NVIDIA TensorRT

Release Notes | NVIDIA Docs

RN-08624-001_v10.2.0 | July 2024

Table of Contents

Chapter 1. TensorRT Release 10.x.x

1.1. TensorRT Release 10.2.0

These are the TensorRT 10.2.0 Release Notes and are applicable to x86 Linux and Windows users, and Arm®-based CPU cores for Server Base System Architecture (SBSA) users on Linux. This release includes several fixes from the previous TensorRT releases as well as the following additional changes.

For previously released TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- Added support for normal FP8 Convolutions on Hopper GPUs.
- Added support for FP8 MHA fusion for SeqLen>512 on Hopper GPUs.
- Improved `InstanceNorm` and `GroupNorm` fusions for `StableDiffusion` models.
- Improved DRAM utilization for `LayerNorm`, pointwise, and data movements (for example, Concats, Slices, Reshapes, Transposes) kernels on GPUs with HBM memory.
- Added new APIs in `INetworkDefinition` for fine grained control of refittable weights:
  - `markWeightsRefittable` to mark weights as refittable
  - `unmarkWeightsRefittable` to unmark weights as refittable
  - `areWeightsMarkedRefittable` to query if a weight is marked as refittable

  This fine grained refit control is only valid when the new `kREFIT_INDIVIDUAL` builder flag is used during engine build. It also works with `kSTRIP_PLAN`, enabling the construction of a weight-stripped engine that can be updated from a fine-tuned checkpoint when all weights are marked as refittable in the fine-tuned checkpoint.

Compatibility

- TensorRT 10.2.0 has been tested with the following:

  - TensorFlow 2.13.1
  - PyTorch >= 2.0 (refer to the `requirements.txt` file for each sample)
  - ONNX 1.16.0
- This TensorRT release supports CUDA®:
  - 12.5
  - 12.4 update 1
  - 12.3 update 2
  - 12.2 update 1
  - 12.1 update 1
  - 12.0 update 1
  - 11.8
  - 11.7 update 1
  - 11.6 update 2
  - 11.5 update 2
  - 11.4 update 4
  - 11.3 update 1
  - 11.2 update 2
  - 11.1 update 3
  - 11.0 update 1
- This TensorRT release requires at least NVIDIA driver r450 on Linux or r452 on Windows as required by CUDA 11.0, which is the minimum CUDA version supported by this TensorRT release.

Limitations

- There are no optimized FP8 Convolutions for Group Convolutions and Depthwise Convolutions, therefore, INT8 is still recommended for ConvNets containing these convolution ops.
- The accumulation `dtype` for the batched GEMMS in the FP8 MHA must be in FP32.
  - This can be achieved by adding Cast (to FP32) ops before the batched GEMM and a Cast (to FP16) op after the batched GEMM.
  - Alternatively, you can convert your ONNX model using TensorRT Model Optimizer which adds the Cast ops automatically.
- There cannot be any pointwise operations between the first batched GEMM and the softmax inside FP8 MHAs, such as having an attention mask. This will be improved in future TensorRT releases.
- On QNX, networks that are segmented into a large number of DLA loadables may fail during inference.

- The DLA compiler is capable of removing identity transposes, but it cannot fuse multiple adjacent transpose layers into a single transpose layer (likewise for reshape). For example, given a TensorRT `IShuffleLayer` consisting of two non-trivial transposes and an identity reshapes in between. The shuffle layer is translated into two consecutive DLA transpose layers, unless you merge the transposes together manually in the model definition in advance.
- `nvinfer1::UnaryOperation::kROUND` or `nvinfer1::UnaryOperation::kSIGN` operations of `IUnaryLayer` are not supported in the implicit batch mode.
- For networks containing normalization layers, particularly if deploying with mixed precision, target the latest ONNX opset that contains the corresponding function ops, for example: opset 17 for LayerNormalization or opset 18 GroupNormalization. Numerical accuracy using function ops is superior to corresponding implementation with primitive ops for normalization layers.
- The `kREFIT` and `kREFIT_IDENTICAL` have performance regressions compared with non-refit engines where convolution layers are present within a branch or loop and the precision is FP16/INT8. This issue will be addressed in future releases.
- Weight streaming mainly supports GEMM-based networks like Transformers for now. Convolution-based networks may have only a few weights that can be streamed.
- Deprecated INT8 implicit quantization and calibrator APIs including `dynamicRangeIsSet`, `CalibrationAlgoType`, `IInt8Calibrator`, `IInt8EntropyCalibrator`, `IInt8EntropyCalibrator2`, `IInt8MinMaxCalibrator`, `IInt8Calibrator`, `setInt8Calibrator`, `getInt8Calibrator`, `setCalibrationProfile`, `getCalibrationProfile`, `setDynamicRange`, `getDynamicRangeMin`, `getDynamicRangeMax`, and `getTensorsWithDynamicRange`. They may not give the optimal performance and accuracy. As a workaround, use INT8 explicit quantization instead.

Deprecated API Lifetime

- APIs deprecated in TensorRT 10.2 will be retained until at least 7/2025.
- APIs deprecated in TensorRT 10.1 will be retained until at least 5/2025.
- APIs deprecated in TensorRT 10.0 will be retained until at least 3/2025.
- APIs deprecated in TensorRT 9.3 will be retained until at least 1/2025.
- APIs deprecated in TensorRT 9.2 will be retained until at least 11/2024.
- APIs deprecated in TensorRT 9.1 will be retained until at least 10/2024.
- APIs deprecated in TensorRT 9.0 will be retained until at least 8/2024.

Refer to the API documentation (C++, Python) for how to update your code to remove the use of deprecated features.

Deprecated and Removed Features

The following features have been deprecated or removed in TensorRT 10.2.0.

- Deprecated NVIDIA Volta support (GPUs with compute capability 7.0) starting with TensorRT 10.0. Volta support may be removed after 9/2024.

Fixed Issues

- Fixed issue where engine building with weight streaming enabled would fail when the model size exceeds the available device memory.
- Fixed issue of decreased weight streaming performance when creating execution contexts with multiple optimization profiles using external device memory and call `setDeviceMemory`/`setDeviceMemoryV2` before `setOptimizationProfileAsync`.
- Fixed issue with header files in the `include` directory for Windows being encoded as UTF-16 instead of UTF-8.
- In the previous 10.0 and 10.1 TensorRT releases the `tensorrt` Python metapackage did not pin the version for the Python module dependency `tensorrt-cu12`. This caused the latest TensorRT version to always be installed. This issue has been fixed.
- Fixed issue with large models where TensorRT does not free memory after an OOM error, causing tactics that should fit in memory to also fail.
- If a network has a tensor of type bool with an *implicitly* data-dependent shape, engine building should now work.
- The ONNX version in `requirements.txt` for sample python/efficientdet and python/tensorflow_object_detection_api is incompatible with the samples. The workaround was to pin the ONNX version to 1.14.0 for the samples to function correctly. This issue has been fixed.

Known Issues

Functional

- `nvinfer1::ISliceLayer` with modes `nvinfer1::SampleMode::kCLAMP` or `nvinfer1::SampleMode::kFILL` (ONNX equivalent being a `Pad` op with either `constant` or `edge` mode) may break during engine compilation if the slice input is a constant tensor. To workaround this issue use TensorRT 10.1. This will be addressed in future TensorRT releases.
- There is a known accuracy issue when running ResNet18/ResNet50 with FP8 convolutions. This will be fixed in the next TensorRT version.
- The Python sample detectron2, efficientdet, efficientnet, engine_refit_onnx_bidaf, introductory_parser_samples, network_api_pytorch_mnist, onnx_custom_plugin, onnx_packnet, sample_weight_stripping, simple_progress_monitor, tensorflow_object_detection_api, and yolo_v3_onnx does not support Python 3.12. Support will be added in 10.3. The issue is fixed in OSS 10.2 release
- If TensorRT 8.6 or 9.x was installed using the Python Package Index (PyPI) you will not be able to upgrade TensorRT to 10.x using PyPI. You must first uninstall TensorRT using `pip uninstall tensorrt tensorrt-libs tensorrt-bindings` and then

reinstall TensorRT using "`pip install tensorrt`". This will remove the previous TensorRT version and install the latest TensorRT 10.x. This step is required because the suffix `-cuXX` was added to the Python package names, which prevents the upgrade from working properly.

- CUDA compute sanitizer may report racecheck hazards for some legacy kernels, however, related kernels do not have functional issues at runtime.
- The compute sanitizer `initcheck` tool may flag false positive `Uninitialized __global__ memory read` errors when running TensorRT applications on NVIDIA Hopper GPUs. These errors can be safely ignored and will be fixed in an upcoming CUDA release.
- Multihead attention fusion might not happen and affect performance if the number of heads is small.
- There is an occurrence of use-after-free in NVRTC that has been fixed in CUDA 12.1. When using NVRTC from CUDA 12.0 together with the TensorRT static library, you may encounter a crash in certain scenarios. Linking with the NVRTC and PTXJIT compiler from CUDA 12.1 or newer will resolve this issue.
- There are known issues reported by the Valgrind memory leak check tool when detecting potential memory leaks from TensorRT applications. The recommendation to suppress the issues is to provide a Valgrind suppression file with the following contents when running the Valgrind memory leak check tool. Add the option `--keep-debuginfo=yes` to the Valgrind command line to suppress these errors.

```
{
  Memory leak errors with dlopen.
   Memcheck:Leak
   match-leak-kinds: definite
   ...
   fun:*dlopen*
   ...
}
{
    Memory leak errors with nvrtc
    Memcheck:Leak
    match-leak-kinds: definite
    fun:malloc
    obj:*libnvrtc.so*
    ...
}
```

- SM 7.5 and earlier devices may not have INT8 implementations for all layers with Q/DQ nodes. In this case, you will encounter a `could not find any implementation` error while building your engine. To resolve this, remove the Q/DQ nodes, which quantize the failing layers.
- Installing the `cuda-compat-11-4` package may interfere with CUDA enhanced compatibility and cause TensorRT to fail even when the driver is r465. The workaround is to remove the `cuda-compat-11-4` package or upgrade the driver to r470. *(not applicable for Jetson platforms)*
- For some networks, using a batch size of 4096 may cause accuracy degradation on DLA.

- For broadcasting elementwise layers running on DLA with GPU fallback enabled with one NxCxHxW input and one Nx1x1x1 input, there is a known accuracy issue if at least one of the inputs is consumed in `kDLA_LINEAR` format. It is recommended to explicitly set the input formats of such elementwise layers to different tensor formats.
- Exclusive padding with `kAVERAGE` pooling is not supported.
- The Valgrind tool found a memory leak on L4T with CUDA 12.4 due to a known driver issue. This is expected to be fixed in CUDA 12.6.
- Asynchronous CUDA calls are not supported in the user defined `processDebugTensor` function for the debug tensor feature due to a bug in Windows 10.
- There is a known accuracy issue when the network contains two consecutive GEMV operations (that is, MatrixMultiply with `gemmM` or `gemmN == 1`). To workaround this issue, try padding the MatrixMultiply input to have dimensions greater than `1`.
- The size of the `libnvinfer_lean.so` library has increased by 10 MB. This issue will be resolved in TensorRT 10.3.
- When compiling samples with static linking, if the error message `/usr/bin/ld: failed to convert GOTPCREL relocation; relink with -no-relax` is shown, then add `-Wl,--no-relax` to the linking steps in `samples/Makefile.config`.
- There is a known accuracy issue when binding an INT4 tensor as a network output. To workaround this, add an `IDequantizeLayer` before the output.
- With CUDA 12.5 on Windows, `fcPlugin` (`CustomFCPluginDynamic`) may result in CUDA errors on certain GPUs.
- There may be divide-by-zero errors for TopK when K equals to 0 and the reduction dimension is large.

Performance

- There is an up to 20% CPU memory usage increase compared to TensorRT 10.1 when building engines on A10 GPUs.
- There is an up to 25% performance regression when running TensorRT-LLM without the attention plugin. The current recommendation is to always enable the attention plugin when using TensorRT-LLM.
- There is an up to 24% performance regression for EfficientNets, StableDiffusion CLIP, and StableDiffusion UNet on RTX 3070 GPU on Windows.
- There is an up to 10% performance regression for ConvNext on NVIDIA Orin compared to TensorRT 9.3.
- There are known performance gaps between engines built with REFIT enabled and engines built with REFIT disabled.
- There is an up to 4x performance regression for networks containing `GridSample` ops compared to TensorRT 9.2.

- There are up to 60 MB engine size fluctuations for the BERT-Large INT8-QDQ model on Orin due to unstable tactic selection among tactics.
- There is an up to 16% performance regression for BasicUNet, DynUNet, and HighResNet in INT8 precision compared to TensorRT 9.3.
- There is an up to 40 second increase in engine building for BART networks on NVIDIA Hopper GPUs.
- There is an up to 20 second increase in engine building for some large language models (LLMs) on NVIDIA Ampere GPUs.
- There is an up to 2.5x build time increase compared to TensorRT 9.0 for certain Bert like models due to additional tactics available for evaluation.
- There is an up to 13% performance drop for the CortanaASR model on NVIDIA Ampere GPUs compared to TensorRT 8.5.
- There is an up to 18% performance drop for the ShuffleNet model on A30/A40 compared to TensorRT 8.5.1.
- Convolution on a tensor with an *implicitly* data-dependent shape may run significantly slower than on other tensors of the same size. Refer to the Glossary for the definition of implicitly data-dependent shapes.
- For some Transformer models, including ViT, Swin-Transformer, and DETR, there is a performance drop in INT8 precision (including both explicit and implicit quantization) compared to FP16 precision.
- There is a known issue with DLA clocks that requires users to reboot the system after changing the `nvpmodel` power mode or otherwise experience a performance drop. Refer to the L4T board support package Release Notes for details.
- There is an up to 5% performance drop for networks using sparsity in FP16 precision.
- There is an up to 6% performance regression compared to TensorRT 8.5 on OpenRoadNet in FP16 precision on NVIDIA A10 GPUs.
- There is an up to 70% performance regression compared to TensorRT 8.6 on BERT networks in INT8 precision with FP16 disabled on L4 GPUs. To work around this, enable FP16 and disable INT8 in the builder config.
- In explicitly quantized networks, a group convolution that has a Q/DQ pair before but no Q/DQ pair after is expected to run with INT8-IN-FP32-OUT mixed precision. However, on NVIDIA Hopper, it may fall back to FP32-IN-FP32-OUT if the input channel count is small. This will be fixed in a future release.

1.2. TensorRT Release 10.1.0

These are the TensorRT 10.1.0 Release Notes and are applicable to x86 Linux and Windows users, Arm®-based CPU cores for Server Base System Architecture (SBSA) users on Linux, and JetPack users. This release includes several fixes from the previous TensorRT releases as well as the following additional changes.

For previously released TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Announcements

- The ONNX GraphSurgeon is no longer included inside the TensorRT package. Ensure you remove any previous Debian or RPM packages that may have been installed by a previous release and instead install ONNX GraphSurgeon using `pip`. For more information, refer to onnx-graphsurgeon.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- When using the TensorRT ONNX parser, shape inputs can now be passed to custom ops supported by IPluginV3-based plugins. The indices of the inputs to be interpreted as shape inputs must be indicated by a node attribute named `tensorrt_plugin_shape_input_indices` containing a list of integers.
- Added a new sample `non_zero_plugin`, which is a Python version of the C++ sample `sampleNonZeroPlugin`.
- TensorRT Python bindings natively support accessing the `data` attribute of a `PluginField` of `PluginFieldType.INT64` and `PluginFieldType.UNKNOWN` as NumPy arrays. For example, the NumPy functions `tobytes()` and `frombuffer()` may be used during storage and retrieval to embed an arbitrary NumPy array in a `PluginFieldType.UNKNOWN`.
- Add new APIs for weight streaming including `setWeightStreamingBudgetV2`, `getWeightStreamingBudgetV2`, `getWeightStreamingAutomaticBudget`, and `getWeightStreamingScratchMemorySize`. Now, weight streaming supports CUDA graph and multiple contexts of an engine running in parallel.
- Added new APIs for ascertaining an `IExecutionContext` device memory size and setting it: `ICudaEngine::getDeviceMemorySizeV2` and `IExecutionContext::setDeviceMemorySizeV2`. The V1 APIs are deprecated.
- Added native TensorRT layer support for ONNX operator `isNaN`, and added TensorRT plugin support for ONNX operator `DeformConv`.
- Internal exceptions are now contained and won't leak through the parser API boundaries.
- During `IPluginV3` auto-tuning, it is guaranteed that `configurePlugin()` is called with the current input/output format combination being timed before `getValidTactics()` is called. Therefore, it is possible to advertise a different set of tactics per each input/output format combination.

Compatibility

- TensorRT 10.1.0 has been tested with the following:

- TensorFlow 2.13.1
- PyTorch >= 2.0 (refer to the `requirements.txt` file for each sample)
- ONNX 1.16.0

- This TensorRT release supports CUDA®:
  - 12.4 update 1
  - 12.3 update 2
  - 12.2 update 1
  - 12.1 update 1
  - 12.0 update 1
  - 11.8
  - 11.7 update 1
  - 11.6 update 2
  - 11.5 update 2
  - 11.4 update 4
  - 11.3 update 1
  - 11.2 update 2
  - 11.1 update 3
  - 11.0 update 1
- This TensorRT release requires at least NVIDIA driver r450 on Linux or r452 on Windows as required by CUDA 11.0, which is the minimum CUDA version supported by this TensorRT release.

Limitations

- On QNX, networks that are segmented into a large number of DLA loadables may fail during inference.
- The DLA compiler is capable of removing identity transposes, but it cannot fuse multiple adjacent transpose layers into a single transpose layer (likewise for reshape). For example, given a TensorRT `IShuffleLayer` consisting of two non-trivial transposes and an identity reshapes in between. The shuffle layer is translated into two consecutive DLA transpose layers, unless you merge the transposes together manually in the model definition in advance.
- `nvinfer1::UnaryOperation::kROUND` or `nvinfer1::UnaryOperation::kSIGN` operations of `IUnaryLayer` are not supported in the implicit batch mode.
- For networks containing normalization layers, particularly if deploying with mixed precision, target the latest ONNX opset that contains the corresponding function ops, for example: opset 17 for LayerNormalization or opset 18 GroupNormalization. Numerical accuracy using function ops is superior to corresponding implementation with primitive ops for normalization layers.

- The `kREFIT` and `kREFIT_IDENTICAL` have performance regressions compared with non-refit engines where convolution layers are present within a branch or loop and the precision is FP16/INT8. This issue will be addressed in future releases.
- Weight streaming mainly supports GEMM-based networks like Transformers for now. Convolution-based networks may have only a few weights that can be streamed.

Deprecated API Lifetime

- APIs deprecated in TensorRT 10.1 will be retained until at least 5/2025.
- APIs deprecated in TensorRT 10.0 will be retained until at least 3/2025.
- APIs deprecated in TensorRT 9.3 will be retained until at least 1/2025.
- APIs deprecated in TensorRT 9.2 will be retained until at least 11/2024.
- APIs deprecated in TensorRT 9.1 will be retained until at least 10/2024.
- APIs deprecated in TensorRT 9.0 will be retained until at least 8/2024.

Refer to the API documentation (C++, Python) for how to update your code to remove the use of deprecated features.

Deprecated and Removed Features

The following features have been deprecated or removed in TensorRT 10.1.0.

- Deprecated NVIDIA Volta support (GPUs with compute capability 7.0) starting with TensorRT 10.0. Volta support may be removed after 9/2024.
- Version 1 of the ROIAlign plugin (`ROIAlign_TRT`), which implemented `IPluginV2DynamicExt`, is deprecated. It is superseded by version 2, which implements `IPluginV3`.
- The TensorRT standard plugin shared library (`libnvinfer_plugin.so` / `nvinfer_plugin.dll`) only exports `initLibNvInferPlugins`. No symbols in the `nvinfer1::plugin` namespace are exported anymore.
- Deprecated `IParser::supportsModel` and replaced this method with `IParser::supportsModelV2`, `IParser::getNbSubgraphs`, `IParser::isSubgraphSupported`, and `IParser::getSubgraphNodes`.
- Deprecated some weight streaming APIs including `setWeightStreamingBudget`, `getWeightStreamingBudget`, and `getMinimumWeightStreamingBudget`. Replaced by new versions of weight streaming APIs.
- Deprecated INT8 implicit quantization and calibrator APIs including `dynamicRangeIsSet`, `CalibrationAlgoType`, `IInt8Calibrator`, `IInt8EntropyCalibrator`, `IInt8EntropyCalibrator2`, `IInt8MinMaxCalibrator`, `IInt8Calibrator`, `setInt8Calibrator`, `getInt8Calibrator`, `setCalibrationProfile`, `getCalibrationProfile`, `setDynamicRange`, `getDynamicRangeMin`, `getDynamicRangeMax`, and `getTensorsWithDynamicRange`. They may not give the

optimal performance and accuracy. As a workaround, use INT8 explicit quantization instead.

Fixed Issues

- The `sampleNonZeroPlugin` sample failed to build when cross compiling for L4T. The workaround was to continue building the other samples by modifying `samples/Makefile` and removing the line containing `sampleNonZeroPlugin`. This issue has been fixed.
- The `sampleNonZeroPlugin` sample did not guarantee CUDA minor version compatibility. That is, if built against a newer CUDA Toolkit release, it may not function properly on older drivers, even within the same major CUDA release family. This issue has been fixed.
- `IPluginRegistry::deregisterLibrary()` did not work with plugin shared libraries with the defined entry point `getPluginCreators()`. `IPluginRegistry::loadLibrary()` was not impacted. The workaround was to deregister the plugins that were contained in such a library, manually query the library for `getPluginCreators()`, and invoke `IPluginRegistry::deregisterCreator()` for each creator retrieved. This issue has been fixed.
- On A30, some fused MHA (multi-head attention) performance was not optimized yet. This issue has been fixed.
- If building the TensorRT backend of ONNX runtime we are not able to use the prebuilt parser. This issue has been fixed.
- When using the Polygraphy `engine_from_network` API, if we enabled both refittable and `strip_plan` in the `create_config`, the final engine weights were not stripped. The workaround was, only include `strip_plan` in the `create_config`. This issue has been fixed.
- TensorRT did not support attention operations for tensors larger than `int32_t` maximum. Plugins could be used to workaround this issue. The issue has been fixed.
- The API docs incorrectly stated that `Cast` to the INT8 format is possible but this path is not supported. Use a `QuantizeLinear` node instead. This issue has been fixed in the API docs.
- When using refit on multi-head attention or if/while loops with explicit quantization, the refit process could have been slow due to the implementation's `memcpyDeviceToHost` for the Q/DQ scales. This issue has been fixed.
- There was an up to 9% performance regression for StableDiffusion VAE networks on A16 and A40 compared to TensorRT 9.2. The workaround was to disable the `kNATIVE_INSTANCENORM` flag in ONNX parser or add the `--pluginInstanceNorm` flag to `trtexec`. This issue has been fixed.

- There was a small chance that TensorRT would hang when running on H100 with the r550 CUDA driver when CUDA graphs were used. The workaround was to use the r535 CUDA driver instead or to avoid using CUDA graphs. This issue has been fixed.
- There was a known issue on H100 that may have led to a GPU hang when running TensorRT with high persistentCache usage. The workaround was to limit the usage to 40% of L2 cache size. This issue has been fixed.
- There was a known performance issue when running instance normalization layers on Arm Server Base System Architecture (SBSA). This issue has been fixed.
- There were some issues when running TensorRT-LLM with TensorRT 10.0 with the `StronglyTyped` mode enabled. The workaround was to disable `StronglyTyped` mode.
- Running sync/race check with newer Compute Sanitizer on L4T may have hit a hang issue. The workaround was to try an older version of Compute Sanitizer. This issue has been fixed.
- Hardware forward compatibility (HFC) was broken on L4T Concord for ViT, Swin-Transformers, and BERT networks in FP16 mode. The workaround was to only use FP32 mode on L4T Concord or turn off HFC. This issue has been fixed.
- LSTM networks could fail to build with timing cache enabled. This has been observed on one GPU platform and only when building with a cache that has pre-existing entries. Error signature would contain

```
Skipping tactic 0x0000000000000000 due to exception
 [autotuner.cpp:operator():1502]
        Internal bug. Please report with reproduction steps.
```

  The workaround was to disable the timing cache or start a fresh one. This issue has been fixed.

Known Issues

Functional

- The `tensorrt` Python metapackage does not pin the version for the Python module dependency `tensorrt-cu12`. For example, using `pip install tensorrt==10.0.1` will install `tensorrt-cu12==10.1.0` rather than `tensorrt-cu12==10.0.1` as expected. The workaround is to instead specify the package name including the CUDA version, such as `pip install tensorrt-cu12==10.0.1`. This issue will be fixed in TensorRT 10.2.
- The Python sample yolo_v3_onnx does not support Python 3.12. Support will be added in 10.2.
- If TensorRT 8.6 or 9.x was installed using the Python Package Index (PyPI) you will not be able to upgrade TensorRT to 10.x using PyPI. You must first uninstall TensorRT using `pip uninstall tensorrt tensorrt-libs tensorrt-bindings` and then reinstall TensorRT using "`pip install tensorrt`". This will remove the previous TensorRT version and install the latest TensorRT 10.x. This step is required because the suffix `-cuXX` was added to the Python package names, which prevents the upgrade from working properly.

- Allocated GPU memory during autotuning might not be freed correctly if allocation failed due to inadequate resources, causing build time memory usage to be larger than that of inference time.
- CUDA compute sanitizer may report racecheck hazards for some legacy kernels, however, related kernels do not have functional issues at runtime.
- The compute sanitizer `initcheck` tool may flag false positive `Uninitialized __global__ memory read` errors when running TensorRT applications on NVIDIA Hopper GPUs. These errors can be safely ignored and will be fixed in an upcoming CUDA release.
- Multihead attention fusion might not happen and affect performance if the number of heads is small.
- If a network has a tensor of type bool with an *implicitly* data-dependent shape, engine building will likely fail.
- There is an occurrence of use-after-free in NVRTC that has been fixed in CUDA 12.1. When using NVRTC from CUDA 12.0 together with the TensorRT static library, you may encounter a crash in certain scenarios. Linking with the NVRTC and PTXJIT compiler from CUDA 12.1 or newer will resolve this issue.
- There are known issues reported by the Valgrind memory leak check tool when detecting potential memory leaks from TensorRT applications. The recommendation to suppress the issues is to provide a Valgrind suppression file with the following contents when running the Valgrind memory leak check tool. Add the option `--keep-debuginfo=yes` to the Valgrind command line to suppress these errors.

```
{
  Memory leak errors with dlopen.
   Memcheck:Leak
   match-leak-kinds: definite
   ...
   fun:*dlopen*
   ...
}
{
    Memory leak errors with nvrtc
    Memcheck:Leak
    match-leak-kinds: definite
    fun:malloc
    obj:*libnvrtc.so*
    ...
}
```

- SM 7.5 and earlier devices may not have INT8 implementations for all layers with Q/DQ nodes. In this case, you will encounter a `could not find any implementation` error while building your engine. To resolve this, remove the Q/DQ nodes, which quantize the failing layers.
- Installing the `cuda-compat-11-4` package may interfere with CUDA enhanced compatibility and cause TensorRT to fail even when the driver is r465. The workaround is to remove the `cuda-compat-11-4` package or upgrade the driver to r470. *(not applicable for Jetson platforms)*

- For some networks, using a batch size of 4096 may cause accuracy degradation on DLA.
- For broadcasting elementwise layers running on DLA with GPU fallback enabled with one NxCxHxW input and one Nx1x1x1 input, there is a known accuracy issue if at least one of the inputs is consumed in `kDLA_LINEAR` format. It is recommended to explicitly set the input formats of such elementwise layers to different tensor formats.
- Exclusive padding with `kAVERAGE` pooling is not supported.
- The Valgrind tool found a memory leak on L4T with CUDA 12.4 due to a known driver issue. This is expected to be fixed in CUDA 12.6.
- Asynchronous CUDA calls are not supported in the user defined `processDebugTensor` function for the debug tensor feature due to a bug in Windows 10.
- For sample python/efficientdet and python/tensorflow_object_detection_api, the ONNX version needs to be manually downgraded in their respective `requirements.txt` file to 1.14 for the sample to function correctly.
- There is a known accuracy issue when the network contains two consecutive GEMV operations (that is, MatrixMultiply with `gemmM` or `gemmN == 1`). To workaround this issue, try padding the MatrixMultiply input to have dimensions greater than `1`.
- Engine building with weight streaming enabled will fail when the model size is larger than the free device memory size. This issue will be fixed in the next version.

