

Package ‘gpuinfo’

October 14, 2026

Title Lightweight Hardware and GPU Compute Detection

Version 0.2.0

Description Detects central processing unit topology, process-available cores, host and container memory, graphics processing unit hardware, and reports the apparent availability of 'CUDA', 'Metal', 'ROCm', and 'OpenCL' compute backends, with a read-only shared-library inventory for the 'CUDA' runtime, 'cuBLAS', 'cuFFT', 'cuSOLVER', 'cuRAND', and optional 'NCCL' libraries. Detection uses operating-system information, documented platform interfaces, and optional command-line utilities, without requiring a GPU framework, 'Python', or a vendor software development kit. Backend interpretation follows the official 'CUDA' <<https://docs.nvidia.com/cuda/cuda-driver-api/>>, 'Metal' <<https://developer.apple.com/documentation/metal>>, 'ROCm' <<https://rocm.docs.amd.com/>>, and 'OpenCL' <<https://registry.khronos.org/OpenCL/>> documentation. 'NCCL' version inspection follows its documented API <<https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/api/comms.html>>. Process-level GPU memory samples, multi-device inventories, and versioned hardware manifests support reproducible benchmarks. Missing hardware, drivers, libraries, and utilities are handled safely.

License MIT + file LICENSE

URL <https://github.com/tkcaccia/gpuinfo>

BugReports <https://github.com/tkcaccia/gpuinfo/issues>

Encoding UTF-8

NeedsCompilation yes

Depends R (>= 3.6.0)

Suggests jsonlite, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Author Stefano Cacciatore [aut, cre] (ORCID:
<<https://orcid.org/0000-0001-7052-7156>>)

Maintainer Stefano Cacciatore <tkcaccia@gmail.com>
Repository CRAN
Date/Publication 2026-10-14 14:12:07 UTC

Contents

cpu_info	2
cuda_build_info	3
cuda_info	4
cuda_libraries	4
cuda_preflight	5
gpu_devices	6
gpu_memory_info	6
gpu_memory_peak	7
gpu_memory_sample	8
gpu_sitrep	9
hardware_info	10
hardware_manifest	12
has_cuda	13
has_gpu	13
has_metal	14
has_opengl	15
has_rocm	15
nccl_info	16
system_memory_info	17
Index	18

cpu_info	<i>Inspect CPU resources visible to the current process</i>
----------	---

Description

Reports CPU identity and topology using base R and read-only operating-system interfaces. available_cores estimates the processors available after affinity, container, and common scheduler limits. Unknown values are NA.

Usage

```
cpu_info(refresh = FALSE)
```

Arguments

refresh Re-run cached static discovery.

Value

A named list containing identity, topology, process-available cores, instruction-set flags when available, source, status, and reason codes.

See Also

[system_memory_info\(\)](#), [hardware_info\(\)](#)

Examples

```
cpu <- cpu_info()
cpu$logical_cores
cpu$available_cores
```

cuda_build_info	<i>Generate CUDA build architecture information</i>
-----------------	---

Description

Discovers visible device architectures and the local CUDA Toolkit/compiler layout without installing or modifying CUDA. Native code is selected for every requested architecture and PTX defaults to the highest architecture.

Usage

```
cuda_build_info(architectures = "auto", ptx = "auto", refresh = FALSE)
```

Arguments

architectures	'auto' or CUDA architectures such as 8.0, 80, 'sm_80', or 'compute_120'.
ptx	'auto' for PTX at the highest architecture, FALSE for no PTX, or one explicit architecture.
refresh	Re-run cached static CUDA discovery.

Value

A structured list containing detection status, Toolkit paths, supported/requested architectures, CMake values, and nvcc flags.

 cuda_info

Inspect CUDA hardware and system components

Description

Independently reports NVIDIA hardware, the CUDA driver library and driver initialization, the CUDA runtime library and runtime initialization, the Toolkit, and nvcc. Devices visible to the process are obtained through the CUDA driver API and reconciled with physical nvidia-smi devices by UUID. The libraries component is the consolidated table returned by [cuda_libraries\(\)](#).

