Package: Rggml (via r-universe)

July 21, 2026

Title Vendored 'GGML' Tensor Library with C-Callable Compute API

Version 0.1.0

Description Vendors the core, official 'GGUF' implementation, and CPU backend of the 'GGML' tensor library (<https://github.com/ggml-org/ggml>) as a static library and exposes its core tensor-context and matrix-multiply compute path through 'R_RegisterCCallable' C-callable entry points. This is a carrier package: it has no high-level R modeling API of its own. Other R packages link to it ('LinkingTo') to build and compute 'GGML' tensor graphs (including quantized types such as Q4_K) from their own C or C++ code without re-vendoring 'GGML'. The CPU backend is always built. Native targets also build the BLAS backend for dense products; webR uses the CPU backend because its Fortran ABI differs. The 'Vulkan' and NVIDIA 'CUDA' backends are independent opt-in builds.

License GPL (>= 2)

Encoding UTF-8

SystemRequirements GNU make, C++17

Biarch false

Depends R (>= 4.4.0)

Suggests tinytest

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://github.com/sounkou-bioinfo/Rfmalloc, https://sounkou-bioinfo.github.io/Rfmalloc/Rggml/

BugReports https://github.com/sounkou-bioinfo/Rfmalloc/issues

Config/pak/sysreqs make

Repository https://rgenomicsetl.r-universe.dev

Date/Publication 2026-07-18 21:34:09 UTC

RemoteUrl https://github.com/RGenomicsETL/Rfmalloc

RemoteRef HEAD

RemoteSha 6e05ad03613e89cab2f8bf8eb3a49d553d5a3bed

RemoteSubdir packages/Rggml

Contents

`ggml_version` *Vendored 'GGML' library version*

Description

Returns the version string reported by the vendored 'GGML' library at runtime (`ggml_version()`), resolved through Rggml's own registered C-callable rather than a compile-time constant, so it always reflects the code that was actually built.

Usage

```
ggml_version()
```

Value

A length-1 character vector.

Examples

```
ggml_version()
```

rggml_cpu_info *What this Rggml was actually built with*

Description

Reports the build decisions `configure` made, read off the preprocessor symbols compiled into the shared object. This is deliberately not a re-detection of the host: it answers "what is in this binary", not "what could this machine run".

Usage

```
rggml_cpu_info()
```

Details

It exists because the failure mode it guards against is invisible. A build that misses its intended branch - falling back to GGML's portable scalar kernels where the native NEON ones were meant, or to the `dgemm_` promotion where `sgemm_` was available - compiles cleanly and passes every numerical test, because both branches are correct. Only the speed differs. Asserting on this list in the test suite turns those silent fallbacks into failures.

Value

A list with

`arch_kernels` `"arm"` for GGML's hand-tuned NEON kernels, `"wasm"` for its SIMD128 kernels, and `"generic"` for the portable reference kernels.

`simd_dispatch` `TRUE` when the runtime CPUID dispatcher is active (x86: the staged AVX2 `q4_K` variant). Always `FALSE` alongside `arch_kernels` = `"arm"` or `"wasm"`, which supersede it.

`blas` `TRUE` when GGML's BLAS backend and R's Fortran bridge are part of this target build. It is `FALSE` on wasm, where webR's hidden character-length ABI is incompatible with the native bridge.

`sgemm` `TRUE` when R's BLAS exports `sgemm_` and GGML's BLAS backend calls it directly; `FALSE` when the shim promotes to `dgemm_` or when `blas` is `FALSE`.

`vulkan` `TRUE` when built with `--with-vulkan`. Whether a *device* is visible is a separate question: see `rggml_vulkan_info()`.

`cuda` `TRUE` when built with `--with-cuda`. Whether a *device* is visible is a separate question: see `rggml_cuda_info()`.

