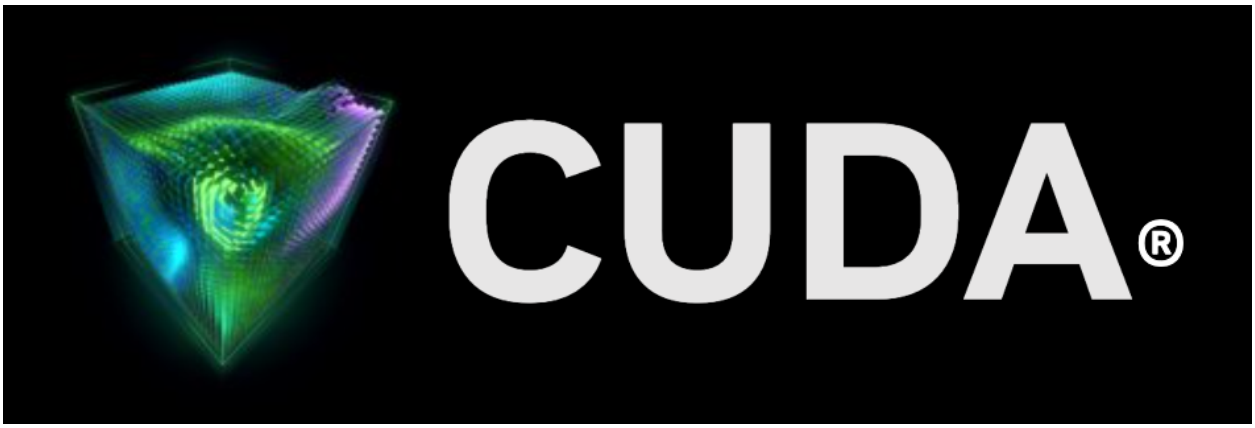

Release Notes

Release 13.4

NVIDIA Corporation

Sep 11, 2026

Contents

Chapter 1

Overview

Welcome to the release notes for NVIDIA® CUDA® Toolkit 13.4. This release includes enhancements and fixes across the CUDA Toolkit and its libraries.

This documentation is organized into two main sections:

- **CUDA Platform**
  Focuses on the core CUDA infrastructure including component versions, driver compatibility, compiler/runtime features, issues, and deprecations.
- **CUDA Libraries**
  Covers the specialized computational libraries with their feature updates, performance improvements, API changes, and version history across CUDA 13.x releases.

Chapter 2

CUDA Platform

2.1. CUDA Toolkit Major Components

Note:
Individual components within the CUDA Toolkit (for example: compiler, libraries, tools) are versioned independently.
The CUDA Toolkit 13.4 GA release is versioned 13.4.1 and supersedes the 13.4.0 developer preview.
The third digit is a build number and does not indicate an update release.

For CUDA 13.4, the table below indicates the versions:

Table 1: CUDA 13.4 Component Versions

Component Name		Version Information	Supported Architectures	Supported Platforms
CUDA C++ Core Compute Libraries	Thrust	3.4.2	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
	CUB	3.4.2		
	libcu++	3.4.2		
	Cooperative Groups	13.3.4.2.1		
CUDA Compatibility Package (Orin)		13.4.46343957	arm64-sbsa	Linux
CUDA Application Compiler (crt)		13.4.59	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA Compilation Optimizer (ctadvisor)		13.4.49	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA Runtime (cudart)		13.4.49	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA culibos		13.4.49	x86_64, arm64-sbsa	Linux
CUDA cuobjdump		13.4.49	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUPTI		13.4.58	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA cuxxfilt (demangler)		13.4.49	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows

continues on next page

Table 1 – continued from previous page

Component Name	Version Information	Supported Architectures	Supported Platforms
CUDA Documentation	13.4.49	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA GDB	13.4.49	x86_64, arm64-sbsa	Linux
CUDA NVCC	13.4.59	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA nvdisasm	13.4.49	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA NVML Headers	13.4.61	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA nvprune	13.4.49	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA NVRTC	13.4.59	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA NVTX	13.4.49	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA OpenCL	13.4.49	x86_64, arm64 (Windows)	Linux, Windows
CUDA Profiler API	13.4.49	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA Sandbox dev	13.4.49	x86_64, arm64-sbsa	Linux
CUDA Compute Sanitizer API	13.4.57	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA TILE-IR AS	13.4.59	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA cuBLAS	13.7.0.27	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA cuDLA	13.4.49	arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA cuFFT	12.4.0.34	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA cuFile	1.19.0.109	x86_64, arm64-sbsa	Linux
CUDA cuobjclient	1.3.0.109	x86_64, arm64-sbsa	Linux
CUDA cuRAND	10.4.4.49	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA cuSOLVER	12.3.2.15	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA cuSPARSE	12.8.6.49	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA NPP	13.2.0.35	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA nvFatbin	13.4.49	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA nvJitLink	13.4.52	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA nvJPEG	13.2.2.35	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows

continues on next page

Table 1 – continued from previous page

Component Name	Version Information	Supported Architectures	Supported Platforms
CUDA nvptxcompiler	13.4.59	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
CUDA nvvm	13.4.59	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
Nsight Compute	2026.3.0.13	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
Nsight Systems	2026.3.2.313	x86_64, arm64-sbsa, arm64 (Windows)	Linux, Windows
Nsight Visual Studio Edition (VSE)	2026.3.0.26187	x86_64, arm64 (Windows)	Windows
Visual Studio Integration	13.4.49	x86_64, arm64 (Windows)	Windows

2.2. CUDA Driver

Note:
The NVIDIA driver is **no longer bundled** with the CUDA Toolkit – on Windows starting with CUDA 13.1, and on Linux starting with CUDA 13.4. Download and install the appropriate driver from the official NVIDIA Driver Downloads page.

Running a CUDA application requires a system with at least one CUDA-capable GPU and a driver that is compatible with the CUDA Toolkit. For more information about various GPU products that are CUDA-capable, visit `https://developer.nvidia.com/cuda/gpus`.

The NVIDIA driver branch corresponding to each CUDA Toolkit release is shown below. Update releases within a CUDA minor version use the same driver branch.

Table 2: CUDA Toolkit and Corresponding Driver Branch

CUDA Toolkit	Corresponding Driver Branch
CUDA 13.4	R615
CUDA 13.3	R610
CUDA 13.2	R595
CUDA 13.1	R590
CUDA 13.0	R580

Note:
Existing CUDA 13.x applications run on drivers >=580 under CUDA minor version compatibility. CUDA 13.4 new features and newly enabled platforms require an R615 or later driver that supports them. The Windows driver for the RTX Spark device is 616.41 or later.

The CUDA driver is backward compatible: applications compiled against a particular CUDA Toolkit version continue to work on subsequent (later) driver releases. In addition, CUDA minor version compatibility allows applications to run on a driver older than the corresponding driver branch, within the ranges shown below. The installed driver must meet or exceed the minimum required version for the CUDA Toolkit. For details, see the CUDA Compatibility Guide and CUDA Compatibility and Upgrades.

Table 3: CUDA Toolkit and Minimum Required Driver Version for CUDA Minor Version Compatibility

CTK Version	Driver Range for Minor Version Compatibility	
	Min	Max
13.x	>= 580	N/A
12.x	>= 525	< 580
11.x	>= 450	< 525

CUDA 11.0 shipped with earlier driver versions. Minor-version compatibility across the CUDA 11.x family requires driver version 450.80.02 or later on Linux, or 452.39 or later on Windows.

Older CUDA versions (12.9 and earlier)

Table 4: Corresponding Driver Versions (CUDA 12.9 and earlier)

CUDA Toolkit	Corresponding Driver Version	
	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 12.9 Update 1	>=575.57.08	>=576.57
CUDA 12.9 GA	>=575.51.03	>=576.02
CUDA 12.8 Update 1	>=570.124.06	>=572.61
CUDA 12.8 GA	>=570.26	>=570.65
CUDA 12.6 Update 3	>=560.35.05	>=561.17
CUDA 12.6 Update 2	>=560.35.03	>=560.94
CUDA 12.6 Update 1	>=560.35.03	>=560.94
CUDA 12.6 GA	>=560.28.03	>=560.76
CUDA 12.5 Update 1	>=555.42.06	>=555.85
CUDA 12.5 GA	>=555.42.02	>=555.85
CUDA 12.4 Update 1	>=550.54.15	>=551.78
CUDA 12.4 GA	>=550.54.14	>=551.61
CUDA 12.3 Update 1	>=545.23.08	>=546.12
CUDA 12.3 GA	>=545.23.06	>=545.84
CUDA 12.2 Update 2	>=535.104.05	>=537.13
CUDA 12.2 Update 1	>=535.86.09	>=536.67
CUDA 12.2 GA	>=535.54.03	>=536.25

continues on next page

Table 4 – continued from previous page

CUDA Toolkit	Corresponding Driver Version	
CUDA 12.1 Update 1	>=530.30.02	>=531.14
CUDA 12.1 GA	>=530.30.02	>=531.14
CUDA 12.0 Update 1	>=525.85.12	>=528.33
CUDA 12.0 GA	>=525.60.13	>=527.41
CUDA 11.8 GA	>=520.61.05	>=520.06
CUDA 11.7 Update 1	>=515.48.07	>=516.31
CUDA 11.7 GA	>=515.43.04	>=516.01
CUDA 11.6 Update 2	>=510.47.03	>=511.65
CUDA 11.6 Update 1	>=510.47.03	>=511.65
CUDA 11.6 GA	>=510.39.01	>=511.23
CUDA 11.5 Update 2	>=495.29.05	>=496.13
CUDA 11.5 Update 1	>=495.29.05	>=496.13
CUDA 11.5 GA	>=495.29.05	>=496.04
CUDA 11.4 Update 4	>=470.82.01	>=472.50
CUDA 11.4 Update 3	>=470.82.01	>=472.50
CUDA 11.4 Update 2	>=470.57.02	>=471.41
CUDA 11.4 Update 1	>=470.57.02	>=471.41
CUDA 11.4.0 GA	>=470.42.01	>=471.11
CUDA 11.3.1 Update 1	>=465.19.01	>=465.89
CUDA 11.3.0 GA	>=465.19.01	>=465.89
CUDA 11.2.2 Update 2	>=460.32.03	>=461.33
CUDA 11.2.1 Update 1	>=460.32.03	>=461.09
CUDA 11.2.0 GA	>=460.27.03	>=460.82
CUDA 11.1.1 Update 1	>=455.32	>=456.81
CUDA 11.1 GA	>=455.23	>=456.38
CUDA 11.0.3 Update 1	>= 450.51.06	>= 451.82
CUDA 11.0.2 GA	>= 450.51.05	>= 451.48
CUDA 11.0.1 RC	>= 450.36.06	>= 451.22
CUDA 10.2.89	>= 440.33	>= 441.22
CUDA 10.1 (10.1.105 general release, and updates)	>= 418.39	>= 418.96
CUDA 10.0.130	>= 410.48	>= 411.31
CUDA 9.2 (9.2.148 Update 1)	>= 396.37	>= 398.26
CUDA 9.2 (9.2.88)	>= 396.26	>= 397.44
CUDA 9.1 (9.1.85)	>= 390.46	>= 391.29
CUDA 9.0 (9.0.76)	>= 384.81	>= 385.54
CUDA 8.0 (8.0.61 GA2)	>= 375.26	>= 376.51
CUDA 8.0 (8.0.44)	>= 367.48	>= 369.30
CUDA 7.5 (7.5.16)	>= 352.31	>= 353.66

continues on next page

Table 4 – continued from previous page

CUDA Toolkit	Corresponding Driver Version	
CUDA 7.0 (7.0.28)	>= 346.46	>= 347.62

2.3. New Features

2.3.1. CUDA Platform

- CUDA 13.4 adds support for Windows on Arm on RTX Spark devices.
- Vera Rubin platform support is available in CUDA 13.4 as a developer preview. Vera Rubin support in CUDA Toolkit 13.4 is not intended for benchmarking, performance analysis, or production deployment.
- Separately released R615 driver packages no longer include the proprietary kernel modules; supported Linux systems use the NVIDIA open kernel modules.
- **Note:** The following change requires the R615 driver (corresponding to the CUDA 13.4 Toolkit) or later. On all NVIDIA hardware-coherent platforms, the NVIDIA driver now defaults to Coherent Driver-based Memory Management (CDMM) instead of onlining GPU memory to the operating system as a NUMA node. NUMA mode remains fully supported. The mode is a node-wide setting controlled by a kernel module parameter and takes effect after a driver reload or reboot, so it should be selected before upgrading.
  To select CDMM mode:

  ```
  echo 'options nvidia NVreg_CoherentGPUMemoryMode=driver' | sudo tee
  ↪  /etc/modprobe.d/nvidia-openrm.conf
  ```