Performance

- There is an up to 10% performance regression for ConvNext on NVIDIA Orin compared to TensorRT 9.3.
- There are known performance gaps between engines built with REFIT enabled and engines built with REFIT disabled.
- There is an up to 4x performance regression for networks containing `GridSample` ops compared to TensorRT 9.2.
- There are up to 60 MB engine size fluctuations for the BERT-Large INT8-QDQ model on Orin due to unstable tactic selection among tactics.
- There is an up to 16% performance regression for BasicUNet, DynUNet, and HighResNet in INT8 precision compared to TensorRT 9.3.
- There are performance gaps for StableDiffusion networks between Windows and Linux platforms.
- There is an up to 40 second increase in engine building for BART networks on NVIDIA Hopper GPUs.
- There is an up to 20 second increase in engine building for some large language models (LLMs) on NVIDIA Ampere GPUs.
- There is an up to 2.5x build time increase compared to TensorRT 9.0 for certain Bert like models due to additional tactics available for evaluation.

- There is an up to 13% performance drop for the CortanaASR model on NVIDIA Ampere GPUs compared to TensorRT 8.5.
- There is an up to 18% performance drop for the ShuffleNet model on A30/A40 compared to TensorRT 8.5.1.
- Convolution on a tensor with an *implicitly* data-dependent shape may run significantly slower than on other tensors of the same size. Refer to the Glossary for the definition of implicitly data-dependent shapes.
- For some Transformer models, including ViT, Swin-Transformer, and DETR, there is a performance drop in INT8 precision (including both explicit and implicit quantization) compared to FP16 precision.
- There is a known issue with DLA clocks that requires users to reboot the system after changing the `nvpmodel` power mode or otherwise experience a performance drop. Refer to the L4T board support package Release Notes for details.
- There is an up to 5% performance drop for networks using sparsity in FP16 precision.
- There is an up to 6% performance regression compared to TensorRT 8.5 on OpenRoadNet in FP16 precision on NVIDIA A10 GPUs.
- There is an up to 70% performance regression compared to TensorRT 8.6 on BERT networks in INT8 precision with FP16 disabled on L4 GPUs. To work around this, enable FP16 and disable INT8 in the builder config.
- In explicitly quantized networks, a group convolution that has a Q/DQ pair before but no Q/DQ pair after is expected to run with INT8-IN-FP32-OUT mixed precision. However, on NVIDIA Hopper, it may fall back to FP32-IN-FP32-OUT if the input channel count is small. This will be fixed in a future release.
- Weight streaming performance may decrease when you create execution contexts with multiple optimization profiles using external device memory and call `setDeviceMemory`/`setDeviceMemoryV2` before `setOptimizationProfileAsync`. This issue will be fixed in the next version.

1.3. TensorRT Release 10.1.0

These are the TensorRT 10.1.0 Release Notes and are applicable to x86 Linux and Windows users, Arm®-based CPU cores for Server Base System Architecture (SBSA) users on Linux, and JetPack users. This release includes several fixes from the previous TensorRT releases as well as the following additional changes.

For previously released TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Announcements

- The ONNX GraphSurgeon is no longer included inside the TensorRT package. Ensure you remove any previous Debian or RPM packages that may have been installed

by a previous release and instead install ONNX GraphSurgeon using `pip`. For more information, refer to onnx-graphsurgeon.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- When using the TensorRT ONNX parser, shape inputs can now be passed to custom ops supported by IPluginV3-based plugins. The indices of the inputs to be interpreted as shape inputs must be indicated by a node attribute named `tensorrt_plugin_shape_input_indices` containing a list of integers.
- Added a new sample `non_zero_plugin`, which is a Python version of the C++ sample `sampleNonZeroPlugin`.
- TensorRT Python bindings natively support accessing the `data` attribute of a `PluginField` of `PluginFieldType.INT64` and `PluginFieldType.UNKNOWN` as NumPy arrays. For example, the NumPy functions `tobytes()` and `frombuffer()` may be used during storage and retrieval to embed an arbitrary NumPy array in a `PluginFieldType.UNKNOWN`.
- Add new APIs for weight streaming including `setWeightStreamingBudgetV2`, `getWeightStreamingBudgetV2`, `getWeightStreamingAutomaticBudget`, and `getWeightStreamingScratchMemorySize`. Now, weight streaming supports CUDA graph and multiple contexts of an engine running in parallel.
- Added new APIs for ascertaining an `IExecutionContext` device memory size and setting it: `ICudaEngine::getDeviceMemorySizeV2` and `IExecutionContext::setDeviceMemorySizeV2`. The V1 APIs are deprecated.
- Added native TensorRT layer support for ONNX operator `isNaN`, and added TensorRT plugin support for ONNX operator `DeformConv`.
- Internal exceptions are now contained and won't leak through the parser API boundaries.
- During `IPluginV3` auto-tuning, it is guaranteed that `configurePlugin()` is called with the current input/output format combination being timed before `getValidTactics()` is called. Therefore, it is possible to advertise a different set of tactics per each input/output format combination.

Compatibility

- TensorRT 10.1.0 has been tested with the following:
  - TensorFlow 2.13.1
  - PyTorch >= 2.0 (refer to the `requirements.txt` file for each sample)
  - ONNX 1.16.0
- This TensorRT release supports CUDA®:
  - 12.4 update 1

  - 12.3 update 2
  - 12.2 update 1
  - 12.1 update 1
  - 12.0 update 1
  - 11.8
  - 11.7 update 1
  - 11.6 update 2
  - 11.5 update 2
  - 11.4 update 4
  - 11.3 update 1
  - 11.2 update 2
  - 11.1 update 3
  - 11.0 update 1
- This TensorRT release requires at least NVIDIA driver r450 on Linux or r452 on Windows as required by CUDA 11.0, which is the minimum CUDA version supported by this TensorRT release.

Limitations

- On QNX, networks that are segmented into a large number of DLA loadables may fail during inference.
- The DLA compiler is capable of removing identity transposes, but it cannot fuse multiple adjacent transpose layers into a single transpose layer (likewise for reshape). For example, given a TensorRT `IShuffleLayer` consisting of two non-trivial transposes and an identity reshapes in between. The shuffle layer is translated into two consecutive DLA transpose layers, unless you merge the transposes together manually in the model definition in advance.
- `nvinfer1::UnaryOperation::kROUND` or `nvinfer1::UnaryOperation::kSIGN` operations of `IUnaryLayer` are not supported in the implicit batch mode.
- For networks containing normalization layers, particularly if deploying with mixed precision, target the latest ONNX opset that contains the corresponding function ops, for example: opset 17 for LayerNormalization or opset 18 GroupNormalization. Numerical accuracy using function ops is superior to corresponding implementation with primitive ops for normalization layers.
- The `kREFIT` and `kREFIT_IDENTICAL` have performance regressions compared with non-refit engines where convolution layers are present within a branch or loop and the precision is FP16/INT8. This issue will be addressed in future releases.
- Weight streaming mainly supports GEMM-based networks like Transformers for now. Convolution-based networks may have only a few weights that can be streamed.

Deprecated API Lifetime

- APIs deprecated in TensorRT 10.1 will be retained until at least 5/2025.
- APIs deprecated in TensorRT 10.0 will be retained until at least 3/2025.
- APIs deprecated in TensorRT 9.3 will be retained until at least 1/2025.
- APIs deprecated in TensorRT 9.2 will be retained until at least 11/2024.
- APIs deprecated in TensorRT 9.1 will be retained until at least 10/2024.
- APIs deprecated in TensorRT 9.0 will be retained until at least 8/2024.

Refer to the API documentation (C++, Python) for how to update your code to remove the use of deprecated features.

Deprecated and Removed Features

The following features have been deprecated or removed in TensorRT 10.1.0.

- Deprecated NVIDIA Volta support (GPUs with compute capability 7.0) starting with TensorRT 10.0. Volta support may be removed after 9/2024.
- Version 1 of the ROIAlign plugin (`ROIAlign_TRT`), which implemented `IPluginV2DynamicExt`, is deprecated. It is superseded by version 2, which implements `IPluginV3`.
- The TensorRT standard plugin shared library (`libnvinfer_plugin.so` / `nvinfer_plugin.dll`) only exports `initLibNvInferPlugins`. No symbols in the `nvinfer1::plugin` namespace are exported anymore.
- Deprecated `IParser::supportsModel` and replaced this method with `IParser::supportsModelV2`, `IParser::getNbSubgraphs`, `IParser::isSubgraphSupported`, and `IParser::getSubgraphNodes`.
- Deprecated some weight streaming APIs including `setWeightStreamingBudget`, `getWeightStreamingBudget`, and `getMinimumWeightStreamingBudget`. Replaced by new versions of weight streaming APIs.
- Deprecated INT8 implicit quantization and calibrator APIs including `dynamicRangeIsSet`, `CalibrationAlgoType`, `IInt8Calibrator`, `IInt8EntropyCalibrator`, `IInt8EntropyCalibrator2`, `IInt8MinMaxCalibrator`, `IInt8Calibrator`, `setInt8Calibrator`, `getInt8Calibrator`, `setCalibrationProfile`, `getCalibrationProfile`, `setDynamicRange`, `getDynamicRangeMin`, `getDynamicRangeMax`, and `getTensorsWithDynamicRange`. They may not give the optimal performance and accuracy. As a workaround, use INT8 explicit quantization instead.

Fixed Issues

- The `sampleNonZeroPlugin` sample failed to build when cross compiling for L4T. The workaround was to continue building the other samples by modifying `samples/`

`Makefile` and removing the line containing `sampleNonZeroPlugin`. This issue has been fixed.

- The `sampleNonZeroPlugin` sample did not guarantee CUDA minor version compatibility. That is, if built against a newer CUDA Toolkit release, it may not function properly on older drivers, even within the same major CUDA release family. This issue has been fixed.
- `IPluginRegistry::deregisterLibrary()` did not work with plugin shared libraries with the defined entry point `getPluginCreators()`. `IPluginRegistry::loadLibrary()` was not impacted. The workaround was to deregister the plugins that were contained in such a library, manually query the library for `getPluginCreators()`, and invoke `IPluginRegistry::deregisterCreator()` for each creator retrieved. This issue has been fixed.
- On A30, some fused MHA (multi-head attention) performance was not optimized yet. This issue has been fixed.
- If building the TensorRT backend of ONNX runtime we are not able to use the prebuilt parser. This issue has been fixed.
- When using the Polygraphy `engine_from_network` API, if we enabled both refittable and `strip_plan` in the `create_config`, the final engine weights were not stripped. The workaround was, only include `strip_plan` in the `create_config`. This issue has been fixed.
- TensorRT did not support attention operations for tensors larger than `int32_t` maximum. Plugins could be used to workaround this issue. The issue has been fixed.
- The API docs incorrectly stated that `Cast` to the INT8 format is possible but this path is not supported. Use a `QuantizeLinear` node instead. This issue has been fixed in the API docs.
- When using refit on multi-head attention or if/while loops with explicit quantization, the refit process could have been slow due to the implementation's `memcpyDeviceToHost` for the Q/DQ scales. This issue has been fixed.
- There was an up to 9% performance regression for StableDiffusion VAE networks on A16 and A40 compared to TensorRT 9.2. The workaround was to disable the `kNATIVE_INSTANCENORM` flag in ONNX parser or add the `--pluginInstanceNorm` flag to `trtexec`. This issue has been fixed.
- There was a small chance that TensorRT would hang when running on H100 with the r550 CUDA driver when CUDA graphs were used. The workaround was to use the r535 CUDA driver instead or to avoid using CUDA graphs. This issue has been fixed.
- There was a known issue on H100 that may have led to a GPU hang when running TensorRT with high persistentCache usage. The workaround was to limit the usage to 40% of L2 cache size. This issue has been fixed.
- There was a known performance issue when running instance normalization layers on Arm Server Base System Architecture (SBSA). This issue has been fixed.

- There were some issues when running TensorRT-LLM with TensorRT 10.0 with the `StronglyTyped` mode enabled. The workaround was to disable `StronglyTyped` mode.
- Running sync/race check with newer Compute Sanitizer on L4T may have hit a hang issue. The workaround was to try an older version of Compute Sanitizer. This issue has been fixed.
- Hardware forward compatibility (HFC) was broken on L4T Concord for ViT, Swin-Transformers, and BERT networks in FP16 mode. The workaround was to only use FP32 mode on L4T Concord or turn off HFC. This issue has been fixed.
- LSTM networks could fail to build with timing cache enabled. This has been observed on one GPU platform and only when building with a cache that has pre-existing entries. Error signature would contain

```
Skipping tactic 0x0000000000000000 due to exception
 [autotuner.cpp:operator():1502]
        Internal bug. Please report with reproduction steps.
```

  The workaround was to disable the timing cache or start a fresh one. This issue has been fixed.

Known Issues

Functional

- The `tensorrt` Python metapackage does not pin the version for the Python module dependency `tensorrt-cu12`. For example, using `pip install tensorrt==10.0.1` will install `tensorrt-cu12==10.1.0` rather than `tensorrt-cu12==10.0.1` as expected. The workaround is to instead specify the package name including the CUDA version, such as `pip install tensorrt-cu12==10.0.1`. This issue will be fixed in TensorRT 10.2.
- The Python sample yolo_v3_onnx does not support Python 3.12. Support will be added in 10.2.
- If TensorRT 8.6 or 9.x was installed using the Python Package Index (PyPI) you will not be able to upgrade TensorRT to 10.x using PyPI. You must first uninstall TensorRT using `pip uninstall tensorrt tensorrt-libs tensorrt-bindings` and then reinstall TensorRT using "`pip install tensorrt`". This will remove the previous TensorRT version and install the latest TensorRT 10.x. This step is required because the suffix `-cuXX` was added to the Python package names, which prevents the upgrade from working properly.
- Allocated GPU memory during autotuning might not be freed correctly if allocation failed due to inadequate resources, causing build time memory usage to be larger than that of inference time.
- CUDA compute sanitizer may report racecheck hazards for some legacy kernels, however, related kernels do not have functional issues at runtime.
- The compute sanitizer `initcheck` tool may flag false positive `Uninitialized __global__ memory read` errors when running TensorRT applications on NVIDIA Hopper GPUs. These errors can be safely ignored and will be fixed in an upcoming CUDA release.

- Multihead attention fusion might not happen and affect performance if the number of heads is small.
- If a network has a tensor of type bool with an *implicitly* data-dependent shape, engine building will likely fail.
- There is an occurrence of use-after-free in NVRTC that has been fixed in CUDA 12.1. When using NVRTC from CUDA 12.0 together with the TensorRT static library, you may encounter a crash in certain scenarios. Linking with the NVRTC and PTXJIT compiler from CUDA 12.1 or newer will resolve this issue.
- There are known issues reported by the Valgrind memory leak check tool when detecting potential memory leaks from TensorRT applications. The recommendation to suppress the issues is to provide a Valgrind suppression file with the following contents when running the Valgrind memory leak check tool. Add the option `--keep-debuginfo=yes` to the Valgrind command line to suppress these errors.

```
{
  Memory leak errors with dlopen.
   Memcheck:Leak
   match-leak-kinds: definite
   ...
   fun:*dlopen*
   ...
}
{
    Memory leak errors with nvrtc
    Memcheck:Leak
    match-leak-kinds: definite
    fun:malloc
    obj:*libnvrtc.so*
    ...
}
```

- SM 7.5 and earlier devices may not have INT8 implementations for all layers with Q/DQ nodes. In this case, you will encounter a `could not find any implementation` error while building your engine. To resolve this, remove the Q/DQ nodes, which quantize the failing layers.
- Installing the `cuda-compat-11-4` package may interfere with CUDA enhanced compatibility and cause TensorRT to fail even when the driver is r465. The workaround is to remove the `cuda-compat-11-4` package or upgrade the driver to r470. *(not applicable for Jetson platforms)*
- For some networks, using a batch size of 4096 may cause accuracy degradation on DLA.
- For broadcasting elementwise layers running on DLA with GPU fallback enabled with one NxCxHxW input and one Nx1x1x1 input, there is a known accuracy issue if at least one of the inputs is consumed in `kDLA_LINEAR` format. It is recommended to explicitly set the input formats of such elementwise layers to different tensor formats.
- Exclusive padding with `kAVERAGE` pooling is not supported.
- The Valgrind tool found a memory leak on L4T with CUDA 12.4 due to a known driver issue. This is expected to be fixed in CUDA 12.6.

- Asynchronous CUDA calls are not supported in the user defined `processDebugTensor` function for the debug tensor feature due to a bug in Windows 10.
- For sample python/efficientdet and python/tensorflow_object_detection_api, the ONNX version needs to be manually downgraded in their respective `requirements.txt` file to 1.14 for the sample to function correctly.
- There is a known accuracy issue when the network contains two consecutive GEMV operations (that is, MatrixMultiply with `gemmM` or `gemmN == 1`). To workaround this issue, try padding the MatrixMultiply input to have dimensions greater than `1`.
- Engine building with weight streaming enabled will fail when the model size is larger than the free device memory size. This issue will be fixed in the next version.

Performance

- There is an up to 10% performance regression for ConvNext on NVIDIA Orin compared to TensorRT 9.3.
- There are known performance gaps between engines built with REFIT enabled and engines built with REFIT disabled.
- There is an up to 4x performance regression for networks containing `GridSample` ops compared to TensorRT 9.2.
- There are up to 60 MB engine size fluctuations for the BERT-Large INT8-QDQ model on Orin due to unstable tactic selection among tactics.
- There is an up to 16% performance regression for BasicUNet, DynUNet, and HighResNet in INT8 precision compared to TensorRT 9.3.
- There are performance gaps for StableDiffusion networks between Windows and Linux platforms.
- There is an up to 40 second increase in engine building for BART networks on NVIDIA Hopper GPUs.
- There is an up to 20 second increase in engine building for some large language models (LLMs) on NVIDIA Ampere GPUs.
- There is an up to 2.5x build time increase compared to TensorRT 9.0 for certain Bert like models due to additional tactics available for evaluation.
- There is an up to 13% performance drop for the CortanaASR model on NVIDIA Ampere GPUs compared to TensorRT 8.5.
- There is an up to 18% performance drop for the ShuffleNet model on A30/A40 compared to TensorRT 8.5.1.
- Convolution on a tensor with an *implicitly* data-dependent shape may run significantly slower than on other tensors of the same size. Refer to the Glossary for the definition of implicitly data-dependent shapes.
- For some Transformer models, including ViT, Swin-Transformer, and DETR, there is a performance drop in INT8 precision (including both explicit and implicit quantization) compared to FP16 precision.

- There is a known issue with DLA clocks that requires users to reboot the system after changing the `nvpmodel` power mode or otherwise experience a performance drop. Refer to the L4T board support package Release Notes for details.
- There is an up to 5% performance drop for networks using sparsity in FP16 precision.
- There is an up to 6% performance regression compared to TensorRT 8.5 on OpenRoadNet in FP16 precision on NVIDIA A10 GPUs.
- There is an up to 70% performance regression compared to TensorRT 8.6 on BERT networks in INT8 precision with FP16 disabled on L4 GPUs. To work around this, enable FP16 and disable INT8 in the builder config.
- In explicitly quantized networks, a group convolution that has a Q/DQ pair before but no Q/DQ pair after is expected to run with INT8-IN-FP32-OUT mixed precision. However, on NVIDIA Hopper, it may fall back to FP32-IN-FP32-OUT if the input channel count is small. This will be fixed in a future release.
- Weight streaming performance may decrease when you create execution contexts with multiple optimization profiles using external device memory and call `setDeviceMemory`/`setDeviceMemoryV2` before `setOptimizationProfileAsync`. This issue will be fixed in the next version.

1.4. TensorRT Release 10.0.1

These are the TensorRT 10.0.1 Release Notes and are applicable to x86 Linux and Windows users, Arm®-based CPU cores for Server Base System Architecture (SBSA) users on Linux, and JetPack users. This release includes several fixes from the previous TensorRT releases as well as the following additional changes.

For previously released TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Announcements

- For TensorRT 10.0.0 EA the minimum glibc version for the Linux x86 build was 2.28. This toolchain change was reverted for TensorRT 10.0.1 GA and will be compatible with glibc 2.17, which was the minimum glibc version supported by TensorRT 8.6.
- RedHat/CentOS 7.x are no longer officially supported starting with TensorRT 10.0.
- RedHat/Rocky Linux 9.x are supported starting with TensorRT 10.0.
- Support for Python 3.6 and 3.7 has been dropped starting with TensorRT 10.0.
- Python 3.12 support has been added starting with TensorRT 10.0.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- TensorRT 10 is supported by Nsight Deep Learning Designer 2024.1 (Early Access).

- The ONNX parser returns the list of all nodes that can be statically determined as unsupported when the call to `parse()` fails. The error reporting contains node name, node type, reason for failure, as well as the local function stack if the node is located in an ONNX local function. The number of these errors can be queried with the `getNbErrors()` function, and information about individual errors can be obtained from the `getError()` function.
- Debug Tensors: Added an API to mark tensors as debug tensors at build time. At runtime, each time the value of the tensor is written, a user-defined callback function is invoked with the value, type, and dimensions.
- Runtime Allocation: `createExecutionContext` now accepts an argument specifying the allocation strategy (`kSTATIC`, `kON_PROFILE_CHANGE`, and `kUSER_MANAGED`) of execution context device memory. For user-managed allocation, an additional API `updateDeviceMemorySizeForShapes` is added to query the required size based on actual input shapes.
- Added a new Python sample sample_weight_stripping to showcase building and refitting weight-stripped engines from ONNX models.
- The new `REFIT_IDENTICAL` flag instructs the TensorRT builder to optimize under the assumption that the engine will be refitted with weights identical to those provided at build time. Using this flag in conjunction with `kSTRIP_PLAN` minimizes plan size in deployment scenarios where, for example, the plan is being shipped alongside an ONNX model containing the weights.
- Weight Streaming: Added a new `kWEIGHT_STREAMING` flag to the builder and a set of new streaming budget APIs in the runtime to allow you to run strongly typed models larger than device memory on the device. For example, a strongly typed model with 32 GB of weights can run on a device with less than 32 GB of VRAM.
- The `tensorrt` Debian and RPM meta-packages now install the TensorRT Python binding packages `python3-libnvinfer`, `python3-libnvinfer-lean`, and `python3-libnvinfer-dispatch` as well. Previously, installing the `python3-libnvinfer-dev(el)` package was required as well to support both C++ and Python.
- V3 plugins: A new generation of TensorRT custom layers is now available with plugins implementing `IPluginV3`, which are to be accompanied by plugin creators implementing `IPluginCreatorV3One`. New features available with `IPluginV3` include data-dependent output shapes, shape tensor inputs, custom tactics, and timing caching.
- A key-value store has been added to the plugin registry which allows the registration and lookup of user-defined resources.
- The new `kTACTIC_SHARED_MEMORY` flag which allows control over the overall shared memory budget used for TensorRT backend CUDA kernels. This is useful in scenarios where TensorRT must share GPUs with other applications. By default, the value is set to device max capability.
- QAT transformer networks now work with refit.

- INT4 Weight Only Quantization (WoQ): Added support for weight compression using INT4 (Hopper only). WoQ performs extra copies that increase latency and will be further optimized in future TensorRT releases.
- Block Quantization: Added Block Quantization mode, allowing setting scales in high granularity (supported by INT4 WoQ only).

Breaking API Changes

- ATTENTION: TensorRT 10.0 GA broke ABI compatibility relative to TensorRT 10.0 EA on Windows by adding the TensorRT major version to the DLL filename. TensorRT 10.0 EA and prior TensorRT releases have historically named the DLL file `nvinfer.dll`, while 10.0 GA renamed the DLL file `nvinfer_10.dll`. This same naming pattern was also applied to the other TensorRT DLL files in the zip package. We strive not to break backward compatibility between releases with the same major version, but this change will allow applications to link against different TensorRT major versions at the same time.
- ATTENTION: In TensorRT 9.0, due to the introduction of INT64 as a supported data type, ONNX models with INT64 I/O require INT64 bindings. Note that prior to this release, such models required INT32 bindings.
- Release 10.0 GA enforces the restriction that `NvInferRuntimeBase.h` should not be directly included. The restriction was merely documented when 8.6 introduced the header.
- In TensorRT 9.0, we removed `ICaffeParser`, `IUffParser`, and related classes and functions. The following APIs are removed:
  - `nvcaffeparser1::IBlobNameToTensor`
  - `nvcaffeparser1::IBinaryProtoBlob`
  - `nvcaffeparser1::IPluginFactoryV2`
  - `nvcaffeparser1::ICaffeParser`
  - `nvcaffeparser1::createCaffeParser`
  - `nvcaffeparser1::shutdownProtobufLibrary`
  - `createNvCaffeParser_INTERNAL`
  - `nvinfer1::utils::reshapeWeights`
  - `nvinfer1::utils::reorderSubBuffers`
  - `nvinfer1::utils::ransposeSubBuffers`
  - `nvuffparser::UffInputOrder`
  - `nvuffparser::FieldType`
  - `nvuffparser::FieldMap`
  - `nvuffparser::FieldCollection`
  - `nvuffparser::IUffParser`

  - `nvuffparser::createUffParser`
  - `nvuffparser::shutdownProtobufLibrary`
  - `createNvUffParser_INTERNAL`
- With removal of `ICaffeParser` and `IUffParsers`, the `libnvparsers` library is removed.
- `uff`, `graphsurgeon`, and related networks are removed from TensorRT packages.
- `TacticSource::kCUDNN` and `TacticSource::kCUBLAS` are disabled by default. The `cudnnContext*` and `cublasContext*` parameters of the `nvinfer1::IPluginV2Ext::attachToContext` function are set to `nullptrs` when the corresponding `TacticSource` flags are unset.
- `IPluginCreatorInterface` has been added as a base class to `IPluginCreator`.
- Overloads have been added to the methods `IPluginRegistry::deregisterCreator` and `IPluginRegistry::registerCreator` that take in `IPluginCreatorInterface` references.

Compatibility

- TensorRT 10.0.1 has been tested with the following:
  - TensorFlow 2.12.0
  - PyTorch >= 2.0 (refer to the `requirements.txt` file for each sample)
  - ONNX 1.15.0
- This TensorRT release supports CUDA®:
  - 12.4 update 1
  - 12.3 update 2
  - 12.2 update 1
  - 12.1 update 1
  - 12.0 update 1
  - 11.8
  - 11.7 update 1
  - 11.6 update 2
  - 11.5 update 2
  - 11.4 update 4
  - 11.3 update 1
  - 11.2 update 2
  - 11.1 update 3
  - 11.0 update 1

- This TensorRT release requires at least NVIDIA driver r450 on Linux or r452 on Windows as required by CUDA 11.0, which is the minimum CUDA version supported by this TensorRT release.

Limitations

- There are two modes of DLA softmax where the mode is chosen automatically based on the shape of the input tensor, where:
  - the first mode triggers when all nonbatch, non-axis dimensions are 1, and
  - the second mode triggers in other cases if valid.

  The second of the two modes is supported only for DLA 3.9.0 and later. It involves approximations that may result in errors of a small degree. Also, batch size greater than 1 is supported only for DLA 3.9.0 and later. Refer to DLA Supported Layers for more information.
- On QNX, networks that are segmented into a large number of DLA loadables may fail during inference.
- The DLA compiler is capable of removing identity transposes, but it cannot fuse multiple adjacent transpose layers into a single transpose layer (likewise for reshape). For example, given a TensorRT `IShuffleLayer` consisting of two non-trivial transposes and an identity reshapes in between. The shuffle layer is translated into two consecutive DLA transpose layers, unless you merge the transposes together manually in the model definition in advance.
- In explicitly quantized networks, a group convolution that has a Q/DQ pair before but no Q/DQ pair after is expected to run with INT8-IN-FP32-OUT mixed precision. However, on NVIDIA Hopper™ it may fall back to FP32-IN-FP32-OUT if the input channel count is small. This will be fixed in a future release.
- `nvinfer1::UnaryOperation::kROUND` or `nvinfer1::UnaryOperation::kSIGN` operations of `IUnaryLayer` are not supported in the implicit batch mode.
- For networks containing normalization layers, particularly if deploying with mixed precision, target the latest ONNX opset that contains the corresponding function ops, for example: opset 17 for LayerNormalization or opset 18 GroupNormalization. Numerical accuracy using function ops is superior to corresponding implementation with primitive ops for normalization layers.
- `QuantizeLayer` and `DequantizeLayer` only support FP32 scale and data, even when using ONNX opset 19. If the input is not FP32, you must add a `Cast` to FP32 on the input to `QuantizeLayer`, and a `Cast` from FP32 at the output of `DequantizeLayer`.
- `EngineInspector::getLayerInformation` may return incomplete JSON data for some engines produced by TensorRT 9.0. When this happens, TensorRT Engine Explorer cannot be used to analyze the engine or generate a graph of the engine layers.

