



CUDA Compatibility Matrix: Driver, Toolkit, Container, GPU

Plan and build private AI compute with GPU Smith. We connect workload requirements with GPU hardware, networking, deployment and operating decisions so your infrastructure fits the work it must perform.

Published by GPU Smith · 2026-10-18 · cuda compatibility matrix · cuda driver toolkit compatibility · nvidia driver cuda version · pytorch cuda compatibility · tensorflow cuda cudnn · nvidia container toolkit · cuda forward compatibility · cuda minor version compatibility · cuda docker image

Original article: <https://gpusmith.com/articles/en/cuda-compatibility-matrix-driver-toolkit-container-gpu>



In this article

1. Executive Summary
 2. Introduction and Background
 3. The Six Meanings of "CUDA Version" and a One-Minute Inventory
 4. Compatibility Mechanisms: Directions, Conditions and Limits
 5. The Layered Compatibility Ledger
 6. Framework, Library and Container Layers
 7. Decision Procedure and Failure Signatures
 8. Data Analysis and Evidence
 9. Production Preflight, Rollback Evidence and Ownership
 10. Implications and Future Directions
 11. Frequently Asked Questions (FAQs)
 12. Conclusion
-

Executive Summary

"CUDA version" is not one number. On a single GPU server it can mean the compute capability of the chip, the installed NVIDIA driver, the CUDA runtime inside a framework wheel or container, a compatibility package, the versions of cuDNN, TensorRT and NCCL, or the label on a framework build or container tag. This report separates those meanings into a layer-by-layer ledger so that a proposed change can be classified as upgrade, hold, rebuild or test. All facts were read from vendor or project pages during this research session (retrieval date 18 October 2026).

NVIDIA documents three mechanisms. Within a major release family, minor version compatibility allows a limited feature set above a family floor: R580 for CUDA 13.x and R525 for CUDA 12.x. Across CUDA 11.x, the floor is 450.80.02 on Linux or 452.39 on Windows ([NVIDIA compatibility guide](#)). The release notes page is titled CUDA Toolkit 13.4 Update 2 (docs.nvidia.com), and each 13.x minor maps to a driver branch: 13.0 to R580, 13.1 to R590, 13.2 to R595, 13.3 to R610 and 13.4 to R615 (docs.nvidia.com). Forward compatibility, a package that lets a newer toolkit run on an older base driver, is limited to systems with data center GPUs, select NGC Server Ready RTX SKUs and Jetson (docs.nvidia.com).

Each mechanism has an edge. Minor version compatibility cannot run PTX (Parallel Thread Execution, NVIDIA's intermediate code) on older drivers (docs.nvidia.com), and a binary with neither a matching cubin nor PTX fails at kernel launch (docs.nvidia.com). Library and framework layers add their own floors: cuDNN 9.27.0 ships separate CUDA 12.x and 13.x builds (docs.nvidia.com), and PyTorch 2.14 lists stable CUDA 12.6, 13.0 and 13.2 builds (raw.githubusercontent.com). The 2.14 CUDA 13 wheels start at Turing (compute capability 7.5), so Pascal and Volta GPUs stay on the CUDA 12.6 build (raw.githubusercontent.com).

The practical conclusion is that "runnable" and "supported" are separate columns. This report supplies a blank ledger schema (YAML), a decision procedure, a table of failure signatures and a preflight checklist. It publishes only rows that carry an exact source and a retrieval date, and it marks everything else as a test to run.

Introduction and Background

Private AI platform operators rarely upgrade one thing at a time. A new PyTorch release arrives with a different CUDA runtime, a container tag is rebuilt on a newer base image, a security team schedules a driver update, and a procurement team adds a new GPU generation. Each change is documented in a different place by a different owner, so fleet outages and unnecessary driver changes often come from conflating several meanings of "CUDA version". This report is written for platform operators, release engineers, framework owners and infrastructure buyers who must decide whether a proposed driver, toolkit, cuDNN, framework, container and GPU combination is supported and deployable.

GPU Smith is an independent private AI infrastructure engineering consultancy. It does not manufacture GPUs, sell cloud capacity or publish a framework, so this report takes the position of an adjacent advisor: it relies on dated vendor and project documentation, states where sources disagree, and presents no benchmark or client result. Anything not documented in a cited source is labelled a test to run, not a conclusion.

The scope is a dependency-resolution method, not a compute-feature lookup. Instruction-level feature tables per architecture, and version-specific rollout runbooks for Triton, vLLM, the GPU Operator and ROCm, are out of scope.

Three terms recur. **Backward compatibility** means a newer driver works with an older toolkit (docs.nvidia.com). **Minor version compatibility** means an application built within one CUDA major family runs on an older driver of the same family, with limits. **Forward compatibility** means a separately installed package lets a newer toolkit run on an older base driver. **Vendor support status** is not a mechanism: it is whether a vendor or project lists a combination in its own matrix, which is a different question from whether the combination runs.

Two caveats apply. The combinatorial space is too large for an exhaustive matrix, so every row below carries a source and a date. And many cited pages are rolling documents: NVIDIA's frameworks support matrix states "Last updated on Oct 15, 2026" (docs.nvidia.com), and Google Kubernetes Engine's GPU page states "Last updated 2026-10-13 UTC." (cloud.google.com). Re-check any copied row against the live page.