  To select NUMA mode:

  ```
  # Loading NVIDIA module instructions
  echo 'options nvidia NVreg_CoherentGPUMemoryMode=numa' \
  | sudo tee /etc/modprobe.d/nvidia-openrm.conf

  # NVIDIA-UVM module parameter
  echo 'options nvidia-uvm uvm_disable_sam_migration=false' \
  | sudo tee -a /etc/modprobe.d/nvidia-openrm.conf
  ```

  Reload the driver or reboot for the change to take effect. To verify the active mode:

  ```
  grep Coherent /proc/driver/nvidia/params

  # Check for SAM migration
  cat /sys/module/nvidia_uvm/parameters/uvm_disable_sam_migration
  ```

  For a comparison of both modes, see the CUDA Memory Management blog post.
- Added `cuMemGetLocationInfo` and `cudaMemGetLocationInfo` APIs for querying residency information for Unified Memory allocations, enabling libraries and runtimes to select compute and communication resources based on current data locality.
- CUDA memory pool IPC is now supported on Windows WDDM systems for allocations created with the stream-ordered allocator.
- CUDA MPS adds a new command-line interface, file-based configuration, process namespaces, cgroup-integrated device-memory limits, and per-container time slicing for GPU fractionalization. For more information, see the CUDA Toolkit 13.4 release blog.

- `cuda-checkpoint` can now write checkpoint state to persistent storage instead of retaining it only in process memory.
- CUDA and NVML add cgroup-based GPU-memory limits. Allocation and memory-reporting APIs now honor the configured limit, and NVML and `nvidia-smi` expose corresponding management support.
- CUDA Graphs now support associating Green Contexts with event-record, host, external-semaphore, and kernel nodes. User-specified contexts on kernel nodes are now honored consistently.
- On supported multi-chip GPUs, CUDA APIs expose GPU topology and allow applications to select compute and memory resources according to locality.
- CUDA Compute Fabric Transport (CFT) is enhanced for handle-based fabric programming. For more information, see the CUDA Toolkit 13.4 release blog.
- CUDA device attribute query introduces the new attribute `CU_DEVICE_ATTRIBUTE_LOGICAL_ENDPOINT_SUPPORTED_HANDLE_TYPES` to indicate the supported IPC handle types for logical endpoints.
- Structured CPER (Common Platform Error Record) delivery for GPU errors is available as a preview.

2.3.2. CUDA Developer Tools

For details on new features, improvements, and bug fixes, see the changelogs for:

- Nsight Systems.
- Nsight Visual Studio Edition.
- CUPTI.
- Nsight Compute.
- Compute Sanitizer.
- NVML.
- cuda-gdb.

2.3.3. CUDA Compiler

- PTX ISA 9.4 is supported. For new PTX features, see the PTX ISA 9.4 documentation.
- Added support for GCC 16 and Clang 22 as host-side compilers with NVCC.
- Added support for libc++ 22 as a C++ standard library with NVCC.
- Added support for concatenated-entry fatbins in `nvFatbin` and `fatbinary` for cross-entry similarity compression.

2.3.4. CUDA C++ Core Libraries (CCCL)

- CCCL 3.4 significantly improves `cub::DeviceScan` performance on Blackwell with a warp-specialized implementation using the Tensor Memory Accelerator (TMA).
- CUB now provides single-call, environment-based overloads for all device-wide algorithms, removing the required temporary-storage query phase while retaining the traditional two-phase APIs.
- Added `cub::WarpReduceBatched` for reducing multiple independent batches across a warp.
- Added `cuda::std` parallel algorithms selected with the `cuda::execution::gpu` execution policy.
- Deprecated Thrust constant and strided iterators in favor of their `cuda::` equivalents, along with default-alignment overloads on type-erased memory-resource wrappers and several legacy helper APIs.
- CCCL 3.4.1 and 3.4.2 add scan and reduction fixes, updated tuning, OpenMP scan bounds fixes, and disable Tile compilation mode.
- For the full changelog, see the CCCL 3.4.0 release notes.

2.3.5. CUDA Python

- cuda-bindings 13.4.0 adds the new CUDA Toolkit 13.4 Driver, Runtime, cuFile, and NVML APIs, including fabric-cluster and clique queries, memory-location information, graph node v3 APIs, checkpoint completion, vectored cuFile I/O, adaptive TGP, memory-limit, GPU-fabric, performance-metric, NVLink-telemetry, and operational-event interfaces.
- Added an experimental `cuda.bindings._v2.nvrtc` API with exception-based error handling, PEP 8-compliant names, and improved performance.
- cuda-bindings now raises `OverflowError` for out-of-range `c_int` and `c_byte` kernel arguments, adds linked-LTOIR support for `nvJitLink`, improves thread safety, and adds Windows Arm64 build support.
- cuda.core 1.1.1 adds `ObjectCode.get_module()` for legacy `CUmodule` interoperability, ships C++ headers in wheels and source distributions, fixes graph-node resource lifetimes and DLPack handling, hardens program-cache permissions, and supports prerelease cuda-bindings version strings such as `13.4.0b1`.
- cuda-pathfinder 1.6.0 improves CUDA Toolkit binary discovery and adds library and header discovery for cuQuantum libraries, cuTENSORMp, and Windows Arm64 cuDLA.
- cuda.compute 1.1 adds serialization and deserialization of algorithm objects, ahead-of-time compilation for multiple compute capabilities and GPU-less systems, and build-cache fixes. Version 1.1.1 fixes warp-speed scan on Windows with `sm_120`.
- For more information, see the CUDA Python 13.4 release notes and the CCCL Python 1.1 release notes.

2.3.6. CUDA Tile

Tile C++ Features

- Added `ct::strided_view` type for performing stencil operations.
- Added `ct::gather_scatter_view` type for sparse memory access patterns.
- For additional features and improvements, see the CUDA Tile C++ API Reference Release Notes.

Tile Python Features

- **CUDA Tile Python 1.5.0** adds the following:
  - `ct.assume_divisible_by()` to communicate scalar-divisibility and alignment assumptions to the compiler.
  - Compile-time specialization of selected array-shape dimensions through `ct.ArrayAnnotation(static_shape_dims=...)`.
  - Tuple-valued kernel arguments, including nested tuples and per-element constant annotations.
  - `single_run_timeout_sec` for `ct.tune.exhaustive_search()` so that hanging kernels do not stall an entire tuning run.
  - Improved autotuning performance by stopping slow configurations early.
  - Dynamic start and step arguments for `ct.arange()`.
  - Tuple comprehensions and tuple `in`/`not in` operations.
  - Limited support for dictionaries, `**kwargs`, and dictionary unpacking.
  - `enum.Enum` construction and comparison inside kernels.
  - Printing of dataclass instances, and support for frozen dataclasses as device-code globals.
  - Corrected conversion of strictly typed numeric constants, including integer wrapping and floating-point rounding or clamping.
  - Support for wrappers and decorators applied to user-defined functions.
  - Rejection of repeated axes in `load()`, `store()`, and `num_tiles()` ordering arguments.

- Calling-convention v2 for tuple arguments and static-shape annotations.

- **CUDA Tile Python 1.6.0** adds the following:
  - A `cutile-cache log` command for inspecting compilation history, including compiler versions, compilation time, kernel names, and TileIR 13.4 compiler remarks.
  - `check_bounds=False` for `ct.load()`, `ct.store()`, and tiled-view load/store methods to omit bounds checks when the programmer guarantees accesses are valid. Requires TileIR 13.4 or later.
  - Programmatic Dependent Launch support: `ct.grid_dependency_control_launch_dependents()`, `ct.grid_dependency_control_wait()`, and a `programmatic_dependent_launch` option for `ct.launch()`.
  - `float8_e5m3fnu` support for FP4 block-scaled MMA on Rubin `sm_107`.
  - Improved floating-point powers with integer exponents using `FPowI` on TileIR 13.4 or later.
  - IEEE rounding modes for float-to-float `astype()` conversions.
  - A `propagate_nan` option for min/max and argmin/argmax operations.
  - `ct.insert()` and `Tile.insert()` for inserting a subtile into a larger tile.
  - Static specialization of selected array-stride dimensions.
  - `ct.ensure_constant()` for asserting that a value is compile-time constant.
  - `ct.divmod()` and support for Python's built-in integer `divmod()`.
  - User-defined context-manager support.
  - User-defined dataclass methods, including `__post_init__`, `__call__`, indexing, and string representations.
  - `break` support in non-static loops.
  - Autotuning can now select a different kernel for each configuration.
  - `export_kernel()` can now export portable TileIR bytecode without GPU code.
  - Kernels are now compiled for the actual launch device rather than always targeting device 0.
  - Tuple parameters in the JAX integration.
  - Fixed runtime-loop scalar updates and CUDA crash-dump handling.

2.3.7. CUDA Tile IR

- **Supported Architectures**
  - (Developer Preview) Added support for the `sm_107` (Rubin) architecture.
- **New Operations**
  - Added op `cuda_tile.fpowi` for element-wise floating-point exponentiation with a signed-integer exponent.
  - Added op `cuda_tile.insert` to insert a source subtile into a destination tile at a given subtile index. The source shape must evenly divide the destination shape.
  - Added op `cuda_tile.memory_fence_alias_tko`, a token-ordered fence that orders operations accessing the same physical memory through different virtual aliases.
  - Added op `cuda_tile.gdc_launch_dependents_tko` for grid dependency control, signaling that programmatic-dependent-launch (PDL) dependent kernels may begin. It is a no-op below `sm_90`.
  - Added op `cuda_tile.gdc_wait_tko` for grid dependency control, waiting for predecessor-kernel completion with acquire semantics. It is a no-op below `sm_90` and pairs with `cuda_tile.gdc_launch_dependents_tko`.
- **New Types and Attributes**
  - (Developer Preview) Added the `fnv8E5M3FNU` (8-bit floating-point) type. It is an alternative floating-

point type intended for use as a block-scale type in `cuda_tile.mmaf_scaled` with `f4E2M1FN` (fp4) elements.