- The `kREFIT` and `kREFIT_IDENTICAL` have performance regressions where convolution layers are present within a branch or loop and the precision is FP16/INT8. This issue will be addressed in future releases.
- The new `kTACTIC_SHARED_MEMORY` flag cannot restrict shared memory usage for depthwise convolution, depth separate convolution, and certain corner case conv activation fused kernels. You need to run Nsight to verify the shared memory usage of the result engine. This issue will be addressed in a future release.
- Shape tensor inputs will not be added to TensorRT plugins implementing `IPluginV3` by the TensorRT ONNX parser. All inputs will be passed as regular device inputs. This is in contrast to the `addPluginV3` API which allows the specification of shape tensor inputs to be passed to the plugin.
- Weight streaming currently does not work with CUDA Graph.
- Multiple contexts for one engine with Weight Streaming enabled cannot run parallel on devices and will be serialized automatically.
- Weight streaming mainly supports GEMM-based networks like Transformers for now. Convolution-based networks may have only a few weights that can be streamed.
- `IPluginRegistry`'s `loadLibrary()` and `deregisterLibrary()` functionality is not supported for plugin shared libraries containing V3 plugin creators (`IPluginCreatorV3One`). This limitation will be removed in a future release.

Deprecated API Lifetime

- APIs deprecated in TensorRT 10.0 will be retained until at least 3/2025.
- APIs deprecated in TensorRT 9.3 will be retained until at least 1/2025.
- APIs deprecated in TensorRT 9.2 will be retained until at least 11/2024.
- APIs deprecated in TensorRT 9.1 will be retained until at least 10/2024.
- APIs deprecated in TensorRT 9.0 will be retained until at least 8/2024.

Refer to the API documentation (C++, Python) for how to update your code to remove the use of deprecated features.

Deprecated and Removed Features

The following features have been deprecated or removed in TensorRT 10.0.1.

- Deprecated NVIDIA Volta support (GPUs with compute capability 7.0). Volta support will be removed starting with TensorRT 10.2.
- Deprecated the `kWEIGHTLESS` builder flag. Superseded by the `kSTRIP_PLAN` builder flag. `kSTRIP_PLAN` works with either the `kREFIT` flag or the new `kREFIT_IDENTICAL` flag, defaulting to the latter if neither is set.
- In 10.0, we removed deprecated APIs compared to 9.3 and earlier releases. These removed APIs were deprecated before March 2023. We ensured that Version Compatibility is expected between 8.6, 9.x, and 10.0 versions. Note that version

compatibility is not supported for implicit batch mode, which was removed in 10.0. If you are unfamiliar with these changes, refer to our sample code for clarification. In light of the changes to the API in TensorRT 10.0, we've prepared an API Migration Guide to highlight the API modifications.

- We removed implicit batch support and worked with networks as they always have an explicit batch.
- Deprecated `TacticSource::kCUDNN` and `TacticSource::kCUBLAS` flags.
- Deprecated `IPluginV2DynamicExt`, implement `IPluginV3` instead. Refer to the Migrating V2 Plugins to IPluginV3 for how existing `IPluginV2DynamicExt` plugins can be migrated to `IPluginV3`.
- `IPluginCreator::getTensorRTVersion()` has been removed.
- Deprecated `IPluginV2IOExt`; implement `IPluginV3` instead.
- Deprecated `IPluginCreator`. There is no alternative factory class for `IPluginV2`-derivative plugin base classes, as they are all deprecated as well. Implement `IPluginV3` and its corresponding factory class `IPluginV3CreatorOne`.
- Deprecated the following APIs in `IPluginRegistry`:
  - `IPluginRegistry::registerCreator(IPluginCreator&)`. Use its overload `IPluginRegistry::registerCreator(IPluginCreatorInterface&)` instead.
  - `IPluginRegistry::deregisterCreator(IPluginCreator const&)`. Use its overload `IPluginRegistry::deregisterCreator(IPluginCreatorInterface const&)` instead.
  - `IPluginRegistry::getPluginCreator`. Use `IPluginRegistry::getCreator` instead.
  - `IPluginRegistry::getPluginCreatorList`. Use `IPluginRegistry::getAllCreators` instead.

Fixed Issues

- The `nvinfer_plugin.lib` library within the Windows package was incorrectly distributed as a static linking library starting with TensorRT 9.0. TensorRT 10.0 reverts this library to a dynamic linking library matching the behavior of TensorRT 8.6.
- There was an up to 9% performance drop for BERT networks with `gelu_erf` activation in BF16 precision compared to TensorRT 9.1 on NVIDIA Ampere GPUs.
- There was an up to 11% performance drop for ViT networks in TF32 precision compared to TensorRT 9.0 on NVIDIA Ampere GPUs.
- There was an up to 23% performance regression compared to TensorRT 8.5 on Temporal Fusion Transformers in FP32 precision on NVIDIA Turing and NVIDIA Ampere GPUs.
- A higher builder optimization level did not always give a better performance when compared to a lower builder optimization level; which could happen on all platforms

and up to 27%. The workaround was to build an engine using a lower builder optimization level.

- If an ONNX model contained a `Range` operator and its `limit` input was a data-dependent tensor, engine building would likely fail.
- There was an up to 15% performance regression for SegResNet and StableDiffusion VAE in FP16 precision compared to TensorRT 9.3.
- TensorRT did not clean temporary DLL files automatically on Windows when running in `vc` mode. The TensorRT library was internally holding open file references when the application finished.
- TensorRT may have crashed when building transformer based networks on Windows 10 and H100.
- There was a performance drop on QDQ-Gemm pattern on RTX-Titan in weightless mode.
- There was a known issue with the compute sanitizer in CUDA Toolkit 12.3 that might cause the target application to crash. This has been fixed in CUDA Toolkit 12.4.
- Indexing for layer information (`--dumpLayer`) and its profiling information (`--dumpProfile`) has been added; the layer names reported by `IEngineInspector` now match the layer names reported by `IProfiler`.
- Multihead attention fusion now works with refit enabled.
- There was an up to 144 MB peak GPU memory usage increase compared to TensorRT 8.6 when building engines for ResNet-50 in INT8 precision on the L4T Orin platform.
- Hardware compatible engines built with CUDA versions older than 11.5 will no longer crash during inference when run on a GPU with a compute capability lower than that of the GPU where the engine was built.
- Some networks fail at the engine building phase on Windows and H100, but can execute on Linux. The root cause is a builder issue, where fusion compilation fails.
- Using FP16 scales for Q/DQ ops may have resulted in numerical overflow. The workaround was to use FP32 scales for Q/DQ ops instead. This issue has been fixed.
- UNets with tensors containing >2^31 elements may have failed during the engine building step.
- Running TensorRT-LLM with TensorRT 10.0 with INT8 kv-cache would result in engine build failure due to insufficient custom scales. The workaround was to enable `StronglyTyped` mode. This issue has been fixed.
- There were up to 21% peak GPU memory usage fluctuations when building the engine for the same network back to back due to different tactics being selected.
- The ONNX Parser Refitter could not refit weights defined in nested ONNX structures such as `If`, `Loop`, or `Scan` operations. This issue has been fixed.
- When the `_gemm_mha_v2` operation was used, the outputs mismatched the output of PyTorch or the CPU executor (onnxRT). This problem showed up only when building

engines with FP16 precision, as `_gemm_mha_v2` has an implementation only for FP16. This issue has been fixed.

- Compute Sanitizer from CUDA Toolkit 12.0/12.1 may report a false alarm about invalid memory access in `generatedNativePointwise` kernels. This issue was fixed in CUDA Toolkit 12.2.
- There were up to 14% context memory usage fluctuations compared to TensorRT 9.1 when building the engine for 3DUnet networks due to different tactics being selected. This issue has been fixed.

Known Issues

Functional

- When using the polygraphy `engine_from_network` API, if we enable both refittable and `strip_plan` in the `create_config`, the final engine weights are not stripped. To workaround this, only include `strip_plan` in the `create_config`.
- TensorRT does not support attention operations for tensors larger than `int32_t` maximum. Plugins can be used to workaround this issue. The issue will be fixed in a future release.
- The API docs incorrectly state that `Cast` to the INT8 format is possible but this path is not supported. Use a `QuantizeLinear` node instead.
- Allocated GPU memory during autotuning might not be freed correctly if allocation failed due to inadequate resources, causing build time memory usage to be larger than that of inference time.
- When using refit on multi-head attention or if/while loops with explicit quantization, the refit process might be slow due to the implementation's `memcpyDeviceToHost` for the Q/DQ scales. This issue will be addressed in a future release.
- There are some issues when running TensorRT-LLM with TensorRT 10.0 with the `StronglyTyped` mode enabled. This can be worked around by disabling the `StronglyTyped` mode.
- CUDA compute sanitizer may report racecheck hazards for some legacy kernels, however, related kernels do not have functional issues at runtime.
- The compute sanitizer `initcheck` tool may flag false positive `Uninitialized __global__ memory read` errors when running TensorRT applications on NVIDIA Hopper GPUs. These errors can be safely ignored and will be fixed in an upcoming CUDA release.
- Multihead attention fusion might not happen and affect performance if the number of heads is small.
- Hardware forward compatibility (HFC) is broken on L4T Concord for ViT, Swin-Transformers, and BERT networks in FP16 mode. A workaround is to only use FP32 mode on L4T Concord or turn off HFC.

- If a network has a tensor of type bool with an *implicitly* data-dependent shape, engine building will likely fail.
- There is an occurrence of use-after-free in NVRTC that has been fixed in CUDA 12.1. When using NVRTC from CUDA 12.0 together with the TensorRT static library, you may encounter a crash in certain scenarios. Linking with the NVRTC and PTXJIT compiler from CUDA 12.1 or newer will resolve this issue.
- There are known issues reported by the Valgrind memory leak check tool when detecting potential memory leaks from TensorRT applications. The recommendation to suppress the issues is to provide a Valgrind suppression file with the following contents when running the Valgrind memory leak check tool. Add the option `--keep-debuginfo=yes` to the Valgrind command line to suppress these errors.

```
{
  Memory leak errors with dlopen.
   Memcheck:Leak
   match-leak-kinds: definite
   ...
   fun:*dlopen*
   ...
}
{
    Memory leak errors with nvrtc
    Memcheck:Leak
    match-leak-kinds: definite
    fun:malloc
    obj:*libnvrtc.so*
    ...
}
```

- SM 7.5 and earlier devices may not have INT8 implementations for all layers with Q/DQ nodes. In this case, you will encounter a `could not find any implementation` error while building your engine. To resolve this, remove the Q/DQ nodes, which quantize the failing layers.
- Installing the `cuda-compat-11-4` package may interfere with CUDA enhanced compatibility and cause TensorRT to fail even when the driver is r465. The workaround is to remove the `cuda-compat-11-4` package or upgrade the driver to r470. *(not applicable for Jetson platforms)*
- For some networks, using a batch size of 4096 may cause accuracy degradation on DLA.
- For broadcasting elementwise layers running on DLA with GPU fallback enabled with one NxCxHxW input and one Nx1x1x1 input, there is a known accuracy issue if at least one of the inputs is consumed in `kDLA_LINEAR` format. It is recommended to explicitly set the input formats of such elementwise layers to different tensor formats.
- Exclusive padding with `kAVERAGE` pooling is not supported.
- Running sync/race check with newer Compute Sanitizer on L4T may hit a hang issue. The workaround is to try an older version of Compute Sanitizer.
- The Valgrind tool found a memory leak on L4T with CUDA 12.4 due to a known driver issue. This is expected to be fixed in CUDA 12.6.

- Asynchronous CUDA calls are not supported in the user defined `processDebugTensor` function for the debug tensor feature due to a bug in Windows 10.
- The sample `sampleNonZeroPlugin` fails to build when cross compiling for L4T. You can workaround this issue and continue building the other samples by modifying `samples/Makefile` and removing the line containing `sampleNonZeroPlugin`. This issue will be fixed in the next release.
- The sample `sampleNonZeroPlugin` does not guarantee CUDA minor version compatibility. That is, if built against a newer CUDA Toolkit release, it may not function properly on older drivers, even within the same major CUDA release family.
- LSTM networks may fail to build with timing cache enabled. This has been observed on one GPU platform and only when building with a cache that has pre-existing entries. Error signature will contain

```
Skipping tactic 0x0000000000000000 due to exception
 [autotuner.cpp:operator():1502]
        Internal bug. Please report with reproduction steps.
```

  You can work around this issue by disabling the timing cache or starting a fresh one.
- `IPluginRegistry::deregisterLibrary()` will not work with plugin shared libraries with the defined entry point `getPluginCreators()`. `IPluginRegistry::loadLibrary()` is not impacted. To deregister plugins contained in such a library, manually query the library for `getPluginCreators()`, and invoke `IPluginRegistry::deregisterCreator()` for each creator retrieved.

Performance

- There is an up to 9% performance regression for StableDiffusion VAE networks on A16 and A40 compared to TensorRT 9.2. This can be worked around by disabling the `kNATIVE_INSTANCENORM` flag in ONNX parser or adding the `--pluginInstanceNorm` flag to `trtexec`.
- There is an up to 4x performance regression for networks containing `GridSample` ops compared to TensorRT 9.2.
- There are up to 60 MB engine size fluctuations for the BERT-Large INT8-QDQ model on Orin due to unstable tactic selection among tactics.
- There is a small chance that TensorRT will hang when running on H100 with the r550 CUDA driver when CUDA graphs are used. A workaround is to use the r535 CUDA driver instead or to avoid using CUDA graphs.
- There is an up to 16% performance regression for BasicUNet, DynUNet, and HighResNet in INT8 and FP16 precision compared to TensorRT 9.3.
- There are performance gaps for StableDiffusion networks between Windows and Linux platforms.
- On A30, some fused MHA (multi-head attention) performance is not optimized yet. This will be improved upon in future TensorRT versions.
- There is an up to 40 second increase in engine building for BART networks on NVIDIA Hopper GPUs.

- There is an up to 20 second increase in engine building for some large language models (LLMs) on NVIDIA Ampere GPUs.
- There is an up to 2.5x build time increase compared to TensorRT 9.0 for certain Bert like models due to additional tactics available for evaluation.
- There is an up to 13% performance drop for the CortanaASR model on NVIDIA Ampere GPUs compared to TensorRT 8.5.
- There is an up to 18% performance drop for the ShuffleNet model on A30/A40 compared to TensorRT 8.5.1.
- Convolution on a tensor with an *implicitly* data-dependent shape may run significantly slower than on other tensors of the same size. Refer to the Glossary for the definition of implicitly data-dependent shapes.
- For some Transformer models, including ViT, Swin-Transformer, and DETR, there is a performance drop in INT8 precision (including both explicit and implicit quantization) compared to FP16 precision.
- There is a known issue on H100 that may lead to GPU hang when running TensorRT with high persistentCache usage. Limit the usage to 40% of L2 cache size as a workaround.
- There is a known performance issue when running instance normalization layers on Arm Server Base System Architecture (SBSA).
- There is a known issue with DLA clocks that requires users to reboot the system after changing the `nvpmodel` power mode or otherwise experience a performance drop. Refer to the L4T board support package Release Notes for details.
- There is an up to 5% performance drop for networks using sparsity in FP16 precision.
- H100 performance for some LSTMs in FP16 precision is not fully optimized. This will be improved in future TensorRT versions.
- There is an up to 6% performance regression compared to TensorRT 8.5 on OpenRoadNet in FP16 precision on NVIDIA A10 GPUs.
- There is an up to 70% performance regression compared to TensorRT 8.6 on BERT networks in INT8 precision with FP16 disabled on L4 GPUs. To work around this, enable FP16 and disable INT8 in the builder config.

1.5. TensorRT Release 10.0.0 Early Access (EA)

These are the TensorRT 10.0.0 Early Access (EA) Release Notes and are applicable to x86 Linux and Windows users, Arm®-based CPU cores for Server Base System Architecture (SBSA) users on Linux, and JetPack users. This release includes several fixes from the previous TensorRT releases as well as the following additional changes. Continue to use TensorRT 8.6.1 for production use.

For previously released TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Announcements

- For TensorRT 10.0.0 EA the minimum glibc version for the Linux x86 build is 2.28. TensorRT 10.0.0 EA is expected to be compatible with RedHat 8.x (and derivatives) and newer RedHat distributions. TensorRT 10.0.0 EA is expected to also be compatible with Ubuntu 20.04 and newer Ubuntu distributions. This toolchain change will be reverted for TensorRT 10.0 GA and will be compatible with glibc 2.17, which was the minimum glibc version supported by TensorRT 8.6.
- RedHat/CentOS 7.x are no longer officially supported starting with TensorRT 10.0.
- RedHat/Rocky Linux 9.x are supported starting with TensorRT 10.0.
- Support for Python 3.6 and 3.7 has been dropped starting with TensorRT 10.0.
- Python 3.12 support has been added starting with TensorRT 10.0.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- TensorRT 10 is supported by Nsight Deep Learning Designer 2024.1 (Early Access).
- INT4 Weight Only Quantization: Added support for weight compression using INT4 (hardware agnostic).
- Block Quantization: Added Block Quantization mode, allowing setting scales in high granularity.
- The ONNX parser returns the list of all nodes that can be statically determined as unsupported when the call to `parse()` fails. The error reporting contains node name, node type, reason for failure, as well as the local function stack if the node is located in an ONNX local function. The number of these errors can be queried with the `getNbErrors()` function, and information about individual errors can be obtained from the `getError()` function.
- Debug Tensors: Added an API to mark tensors as debug tensors at build time. At runtime, each time the value of the tensor is written, a user-defined callback function is invoked with the value, type, and dimensions.
- Runtime Allocation: `createExecutionContext` now accepts an argument specifying the allocation strategy (`kSTATIC`, `kON_PROFILE_CHANGE`, and `kUSER_MANAGED`) of execution context device memory. For user-managed allocation, an additional API `updateDeviceMemorySizeForShapes` is added to query the required size based on actual input shapes.
- Added a new Python sample sample_weight_stripping to showcase building and refitting weight-stripped engines from ONNX models.
- The new `REFIT_IDENTICAL` flag instructs the TensorRT builder to optimize under the assumption that the engine will be refitted with weights identical to those provided

at build time. Using this flag in conjunction with `kSTRIP_PLAN` minimizes plan size in deployment scenarios where, for example, the plan is being shipped alongside an ONNX model containing the weights.

- Weight Streaming: Added a new `kWEIGHT_STREAMING` flag to the builder and a set of new streaming budget APIs in the runtime to allow you to run strongly typed models larger than device memory on the device. For example, a strongly typed model with 32 GB of weights can run on a device with less than 32 GB of VRAM.
- The `tensorrt` Debian and RPM meta-packages now install the TensorRT Python binding packages `python3-libnvinfer`, `python3-libnvinfer-lean`, and `python3-libnvinfer-dispatch` as well. Previously, installing the `python3-libnvinfer-dev(el)` package was required as well to support both C++ and Python.
- V3 plugins: A new generation of TensorRT custom layers is now available with plugins implementing `IPluginV3`, which are to be accompanied by plugin creators implementing `IPluginCreatorV3One`. New features available with `IPluginV3` include data-dependent output shapes, shape tensor inputs, custom tactics, and timing caching.
- A key-value store has been added to the plugin registry which allows the registration and lookup of user-defined resources.
- The new `kTACTIC_SHARED_MEMORY` flag which allows control over the overall shared memory budget used for TensorRT backend CUDA kernels. This is useful in scenarios where TensorRT must share GPUs with other applications. By default, the value is set to device max capability.

Breaking API Changes

- **!** ATTENTION: TensorRT 10.0 GA will break ABI compatibility relative to TensorRT 10.0 EA on Windows by adding the TensorRT major version to the DLL filename. TensorRT 10.0 EA and prior TensorRT releases have historically named the DLL file `nvinfer.dll`, while 10.0 GA will rename the DLL file `nvinfer_10.dll`. This same naming pattern will also apply to the other TensorRT DLL files in the zip package. We strive not to break backward compatibility between releases with the same major version, but this change will allow applications to link against different TensorRT major versions at the same time.
- **!** ATTENTION: In TensorRT 9.0, due to the introduction of INT64 as a supported data type, ONNX models with INT64 I/O require INT64 bindings. Note that prior to this release, such models required INT32 bindings.
- In 10.0, we removed deprecated APIs compared to 9.3 and earlier releases. These removed APIs were deprecated before March 2023. We ensured that Version Compatibility is expected between 8.6, 9.x, and 10.0 versions. If you are unfamiliar with these changes, refer to our sample code for clarification. In light of the changes

to the API in TensorRT 10.0, we've prepared an API Migration Guide to highlight the API modifications.

- In TensorRT 9.0, we removed `ICaffeParser`, `IUffParser`, and related classes and functions. The following APIs are removed:
  - `nvcaffeparser1::IBlobNameToTensor`
  - `nvcaffeparser1::IBinaryProtoBlob`
  - `nvcaffeparser1::IPluginFactoryV2`
  - `nvcaffeparser1::ICaffeParser`
  - `nvcaffeparser1::createCaffeParser`
  - `nvcaffeparser1::shutdownProtobufLibrary`
  - `createNvCaffeParser_INTERNAL`
  - `nvinfer1::utils::reshapeWeights`
  - `nvinfer1::utils::reorderSubBuffers`
  - `nvinfer1::utils::ransposeSubBuffers`
  - `nvuffparser::UffInputOrder`
  - `nvuffparser::FieldType`
  - `nvuffparser::FieldMap`
  - `nvuffparser::FieldCollection`
  - `nvuffparser::IUffParser`
  - `nvuffparser::createUffParser`
  - `nvuffparser::shutdownProtobufLibrary`
  - `createNvUffParser_INTERNAL`
- With removal of `ICaffeParser` and `IUffParsers`, the `libnvparsers` library is removed.
- `uff`, `graphsurgeon`, and related networks are removed from TensorRT packages.
- `TacticSource::kCUDNN` and `TacticSource::kCUBLAS` are disabled by default. The `cudnnContext*` and `cublasContext*` parameters of the `nvinfer1::IPluginV2Ext::attachToContext` function are set to `nullptrs` when the corresponding `TacticSource` flags are unset.
- `IPluginCreatorInterface` has been added as a base class to `IPluginCreator`.
- Overloads have been added to the methods `IPluginRegistry::deregisterCreator` and `IPluginRegistry::registerCreator` that take in `IPluginCreatorInterface` references.

Deprecated API Lifetime

- APIs deprecated in TensorRT 10.0 will be retained until at least 3/2025.
- APIs deprecated in TensorRT 9.3 will be retained until at least 1/2025.

- APIs deprecated in TensorRT 9.2 will be retained until at least 11/2024.
- APIs deprecated in TensorRT 9.1 will be retained until at least 10/2024.
- APIs deprecated in TensorRT 9.0 will be retained until at least 8/2024.

Refer to the API documentation (C++, Python) for how to update your code to remove the use of deprecated features.

Compatibility

- TensorRT 10.0.0 has been tested with the following:
  - TensorFlow 2.12.0
  - PyTorch >= 2.0 (refer to the `requirements.txt` file for each sample)
  - ONNX 1.15.0
- This TensorRT release supports CUDA®:
  - 12.4
  - 12.3 update 2
  - 12.2 update 1
  - 12.1 update 1
  - 12.0 update 1
  - 11.8
  - 11.7 update 1
  - 11.6 update 2
  - 11.5 update 2
  - 11.4 update 4
  - 11.3 update 1
  - 11.2 update 2
  - 11.1 update 3
  - 11.0 update 1
- This TensorRT release requires at least NVIDIA driver r450 on Linux or r452 on Windows as required by CUDA 11.0, which is the minimum CUDA version supported by this TensorRT release.

Limitations

- There are two modes of DLA softmax where the mode is chosen automatically based on the shape of the input tensor, where:
  - the first mode triggers when all nonbatch, non-axis dimensions are 1, and
  - the second mode triggers in other cases if valid.

The second of the two modes is supported only for DLA 3.9.0 and later. It involves approximations that may result in errors of a small degree. Also, batch size greater than 1 is supported only for DLA 3.9.0 and later. Refer to DLA Supported Layers for more information.

- On QNX, networks that are segmented into a large number of DLA loadables may fail during inference.
- The DLA compiler is capable of removing identity transposes, but it cannot fuse multiple adjacent transpose layers into a single transpose layer (likewise for reshape). For example, given a TensorRT `IShuffleLayer` consisting of two non-trivial transposes and an identity reshapes in between. The shuffle layer is translated into two consecutive DLA transpose layers, unless you merge the transposes together manually in the model definition in advance.
- In explicitly quantized networks, a group convolution that has a Q/DQ pair before but no Q/DQ pair after is expected to run with INT8-IN-FP32-OUT mixed precision. However, on NVIDIA Hopper™ it may fall back to FP32-IN-FP32-OUT if the input channel count is small. This will be fixed in a future release.
- `nvinfer1::UnaryOperation::kROUND` or `nvinfer1::UnaryOperation::kSIGN` operations of `IUnaryLayer` are not supported in the implicit batch mode.
- For networks containing normalization layers, particularly if deploying with mixed precision, target the latest ONNX opset that contains the corresponding function ops, for example: opset 17 for LayerNormalization or opset 18 GroupNormalization. Numerical accuracy using function ops is superior to corresponding implementation with primitive ops for normalization layers.
- `QuantizeLayer` and `DequantizeLayer` only support FP32 scale and data, even when using ONNX opset 19. If the input is not FP32, you must add a `Cast` to FP32 on the input to `QuantizeLayer`, and a `Cast` from FP32 at the output of `DequantizeLayer`.
- `EngineInspector::getLayerInformation` may return incomplete JSON data for some engines produced by TensorRT 9.0. When this happens, TensorRT Engine Explorer cannot be used to analyze the engine or generate a graph of the engine layers.
- The `kREFIT` and `kREFIT_IDENTICAL` have performance regressions where convolution layers are present within a branch or loop and the precision is FP16/INT8. This issue will be addressed in future releases.
- The new `kTACTIC_SHARED_MEMORY` flag cannot restrict shared memory usage for depthwise convolution, depth separate convolution, and certain corner case conv activation fused kernels. You need to run Nsight to verify the shared memory usage of the result engine. This issue will be addressed in a future release.
- Shape tensor inputs will not be added to TensorRT plugins implementing `IPluginV3` by the TensorRT ONNX parser. All inputs will be passed as regular device inputs. This is in contrast to the `addPluginV3` API which allows the specification of shape tensor inputs to be passed to the plugin.

- Weight streaming currently does not work with CUDA Graph.
- Multiple contexts for one engine with Weight Streaming enabled cannot run parallel on devices and will be serialized automatically.
- Weight streaming mainly supports GEMM-based networks like Transformers for now. Convolution-based networks may have only a few weights that can be streamed.

Deprecated and Removed Features

The following features have been deprecated or removed in TensorRT 10.0.0.

- Deprecated the `kWEIGHTLESS` builder flag. Superseded by the `kSTRIP_PLAN` builder flag. `kSTRIP_PLAN` works with either the `kREFIT` flag or the new `kREFIT_IDENTICAL` flag, defaulting to the latter if neither is set.
- In 10.0, we removed deprecated APIs compared to 9.3 and earlier releases. These removed APIs were deprecated before March 2023. We ensured that Version Compatibility is expected between 8.6, 9.x, and 10.0 versions. If you are unfamiliar with these changes, refer to our sample code for clarification. In light of the changes to the API in TensorRT 10.0, we've prepared an API Migration Guide to highlight the API modifications.
- We removed implicit batch support and worked with networks as they always have an explicit batch.
- Deprecated `TacticSource::kCUDNN` and `TacticSource::kCUBLAS` flags.
- Deprecated `IPluginV2DynamicExt`, implement `IPluginV3` instead. Refer to the Migrating V2 Plugins to IPluginV3 for how existing `IPluginV2DynamicExt` plugins can be migrated to `IPluginV3`.
- `IPluginCreator::getTensorRTVersion()` has been removed.
- Deprecated `IPluginV2IOExt`; implement `IPluginV3` instead.
- Deprecated `IPluginCreator`. There is no alternative factory class for `IPluginV2`-derivative plugin base classes, as they are all deprecated as well. Implement `IPluginV3` and its corresponding factory class `IPluginV3CreatorOne`.
- Deprecated the following APIs in `IPluginRegistry`:
  - `IPluginRegistry::registerCreator(IPluginCreator&)`. Use its overload `IPluginRegistry::registerCreator(IPluginCreatorInterface&)` instead.
  - `IPluginRegistry::deregisterCreator(IPluginCreator const&)`. Use its overload `IPluginRegistry::deregisterCreator(IPluginCreatorInterface const&)` instead.
  - `IPluginRegistry::getPluginCreator`. Use `IPluginRegistry::getCreator` instead.
  - `IPluginRegistry::getPluginCreatorList`. Use `IPluginRegistry::getAllCreators` instead.