The Six Meanings of "CUDA Version" and a One-Minute Inventory

The six meanings

- **GPU architecture (compute capability).** NVIDIA's CUDA GPU list gives the GeForce RTX 4090 as compute capability 8.9 (developer.nvidia.com) and the H100 as 9.0 (developer.nvidia.com). Cubins (compiled GPU binaries) do not cross major capability: NVIDIA's programming guide states that its GPUs are not binary compatible between major compute capability versions (docs.nvidia.com).
- **Installed driver.** The kernel-mode driver and the user-mode CUDA driver (libcuda) are the layer that the operating system, the fleet manager and the container runtime depend on.
- **CUDA runtime and toolkit.** The runtime library (cudart) ships with the toolkit, not the driver. From CUDA 13.4 the Linux driver is distributed separately and is no longer bundled with the toolkit (docs.nvidia.com).

- **Compatibility package.** The forward compatibility package for CUDA 13.4 is built using the NVIDIA driver libraries version 615 (docs.nvidia.com).
- **Libraries.** cuDNN, TensorRT and NCCL carry their own versions and CUDA-major builds.
- **Framework build or container tag.** A label such as a PyTorch cu132 wheel index (pytorch.org), or an NGC container named by year and month.

Above all six sits the **application binary layer**: which cubins (SASS, native GPU machine code) and which PTX are embedded. It decides whether a binary can reach a GPU that did not exist when it was built.

Why the nvidia-smi CUDA field does not settle the question

The nvidia-smi manual defines the CUDA field as the latest CUDA version the driver supports (docs.nvidia.com), and an NVIDIA staff answer on the developer forum reads it the same way (forums.developer.nvidia.com). It is a ceiling set by the driver. It does not report the toolkit on the host, the runtime a container carries, or what a framework wheel was built against.

The one-minute inventory

Collect these facts on a representative node before any change and attach them to the ticket:

- **Host evidence.** The nvidia-smi driver version, GPU names and kernel module flavor. The open kernel modules run on Turing and later GPUs (github.com), and Blackwell and later are supported only by the open flavor (us.download.nvidia.com).
- **Compute capability.** One value per GPU model. The deviceQuery sample prints driver and runtime versions side by side (github.com).
- **Framework evidence.** PyTorch documents `torch.cuda.get_arch_list()` as returning the CUDA architectures its library was compiled for (docs.pytorch.org).
- **Container evidence.** Image digest and tag, toolkit version, and the `NVIDIA_REQUIRE_CUDA` variable that official CUDA images set (gitlab.com). Docker's guide describes the sequence of drivers, toolkit, then the `--gpus` flag (docs.docker.com).
- **Library evidence.** cuDNN, TensorRT and NCCL versions from the package manager or the build, never inferred from the toolkit label.

```
nvidia-smi --query-gpu=name,driver_version --format=csv
python -c "import torch; print(torch.version.cuda, torch.cuda.get_arch_list(),
torch.cuda.get_device_capability())"
docker run --rm --gpus all IMAGE nvidia-smi
```

Compatibility Mechanisms: Directions, Conditions and Limits

Table 1 below lists each mechanism, the direction it runs in, the documented condition, and what it does not promise.

Mechanism	Direction	Documented condition	What it does not guarantee
Backward compatibility	Newer driver, older toolkit	Applications built with CUDA 12.9 or earlier keep running on all hardware the driver supports (developer.nvidia.com)	That the library or framework above the toolkit lists the newer driver
Minor version compatibility	Older driver, newer toolkit in one major family	Floors: CUDA 11.x needs Linux 450.80.02 or Windows 452.39; CUDA 12.x needs R525; CUDA 13.x needs R580	Full features: applications run with a limited feature set (docs.nvidia.com); PTX device code does not work on older drivers
Forward compatibility package	Older base driver, newer toolkit across major families	Eligible GPUs only (see the Executive Summary); the package is built from a specific driver branch	Anything beyond the CUDA user-mode driver: other components stay on the old version
PTX just-in-time (JIT) compilation	Older binary, newer GPU	PTX can be JIT compiled for any GPU of equal or higher compute capability (developer.nvidia.com)	Load time: JIT compilation happens when the application starts, and architecture-specific targets get no such fallback
Framework build support	Set by the framework project	The project's own matrix, such as PyTorch's release compatibility matrix (raw.githubusercontent.com)	That a combination outside the matrix fails, or that one inside it is tested on your fleet
Vendor support status	Set by each vendor's lifecycle	A production branch of NVIDIA application software has a 9-month support window (docs.nvidia.com)	That a runnable combination stays in a supported branch

The directions can be drawn as a single diagram. Each arrow reads "can run on".

```

Layer order: GPU (sm_XY) => kernel + user-mode driver => CUDA runtime (wheel or image) => libraries =>
framework build => container
Backward:      older toolkit          => newer driver
Minor:         newer toolkit          => older driver, same major family (SASS present, limited features)
Forward:       newer toolkit + cuda-compat package => older base driver, other major family (eligible GPUs
only)
PTX JIT:       older PTX-bearing binary => newer GPU (same or higher compute capability)

```

Only the first four rows change what the machine does. The last two state what a party will stand behind. Runnable is not supported, and neither is the same as tested on your hardware.

Reading the driver-version tables

For toolkits up to 12.9 a table lists corresponding toolkit driver versions for Linux x86_64 and Windows: CUDA 12.0 GA at 525.60.13 or later, 12.4 GA at 550.54.14, 12.6 GA at 560.28.03, 12.8 GA at 570.26 and 12.9 Update 1 at 575.57.08 on Linux (docs.nvidia.com). For 13.x the notes give a branch per minor and one family floor. A per-update Linux and Windows build number for 13.x was not found in the pages read, and none is invented here. Existing CUDA 13.x applications run on drivers 580 and later, but new 13.4 features and newly enabled platforms require R615 (docs.nvidia.com).