- **Modified Operations**
  - (Developer Preview) Modified op `cuda_tile.mmaf_scaled` to add support for `f4E2M1FN` inputs with `fnv8E5M3FNU` scale factors accumulating to `f32`.
  - Modified op `cuda_tile.loop` to support function return: a `cuda_tile.return` may appear inside a loop body to return from the enclosing function.
  - Modified ops `cuda_tile.load_view_tko` and `cuda_tile.store_view_tko` to add a new `inbounds` attribute so that programs can convey that accesses along specified dimensions are statically known to be in-bounds, allowing the compiler to skip bounds checking for those dimensions. Dimensions left unmarked default to conservative bounds checking, which matches the behavior of prior versions.
  - Modified op `cuda_tile.ftoi` to add an optional saturating modifier that clamps out-of-range values and converts NaN to 0.
  - Modified op `ftof` to support a richer rounding-mode matrix, including `nearest_away` for `f32` to `tf32`.
  - Renamed op `cuda_tile.pow` to `cuda_tile.fpowf`, complementing the new integer-exponent `cuda_tile.fpowi`.
- **Compiler Improvements**
  - Added opt-in compiler optimization remarks that identify selected load and store instructions and explain TMA instruction-selection failures. Use `tileiras --remarks=all` to enable all remarks; see the documentation for further details.
  - `tileiras` and the CUDA driver JIT compiler flow now re-use the Tile IR-level optimization pipelines from `cuda-tile-optimize`, applying canonicalization, common-subexpression elimination, and loop-invariant code motion according to the selected optimization level. See the documentation for further details.
- **Documentation Improvements**
  - Documented that the combiner function for `cuda_tile.reduce` and `cuda_tile.scan` must be commutative as well as associative. This was already required by the implementation, which may reorder the combine freely.
  - Added memory-alignment guidance for pointer loads, stores, atomic operations, and view accesses, including byte-alignment requirements for sub-byte element types.
  - Clarified `cuda_tile.partition_view` and `cuda_tile.strided_view` out-of-bounds semantics, distinguishing in-bounds view indices from partially out-of-bounds tiles and documenting load padding and store masking.
  - Clarified the `cuda_tile.tensor_view` memory contract for 4-bit elements, including dense packing, byte-alignment requirements, and little-endian nibble order.
  - Expanded `cuda_tile.gather_scatter_view` documentation with sparse-dimension indexing rules and multidimensional gather, scatter, padding, and out-of-bounds examples.
  - Expanded optimization-hint reference documentation with per-operation and architecture-specific constraints.
- **Fixed Issues**
  - Fixed an `sm_120` compiler crash when an `f16` constant was converted to an FP8 type with `cuda_tile.ftof` and the result was passed to `cuda_tile.print_tko`.
- **Known Issues**
  - A `cuda_tile.loop` or `cuda_tile.for` operation may fail compilation or produce incorrect results when a tile produced by a load is carried between iterations. The issue may occur when the next iteration's tile is loaded before the carried tile's final use. As a workaround, place the next load after the carried tile's final use, or restructure the loop so the loaded tile is not carried between iterations.

- Converting a `tf32` tile loaded from global memory to `f32` may produce incorrect values if the lower 13 mantissa bits of any loaded value are nonzero. These bits do not affect MMA operations on the loaded tile; the issue becomes observable only after conversion to `f32`. As a workaround, ensure that the lower 13 mantissa bits of every `tf32` value loaded from global memory are zero.

2.4. Resolved Issues

2.4.1. General CUDA

- Fixed CUDA Graphs containing more than approximately 2.3 million kernel nodes that could instantiate successfully but crash, hang, or report an illegal-address error when launched. *[6432056]*
- Fixed updating the dynamic shared-memory size of a cooperative kernel node in an instantiated graph, which could cause a hang or illegal-memory access. *[6090062]*
- Fixed a use-after-free in `cuGraphNodeSetParams` when used with `CU_GRAPH_CHILD_GRAPH_OWNERSHIP_MOVE`. *[6288631]*
- Fixed Xid 32 errors and application crashes when `CUDA_SCALE_LAUNCH_QUEUES` was used with CUDA Graphs containing long chains of kernel nodes. *[5686696]*
- Fixed an HMM/UVM GPU-unregistration issue that could cause a kernel panic or system restart during CUDA or DCGM workloads on Red Hat Enterprise Linux. *[6185077]*
- Fixed a Linux driver deadlock that could leave the GPU unusable in rare situations. *[6095595]*
- Fixed an issue that could cause a GPU to disappear after an in-progress GPU reset was interrupted in a virtual machine. *[5967535]*
- Fixed `libnvidia-ml.a` deadlocking when `nvmlInit_v2()` was invoked from a shared library. *[6174166]*
- Fixed NVML and `nvidia-smi` on Windows Arm64 exposing unsupported TCC-mode controls that could leave the system unstable. *[6225285]*

2.4.2. CUDA Compiler

- Fixed `ptxas` scheduling that could move a predicated `REDUX.SUM` operation above its predicate definition, producing incorrect results on Blackwell GPUs. *[6260886]*
- Fixed a CUDA C++ compiler issue that could produce incorrect results for floating-point comparisons inside a function. *[6235207]*
- Fixed `ptxas` incorrectly dropping `createpolicy.fractional.L2::evict_first`, which could leave a dependent `cp.async` cache-hint descriptor uninitialized and cause an illegal-instruction error on `sm_90`. *[6132797]*
- Fixed incorrect SASS generation for `tensormap.cp_fenceproxy` with `.sys` scope. *[5891184]*
- Fixed missing ordering between `UTMALDG` and `UTMACMDFLUSH` operations that could cause silent data corruption. *[5996051]*
- Fixed an optimization regression introduced in CUDA 13.1 that could cause illegal memory accesses in cuSOLVERDx HTEV and BDSVD operations on `sm_90`. *[5914878]*
- Fixed an `nvcc` 13.3 regression that prevented applications using Abseil `flat_hash_map`, including some ONNX configurations, from compiling. *[6302392]*
- Fixed CUDA C++ frontend handling of hidden friend functions with constrained auto parameters. *[6047481]*
- Fixed a template-substitution issue whose result could change when an otherwise-unused template declaration was present. *[5818596]*
- Fixed an internal compiler assertion triggered by valid CUDA C++ source. *[5933457]*

- Fixed multiple issues found through fuzz testing. *[6194108]*
  - Fixed an issue where the integer constant `0x7fffffff` (`INT_MAX`) reaching a negation or subtraction could produce an incorrect result.
  - Fixed an issue where a `prmt.b32` using a sign-replicating selector, followed by a byte or halfword mask (`and.b32` or `cvt.u32.u16`), could incorrectly produce zero.
  - Fixed cases where extended-precision carry chains (`add.cc/addc`, `sub.cc/subc`) whose inputs were produced by shifts or multiplies could consume an incorrect carry or borrow, giving a result off by one.
  - Fixed an issue where adding a value back after a saturating subtraction of that same value (`sub.sat.s32`) could be simplified in a way that ignored saturation.
  - Fixed an issue where the bits of a value produced by a multiply or an arithmetic right shift could be incorrectly inferred, causing a subsequent mask or unsigned comparison to fold to an incorrect constant.
  - Fixed cases where a bit-field insert – either `bfi.b32` with pos + len > 32, or one formed by the compiler from a masked shift – could write the field at the wrong bit position or leave part of the field unwritten.
  - Fixed an issue where a shift or funnel shift (`shf.l.wrap.b32`, `shf.r.wrap.b32`, `shr`) whose source was negated or complemented (for example by `neg` or `cnot`) and whose result immediately fed an add could produce an incorrect result.
  - Fixed an issue where using a complemented or negated value inside a branch that pins that value to a known constant could drop the complement or negation.
  - Fixed an issue where two comparisons of the same operands differing only in signedness could be treated as equivalent, causing one of the branches to be dropped.
  - Fixed an issue where a min/max clamp pair against zero that mixed signed and unsigned forms could be incorrectly folded to zero.
  - Fixed cases where an unsigned comparison against zero of an all-ones boolean (as produced by `set`) or of a negated `abs` value could be folded to the wrong result, taking the wrong branch or `selp` arm.
  - Fixed an issue where `slct` with a large constant selector could select the wrong operand.
  - Fixed an issue where a `lop3.b32` that reads the same register in more than one input, combined with an adjacent bitwise operation, could produce an incorrect result.
  - Fixed an issue where an unrolled loop containing repeated high multiplies (`mul.hi`, `mad.hi.cc`) could have some of those computations incorrectly eliminated.
  - Fixed an issue where packed 16-bit operations (`add.u16x2`, `min.u16x2`, `max.u16x2`) with an operand produced by a 16-bit load or by a scalar value could compute an incorrect high lane.
  - Fixed an issue where a 16-bit min/max followed by a conversion and an equality test against the second operand could produce the opposite result when the two operands are equal.
  - Fixed an issue where a conditional assignment could be incorrectly removed when the register's prior value had the same logical truth value but a different bit pattern.
  - Fixed an internal compiler error when compiling a warp-uniform `mul.hi` for Blackwell targets.
  - Fixed an internal compiler error when a `cvt.pack.sat` instruction's merge operand became a compile-time constant.
  - Fixed an internal compiler error on an `addc/madc` whose carry-in was provably zero.
  - Fixed an internal compiler error when a kernel reads a `.param` variable with `ld.param` that is never used as a call argument or return value. `ptxas` now reports an error for this construct, which is not defined by the PTX ISA.

2.4.3. CUDA Tools

- Fixed malformed `graphNodeId` values in CUDA Graph memcpy activity records when CUPTI hardware-event sampling was enabled. *[6395204]*

- Fixed CUPTI runtime- and driver-API callback filtering not taking effect when user-defined records were enabled. *[6342060]*
- Fixed host heap corruption when CUPTI hardware tracing and activity collection were enabled together on GB200. *[6274883]*
- Fixed a Compute Sanitizer false positive for `tensormap.cp_fenceproxy`. *[6090556]*
- Fixed a performance regression that made `ncu --import` substantially slower in Nsight Compute 2026.3.0.5. *[6412841]*
- Fixed Nsight Compute failing after profiling with an `unknown encoding: utf-8-sig` error. *[5872516]*
- Fixed a CUDA-GDB regression that prevented standard C++ pretty-printers from working. *[5503986]*

2.5. Known Issues

2.5.1. CUDA Platform

- Applications statically built with, or dynamically linking to, an older CUDA Runtime with the R615 driver may report benign public error messages indicating that IMEX channels are incorrectly set up. These can be ignored if fabric handle support is not required.
- Device attribute queries for `CU_DEVICE_ATTRIBUTE_HANDLE_TYPE_FABRIC_SUPPORTED`, `CU_DEVICE_ATTRIBUTE_MEMPOOL_SUPPORTED_HANDLE_TYPES`, `CU_DEVICE_ATTRIBUTE_HOST_NUMA_MULTINODE_IPC_SUPPORTED`, and `CU_DEVICE_ATTRIBUTE_LOGICAL_ENDPOINT_SUPPORTED_HANDLE_TYPES` may report that fabric handles are unsupported if IMEX channels are incorrectly configured. If fabric usage reports as unsupported, check the public CUDA error logs to verify whether IMEX setup was the cause.