Fixed Issues

- The `nvinfer_plugin.lib` library within the Windows package was incorrectly distributed as a static linking library starting with TensorRT 9.0. TensorRT 10.0 reverts this library to a dynamic linking library matching the behavior of TensorRT 8.6.
- There was an up to 9% performance drop for BERT networks with `gelu_erf` activation in BF16 precision compared to TensorRT 9.1 on NVIDIA Ampere GPUs.
- There was an up to 11% performance drop for ViT networks in TF32 precision compared to TensorRT 9.0 on NVIDIA Ampere GPUs.
- There was an up to 23% performance regression compared to TensorRT 8.5 on Temporal Fusion Transformers in FP32 precision on NVIDIA Turing and NVIDIA Ampere GPUs.
- A higher builder optimization level did not always give a better performance when compared to a lower builder optimization level; which could happen on all platforms and up to 27%. The workaround was to build an engine using a lower builder optimization level.
- If an ONNX model contained a `Range` operator and its `limit` input was a data-dependent tensor, engine building would likely fail.

Known Issues

Functional

- Indexing for layer information (`--dumpLayer`) and its profiling information (`--dumpProfile`) will be added in the GA release. Currently, you may see duplicating layer names if the layer consists of identical components.
- CUDA compute sanitizer may report racecheck hazards for some legacy kernels, however, related kernels do not have functional issues at runtime.
- There is a known issue that the compute sanitizer in CUDA Toolkit 12.3 might cause target application crash.
- The compute sanitizer initcheck tool may flag false positive `Uninitialized __global__ memory read` errors when running TensorRT applications on NVIDIA Hopper GPUs. These errors can be safely ignored and will be fixed in an upcoming CUDA release.
- Multihead attention fusion might not happen and affect performance if the number of heads is small.
- Hardware forward compatibility (HFC) is broken on L4T Concord for ViT, Swin-Transformers, and BERT networks in FP16 mode. A workaround is to only use FP32 mode on L4T Concord or turn off HFC.
- Compute Sanitizer from CUDA Toolkit 12.0/12.1 may report a false alarm about invalid memory access in `generatedNativePointwise` kernels.

- If a network has a tensor of type bool with an *implicitly* data-dependent shape, engine building will likely fail.
- There is an occurance of use-after-free in NVRTC that has been fixed in CUDA 12.1. When using NVRTC from CUDA 12.0 together with the TensorRT static library, you may encounter a crash in certain scenarios. Linking with the NVRTC and PTXJIT compiler from CUDA 12.1 or newer will resolve this issue.
- Although the version compatible runtime is optimized for efficiency, it may result in slower performance than the full runtime in certain use cases. Most networks can expect no more than a 10% slowdown when using a version-compatible engine compared to a version-locked engine. However, in some cases, a larger performance drop may occur. For example:
  - When running ResNet50_v2 with QAT, there may be up to a 11% decrease in performance.
  - When running DynUNet in FP16 precision, there may be up to a 32% decrease in performance.
- There are known issues reported by the Valgrind memory leak check tool when detecting potential memory leaks from TensorRT applications. The recommendation to suppress the issues is to provide a Valgrind suppression file with the following contents when running the Valgrind memory leak check tool. Add the option `--keep-debuginfo=yes` to the Valgrind command line to suppress these errors.

```
{
  Memory leak errors with dlopen.
   Memcheck:Leak
   match-leak-kinds: definite
   ...
   fun:*dlopen*
   ...
}
{
    Memory leak errors with nvrtc
    Memcheck:Leak
    match-leak-kinds: definite
    fun:malloc
    obj:*libnvrtc.so*
    ...
}
```

- SM 7.5 and earlier devices may not have INT8 implementations for all layers with Q/DQ nodes. In this case, you will encounter a `could not find any implementation` error while building your engine. To resolve this, remove the Q/DQ nodes, which quantize the failing layers.
- Installing the `cuda-compat-11-4` package may interfere with CUDA enhanced compatibility and cause TensorRT to fail even when the driver is r465. The workaround is to remove the `cuda-compat-11-4` package or upgrade the driver to r470. *(not applicable for Jetson platforms)*
- For some networks, using a batch size of 4096 may cause accuracy degradation on DLA.

- Hardware compatible engines built with CUDA versions older than 11.5 may crash during inference when run on a GPU with a compute capability lower than that of the GPU where the engine was built. A workaround is to build on the GPU with the lowest compute capability.
- For broadcasting elementwise layers running on DLA with GPU fallback enabled with one NxCxHxW input and one Nx1x1x1 input, there is a known accuracy issue if at least one of the inputs is consumed in `kDLA_LINEAR` format. It is recommended to explicitly set the input formats of such elementwise layers to different tensor formats.
- The ONNX Parser Refitter cannot refit weights defined in nested ONNX structures such as If, Loop, or Scan operations. In these cases it's recommended to perform the refit directly through the TensorRT APIs.
- BERT-like networks with QAT may not build engines successfully with refit on.
- The OnnxParserRefitter Python API documentation is missing. Refer to Refitting a Weight-Stripped Engine Directly from ONNX on how to use this class in Python.
- Exclusive padding with `kAVERAGE` pooling is not supported.
- If the `_gemm_mha_v2` operation is used, the outputs will mismatch the output of PyTorch or the CPU executor. This problem may show up only when building engines with FP16 precision, as `_gemm_mha_v2` has an implementation only for FP16.
- The layer names reported by `IEngineInspector` may not match the layer names reported by `IProfiler`.
- TensorRT does not clean temporary DLL files automatically on Windows when running in `vc` mode.
- TensorRT may crash when building transformer based networks on Windows 10 and H100.
- Running sync/race check with newer Compute Sanitizer on L4T may hit a hang issue. The workaround is to try an older version of Compute Sanitizer.
- There is an accuracy drop running DINO-FAN-base models compared to TensorRT 8.6.1.6.
- The Valgrind tool found a memory leak on L4T with CUDA 12.4 due to a known driver issue. This is expected to be fixed in CUDA 12.6.
- Some networks fail at the engine building phase on Windows + H100, but can execute on Linux. The root cause is a builder issue, where fusion compilation fails.
- While running some networks on Windows in version compatible mode (`--vc`), you may see an error `Unable to remove temporary DLL file` when an application, such as `trtexec`, is finished. The TensorRT library internally is still holding open file references when the application finishes. This issue does not have an impact on model performance and it should be fixed in the next release.

Performance

- Using FP16 scales for Q/DQ ops may result in numerical overflow. If this happens, use FP32 scales for Q/DQ ops instead.
- There is an up to 15% performance regression for SegResNet and StableDiffusion VAE in FP16 precision compared to TensorRT 9.3.
- There is an up to 16% performance regression for BasicUNet, DynUNet, and HighResNet in INT8 and FP16 precision compared to TensorRT 9.3.
- There is an up to 144 MB peak GPU memory usage increase compared to TensorRT 8.6 when building engines for ResNet-50 in INT8 precision on the L4T Orin platform.
- There is a performance drop on QDQ-Gemm pattern on RTX-Titan in weightless mode.
- There are performance gaps for StableDiffusion networks between Windows and Linux platforms.
- UNets with tensors containing >2^31 elements may fail during the engine building step.
- On A30, some fused MHA (multi-head attention) performance is not optimized yet. This will be improved upon in future TensorRT versions.
- There is up to 100% engine size increase for Transformer networks on Windows in FP16 precision.
- Enabling refit breaks multihead attention fusions.
- Running TensorRT-LLM with TensorRT 10.0 with INT8 kv-cache results in engine build failure due to insufficient custom scales. This workaround is to enable `StronglyTyped` mode.
- There is an up to 40 second increase in engine building for BART networks on NVIDIA Hopper GPUs.
- There is an accuracy drop running OSS HuggingFace Demo gptj-6b model when batch size > 1.
- There is an up to 14% context memory usage fluctuations compared to TensorRT 9.1 when building the engine for 3DUnet networks due to different tactics being selected.
- There is an up to 20 second increase in engine building for some large language models (LLMs) on NVIDIA Ampere GPUs.
- There are up to 21% peak GPU memory usage fluctuations when building the engine for the same network back to back due to different tactics being selected.
- There is an up to 2.5x build time increase compared to TensorRT 9.0 for certain Bert like models due to additional tactics available for evaluation.
- There is an up to 13% performance drop for the CortanaASR model on NVIDIA Ampere GPUs compared to TensorRT 8.5.
- There is an up to 18% performance drop for the ShuffleNet model on A30/A40 compared to TensorRT 8.5.1.

- Convolution on a tensor with an *implicitly* data-dependent shape may run significantly slower than on other tensors of the same size. Refer to the Glossary for the definition of implicitly data-dependent shapes.
- For some Transformer models, including ViT, Swin-Transformer, and DETR, there is a performance drop in INT8 precision (including both explicit and implicit quantization) compared to FP16 precision.
- There is a known issue on H100 that may lead to GPU hang when running TensorRT with high persistentCache usage. Limit the usage to 40% of L2 cache size as a workaround.
- There is a known performance issue when running instance normalization layers on Arm Server Base System Architecture (SBSA).
- There is a performance drop when offloading a SoftMax layer to DLA on NVIDIA Orin as compared to when running the layer on a GPU, with a larger drop for larger batch sizes. As an example, FP16 AlexNet with batch size 16 shows 32% drop when the network runs on DLA as compared to when the last SoftMax layer runs on a GPU.
- There is a known issue with DLA clocks that requires users to reboot the system after changing the nvpmodel power mode or otherwise experience a performance drop. Refer to the L4T board support package Release Notes for details.
- There is an up to 5% performance drop for networks using sparsity in FP16 precision.
- H100 performance for some LSTMs in FP16 precision is not fully optimized. This will be improved in future TensorRT versions.
- There is an up to 6% performance regression compared to TensorRT 8.5 on OpenRoadNet in FP16 precision on NVIDIA A10 GPUs.
- There is an up to 70% performance regression compared to TensorRT 8.6 on BERT networks in INT8 precision with FP16 disabled on L4 GPUs. To work around this, enable FP16 and disable INT8 in the builder config.

Chapter 2. TensorRT Release 9.x.x

2.1. TensorRT Release 9.3.0

These are the TensorRT 9.3.0 Release Notes and are applicable to x86 Linux users and Arm®-based CPU cores for Server Base System Architecture (SBSA) users on Linux. This release includes several fixes from the previous TensorRT releases as well as the following additional changes.

This GA release is for Large Language Models (LLMs) on NVIDIA A100, A10G, L4, L40, L40S, H100 GPUs, and NVIDIA GH200 Grace Hopper™ Superchip only. For applications other than LLMs, or other GPU platforms, continue to use TensorRT 8.6.1 for production use.

For previously released TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- A new `getDeviceMemorySizeForProfile` API was added, allowing users to query the device memory requirement for a certain profile. Combined with `createExecutionContextWithoutDeviceMemory`, it allows you to reallocate memory after switching a profile, avoiding wasting memory. Previously, you could only allocate a max size across all profiles.
- Improved builder speed for Large Language Models (LLMs).

Breaking API Changes

- > **!** ATTENTION: In TensorRT 9.0, due to the introduction of INT64 as a supported data type, ONNX models with INT64 I/O require INT64 bindings. Note that prior to this release, such models required INT32 bindings.
- In TensorRT 9.0, we removed `ICaffeParser`, `IUffParser`, and related classes and functions. The following APIs are removed:

- `nvcaffeparser1::IBlobNameToTensor`
- `nvcaffeparser1::IBinaryProtoBlob`
- `nvcaffeparser1::IPluginFactoryV2`
- `nvcaffeparser1::ICaffeParser`
- `nvcaffeparser1::createCaffeParser`
- `nvcaffeparser1::shutdownProtobufLibrary`
- `createNvCaffeParser_INTERNAL`
- `nvinfer1::utils::reshapeWeights`
- `nvinfer1::utils::reorderSubBuffers`
- `nvinfer1::utils::ransposeSubBuffers`
- `nvuffparser::UffInputOrder`
- `nvuffparser::FieldType`
- `nvuffparser::FieldMap`
- `nvuffparser::FieldCollection`
- `nvuffparser::IUffParser`
- `nvuffparser::createUffParser`
- `nvuffparser::shutdownProtobufLibrary`
- `createNvUffParser_INTERNAL`

- With removal of `ICaffeParser` and `IUffParsers`, the `libnvparsers` library is removed.
- `uff`, `graphsurgeon`, and related networks are removed from TensorRT packages.

Deprecated API Lifetime

- APIs deprecated in TensorRT 9.3 will be retained until at least 1/2025.
- APIs deprecated in TensorRT 9.2 will be retained until at least 11/2024.
- APIs deprecated in TensorRT 9.1 will be retained until at least 10/2024.
- APIs deprecated in TensorRT 9.0 will be retained until at least 8/2024.
- APIs deprecated in TensorRT 8.6 will be retained until at least 2/2024.
- APIs deprecated in TensorRT 8.4 or before will be removed in TensorRT 10.0.

Refer to the API documentation (C++, Python) for how to update your code to remove the use of deprecated features.

Compatibility

- TensorRT 9.3.0 has been tested with the following:
  - cuDNN 8.9.7
  - TensorFlow 2.12.0

  - PyTorch >= 2.0 (refer to the `requirements.txt` file for each sample)
  - ONNX 1.14.1
- This TensorRT release supports CUDA®:
  - 12.3 update 2
  - 12.2 update 1
  - 12.1 update 1
  - 12.0 update 1
  - 11.8
  - 11.7 update 1
  - 11.6 update 2
  - 11.5 update 2
  - 11.4 update 4
  - 11.3 update 1
  - 11.2 update 2
  - 11.1 update 1
  - 11.0 update 1
- This TensorRT release requires at least NVIDIA driver r450 on Linux or r452 on Windows as required by CUDA 11.0, which is the minimum CUDA version supported by this TensorRT release.
- It is suggested that you use TensorRT with a software stack that has been tested; including cuDNN and cuBLAS versions as documented in the Features For Platforms And Software section. Other semantically compatible releases of cuDNN and cuBLAS can be used; however, other versions may have performance improvements as well as regressions. In rare cases, functional regressions might also be observed.

Limitations

- There are two modes of DLA softmax where the mode is chosen automatically based on the shape of the input tensor, where:
  - the first mode triggers when all nonbatch, non-axis dimensions are 1, and
  - the second mode triggers in other cases if valid.

  The second of the two modes is supported only for DLA 3.9.0 and later. It involves approximations that may result in errors of a small degree. Also, batch size greater than 1 is supported only for DLA 3.9.0 and later. Refer to DLA Supported Layers for more information.
- On QNX, networks that are segmented into a large number of DLA loadables may fail during inference.

- You may encounter an error such as, "*Unable to load library: nvinfer_builder_resource.dll*", if using Python 3.9.10 on Windows. You can workaround this issue by downgrading to an earlier version of Python 3.9.
- The DLA compiler is capable of removing identity transposes, but it cannot fuse multiple adjacent transpose layers into a single transpose layer (likewise for reshape). For example, given a TensorRT `IShuffleLayer` consisting of two non-trivial transposes and an identity reshapes in between. The shuffle layer is translated into two consecutive DLA transpose layers, unless you merge the transposes together manually in the model definition in advance.
- In explicitly quantized networks, a group convolution that has a Q/DQ pair before but no Q/DQ pair after is expected to run with INT8-IN-FP32-OUT mixed precision. However, on NVIDIA Hopper™ it may fall back to FP32-IN-FP32-OUT if the input channel count is small. This will be fixed in a future release.
- On PowerPC platforms, samples that depend on TensorFlow, ONNX Runtime, and PyTorch are unable to run due to missing Python module dependencies. These frameworks have not been built for PowerPC and/or published to standard repositories.
- `nvinfer1::UnaryOperation::kROUND` or `nvinfer1::UnaryOperation::kSIGN` operations of `IUnaryLayer` are not supported in the implicit batch mode.
- For networks containing normalization layers, particularly if deploying with mixed precision, target the latest ONNX opset that contains the corresponding function ops, for example: opset 17 for LayerNormalization or opset 18 GroupNormalization. Numerical accuracy using function ops is superior to corresponding implementation with primitive ops for normalization layers.
- `QuantizeLayer` and `DequantizeLayer` only support FP32 scale and data, even when using ONNX opset 19. If the input is not FP32, you must add a `Cast` to FP32 on the input to `QuantizeLayer`, and a `Cast` from FP32 at the output of `DequantizeLayer`.
- `EngineInspector::getLayerInformation` may return incomplete JSON data for some engines produced by TensorRT 9.0. When this happens, TensorRT Engine Explorer cannot be used to analyze the engine or generate a graph of the engine layers.
- The released Windows DLLs are built with the `MT_Static` flag. TensorRT will switch back to the `MT_Dynamic` flag in the next major release.

Deprecated and Removed Features

The following features have been deprecated or removed in TensorRT 9.0. Some deprecations that were planned to be removed in 9.0, but have not yet been removed, may be removed in TensorRT 10.0.

- TensorRT samples and open source demos are no longer supported on Python < 3.8.
- Ubuntu 18.04 has reached end of life and is no longer supported by TensorRT starting with TensorRT 9.0.

- The following plugins were deprecated:
  - `BatchedNMS_TRT`
  - `BatchedNMSDynamic_TRT`
  - `BatchTilePlugin_TRT`
  - `Clip_TRT`
  - `CoordConvAC`
  - `CropAndResize`
  - `EfficientNMS_ONNX_TRT`
  - `CustomGeluPluginDynamic`
  - `LReLU_TRT`
  - `NMSDynamic_TRT`
  - `NMS_TRT`
  - `Normalize_TRT`
  - `Proposal`
  - `SingleStepLSTMPlugin`
  - `SpecialSlice_TRT`
  - `Split`
- The following C++ API classes were deprecated:
  - `NvUtils`
- The following C++ API methods were deprecated:
  - ```
    nvinfer1::INetworkDefinition::addFill(nvinfer1::Dims dimensions,
     nvinfer1::FillOperation
            op)
    ```
    - Only the 2-parameter version of this function is deprecated.
  - ```
    nvinfer1::INetworkDefinition::addDequantize(nvinfer1::ITensor &input,
     nvinfer1::ITensor
            &scale)
    ```
    - Only the 2-parameter version of this function is deprecated.
  - ```
    nvinfer1::INetworkDefinition::addQuantize(nvinfer1::ITensor &input,
     nvinfer1::ITensor
            &scale)
    ```
    - Only the 2-parameter version of this function is deprecated.
- The following C++ API enums were deprecated:
  - `nvinfer1::TacticSource::kCUBLAS_LT`
  - `nvonnxparser::OnnxParserFlag::kNATIVE_INSTANCENORM`
- The following Python API methods were deprecated:
  - `INetworkDefinition.add_fill(shape, op)`
  - `INetworkDefinition.add_dequantize(input, scale)`
- The following Python API enums were deprecated:
    ```

- `TacticSource.CUBLAS_LT`
- `OnnxParserFlag.NATIVE_INSTANCENORM`

Known Issues

Functional

- CUDA compute sanitizer may report racecheck hazards for some legacy kernels, however, related kernels do not have functional issues at runtime.
- There is a known issue that the compute sanitizer in CUDA Toolkit 12.3 might cause target application crash.
- The compute sanitizer initcheck tool may flag false positive `Uninitialized __global__ memory read` errors when running TensorRT applications on NVIDIA Hopper GPUs. These errors can be safely ignored and will be fixed in an upcoming CUDA release.
- Multihead attention fusion might not happen and affect performance if the number of heads is small.
- If a ONNX model contains a `Range` operator and its `limit` input is a data-dependent tensor, engine building will likely fail.
- Hardware forward compatibility (HFC) is broken on L4T Concord for ViT, Swin-Transformers, and BERT networks in FP16 mode. A workaround is to only use FP32 mode on L4T Concord or turn off HFC.
- Compute Sanitizer from CUDA Toolkit 12.0/12.1 may report a false alarm about invalid memory access in `generatedNativePointwise` kernels.
- If a network has a tensor of type bool with an *implicitly* data-dependent shape, engine building will likely fail.
- There is an occurance of use-after-free in NVRTC that has been fixed in CUDA 12.1. When using NVRTC from CUDA 12.0 together with the TensorRT static library, you may encounter a crash in certain scenarios. Linking with the NVRTC and PTXJIT compiler from CUDA 12.1 or newer will resolve this issue.
- Although the version compatible runtime is optimized for efficiency, it may result in slower performance than the full runtime in certain use cases. Most networks can expect no more than a 10% slowdown when using a version-compatible engine compared to a version-locked engine. However, in some cases, a larger performance drop may occur. For example:
  - When running ResNet50_v2 with QAT, there may be up to a 22% decrease in performance.
  - When running DynUNet in FP16 precision, there may be up to a 32% decrease in performance.
- There are known issues reported by the Valgrind memory leak check tool when detecting potential memory leaks from TensorRT applications. The recommendation

to suppress the issues is to provide a Valgrind suppression file with the following contents when running the Valgrind memory leak check tool. Add the option `--keep-debuginfo=yes` to the Valgrind command line to suppress these errors.