Forward compatibility in detail

The package provides only libraries and does not configure the system to find them (docs.nvidia.com), so the operator or container entrypoint must set the loader path. Because the package supplies a newer CUDA driver library, the PTX restriction of minor version compatibility does not apply to it. In the support matrix, the package for CUDA 13.4 is compatible with base driver branches 535, 570, 580, 590, 595 and 610 and not applicable on 615 (docs.nvidia.com).

A caution follows: a branch can appear in the compatibility matrix and be past its lifecycle date on NVIDIA's separate data center driver lifecycle page, so a forward-compatibility plan on an older branch should read both. When the visible GPU is ineligible, the driver API reports that the system was upgraded to run with forward compatibility but the visible hardware detected by CUDA does not support it (docs.nvidia.com).

PTX and SASS: the application layer

Compiler flags decide which rules can rescue a binary. A typical broad-compatibility build embeds cubins for known targets plus PTX for the newest (developer.nvidia.com), and that PTX can be JIT compiled for any GPU of equal or higher compute capability (developer.nvidia.com). Minor version compatibility needs SASS and an explicit target, so a PTX-only fallback is unavailable on an older driver. PyTorch's extension documentation shows the build-side control: the +PTX option makes extension kernel binaries include PTX instructions (docs.pytorch.org).

The Layered Compatibility Ledger

What the ledger records

A ledger row answers: for this layer, what is installed or requested, what dependency does the source document, which mechanism lets it work, and what evidence exists on your fleet. The columns proposed are layer, installed or requested version, minimum and maximum documented dependency, support status, mechanism, exact source, source date, retrieval date, test evidence, a "runnable is not supported" flag and a decision. *Table 2* below fills in documented state for the stack current on 18 October 2026. A cell is left without a claim when no source was found.

Layer	Documented state (retrieved 2026-10-18)	Dependency documented by the source	Runnable is not supported
GPU	RTX 4090 is 8.9 (developer.nvidia.com); H100 is 9.0 (developer.nvidia.com)	Offline compilation for compute capability below 7.5 was removed in CUDA 13.0 (developer.nvidia.com)	Pre-Turing GPUs keep running CUDA 12.9 or earlier binaries but a CUDA 13 build cannot target them

Layer	Documented state (retrieved 2026-10-18)	Dependency documented by the source	Runnable is not supported
Kernel driver	R580 is a long-term support branch (developer.nvidia.com); CUDA 13.4 maps to R615	R580 is the last branch to support pre-Turing GPUs (developer.nvidia.com)	A driver below the toolkit's own branch but above the family floor runs with limited features
Toolkit	Release notes titled 13.4 Update 2; the toolkit archive page lists previous releases (developer.nvidia.com)	Family floors: 11.x Linux 450.80.02 / Windows 452.39; 12.x R525; 13.x R580	A runtime from a newer major family than the driver supports needs a forward package or a driver change
Compatibility package	Built from driver 615 for CUDA 13.4	Data center GPUs, select NGC Server Ready RTX SKUs and Jetson only	On ineligible GPUs the package can be installed yet rejected at initialization
cuDNN	9.27.0, the latest in the cuDNN documentation read	CUDA 13.x build lists a Linux driver of 615.71.09 (docs.nvidia.com)	A driver that satisfies the toolkit family floor (580) can sit below the library's own floor
TensorRT	11.4.0	Compute capability SM 7.5 or higher (docs.nvidia.com)	The CUDA 13 build is compatible with CUDA 13.x only, so a CUDA 12 host stack needs the separate CUDA 12 packages
NCCL	2.32.3	PyTorch 2.14 lists NCCL 2.30.7 with its CUDA 13.0 build (raw.githubusercontent.com)	Mixed NCCL versions across ranks should be treated as a configuration error
Framework wheel	PyTorch 2.14: CUDA 12.6, 13.0 and 13.2 stable	CUDA 13 wheels start at Turing (raw.githubusercontent.com)	A wheel absent from the project matrix is "not listed", which is not the same as broken
Container	NGC 26.09, CUDA 13.4.1 based	26.x notes point to the compatibility guide for driver information	Overriding an image's version checks lets it start without making the combination supported
Application binary	Unknown until inspected	Compatible cubin or usable PTX for each kernel	Kernel launch fails if neither is available. Ordinary cubins can run on GPUs with the same major and an equal or higher minor compute capability; a later GPU release date alone does not imply failure (NVIDIA guide). Architecture-specific targets have stricter rules: their PTX and cubins are not forward-compatible with future GPU architectures (NVIDIA explanation).

The ledger is deliberately narrow. It does not turn "driver 580" into "supports every CUDA 13 library", because no source says that. The cuDNN row shows why: the cuDNN support matrix lists a Linux driver floor of 615.71.09 for its CUDA 13.x build, which is above the family floor of 580 that governs the toolkit. A decision that reads only the toolkit table would accept a stack the library page does not list.

Blank ledger schema (YAML)

The YAML below is a blank schema to copy into a repository and fill row by row. A row with no source must be marked unverified, not omitted.