Usage

```
cuda_info(refresh = FALSE)
```

Arguments

refresh Re-run cached static Toolkit and compiler discovery.

Details

driver_supported_cuda_version is the maximum CUDA version advertised by the installed driver. It is not the installed runtime or Toolkit version. runtime_version comes only from cudaRuntimeGetVersion() in a dynamically loaded CUDA runtime library. driver_initialization_status records cuInit() separately from driver_device_count_status, which records cuDeviceGetCount(). Therefore a successfully initialized driver can still have an unavailable device count, and neither state alone implies that a CUDA device is visible.

Value

A named list containing component states, versions, a CUDA-library table, status codes, visible and physical device information, and compatibility aliases.

 cuda_libraries

Inspect NVIDIA CUDA shared libraries

Description

Returns one consolidated table describing whether the CUDA driver, CUDA runtime, cuBLAS, cuFFT, cuSOLVER, cuRAND, and NCCL shared libraries can be loaded by the current R process. Loading is immediately reversed and no library handle, CUDA context, communicator, solver, FFT plan, or random generator is created.

Usage

```
cuda_libraries(refresh = FALSE)
```

Arguments

refresh Re-run cached CUDA path discovery.

Details

A loadable shared library is stronger evidence than a matching file name, because dependent-library and process-architecture problems also prevent a successful load. It does not prove that headers are installed, that every symbol required by another package exists, or that an operation will succeed. cuVS and RAFT are deliberately outside this generic CUDA component inventory and should be checked by the package that uses them.

Only the CUDA driver contributes to `has_cuda()`, together with successful driver initialization and at least one process-visible device. All other rows are informational and optional for ordinary CUDA detection.

Value

A data frame with one row per library component. `loadable` reports dynamic-loader success, `library_path` gives the loaded candidate, `version` is populated only when a safe version-only query is available, and `status/reason` fields explain unavailable results.

See Also

`cuda_info()`, `nccl_info()`, `has_cuda()`

cuda_preflight	<i>Diagnose CUDA prerequisites</i>
----------------	------------------------------------

Description

Performs read-only CUDA checks and returns structured diagnostics. It does not install software, alter environment variables, create contexts beyond normal driver/runtime enumeration, or change device visibility.

Usage

```
cuda_preflight(
  require = c("driver", "runtime"),
  architectures = "auto",
  strict = FALSE
)
```

Arguments

require Components that must pass. Supported values are 'driver', 'runtime', 'toolkit', 'compiler', and 'architecture'.

architectures Architecture request passed to `cuda_build_info()`.

strict If TRUE, fail with an error when a required component fails.

Value

A `gpuinfo_cuda_preflight` object. The `ok` element is the overall required-check result; remaining fields contain one entry per component.

<code>gpu_devices</code>	<i>List detected GPU devices</i>
--------------------------	----------------------------------

Description

Returns process-visible CUDA devices and, when `nvidia-smi` is available, physical NVIDIA devices that are not visible to the current process. CUDA driver and `nvidia-smi` records are reconciled only by UUID. Other vendors are reported through the available native or operating-system probes.

Usage

```
gpu_devices(refresh = FALSE)
```

Arguments

<code>refresh</code>	Re-run cached static CUDA discovery used while constructing the device report. Dynamic NVIDIA memory values are always queried again.
----------------------	---

Value

A data frame with one row per detected device. Memory values are bytes. `logical_id` is the CUDA process-visible ordinal; `physical_id` is the `nvidia-smi` index and may differ after visibility filtering or reordering. `visibility_status` distinguishes process-visible devices from physical-only observations.

<code>gpu_memory_info</code>	<i>Report total, used, and free GPU memory</i>
------------------------------	--

Description

Total memory is a static device property. Used and free memory are dynamic snapshots queried from `nvidia-smi` when available and are not cached. Other backends retain known total memory and return NA for dynamic values. These are device-wide values, not memory attributed to the current process; use `gpu_memory_sample()` for process-level benchmark observations.