2.5.2. CUDA Compiler

- There is an open issue in the PTX compiler for the `tcgen05.alloc` instruction when used with `-g` or `-g-tmem-access-check`. This manifests when `tcgen05.alloc` is used with the `.exclusive` qualifier and `nCols` is explicitly set to 576. Mitigation: use a value smaller than 576 for `nCols` with `.exclusive`, or avoid using `-g` or `-g-tmem-access-check` with `tcgen05.alloc`.

2.6. Deprecated or Dropped Features

2.6.1. CUDA Platform

- Legacy Nsight Eclipse Edition plugins are no longer delivered in CUDA Toolkit packages beginning with CUDA 13.3.
- Python 3.10 support is deprecated across the CUDA Python 13.4 packages.

2.6.2. Architectures

- CUDA 14.0 will move to Armv8.2-A as the minimum supported architecture for ARM64-SBSA.

2.6.3. Operating Systems

- None

2.6.4. CUDA Toolchains

- None

Chapter 3

CUDA Libraries

This section covers CUDA Libraries release notes for 13.x releases.

Note:
Documentation will be updated to accurately reflect supported C++ standard libraries for CUDA Math Libraries.

3.1. cuBLAS Library

3.1.1. cuBLAS: Release 13.4

- **New Features**
  - Emulated FP64 matrix multiplications:
    - Emulated FP64 matrix multiplications now leverage the Ozaki-II scheme when it provides a performance benefit over the Ozaki-I scheme; the Ozaki-II scheme is supported on NVIDIA Ampere and newer GPUs.
    - On B200 and RTX PRO 6000 Blackwell Server Edition GPUs, this enables up to 175 and 45 TFLOPS of emulated DGEMM performance, respectively, and up to 295 and 70 TFLOPS of emulated ZGEMM performance, respectively, by leveraging the 2M algorithm.
    - Emulated FP64 matrix multiplications now support Rubin GPUs (compute capability 10.7), leveraging the Ozaki-I and Ozaki-II schemes and the new TI16 type via the `tcgen05.mma` instruction to achieve up to 212 TFLOPS of emulated DGEMM performance. Emulated ZGEMM can reach up to 301 TFLOPS, with further improvements to come.
  - cuBLASLt adds experimental support for an alternative scaling-factor layout for MXFP8 matmuls through the `CUBLASLT_MATMUL_MATRIX_SCALE_VEC32_MN_K4_UE8M0` and `CUBLASLT_MATMUL_MATRIX_SCALE_VEC128_MN_K4_UE8M0` scaling modes. For more information, see the cuBLAS documentation.
  - Added support for the NVIDIA Rubin (compute capability 10.7) GPU architecture.
  - cuBLASLt Grouped GEMM performance is improved on Blackwell data center GPUs through dynamic scheduling of matrix computations. Improvements of up to 20% can be seen for Grouped GEMM calls with a large number of groups (for example, 32). *[5913842] [5992105] [CUB-9925]*
- **Known Issues**
  - When running NVIDIA Compute Sanitizer (versions up to CUDA Toolkit 13.4) with cuBLAS-linked workloads, the tool may report false-positive errors regarding invalid global reads or out-of-bounds (OOB) memory accesses inside cuBLAS accelerated kernels. This is a known issue caused by a limitation in Compute Sanitizer's ability to accurately identify the PTX semantics of the asynchronous global-to-shared memory copy instruction `cp.async.cg.shared.global [dst], [src], cp-size, src-size` when `src-size < cp-size`. In this specific scenario, where the number of bytes to copy exceeds the source buffer size, the hardware guarantees that the remaining bytes (`cp-size - src-size`) in the destination shared memory are automatically padded with zeros, ensuring that the

memory access is neither invalid nor OOB. Hence, Compute Sanitizer reports for these specific cases can be safely ignored. *[6196059]*

- cuBLASLt Grouped GEMM with per-batch tensor-wide scales causes an invalid memory access for groups where `m > 0`, `n > 0`, and `k = 0`. As a workaround, always pass valid scale pointers for each group. This issue was introduced in CUDA Toolkit 13.1 (cuBLAS 13.2.0). *[CUB-10481]*
- `cublasLtMatmul()` can return incorrect output when using split-K if the number of K splits (`SPLITK_NUM`) does not evenly divide the K dimension and the floor division of K / `SPLITK_NUM` is an exact multiple of the stage size. This affects only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 9.0, 10.x, and 11.x. As a workaround, use `cublasLtMatmulAlgoConfigGetAttribute()` to query the number of K splits (`CUBLASLT_ALGO_CONFIG_SPLITK_NUM`) and the stage size (derived from `CUBLASLT_ALGO_CONFIG_STAGES_ID`), then use `cublasLtMatmulAlgoConfigSetAttribute()` to set a value that evenly divides the K dimension, or for which K / `SPLITK_NUM` is not an exact multiple of the stage size. This issue was first introduced in CUDA Toolkit 12.6 Update 2 (cuBLAS 12.6.3). *[6580689]*
- cuBLASLt Grouped GEMM operations with `CUBLAS_POINTER_MODE_HOST` can lead to incorrect results on Hopper GPUs when the number of waves is larger than 2. As a workaround, pass device alpha and beta using `CUBLAS_POINTER_MODE_DEVICE`. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6681084]*
- cuBLASLt Grouped GEMM operations can lead to incorrect results on B300 and Rubin GPUs when the input matrices use the NVFP4 data type, the number of waves is larger than 3, and the algorithm attribute `CUBLASLT_ALGO_CONFIG_STAGES_ID` is `CUBLASLT_MATMUL_STAGES_768xAUTO`. As a workaround, skip such candidates when using `cublasLtMatmulAlgoGetHeuristic()`. *[6681084]*
- `cublasLtMatmul()` can produce incorrect results on B300 and Rubin GPUs when the input matrices use the NVFP4 data type. This affects only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 and the algorithm attribute `CUBLASLT_ALGO_CONFIG_STAGES_ID` is `CUBLASLT_MATMUL_STAGES_768xAUTO`. *[CUB-10573]*

- **Resolved Issues**
  - Between CUDA Toolkit 13.3 Update 1 and 13.4, cuBLAS released an independent patch release (cuBLAS 13.6.1) that resolves several issues. Refer to the associated cuBLAS patch release notes for details.
  - Fixed an issue where `cublasLtMatmulAlgoGetHeuristic()` could return no algorithms, and `cublasLtMatmul()` could return `CUBLAS_STATUS_NOT_SUPPORTED`, for some NVFP4 matmuls with NVFP4 output and the `CUBLASLT_EPILOGUE_BIAS` epilogue on GPUs with compute capability 12.x. *[CUB-10222]*
  - Fixed an issue where `cublasLtMatmulAlgoGetHeuristic()` returned `CUBLAS_STATUS_INTERNAL_ERROR` and invalidated an in-flight CUDA graph capture when called while a non-default blocking stream was being captured. This issue was introduced in CUDA Toolkit 13.3 (cuBLAS 13.5.1). *[6288786]*
  - Fixed an issue where `cublas<t>gemv()` with `trans` equal to `CUBLAS_OP_T` or `CUBLAS_OP_C` could perform an illegal memory access when the number of elements addressed by the output vector (`n * incy`) exceeded the 32-bit index range. `cublasZgemv()` could additionally return `CUBLAS_STATUS_NOT_SUPPORTED` when `n * lda` exceeded that range. *[6207926]*
  - Fixed an issue where `cublasZgemv()` with `trans` equal to `CUBLAS_OP_C` and `incy` greater than 1 produced incorrect results when FP64 fixed-point emulation was enabled (`CUBLAS_FP64_EMULATED_FIXEDPOINT_MATH` with `CUBLAS_EMULATION_STRATEGY_EAGER`). *[6207926]*

3.1.2. cuBLAS: Release 13.3 Update 1

- **New Features**
  - The TMA-based kernel (Hopper and newer) now accelerates DSYMV in addition to the already-

enabled SSYMV. The 16-byte alignment requirement for the `A` pointer was dropped for this kernel, and support for atomics was added through `cublasSetAtomicsMode()`. The geomean speedup across architectures and datatypes is 1.3x, and up to 5.9x.

- **Known Issues**
  - Non-default epilogues are unintentionally allowed for `cublasLtMatmul()` with int8 inputs using regular data ordering and scale type `CUDA_R_32F`. This is an undocumented and lightly tested feature that users are discouraged from using, and it is planned for removal in the next major release. *[CUB-10067]*
  - In cuBLASLt, the heuristics for the Grouped GEMM API return sub-optimal algorithms when the `C` and `D` matrices use `CUBLASLT_ORDER_ROW` ordering. As a workaround, swap `CUBLASLT_MATMUL_PREF_GROUPED_DESC_D_AVERAGE_ROWS` and `CUBLASLT_MATMUL_PREF_GROUPED_DESC_D_AVERAGE_COLS` in the preferences before calling `cublasLtMatmulAlgoGetHeuristic()`. *[6335555]*
  - Multiple cuBLASLt Grouped GEMM operations reusing the same workspace on Hopper GPUs may lead to hangs or unspecified launch failures. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6456362]*
  - Calling `cublasLtMatmulAlgoGetHeuristic()` while a non-default blocking stream is being captured returns `CUBLAS_STATUS_INTERNAL_ERROR`. This issue was introduced in CUDA Toolkit 13.3 (cuBLAS 13.5.1). *[6288786]*
  - GEMV-like operations (e.g., `cublas<t>gemv()`, or `cublasLtMatmul()` with M or N equal to 1) may return `CUBLAS_STATUS_NOT_SUPPORTED` for certain shapes and workspace configurations. This issue was introduced in CUDA Toolkit 13.3 (cuBLAS 13.5.1).
  - Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*
  - GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*
  - GEMV-like operations (e.g., `cublas<t>gemv()`, or `cublasLtMatmul()` with M or N equal to 1) may produce incorrect results on GPUs with compute capability 12.0 or 12.1 when the matrix is transposed, its non-accumulation dimension is greater than 3145680, and beta is not equal to 0. This issue was introduced in CUDA Toolkit 13.3 Update 1 (cuBLAS 13.6.0). *[CUB-10445]*
  - cuBLASLt Grouped GEMM operations with `CUBLAS_POINTER_MODE_HOST` can lead to incorrect results on Hopper GPUs when the number of waves is larger than 2. As a workaround, pass device alpha and beta using `CUBLAS_POINTER_MODE_DEVICE`. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6681084]*
- **Resolved Issues**
  - Fixed an issue where `cublasXt<t>spmm()` could produce incorrect results with `m` greater than 46340. *[6155165]*
  - Fixed an issue where `cublasLtMatmul()` could run an unsupported combination of data types: an FP32-like compute type with FP32 C and D and non-FP32 A and B, in which case A and B are incorrectly interpreted as FP32 matrices. *[CUB-9942]*
  - Fixed an issue where `cublasLtMatmul()` returned `CUBLAS_STATUS_NOT_SUPPORTED` for FP8 Grouped GEMM problems with scale modes `CUBLASLT_MATMUL_MATRIX_SCALE_VEC128_32F` and `CUBLASLT_MATMUL_MATRIX_SCALE_BLK128x128_32F` on Hopper GPUs. *[CUB-10031]*
  - Fixed an issue where `cublasLtMatmul()` with int8 inputs and scale type `CUDA_R_32I` would allow

non-default epilogues on Blackwell GPUs with compute capability 10.x and return incorrect results. The correct behavior is to disallow all but the default epilogue, as documented. *[CUB-10066]*