Save the file as cuda-compat-ledger.yaml.

```
ledger_version: 1
retrieved_on: YYYY-MM-DD
owner: team-or-person
rows:
  - layer: gpu|driver|cuda_runtime|compat_package|cudnn|tensorrt|nccl|framework|container|app_binary
    installed_or_requested: ""
    min_documented_dependency: ""
    max_documented_dependency: ""
    support_status: supported|listed_experimental|not_listed|unverified
    mechanism: backward|minor_version|forward_package|ptx_jit|library_build|none
    source_url: ""
    source_date: YYYY-MM-DD|none
    retrieval_date: YYYY-MM-DD
    runnable_is_not_supported: true|false
    test_evidence: ["import", "kernel", "library_call", "multi_gpu"]
    decision: upgrade|hold|rebuild|test
```

Framework, Library and Container Layers

PyTorch

PyTorch wheels are published to separate indexes per CUDA runtime: the previous-versions page gives a CUDA 13.2 command for torch 2.13.0 on the cu132 index (pytorch.org), and the Start Locally selector gives a CUDA 12.6 command on the cu126 index (pytorch.org). A maintainer answer on the PyTorch forum says the binaries ship with all CUDA runtime dependencies (discuss.pytorch.org), so the host needs only a sufficient driver. That is community evidence, not a specification.

The project's release compatibility matrix is the authoritative table (raw.githubusercontent.com). For 2.14 it lists CUDA 13.0 with cuDNN 9.24.0.43 and NCCL 2.30.7 (raw.githubusercontent.com), and the in-development 2.15 row lists CUDA 13.4 with cuDNN 9.26.0.51 (raw.githubusercontent.com). For 2.14 the CUDA 12.6 build still includes Maxwell, Pascal and Volta (raw.githubusercontent.com), while the CUDA 13 builds start at Turing and add Blackwell (raw.githubusercontent.com). PTX for the Blackwell 12.0 target is available on Linux builds only (raw.githubusercontent.com), and the 2.9 release enabled Linux aarch64 wheels across all supported CUDA versions (pytorch.org).

The CUDA set has moved with recent releases. The 2.7 blog introduced Blackwell support with pre-built CUDA 12.8 wheels (pytorch.org). The 2.11 blog made CUDA 13 the default (pytorch.org). The 2.12 blog told Pascal and Volta users to switch to the CUDA 12.6 wheel (pytorch.org) and gave a Blackwell driver floor of 580.65.06 on Linux and 580.88 on Windows for CUDA 13.0 wheels (pytorch.org). The 2.13 blog records that the CUDA 12.8 and 12.9 builds were removed (pytorch.org). A proposal for 2.15 states that removing CUDA 12.6 would drop binary support for pre-Turing architectures (github.com). A maintainer comment says the intent is that 12.x builds will be removed entirely (github.com), and an earlier 2.12 proposal had kept CUDA 12.6 to provide continued support for older GPU architectures (github.com). The 2.15 thread was open when read on 2026-10-15, so it is a plan under discussion, not a shipped change. A 2.14 discussion also noted that CUDA 13.2 requires a newer NVIDIA driver than 13.0 (github.com), so a fleet on R580 should treat a cu132 wheel as a test case.

TensorFlow, JAX and ONNX Runtime

TensorFlow publishes a table of tested build configurations (www.tensorflow.org). The 2.21.0 row read for this report lists CUDA 12.5 and cuDNN 9.3, while the pip page still describes CUDA toolkit 12.3 and cuDNN 8.9.7, so the two pages disagree. The GitHub release notes record pip packages built with CUDA 12.3 and cuDNN 8.9.7 from 2.16 (raw.githubusercontent.com), hermetic CUDA builds from 2.18 (raw.githubusercontent.com), and the removal of Maxwell (compute capability 5.x) from binary distributions in 2.17 (raw.githubusercontent.com). The pip page gives a Linux driver floor of 525.60.13 (www.tensorflow.org), says there is no GPU support on native Windows after TF 2.10 (www.tensorflow.org), and says packages carry no PTX except for the latest supported architecture (www.tensorflow.org).

JAX documents a CUDA 13 support floor of SM 7.5 (docs.jax.dev), driver floors per CUDA major on Linux (docs.jax.dev), a recommendation to migrate to the CUDA 13 wheels (docs.jax.dev), and a pointer to forward compatibility packages for a newer toolkit on an older driver (docs.jax.dev).

ONNX Runtime states minor-version behavior explicitly. Builds with CUDA 11.8 are compatible with any CUDA 11.x (onnxruntime.ai), builds with CUDA 12.8 require CUDA 12.8 or newer (onnxruntime.ai), and cuDNN 8.x and 9.x builds are not interchangeable (onnxruntime.ai). It documents compatibility with PyTorch's CUDA builds provided both use the same CUDA and cuDNN major versions (onnxruntime.ai), an example of coupling that a driver table cannot show.

Library packaging by CUDA major

NVIDIA's CUDA 13.0 blog says toolkit package names lose the CUDA tag suffix (developer.nvidia.com) while non-toolkit packages keep it (developer.nvidia.com), which is why PyPI carries names such as `nvidia-cudnn-cu12` (pypi.org).

Containers and the NVIDIA Container Toolkit

The model is host driver plus toolkit plus image runtime. The toolkit's README says the CUDA Toolkit need not be installed on the host (github.com), and at container start it injects the requested GPUs and driver (docs.nvidia.com). The Docker Hub page says the toolkit is required to run CUDA images (hub.docker.com), and lists three image flavors (hub.docker.com). Docker's guide adds that CUDA images set the GPU

environment variables automatically (docs.docker.com). Release notes for toolkit 1.20.1 also tie one feature (per-GPU memory limits) to R615 or later drivers (github.com), a reminder that the toolkit has its own driver dependencies.

Official images carry a gate. The toolkit enforces the `NVIDIA_REQUIRE_*` constraints, and when the host driver is insufficient the container does not start (docs.nvidia.com). The base image installs the compatibility package where the repository offers it (gitlab.com), and NVIDIA's compatibility FAQ says minor and forward compatibility work in NGC and CUDA base containers.