```
{
  Memory leak errors with dlopen.
   Memcheck:Leak
   match-leak-kinds: definite
   ...
   fun:*dlopen*
   ...
}
{
    Memory leak errors with nvrtc
    Memcheck:Leak
    match-leak-kinds: definite
    fun:malloc
    obj:*libnvrtc.so*
    ...
}
```

- SM 7.5 and earlier devices may not have INT8 implementations for all layers with Q/DQ nodes. In this case, you will encounter a `could not find any implementation` error while building your engine. To resolve this, remove the Q/DQ nodes, which quantize the failing layers.
- Installing the `cuda-compat-11-4` package may interfere with CUDA enhanced compatibility and cause TensorRT to fail even when the driver is r465. The workaround is to remove the `cuda-compat-11-4` package or upgrade the driver to r470. *(not applicable for Jetson platforms)*
- For some networks, using a batch size of 4096 may cause accuracy degradation on DLA.
- Hardware compatible engines built with CUDA versions older than 11.5 may crash during inference when run on a GPU with a compute capability lower than that of the GPU where the engine was built. A workaround is to build on the GPU with the lowest compute capability.
- For broadcasting elementwise layers running on DLA with GPU fallback enabled with one NxCxHxW input and one Nx1x1x1 input, there is a known accuracy issue if at least one of the inputs is consumed in `kDLA_LINEAR` format. It is recommended to explicitly set the input formats of such elementwise layers to different tensor formats.

Performance

- There is an up to 9% performance drop for BERT networks with `gelu_erf` activation in BF16 precision compared to TensorRT 9.1 on NVIDIA Ampere GPUs.
- There is an up to 40 second increase in engine building for BART networks on NVIDIA Hopper GPUs.
- There is an accuracy drop running OSS HuggingFace Demo gptj-6b model when batch size > 1.

- There is an up to 14% context memory usage fluctuations compared to TensorRT 9.1 when building the engine for 3DUnet networks due to different tactics being selected.
- There is an up to 20 second increase in engine building for some large language models (LLMs) on NVIDIA Ampere GPUs.
- There is an up to 6% performance drop for BERT networks in FP32 precision compared to TensorRT 9.0 on NVIDIA Volta GPUs.
- There are up to 21% peak GPU memory usage fluctuations when building the engine for the same network back to back due to different tactics being selected.
- There is an up to 11% performance drop for ViT networks in TF32 precision compared to TensorRT 9.0 on NVIDIA Ampere GPUs.
- There is an up to 12% performance drop for BERT networks in FP16 precision compared to TensorRT 9.0 on NVIDIA Ada Lovelace GPUs.
- There is an up to 2.5x build time increase compared to TensorRT 9.0 for certain Bert like models due to additional tactics available for evaluation.
- There is an up to 13% performance drop for the CortanaASR model on NVIDIA Ampere GPUs compared to TensorRT 8.5.
- There is a known performance regression in the grouped deconvolution layer due to disabling cuDNN tactics. In TensorRT 8.6, performance can be recovered by unsetting `nvinfer1::PreviewFeature::kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805`. We will close the performance gap in a future release.
- There is an up to 27% performance drop for the SegResNet model on Ampere GPUs compared to TensorRT 8.6 EA. This drop can be avoided by enabling the `kVERSION_COMPATIBLE` flag in the ONNX parser.
- There is an up to 18% performance drop for the ShuffleNet model on A30/A40 compared to TensorRT 8.5.1.
- Convolution on a tensor with an *implicitly* data-dependent shape may run significantly slower than on other tensors of the same size. Refer to the Glossary for the definition of implicitly data-dependent shapes.
- For some Transformer models, including ViT, Swin-Transformer, and DETR, there is a performance drop in INT8 precision (including both explicit and implicit quantization) compared to FP16 precision.
- There is an up to 30% performance regression for LSTM variants with dynamic shapes. This issue can be resolved by disabling the `kFASTER_DYNAMIC_SHAPES_0805` preview feature in TensorRT 8.6.
- There is a known issue on H100 that may lead to GPU hang when running TensorRT with high persistentCache usage. Limit the usage to 40% of L2 cache size as a workaround.
- There is a known performance issue when running instance normalization layers on Arm Server Base System Architecture (SBSA).

- There is a performance drop when offloading a SoftMax layer to DLA on NVIDIA Orin as compared to when running the layer on a GPU, with a larger drop for larger batch sizes. As an example, FP16 AlexNet with batch size 16 shows 32% drop when the network runs on DLA as compared to when the last SoftMax layer runs on a GPU.
- There is a known issue with DLA clocks that requires users to reboot the system after changing the nvpmodel power mode or otherwise experience a performance drop. Refer to the L4T board support package Release Notes for details.
- There is an up to 5% performance drop for networks using sparsity in FP16 precision.
- H100 performance for some LSTMs in FP16 precision is not fully optimized. This will be improved in future TensorRT versions.
- There is an up to 6% performance regression compared to TensorRT 8.5 on OpenRoadNet in FP16 precision on NVIDIA A10 GPUs.
- There is an up to 23% performance regression compared to TensorRT 8.5 on LSTMs in FP32 precision when dynamic shapes are used on NVIDIA Turing GPUs. Set the `kFASTER_DYNAMIC_SHAPES_0805` preview flag to `false` as a workaround.
- There is an up to 23% performance regression compared to TensorRT 8.5 on Temporal Fusion Transformers in FP32 precision on NVIDIA Turing and NVIDIA Ampere GPUs.
- A higher builder optimization level does not always give a better performance when compared to a lower builder optimization level; which may happen on all platforms and up to 27%. The workaround is to build an engine using a lower builder optimization level.
- There is an up to 70% performance regression compared to TensorRT 8.6 on BERT networks in INT8 precision with FP16 disabled on L4 GPUs. To work around this, enable FP16 and disable INT8 in the builder config.

2.2. TensorRT Release 9.2.0

These are the TensorRT 9.2.0 Release Notes and are applicable to x86 Linux users and Arm®-based CPU cores for Server Base System Architecture (SBSA) users on Linux. This release includes several fixes from the previous TensorRT releases as well as the following additional changes.

This GA release is for Large Language Models (LLMs) on NVIDIA A100, A10G, L4, L40, L40S, H100 GPUs, and NVIDIA GH200 Grace Hopper™ Superchip only. For applications other than LLMs, or other GPU platforms, continue to use TensorRT 8.6.1 for production use.

For previously released TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Announcements

- Added support for NVIDIA GH200 Grace Hopper Superchip.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- TensorRT 9.2 provides enhanced support for Large Language Models (LLMs). Refer to TensorRT-LLM for a list of supported LLMs.
- The following C++ API classes were added:
  - `ISerializationConfig`
- The following C++ API methods were added:
  - `ICudaEngine::createSerializationConfig()`
  - `ICudaEngine::serializeWithConfig(ISerializationConfig& config)`
- The following C++ API enums were added:
  - `SerializationFlag::kEXCLUDE_WEIGHTS`
  - `SerializationFlag::kEXCLUDE_LEAN_RUNTIME`
- The following Python API classes were added:
  - `ISerializationConfig`
- The following Python API methods were added:
  - `ICudaEngine::create_serialization_config()`
  - `ICudaEngine::serialize_with_config(config)`
- The following Python API enums were added:
  - `SerializationFlag.EXCLUDE_WEIGHTS`
  - `SerializationFlag.EXCLUDE_LEAN_RUNTIME`
- Added a new `kWEIGHTLESS` builder flag to build the engine without saving weights with no impact on runtime performance. You need to use the refit API to pull those weights back before inference.
- Added a new `serializeWithConfig` API to serialize the engine with optional new flags `kEXCLUDE_WEIGHTS` and `kEXCLUDE_LEAN_RUNTIME`.
- Added support for FP32 accumulation (single precision) for inputs/outs in FP16 (half precision) format.

Breaking API Changes

- > **!** ATTENTION: In TensorRT 9.0, due to the introduction of INT64 as a supported data type, ONNX models with INT64 I/O require INT64 bindings. Note that prior to this release, such models required INT32 bindings.
- In TensorRT 9.0, we removed `ICaffeParser`, `IUffParser`, and related classes and functions. The following APIs are removed:
  - `nvcaffeparser1::IBlobNameToTensor`
  - `nvcaffeparser1::IBinaryProtoBlob`
  - `nvcaffeparser1::IPluginFactoryV2`
  - `nvcaffeparser1::ICaffeParser`
  - `nvcaffeparser1::createCaffeParser`
  - `nvcaffeparser1::shutdownProtobufLibrary`
  - `createNvCaffeParser_INTERNAL`
  - `nvinfer1::utils::reshapeWeights`
  - `nvinfer1::utils::reorderSubBuffers`
  - `nvinfer1::utils::ransposeSubBuffers`
  - `nvuffparser::UffInputOrder`
  - `nvuffparser::FieldType`
  - `nvuffparser::FieldMap`
  - `nvuffparser::FieldCollection`
  - `nvuffparser::IUffParser`
  - `nvuffparser::createUffParser`
  - `nvuffparser::shutdownProtobufLibrary`
  - `createNvUffParser_INTERNAL`
- With removal of `ICaffeParser` and `IUffParsers`, the `libnvparsers` library is removed.
- `uff`, `graphsurgeon`, and related networks are removed from TensorRT packages.

Deprecated API Lifetime

- APIs deprecated in TensorRT 9.2 will be retained until at least 11/2024.
- APIs deprecated in TensorRT 9.1 will be retained until at least 10/2024.
- APIs deprecated in TensorRT 9.0 will be retained until at least 8/2024.
- APIs deprecated in TensorRT 8.6 will be retained until at least 2/2024.
- APIs deprecated in TensorRT 8.5 will be retained until at least 9/2023.
- APIs deprecated in TensorRT 8.4 or before will be removed in TensorRT 10.0.

Refer to the API documentation (C++, Python) for how to update your code to remove the use of deprecated features.

Compatibility

- TensorRT 9.2.0 has been tested with the following:
  - cuDNN 8.9.6
  - TensorFlow 2.12.0
  - PyTorch >= 2.0 (refer to the `requirements.txt` file for each sample)
  - ONNX 1.14.1
- This TensorRT release supports CUDA®:
  - 12.3 update 1
  - 12.2 update 1
  - 12.1 update 1
  - 12.0 update 1
  - 11.8
  - 11.7 update 1
  - 11.6 update 2
  - 11.5 update 2
  - 11.4 update 4
  - 11.3 update 1
  - 11.2 update 2
  - 11.1 update 1
  - 11.0 update 1
- This TensorRT release requires at least NVIDIA driver r450 on Linux or r452 on Windows as required by CUDA 11.0, which is the minimum CUDA version supported by this TensorRT release.
- It is suggested that you use TensorRT with a software stack that has been tested; including cuDNN and cuBLAS versions as documented in the Features For Platforms And Software section. Other semantically compatible releases of cuDNN and cuBLAS can be used; however, other versions may have performance improvements as well as regressions. In rare cases, functional regressions might also be observed.

Limitations

- There are two modes of DLA softmax where the mode is chosen automatically based on the shape of the input tensor, where:
  - the first mode triggers when all nonbatch, non-axis dimensions are 1, and
  - the second mode triggers in other cases if valid.

The second of the two modes is supported only for DLA 3.9.0 and later. It involves approximations that may result in errors of a small degree. Also, batch size greater than 1 is supported only for DLA 3.9.0 and later. Refer to DLA Supported Layers for more information.

- On QNX, networks that are segmented into a large number of DLA loadables may fail during inference.
- You may encounter an error such as, "*Unable to load library: nvinfer_builder_resource.dll*", if using Python 3.9.10 on Windows. You can workaround this issue by downgrading to an earlier version of Python 3.9.
- The DLA compiler is capable of removing identity transposes, but it cannot fuse multiple adjacent transpose layers into a single transpose layer (likewise for reshape). For example, given a TensorRT `IShuffleLayer` consisting of two non-trivial transposes and an identity reshapes in between. The shuffle layer is translated into two consecutive DLA transpose layers, unless you merge the transposes together manually in the model definition in advance.
- In explicitly quantized networks, a group convolution that has a Q/DQ pair before but no Q/DQ pair after is expected to run with INT8-IN-FP32-OUT mixed precision. However, on NVIDIA Hopper™ it may fall back to FP32-IN-FP32-OUT if the input channel count is small. This will be fixed in a future release.
- On PowerPC platforms, samples that depend on TensorFlow, ONNX Runtime, and PyTorch are unable to run due to missing Python module dependencies. These frameworks have not been built for PowerPC and/or published to standard repositories.
- `nvinfer1::UnaryOperation::kROUND` or `nvinfer1::UnaryOperation::kSIGN` operations of `IUnaryLayer` are not supported in the implicit batch mode.
- For networks containing normalization layers, particularly if deploying with mixed precision, target the latest ONNX opset that contains the corresponding function ops, for example: opset 17 for LayerNormalization or opset 18 GroupNormalization. Numerical accuracy using function ops is superior to corresponding implementation with primitive ops for normalization layers.
- `QuantizeLayer` and `DequantizeLayer` only support FP32 scale and data, even when using ONNX opset 19. If the input is not FP32, you must add a `Cast` to FP32 on the input to `QuantizeLayer`, and a `Cast` from FP32 at the output of `DequantizeLayer`.
- `EngineInspector::getLayerInformation` may return incomplete JSON data for some engines produced by TensorRT 9.0. When this happens, TensorRT Engine Explorer cannot be used to analyze the engine or generate a graph of the engine layers.
- The released Windows DLLs are built with the `MT_Static` flag. TensorRT will switch back to the `MT_Dynamic` flag in the next major release.

Deprecated and Removed Features

The following features have been deprecated or removed in TensorRT 9.0. Some deprecations that were planned to be removed in 9.0, but have not yet been removed, may be removed in TensorRT 10.0.

- TensorRT samples and open source demos are no longer supported on Python < 3.8.
- Ubuntu 18.04 has reached end of life and is no longer supported by TensorRT starting with TensorRT 9.0.
- The following plugins were deprecated:
  - `BatchedNMS_TRT`
  - `BatchedNMSDynamic_TRT`
  - `BatchTilePlugin_TRT`
  - `Clip_TRT`
  - `CoordConvAC`
  - `CropAndResize`
  - `EfficientNMS_ONNX_TRT`
  - `CustomGeluPluginDynamic`
  - `LReLU_TRT`
  - `NMSDynamic_TRT`
  - `NMS_TRT`
  - `Normalize_TRT`
  - `Proposal`
  - `SingleStepLSTMPlugin`
  - `SpecialSlice_TRT`
  - `Split`
- The following C++ API classes were deprecated:
  - `NvUtils`
- The following C++ API methods were deprecated:
  - ```
    nvinfer1::INetworkDefinition::addFill(nvinfer1::Dims dimensions,
     nvinfer1::FillOperation
            op)
    ```
    - Only the 2-parameter version of this function is deprecated.
  - ```
    nvinfer1::INetworkDefinition::addDequantize(nvinfer1::ITensor &input,
     nvinfer1::ITensor
            &scale)
    ```
    - Only the 2-parameter version of this function is deprecated.
  - ```
    nvinfer1::INetworkDefinition::addQuantize(nvinfer1::ITensor &input,
     nvinfer1::ITensor
            &scale)
    ```
    - Only the 2-parameter version of this function is deprecated.
    ```

- The following C++ API enums were deprecated:
  - `nvinfer1::TacticSource::kCUBLAS_LT`
  - `nvonnxparser::OnnxParserFlag::kNATIVE_INSTANCENORM`
- The following Python API methods were deprecated:
  - `INetworkDefinition.add_fill(shape, op)`
  - `INetworkDefinition.add_dequantize(input, scale)`
- The following Python API enums were deprecated:
  - `TacticSource.CUBLAS_LT`
  - `OnnxParserFlag.NATIVE_INSTANCENORM`

Fixed Issues

- There was an 1x FP32 model CPU memory leak from `onnx.export` tracked in issue 106976 for the TensorRT open source software HuggingFace demo. You may have encountered higher peak CPU memory usage for fresh runs, but will drop for cached-ONNX runs. This issue has been fixed.
- The compute sanitizer synccheck tool may have reported a false alarm when running conv and GEMM kernels with CGA size >= 4 on NVIDIA Hopper GPUs. This issue has been fixed.
- When using cuDNN 8.9.2.26 and the TensorRT RNNv2 API, the compute sanitizer from CUDA Toolkit 12.2 may have reported race conditions in CUTLASS kernels. This issue has been fixed.
- There was a known floating point exception when running BERT networks on H100 multi-instance GPU (MIG). This issue has been fixed.
- TensorRT did not preserve precision for operations that are imported from ONNX models if using weakly typed networks. The fix for the issue is to import networks as strongly typed which preserve precision for operations.

Known Issues

Functional

- CUDA compute sanitizer may report racecheck hazards for some legacy kernels written directly in SASS, however, related kernels do not have functional issues at runtime.
- There is a known issue that the compute sanitizer in CUDA Toolkit 12.3 might cause target application crash.
- The compute sanitizer initcheck tool may flag false positive `Uninitialized __global__ memory read` errors when running TensorRT applications on NVIDIA Hopper GPUs. These errors can be safely ignored and will be fixed in an upcoming CUDA release.

- Multihead attention fusion might not happen and affect performance if the number of heads is small.
- If a ONNX model contains a `Range` operator and its `limit` input is a data-dependent tensor, engine building will likely fail.
- Hardware forward compatibility (HFC) is broken on L4T Concord for ViT, Swin-Transformers, and BERT networks in FP16 mode. A workaround is to only use FP32 mode on L4T Concord or turn off HFC.
- Compute Sanitizer from CUDA Toolkit 12.0/12.1 may report a false alarm about invalid memory access in `generatedNativePointwise` kernels.
- If a network has a tensor of type bool with an *implicitly* data-dependent shape, engine building will likely fail.
- There is an occurance of use-after-free in NVRTC that has been fixed in CUDA 12.1. When using NVRTC from CUDA 12.0 together with the TensorRT static library, you may encounter a crash in certain scenarios. Linking with the NVRTC and PTXJIT compiler from CUDA 12.1 or newer will resolve this issue.
- Although the version compatible runtime is optimized for efficiency, it may result in slower performance than the full runtime in certain use cases. Most networks can expect no more than a 10% slowdown when using a version-compatible engine compared to a version-locked engine. However, in some cases, a larger performance drop may occur. For example:
  - When running ResNet50_v2 with QAT, there may be up to a 22% decrease in performance.
  - When running DynUNet in FP16 precision, there may be up to a 32% decrease in performance.
- There are known issues reported by the Valgrind memory leak check tool when detecting potential memory leaks from TensorRT applications. The recommendation to suppress the issues is to provide a Valgrind suppression file with the following contents when running the Valgrind memory leak check tool. Add the option `--keep-debuginfo=yes` to the Valgrind command line to suppress these errors.

```
{
  Memory leak errors with dlopen.
   Memcheck:Leak
   match-leak-kinds: definite
   ...
   fun:*dlopen*
   ...
}
{
    Memory leak errors with nvrtc
    Memcheck:Leak
    match-leak-kinds: definite
    fun:malloc
    obj:*libnvrtc.so*
    ...
}
```

- SM 7.5 and earlier devices may not have INT8 implementations for all layers with Q/DQ nodes. In this case, you will encounter a `could not find any implementation`

error while building your engine. To resolve this, remove the Q/DQ nodes, which quantize the failing layers.

- Installing the `cuda-compat-11-4` package may interfere with CUDA enhanced compatibility and cause TensorRT to fail even when the driver is r465. The workaround is to remove the `cuda-compat-11-4` package or upgrade the driver to r470. *(not applicable for Jetson platforms)*
- For some networks, using a batch size of 4096 may cause accuracy degradation on DLA.
- Hardware compatible engines built with CUDA versions older than 11.5 may crash during inference when run on a GPU with a compute capability lower than that of the GPU where the engine was built. A workaround is to build on the GPU with the lowest compute capability.
- For broadcasting elementwise layers running on DLA with GPU fallback enabled with one NxCxHxW input and one Nx1x1x1 input, there is a known accuracy issue if at least one of the inputs is consumed in `kDLA_LINEAR` format. It is recommended to explicitly set the input formats of such elementwise layers to different tensor formats.

Performance

- There is an up to 9% performance drop for BERT networks with `gelu_erf` activation in BF16 precision compared to TensorRT 9.1 on NVIDIA Ampere GPUs.
- There is an up to 40 second increase in engine building for BART networks on NVIDIA Hopper GPUs.
- There is an accuracy drop running OSS HuggingFace Demo gptj-6b model when batch size > 1.
- There is an up to 14% context memory usage fluctuations compared to TensorRT 9.1 when building the engine for 3DUnet networks due to different tactics being selected.
- There is an up to 20 second increase in engine building for some large language models (LLMs) on NVIDIA Ampere GPUs.
- There is an up to 6% performance drop for BERT networks in FP32 precision compared to TensorRT 9.0 on NVIDIA Volta GPUs.
- There are up to 21% peak GPU memory usage fluctuations when building the engine for the same network back to back due to different tactics being selected.
- There is an up to 11% performance drop for ViT networks in TF32 precision compared to TensorRT 9.0 on NVIDIA Ampere GPUs.
- There is an up to 12% performance drop for BERT networks in FP16 precision compared to TensorRT 9.0 on NVIDIA Ada Lovelace GPUs.
- There is an up to 2.5x build time increase compared to TensorRT 9.0 for certain Bert like models due to additional tactics available for evaluation.

- There is an up to 13% performance drop for the CortanaASR model on NVIDIA Ampere GPUs compared to TensorRT 8.5.
- There is a known performance regression in the grouped deconvolution layer due to disabling cuDNN tactics. In TensorRT 8.6, performance can be recovered by unsetting `nvinfer1::PreviewFeature::kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805`. We will close the performance gap in a future release.
- There is an up to 27% performance drop for the SegResNet model on Ampere GPUs compared to TensorRT 8.6 EA. This drop can be avoided by enabling the `kVERSION_COMPATIBLE` flag in the ONNX parser.
- There is an up to 18% performance drop for the ShuffleNet model on A30/A40 compared to TensorRT 8.5.1.
- Convolution on a tensor with an *implicitly* data-dependent shape may run significantly slower than on other tensors of the same size. Refer to the Glossary for the definition of implicitly data-dependent shapes.
- For some Transformer models, including ViT, Swin-Transformer, and DETR, there is a performance drop in INT8 precision (including both explicit and implicit quantization) compared to FP16 precision.
- There is an up to 30% performance regression for LSTM variants with dynamic shapes. This issue can be resolved by disabling the `kFASTER_DYNAMIC_SHAPES_0805` preview feature in TensorRT 8.6.
- There is a known issue on H100 that may lead to GPU hang when running TensorRT with high persistentCache usage. Limit the usage to 40% of L2 cache size as a workaround.
- There is a known performance issue when running instance normalization layers on Arm Server Base System Architecture (SBSA).
- There is a performance drop when offloading a SoftMax layer to DLA on NVIDIA Orin as compared to when running the layer on a GPU, with a larger drop for larger batch sizes. As an example, FP16 AlexNet with batch size 16 shows 32% drop when the network runs on DLA as compared to when the last SoftMax layer runs on a GPU.
- There is a known issue with DLA clocks that requires users to reboot the system after changing the nvpmodel power mode or otherwise experience a performance drop. Refer to the L4T board support package Release Notes for details.
- There is an up to 5% performance drop for networks using sparsity in FP16 precision.
- H100 performance for some LSTMs in FP16 precision is not fully optimized. This will be improved in future TensorRT versions.
- There is an up to 6% performance regression compared to TensorRT 8.5 on OpenRoadNet in FP16 precision on NVIDIA A10 GPUs.
- There is an up to 23% performance regression compared to TensorRT 8.5 on LSTMs in FP32 precision when dynamic shapes are used on NVIDIA Turing GPUs. Set the `kFASTER_DYNAMIC_SHAPES_0805` preview flag to `false` as a workaround.

- There is an up to 23% performance regression compared to TensorRT 8.5 on Temporal Fusion Transformers in FP32 precision on NVIDIA Turing and NVIDIA Ampere GPUs.
- A higher builder optimization level does not always give a better performance when compared to a lower builder optimization level; which may happen on all platforms and up to 27%. The workaround is to build an engine using a lower builder optimization level.
- There is an up to 70% performance regression compared to TensorRT 8.6 on BERT networks in INT8 precision with FP16 disabled on L4 GPUs. To work around this, enable FP16 and disable INT8 in the builder config.

2.3. TensorRT Release 9.1.0

These are the TensorRT 9.1.0 Release Notes and are applicable to x86 Linux users and Arm®-based CPU cores for Server Base System Architecture (SBSA) users on Linux. This release includes several fixes from the previous TensorRT releases as well as the following additional changes.

This GA release is for Large Language Models (LLMs) on NVIDIA A100, A10G, L4, L40, L40S, H100 GPUs, and NVIDIA GH200 Grace Hopper™ Superchip only. For applications other than LLMs, or other GPU platforms, continue to use TensorRT 8.6.1 for production use.

For previously released TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Announcements

- Added support for NVIDIA GH200 Grace Hopper Superchip.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- TensorRT 9.1 provides enhanced support for Large Language Models (LLMs), including the following networks:
  - GPT
  - GPT-J
  - GPT-Neo
  - GPT-NeoX
  - BART
  - Bloom
  - Bloomz
  - OPT

  - T5
  - FLAN-T5
- Added support for `bfloat16` data types on NVIDIA Ampere GPUs and newer architectures.
- Added support for E4M3 FP8 data type on NVIDIA Hopper GPUs using explicit quantization. This allows utilizing TensorRT with TransformerEngine based FP8 models.
- Added the NeMo demo into the TensorRT OSS repository to demonstrate the performance benefit of using E4M3 FP8 data type with the GPT models trained with the NVIDIA NeMo Toolkit and TransformerEngine.
- Added `bfloat16` and FP8 I/O datatypes in plugins.
- Added support of networks running in INT8 precision where Smooth Quantization is used. Smooth Quantization is an approach that allows to improve accuracy of INT8 compute for LLM.
- Added support for INT64 data type. The ONNX parser no longer automatically casts INT64 to INT32.
- Added support for ONNX local functions when parsing ONNX models with the ONNX parser.
- Added support for caching JIT-compiled code. It can be disabled by setting `BuilderFlag::DISABLE_COMPILATION_CACHE`. The compilation cache is part of the timing cache, caches JIT-compiled code, and will be serialized as part of the timing cache by default, which may significantly increase the cache size.
- Added `NetworkDefinitionCreationFlag::kSTRONGLY_TYPED`. A strongly typed network faithfully translates the type specification of the network definition to the built engine.
- Added new refit APIs to accept weights from GPU and refit engines asynchronously. Execution contexts become reusable after refitting.
- Added two new TensorRT samples; sampleProgressMonitor (C++) and simple_progress_reporter (Python) that are examples for using Progress Monitor during engine build. For more information, refer to the NVIDIA TensorRT Samples Support Guide.
- Added support for Python-based TensorRT plugin definitions. The TensorRT sample python_plugin has been added with a few examples demonstrating Python-based plugins. For more information, refer to the Adding Custom Layers using the Python API section in the NVIDIA TensorRT Developer Guide.
- The following C++ API classes were added:
  - `IProgressMonitor`

- The following C++ API methods were added:
  - `INetworkDefinition::IFillLayer* addFill(Dims dimensions, FillOperation op, DataType outputType)`
  - `INetworkDefinition::IQuantizeLayer* addQuantize(ITensor& input, ITensor& scale, DataType outputType)`
  - `INetworkDefinition::IDequantizeLayer* addDequantize(ITensor& input, ITensor& scale, DataType outputType)`
  - `IBuilder::setProgressMonitor(IProgressMonitor* monitor)`
  - `IBuilder::getProgressMonitor()`
  - `IFillLayer::setAlphaInt64(int64_t alpha)`
  - `IFillLayer::getAlphaInt64()`
  - `IFillLayer::setBetaInt64(int64_t beta)`
  - `IFillLayer::getBetaInt64()`
  - `IFillLayer::isAlphaBetaInt64()`
  - `IFillLayer::getToType()`
  - `IFillLayer::setToType(DataType toType)`
  - `IQuantizeLayer::getToType()`
  - `IQuantizeLayer::setToType(DataType toType)`
  - `IDequantizeLayer::getToType()`
  - `IDequantizeLayer::setToType(DataType toType)`
  - `INetworkDefinition::getFlags()`
  - `INetworkDefinition::getFlag(NetworkDefinitionCreationFlag networkDefinitionCreationFlag)`
  - `IRefitter::setNamedWeights(char const* name, Weights weights, TensorLocation location)`
  - `IRefitter::getNamedWeights(char const* weightsName)`
  - `IRefitter::getWeightsLocation(char const* weightsName)`
  - `IRefitter::unsetNamedWeights(char const* weightsName)`
  - `IRefitter::setWeightsValidation(bool weightsValidation)`
  - `IRefitter::getWeightsValidation()`
  - `IRefitter::refitCudaEngineAsync(cudaStream_t stream)`
  - `nvonnxparser::IParserError::nodeName()`
  - `nvonnxparser::IParserError::nodeOperator()`
- The following C++ API enums were added:
  - `IBuilderFlag::kFP8`
  - `IBuilderFlag::kERROR_ON_TIMING_CACHE_MISS`
  - `IBuilderFlag::kBF16`
  - `IBuilderFlag::kDISABLE_COMPILATION_CACHE`

- `DataType::kBF16`
- `DataType::kINT64`
- `NetworkDefinitionCreationFlag::kSTRONGLY_TYPED`
- `nvonnxparser::ErrorCode::kUNSUPPORTED_NODE_ATTR`
- `nvonnxparser::ErrorCode::kUNSUPPORTED_NODE_INPUT`
- `nvonnxparser::ErrorCode::kUNSUPPORTED_NODE_DATATYPE`
- `nvonnxparser::ErrorCode::kUNSUPPORTED_NODE_DYNAMIC`
- `nvonnxparser::ErrorCode::kUNSUPPORTED_NODE_SHAPE`

- The following Python API classes were added:
  - `IProgressMonitor`
- The following Python API methods were added:
  - `INetworkDefinition.add_fill(shape, op, output_type)`
  - `INetworkDefinition.add_dequantize(input, scale, output_type)`
  - `INetworkDefinition.add_quantize(input, scale, output_type)`
  - `INetworkDefinition.get_flag`
  - `IRefitter.set_named_weights(name, weights, location)`
  - `IRefitter.get_named_weights(weights_name)`
  - `IRefitter.get_weights_location(weights_name)`
  - `IRefitter.unset_named_weights(weights_name)`
  - `IRefitter.refit_cuda_engine_async(stream_handle)`
- The following Python API attributes were added:
  - `IBuilder.progress_monitor`
  - `IFillLayer.is_alpha_beta_int64`
  - `IFillLayer.to_type`
  - `IQuantizeLayer.to_type`
  - `IDequantizeLayer.to_type`
  - `INetworkDefinition.flags`
  - `IRefitter.weights_validation`
  - `IParserError.node_name`
  - `IParserError.node_operator`
- The following Python API enums were added:
  - `IBuilderFlag.FP8`
  - `IBuilderFlag.ERROR_ON_TIMING_CACHE_MISS`
  - `IBuilderFlag.BF16`
  - `IBuilderFlag.DISABLE_COMPILATION_CACHE`

- `DataType.BF16`
- `DataType.INT64`
- `NetworkDefinitionCreationFlag.STRONGLY_TYPED`
- `ErrorCode.UNSUPPORTED_NODE_ATTR`
- `ErrorCode.UNSUPPORTED_NODE_INPUT`
- `ErrorCode.UNSUPPORTED_NODE_DATATYPE`
- `ErrorCode.UNSUPPORTED_NODE_DYNAMIC`
- `ErrorCode.kUNSUPPORTED_NODE_SHAPE`

- The `IEngineInspector` now prints more detailed layer information for LSTMs and Transformers networks.

Breaking API Changes

- > **!** ATTENTION: In TensorRT 9.0, due to the introduction of INT64 as a supported data type, ONNX models with INT64 I/O require INT64 bindings. Note that prior to this release, such models required INT32 bindings.

- In TensorRT 9.0, we removed `ICaffeParser`, `IUffParser`, and related classes and functions. The following APIs are removed:
  - `nvcaffeparser1::IBlobNameToTensor`
  - `nvcaffeparser1::IBinaryProtoBlob`
  - `nvcaffeparser1::IPluginFactoryV2`
  - `nvcaffeparser1::ICaffeParser`
  - `nvcaffeparser1::createCaffeParser`
  - `nvcaffeparser1::shutdownProtobufLibrary`
  - `createNvCaffeParser_INTERNAL`
  - `nvinfer1::utils::reshapeWeights`
  - `nvinfer1::utils::reorderSubBuffers`
  - `nvinfer1::utils::ransposeSubBuffers`
  - `nvuffparser::UffInputOrder`
  - `nvuffparser::FieldType`
  - `nvuffparser::FieldMap`
  - `nvuffparser::FieldCollection`
  - `nvuffparser::IUffParser`
  - `nvuffparser::createUffParser`
  - `nvuffparser::shutdownProtobufLibrary`
  - `createNvUffParser_INTERNAL`

- With removal of `ICaffeParser` and `IUffParsers`, the `libnvparsers` library is removed.
- `uff`, `graphsurgeon`, and related networks are removed from TensorRT packages.

Deprecated API Lifetime

- APIs deprecated in TensorRT 9.1 will be retained until at least 10/2024.
- APIs deprecated in TensorRT 9.0 will be retained until at least 8/2024.
- APIs deprecated in TensorRT 8.6 will be retained until at least 2/2024.
- APIs deprecated in TensorRT 8.5 will be retained until at least 9/2023.
- APIs deprecated in TensorRT 8.4 or before will be removed in TensorRT 10.0.

Refer to the API documentation (C++, Python) for how to update your code to remove the use of deprecated features.

Compatibility

- TensorRT 9.1.0 has been tested with the following:
  - cuDNN 8.9.5
  - TensorFlow 2.12.0
  - PyTorch >= 2.0 (refer to the `requirements.txt` file for each sample)
  - ONNX 1.14.0
- This TensorRT release supports CUDA®:
  - 12.2 update 1
  - 12.1 update 1
  - 12.0 update 1
  - 11.8
  - 11.7 update 1
  - 11.6 update 2
  - 11.5 update 2
  - 11.4 update 4
  - 11.3 update 1
  - 11.2 update 2
  - 11.1 update 1
  - 11.0 update 1
- This TensorRT release requires at least NVIDIA driver r450 on Linux or r452 on Windows as required by CUDA 11.0, which is the minimum CUDA version supported by this TensorRT release.
- It is suggested that you use TensorRT with a software stack that has been tested; including cuDNN and cuBLAS versions as documented in the Features For Platforms

And Software section. Other semantically compatible releases of cuDNN and cuBLAS can be used; however, other versions may have performance improvements as well as regressions. In rare cases, functional regressions might also be observed.

Limitations

- There are two modes of DLA softmax where the mode is chosen automatically based on the shape of the input tensor, where:
  - the first mode triggers when all nonbatch, non-axis dimensions are 1, and
  - the second mode triggers in other cases if valid.