Usage

```
gpu_memory_info(refresh = FALSE)
```

Arguments

refresh	Re-run cached static CUDA discovery. Dynamic memory is always refreshed regardless of this argument.
---------	--

Value

A data frame whose memory columns are explicitly measured in bytes.

gpu_memory_peak	<i>Calculate observed peak process memory for each GPU</i>
-----------------	--

Description

Aggregates repeated `gpu_memory_sample()` results into one row per expected GPU. Worker processes are combined within each sampling instant by UUID, and the largest complete sum is returned. Logical and physical device numbers are metadata only and are never used to match devices.

Usage

```
gpu_memory_peak(samples, expected_devices)
```

Arguments

samples	A data frame returned by <code>gpu_memory_sample()</code> , several such data frames combined with <code>rbind()</code> , or a list of sample data frames. A list preserves separate observations even if timestamps are identical.
expected_devices	A device table, normally the process-visible rows from <code>gpu_devices()</code> , or a character vector of expected GPU UUIDs. Passing this inventory is what allows completely missing GPUs to be reported as unmeasured.

Details

Use the same worker PID set for all samples. The worker set is the union of PIDs observed across the supplied samples. An instant is complete only when every worker has one row for that GPU, a finite non-negative memory value, and status measured or not_active. Incomplete instants are ignored rather than treated as zero. A device with no complete observation is returned with `peak_used_memory_bytes = NA`, status unmeasured, and an explanatory reason code.

Value

A data frame with one row per expected device, identified by UUID, containing the highest complete worker-process sum in bytes, the time of that observation, observation counts, worker count, status, and reason.

See Also

`gpu_memory_sample()`, `gpu_memory_info()`

Examples

```
devices <- gpu_devices()
if (nrow(devices)) {
  samples <- list(gpu_memory_sample(), gpu_memory_sample())
  gpu_memory_peak(samples, devices[devices$process_visible, ])
}
```

gpu_memory_sample

Sample process-level GPU memory

Description

Takes one timestamped, non-cached memory sample for one or more processes. CUDA process accounting is queried once for all requested PIDs, so the returned device rows share an observation time. Call the function repeatedly from benchmark code to construct a time series. The function does not run a background monitor and does not synchronize GPU work.

Usage

```
gpu_memory_sample(pid = Sys.getpid(), refresh = FALSE)
```

Arguments

pid	One or more distinct positive process identifiers. CUDA can report other visible processes when permitted. Metal supports only the current R process.
refresh	Re-run cached static discovery before constructing the device table. The memory observation itself is always fresh.

Details

CUDA memory is read from `nvidia-smi`'s compute-process table and matched to process-visible devices by UUID. It represents memory attributed by the NVIDIA driver to the selected process. On Windows in WDDM mode, NVIDIA may report per-process memory as unavailable; in that case the value is NA and `reason_code` explains why. A successful query with no row for the process is reported as zero bytes with status `not_active`.

Metal memory uses `MTLDevice.currentAllocatedSize` for the current R process. This is the current size of Metal resources allocated by this process; it is not resident memory, peak memory, total unified memory, or memory allocated by other processes. Metal cannot inspect an arbitrary PID through this API.

Values for unsupported or unavailable measurements are NA, never inferred from total device memory. All byte columns use bytes.

Value

A data frame with one row per PID and process-visible GPU. `measurement_status` is one of `measured`, `not_active`, `unavailable`, or `unsupported`.

See Also

[gpu_memory_info\(\)](#), [hardware_manifest\(\)](#)

Examples

```
before <- gpu_memory_sample()
# Run GPU work here, optionally synchronizing through the compute framework.
after <- gpu_memory_sample()
```

`gpu_sitrep`*Create a GPU diagnostic report*

Description

Creates a structured diagnostic report suitable for an issue, bug report, or hardware-validation record. When printed, the report includes the R and operating-system versions, CPU, visible GPUs, CUDA components, other backend results, and the selected fallback backend.