3.1.3. cuBLAS: Release 13.3

- **New Features**
  - Enabled memory-parsimonious tiling for FP64 emulated matrix multiplications. This improvement ensures that the workspace memory budget no longer exceeds 8 GB.
  - Added support for CUDA Green contexts.
  - Improved FP4 matrix multiplication performance on Blackwell Ultra GPUs by a geometric mean of 5% across a wide range of problems, with up to 7% speedup for some small problems.
  - Improved TF32 matrix multiplication performance on Blackwell and Blackwell Ultra GPUs by a geometric mean of 27% across a wide range of problems and layouts, with up to 3.5x speedup for some small problems.
  - Improved TF32 TN matrix multiplication performance on Hopper GPUs by a geometric mean of 11% across a wide range of problems, with up to 40% speedup for some small problems.
  - Improved SYMV performance with TMA-based acceleration for Hopper, Blackwell, and Blackwell Ultra kernels with up to 27% geomean speedup.
- **Known Issues**
  - Multiple cuBLASLt Grouped GEMM operations reusing the same workspace on Hopper GPUs may lead to hangs or unspecified launch failures. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6456362]*
  - cuBLASLt Grouped GEMM operations with `CUBLAS_POINTER_MODE_HOST` can lead to incorrect results on Hopper GPUs when the number of waves is larger than 2. As a workaround, pass device alpha and beta using `CUBLAS_POINTER_MODE_DEVICE`. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6681084]*
  - Calling `cublasLtMatmulAlgoGetHeuristic()` while a non-default blocking stream is being captured returns `CUBLAS_STATUS_INTERNAL_ERROR`. This issue was introduced in CUDA Toolkit 13.3 (cuBLAS 13.5.1). *[6288786]*
  - GEMV-like operations (e.g., `cublas<t>gemv()`, or `cublasLtMatmul()` with M or N equal to 1) may return `CUBLAS_STATUS_NOT_SUPPORTED` for certain shapes and workspace configurations. This issue was introduced in CUDA Toolkit 13.3 (cuBLAS 13.5.1).
  - Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*
  - GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*

3.1.4. cuBLAS: Release 13.2 Update 2

- **Known Issues**
  - Multiple cuBLASLt Grouped GEMM operations reusing the same workspace on Hopper GPUs may lead to hangs or unspecified launch failures. This issue was introduced in CUDA Toolkit 13.2 Update

1 (cuBLAS 13.4.0). *[6456362]*

- Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*
- GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*

- **Resolved Issues**
  - Fixed an issue where `cublasLtMatmul()` ignored the tensor-wide scaling value specified by `CUBLASLT_MATMUL_DESC_D_SCALE_POINTER` for NVFP4 matrix multiplications with NVFP4 output, resulting in incorrect results. This affected algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 10.x and 11.x. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6059292]*

3.1.5. cuBLAS: Release 13.2 Update 1

Note:
CUDA Toolkit 13.2 Update 1 contains a critical cuBLAS bug for an issue where `cublasLtMatmul()` could ignore tensor-wide scaling for NVFP4 matrix multiplications, resulting in incorrect results. Please see the cuBLAS patch release notes for an available cuBLAS patch (13.4.1) to resolve this issue.

- **New Features**
  - Extended the experimental Grouped GEMM API in cuBLASLt to support NVFP4 inputs and bias epilogues on Blackwell GPUs with Compute Capability 10.x and 11.0. Grouped GEMM NVFP4 support currently uses only MMA tile sizes with K equal to 64.
  - Extended the experimental Grouped GEMM API in cuBLASLt to support BF16, FP16, and FP8 input data types with BF16, FP16, and FP32 output data types on Hopper GPUs. For FP8 inputs, tensorwide scaling and block scaling (`VEC128` and `BLK128x128`) are supported.
  - Improved Grouped GEMM performance on Blackwell GPUs, providing up to 20% higher performance for large problem sizes where the matrices exceed the L2 cache size.
- **Known Issues**
  - `cublasLtMatmul()` ignores the tensor-wide scaling value provided by `CUBLASLT_MATMUL_DESC_D_SCALE_POINTER` for NVFP4 matrix multiplications with NVFP4 output, leading to incorrect results. This affects algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 10.x and 11.x. This issue was introduced in CUDA Toolkit 13.2 Update 1. *[6059292]*
  - Multiple cuBLASLt Grouped GEMM operations reusing the same workspace on Hopper GPUs may lead to hangs or unspecified launch failures. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6456362]*
  - cuBLASLt Grouped GEMM operations with `CUBLAS_POINTER_MODE_HOST` can lead to incorrect results on Hopper GPUs when the number of waves is larger than 2. As a workaround, pass device alpha and beta using `CUBLAS_POINTER_MODE_DEVICE`. This issue was introduced in CUDA Toolkit 13.2 Update 1 (cuBLAS 13.4.0). *[6681084]*
  - Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory

reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*

- GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*

- **Resolved Issues**
  - Fixed an issue in `cublasLtMatmulAlgoGetHeuristic()` that could result in no algorithm candidates being returned for Grouped GEMM on Blackwell GPUs. *[CUB-9657]*

3.1.6. cuBLAS: Release 13.2

- **New Features**
  - Extended the experimental Grouped GEMM API in cuBLASLt to support MXFP8 inputs on GPUs with Compute Capability 10.x and 11.0.
  - Added control over special-case handling in FP32 emulation via the environment variable `CUBLAS_EMULATION_SPECIAL_VALUES_SUPPORT_MASK`. Setting `CUBLAS_EMULATION_SPECIAL_VALUES_SUPPORT_MASK=0` can improve performance for applications that do not require preservation of infinity and NaN values, without requiring code changes. For more information, see the `cudaEmulationSpecialValuesSupport_t` documentation.
  - Added FP64 fixed-point emulation support to the `cublas[D|Z]syrk`, `cublas[D|Z]syr2k`, `cublasZherk`, and `cublasZher2k` routines. When the math mode is set to `CUBLAS_FP64_EMULATED_FIXEDPOINT_MATH`, cuBLAS will automatically use FP64 emulation for sufficiently large SYRK and HERK problems. Current support is limited to GPUs with Compute Capability 10.0.
  - Improved performance on RTX PRO 6000 GPUs, delivering up to 20% speedup for FP8, FP16/BF16, TF32, and INT8 precisions.
  - Improved GEMM performance on DGX Spark systems for MXFP8 and NVFP4 data types in large M and N problem sizes, with up to 3× performance improvement for selected matrix shapes.
- **Known Issues**
  - On Blackwell GPUs, FP64 fixed-point emulation kernels may produce incorrect results or experience data corruption when executed concurrently with third-party kernels that allocate tensor memory. *[CUB-9633]*
  - Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*
  - GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*
- **Resolved Issues**
  - Fixed an issue in `cublasLtMatmul` that could lead to incorrect results when it ran concurrently with

another kernel that uses Tensor Memory. This issue only affected algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with Compute Capability 10.x and 11.x, and existed since cuBLAS 12.8. *[5807900]*

- Fixed an issue in `cublasLtMatmul` that could lead to incorrect results or invalid memory access errors for large leading dimensions, specifically when the product of the data type size and the leading dimension of a matrix exceeded the bounds of a signed 32 bit integer. This issue affected GPUs with Compute Capability 9.0, 10.x, or 11.0, existed since cuBLAS 12.6 Update 2, and only affected algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66. *[CUB-9572]*
- Fixed an issue in the cuBLASLt Matmul API that could cause FP8 kernels to hang on GPUs with Compute Capability 9.0 when `beta != 0` and `scale_C = 0`. This issue only affected algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66. *[CUB-9627]*
- Fixed an issue in the cuBLASLt Grouped GEMM API that ignored groups with `k = 0`, leading to incorrect results. This issue existed since CUDA 13.1. *[CUB-9529]*
- Fixed an issue in the cuBLASLt Matmul API that could cause incorrect results when C broadcasting was used (`LDC = 0`). *[5845724]*
- Added missing checks for matrix pointer alignment in the `cublasLtMatmul` API. *[CUB-9577] [CUB-9599] [CUB-9585]*
- Fixed an issue in `cublasLtMatmul` that could lead to incorrect results for NVFP4 precision on B300 and GB300 GPUs when the `m` dimension was not a multiple of 64. *[CUB-9577]*
- Fixed an issue in `cublasLtMatmul` that could lead to incorrect results for NVFP4 precision on future GPUs, impacting future hardware compatibility. *[CUB-9570]*
- Fixed an issue in GEMM and Matmul APIs with BF16 and FP16 inputs on DGX Spark and FP8 inputs on GeForce that could potentially cause illegal memory accesses. *[5846563]*
- Fixed an issue in cuBLASLt to enable `CUBLASLT_EPILOGUE_BGRADA` and `CUBLASLT_EPILOGUE_BGRADB` epilogues when the C matrix `CUBLASLT_MATRIX_LAYOUT_ORDER` was set to `CUBLASLT_ORDER_ROW`. *[4617436]*
- Fixed an integer overflow bug in complex, emulated FP64 matrix multiplication. The affected routines include `cublasZgemm`, `cublasZtrsm`, `cublasGemmEx`, and `cublasLtMatmul`. The overflow occurred when `2*m*n + m` exceeded `UINT_MAX`, where `m` is the number of rows of `op(A)` and C, and `n` is the number of columns of `op(B)` and C. *[5720478]*
- Improved GB200 and B200 performance for MXFP8 and NVFP4 precisions when `M` and `N` were less than or equal to 32. *[CUB-9646]*

3.1.7. cuBLAS: Release 13.1 Update 1

- **Known Issues**
  - The cuBLASLt Grouped GEMM API ignores groups with `k = 0`, which can lead to incorrect results. As a workaround, initialize output matrices `D` with `beta*C` for all groups, and then compute Grouped GEMM as `D += A*B` so the result for groups with `k = 0` is computed properly. This issue applies to the experimental cuBLASLt Grouped GEMM API introduced in CUDA 13.1. *[CUB-9529]*
  - Complex FP64 GEMM routines using fixed-point emulation can produce incorrect results when matrix dimensions are large enough that `m*n > 2^31` due to integer overflow in an address calculation. *[5720478]*
  - `cublasLtMatmul()` may produce incorrect results when run concurrently with another kernel that uses Tensor Memory. This issue affects only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 10.x and 11.x, and has existed since cuBLAS 12.8. *[5807900]*
  - Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected,

but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*

- cuBLAS GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*

- **Resolved Issues**
  - Fixed an issue where fixed point emulation with 7 mantissa bits or less could trigger unspecified launch failures. *[5692684]*
  - Fixed an issue where `cublasLtMatmul` with FP8 arguments and `CUBLASLT_MATMUL_MATRIX_SCALE_SCALAR_32F` scaling mode (default) incorrectly required scaling factor addresses to be 16-byte aligned. This issue existed since cuBLAS 12.9. *[5728938]*

3.1.8. cuBLAS: Release 13.1

- **New Features**
  - Introduced experimental support for grouped GEMM in cuBLASLt. Users can create a matrix with grouped layout using `cublasLtGroupedMatrixLayoutCreate` or `cublasLtGroupedMatrixLayoutInit`, where matrix shapes are passed as device arrays. `cublasLtMatmul` now accepts matrices with grouped layout, in which case matrices are passed as a device array of pointers, where each pointer is a separate matrix that represents a group with its own shapes. Initial support covers A/B types FP8 (E4M3/E5M2), FP16, and BF16, with C/D types FP16, BF16, and FP32; column-major only, default epilogue, 16-byte alignment; requires GPUs with compute capability 10.x or 11.0.
    In addition, the following experimental features were added as part of grouped GEMM:
    - Per-batch tensor-wide scaling for FP8 inputs, enabled by the new `cublasLtMatmulDescAttributes_t` entry `CUBLASLT_MATMUL_MATRIX_SCALE_PER_BATCH_SCALAR_32F`.
    - Per-batch device-side alpha and beta, enabled by the new `cublasLtMatmulDescAttributes_t` entries `CUBLASLT_MATMUL_DESC_ALPHA_BATCH_STRIDE` and `CUBLASLT_MATMUL_DESC_BETA_BATCH_STRIDE`.
  - Improved performance on NVIDIA DGX Spark for CFP32 GEMMs. *[5514146]*
  - Added `sm_121` DriveOS support.
  - Improved performance on Blackwell (`sm_100` and `sm_103`) via heuristics tuning for FP32 GEMMs whose shapes satisfy `M, N >> K`. *[CUB-8572]*
  - Improved performance of FP16, FP32, and CFP32 GEMMs on Blackwell Thor.