  The second of the two modes is supported only for DLA 3.9.0 and later. It involves approximations that may result in errors of a small degree. Also, batch size greater than 1 is supported only for DLA 3.9.0 and later. Refer to DLA Supported Layers for more information.
- On QNX, networks that are segmented into a large number of DLA loadables may fail during inference.
- You may encounter an error such as, "*`Unable to load library: nvinfer_builder_resource.dll`*", if using Python 3.9.10 on Windows. You can workaround this issue by downgrading to an earlier version of Python 3.9.
- The DLA compiler is capable of removing identity transposes, but it cannot fuse multiple adjacent transpose layers into a single transpose layer (likewise for reshape). For example, given a TensorRT `IShuffleLayer` consisting of two non-trivial transposes and an identity reshapes in between. The shuffle layer is translated into two consecutive DLA transpose layers, unless you merge the transposes together manually in the model definition in advance.
- In explicitly quantized networks, a group convolution that has a Q/DQ pair before but no Q/DQ pair after is expected to run with INT8-IN-FP32-OUT mixed precision. However, on NVIDIA Hopper™ it may fall back to FP32-IN-FP32-OUT if the input channel count is small. This will be fixed in a future release.
- On PowerPC platforms, samples that depend on TensorFlow, ONNX Runtime, and PyTorch are unable to run due to missing Python module dependencies. These frameworks have not been built for PowerPC and/or published to standard repositories.
- `nvinfer1::UnaryOperation::kROUND` or `nvinfer1::UnaryOperation::kSIGN` operations of `IUnaryLayer` are not supported in the implicit batch mode.
- For networks containing normalization layers, particularly if deploying with mixed precision, target the latest ONNX opset that contains the corresponding function ops, for example: opset 17 for LayerNormalization or opset 18 GroupNormalization. Numerical accuracy using function ops is superior to corresponding implementation with primitive ops for normalization layers.

- `QuantizeLayer` and `DequantizeLayer` only support FP32 scale and data, even when using ONNX opset 19. If the input is not FP32, you must add a `Cast` to FP32 on the input to `QuantizeLayer`, and a `Cast` from FP32 at the output of `DequantizeLayer`.
- `EngineInspector::getLayerInformation` may return incomplete JSON data for some engines produced by TensorRT 9.0. When this happens, TensorRT Engine Explorer cannot be used to analyze the engine or generate a graph of the engine layers.

Deprecated and Removed Features

The following features have been deprecated or removed in TensorRT 9.0. Some deprecations that were planned to be removed in 9.0, but have not yet been removed, may be removed in TensorRT 10.0.

- TensorRT samples and open source demos are no longer supported on Python < 3.8.
- Ubuntu 18.04 has reached end of life and is no longer supported by TensorRT starting with TensorRT 9.0.
- The following plugins were deprecated:
  - `BatchedNMS_TRT`
  - `BatchedNMSDynamic_TRT`
  - `BatchTilePlugin_TRT`
  - `Clip_TRT`
  - `CoordConvAC`
  - `CropAndResize`
  - `EfficientNMS_ONNX_TRT`
  - `CustomGeluPluginDynamic`
  - `LReLU_TRT`
  - `NMSDynamic_TRT`
  - `NMS_TRT`
  - `Normalize_TRT`
  - `Proposal`
  - `SingleStepLSTMPlugin`
  - `SpecialSlice_TRT`
  - `Split`
- The following C++ API classes were deprecated:
  - `NvUtils`
- The following C++ API methods were deprecated:
  - ```
    nvinfer1::INetworkDefinition::addFill(nvinfer1::Dims dimensions,
     nvinfer1::FillOperation
            op)
    ```
    ```

- Only the 2-parameter version of this function is deprecated.

- `nvinfer1::INetworkDefinition::addDequantize(nvinfer1::ITensor &input, nvinfer1::ITensor &scale)`
  - Only the 2-parameter version of this function is deprecated.
- `nvinfer1::INetworkDefinition::addQuantize(nvinfer1::ITensor &input, nvinfer1::ITensor &scale)`
  - Only the 2-parameter version of this function is deprecated.

- The following C++ API enums were deprecated:
  - `nvinfer1::TacticSource::kCUBLAS_LT`
  - `nvonnxparser::OnnxParserFlag::kNATIVE_INSTANCENORM`
- The following Python API methods were deprecated:
  - `INetworkDefinition.add_fill(shape, op)`
  - `INetworkDefinition.add_dequantize(input, scale)`
- The following Python API enums were deprecated:
  - `TacticSource.CUBLAS_LT`
  - `OnnxParserFlag.NATIVE_INSTANCENORM`

Fixed Issues

- There was an up to 9% performance regression compared to TensorRT 8.5 on Yolov3 batch size 1 in FP32 precision on NVIDIA Ada Lovelace GPUs. This issue has been fixed.
- There was an up to 13% performance regression compared to TensorRT 8.5 on GPT2 without kv-cache in FP16 precision when dynamic shapes are used on NVIDIA Volta and NVIDIA Ampere GPUs. The workaround was to set the `kFASTER_DYNAMIC_SHAPES_0805` preview flag to `false`. This issue has been fixed.
- For transformer-based networks such as BERT and GPT, TensorRT could consume CPU memory up to 2 times the model size during compilation plus any additional formats timed. This issue has been fixed.
- In some cases, when using the `OBEY_PRECISION_CONSTRAINTS` builder flag and the required type was set to FP32, the network could fail with a missing tactic due to an incorrect optimization converting the output of an operation to FP16. This could be resolved by removing the `OBEY_PRECISION_CONSTRAINTS` option. This issue has been fixed.
- TensorRT could fail on devices with small RAM and swap space. This could be resolved by ensuring the RAM and swap space is at least 7 times the size of the network. For example, at least 21 GB of combined CPU memory for a 3 GB network. This issue has been fixed.
- If an `IShapeLayer` was used to get the output shape of an `INonZeroLayer`, engine building would likely fail. This issue has been fixed.

- If a one-dimension INT8 input was used for a Unary or ElementWise operation, engine building would throw an internal error "`Could not find any implementation for node`". This issue has been fixed.
- Using the compute sanitizer racecheck tool could cause the process to be terminated unexpectedly. The root cause was a wrong false alarm. The issue could be bypassed with `--kernel-regex-exclude kns=scudnn_winograd`. This issue has been fixed.
- TensorRT in FP16 mode did not perform cast operations correctly when only the output types were set, but not the layer precisions. This issue has been fixed.
- There was a known functional issue (fails with a CUDA error during compilation) with networks using `ILoop` layers on the WSL platform. This issue has been fixed.
- When using DLA, INT8 convolutions followed by FP16 layers could cause accuracy degradation. In such cases, you could either change the convolution to FP16 or the subsequent layer to INT8. This issue has been fixed.
- Using the compute sanitizer tool from CUDA 12.0 could report a `cudaErrorIllegalInstruction` error on Hopper GPUs in unusual scenarios. This could be ignored. This issue has been fixed.
- There was an up to 53% engine build time increase for ViT networks in FP16 precision compared to TensorRT 8.6 on NVIDIA Ampere GPUs. This issue has been fixed.
- There was an up to 93% engine build time increase for BERT networks in FP16 precision compared to TensorRT 8.6 on NVIDIA Ampere GPUs. This issue has been fixed.
- There was an up to 22% performance drop for GPT2 networks in FP16 precision with kv-cache stacked together compared to TensorRT 8.6 on NVIDIA Ampere GPUs. This issue has been fixed.
- There was an up to 28% performance regression compared to TensorRT 8.5 on Transformer networks in FP16 precision on NVIDIA Volta GPUs, and up to 85% performance regression on NVIDIA Pascal GPUs. Disabling the `kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805` preview flag was a workaround. This issue has been fixed.
- There was an issue on NVIDIA A2 GPUs due to a known hang that can impact multiple networks. This issue has been fixed.
- When using hardware compatibility features, TensorRT would potentially fail while compiling transformer based networks such as BERT. This issue has been fixed.
- The ONNX parser incorrectly used the InstanceNormalization plugin when processing ONNX InstanceNormalization layers with FP16 scale and bias inputs, which led to unexpected NaN and infinite outputs. This issue has been fixed.
- There was an up to 12% performance drop for WaveRNN networks in TF32 precision compared to TensorRT 8.6 on NVIDIA Ampere GPUs.
- There was an up to 6% performance regression compared to TensorRT 8.6 on BART networks with kv-cache stacked together in FP16 precision on H100 GPUs.

- A RHEL/CentOS 7 RPM package built with CUDA 12.2 for cuDNN 8.9.4 was not published to the CUDA network repo at the time of the TensorRT 9.0 release. When installing TensorRT on RHEL/CentOS 7 using the CUDA network repo, the cuDNN 8.9.4 RPM package built with CUDA 11.8 was installed instead. This has been resolved and the missing cuDNN 8.9.4 RPM package for RHEL/CentOS 7 and CUDA 12.2 has been published.
- There was an up to 25% CPU memory usage increase for ViT networks when building the engine in FP16 precision compared to TensorRT 8.6 on NVIDIA Ampere GPUs. This issue has been fixed.
- For some Transformer models, including ViT, Swin-Transformer, and DETR, there was a performance drop in INT8 precision (including both explicit and implicit quantization) compared to FP16 precision. This issue has been fixed.
- There was an up to 10% performance drop for the SegResNet network compared to TensorRT 8.2 when running in FP16 precision on NVIDIA Ampere Architecture GPUs due to a cuDNN regression in the `InstanceNormalization` plug-in. This issue has been fixed.
- There was an up to 7% performance regression compared to TensorRT 8.5 on CortanaASR networks in FP16 precision on NVIDIA Volta GPUs. This issue has been fixed.
- For enqueue-bound workloads, the latency of the workloads could have been longer in Windows than in Linux operating systems. This issue has been fixed.
- Compared to TensorRT 8.6, TensorRT 9.0 had more aggressive multi-head attention (MHA) fusions. While this is beneficial in most cases, it caused up to 7% performance regression when the workload was too small. This issue has been fixed.
- There may have been minor performance regressions when running ONNX models with `InstanceNormalization` operators in version compatible mode. This issue has been fixed.
- There was an up to 8% performance regression compared to TensorRT 8.6 on PilotNet4 network on H100 GPUs. This issue has been fixed.
- There was an up to 6% performance regression compared to TensorRT 8.6 on the SqueezeNet network in TF32 precision on H100 GPUs. This issue has been fixed.
- TensorRT preserves precision for operations that are imported from ONNX models if using strongly typed networks.
- When running ONNX networks with `InstanceNormalization` operations, there could have been up to a 50% decrease in performance. This issue has been fixed.

Known Issues

Functional

- There is a known floating point exception when running BERT networks on H100 multi-instance GPU (MIG).

- TensorRT does not preserve precision for operations that are imported from ONNX models if using weakly typed networks.
- The compute sanitizer synccheck tool may report a false alarm when running conv and GEMM kernels with CGA size >= 4 on NVIDIA Hopper GPUs.
- The compute sanitizer initcheck tool may flag false positive `Uninitialized __global__ memory read` errors when running TensorRT applications on NVIDIA Hopper GPUs. These errors can be safely ignored and will be fixed in an upcoming CUDA release.
- When using cuDNN 8.9.2.26 and the TensorRT RNNv2 API, the compute sanitizer from CUDA Toolkit 12.2 may report race conditions in CUTLASS kernels.
- Multihead attention fusion might not happen and affect performance if the number of heads is small.
- If a ONNX model contains a `Range` operator and its `limit` input is a data-dependent tensor, engine building will likely fail.
- Hardware forward compatibility (HFC) is broken on L4T Concord for ViT, Swin-Transformers, and BERT networks in FP16 mode. A workaround is to only use FP32 mode on L4T Concord or turn off HFC.
- Compute Sanitizer from CUDA Toolkit 12.0/12.1 may report a false alarm about invalid memory access in `generatedNativePointwise` kernels.
- If a network has a tensor of type bool with an *implicitly* data-dependent shape, engine building will likely fail.
- There is an occurance of use-after-free in NVRTC that has been fixed in CUDA 12.1. When using NVRTC from CUDA 12.0 together with the TensorRT static library, you may encounter a crash in certain scenarios. Linking with the NVRTC and PTXJIT compiler from CUDA 12.1 or newer will resolve this issue.
- Although the version compatible runtime is optimized for efficiency, it may result in slower performance than the full runtime in certain use cases. Most networks can expect no more than a 10% slowdown when using a version-compatible engine compared to a version-locked engine. However, in some cases, a larger performance drop may occur. For example:
  - When running ResNet50_v2 with QAT, there may be up to a 22% decrease in performance.
  - When running DynUNet in FP16 precision, there may be up to a 32% decrease in performance.
- There are known issues reported by the Valgrind memory leak check tool when detecting potential memory leaks from TensorRT applications. The recommendation to suppress the issues is to provide a Valgrind suppression file with the following contents when running the Valgrind memory leak check tool. Add the option `--keep-debuginfo=yes` to the Valgrind command line to suppress these errors.

```
{
  Memory leak errors with dlopen.
```

```
   Memcheck:Leak
   match-leak-kinds: definite
   ...
   fun:*dlopen*
   ...
}
{
    Memory leak errors with nvrtc
    Memcheck:Leak
    match-leak-kinds: definite
    fun:malloc
    obj:*libnvrtc.so*
    ...
}
```

- SM 7.5 and earlier devices may not have INT8 implementations for all layers with Q/DQ nodes. In this case, you will encounter a `could not find any implementation` error while building your engine. To resolve this, remove the Q/DQ nodes, which quantize the failing layers.
- Installing the `cuda-compat-11-4` package may interfere with CUDA enhanced compatibility and cause TensorRT to fail even when the driver is r465. The workaround is to remove the `cuda-compat-11-4` package or upgrade the driver to r470. *(not applicable for Jetson platforms)*
- For some networks, using a batch size of 4096 may cause accuracy degradation on DLA.
- Hardware compatible engines built with CUDA versions older than 11.5 may crash during inference when run on a GPU with a compute capability lower than that of the GPU where the engine was built. A workaround is to build on the GPU with the lowest compute capability.

Performance

- There is an up to 20 second increase in engine building for some large language models (LLMs) on NVIDIA Ampere GPUs.
- There is an up to 6% performance drop for BERT networks in FP32 precision compared to TensorRT 9.0 on NVIDIA Volta GPUs.
- There are up to 21% peak GPU memory usage fluctuations when building the engine for the same network back to back due to different tactics being selected.
- There is an up to 11% performance drop for ViT networks in TF32 precision compared to TensorRT 9.0 on NVIDIA Ampere GPUs.
- There is an up to 12% performance drop for BERT networks in FP16 precision compared to TensorRT 9.0 on NVIDIA Ada Lovelace GPUs.
- There is an up to 2.5x build time increase compared to TensorRT 9.0 for certain Bert like models due to additional tactics available for evaluation.
- There is an up to 13% performance drop for the CortanaASR model on NVIDIA Ampere GPUs compared to TensorRT 8.5.
- There is a known performance regression in the grouped deconvolution layer due to disabling cuDNN tactics. In TensorRT 8.6, performance can be recovered by unsetting

`nvinfer1::PreviewFeature::kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805`. We will close the performance gap in a future release.

- There is an up to 27% performance drop for the SegResNet model on Ampere GPUs compared to TensorRT 8.6 EA. This drop can be avoided by enabling the `kVERSION_COMPATIBLE` flag in the ONNX parser.
- There is an up to 18% performance drop for the ShuffleNet model on A30/A40 compared to TensorRT 8.5.1.
- Convolution on a tensor with an *implicitly* data-dependent shape may run significantly slower than on other tensors of the same size. Refer to the Glossary for the definition of implicitly data-dependent shapes.
- For some Transformer models, including ViT, Swin-Transformer, and DETR, there is a performance drop in INT8 precision (including both explicit and implicit quantization) compared to FP16 precision.
- There is an up to 30% performance regression for LSTM variants with dynamic shapes. This issue can be resolved by disabling the `kFASTER_DYNAMIC_SHAPES_0805` preview feature in TensorRT 8.6.
- There is a known issue on H100 that may lead to GPU hang when running TensorRT with high persistentCache usage. Limit the usage to 40% of L2 cache size as a workaround.
- There is a known performance issue when running instance normalization layers on Arm Server Base System Architecture (SBSA).
- There is a performance drop when offloading a SoftMax layer to DLA on NVIDIA Orin as compared to when running the layer on a GPU, with a larger drop for larger batch sizes. As an example, FP16 AlexNet with batch size 16 shows 32% drop when the network runs on DLA as compared to when the last SoftMax layer runs on a GPU.
- There is a known issue with DLA clocks that requires users to reboot the system after changing the nvpmodel power mode or otherwise experience a performance drop. Refer to the L4T board support package Release Notes for details.
- There is an up to 5% performance drop for networks using sparsity in FP16 precision.
- H100 performance for some LSTMs in FP16 precision is not fully optimized. This will be improved in future TensorRT versions.
- There is an up to 6% performance regression compared to TensorRT 8.5 on OpenRoadNet in FP16 precision on NVIDIA A10 GPUs.
- There is an up to 23% performance regression compared to TensorRT 8.5 on LSTMs in FP32 precision when dynamic shapes are used on NVIDIA Turing GPUs. Set the `kFASTER_DYNAMIC_SHAPES_0805` preview flag to `false` as a workaround.
- There is an up to 23% performance regression compared to TensorRT 8.5 on Temporal Fusion Transformers in FP32 precision on NVIDIA Turing and NVIDIA Ampere GPUs.

- A higher builder optimization level does not always give a better performance when compared to a lower builder optimization level; which may happen on all platforms and up to 27%. The workaround is to build an engine using a lower builder optimization level.
- There is an up to 8% performance regression compared to TensorRT 8.6 on PilotNet4 network on H100 GPUs. The regression will be fixed in a future TensorRT release.
- There is an 1x FP32 model CPU memory leak from `onnx.export` tracked in issue 106976 for the TensorRT open source software HuggingFace demo. You may encounter higher peak CPU memory usage for fresh runs, but will drop for cached-ONNX runs.
- There is an up to 6% performance regression compared to TensorRT 8.6 on the SqueezeNet network in TF32 precision on H100 GPUs.
- There is an up to 70% performance regression compared to TensorRT 8.6 on BERT networks in INT8 precision with FP16 disabled on L4 GPUs. To work around this, enable FP16 and disable INT8 in the builder config.
- Compared to TensorRT 8.6, TensorRT 9.0 has more aggressive multi-head attention (MHA) fusions. While this is beneficial in most cases, it causes up to 7% performance regression when the workload is too small. Increasing batch size would help improve the performance.

2.4. TensorRT Release 9.0.1

These are the TensorRT 9.0.1 Release Notes and are applicable to x86 Linux users. This release includes several fixes from the previous TensorRT releases as well as the following additional changes.

This GA release is for Large Language Models (LLMs) on A100, A10G, L4, L40, and H100 GPUs only. For applications other than LLMs, or other GPU platforms, continue to use TensorRT 8.6.1 for production use.

For previously released TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- TensorRT 9.0 provides enhanced support for Large Language Models (LLMs), including the following networks:
  - GPT
  - GPT-J
  - GPT-Neo
  - GPT-NeoX

  - BART
  - Bloom
  - Bloomz
  - OPT
  - T5
  - FLAN-T5
- Added support for `bfloat16` data types on NVIDIA Ampere GPUs and newer architectures.
- Added support for E4M3 FP8 data type on NVIDIA Hopper GPUs using explicit quantization. This allows utilizing TensorRT with TransformerEngine based FP8 models.
- Added the NeMo demo into the TensorRT OSS repository to demonstrate the performance benefit of using E4M3 FP8 data type with the GPT models trained with the NVIDIA NeMo Toolkit and TransformerEngine.
- Added `bfloat16` and FP8 I/O datatypes in plugins.
- Added support of networks running in INT8 precision where Smooth Quantization is used. Smooth Quantization is an approach that allows to improve accuracy of INT8 compute for LLM.
- Added support for INT64 data type. The ONNX parser no longer automatically casts INT64 to INT32.
- Added support for ONNX local functions when parsing ONNX models with the ONNX parser.
- Added support for caching JIT-compiled code. It can be disabled by setting `BuilderFlag::DISABLE_COMPILATION_CACHE`. The compilation cache is part of the timing cache, caches JIT-compiled code, and will be serialized as part of the timing cache by default, which may significantly increase the cache size.
- Added `NetworkDefinitionCreationFlag::kSTRONGLY_TYPED`. A strongly typed network faithfully translates the type specification of the network definition to the built engine.
- Added two new TensorRT samples; sampleProgressMonitor (C++) and simple_progress_reporter (Python) that are examples for using Progress Monitor during engine build.
- The following C++ API classes were added:
  - `IProgressMonitor`
- The following C++ API methods were added:
  - `INetworkDefinition::IFillLayer* addFill(Dims dimensions, FillOperation op, DataType outputType)`
  - `INetworkDefinition::IQuantizeLayer* addQuantize(ITensor& input, ITensor& scale, DataType`

```
        outputType)
```

- ```
  INetworkDefinition::IDequantizeLayer* addDequantize(ITensor& input, ITensor&
   scale,
          DataType outputType)
  ```
- `IBuilder::setProgressMonitor(IProgressMonitor* monitor)`
- `IBuilder::getProgressMonitor()`
- `IFillLayer::setAlphaInt64(int64_t alpha)`
- `IFillLayer::getAlphaInt64()`
- `IFillLayer::setBetaInt64(int64_t beta)`
- `IFillLayer::getBetaInt64()`
- `IFillLayer::isAlphaBetaInt64()`
- `IFillLayer::getToType()`
- `IFillLayer::setToType(DataType toType)`
- `IQuantizeLayer::getToType()`
- `IQuantizeLayer::setToType(DataType toType)`
- `IDequantizeLayer::getToType()`
- `IDequantizeLayer::setToType(DataType toType)`
- `INetworkDefinition::getFlags()`
- ```
  INetworkDefinition::getFlag(NetworkDefinitionCreationFlag
          networkDefinitionCreationFlag)
  ```

- The following C++ API enums were added:
  - `IBuilderFlag::kFP8`
  - `IBuilderFlag::kERROR_ON_TIMING_CACHE_MISS`
  - `IBuilderFlag::kBF16`
  - `IBuilderFlag::kDISABLE_COMPILATION_CACHE`
  - `DataType::kBF16`
  - `DataType::kINT64`
  - `NetworkDefinitionCreationFlag::kSTRONGLY_TYPED`
- The following Python API classes were added:
  - `IProgressMonitor`
- The following Python API methods were added:
  - `INetworkDefinition.add_fill(shape, op, output_type)`
  - `INetworkDefinition.add_dequantize(input, scale, output_type)`
  - `INetworkDefinition.add_quantize(input, scale, output_type)`
  - `INetworkDefinition.get_flag`
- The following Python API attributes were added:
  - `IBuilder.progress_monitor`
  - `IFillLayer.is_alpha_beta_int64`

- `IFillLayer.to_type`
- `IQuantizeLayer.to_type`
- `IDequantizeLayer.to_type`
- `INetworkDefinition.flags`

- The following Python API enums were added:
  - `IBuilderFlag.FP8`
  - `IBuilderFlag.ERROR_ON_TIMING_CACHE_MISS`
  - `IBuilderFlag.BF16`
  - `IBuilderFlag.DISABLE_COMPILATION_CACHE`
  - `DataType.BF16`
  - `DataType.INT64`
  - `NetworkDefinitionCreationFlag.STRONGLY_TYPED`
- The `IEngineInspector` now prints more detailed layer information for LSTMs and Transformers networks.

Breaking API Changes

- **!** ATTENTION: In TensorRT 9.0, due to the introduction of INT64 as a supported data type, ONNX models with INT64 I/O require INT64 bindings. Note that prior to this release, such models required INT32 bindings.
- In TensorRT 9.0 we are removing `ICaffeParser`, `IUffParser`, and related classes and functions. The following APIs are removed:
  - `nvcaffeparser1::IBlobNameToTensor`
  - `nvcaffeparser1::IBinaryProtoBlob`
  - `nvcaffeparser1::IPluginFactoryV2`
  - `nvcaffeparser1::ICaffeParser`
  - `nvcaffeparser1::createCaffeParser`
  - `nvcaffeparser1::shutdownProtobufLibrary`
  - `createNvCaffeParser_INTERNAL`
  - `nvinfer1::utils::reshapeWeights`
  - `nvinfer1::utils::reorderSubBuffers`
  - `nvinfer1::utils::ransposeSubBuffers`
  - `nvuffparser::UffInputOrder`
  - `nvuffparser::FieldType`
  - `nvuffparser::FieldMap`
  - `nvuffparser::FieldCollection`
  - `nvuffparser::IUffParser`

- `nvuffparser::createUffParser`
- `nvuffparser::shutdownProtobufLibrary`
- `createNvUffParser_INTERNAL`

- With removal of `ICaffeParser` and `IUffParsers`, the `libnvparsers` library is removed.
- `uff`, `graphsurgeon`, and related networks are removed from TensorRT packages.

Deprecated API Lifetime

- APIs deprecated in TensorRT 9.0 will be retained until at least 8/2024.
- APIs deprecated in TensorRT 8.6 will be retained until at least 2/2024.
- APIs deprecated in TensorRT 8.5 will be retained until at least 9/2023.
- APIs deprecated in TensorRT 8.4 or before will be removed in TensorRT 10.0.

Refer to the API documentation (C++, Python) for how to update your code to remove the use of deprecated features.

Compatibility

- TensorRT 9.0.1 has been tested with the following:
  - cuDNN 8.9.4
  - TensorFlow 2.12.0
  - PyTorch >= 2.0 (refer to the `requirements.txt` file for each sample)
  - ONNX 1.14.0
- This TensorRT release supports CUDA®:
  - 12.2 update 1
  - 12.1 update 1
  - 12.0 update 1
  - 11.8
  - 11.7 update 1
  - 11.6 update 2
  - 11.5 update 2
  - 11.4 update 4
  - 11.3 update 1
  - 11.2 update 2
  - 11.1 update 1
  - 11.0 update 1

- This TensorRT release requires at least NVIDIA driver r450 on Linux or r452 on Windows as required by CUDA 11.0, which is the minimum CUDA version supported by this TensorRT release.
- It is suggested that you use TensorRT with a software stack that has been tested; including cuDNN and cuBLAS versions as documented in the Features For Platforms And Software section. Other semantically compatible releases of cuDNN and cuBLAS can be used; however, other versions may have performance improvements as well as regressions. In rare cases, functional regressions might also be observed.

Limitations

- There are two modes of DLA softmax where the mode is chosen automatically based on the shape of the input tensor, where:
  - the first mode triggers when all nonbatch, non-axis dimensions are 1, and
  - the second mode triggers in other cases if valid.

  The second of the two modes is supported only for DLA 3.9.0 and later. It involves approximations that may result in errors of a small degree. Also, batch size greater than 1 is supported only for DLA 3.9.0 and later. Refer to DLA Supported Layers for more information.
- On QNX, networks that are segmented into a large number of DLA loadables may fail during inference.
- You may encounter an error such as, "*`Unable to load library: nvinfer_builder_resource.dll`*", if using Python 3.9.10 on Windows. You can workaround this issue by downgrading to an earlier version of Python 3.9.
- The DLA compiler is capable of removing identity transposes, but it cannot fuse multiple adjacent transpose layers into a single transpose layer (likewise for reshape). For example, given a TensorRT `IShuffleLayer` consisting of two non-trivial transposes and an identity reshapes in between. The shuffle layer is translated into two consecutive DLA transpose layers, unless you merge the transposes together manually in the model definition in advance.
- In explicitly quantized networks, a group convolution that has a Q/DQ pair before but no Q/DQ pair after is expected to run with INT8-IN-FP32-OUT mixed precision. However, on NVIDIA Hopper™ it may fall back to FP32-IN-FP32-OUT if the input channel count is small. This will be fixed in a future release.
- On PowerPC platforms, samples that depend on TensorFlow, ONNX Runtime, and PyTorch are unable to run due to missing Python module dependencies. These frameworks have not been built for PowerPC and/or published to standard repositories.
- `nvinfer1::UnaryOperation::kROUND` or `nvinfer1::UnaryOperation::kSIGN` operations of `IUnaryLayer` are not supported in the implicit batch mode.

- For networks containing normalization layers, particularly if deploying with mixed precision, target the latest ONNX opset that contains the corresponding function ops, for example: opset 17 for LayerNormalization or opset 18 GroupNormalization. Numerical accuracy using function ops is superior to corresponding implementation with primitive ops for normalization layers.
- `QuantizeLayer` and `DequantizeLayer` only support FP32 scale and data, even when using ONNX opset 19. If the input is not FP32, you must add a `Cast` to FP32 on the input to `QuantizeLayer`, and a `Cast` from FP32 at the output of `DequantizeLayer`.
- `EngineInspector::getLayerInformation` may return incomplete JSON data for some engines produced by TensorRT 9.0. When this happens, TensorRT Engine Explorer cannot be used to analyze the engine or generate a graph of the engine layers.

Deprecated and Removed Features

The following features have been deprecated or removed in TensorRT 9.0. Some deprecations that were planned to be removed in 9.0, but have not yet been removed, may be removed in TensorRT 10.0.

- Ubuntu 18.04 has reached end of life and is no longer supported by TensorRT starting with 9.0.
- The following plugins were deprecated:
  - `BatchedNMS_TRT`
  - `BatchedNMSDynamic_TRT`
  - `BatchTilePlugin_TRT`
  - `Clip_TRT`
  - `CoordConvAC`
  - `CropAndResize`
  - `EfficientNMS_ONNX_TRT`
  - `CustomGeluPluginDynamic`
  - `LReLU_TRT`
  - `NMSDynamic_TRT`
  - `NMS_TRT`
  - `Normalize_TRT`
  - `Proposal`
  - `SingleStepLSTMPlugin`
  - `SpecialSlice_TRT`
  - `Split`
- The following C++ API classes were deprecated:
  - `NvUtils`

- The following C++ API methods were deprecated:
  - ```
    nvinfer1::INetworkDefinition::addFill(nvinfer1::Dims dimensions,
     nvinfer1::FillOperation
           op)
    ```
    - Only the 2-parameter version of this function is deprecated.
  - ```
    nvinfer1::INetworkDefinition::addDequantize(nvinfer1::ITensor &input,
     nvinfer1::ITensor
           &scale)
    ```
    - Only the 2-parameter version of this function is deprecated.
  - ```
    nvinfer1::INetworkDefinition::addQuantize(nvinfer1::ITensor &input,
     nvinfer1::ITensor
           &scale)
    ```
    - Only the 2-parameter version of this function is deprecated.
- The following C++ API enums were deprecated:
  - `nvinfer1::TacticSource::kCUBLAS_LT`
  - `nvonnxparser::OnnxParserFlag::kNATIVE_INSTANCENORM`
- The following Python API methods were deprecated:
  - `INetworkDefinition.add_fill(shape, op)`
  - `INetworkDefinition.add_dequantize(input, scale)`
- The following Python API enums were deprecated:
  - `TacticSource.CUBLAS_LT`
  - `OnnxParserFlag.NATIVE_INSTANCENORM`

Fixed Issues

- There was an up to 9% performance regression compared to TensorRT 8.5 on Yolov3 batch size 1 in FP32 precision on NVIDIA Ada Lovelace GPUs. This issue has been fixed.
- There was an up to 13% performance regression compared to TensorRT 8.5 on GPT2 without kv-cache in FP16 precision when dynamic shapes are used on NVIDIA Volta and NVIDIA Ampere GPUs. The workaround was to set the `kFASTER_DYNAMIC_SHAPES_0805` preview flag to `false`. This issue has been fixed.
- For transformer-based networks such as BERT and GPT, TensorRT could consume CPU memory up to 2 times the model size during compilation plus any additional formats timed. This issue has been fixed.
- In some cases, when using the `OBEY_PRECISION_CONSTRAINTS` builder flag and the required type was set to FP32, the network could fail with a missing tactic due to an incorrect optimization converting the output of an operation to FP16. This could be resolved by removing the `OBEY_PRECISION_CONSTRAINTS` option. This issue has been fixed.
- TensorRT could fail on devices with small RAM and swap space. This could be resolved by ensuring the RAM and swap space is at least 7 times the size of the network. For
    ```

example, at least 21 GB of combined CPU memory for a 3 GB network. This issue has been fixed.