Usage

```
gpu_sitrep(format = c("text", "json"))
```

Arguments

<code>format</code>	Format used by the object's print() method. "text" displays a human-readable report. "json" displays the complete JSON representation and requires the suggested <code>jsonlite</code> package.
---------------------	---

Details

The function performs the same read-only detection as [hardware_info\(\)](#). It does not install software, initialize a framework, change device visibility, or transmit the report. Review the output before sharing it because system and environment details may identify the computer or execution environment.

Value

A visible `gpuinfo_sitrep` object containing the structured output of [hardware_info\(\)](#). The object can be inspected with normal list operations; assigning it to a variable produces no console output.

See Also

[hardware_info\(\)](#)

Examples

```
report <- gpu_sitrep()
report$cpu$vendor
print(report)
```

hardware_info	<i>Collect hardware information</i>
---------------	-------------------------------------

Description

Creates a single structured snapshot of the CPU, system memory, visible GPUs, compute backends, execution environment, and validation evidence. This is the main entry point when more detail than the `has_*()` predicates is required.

Usage

```
hardware_info(as = c("list", "json"))
```

Arguments

as	Output representation. "list" returns native R objects. "json" returns a pretty-printed JSON character scalar and requires the suggested jsonlite package. Missing values become JSON null values.
-----------	--

Details

Detection is read-only and framework-independent. It describes the hardware and system software visible to the current R process; it does not determine whether torch, TensorFlow, or another package was compiled with GPU support. Containers, schedulers, permissions, and device-visibility environment variables can affect the result. Repeated commands and native probes are reused within this call to avoid scanning the same hardware several times. The observations are discarded when the call finishes; the next call collects fresh dynamic values.

Value

A nested list containing R, system, CPU, memory, GPU, backend, environment, and package-validation details when `as = "list"`; otherwise a JSON character scalar containing the same information.

Returned components

The report contains the following named components:

r R version and platform.

system Operating-system name, release, version, and architecture.

cpu CPU identity, topology, process-available cores, features, detection source, status, and reason codes.

- memory** Host, effective container, swap, and selected-process memory, in bytes. Dynamic values are sampled on each call.
- gpu** One row per detected visible GPU, including device id, vendor, model, reported memory, backend, and NVIDIA compute capability when known. This compact table is retained for compatibility.
- gpu_devices** Detailed process-visible and physical device rows with UUIDs, visibility, byte-based memory, and CUDA architecture.
- gpu_summary** Visible and physical device counts plus a multi-GPU visibility flag. It does not claim package-specific multi-GPU execution.
- accelerators** A vendor-neutral accelerator table with memory and backend-status information.
- capabilities** Per-device precision, memory, and backend capability information. Unknown capabilities are represented by NA.
- cuda** NVIDIA GPU, driver, runtime, Toolkit, nvcc, versions, and shared-library inventory, plus apparent CUDA usability. The CUDA version advertised by `nvidia-smi` is a driver-supported maximum, not necessarily an installed Toolkit.
- nccl** Optional NCCL shared-library loading and version information, plus the process-visible CUDA device count. No communicator is created.
- metal** macOS platform, Apple silicon, device, framework, support, and apparent Metal usability.
- rocm** AMD GPU, kernel driver, runtime, utilities, version, and apparent ROCm usability.
- openccl** Loader, platforms, devices, vendors, memory, FP64 support, and apparent OpenCL availability.
- available_backends** Detected usable backends, always including the CPU fallback.
- best_backend** The first usable backend in the package priority order CUDA, Metal, ROCm, OpenCL, then CPU.
- environment** Container, CI, WSL, scheduler, and device-visibility information. Values are reported without changing the environment.
- validation** Real-hardware validation evidence shipped with this package version. This describes maintainer testing, not the current host.

See Also

[gpu_sitrep\(\)](#), [has_gpu\(\)](#), [has_cuda\(\)](#), [has_metal\(\)](#), [has_rocm\(\)](#), [has_openccl\(\)](#)

Examples

```
info <- hardware_info()
names(info)
info$cpu
info$gpu
info$cuda$usable
```

hardware_manifest

Create a reusable hardware manifest

Description

Creates a stable, serializable record of the host, process-visible GPU devices, generic compute components, and an optional memory snapshot. The manifest is intended to accompany benchmark results. It contains no FAISS, cuVS, machine-learning-framework, or package-specific capability checks.

