and `poll`
Defaults	Zero-copy, pinned allocation, DMA-BUF and async enabled; 64 MiB arena, 32 MiB pinned ceiling, queue depth 64, batch 32
Unsupported	CnuasLink operations and PCI/BAR identity

5. Compute backend ABI

Backends are standalone shared objects named `libcnuasrt-${name}.so`. Each one exports a single vtable symbol, `cnuas_compute_backend`, that carries a name, an ABI version pair, a capability mask, and the kernel function pointers.

Parameter	Value
Vtable symbol	`cnuas_compute_backend`
ABI version	Major 0, minor 2
Shipped backends	`libcnuasrt-scalar.so`, `libcnuasrt-avx2.so`, `libcnuasrt-avx512.so`, `libcnuasrt-cnuasir.so`
Selection	Highest capability backend the host processor supports
Discovery	`dlopen` at first compute call

Capability bits

Bit	Meaning	Implemented
`CNUAS_CCAP_VEC_F32`	32 bit float vector kernels	Yes
`CNUAS_CCAP_GEMM_F32`	32 bit float matrix multiply	Yes
`CNUAS_CCAP_GEMM_NT_F32`	32 bit float matrix multiply with the second operand transposed	Yes, except the CnuasIR backend
`CNUAS_CCAP_REDUCE`	Reduction primitives	Yes
`CNUAS_CCAP_ACTIVATION_F32`	Elementwise activations and row wise softmax	Yes, except the CnuasIR backend
`CNUAS_CCAP_CNUASIR_EXEC`	Executes CnuasIR	Yes, the CnuasIR backend only
`CNUAS_CCAP_X86_AVX2`	Host uses AVX2 code paths	Yes
`CNUAS_CCAP_X86_AVX512`	Host uses AVX-512 code paths	Yes
`CNUAS_CCAP_GEMM_F16`, `CNUAS_CCAP_GEMM_BF16`	Reduced precision matrix multiply	Reserved
`CNUAS_CCAP_CONV2D_F32`	Convolution	Reserved
`CNUAS_CCAP_FFT_C32`	Complex transform	Reserved
`CNUAS_CCAP_TENSOR_CORE`	Tensor tile unit	Reserved
`CNUAS_CCAP_ARM_NEON`, `CNUAS_CCAP_ARM_SVE`, `CNUAS_CCAP_RISCV_RVV`, `CNUAS_CCAP_FPGA`	Non x86 hosts and offload	Reserved

Reserved bits are defined in the header so that the ABI does not have to change when the corresponding kernels land. No shipped backend sets a reserved bit.

`CNUAS_CCAP_ACTIVATION_F32` covers the rectifier, Gaussian error linear unit, sigmoid linear unit, logistic, hyperbolic tangent and a row wise softmax. The CnuasIR backend is the one exception: it runs its kernels as interpreted CnuasIR bytecode, and the exponential those activations need is not in the CnuasIR v0.1 subset, so it advertises the bit as absent rather than falling back to host code behind the caller's back.

All six activations are vectorised in the AVX2 and AVX-512 backends. The vector kernels are one macro parameterised algorithm in `lib/compute/common/cnuas_activations_simd.h`, instantiated once per instruction set so that the two cannot drift apart. They have been checked by sweeping all 2^32 binary32 bit patterns per function against a double precision oracle, on AVX2 and, on AVX-512 hardware, on AVX-512, with identical error figures on both.

Two properties of that sweep are worth carrying in a datasheet. `relu` is bit exact; `sigmoid` is within 3 ulp, `tanh` within 1, and `silu` and `gelu` are within the published conformance tolerances. And `gelu` is judged by absolute rather than relative error, because for negative arguments the identity it is built on cancels catastrophically and no finite precision implementation can hold relative accuracy through that cancellation.

The AVX-512 activations are the first dependency in the tree on the DQ subset. `avx512_probe()` accordingly checks DQ as well as F and FMA; a part with F but not DQ, such as Knights Landing, would otherwise have selected the backend and then taken an illegal instruction fault.

The six activation entry points were added in ABI minor 2, and `sgemm_nt` in ABI minor 3. Each occupies the front of the previously reserved area, so the vtable size and every earlier field offset are unchanged, and a backend compiled against an earlier minor still loads with those pointers null. Callers must therefore test the capability bit, or the pointer, before calling.

`sgemm_nt` computes row-major `C = alpha * A * B' + beta * C` with `B` stored `N` by `K`. It exists because that is the shape an inference graph produces, a weight matrix held one row per output neuron applied to a batch of activations, and a caller without it must materialise the transpose in scratch first. In the vectorised backends it reuses the blocking, the microkernel and the edge handling of `sgemm` unchanged; only the operand packing reads a different address. It is exposed publicly as `cnuasSgemmNT`, and through `cnuasblasSgemm` as `CNUASBLAS_OP_T` on the first operand.

Selection controls

Environment variable	Effect
`CNUAS_COMPUTE_BACKEND`	Load the named backend, bypassing detection
`CNUAS_BACKEND_DIR`	Override the backend search directory
`CNUAS_COMPUTE_LIST`	List discovered backends
`CNUAS_COMPUTE_VERBOSE`	Explain the selection decision

Device backend selection

The device axis is independent of the compute axis above: it decides where device memory lives, not which instruction set runs.

Environment variable	Effect
`CNUAS_DEVICE_BACKEND`	`host` serves device memory from process memory; `device` requires the character device and never falls back. Unset selects the character device when a `/dev/cnuasgpuN` node is present and the in-process arena otherwise, announcing the automatic case once on stderr. Any other value is reported and ignored.
`CNUAS_HOST_ARENA_MB`	Arena size for the host backend, default 512

The `host` backend needs no kernel module, no `/dev/cnuasgpuN` and no guest. It presents one device, allocates by bump pointer with 64 byte alignment, and reclaims only the most recent block, matching the driver rather than improving on it so that code which works against it also works against the device. It reproduces the interface of the device and not its timing, so it is a correctness and bring up vehicle rather than a performance model. Allocation is mutex guarded, because collectives run their ranks as real threads. Source is `lib/libcnuasdev/cnuasdev_host.c`.