- If an `IShapeLayer` was used to get the output shape of an `INonZeroLayer`, engine building would likely fail. This issue has been fixed.
- If a one-dimension INT8 input was used for a Unary or ElementWise operation, engine building would throw an internal error `"Could not find any implementation for node"`. This issue has been fixed.
- Using the compute sanitizer racecheck tool could cause the process to be terminated unexpectedly. The root cause was a wrong false alarm. The issue could be bypassed with `--kernel-regex-exclude kns=scudnn_winograd`. This issue has been fixed.
- TensorRT in FP16 mode did not perform cast operations correctly when only the output types were set, but not the layer precisions. This issue has been fixed.
- There was a known functional issue (fails with a CUDA error during compilation) with networks using `ILoop` layers on the WSL platform. This issue has been fixed.
- When using DLA, INT8 convolutions followed by FP16 layers could cause accuracy degradation. In such cases, you could either change the convolution to FP16 or the subsequent layer to INT8. This issue has been fixed.
- Using the compute sanitizer tool from CUDA 12.0 could report a `cudaErrorIllegalInstruction` error on Hopper GPUs in unusual scenarios. This could be ignored. This issue has been fixed.
- There was an up to 53% engine build time increase for ViT networks in FP16 precision compared to TensorRT 8.6 on NVIDIA Ampere GPUs. This issue has been fixed.
- There was an up to 93% engine build time increase for BERT networks in FP16 precision compared to TensorRT 8.6 on NVIDIA Ampere GPUs. This issue has been fixed.
- There was an up to 22% performance drop for GPT2 networks in FP16 precision with kv-cache stacked together compared to TensorRT 8.6 on NVIDIA Ampere GPUs. This issue has been fixed.
- There was an up to 28% performance regression compared to TensorRT 8.5 on Transformer networks in FP16 precision on NVIDIA Volta GPUs, and up to 85% performance regression on NVIDIA Pascal GPUs. Disabling the `kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805` preview flag was a workaround. This issue has been fixed.
- There was an issue on NVIDIA A2 GPUs due to a known hang that can impact multiple networks. This issue has been fixed.
- When using hardware compatibility features, TensorRT would potentially fail while compiling transformer based networks such as BERT. This issue has been fixed.
- The ONNX parser incorrectly used the InstanceNormalization plugin when processing ONNX InstanceNormalization layers with FP16 scale and bias inputs, which led to unexpected NaN and infinite outputs. This issue has been fixed.

Known Issues

Functional

- A RHEL/CentOS 7 RPM package built with CUDA 12.2 for cuDNN 8.9.4 has not been published to the CUDA network repo. When installing TensorRT 9.0 on RHEL/CentOS 7 using the CUDA network repo, the cuDNN 8.9.4 RPM package built with CUDA 11.8 will be installed instead. Often, this scenario remains functional, but it's undefined behavior to mix libraries with different CUDA major versions. This will be resolved soon once the missing cuDNN 8.9.4 RPM package for RHEL/CentOS 7 and CUDA 12.2 has been published.
- The compute sanitizer synccheck tool may report a false alarm when running conv and GEMM kernels with CGA size >= 4 on NVIDIA Hopper GPUs.
- The compute sanitizer initcheck tool may flag false positive `Uninitialized __global__ memory read` errors when running TensorRT applications on NVIDIA Hopper GPUs. These errors can be safely ignored and will be fixed in an upcoming CUDA release.
- When using cuDNN 8.9.2.26 and the TensorRT RNNv2 API, the compute sanitizer from CUDA Toolkit 12.2 may report race conditions in CUTLASS kernels.
- Multihead attention fusion might not happen and affect performance if the number of heads is small.
- If a ONNX model contains a `Range` operator and its `limit` input is a data-dependent tensor, engine building will likely fail.
- Hardware forward compatibility (HFC) is broken on L4T Concord for ViT, Swin-Transformers, and BERT networks in FP16 mode. A workaround is to only use FP32 mode on L4T Concord or turn off HFC.
- Compute Sanitizer from CUDA Toolkit 12.0/12.1 may report a false alarm about invalid memory access in `generatedNativePointwise` kernels.
- If a network has a tensor of type bool with an *implicitly* data-dependent shape, engine building will likely fail.
- There is an occurance of use-after-free in NVRTC that has been fixed in CUDA 12.1. When using NVRTC from CUDA 12.0 together with the TensorRT static library, you may encounter a crash in certain scenarios. Linking with the NVRTC and PTXJIT compiler from CUDA 12.1 or newer will resolve this issue.
- Although the version compatible runtime is optimized for efficiency, it may result in slower performance than the full runtime in certain use cases. Most networks can expect no more than a 10% slowdown when using a version-compatible engine compared to a version-locked engine. However, in some cases, a larger performance drop may occur. For example:
  - When running ResNet50_v2 with QAT, there may be up to a 22% decrease in performance.

  - When running DynUNet in FP16 precision, there may be up to a 32% decrease in performance.
  - When running ONNX networks with `InstanceNormalization` operations, there may be up to a 50% decrease in performance.
- There are known issues reported by the Valgrind memory leak check tool when detecting potential memory leaks from TensorRT applications. The recommendation to suppress the issues is to provide a Valgrind suppression file with the following contents when running the Valgrind memory leak check tool. Add the option `--keep-debuginfo=yes` to the Valgrind command line to suppress these errors.

```
{
  Memory leak errors with dlopen.
   Memcheck:Leak
   match-leak-kinds: definite
   ...
   fun:*dlopen*
   ...
}
{
    Memory leak errors with nvrtc
    Memcheck:Leak
    match-leak-kinds: definite
    fun:malloc
    obj:*libnvrtc.so*
    ...
}
```

- SM 7.5 and earlier devices may not have INT8 implementations for all layers with Q/DQ nodes. In this case, you will encounter a `could not find any implementation` error while building your engine. To resolve this, remove the Q/DQ nodes, which quantize the failing layers.
- TensorRT does not preserve precision for operations that are imported from ONNX models in FP16 mode.
- Installing the `cuda-compat-11-4` package may interfere with CUDA enhanced compatibility and cause TensorRT to fail even when the driver is r465. The workaround is to remove the `cuda-compat-11-4` package or upgrade the driver to r470. *(not applicable for Jetson platforms)*
- For some networks, using a batch size of 4096 may cause accuracy degradation on DLA.
- Hardware compatible engines built with CUDA versions older than 11.5 may crash during inference when run on a GPU with a compute capability lower than that of the GPU where the engine was built. A workaround is to build on the GPU with the lowest compute capability.
- When enabling the cuDNN tactic source manually, there is a potential memory leak from the cuDNN library. This issue will be fixed in a future cuDNN release.

Performance

- There is an up to 12% performance drop for WaveRNN networks in TF32 precision compared to TensorRT 8.6 on NVIDIA Ampere GPUs.

- There is an up to 25% CPU memory usage increase for ViT networks when building the engine in FP16 precision compared to TensorRT 8.6 on NVIDIA Ampere GPUs.
- There is an up to 13% performance drop for the CortanaASR model on NVIDIA Ampere GPUs compared to TensorRT 8.5.
- There is a known performance regression in the grouped deconvolution layer due to disabling cuDNN tactics. In TensorRT 8.6, performance can be recovered by unsetting `nvinfer1::PreviewFeature::kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805`. We will close the performance gap in a future release.
- There is an up to 27% performance drop for the SegResNet model on Ampere GPUs compared to TensorRT 8.6 EA. This drop can be avoided by enabling the `kVERSION_COMPATIBLE` flag in the ONNX parser.
- There is an up to 18% performance drop for the ShuffleNet model on A30/A40 compared to TensorRT 8.5.1.
- There may be minor performance regressions when running ONNX models with `InstanceNormalization` operators in version compatible mode. Refer to the NVIDIA TensorRT Developer Guide for more information.
- Convolution on a tensor with an *implicitly* data-dependent shape may run significantly slower than on other tensors of the same size. Refer to the Glossary for the definition of implicitly data-dependent shapes.
- For some Transformer models, including ViT, Swin-Transformer, and DETR, there is a performance drop in INT8 precision (including both explicit and implicit quantization) compared to FP16 precision.
- There is an up to 30% performance regression for LSTM variants with dynamic shapes. This issue can be resolved by disabling the `kFASTER_DYNAMIC_SHAPES_0805` preview feature in TensorRT 8.6.
- There is a known issue on H100 that may lead to GPU hang when running TensorRT with high persistentCache usage. Limit the usage to 40% of L2 cache size as a workaround.
- There is a known performance issue when running instance normalization layers on Arm Server Base System Architecture (SBSA).
- There is an up to 10% performance drop for the SegResNet network compared to TensorRT 8.2 when running in FP16 precision on NVIDIA Ampere architecture GPUs due to a cuDNN regression in the `InstanceNormalization` plug-in. This will be fixed in a future TensorRT release. You can work around the regression by reverting the cuDNN version to cuDNN 8.2.1.
- There is a performance drop when offloading a SoftMax layer to DLA on NVIDIA Orin as compared to when running the layer on a GPU, with a larger drop for larger batch sizes. As an example, FP16 AlexNet with batch size 16 shows 32% drop when the network runs on DLA as compared to when the last SoftMax layer runs on a GPU.

- There is a known issue with DLA clocks that requires users to reboot the system after changing the nvpmodel power mode or otherwise experience a performance drop. Refer to the L4T board support package Release Notes for details.
- There is an up to 5% performance drop for networks using sparsity in FP16 precision.
- H100 performance for some LSTMs in FP16 precision is not fully optimized. This will be improved in future TensorRT versions.
- There is an up to 6% performance regression compared to TensorRT 8.5 on OpenRoadNet in FP16 precision on NVIDIA A10 GPUs.
- There is an up to 7% performance regression compared to TensorRT 8.5 on CortanaASR networks in FP16 precision on NVIDIA Volta GPUs. Disable the `kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805` preview flag as a workaround.
- There is an up to 23% performance regression compared to TensorRT 8.5 on LSTMs in FP32 precision when dynamic shapes are used on NVIDIA Turing GPUs. Set the `kFASTER_DYNAMIC_SHAPES_0805` preview flag to `false` as a workaround.
- There is an up to 23% performance regression compared to TensorRT 8.5 on Temporal Fusion Transformers in FP32 precision on NVIDIA Turing and NVIDIA Ampere GPUs.
- A higher builder optimization level does not always give a better performance when compared to a lower builder optimization level; which may happen on all platforms and up to 27%. The workaround is to build an engine using a lower builder optimization level.
- There is an up to 8% performance regression compared to TensorRT 8.6 on PilotNet4 network on H100 GPUs. The regression will be fixed in a future TensorRT release.
- There is an 1x FP32 model CPU memory leak from `onnx.export` tracked in issue 106976 for the TensorRT open source software HuggingFace demo. You may encounter higher peak CPU memory usage for fresh runs, but will drop for cached-ONNX runs.
- For enqueue-bound workloads, the latency of the workloads may be longer in Windows than in Linux operating systems. Upgrade the driver or use CUDA graph to work around the issue. Refer to the documentation about enqueue-bound workloads for more detailed information.
- There is an up to 6% performance regression compared to TensorRT 8.6 on the SqueezeNet network in TF32 precision on H100 GPUs.
- There is an up to 6% performance regression compared to TensorRT 8.6 on BART networks with kv-cache stacked together in FP16 precision on H100 GPUs.
- There is an up to 70% performance regression compared to TensorRT 8.6 on BERT networks in INT8 precision with FP16 disabled on L4 GPUs. To work around this, enable FP16 and disable INT8 in the builder config.
- Compared to TensorRT 8.6, TensorRT 9.0 has more aggressive multi-head attention (MHA) fusions. While this is beneficial in most cases, it causes up to 7% performance

regression when the workload is too small. Increasing batch size would help improve the performance.

Chapter 3. TensorRT Release 8.x.x

3.1. TensorRT Release 8.6.1

These are the TensorRT 8.6.1 Release Notes and are applicable to x86 Linux and Windows users. This release incorporates Arm®-based CPU cores for Server Base System Architecture (SBSA) users on Linux only. This release includes several fixes from the previous TensorRT releases as well as the following additional changes.

These Release Notes are applicable to workstation, server, and NVIDIA JetPack™ users unless appended specifically with *(not applicable for Jetson platforms)*.

For previously released TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Announcements

- In TensorRT 8.6, cuDNN, cuBLAS, and cuBLASLt tactic sources are turned off by default in builder profiling. TensorRT plans to remove the cuDNN, cuBLAS, and cuBLASLt dependency in future releases. If there are critical regressions, set the `PreviewFeature` flag `kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805` to false to re-enable cuDNN, cuBLAS, and cuBLASLt.
- Stubbed static libraries for cuDNN, cuBLAS, and cuBLASLt are provided in `$(TRT_LIB_DIR)/stubs`. When statically linking TensorRT with no requirement for cuDNN, cuBLAS, or cuBLASLt, the stubbed library can be used to reduce CPU and GPU memory usage.
- Debian and RPM packages are now using 4-component versioning instead of 3 to better identify differences between TensorRT builds.
- The tar and zip filenames no longer contain the cuDNN version due to cuDNN no longer being a primary tactic source for TensorRT. cuDNN has a lesser impact on TensorRT's performance than in previous releases where the cuDNN version was prominent in the package filename.
- The TensorRT Python Package Index installation has been split into multiple modules:
  - TensorRT libraries (`tensorrt_libs`)

- Python bindings matching the Python version in use (`tensorrt_bindings`)
- Frontend source package, which pulls in the correct version of dependent TensorRT modules from pypi.nvidia.com (`tensorrt`)

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- Python 3.11 is now supported starting with TensorRT 8.6 GA.
- CUDA 12.x is now supported starting with the TensorRT 8.6 release. CUDA 11.x builds from this release or a previous release are not compatible with libraries or applications compiled with CUDA 12.x and may lead to unexpected behavior.
- Added support for hardware compatibility. This allows an engine built on one GPU architecture to work on GPUs of other architectures. This feature is only supported for NVIDIA Ampere and newer architectures. Note that enabling hardware compatibility may result in a degradation in latency/throughput, and is therefore intended to be used primarily to ease the process of upgrading to new hardware. This feature is not supported on JetPack.
- Added the following layers:
  - `IReverseSequence` layer has been added to support the `ReverseSequence` operator in ONNX.
  - `INormalization` layer has been added to support `InstanceNormalization`, `GroupNormalization`, and `LayerNormalization` operations in ONNX.
- A new `nvinfer1::ICastLayer` interface was introduced, which provides a conversion of the data type of the input tensor between FP32, FP16, INT32, INT8, UINT8, and BOOL. The ONNX parser was updated to use `ICastLayer` instead of `IIdentityLayer` to implement cast.
- Introduced restricted runtime installation options when memory consumption is a high priority for deployment: lean or dispatch runtime mode.
  - Lean runtime installation: This installation is significantly smaller than the full installation and allows you to load and run engines that were built with a version compatible builder flag. This installation will not provide the functionality to build a TensorRT plan file.
  - Dispatch runtime installation: This installation allows for deployments with the minimum memory consumption and allows you to load and run engines that were built with a version compatible builder flag and include the lean runtime. This installation does not provide the functionality to build a TensorRT plan file.
  - Added version compatibility support to `trtexec`. Refer to the NVIDIA TensorRT Developer Guide for more information on the `trtexec` flags.

  For more information, refer to the NVIDIA TensorRT Installation Guide.

- By default, TensorRT engines are compatible only with the version of TensorRT with which they are built. Starting in TensorRT 8.6, with the appropriate build-time configuration, engines can be built that are compatible with other TensorRT minor versions within a major version. For more information, refer to Version Compatibility.

  Note: Use of certain version compatibility features requires explicit configuration of the TensorRT runtime to allow host executable code. For more information, refer to Runtime Options.

- Implemented the following performance enhancements:
  - Improved the Multi-Head Attention (MHA) fusions, speeding up Transformer-like networks.
  - Improved the engine building time and the performance of Transformer-like networks with dynamic shapes.
  - Avoided unnecessary `cuStreamSynchronize()` calls in `enqueueV2()` and `enqueueV3()` when running LSTMs or Transformer-like networks.
  - Improved the performance of various networks on NVIDIA Hopper GPUs.
- Added an optimization level builder flag, which allows TensorRT to spend more engine building time searching for better tactics, or to build the engine much faster by reducing the searching scope. For more information, refer to Builder Optimization Level.
- Added the multi-stream APIs, which allows users to control how many streams TensorRT can use to run different parts of the network in parallel, potentially resulting in better performance. For more information, refer to Within-Inference Multi-Streaming.
- *Experimental*. Extended DLA so that `IElementWiseLayer` now supports equal operation (`ElementWiseOperation::kEQUAL`). This is the first `ElementWise` logical operation that DLA supports. There are several restrictions and requirements imposed when adopting this operation in DLA. Refer to DLA Supported Layers for more information.
  - One such requirement is that you must explicitly set the device type of the `ElementWise` equal layer to `DLA`. To enable this feature on `trtexec`, it now supports a flag `--layerDeviceTypes` to let you explicitly specify the device type for individual layers. Refer to Commonly Used Command-line Flags for more information on the new flag.
- Added a new sample called onnx_custom_plugin, which demonstrates how to use plugins written in C++ to run TensorRT on ONNX models with custom or unsupported layers. For specifics about this sample, refer to the GitHub: /onnx_custom_plugin/README.md file for detailed information about how this sample works, sample code, and step-by-step instructions on how to run and verify its output.
- Made the following C++ API changes:

- Added classes:
  - `IReverseSequenceLayer`
  - `INormalizationLayer`
  - `ILoggerFinder`
- Added macros:
  - `NV_TENSORRT_RELEASE_TYPE`
  - `NV_TENSORRT_RELEASE_TYPE_EARLY_ACCESS`
  - `NV_TENSORRT_RELEASE_TYPE_RELEASE_CANDIDATE`
  - `NV_TENSORRT_RELEASE_TYPE_GENERAL_AVAILABILITY`
- Added functions:
  - `getBuilderSafePluginRegistry()`
  - `IAlgorithmIOInfo::getVectorizedDim()`
  - `IAlgorithmIOInfo::getComponentsPerElement()`
  - `IBuilderConfig::setBuilderOptimizationLevel()`
  - `IBuilderConfig::getBuilderOptimizationLevel()`
  - `IBuilderConfig::setHardwareCompatibilityLevel()`
  - `IBuilderConfig::getHardwareCompatibilityLevel()`
  - `IBuilderCOnfig::setPluginsToSerialize()`
  - `IBuilderConfig::getPluginToSerialize()`
  - `IBuilderConfig::getNbPluginsToSerialize()`
  - `IBuilderConfig::getMaxAuxStreams()`
  - `IBuilderConfig::setMaxAuxStreams()`
  - `IBuilder::getPluginRegistry()`
  - `ICudaEngine::getHardwareCompatibilityLevel()`
  - `ICudaEngine::getNbAuxStreams()`
  - `IExecutionContext::setAuxStreams()`
  - `ILayer::setMetadata()`
  - `ILayer::getMetadata()`
  - `INetworkDefinition::addCast()`
  - `INetworkDefinition::addNormalization()`
  - `INetworkDefinition::addReverseSequence()`
  - `INetworkDefinition::getBuilder()`
  - `IPluginRegistry::isParentSearchEnabled()`
  - `IPluginRegistry::setParentSearchEnabled()`
  - `IPluginRegistry::loadLibrary()`

        - `IPluginRegistry::deregisterLibrary()`
        - `IRuntime::setTemporaryDirectory()`
        - `IRuntime::getTemporaryDirectory()`
        - `IRuntime::setTempfileControlFlags()`
        - `IRuntime::getTempfileControlFlags()`
        - `IRuntime::getPluginRegistry()`
        - `IRuntime::setPluginRegistryParent()`
        - `IRuntime::loadRuntime()`
        - `IRuntime::setEngineHostCodeAllowed()`
        - `IRuntime::getEngineHostCodeAllowed()`
        - `ITopKLayer::setInput()`
    - Added enums:
        - `HardwareCompatibilityLevel`
        - `TempfileControlFlag`
    - Added enum values:
        - `BuilderFLag::kVERSION_COMPATIBLE`
        - `BuilderFlag::kEXCLUDE_LEAN_RUNTIME`
        - `DataType::kFP8`
        - `LayerType::kREVERSE_SEQUENCE`
        - `LayerType::kNORMALIZATION`
        - `LayerType::kCAST`
        - `MemoryPoolType::kTACTIC_DRAM`
        - `PreviewFeature::kPROFILE_SHARING_0806`
        - `TensorFormat::kDHWC`
        - `UnaryOperation::kISINF`
    - Deprecated enums:
        - `PreviewFeature::kFASTER_DYNAMIC_SHAPES`
    - Deprecated macros:
        - `NV_TENSORRT_SONAME_MAJOR`
        - `NV_TENSORRT_SONAME_MINOR`
        - `NV_TENSORRT_SONAME_PATCH`
    - Deprecated funtions:
        - `FieldMap::FieldMap()`
        - `IAlgorithmIOInfo::getTensorFormat()`
- Made the following Python API changes:

- Added classes:
  - `ICastLayer`
  - `IReverseSequenceLayer`
  - `INormalizationLayer`
- Added functions:
  - `IExecutionContext.set_aux_streams()`
  - `INetworkDefinition.add_cast()`
  - `INetworkDefinition.add_normalization()`
  - `INetworkDefinition.add_reverse_sequence()`
  - `IPluginRegistry.load_library()`
  - `IPluginRegistry.deregister_library()`
  - `IRuntime.load_runtime()`
  - `ITopKLayer.set_input()`
- Added properties:
  - `BuilderFlag.EXCLUDE_LEAN_RUNTIME`
  - `BuilderFlag.FP8`
  - `BuilderFlag.VERSION_COMPATIBLE`
  - `DataType.FP8`
  - `HardwareCompatibilityLevel.AMPERE_PLUS`
  - `HardwareCompatibilityLevel.NONE`
  - `IAlgorithmIOInfo.components_per_element`
  - `IAlgorithmIOInfo.vectorized_dim`
  - `IBuilder.get_plugin_registry`
  - `IBuilderConfig.builder_optimization_level`
  - `IBuilderConfig.hardware_compatibility_level`
  - `IBuilderConfig.max_aux_streams`
  - `IBuilderConfig.plugins_to_serialize`
  - `ICudaEngine.hardware_compatibility_level`
  - `ICudaEngine.num_aux_streams`
  - `ILayer.metadata`
  - `INetworkDefinition.builder`
  - `INormalizationLayer.axes`
  - `INormalizationLayer.compute_precision`
  - `INormalizationLayer.epsilon`
  - `INormalizationLayer.num_groups`

        - `IPluginRegistry.parent_search_enabled`
        - `IReverseSequenceLayer.batch_axis`
        - `IReverseSequenceLayer.sequence_axis`
        - `IRuntime.engine_host_code_allowed`
        - `IRuntime.load_runtime`
        - `IRuntime.tempfile_control_flags`
        - `IRuntime.temporary_directory`
        - `LayerType.CAST`
        - `LayerType.NORMALIZATION`
        - `LayerType.REVERSE_SEQUENCE`
        - `MemoryPoolType.TACTIC_DRAM`
        - `PreviewFeature.PROFILE_SHARING_0806`
        - `TempfileControlFlag.ALLOW_IN_MEMORY_FILES`
        - `TempfileControlFlag.ALLOW_TEMPORARY_FILES`
        - `TensorFormat.DHWC`
    - Added enums:
        - `HardwareCompatibilityLevel`
        - `TempfileControlFlag`
    - Added enum values:
        - `UnaryOperation.ISINF`
    - Deprecated enums:
        - `PreviewFeature.FASTER_DYNAMIC_SHAPES`
    - Deprecated properties:
        - `IAlgorithmIOInfo.tensor_format`

Deprecated API Lifetime

- APIs deprecated in TensorRT 8.6 will be retained until at least 2/2024.
- APIs deprecated in TensorRT 8.5 will be retained until at least 9/2023.
- APIs deprecated in TensorRT 8.4 will be retained until at least 2/2023.
- APIs deprecated before TensorRT 8.4 will be removed in TensorRT 9.0.

Refer to the API documentation (C++, Python) for how to update your code to remove the use of deprecated features.

Compatibility

- TensorRT 8.6.1 has been tested with the following:

  - cuDNN 8.9.0
  - TensorFlow 1.15.5
  - PyTorch 1.13.1
  - ONNX 1.12.0
- This TensorRT release supports CUDA®:
  - 12.1 update 1
  - 12.0 update 1
  - 11.8
  - 11.7 update 1
  - 11.6 update 2
  - 11.5 update 2
  - 11.4 update 4
  - 11.3 update 1
  - 11.2 update 2
  - 11.1 update 1
  - 11.0 update 1
- This TensorRT release requires at least NVIDIA driver r450 on Linux or r452 on Windows as required by CUDA 11.0, which is the minimum CUDA version supported by this TensorRT release.
- It is suggested that you use TensorRT with a software stack that has been tested; including cuDNN and cuBLAS versions as documented in the Features For Platforms And Software section. Other semantically compatible releases of cuDNN and cuBLAS can be used; however, other versions may have performance improvements as well as regressions. In rare cases, functional regressions might also be observed.