Usage

```
hardware_manifest(  
  memory = c("process", "device", "none"),  
  pid = Sys.getpid(),  
  refresh = FALSE,  
  as = c("list", "json")  
)
```

Arguments

memory	Memory observation to include. "process" uses gpu_memory_sample() , "device" uses gpu_memory_info() , and "none" omits memory values.
pid	Process identifier used when memory = "process".
refresh	Re-run cached static discovery. Dynamic memory is always sampled again.
as	Output representation. "list" returns a <code>gpuinfo_manifest</code> object; "json" returns JSON and requires the suggested <code>jsonlite</code> package.

Value

A versioned list or JSON character scalar. Memory units are bytes.

See Also

[hardware_info\(\)](#), [gpu_memory_sample\(\)](#)

Examples

```
manifest <- hardware_manifest(memory = "none")  
manifest$`schema_version`  
manifest$`components$cuda$driver_version`
```

`has_cuda`*Test whether CUDA appears usable*

Description

Returns TRUE only when at least one CUDA device is visible to the current process through a successfully initialized NVIDIA CUDA driver. The CUDA runtime, Toolkit, nvcc, FAISS, cuVS, and machine-learning frameworks are independent and are not required.

Usage

```
has_cuda()
```

Value

One non-missing logical value.

See Also

[cuda_info\(\)](#), [hardware_info\(\)](#), [has_gpu\(\)](#)

Examples

```
has_cuda()
cuda_info()$devices
```

`has_gpu`*Test whether a GPU is detected*

Description

Returns TRUE when at least one graphics processor is visible to the current R process. Detection combines native vendor-library queries with portable operating-system and command-line fallbacks when available.

Usage

```
has_gpu()
```

Details

This tests hardware visibility, not whether any particular compute backend or R framework can use the device. A GPU with a missing driver can therefore be detected while all backend predicates remain FALSE. Conversely, a container or scheduler can hide a physical GPU from the process.

Value

One non-missing logical value: TRUE when a visible GPU is detected, otherwise FALSE.

See Also

[hardware_info\(\)](#), [has_cuda\(\)](#), [has_metal\(\)](#), [has_rocm\(\)](#), [has_opengl\(\)](#)

Examples

```
has_gpu()
```

has_metal

Test whether Metal appears usable

Description

Returns TRUE on macOS when a Metal-capable graphics device is visible. Detection uses the Metal framework when it can be loaded and supplements it with `system_profiler` and Apple-silicon platform information.

Usage

```
has_metal()
```

Details

This reports operating-system and hardware capability, not framework-specific support such as torch MPS. A positive result therefore does not guarantee that another R package can execute work through Metal. Detailed fields are available in `hardware_info()$metal`.

Value

One non-missing logical value: TRUE when Metal appears usable, otherwise FALSE. It is FALSE on non-macOS platforms.

See Also

[hardware_info\(\)](#), [has_gpu\(\)](#)

Examples

```
has_metal()
hardware_info()$metal
```

has_opengl	<i>Test whether OpenCL is available</i>
------------	---

Description

Returns TRUE when an OpenCL loader is present, at least one platform can be enumerated, and at least one compute device is visible. Detection uses the OpenCL library directly and can supplement it with the read-only `clinfo` utility.

Usage

```
has_opengl()
```

Details

OpenCL devices can be GPUs, CPUs, or other accelerators, so this predicate is not synonymous with [has_gpu\(\)](#). It also does not guarantee that a particular R package has OpenCL support. Inspect `hardware_info()$opengl` for platform, device, vendor, memory, and FP64 details.

Value

One non-missing logical value: TRUE when OpenCL appears available, otherwise FALSE.