Image lifetimes matter for rebuild planning. CUDA image tags have a lifetime (hub.docker.com), only the latest point release of each supported toolkit is supported (gitlab.com), and toolkit images reach end of life when the driver they shipped with does (gitlab.com).

Cloud, Kubernetes and distributions

Managed platforms add another owner for the driver. AWS states that GPU instances need the appropriate NVIDIA driver installed (docs.aws.amazon.com). GKE tells users to check the minimum driver for their CUDA version (cloud.google.com), and lists R535 as default with R570, R575 or R580 available for GKE 1.31 to 1.33 (cloud.google.com). AWS publishes dated images such as the Deep Learning Base OSS Nvidia Driver GPU AMI for Ubuntu 24.04, release 20261006 (docs.aws.amazon.com), and notes that on its pre-installed driver images the CUDA toolkit package depends on the drivers (docs.aws.amazon.com). Azure states that its driver extensions do not automatically update the driver after install (learn.microsoft.com), and that the latest supported CUDA driver for K80-based NC-series VMs is 470.82.01 (learn.microsoft.com). Kubernetes places driver installation with the node administrator (kubernetes.io). Canonical will transition installations to the next supported driver branch automatically (documentation.ubuntu.com), and Red Hat states that Blackwell GPUs require the open kernel modules (access.redhat.com), with a driver tool that detects the accelerator hardware present (www.redhat.com). For conda users, `__cuda` is the maximum CUDA version supported by the display driver (docs.conda.io), and CUDA packages are installable without a driver but will not run properly (conda-forge.org).

Decision Procedure and Failure Signatures

The procedure

Apply the same order to any proposed change. Stop at the first failing step and classify the change as hold, rebuild or test.

1. **Name the changed layer** among the ten ledger layers, and the layers it implicitly moves (a new base image moves runtime, cuDNN and NCCL together).
2. **Check the GPU against the build.** Compare each GPU's compute capability to the build's architecture list (docs.pytorch.org). Without a matching cubin or PTX the launch fails. Result: rebuild or hold.
3. **Check the driver against every library, not only the toolkit.** Use the family floor, then each library's own row.

4. **Identify the mechanism relied on.** A newer driver running an older toolkit uses backward compatibility, including across CUDA major families; it does not need a forward package for that reason. A driver older than the toolkit's corresponding driver but within the major family's compatibility range can use minor version compatibility: confirm native code for each GPU and check newer driver-dependent features and PTX restrictions. For a newer runtime on an older base driver that needs forward compatibility, verify the exact package against NVIDIA's base-driver matrix and GPU eligibility list ([release notes](#); [forward compatibility guide](#)). Different major labels alone do not determine the mechanism.
5. **Check support status** in the framework and container matrices. An absent combination is "not listed", not "broken".
6. **Run the preflight** on a canary node, then roll out in stages with the previous image digest and driver package pinned.

(Hypothetical Example) Version resolution with declared sample values

The values below are declared for illustration, not measured. A node has an H100 (compute capability 9.0) on the R580 driver branch, and a team proposes two PyTorch 2.14 wheels.

- **Candidate A, a cu130 wheel.** CUDA 13.0 maps to R580, so the driver matches the toolkit's branch. Hopper is in the 2.14 CUDA 13 architecture list ([raw.githubusercontent.com](#)). The open question is the driver floor of the bundled cuDNN: the matrix lists cuDNN 9.24.0.43 ([raw.githubusercontent.com](#)), but the cuDNN page read for this report documents a floor only for 9.27.0, so no floor for 9.24 is stated. Decision: test, cuDNN row marked unverified.
- **Candidate B, a cu132 wheel.** CUDA 13.2 maps to R595, so R580 sits below the toolkit's branch but at the 13.x family floor of 580. The run would rely on minor version compatibility with limited features, and the project has noted that 13.2 needs a newer driver ([github.com](#)). Decision: test under that bounded mechanism, or raise the driver branch first.

Neither case invents a supported row. The procedure yields rows that need a source and a test.

Failure signatures and first checks

Table 3 maps common failure messages to their documented meaning and the first ledger row to inspect.

Signature	Documented meaning	First check
Insufficient driver	The installed NVIDIA CUDA driver is older than the CUDA runtime library (docs.nvidia.com)	Compare the runtime's toolkit family to the host driver floor, or the image's NVIDIA_REQUIRE_CUDA
Unsupported architecture ("no kernel image")	There is no kernel image available that is suitable for the device (docs.nvidia.com)	Compare the GPU's compute capability to the build's architecture list; check for PTX
Forward package rejected	Forward compatibility is configured but the visible hardware does not support it	Check for a GeForce or workstation GPU in the visible set; restrict visible devices
Host and container confusion	The container starts only when the host driver satisfies the image's gate	Confirm which nvidia-smi was run (host or container) and what the toolkit injected

Signature	Documented meaning	First check
Library ABI mismatch	Libraries built for different majors do not interoperate, for example cuDNN 8 and 9 (onnxruntime.ai)	Print each library version inside the running process and compare to the framework matrix
Driver and library version mismatch	Userspace packages were upgraded while the kernel module is still on the older version (documentation.ubuntu.com)	Compare loaded module and installed package versions; reboot or reload

Most errors point to one layer, but one symptom ("CUDA error") can come from several, so the first check is always an inventory comparison, not a reinstall. Missing symbol errors at import time have no NVIDIA error code; they can indicate a library built for a different CUDA major than the process loaded.

