# Cnuas Math Libraries, Datasheet

Dense, deep learning, transform, sparse and solver libraries for CnuasGPU

**Document** DS-CNU-019 • **Revision** B • **Issued** 12 August 2026 • **Status** Partial, CnuasBLAS v0.1 implemented

| Item | Value |
|---|---|
| Parts | CnuasBLAS, CnuasDNN, CnuasFFT, CnuasSPARSE, CnuasSOLVER |
| Type | Numerical libraries |
| Models | cuBLAS, cuDNN, cuFFT, cuSPARSE, cuSOLVER |
| Status | Partial, CnuasBLAS v0.1 implemented, four libraries specified only |
| Repo | PacketFive/CnuasGPU |

**Partly a preview datasheet**

**CnuasBLAS v0.1 is implemented** and is described in section 4a. **CnuasDNN, CnuasFFT, CnuasSPARSE and CnuasSOLVER carry no implementation** and the rest of this datasheet records their specified design only. Note also that CnuasBLAS v0.1 covers a small part of the CnuasBLAS scope set out in section 2, and refuses everything outside it.

## 1. Overview

The five libraries are the intended numerical layer of the accelerator stack. Each mirrors a familiar NVIDIA library so that ported code finds the call it expects, and all five are specified as CnuasIR kernels reached through the runtime API.

They share one blocking dependency: every routine is a compiled kernel, so none of them can begin before CnuasCC and CnuasIR exist.

## 2. Specified scope

| Library | Models | Scope at version 1 |
|---|---|---|
| CnuasBLAS | cuBLAS | General matrix multiply, matrix vector product, scaled vector addition, batched matrix multiply |
| CnuasDNN | cuDNN | Two dimensional convolution, maximum pooling, rectified linear activation, softmax, batch normalisation |
| CnuasFFT | cuFFT | One and two dimensional forward and inverse transforms |
| CnuasSPARSE | cuSPARSE | Sparse matrix vector and sparse matrix matrix products, compressed sparse row and column formats |
| CnuasSOLVER | cuSOLVER | Dense factorisation, lower upper, Cholesky, and singular value decomposition |

## 3. Dependency order

| Order | Component | Blocked on |
|---|---|---|
| 1 | CnuasBLAS v0.1, over the shipped kernels | Nothing, startable now |
| 2 | CnuasIR instruction set freeze | Design decision |
| 3 | `cnuascc` compiler | CnuasIR |
| 4 | CnuasBLAS v0.2, CnuasDNN, CnuasFFT, CnuasSPARSE | `cnuascc` |
| 5 | CnuasSOLVER | CnuasBLAS |

Only order 1 escapes the compiler. libcnuasrt already implements a single precision general matrix multiply and the element wise vector kernels, so a first CnuasBLAS is a matter of putting the documented interface over routines that exist and pass their tests. The rest need kernels nobody can write until `cnuascc` emits them.

The relevant compute backend capability bits, covering reduced precision matrix multiply, convolution and complex transform, are already reserved in the backend ABI, so a backend can advertise them without an ABI break. See CnuasDev section 5.

One layer a deep learning library would need does now exist beneath it: the compute backends implement the common activation functions and a row wise softmax, advertised as `CNUAS_CCAP_ACTIVATION_F32`. That is a backend primitive rather than a CnuasDNN, since there is no tensor descriptor, no layer object and no convolution, but it means the elementwise part of an inference graph no longer has to be written by hand.

## 4. What exists today

| Routine | Where |
|---|---|
| Single precision general matrix multiply | `cnuasSgemm` in `libcnuasrt.so` |
| Vector add, scale, dot product in single precision | `libcnuasrt.so` |

These are fixed function entry points implemented directly in the compute backends. They are not a library in the sense used above: there is no handle, no stream, no layout or transpose selection, and no batching.

## 4a. CnuasBLAS v0.1

Implemented in `lib/libcnuasblas`, built as `libcnuasblas.so.0`. It is an adapter and contributes no kernels of its own: it presents the cuBLAS calling convention over the fixed function routines in section 4.

| Routine | Operation |
|---|---|
| `cnuasblasCreate`, `cnuasblasDestroy`, `cnuasblasGetVersion` | Handle lifetime |
| `cnuasblasSscal` | `x = alpha * x` |
| `cnuasblasSaxpy` | `y = alpha * x + y` |
| `cnuasblasSdot` | Dot product, returned to host memory |
| `cnuasblasSnrm2` | Euclidean norm, returned to host memory |
| `cnuasblasSgemv` | `y = alpha * A * x + beta * y` |
| `cnuasblasSgemm` | `C = alpha * A * B + beta * C` |

Matrices are column major, as in cuBLAS, while `cnuasSgemm` underneath is row major. The library exploits the fact that a column major matrix occupies the same bytes as its own transpose read row major, so it swaps the operands and exchanges the shapes rather than moving any data.

What v0.1 refuses, always with `CNUASBLAS_STATUS_NOT_SUPPORTED` and never by computing something else:

| Refused | Why |
|---|---|
| `CNUASBLAS_OP_T` and `CNUASBLAS_OP_C` | No transpose kernel exists |
| A vector stride other than 1 | The element wise primitives are unit stride |
| A leading dimension larger than the extent | `cnuasSgemm` takes packed operands |
| Any type other than single precision | Only single precision kernels exist |

Both restrictions lift with the kernels that arrive with CnuasCC, at which point the interface here does not change.

Verified by `lib/test/cnuasblas_host_smoke`, 39 checks, which runs on any build machine because it substitutes a host stub for the runtime, and by `lib/test/cnuasblas_smoke`, the same 39 checks against real device memory in a guest. Both are recorded in the Validation Matrix.

## 5. Interim position

Workloads needing dense linear algebra today should call CnuasBLAS v0.1 where its scope reaches, and use the host libraries for everything outside it. Numerical accuracy of the shipped kernels is checked by `sgemm_smoke` and `compute_backend_smoke`, recorded in the Validation Matrix.

## 6. Revision history

| Revision | Notes |
|---|---|
| A | First publication as a preview. Scope taken from the CnuasGPU design document, section 7.5. |
| B | CnuasBLAS v0.1 implemented, section 4a added. |

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