- **Resolved Issues**
  - Fixed missing memory initialization in `cublasCreate()` that could result in emulation environment variables being ignored. *[CUB-9302]*
  - Removed unnecessary overhead related to loading kernels on GPUs with compute capability 10.3. *[5547886]*
  - Fixed FP8 matmuls potentially failing to launch on multi-device Blackwell GeForce systems. *[CUB-9487]*
  - Added stricter checks for in-place matmul to prevent invalid use cases (`C == D` is allowed if and only if `Cdesc == Ddesc`). As a side effect, users are no longer able to use `D` as a dummy pointer for `C` when using `CUBLASLT_POINTER_MODE_DEVICE` with `beta = 0`. However, a distinct dummy pointer may still be passed. The stricter checking was added in CUDA Toolkit 13.0 Update 2. *[5471880]*
  - Fixed `cublasLtMatmul` with `INT8` inputs, `INT32` accumulation, and `INT32` outputs potentially returning

CUBLAS_STATUS_NOT_SUPPORTED when dimension N is larger than 65,536 or when batch count is larger than 1. *[5541380]*

- Added validation for batched matmul to reject invalid configurations where the batch counts differ (Adesc batch count != Bdesc batch count). *[5645772]*

- **Known Issues**
  - The Grouped GEMM cuBLASLt API ignores groups with k = 0, which can lead to incorrect results. As a workaround, initialize each output matrix D with beta * C for all groups before the call, then compute Grouped GEMM as D += A * B so that the result for groups with k = 0 is preserved. This issue applies to the experimental Grouped GEMM cuBLASLt API released in CUDA 13.1. *[CUB-9529]*
  - cublasLtMatmul() may produce incorrect results when run concurrently with another kernel that uses Tensor Memory. This issue affects only algorithms with CUBLASLT_ALGO_CONFIG_ID equal to 66 on GPUs with compute capability 10.x and 11.x, and has existed since cuBLAS 12.8. *[5807900]*
  - Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*
  - cuBLAS GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling cudaGridDependencySynchronize(). This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling cudaTriggerProgrammaticLaunchCompletion(). This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*

3.1.9. cuBLAS: Release 13.0 Update 2

- **New Features**
  - Enabled opt-in fixed-point emulation for FP64 matmuls (D/ZGEMM) which improves performance and power-efficiency. The implementation follows the Ozaki-1 Scheme and leverages an automatic dynamic precision framework to ensure FP64-level accuracy. See here for more details on fixed-point emulation along with the table of supported compute-capabilities and the CUDA library samples for example usages.
  - Improved performance on NVIDIA DGX Spark for FP16/BF16 and FP8 GEMMs.
  - Added support for BF16x9 FP32 emulation to cublas[SC]syr[2]k and cublasCher[2]k routines. With the math mode set to CUBLAS_FP32_EMULATED_BF16X9_MATH, for large enough problems, cuBLAS will automatically dispatch SYRK and HERK to BF16x9-accelerated algorithms.

- **Resolved Issues**
  - Fixed undefined behavior caused by dereferencing a nullptr when passing an uninitialized matrix layout descriptor for Cdesc in cublasLtMatmul. *[CUB-8911]*
  - Improved performance of cublas[SCDZ]syr[2]k and cublas[CZ]her[2]k on Hopper GPUs when dimension N is large. *[CUB-8293] [5384826]*

- **Known Issues**
  - cublasLtMatmul with INT8 inputs, INT32 accumulation, and INT32 outputs might return CUBLAS_STATUS_NOT_SUPPORTED when dimension N is larger than 65,536 or when the batch count is larger than 1. The issue has existed since CUDA Toolkit 13.0 Update 1 and will be fixed in a later release. *[5541380]*
  - FP8 matmuls may fail to launch on multi-device Blackwell GeForce systems. As a workaround, run a separate process per device. *[CUB-9487]*

- `cublasLtMatmul()` may produce incorrect results when run concurrently with another kernel that uses Tensor Memory. This issue affects only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 10.x and 11.x, and has existed since cuBLAS 12.8. *[5807900]*
- Strided batched GEMM operations using broadcast operands may perform out-of-bounds memory reads on GPUs with compute capability 12.0 or 12.1 when an input matrix shared across batches (via zero or overlapping strides) is smaller than a few hundred KB. Numerical accuracy is unaffected, but these invalid accesses can trigger compute-sanitizer warnings or, in rare instances, illegal memory access errors. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[6040940] [5996751]*
- cuBLAS GEMM kernels on GPUs with compute capability 10.x or 12.x may access device alpha and beta pointers before calling `cudaGridDependencySynchronize()`. This can result in a WAR hazard if the preceding PDL kernel produces alpha and beta values on device after calling `cudaTriggerProgrammaticLaunchCompletion()`. This issue was introduced in CUDA Toolkit 13.0 Update 2 (cuBLAS 13.1.0). *[CUB-10409]*

3.1.10. cuBLAS: Release 13.0 Update 1

- **New Features**
  - Improved performance:
    - Block-scaled FP4 GEMMs on NVIDIA Blackwell and Blackwell Ultra GPUs
    - `SYMV` on NVIDIA Blackwell GPUs *[5171345]*
    - `cublasLtMatmul` for small cases when run concurrently with other CUDA kernels *[5238629]*
    - TF32 GEMMs on Thor GPUs *[5313616]*
    - Programmatic Dependent Launch (PDL) is now supported in some cuBLAS kernels for architectures `sm_90` and above, decreasing kernel launch latencies when executed alongside other PDL kernels.
- **Resolved Issues**
  - Fixed an issue where some `cublasSsyrkx` kernels produced incorrect results when `beta = 0` on NVIDIA Blackwell GPUs. *[CUB-8846]*
  - Resolved issues in `cublasLtMatmul` with INT8 inputs, INT32 accumulation, and INT32 outputs where:
    - `cublasLtMatmul` could have produced incorrect results when A and B matrices used regular ordering (`CUBLASLT_ORDER_COL` or `CUBLASLT_ORDER_ROW`). *[CUB-8874]*
    - `cublasLtMatmul` could have been run with unsupported configurations of `alpha/beta`, which must be 0 or 1. *[CUB-8873]*
- **Known Issues**
  - `cublasLtMatmul()` may produce incorrect results when run concurrently with another kernel that uses Tensor Memory. This issue affects only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 10.x and 11.x, and has existed since cuBLAS 12.8. *[5807900]*

3.1.11. cuBLAS: Release 13.0

- **New Features**
  - The `cublasGemmEx`, `cublasGemmBatchedEx`, and `cublasGemmStridedBatchedEx` functions now accept `CUBLAS_GEMM_AUTOTUNE` as a valid value for the `algo` parameter. When this option is used, the library benchmarks a selection of available algorithms internally and chooses the optimal one based on the given problem configuration. The selected algorithm is cached within the current `cublasHandle_t`, so subsequent calls with the same problem descriptor will reuse the cached configuration for improved performance.

This is an experimental feature. Users are encouraged to transition to the cuBLASLt API, which provides fine-grained control over algorithm selection through the heuristics API and includes support for additional data types such as FP8 and block-scaled formats, as well as kernel fusion. (See autotuning example in cuBLASLt).

- Improved performance of BLAS Level 3 non-GEMM kernels (SYRK, HERK, TRMM, SYMM, HEMM) for FP32 and CF32 precisions on NVIDIA Blackwell GPUs.
- This release adds support for `sm_110` GPUs for arm64-sbsa on Linux.

- **Resolved Issues**
  - Fixed an issue where some `cublasZtrmm()` kernels produced incorrect results when M is equal to 1 and `side` is `CUBLAS_SIDE_RIGHT` on NVIDIA Ada and Blackwell GeForce-class GPUs. *[5452663]*
- **Known Issues**
  - `cublasLtMatmul` previously ignored user-specified auxiliary (Aux) data types for ReLU epilogues and defaulted to using a bitmask. The correct behavior is now enforced: an error is returned if an invalid Aux data type is specified for ReLU epilogues. *[CUB-7984]*
  - Some `cublasSsyrkx()` kernels produce incorrect results when beta is equal to 0 on Blackwell GPUs. *[CUB-8846]*
  - `cublasLtMatmul()` may produce incorrect results when run concurrently with another kernel that uses Tensor Memory. This issue affects only algorithms with `CUBLASLT_ALGO_CONFIG_ID` equal to 66 on GPUs with compute capability 10.x and 11.x, and has existed since cuBLAS 12.8. *[5807900]*
- **Deprecations**
  - The experimental feature for atomic synchronization along the rows (`CUBLASLT_MATMUL_DESC_ATOMIC_SYNC_NUM_CHUNKS_D_ROWS`) and columns (`CUBLASLT_MATMUL_DESC_ATOMIC_SYNC_NUM_CHUNKS_D_COLS`) of the output matrix which was deprecated in 12.8 has now been **removed**.
  - Starting with this release, cuBLAS will return `CUBLAS_STATUS_NOT_SUPPORTED` if any of the following descriptor attributes are set but the corresponding scale is not supported:
    - `CUBLASLT_MATMUL_DESC_A_SCALE_POINTER`
    - `CUBLASLT_MATMUL_DESC_B_SCALE_POINTER`
    - `CUBLASLT_MATMUL_DESC_D_SCALE_POINTER`
    - `CUBLASLT_MATMUL_DESC_D_OUT_SCALE_POINTER`
    - `CUBLASLT_MATMUL_DESC_EPILOGUE_AUX_SCALE_POINTER`
  - Previously, this restriction applied only to non-narrow precision matmuls. It now also applies to narrow precision matmuls when a scale is set for a non-narrow precision tensor.

3.2. cuFFT Library

3.2.1. cuFFT: Release 13.4

- **New Features**
  - Added new plan property `NVFFT_PLAN_PROPERTY_INT64_BLUESTEIN_ONLY` to enforce Bluestein-only kernels.
  - Added new plan property `NVFFT_PLAN_PROPERTY_INT64_DISABLE_FMA` to disable fused multiply-add contraction when using LTO kernels.
- **Resolved Issues**
  - Added alignment checks that prevent unaligned memory access when using user-provided work areas.
  - Fixed a correctness issue in real-side LTO callback kernels for R2C and C2R transforms. This issue was first identified in CUDA Toolkit 13.3 Update 1.