Data Analysis and Evidence

Driver floors in one place

NVIDIA distinguishes corresponding toolkit driver versions from minor-version-compatibility floors ([release notes](#)). The following corresponding versions are for Linux x86_64; the compatibility floor is shown separately.

Toolkit release or family	Corresponding toolkit driver (Linux)	Minor-version-compatibility floor
CUDA 11.x family	Varies by release	Linux 450.80.02; Windows 452.39
CUDA 12.0 GA	525.60.13	R525 family floor
CUDA 12.4 GA	550.54.14	R525 family floor
CUDA 12.6 GA	560.28.03	R525 family floor
CUDA 12.8 GA	570.26	R525 family floor
CUDA 12.9 Update 1	575.57.08	R525 family floor
CUDA 13.x family	Branch varies by minor release	R580 family floor

All floors mean the stated version or later. Under [minor version compatibility](#), newer driver-dependent features and PTX can require a driver upgrade; compile native code for the target GPU. These floors do not replace library or GPU requirements. CUDA 13.4 new features and newly enabled platforms require an R615 or later driver that supports them. The `nvidia-smi` ceiling alone does not resolve these requirements.

Lifecycle windows

An NVIDIA engineering blog states that the 580 long-term support driver will be supported for three years from release (developer.nvidia.com). For application software, NVIDIA's AI Enterprise lifecycle page gives production branches a 9-month support window. Across one 36-month LTS driver lifetime, a fleet therefore sees up to four consecutive 9-month production windows (36 divided by 9). That is arithmetic on the cited windows, not a published NVIDIA statistic.

Release churn at the framework layer

The PyTorch cadence table lists a planned 2.14 release on 2 September 2026 and a planned 2.15 release on 28 October 2026, and marks future dates tentative ([raw.githubusercontent.com](https://raw.githubusercontent.com/pytorch/pytorch/master/RELEASE.md)). Across 2.11 to 2.13 the offered CUDA set changed in every release: 2.11 made CUDA 13 the default, 2.12 deprecated the CUDA 12.8 wheel while keeping 13.0 as default (pytorch.org), and 2.13 removed the 12.8 and 12.9 builds. A team that pins a CUDA runtime variant had a change event to track at each release in that range.

Discrepancies worth recording

- **Release labels and artifact baselines.** The [toolkit archive](#) lists CUDA 13.4.3 as the latest release; its linked documentation uses the [13.4 Update 2 release label](#). The CUDA image tree uses 13.4.3 ([base-image source](#)). The [cuDNN recommendation](#) is a tuning baseline: cuDNN 9.27.0 with CUDA 13.4.1, rather than a claim about the latest toolkit. These labels and baselines do not establish a disagreement about the latest release. Record the exact version of each artifact in the ledger.
- **TensorFlow documentation.** The source-build and pip pages disagree on CUDA and cuDNN versions, as noted above.
- **Per-update driver minimums for 13.x.** Not published as Linux and Windows build numbers in the pages read.

Production Preflight, Rollback Evidence and Ownership

Preflight evidence

A preflight is a scripted run on a canary node whose preserved output becomes the test evidence for the ledger. Items run from cheapest to most expensive:

- **Manifest.** Save the filled ledger, nvidia-smi output, image digest and the lock file. conda-forge notes that mixing and matching CUDA components from the toolkit is currently disallowed (github.com), a reminder to pin as a set.
- **Representative import and model load.** Import the framework, load a representative model, print the architecture list.
- **Kernel execution on every GPU model.** CUDA samples compile for all architectures supported by their release by default (github.com).
- **Library calls.** Run a convolution (cuDNN), a matrix multiply (cuBLAS) and, if used, a TensorRT engine build and load.
- **Multi-GPU path.** Run an NCCL all-reduce across the full intended topology; treat mixed NCCL versions across ranks as a configuration error. Use [NVIDIA's NCCL tests](#), which check both performance and correctness of NCCL operations.
- **Preserved logs and rollback pins.** Keep logs, the previous image digest, the previous driver package and kernel module flavor together. If a forward package is used, configure its library path explicitly.

Refresh policy and ownership

Assign each layer an owner and a refresh trigger. The driver owner (platform operations) refreshes when a branch enters its last year of support or when a Blackwell-class GPU is added, since open kernel modules become the default in the 615 installer (us.download.nvidia.com). The framework owner (ML platform) refreshes at every framework release. The container owner (release engineering) refreshes against the container lifecycle. The architect or buyer re-checks GPU eligibility before procurement: pre-Turing GPUs are outside the CUDA 13 build targets (developer.nvidia.com). A quarterly review of all rows, plus a review on any change to a driver, toolkit, library or framework major version, is a reasonable minimum; the retrieval date on each row makes stale rows visible.

Implications and Future Directions

Three trends in the documents read shape the next two to four quarters. First, CUDA 13 moved the architecture floor to Turing, and frameworks are following: the PyTorch 2.15 proposal would remove the CUDA 12.6 build that carries Pascal and Volta (github.com), and the interim route for those GPUs is the CUDA 12.6 wheel. A fleet with pre-Turing GPUs has a bounded window for staying on current framework releases.

Second, the driver is increasingly a separately managed artifact. CUDA 13.4 stops bundling the Linux driver, cloud images pin a driver and toolkit pairing (docs.aws.amazon.com), and managed Kubernetes selects drivers by node patch level (cloud.google.com). The split between the team that updates drivers and the team that updates frameworks is structural, and the ledger is the shared artifact that lets them negotiate.