6. Validation

Feature	Test	Device required
Information ioctl, allocate, copy round trip, free	`driver/test/cnuasgpu_smoke.c`	Yes
Backend vtable numerics per instruction set	`lib/test/compute_backend_smoke.c`	No
Every activation over the whole binary32 domain, 2^32 inputs each	`lib/test/activation_exhaustive.c`	No, and out of band by cost
Host device backend: enumeration, alignment, copies in all three directions, bounds rejection, exhaustion, and mapping lifetime	`lib/test/cnuasdev_host_smoke.c`	No
Public runtime activation entry points end to end	`lib/test/cnuasrt_activation_smoke.c`	Host backend
Backend shared object ABI, capabilities, and numerics	`lib/test/compute_so_smoke.c`	No
CnuasLink wire format, golden bytes, round trip, and every rejection	`lib/test/cnuaslink_frame_smoke.c`	No
CnuasLink send and receive against a live switch	`driver/test/cnuasgpu_link_test.c`	Yes
Matrix multiply against llama.cpp's CPU reference, 648 shapes and types	`test-backend-ops` via `integration/ggml`	Host backend
Transposed matrix multiply, against a double precision oracle and bit for bit against `sgemm` on an explicitly transposed operand	`lib/test/compute_backend_smoke.c`	No
`CNUASBLAS_OP_T` producing the same product as the untransposed call	`lib/test/cnuasblas_smoke.c`	Host stub
A backend built against ABI minor 2 loading into a minor 3 loader, presenting `sgemm_nt` as null, and the public transposed entry points declining rather than faulting	Out of band, backends rebuilt from a reverted header	No

The tests that need no emulated device make up the fastest regression signal in the accelerator stack, and all of them run under `make -C lib check`. The host device backend widened that set: tests which need a device but not a real one now run there too, so only the multi device test remains gated on a guest.

7. Integration information

Item	Value
Repo	`PacketFive/CnuasGPU`
Submodule path	`src/cnuasgpu`
Source	`lib/libcnuasdev/`, `lib/compute/`

Item	Value
Headers	`cnuasdev.h`, `cnuaslink_frame.h`, `cnuas_compute_ops.h`
Debian package	`cnuas-libcnuasrt`
Depends on	`cnuasgpu.ko` and a CnuasGPU PCI device for the character device path; nothing beyond libc for the host backend
Language	C
Build	Make

8. Revision history

Revision	Notes
G	Device backend selection became automatic: the host arena is used when no character device node is present, `device` forces the character device, and `host` forces the arena. Behaviour is unchanged wherever a node exists, so this cannot mask a missing kernel module in a guest. Driven by an integration finding rather than by preference; Ollama passes only three variables to the process that loads the accelerator, so a mode reachable only by variable was not reachable at all.
H	Records the export-aware allocation lifetime and the DMA-BUF export/capability character-device operations used by `libcnuaspeermem`.
I	Clarifies that QEMU PCIe mode uses the `/dev/cnuasgpuN` nodes created by `cnuasgpu.ko`, while only the in-process Soft-GPU backend has no node.
J	Records the performance and operational tradeoff between the current in-process backend and the planned host character-device backend.
K	Added the implemented `hostdev` backend, capability/pinned/export/async APIs, discovery order and three memory models.
F	Compute backend ABI raised to minor 3 for `sgemm_nt`, with the matching `CNUAS_CCAP_GEMM_NT_F32` bit, the `cnuasSgemmNT` runtime entry point and `CNUASBLAS_OP_T` support in `cnuasblasSgemm`. Added the two section 6 validation rows. The gap was found by the ggml integration, which had been transposing its weights into scratch to work around it, and which no longer does. The minor 2 to minor 3 compatibility was verified by rebuilding two backends from a reverted header and loading them, not by inspection of the struct.
A	First publication. Symbol list taken from the dynamic symbol table of `libcnuasdev.so.0.1.0`, capability bits from `cnuas_compute_ops.h`.
B	Added the CnuasLink transport and wire format of roadmap milestone 8.2a, section 2a and the new symbols in section 2. Frame parameters taken from `cnuaslink_frame.h` and cross checked against the switch header.
E	Records the vectorised activations and their exhaustive verification, the AVX-512 DQ probe, and the ggml integration row in section 6. Error figures are measured from full sweeps on AVX2 and on AVX-512 hardware.
D	Records the host device backend of roadmap milestone 8.3e, the section 5 selection controls and the section 6 validation. Behaviour is taken from `lib/libcnuasdev/cnuasdev_host.c` and verified by running the two tests with the backend selected and deselected.
C	Compute backend ABI raised to minor 2 for the activation entry points of roadmap milestone 8.3d. The capability table records `CNUAS_CCAP_REDUCE` and `CNUAS_CCAP_CNUASIR_EXEC` as set by shipped backends, and lists `libcnuasrt-cnuasir.so`. Bits are taken from `cnuas_compute_ops.h` and cross checked against the `.caps` initialiser of every backend.

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

Cnuas Virtual AI/HPC Infrastructure is emulation software. It is not affiliated with, endorsed by, or derived from any hardware vendor, and every device it models is a software artefact. The work is published by its authors in a personal capacity and is not sponsored or endorsed by any employer.

Specifications describe the referenced revision of the software and may change without notice. Contact info@packetfive.com.