3.2.2. cuFFT: Release 13.3 Update 1

- **Known Issues**
  - An issue identified in CUDA 13.3 affects the correctness of real-side LTO callback kernels for R2C and C2R transforms. It affects only even sizes above certain large thresholds (8192 or greater in single precision, or 4096 or greater in double precision) whose length has a largest prime factor of at least 127.

3.2.3. cuFFT: Release 13.3

- **New Features**
  - Expanded LTO support to include transform sizes divisible by primes larger than 127, along with increased callback support.
- **Resolved Issues**
  - Fixed an issue where `cufftXtQueryPlan` could result in floating-point exceptions when querying multi-GPU plans that are not single-batch one-dimensional FFTs. *[5923044]*

3.2.4. cuFFT: Release 13.2

- **New Features**
  - Using cuFFT link-time optimized (LTO) kernels now requires NVRTC.
- **Deprecations**
  - `cufftDebug` is deprecated and will be removed in a future release.

3.2.5. cuFFT: Release 13.1

- **New Features**
  - Improved performance for transforms whose sizes are powers of 2, 3, 5, and 7 on Blackwell GPUs, in both single and double precision.
  - Improved performance for selected power-of-two sizes in 2D and 3D transforms, in both single and double precision.
  - Introduced an experimental cuFFT device API that provides host functions to query or generate device function code and exposes database metadata through a C++ header for use with the cuFFTDx library.
- **Resolved Issues**
  - Fixed a correctness issue, identified in CUDA 13.0, that affected a very specific subset of kernels: half- and bfloat16-precision strided R2C and C2R FFTs of size 1.

3.2.6. cuFFT: Release 13.0 Update 1

- **Known Issues**
  - In CUDA 13.0, a correctness issue affects a specific subset of kernels, namely half and bfloat precision size 1 strided R2C and C2R kernels. A fix will be included in a future CUDA release.

3.2.7. cuFFT: Release 13.0

- **New Features**

- Added new error codes:
  - `CUFFT_MISSING_DEPENDENCY`
  - `CUFFT_NVRTC_FAILURE`
  - `CUFFT_NVJITLINK_FAILURE`
  - `CUFFT_NVSHMEM_FAILURE`
- Introduced `CUFFT_PLAN_NULL`, a value that can be assigned to a `cufftHandle` to indicate a null handle. It is safe to call `cufftDestroy` on a null handle.
- Improved performance for single-precision C2C multi-dimensional FFTs and large power-of-2 FFTs.

- **Known Issues**
  - An issue identified in CUDA 13.0 affects the correctness of a specific subset of cuFFT kernels, specifically half-precision and bfloat16 size-1 strided R2C and C2R transforms. A fix will be included in a future CUDA release.
- **Deprecations**
  - Removed support for Maxwell, Pascal, and Volta GPUs, corresponding to compute capabilities earlier than Turing.
  - Removed legacy cuFFT error codes:
    - `CUFFT_INCOMPLETE_PARAMETER_LIST`
    - `CUFFT_PARSE_ERROR`
    - `CUFFT_LICENSE_ERROR`
  - Removed the `libcufft../_static_nocallback.a` static library. Users should link against `libcufft../_static.a` instead, as both are functionally equivalent.

3.3. cuSOLVER Library

3.3.1. cuSOLVER: Release 13.4

- **New Features**
  - Further reduced device workspace for `cusolverDnXgesvd` when singular vectors are requested and the input matrix is tall-and-skinny (`m > n`).
  - Reduced device workspace for `cusolverDnXgeev` by optimizing the Householder reflector accumulation (ORGHR) stage, enabling computation of larger eigenvalue problems within available GPU memory.
- **Resolved Issues**
  - Fixed an issue where `cusolverDnXgetrf` silently discarded a NaN value in the input matrix instead of propagating it to the factored output. *[5849138]*

3.3.2. cuSOLVER: Release 13.3 Update 1

- **New Features**
  - Improved `cusolverDnXgetrf` performance with pivoting for `sm_90`, `sm_100`, `sm_103`, and `sm_120`.
- **Resolved Issues**
  - Fixed an issue where `cusolverDn{C/Z}sytrf()` and `cusolverDn{C/Z}sytrs()` with `devIpiv == nullptr` could treat the complex symmetric input as Hermitian instead of symmetric, which could lead to incorrect results for complex symmetric problems. The documented symmetric factorization and solve behaviors have been restored.

- Fixed accuracy issues on ill-conditioned and rank-deficient matrices for `cusolverDnXgesvdp` and `cusolverDnXpolar`.

3.3.3. cuSOLVER: Release 13.3

- **New Features**
  - Improved `cusolverDnXgeev` performance when computing eigenvectors by moving eigenvector post-processing from the host to the device.
- **Known Issues**
  - The `cusolverDn{C,Z}sytrf` and `cusolverDnXsytrs` APIs assume that the complex input matrix `A` is Hermitian instead of symmetric when `devIpiv` is set to `NULL`. This issue exists starting with CUDA Toolkit 13.1. *[5797471]*

3.3.4. cuSOLVER: Release 13.2 Update 1

- **New Features**
  - Improved performance of `cusolverDnXgeqrf()` and `cusolverDn<S,D,C,Z>geqrf()` on `sm_90`, `sm_100`, `sm_103`, and `sm_120` for matrices with `m <= 65536`.
  - Added the new public 64-bit interface `cusolverDnXpolar()`, which exposes the QDWH algorithm implementation for polar decomposition in cuSOLVERDn.
  - Added the new public 64-bit interface `cusolverDnXstedc()`, which computes the eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the divide-and-conquer method.

3.3.5. cuSOLVER: Release 13.2

- **New Features**
  - Added FP64 fixed-point emulation support to cuSOLVERDn. The following new APIs are available:
    - `cusolverDnSetFixedPointEmulationMantissaControl()`
    - `cusolverDnGetFixedPointEmulationMantissaControl()`
    - `cusolverDnSetFixedPointEmulationMaxMantissaBitCount()`
    - `cusolverDnGetFixedPointEmulationMaxMantissaBitCount()`
    - `cusolverDnSetFixedPointEmulationMantissaBitOffset()`
    - `cusolverDnGetFixedPointEmulationMantissaBitOffset()`
    - `cusolverDnSetEmulationSpecialValuesSupport()`
    - `cusolverDnGetEmulationSpecialValuesSupport()`
  - Added the `cusolverDnXsygvd` API to support larger problem sizes.
- **Known Issues**
  - Starting with CUDA Toolkit 13.1, `cusolverDn{C,Z}sytrf` and `cusolverDnXsytrs` assume the complex input matrix `A` is Hermitian (instead of symmetric) when `devIpiv == NULL`. *[5797471]*

3.3.6. cuSOLVER: Release 13.1

- **Resolved Issues**
  - Fixed a bug that prevented users from changing the algorithm for `cusolverDnXsyevBatched` by using `cusolverDnSetAdvOptions`. *[5539844]*

3.3.7. cuSOLVER: Release 13.0 Update 1

- **Resolved Issues**
  - Fixed a race condition in `cusolverDnXgeev` that could occur when using multiple host threads with either separate handles per thread or a shared handle, which caused execution to abort and returned `CUSOLVER_STATUS_INTERNAL_ERROR`.

3.3.8. cuSOLVER: Release 13.0

- **New Features**
  - cuSOLVER offers a new math mode to leverage improved performance of emulated FP32 arithmetic on NVIDIA Blackwell GPUs.
    To enable and control this feature, the following new APIs have been added:
    - `cusolverDnSetMathMode()`
    - `cusolverDnGetMathMode()`
    - `cusolverDnSetEmulationStrategy()`
    - `cusolverDnGetEmulationStrategy()`
  - Performance improvements for `cusolverDnXsyevBatched()` have been made by introducing an internal algorithm switch on Blackwell GPUs for matrices of size `n <= 32`.
    To revert to the previous algorithm for all problem sizes, use cusolverDnSetAdvOptions().
    For more details, refer to the cusolverDnXsyevBatched() documentation.
- **Deprecations**
  - `cuSOLVERMg` is deprecated and may be removed in an upcoming major release. Users are encouraged to use cuSOLVERMp for multi-GPU functionality across both single and multi-node environments. To disable the deprecation warning, add the compiler flag `-DDISABLE_CUSOLVERMG_DEPRECATED`.
  - `cuSOLVERSp` and `cuSOLVERRf` are fully deprecated and may be removed in an upcoming major release. Users are encouraged to use the cuDSS library for better performance and ongoing support.
    For help with the transition, refer to the cuDSS samples or CUDA samples for migrating from `cuSOLVERSp` to `cuDSS`.
    To disable the deprecation warning, add the compiler flag: `-DDISABLE_CUSOLVER_DEPRECATED`.
- **Resolved Issues**
  - The supported input matrix size for `cusolverDnXsyevd`, `cusolverDnXsyevdx`, `cusolverDnXsyevBatched`, `cusolverDn<t>syevd`, and `cusolverDn<t>syevdx` is no longer limited to `n <= 32768`.
    This update also applies to routines that share the same internal implementation: `cusolverDnXgesvdr`, `cusolverDnXgesvdp`, `cusolverDn<t>sygvd`, `cusolverDn<t>sygvdx`, and `cusolverDn<t>gesvdaStridedBatched`.

3.4. cuSPARSE Library

3.4.1. cuSPARSE: Release 13.4

- **Known Issues**
  - Mixed-precision CSR/COO `SpMM` is not supported in some cases. Refer to the cuSPARSE documentation for details.
- **Resolved Issues**
  - Fixed a bug with cached parameters in `SpMVOp`.

3.4.2. cuSPARSE: Release 13.3 Update 1

- **New Features**
  - Improved `SpGEAM` performance by an average of 40%. *[CUSPARSE-3361]*
  - Reduced preprocessing time for `SpMM ALG3`.
- **Known Issues**
  - Incorrect result when running `BSR SDDMM` on a very large matrix.
- **Resolved Issues**
  - Fixed a rarely occurring issue in CSC and transposed CSR `SpMV`. *[5975307]*
  - Fixed an accuracy issue in mixed-precision SELL `SpMV`.
  - Fixed an issue with the algorithm configuration cache in `SpMVOp`.

3.4.3. cuSPARSE: Release 13.3

- **New Features**
  - Added support for the CSC format in `SpSV` and `SpSM`.
  - Improved `CSR SpMV ALG2` performance by an average of 11%.
  - Added the Generic API `SpGEAM` for sparse matrix-matrix addition.
  - Added `SpMVOp ALG1` with reduced preprocessing overhead.
  - Added support for mixed index types in `SpMVOp` computation for CSR matrices with 64-bit offsets and 32-bit indices.
  - Added support for the FP32 data type in `SpMVOp`.
  - Avoided recompilation for the same epilogue in `SpMVOp`.
  - Added mixed-precision support in `SpMV` for 32-bit input matrices and 64-bit input vectors.
  - Added support for updating matrix values after preprocessing in `SpMVOp ALG1`.
- **Resolved Issues**
  - Fixed a memory leak in `SpMVOp` when `destroy_lrb()` was called. *[5974043]*

3.4.4. cuSPARSE: Release 13.2 Update 1

- **New Features**
  - Improved `cusparseSpMVOp_createDescr()` performance by up to 2.5x.
  - Reduced `cusparseSpMVOp_createPlan()` planning latency for default epilogues through ahead-of-time compilation, avoiding JIT compilation in this case.
- **Resolved Issues**
  - Fixed an issue that caused performance regressions in BSR SpMM for certain block sizes. *[5860241]*
- **Deprecation**
  - Deprecated the `SpMMOp` and `SpGEMMreuse` APIs.