See Also

[hardware_info\(\)](#), [has_gpu\(\)](#)

Examples

```
has_opengl()
hardware_info()$opengl
```

has_rocm	<i>Test whether ROCm appears usable</i>
----------	---

Description

Returns TRUE when an AMD GPU is visible and a usable ROCm/HIP runtime path is apparent. Detection can use the HIP runtime library, `/dev/kfd`, `rocm_info`, `rocm-smi`, `hipcc`, and common ROCm installation locations.

Usage

```
has_rocm()
```

Details

Merely finding ROCm files is insufficient: the predicate requires compatible visible hardware and runtime evidence. A positive result does not guarantee that another R package was compiled with ROCm support. Inspect `hardware_info()$rocm` for the individual detection fields and version.

Value

One non-missing logical value: TRUE when ROCm appears usable, otherwise FALSE.

See Also

[hardware_info\(\)](#), [has_gpu\(\)](#)

Examples

```
has_rocm()
hardware_info()$rocm
```

nccl_info

Inspect the NCCL shared library

Description

Loads the NCCL shared library and calls `ncclGetVersion()` without creating a communicator or changing device selection. A successful version query does not prove that two GPUs can communicate. Packages using NCCL must still validate their own build and communicator. The probe is supported on Linux, where it also checks `NCCL_LIBRARY`, `NCCL_HOME`, and `NCCL_ROOT` before normal CUDA and system library paths.

Usage

```
nccl_info(refresh = FALSE)
```

Arguments

refresh	Re-run cached static CUDA path discovery. This does not alter the environment or install software.
---------	--

Value

A list with library presence, loaded path or soname, decoded and integer library versions, version-query status, structured diagnostic fields, and the number of process-visible CUDA devices. `communication_tested` is always FALSE.

See Also

[cuda_info\(\)](#), [hardware_info\(\)](#)

system_memory_info	<i>Inspect host and process memory</i>
--------------------	--

Description

Returns a point-in-time snapshot. Host totals are kept separate from the effective memory visible inside a container or control group. All sizes are numeric byte counts, and dynamic values are never cached.

Usage

```
system_memory_info(pid = Sys.getpid())
```

Arguments

pid	Process identifier whose resident memory should be sampled.
-----	---

Value

A named list containing host, effective, swap, and selected-process memory values plus source, status, and reason codes.

See Also

[cpu_info\(\)](#), [gpu_memory_info\(\)](#), [gpu_memory_sample\(\)](#)

Examples

```
memory <- system_memory_info()
memory$total_memory_bytes
memory$effective_available_memory_bytes
```

Index

`cpu_info`, [2](#)
`cpu_info()`, [17](#)
`cuda_build_info`, [3](#)
`cuda_build_info()`, [5](#)
`cuda_info`, [4](#)
`cuda_info()`, [5](#), [13](#), [16](#)
`cuda_libraries`, [4](#)
`cuda_libraries()`, [4](#)
`cuda_preflight`, [5](#)

`gpu_devices`, [6](#)
`gpu_devices()`, [7](#)
`gpu_memory_info`, [6](#)
`gpu_memory_info()`, [8](#), [9](#), [12](#), [17](#)
`gpu_memory_peak`, [7](#)
`gpu_memory_sample`, [8](#)
`gpu_memory_sample()`, [6–8](#), [12](#), [17](#)
`gpu_sitrep`, [9](#)
`gpu_sitrep()`, [11](#)

`hardware_info`, [10](#)
`hardware_info()`, [3](#), [9](#), [12–16](#)
`hardware_manifest`, [12](#)
`hardware_manifest()`, [9](#)
`has_cuda`, [13](#)
`has_cuda()`, [5](#), [11](#), [14](#)
`has_gpu`, [13](#)
`has_gpu()`, [11](#), [13–16](#)
`has_metal`, [14](#)
`has_metal()`, [11](#), [14](#)
`has_opengl`, [15](#)
`has_opengl()`, [11](#), [14](#)
`has_rocm`, [15](#)
`has_rocm()`, [11](#), [14](#)

`nccl_info`, [16](#)
`nccl_info()`, [5](#)

`print()`, [9](#)

`system_memory_info`, [17](#)

`system_memory_info()`, [3](#)