See Also

`rggml_vulkan_info()`, `rggml_cuda_info()`

Examples

```
rggml_cpu_info()
```

rggml_cuda_info *CUDA GPU backend availability*

Description

Reports the NVIDIA CUDA devices GGML can see. The backend is opt-in at installation because it requires the CUDA toolkit and compiles native GPU kernels:

Usage

```
rggml_cuda_info()

rggml_has_cuda()
```

Details

```
install.packages("Rggml", configure.args = "--with-cuda")
R CMD INSTALL --configure-args=--with-cuda .
```

Set `CUDA_HOME` when the toolkit is outside the search path. By default the source installation targets the GPU visible at build time. Set `RGGML_CUDA_ARCH` to an explicit nvcc architecture such as `sm_89` or `sm_120a` when building for another machine.

A build without CUDA returns zero devices rather than failing, so callers can probe and fall back to Vulkan, BLAS or CPU computation.

Value

A list with `n_devices` (integer) and `device` (the description of device 0, or `NA` when there is none).

`rggml_has_cuda()` returns `TRUE` when at least one CUDA device is usable.

Examples

```
rggml_cuda_info()
```

rggml_mul_mat *Matrix product on a chosen 'GGML' backend*

Description

Computes `crossprod(A, B)` (that is `t(A) %*% B`) through 'GGML' on the requested backend, using the backend-agnostic residency path: the operands and result are allocated in one of the backend's own buffers, the inputs uploaded, the product computed, and the result downloaded. For a GPU backend the tensors live in device memory, so this is the entry point a caller uses to run a dense single-precision GEMM on the GPU.

Usage

```
rggml_mul_mat(A, B, backend = c("cpu", "blas", "vulkan", "cuda"))
```

Arguments

A, B	Numeric matrices with the same number of rows (the contracted dimension). Computed in single precision on every backend.
backend	One of "cpu" (ggml_backend_cpu_init()), "blas" (offloads the dense F32 product to whatever BLAS the R build links against), or "vulkan" (a Vulkan device; requires Rggml built with --with-vulkan and a visible device, see rggml_vulkan_info()), or "cuda" (an NVIDIA CUDA device; requires Rggml built with --with-cuda, see rggml_cuda_info()). Errors if the requested backend is unavailable.

Value

A numeric matrix, dim c(ncol(A), ncol(B)), equal to crossprod(A, B) up to single-precision rounding.

See Also

rggml_vulkan_info(), rggml_cuda_info(), rggml_cpu_info()

Examples

```
A <- matrix(rnorm(12), 4, 3)
B <- matrix(rnorm(8), 4, 2)
rggml_mul_mat(A, B)              # == crossprod(A, B) to fp32
```

---

rggml_vulkan_info *Vulkan GPU backend availability*

Description

Reports the Vulkan devices GGML can see. Rggml only contains the Vulkan backend when it was **built with it** - it is opt-in, because generating and compiling GGML's 156 embedded SPIR-V shaders is expensive (the largest one needs several GB of RAM):

Usage

```
rggml_vulkan_info()

rggml_has_vulkan()
```

Details

```
install.packages("Rggml", configure.args = "--with-vulkan")
R CMD INSTALL --configure-args=--with-vulkan .
```

It also requires `glslc` and the Vulkan headers at build time (`libvulkan-dev` + `glslc` on Debian/Ubuntu, or the LunarG Vulkan SDK with `VULKAN_SDK` set), and a Vulkan driver at run time. A software driver such as Mesa's lavapipe counts: it is slow, but it makes the backend testable without a GPU.

When Rggml was built without Vulkan, this returns zero devices rather than failing, so callers can probe and fall back.

`GGML_VK_ALLOW_CPU=1` opts in to CPU-type Vulkan devices such as Mesa lavapipe. It is off by default, preserving upstream GGML's device selection.

Value

A list with `n_devices` (integer) and `device` (the description of device 0, or `NA` when there is none).

`rggml_has_vulkan()` returns `TRUE` when at least one Vulkan device is usable.

Examples

```
rggml_vulkan_info()
```

Index