3.4.5. cuSPARSE: Release 13.2

- **New Features**
  - Improved the runtime of the `SpMVOp::buffer_size_estimate` API.

3.4.6. cuSPARSE: Release 13.1 Update 1

- **New Features**
  - Added a new `cusparseSpMVOp_bufferSize` API that returns the size of the workspace buffer required for SpMVOp computations. Users provide this buffer when creating `cusparseSpMVOpDescr_t`, removing internal memory allocations.
  - Improved SpMVOp performance on B200. *[CUSPARSE-2931] [CUSPARSE-2932] [CUSPARSE-2933]*
- **Resolved Issues**
  - Fixed an accuracy issue in mixed-precision CSR/COO SpMM computations. *[CUSPARSE-2349]*
  - Fixed an issue in CSR SpMM computations when the input dense matrix has a high number of columns. *[CUSPARSE-2301]*

3.4.7. cuSPARSE: Release 13.1

- **New Features**
  - Introduced an experimental Sparse Matrix-Vector Multiplication (SpMVOp) API that provides improved performance compared with the existing generic CsrMV API. This API supports CSR format with 32-bit indices, double precision, and user-defined epilogues.
  - The nvJitLink shared library is now loaded dynamically at runtime.
  - Improved `cusparseXcsrsort` with reduced memory usage and higher performance. *[CUSPARSE-2630]*
- **Known Issues**
  - When using 32-bit indexing, `cusparseSpSV` and `cusparseSpSM` may crash if the number of nonzero elements (nnz) approaches `2^31 - 1`. *[CUSPARSE-2211]*
- **Resolved Issues**
  - Fixed potential issues when input and output pointers are not 16-byte aligned in `cusparseCsr2cscEx2`, `cusparseSparseToDense`, and CSR/COO `cusparseSpMM`. *[CUSPARSE-2380]*
  - Fixed a determinism issue in CSR `cusparseSpMM` ALG3. *[CUSPARSE-2612]*
  - All routines now support matrices with up to `2^31 - 1` nonzero elements (nnz) when using 32-bit indexing, with the exception of `cusparseSpSV` and `cusparseSpSM`. *[CUSPARSE-2153]*
  - Fixed a potential race condition that could occur when dynamically loading driver APIs. *[CUSPARSE-2764]*

3.4.8. cuSPARSE: Release 13.0 Update 1

- **New Features**
  - Added support for the BSR format in the generic SpMV API *(CUSPARSE-2518)*.
- **Deprecation**
  - Deprecated the legacy BSR SpMV API (replaced by the generic SpMV API).
- **Resolved Issues**
  - Enabled all generic APIs to support zero-dimension matrices/vectors (m, n, k = 0) *(CUSPARSE-2378)*.
  - Enabled all generic APIs to support small-dimension matrices/vectors (small m, n, or k) *(CUSPARSE-2379)*.
  - Fixed incorrect results in mixed-precision CSR/COO SpMV computations *(CUSPARSE-2349)*.

3.4.9. cuSPARSE: Release 13.0

- **New Features**
  - Added support for 64-bit index matrices in SpGEMM computation. *(CUSPARSE-2365)*
- **Known Issues**
  - cuSPARSE logging APIs can crash on Windows.
  - `CUSPARSE_SPMM_CSR_ALG3` does not return deterministic results as stated in the documentation.
- **Deprecation**
  - Dropped support for pre-Turing architectures (Maxwell, Volta, and Pascal).
- **Resolved Issues**
  - Fixed a bug in `cusparseSparseToDense_bufferSize` that caused it to request up to 16× more memory than required. *[CUSPARSE-2352]*
  - Fixed unwanted 16-byte alignment requirements on the external buffer. Most routines will now work with any alignment. In the generic API, only `cusparseSpGEMM` routines are still affected. *[CUSPARSE-2352]*
  - Fixed incorrect results from `cusparseCsr2cscEx2` when any of the input matrix dimensions are zero, such as when `m = 0` or `n = 0`. *[CUSPARSE-2319]*
  - Fixed incorrect results from CSR SpMV when any of the input matrix dimensions are zero, such as when `m = 0` or `n = 0`. *[CUSPARSE-1800]*

3.5. Math Library

3.5.1. CUDA Math: Release 13.4

- **New Features**
  - Added new FP8 scaling factor type `__nv_fp8_ue5m3` and corresponding conversion operations in `cuda_fp8.h`. *[5482730]*
- **Resolved Issues**
  - Fixed `fp128` nearest integral functions returning off-by-one results in pathological cases. This fix also delivers performance improvements of 2x–8x for fp128 operations on GB200 and up to approximately 200x on RTX 5080, covering conversion, nearest integral, `frexp`, `ilogb`, `modf`, and multiplication. Fixed an underflow in fp128 `hypot`.
  - Fixed `__hmin_nan` and `__hmax_nan` bfloat16 functions ignoring the sign of zero in emulation code paths for devices with compute capability below 8.0 and host CPUs; `-0.0` now correctly compares as less than `+0.0`. *[6182113]*
  - Fixed `__nv_fp8_ue5m3` conversions from long integer types that could produce a result off by one. *[6182039]*

3.5.2. CUDA Math: Release 13.3

- **Resolved Issues**
  - Fixed an issue where silent data corruption could occur when the CUDA Math API `__mul24()` intrinsic was called with compile-time constant inputs due to undefined behavior from compiler optimizations applied to overflowing signed integer multiplication. This issue was introduced in CUDA Toolkit 11.1 and resolved in CUDA Toolkit 13.3. *[5807344]*

3.5.3. CUDA Math: Release 13.2 Update 1

- **Known Issues**
  - Silent data corruption can occur when the CUDA Math API `__mul24()` intrinsic is called with compile-time constant inputs. Compiler optimizations applied to overflowing signed integer multiplication can expose the program to undefined behavior. This issue was introduced in CUDA Toolkit 11.1 and will be fixed in a future release. *[5807344]*

3.5.4. CUDA Math: Release 13.2

- **New Features**
  - Accuracy and performance improvements were made to the following libdevice single-precision math functions:
    - `expm1f()`: up to 20% faster, with minor accuracy improvements.
    - `erff()`: 5% to 10% faster, with minor accuracy improvements.

  These gains come from algorithmic simplifications, reduced branching, and tighter approximations. *[5480287]*
- **Resolved Issues**
  - ACLE extension support for GCC: ARM64 users, notice the behavior change for `__clz` and `__clzll` CUDA Math integer intrinsics: the signatures were updated to match the host compiler's declarations, relevant for GCC 11.4 onwards. The (un-)signedness of the return type of the intrinsic affects the caller code relying on integer types promotions. *[6258270]*

3.5.5. CUDA Math: Release 13.0

- **New Features**
  - Single and double precision math functions received targeted performance and accuracy improvements through algorithmic simplifications, reduced branching, and tighter approximations.
    - `atan2f`, `atan2`: Up to 10% faster with minor improvements in accuracy.
    - `sinhf`, `coshf`, `acoshf`, `asinhf`, `asinh`: Up to 50% speedups with minor improvements in accuracy.
    - `cbrtf`, `rcbrtf`: 15% faster with minor improvements in accuracy.
    - `erfinvf`, `erfcinvf`, `normcdfinvf`: Minor accuracy improvements, performance neutral.
    - `ldexpf`, `ldexp`: Up to 3x faster in single precision and 30% faster in double precision, with no accuracy loss.
    - `modff`, `modf`: Up to 50% faster in single precision and 10% faster in double precision, with no accuracy loss.

3.6. nvJPEG Library

3.6.1. nvJPEG: Release 13.3 Update 1

- **Resolved Issues**
  - Fixed an issue that would cause `nvjpegCreate*` calls to error out on Orin. *[6176492]*

3.6.2. nvJPEG: Release 13.3

- **New Features**
  - Added support for region-of-interest decoding with `nvjpegDecodeBatchedEx` when using the `NVJPEG_BACKEND_LOSSLESS_JPEG` backend.
- **Resolved Issues**
  - Fixed an issue with boundary handling when decoding a region of interest with `NVJPEG_FLAGS_UPSAMPLING_WITH_INTERPOLATION` enabled.

3.6.3. nvJPEG: Release 13.2 Update 1

- **New Features**
  - Added the `NVJPEG_OUTPUT_UNCHANGEDI` enum value to `nvjpegOutputFormat_t` for unchanged interleaved output. For chroma subsampling formats other than 4:4:4, chroma values are duplicated so that the chroma and luma dimensions match.

3.6.4. nvJPEG: Release 13.1

- **Resolved Issues**
  - nvJPEG's lossless JPEG 92 (lj92) implementation can now correctly handle lj92 files that contain a comment marker in the header. *[5484797]*

3.6.5. nvJPEG: Release 13.0 Update 1

- **Resolved Issues**
  - Fixed a race condition in certain cases during progressive encoding *[5307748]*.
  - Fixed an uninitialized read when encoding images as 4:1:0 JPEG bitstreams *[5308008]*.

3.6.6. nvJPEG: Release 13.0

- **Deprecations**
  - Removed the `nvjpegEncoderParamsCopyHuffmanTables` API.
- **Resolved Issues**
  - nvJPEG is now more robust and no longer crashes or exhibits undefined behavior when decoding malformed or truncated bitstreams. *[5168024, 5133845, 5143450]*
  - `nvjpegEncodeYUV` now avoids reading outside of allocated device memory in certain cases. *[5133826]*
  - Optimized memory usage when encoding RGB inputs using the hardware encoder.
  - Fixed issues related to rounding in various transform, sampling, and conversion steps, improving image quality for both encoder and decoder. *[5064901, 3976092]*
  - Various bug fixes for improved security.

3.7. NPP Library

3.7.1. NPP: Release 13.4

- **Resolved Issues**

- Fixed a regression in `nppiNV12ToRGB_8u_ColorTwist32f_P2C3R_Ctx` introduced in CUDA 12.9 that could render black NV12 frames as blue. *[6229084]*

3.7.2. NPP: Release 13.1 Update 1

- **Resolved Issues**
  - Reduced nvJPEG Encoder initialization time on Thor. *[5533951]*

3.7.3. NPP: Release 13.1

- **Resolved Issues**
  - Fixed an issue in `nppiCFAToRGB_8u_C1C3R()` affecting SSIM validation for `NPPI_BAYER_GBRG` patterns. *[5192648]*

3.7.4. NPP: Release 13.0

- **Deprecations**
  - **Removal of Legacy Non-Context APIs**
    All legacy NPP APIs without the `_Ctx` suffix have been deprecated and are now removed starting with this release. Developers should transition to the context-aware (`_Ctx`) versions to ensure continued support and compatibility with the latest CUDA releases.
  - **Deprecation of "nppGetStreamContext()"**
    The `nppGetStreamContext()` API has been deprecated and removed. Developers are strongly encouraged to adopt application-managed stream contexts by explicitly managing the `NppStreamContext` structure. For guidance, refer to the NPP Documentation – General Conventions.
- **Resolved Issues**
  - Fixed an issue in `nppiFloodFillRange_8u_C1IR_Ctx` where the flood fill operation did not correctly fill the full target area. *[5141474]*
  - Resolved a bug in the `nppiDebayer()` API that affected proper reconstruction of color data during Bayer pattern conversion. *[5138782]*

Chapter 4

Notices

4.1. Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

4.2. OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

4.3. Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.