Third, kernel module flavor is becoming part of compatibility. Open modules are available on Turing and later (github.com) and are the only supported flavor for Blackwell and later (us.download.nvidia.com). Mixed-generation fleets may need to track module flavor per node class.

For infrastructure buyers, the implication is to request a ledger rather than a single version number from vendors: per GPU SKU, which driver branches, toolkit families and framework builds are listed, with dates. From the perspective of an independent advisor on enterprise GPU procurement and air-gapped deployments, the ledger is a natural part of acceptance documentation; GPU Smith's contact details are at gpusmith.com. The limits are plain: this report records documented compatibility, benchmarks nothing, and cannot replace testing on the exact hardware, firmware and image of a given fleet. Organizations with regulated or safety-sensitive workloads should have a qualified engineer review their own deployment decisions.

Frequently Asked Questions (FAQs)

What is the CUDA compatibility matrix, and where is the official one?

There is no single matrix. NVIDIA publishes driver-to-toolkit minimums in its release notes and the compatibility model in its CUDA compatibility guide; library, framework and container matrices live on separate pages linked above.

How do CUDA driver and toolkit compatibility work, and where is the driver table?

A newer driver works with an older toolkit. An older driver can run a newer toolkit within the same major family down to the family floor, with a limited feature set. The release notes distinguish the 13.x branch map and corresponding driver versions for 12.9 and earlier from the minor-version-compatibility minimums, and GKE's documentation points operators to the same NVIDIA table (cloud.google.com).

How does PyTorch CUDA version compatibility work?

Wheels are published per CUDA runtime on separate indexes, and the project matrix lists stable builds per release. The 2.12 release gave a Blackwell driver floor of 580.65.06 on Linux for CUDA 13.0 wheels (pytorch.org), and the previous-versions page lists the cu132 command for 2.13.0 (pytorch.org).

What are the TensorFlow CUDA and cuDNN requirements?

The source-build page lists tested configurations, the release notes record a CUDA 12.3 and cuDNN 8.9.7 baseline from 2.16, and native Windows has no GPU support after TF 2.10.

How does the NVIDIA Container Toolkit relate to CUDA compatibility?

The toolkit injects the host driver into the container, and the host does not need the CUDA Toolkit. Official images can refuse to start when the host driver is insufficient.

What is CUDA forward compatibility, in plain terms?

A package that supplies a newer CUDA user-mode driver so a newer toolkit can run on an older base driver, for CUDA only. NVIDIA's eligibility list covers data center GPUs, select NGC Server Ready RTX SKUs and Jetson boards; verify the exact GPU and package/base-driver combination in the [forward compatibility guide](#).

Which GPUs does a CUDA version support, and what suits an RTX 4090 or H100?

CUDA 13.0 removed offline compilation for compute capability below 7.5, so Maxwell, Pascal and Volta need CUDA 12.9 or earlier to build (developer.nvidia.com). The RTX 4090 is compute capability 8.9 (developer.nvidia.com) and the H100 is 9.0 (developer.nvidia.com), and the PyTorch 2.14 CUDA 13 wheels list Hopper (raw.githubusercontent.com). The RTX 4090 is a GeForce board and the forward compatibility eligibility list does not name GeForce.

What driver does a CUDA Docker image require?

It depends on the image. Official images set `NVIDIA_REQUIRE_CUDA`, and the container does not start if the host driver is insufficient; the NGC 26.x notes refer readers to the compatibility guide for numbers.

Conclusion

The question "is this CUDA version compatible?" has no single answer because it is asked about several layers at once. NVIDIA documents three mechanisms that change what can run together: backward compatibility, minor version compatibility within a family, and forward compatibility packages for eligible GPUs:

data center GPUs, select NGC Server Ready RTX SKUs and Jetson boards. Library floors, framework matrices, container lifecycles and cloud image pairings are documented by separate parties on separate schedules.

The method proposed here is therefore procedural. Inventory the six meanings of "CUDA version", add the application binary layer, fill a ledger with sourced rows and retrieval dates, name the mechanism each row relies on, and decide among upgrade, hold, rebuild and test. Keep "runnable" and "supported" as separate columns, and leave unsourced cells empty. The preflight and rollback evidence turn the decision into a reviewable artifact, and a dated ledger is a better basis for it than a version number remembered from the last upgrade.

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. <https://docs.nvidia.com/deploy/cuda-compatibility/minor-version-compatibility.html>
2. <https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html>
3. <https://docs.nvidia.com/deploy/cuda-compatibility/forward-compatibility.html>
4. <https://docs.nvidia.com/cuda/blackwell-compatibility-guide/index.html>
5. <https://docs.nvidia.com/deeplearning/cudnn/backend/latest/reference/support-matrix.html>
6. <https://raw.githubusercontent.com/pytorch/pytorch/main/RELEASE.md>
7. <https://docs.nvidia.com/deploy/cuda-compatibility/why-cuda-compatibility.html>
8. <https://docs.nvidia.com/deeplearning/frameworks/support-matrix/index.html>
9. <https://cloud.google.com/kubernetes-engine/docs/how-to/gpus>
10. <https://developer.nvidia.com/cuda-gpus>
11. <https://docs.nvidia.com/cuda/cuda-programming-guide/01-introduction/cuda-platform.html>
12. <https://pytorch.org/get-started/previous-versions>
13. <https://docs.nvidia.com/deploy/nvidia-smi/index.html>
14. <https://forums.developer.nvidia.com/t/on-nvidia-smi-and-nvcc-version-have-asynchronous-cuda/323777>
15. <https://github.com/NVIDIA/open-gpu-kernel-modules>
16. https://us.download.nvidia.com/XFree86/Linux-x86_64/615.78.08/README/kernel_open.html
17. https://github.com/NVIDIA/cuda-samples/blob/master/cpp/1_Uutilities/deviceQuery/deviceQuery.cpp
18. https://docs.pytorch.org/docs/stable/generated/torch.cuda.get_arch_list.html
19. <https://gitlab.com/nvidia/container-images/cuda/-/raw/master/dist/13.4.3/ubuntu2404/base/Dockerfile>
20. <https://docs.docker.com/engine/containers/gpu>
21. <https://developer.nvidia.com/blog/navigating-gpu-architecture-support-a-guide-for-nvidia-cuda-developers>
22. <https://developer.nvidia.com/blog/understanding-ptx-the-assembly-language-of-cuda-gpu-computing>
23. <https://docs.nvidia.com/ai-enterprise/lifecycle/latest/application-software.html>
24. https://docs.nvidia.com/cuda/cuda-driver-api/group__CUDA__TYPES.html
25. <https://developer.nvidia.com/blog/nvidia-blackwell-and-nvidia-cuda-12-9-introduce-family-specific-architecture-features>

26. https://docs.pytorch.org/docs/stable/cpp_extension.html
27. <https://developer.nvidia.com/blog/whats-new-and-important-in-cuda-toolkit-13-0>
28. <https://developer.nvidia.com/cuda-toolkit-archive>
29. <https://docs.nvidia.com/deeplearning/tensorrt/latest/getting-started/support-matrix.html>
30. <https://pytorch.org/get-started/locally>
31. <https://discuss.pytorch.org/t/pytorch-and-cuda-version-compatibility/216242>
32. <https://pytorch.org/blog/pytorch-2-9>
33. <https://pytorch.org/blog/pytorch-2-7>
34. <https://pytorch.org/blog/pytorch-2-11>
35. <https://pytorch.org/blog/pytorch-2-12>
36. <https://pytorch.org/blog/pytorch-2-13>
37. <https://github.com/pytorch/pytorch/issues/190385>
38. <https://github.com/pytorch/pytorch/issues/178665>
39. <https://github.com/pytorch/pytorch/issues/190355>
40. <https://www.tensorflow.org/install/source>
41. <https://raw.githubusercontent.com/tensorflow/tensorflow/master/RELEASE.md>
42. <https://www.tensorflow.org/install/pip>
43. <https://docs.jax.dev/en/latest/installation.html>
44. <https://onnxruntime.ai/docs/execution-providers/CUDA-ExecutionProvider.html>
45. <https://pypi.org/project/nvidia-cudnn-cu12>
46. <https://github.com/NVIDIA/nvidia-container-toolkit>
47. <https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/troubleshooting.html>
48. <https://hub.docker.com/r/nvidia/cuda>
49. <https://github.com/NVIDIA/nvidia-container-toolkit/releases>
50. <https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/docker-specialized.html>
51. <https://gitlab.com/nvidia/container-images/cuda/-/raw/master/doc/support-policy.md>
52. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/install-nvidia-driver.html>
53. <https://cloud.google.com/kubernetes-engine/docs/concepts/gpus>
54. <https://docs.aws.amazon.com/dlami/latest/devguide/aws-deep-learning-ami-gpubaseoss-ul2404-2026-10-08.html>
55. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/preinstalled-nvidia-driver.html>
56. <https://learn.microsoft.com/en-us/azure/virtual-machines/extensions/hpccompute-gpu-linux>
57. <https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus>
58. https://documentation.ubuntu.com/server/_sources/how-to/graphics/install-nvidia-drivers.md.txt
59. <https://access.redhat.com/solutions/7140700>
60. <https://www.redhat.com/en/blog/introducing-new-and-simplified-ai-accelerator-driver-experience-rhel>
61. <https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-virtual.html>
62. https://conda-forge.org/docs/maintainer/knowledge_base

- 63. https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html
- 64. <https://github.com/conda-forge/cuda-version-feedstock/blob/main/recipe/README.md>
- 65. <https://github.com/NVIDIA/cuda-samples>
- 66. <https://github.com/NVIDIA/nccl-tests>

THE PUBLISHER

About GPU Smith

GPU Smith provides independent engineering and research for private AI infrastructure. We help technical buyers and operators reason about GPU workload sizing, hardware selection, procurement, deployment and inference operations, from the compute system to the networking, power and cooling requirements around it.

Start with the workload

Useful infrastructure decisions begin with the models, data, throughput, latency and operating constraints a team actually has. GPU Smith helps connect those requirements with system design and deployment choices. The goal is a privately controlled AI environment whose capacity and operational demands are understood before a purchasing decision.

Hardware and supplier research

Our public reference library covers GPUs, complete systems and networking components. The server-vendor directory and manufacturing research help teams investigate suppliers, compare options and follow sources behind technical claims. We distinguish manufacturer specifications from measured benchmarks and advertised capabilities from tested configurations. Reference pages are research resources, not a live inventory listing or a binding equipment quote.

Deployment and operations

GPU Smith's engineering scope includes infrastructure integration, networking, power, cooling and the practical operation of inference workloads. Our published articles explain the tradeoffs behind deployment, procurement and ongoing operations so teams can ask better questions and document decisions.

Work with GPU Smith

Explore the [hardware library](#), [GPU references](#), [networking references](#) and [vendor research](#). [Contact GPU Smith](#) with your workloads and deployment constraints to discuss a project and confirm current scope, pricing and availability.

Inclusion of a manufacturer or product in our research does not imply a partnership, certification or endorsement.

Website: <https://gpusmith.com>

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.