Limitations

- There are two modes of DLA softmax where the mode is chosen automatically based on the shape of the input tensor, where:
  - the first mode triggers when all nonbatch, non-axis dimensions are 1, and
  - the second mode triggers in other cases if valid.

  The second of the two modes is supported only for DLA 3.9.0 and later. It involves approximations that may result in errors of a small degree. Also, batch size greater than 1 is supported only for DLA 3.9.0 and later. Refer to DLA Supported Layers for more information.
- On QNX, networks that are segmented into a large number of DLA loadables may fail during inference.

- You may encounter an error such as, "*Unable to load library: nvinfer_builder_resource.dll*", if using Python 3.9.10 on Windows. You can workaround this issue by downgrading to an earlier version of Python 3.9.
- The DLA compiler is capable of removing identity transposes, but it cannot fuse multiple adjacent transpose layers into a single transpose layer (likewise for reshape). For example, given a TensorRT `IShuffleLayer` consisting of two non-trivial transposes and an identity reshapes in between. The shuffle layer is translated into two consecutive DLA transpose layers, unless you merge the transposes together manually in the model definition in advance.
- In explicitly quantized networks, a group convolution that has a Q/DQ pair before but no Q/DQ pair after is expected to run with INT8-IN-FP32-OUT mixed precision. However, on NVIDIA Hopper™ it may fall back to FP32-IN-FP32-OUT if the input channel count is small. This will be fixed in a future release.
- On PowerPC platforms, samples that depend on TensorFlow, ONNX Runtime, and PyTorch are unable to run due to missing Python module dependencies. These frameworks have not been built for PowerPC and/or published to standard repositories.
- TensorRT 8.6 adds `nvinfer1::BuilderFlag::kFP8` and `nvinfer1::DataType::kFP8` to the public API as preparation for the introduction of FP8 support in future TensorRT releases. Despite these, FP8 (8-bit floating point) is not supported by TensorRT and attempting to use FP8 will result in an error or undefined behavior.
- `nvinfer1::UnaryOperation::kROUND` or `nvinfer1::UnaryOperation::kSIGN` operations of `IUnaryLayer` are not supported in the implicit batch mode.
- For networks containing normalization layers, particularly if deploying with mixed precision, target the latest ONNX opset that contains the corresponding function ops, for example: opset 17 for LayerNormalization or opset 18 GroupNormalization. Numerical accuracy using function ops is superior to corresponding implementation with primitive ops for normalization layers.

Deprecated and Removed Features

The following features are deprecated in TensorRT 8.6.1:

- Support for CUDA Toolkit 10.2 has been dropped.
- TensorRT 8.5.3 was the last release supporting NVIDIA Kepler (SM 3.x) and NVIDIA Maxwell (SM 5.x) devices. These devices are no longer supported in TensorRT 8.6. NVIDIA Pascal (SM 6.x) devices are deprecated in TensorRT 8.6.
- The `NvInferRuntimeCommon.h` file is being deprecated starting in TensorRT 8.6.0, and will be dropped in TensorRT 9.0. Instead, automotive safety users should include `NvInferSafeRuntime.h`. All other users should include `NvInferRuntime.h`.
- Removed the following samples:
  - C++ samples:

        - sampleSSD
        - sampleUffFasterRCNN
        - sampleUffMNIST
        - sampleUffMaskRCNN
        - sampleUffPluginV2Ext
        - sampleUffSSD
        - sampleMNIST
        - sampleFasterRCNN
        - sampleGoogleNet
        - sampleINT8
        - sampleMNISTAPI
    - Python samples:
        - uff_ssd
        - uff_custom_plugin
        - end_to_end_tensorflow_mnist
        - engine_refit_mnist
        - int8_caffe_mnist
    - introductory_parser_samples - removed from UFF and Caffe options
- The `trtexec` argument `--buildOnly` has been deprecated and was replaced by the argument `--skipInference`. The argument was renamed to better clarify the intention behind the argument.

Fixed Issues

- Fixed an up to 20% performance variation between different engines built from the same network for some LSTM networks due to unstable tactic selections. The tactic selection stability has been improved in this release.
- The r525 or later drivers contain the fix for an up to 11% performance variation for some LSTM networks during inference, depending on the order of CUDA stream creation on NVIDIA Turing GPUs.
- Fixed the issue that TensorRT might output wrong results when there are GEMM, Conv, and MatMul ops followed by a Reshape op.
- Improved the H100 performance for some ConvNets in TF32 precision.
- Improved the H100 performance for some Transformers in FP16 precision.
- Improved the H100 performance for some 3DUNets.
- Fixed an up to 6% performance drop for OpenRoadNet networks in TF32 precision compared to TensorRT 8.4 on NVIDIA Ampere architecture GPUs.

- Fixed an up to 5% performance drop for UNet networks in INT8 precision with explicit quantization on CUDA 11.x compared to CUDA 10.2 on NVIDIA Turing GPUs.
- Fixed an up to 16% performance drop for LSTM networks in FP32 precision compared to TensorRT 8.4 on NVIDIA Pascal GPUs.
- Fixed an up to 6% performance drop for ResNeXt-50 QAT networks in INT8, FP16, and FP32 precision at `batch-size = 1` compared to TensorRT 8.4 on NVIDIA Volta GPUs.
- H100 performance for some ConvNets containing depthwise convolutions (like QuartzNets and EfficientDet-D0) in INT8 precision was not fully optimized. This issue has been fixed in this release.
- There was a ~12% performance drop on NVIDIA Ampere architecture GPUs for the BERT network on Windows systems. This issue has been fixed in this release.
- There was an up to 17% performance regression for DeepASR networks at BS=1 on NVIDIA Turing GPUs. This issue has been fixed in this release.
- There was an up to 6% performance drop for WaveRNN networks in FP16 precision compared to TensorRT 8.4 on CUDA 11.8 on NVIDIA Volta GPUs. This issue has been fixed in this release.
- There was an up to 13% performance drop for Megatron networks in FP16 precision on V100 T4 GPUs when `disableExternalTacticSourcesForCore0805` was enabled. This issue has been fixed in this release.
- Fixed the issue that for some QAT models, when FP16 is enabled and a foreign node is created, if a tensor is the output of the foreign node and also serves as input to another node inside the subgraph of the foreign node, TensorRT may report an error with the following message for the node:

```
[W] [TRT] Skipping tactic 0x0000000000000000 due to Myelin error: Concat operation "XXX"
 has different types of operands.
```

- Resolved an issue with printing Unicode escape sequences while writing JSON files. This fix addresses previous parsing issues with JSON files in the TREx tool.
- The `DqQFusion` would sometimes generate a broken fused node with an empty scale parameter, which violated the following PointWiseFusion's assertion. This issue has been fixed in this release
- Explicit quantization on convolution with dynamic weights would fail to build on some platforms like Windows. This issue has been fixed in this release.
- There was an up to 10% performance regression for WaveRNN on Turing GPUs in FP16 precision. This issue has been fixed in this release.
- There was an up to 72% increase in GPU memory usage on H100 for various networks. This issue has been fixed in this release.
- There was an up to 31% performance drop for DeepASR on Turing GPUs in FP16 precision compared to TensorRT 8.5.1. This issue has been fixed in this release.
- There was an up to 12% performance drop for BERT on H100 in FP16 precision compared to TensorRT 8.5.0 EA. This issue has been fixed in this release.

- There was an up to 11% performance regression compared to TensorRT 8.5 on LSTM networks in FP16 precision on NVIDIA Ampere GPUs. This issue has been fixed in this release.
- The TensorRT engine might fail to build under QAT mode if paired Quantization and Dequantization layers were not adjacent to each other. This issue has been fixed in this release.
- The computation to determine the required number of resize dimensions has been improved; resolving an issue with missing quantized resize implementation where support for the number of resize dimensions is limited to 2.
- When using the algorithm selector API, the HWC1 and HWC4 DLA formats were both reported as `TensorFormat::kDLA_HWC4`. This issue has been fixed in this release.
- For some networks, containing matrix multiplication operations on A100, using TF32 could cause accuracy degradation. Disabling TF32 was the workaround. This issue has been fixed in this release.
- TensorRT engine building could fail if the users added an unnamed layer. This is because TensorRT could generate duplicated layer/tensor names for the unnamed layer. This issue has been fixed in this release.
- HuggingFace demos could fail if the system CUDA version is older than the CUDA version used to build PyTorch. This is because the environment was using the `cublasLT.so` from the system CUDA installation instead of the one distributed by PyTorch. This issue has been fixed in this release.
- There was a small accuracy drop for CortanaASR networks in TF32 precision. This issue has been fixed in this release.
- Using the compute sanitizer racecheck tool from CUDA 12.0 could report read-after-write hazards in unusual scenarios. This issue has been fixed in this release.
- There was a known issue with huge graphs that cause out of memory errors with specific input shapes even though a larger input shape can be run. This issue has been fixed in this release.
- The Python sample yolov3_onnx had a known issue when installing the requirements with Python 3.10. The recommendation was to use a Python version < 3.10 when running the sample. This issue has been fixed in this release.
- There was a known functionality issue when cross compiling TensorRT samples. The workaround was to set the `TRT_LIB_DIR` environment manually. This issue has been fixed in this release.
- For networks such as Tacotron 2 decoder, where there was a Convolution operation within a loop body, TensorRT could potentially fail during compilation. This issue has been fixed in this release.
- There was a known functional issue with thread safety when using multiple TensorRT builders concurrently. This issue has been fixed in this release.
- TensorRT compiled for CUDA 11.4 could fail to compile a graph when there were GEMM ops followed by a `gelu_erf` op. This issue has been fixed in this release.

- For transformer decoder-based models (such as GPT2) with sequence length as dynamic, TensorRT 8.5 required additional workspace (up to 2x) as compared to previous releases. This issue has been fixed in this release.
- There was an up to 27% performance drop for BART compared to TensorRT 8.2 when running with both FP16 and INT8 precisions enabled on T4. This issue has been fixed in this release.
- With the `kFASTER_DYNAMIC_SHAPES_0805` preview feature enabled on the GPT style decoder models, there could be an up to 20% performance regression for odd sequence lengths only compared to TensorRT without the use of the preview feature. This issue has been fixed in this release.
- The auto-tuner assumed that the number of indices returned by `INonZeroLayer` was half of the number of input elements. Thus, networks that depended on tighter assumptions for correctness could fail to build. This issue has been fixed in this release.
- There was an up to 15% performance drop for the CTFM model on V100 compared to TensorRT 8.5.1. This issue has been fixed in this release.
- The tactic optimizer would sometimes still choose a builder that cannot stride according to the timing results when there was a builder for this layer that can stride. If the adjacent reformat node was eliminated in this case, this layer would give outputs with wrong strides. This issue has been fixed in this release.
- For some QAT models, if convolution and pointwise fusion resulted in a multi-output layer with some output tensors quantized and others not, the building of the engine could fail with the following error message:

```
[E] Error[2]: [optimizer.cpp::filterQDQFormats::4422] Error Code 2: Internal Error
 (Assertion !n->candidateRequirements.empty() failed. All of the candidates were removed,
 which points to the node being incorrectly marked as an int8 node.
```

  One workaround was to disable the `kJIT_CONVOLUTIONS` tactic source. This issue has been fixed in this release.

Known Issues

Functional

- In some cases, when using the `OBEY_PRECISION_CONSTRAINTS` builder flag and the required type is set to FP32, the network can fail with a missing tactic due to an incorrect optimization converting the output of an operation to FP16. This can be resolved by removing the `OBEY_PRECISION_CONSTRAINTS` option.
- TensorRT may fail on devices with small RAM and swap space. This can be resolved by ensuring the RAM and swap space is at least 7 times the size of the network. For example, at least 21 GB of combined CPU memory for a 3 GB network.
- If an `IShapeLayer` is used to get the output shape of an `INonZeroLayer`, engine building will likely fail.

- ONNX models containing a large number of "Pad" layers have a potential for error accumulation. This can result in accuracy differences in inference between TensorRT and ONNX runtime.
- In rare cases, when using optimization level 5, you may see build failures with the error message
  ```
  "Skipping tactic 0x* due to exception Assertion index <= signedSize(shaders)
          failed"
  ```
  . A workaround solution is to use optimization level 4 instead.
- If a one-dimension INT8 input is used for a Unary or ElementWise operation, engine building may throw an internal error `"Could not find any implementation for node"`.
- Multihead attention fusion might not happen and affect performance if the number of heads is small.
- If a ONNX model contains a `Range` operator and its `limit` input is a data-dependent tensor, engine building will likely fail.
- Hardware forward compatibility (HFC) is broken on L4T Concord for ViT, Swin-Transformers, and BERT networks in FP16 mode. A workaround is to only use FP32 mode on L4T Concord or turn off HFC.
- Compute Sanitizer from CUDA Toolkit 12.0/12.1 may report a false alarm about invalid memory access in `generatedNativePointwise` kernels.
- Using the compute sanitizer racecheck tool may cause the process to be terminated unexpectedly. The root cause is a wrong false alarm. The issue can be bypassed with `--kernel-regex-exclude kns=scudnn_winograd`.
- If a network has a tensor of type bool with an *implicitly* data-dependent shape, engine building will likely fail.
- When using hardware compatibility features, TensorRT can potentially fail while compiling transformer based networks such as BERT. This issue will be fixed in the 8.6.1 release.
- There is an occurance of use-after-free in NVRTC that has been fixed in CUDA 12.1. When using NVRTC from CUDA 12.0 together with the TensorRT static library, you may encounter a crash in certain scenarios. Linking with the NVRTC and PTXJIT compiler from CUDA 12.1 or newer will resolve this issue.
- Although the version compatible runtime is optimized for efficiency, it may result in slower performance than the full runtime in certain use cases. Most networks can expect no more than a 10% slowdown when using a version-compatible engine compared to a version-locked engine. However, in some cases, a larger performance drop may occur. For example:
  - When running ResNet50_v2 with QAT, there may be up to a 22% decrease in performance.
  - When running DynUNet in FP16 precision, there may be up to a 32% decrease in performance.

  - When running ONNX networks with `InstanceNormalization` operations, there may be up to a 50% decrease in performance.
- There are known issues reported by the Valgrind memory leak check tool when detecting potential memory leaks from TensorRT applications. The recommendation to suppress the issues is to provide a Valgrind suppression file with the following contents when running the Valgrind memory leak check tool. Add the option `--keep-debuginfo=yes` to the Valgrind command line to suppress these errors.

```
{
  Memory leak errors with dlopen.
   Memcheck:Leak
   match-leak-kinds: definite
   ...
   fun:*dlopen*
   ...
}
{
    Memory leak errors with nvrtc
    Memcheck:Leak
    match-leak-kinds: definite
    fun:malloc
    obj:*libnvrtc.so*
    ...
}
```

- SM 7.5 and earlier devices may not have INT8 implementations for all layers with Q/DQ nodes. In this case, you will encounter a `could not find any implementation` error while building your engine. To resolve this, remove the Q/DQ nodes, which quantize the failing layers.
- TensorRT in FP16 mode does not perform cast operations correctly when only the output types are set, but not the layer precisions.
- TensorRT does not preserve precision for operations that are imported from ONNX models in FP16 mode.
- There is a known functional issue (fails with a CUDA error during compilation) with networks using `ILoop` layers on the WSL platform.
- Installing the `cuda-compat-11-4` package may interfere with CUDA enhanced compatibility and cause TensorRT to fail even when the driver is r465. The workaround is to remove the `cuda-compat-11-4` package or upgrade the driver to r470. *(not applicable for Jetson platforms)*
- TensorFlow 1.x is not supported for Python 3.9 or newer. Any Python samples that depend on TensorFlow 1.x cannot be run with Python 3.9 or newer.
- For some networks, using a batch size of 4096 may cause accuracy degradation on DLA.
- When using DLA, INT8 convolutions followed by FP16 layers may cause accuracy degradation. In such cases, either change the convolution to FP16 or the subsequent layer to INT8.
- Using the compute sanitizer tool from CUDA 12.0 may report a `cudaErrorIllegalInstruction` error on Hopper GPUs in unusual scenarios. This can be ignored, and will be fixed in a future CUDA release.

- Hardware compatible engines built with CUDA versions older than 11.5 may crash during inference when run on a GPU with a compute capability lower than that of the GPU where the engine was built. A workaround is to build on the GPU with the lowest compute capability.
- When enabling the cuDNN tactic source manually, there is a potential memory leak from the cuDNN library. This issue will be fixed in a future cuDNN release.

Performance

- There is an up to 28% performance regression compared to TensorRT 8.5 on Transformer networks in FP16 precision on NVIDIA Volta GPUs, and up to 85% performance regression on NVIDIA Pascal GPUs. Disable the `kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805` preview flag as a workaround.
- There may be higher peak GPU memory usage when building the engine on NVIDIA Ampere GPUs compared to TensorRT 8.5.
- There is an up to 13% performance drop for the CortanaASR model on NVIDIA Ampere GPUs compared to TensorRT 8.5.
- There is a known performance regression in the grouped deconvolution layer due to disabling cuDNN tactics. In TensorRT 8.6, performance can be recovered by unsetting `nvinfer1::PreviewFeature::kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805`. We will close the performance gap in a future release.
- There is an up to 27% performance drop for the SegResNet model on Ampere GPUs compared to TensorRT 8.6 EA. This drop can be avoided by enabling the `kVERSION_COMPATIBLE` flag in the ONNX parser.
- There is an up to 18% performance drop for the ShuffleNet model on A30/A40 compared to TensorRT 8.5.1.
- There may be minor performance regressions when running ONNX models with `InstanceNormalization` operators in version compatible mode. Refer to the NVIDIA TensorRT Developer Guide for more information.
- Convolution on a tensor with an *implicitly* data-dependent shape may run significantly slower than on other tensors of the same size. Refer to the Glossary for the definition of implicitly data-dependent shapes.
- For some Transformer models, including ViT, Swin-Transformer, and DETR, there is a performance drop in INT8 precision (including both explicit and implicit quantization) compared to FP16 precision.
- There is an up to 30% performance regression for LSTM variants with dynamic shapes. This issue can be resolved by disabling the `kFASTER_DYNAMIC_SHAPES_0805` preview feature in TensorRT 8.6.
- There is a known issue on H100 that may lead to GPU hang when running TensorRT with high persistentCache usage. Limit the usage to 40% of L2 cache size as a workaround.

- There is a known performance issue when running instance normalization layers on Arm Server Base System Architecture (SBSA).
- There is an up to 10% performance drop for the SegResNet network compared to TensorRT 8.2 when running in FP16 precision on NVIDIA Ampere architecture GPUs due to a cuDNN regression in the `InstanceNormalization` plug-in. This will be fixed in a future TensorRT release. You can work around the regression by reverting the cuDNN version to cuDNN 8.2.1.
- There is a performance drop when offloading a SoftMax layer to DLA on NVIDIA Orin as compared to when running the layer on a GPU, with a larger drop for larger batch sizes. As an example, FP16 AlexNet with batch size 16 shows 32% drop when the network runs on DLA as compared to when the last SoftMax layer runs on a GPU.
- There is a known issue with DLA clocks that requires users to reboot the system after changing the nvpmodel power mode or otherwise experience a performance drop. Refer to the L4T board support package Release Notes for details.
- For transformer-based networks such as BERT and GPT, TensorRT can consume CPU memory up to 7 times the model size during compilation.
- There is an up to 5% performance drop for networks using sparsity in FP16 precision.
- H100 performance for some LSTMs in FP16 precision is not fully optimized. This will be improved in future TensorRT versions.
- There is an up to 6% performance drop for T5 networks in FP32 precision compared to TensorRT 8.4 on NVIDIA Volta GPUs due to a functionality fix.
- There is an up to 17% performance drop for LSTM on Windows in FP16 precision compared to TensorRT 8.4 on NVIDIA Volta GPUs.
- There is an up to 7% performance drop for Artifact Reduction networks involving Deconvolution ops in INT8 precision compared to TensorRT 8.4 on NVIDIA Volta GPUs.
- There is an up to 16% performance regression on GPT2, T5, and Temporal-Fusion Transformers on NVIDIA Turing GPUs in FP32 precision due to a necessary accuracy fix. To recover the performance, enable FP16 precision.
- There is an up to 9% performance regression compared to TensorRT 8.5 on Yolov3 batch size 1 in FP32 precision on NVIDIA Ada Lovelace GPUs.
- There is an up to 6% performance regression compared to TensorRT 8.5 on OpenRoadNet in FP16 precision on NVIDIA A10 GPUs.
- There is an up to 13% performance regression compared to TensorRT 8.5 on GPT2 without kv-cache in FP16 precision when dynamic shapes are used on NVIDIA Volta and NVIDIA Ampere GPUs. Set the `kFASTER_DYNAMIC_SHAPES_0805` preview flag to `false` as a workaround.
- There is an up to 7% performance regression compared to TensorRT 8.5 on CortanaASR networks in FP16 precision on NVIDIA Volta GPUs. Disable the `kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805` preview flag as a workaround.

- There is an up to 23% performance regression compared to TensorRT 8.5 on LSTMs in FP32 precision when dynamic shapes are used on NVIDIA Turing GPUs. Set the `kFASTER_DYNAMIC_SHAPES_0805` preview flag to `false` as a workaround.
- There is an up to 23% performance regression compared to TensorRT 8.5 on Temporal Fusion Transformers in FP32 precision on NVIDIA Turing and NVIDIA Ampere GPUs.
- There is an up to 13% performance regression compared to TensorRT 8.5 on Multi-Layer Perceptron networks in FP16 precision on NVIDIA Ampere GPUs. Set the `kDISABLE_EXTERNAL_TACTIC_SORCES_FOR_CORE_0805` preview flag to `false` as a workaround.

3.2. TensorRT Release 8.5.3

These are the TensorRT 8.5.3 Release Notes and are applicable to x86 Linux, Windows, and JetPack users. This release incorporates Arm® based CPU cores for Server Base System Architecture (SBSA) users on Linux only. This release includes several fixes from the previous TensorRT releases as well as the following additional changes.

These Release Notes are applicable to workstation, server, and NVIDIA JetPack™ users unless appended specifically with *(not applicable for Jetson platforms)*.

For previously released TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Deprecated API Lifetime

- APIs deprecated before TensorRT 8.0 will be removed in TensorRT 9.0.
- APIs deprecated in TensorRT 8.0 will be retained until at least 8/2022.
- APIs deprecated in TensorRT 8.2 will be retained until at least 11/2022.
- APIs deprecated in TensorRT 8.4 will be retained until at least 2/2023.
- APIs deprecated in TensorRT 8.5 will be retained until at least 9/2023.

Refer to the API documentation (C++, Python) for how to update your code to remove the use of deprecated features.

Compatibility

- TensorRT 8.5.3 has been tested with the following:
  - cuDNN 8.6.0
  - TensorFlow 1.15.5
  - PyTorch 1.11.0
  - ONNX 1.12.0
- This TensorRT release supports CUDA®:

- 11.8
- 11.7 update 1
- 11.6 update 2
- 11.5 update 2
- 11.4 update 4
- 11.3 update 1
- 11.2 update 2
- 11.1 update 1
- 11.0 update 1
- 10.2

- It is suggested that you use TensorRT with a software stack that has been tested; including cuDNN and cuBLAS versions as documented in the Features For Platforms And Software section. Other semantically compatible releases of cuDNN and cuBLAS can be used; however, other versions may have performance improvements as well as regressions. In rare cases, functional regressions might also be observed.

Limitations

- There are two modes of DLA softmax where the mode is chosen automatically based on the shape of the input tensor, where:
  - the first mode triggers when all nonbatch, non-axis dimensions are 1, and
  - the second mode triggers in other cases if valid.

  The second of the two modes is supported only for DLA 3.9.0 and later. It involves approximations that may result in errors of a small degree. Also, batch size greater than 1 is supported only for DLA 3.9.0 and later. Refer to DLA Supported Layers for more information.
- On QNX, networks that are segmented into a large number of DLA loadables may fail during inference.
- You may encounter an error such as, "*`Unable to load library: nvinfer_builder_resource.dll`*", if using Python 3.9.10 on Windows. You can workaround this issue by downgrading to an earlier version of Python 3.9.
- Under some conditions, `RNNv2Layer` can require a larger workspace size in TensorRT 8.0 than TensorRT 7.2 in order to run all supported tactics. Consider increasing the workspace size to work around this issue.
- CUDA graph capture will capture `inputConsumed` and profiler events only when using the build for 11.x and >= 11.1 driver (455 or later).
- The DLA compiler is capable of removing identity transposes, but it cannot fuse multiple adjacent transpose layers into a single transpose layer (likewise for reshape). For example, given a TensorRT `IShuffleLayer` consisting of two non-trivial

transposes and an identity reshapes in between. The shuffle layer is translated into two consecutive DLA transpose layers, unless you merge the transposes together manually in the model definition in advance.

- In QAT networks, for group convolution that has a Q/DQ pair before but no Q/DQ pair after, we can run in INT8-IN-FP32-OUT mix precision before. However, GPU kernels may be missed and fall back to FP32-IN-FP32-OUT in the NVIDIA Hopper™ architecture GPUs if the input channel is small. This will be fixed in the future release.
- On PowerPC platforms, samples that depend on TensorFlow, ONNX Runtime, and PyTorch are unable to run due to missing Python module dependencies. These frameworks have not been built for PowerPC and/or published to standard repositories.

Deprecated and Removed Features

The following features are deprecated in TensorRT 8.5.3:

- TensorRT 8.5.3 will be the last release supporting NVIDIA Kepler (SM 3.x) and NVIDIA Maxwell (SM 5.x) devices. These devices will no longer be supported in TensorRT 8.6. NVIDIA Pascal (SM 6.x) devices will be deprecated in TensorRT 8.6.
- In the next TensorRT release, CUDA Toolkit 10.2 support will be dropped.

Fixed Issues

- For INT8 fused MHA plugins, support for sequence length 64 and 96 has been added.
- There was an issue on the PyTorch container where some ONNX models would fail with the error message `SSA validation FAIL`. This issue has now been fixed.
- When using `IExecutionContext::enqueueV3`, a non-null address must be set for every input tensor, using `IExecutionContext::setInputTensorAddress` or `IExecutionContext::setTensorAddress`, even if nothing is computed from the input tensor.
- `ISliceLayer` with `SampleMode::kWRAP` sometimes caused engine build failures when one of the strides was `0`.This issue has been fixed.
- There was an issue when a network used implicit batch and was captured using `cudaGraph`. This issue has now been fixed.
- When the PreviewFeature `kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805` is used, some plugins that use cuBLAS reported an `CUBLAS_STATUS_NOT_INITIALIZED` error (with CUDA version 11.8). This issue has now been fixed.
- For some encoder based transformer networks, if there was a forced precision on some layers, TensorRT reports the error (`Mismatched type for tensor`) during compilation. This issue has now been fixed.
- For some networks with branches, if there was a forced precision on some layers on one side, TensorRT reports the error `Mismatched type for tensor` during compilation. This issue has now been fixed.

- An assertion was triggered when multiple profiles were utilized and the profiles ended up causing different optimizations to occur, thus resulting in an error on the amount of slots being insufficient. This has been fixed to properly initialize the slots used internally.
- When a Matrix Multiply horizontal fusion pass fuses two layers with bias, the bias fusion didn't handle correctly which led to accuracy issues. This issue has now been fixed.
- There was an accuracy issue when a network contains an Exp operator and the Exp operator has a constant input. This issue has now been fixed.
- There was an up to 16% performance drop for LSTM networks in FP32 precision compared to TensorRT 8.4 on Pascal GPUs. This issue has now been fixed.
- TensorRT might output wrong results when there were GEMM/Conv/MatMul ops followed by a Reshape op. This issue has now been fixed.

Announcements

- In the next TensorRT release, cuDNN, cuBLAS, and cuBLASLt tactic sources will be turned off by default in builder profiling. TensorRT plans to remove the cuDNN, cuBLAS, and cuBLASLt dependency in future releases. Use the `PreviewFeature` flag `kDISABLE_EXTERNAL_TACTIC_SOURCES_FOR_CORE_0805` to evaluate the functional and performance impact of disabling cuBLAS and cuDNN and report back to TensorRT if there are critical regressions in your use cases.
- TensorRT Python wheel files before TensorRT 8.5, such as TensorRT 8.4, were published to the NGC PyPI repo. Starting with TensorRT 8.5, Python wheels will instead be published to upstream PyPI. This will make it easier to install TensorRT because it requires no prerequisite steps. Also, the name of the Python package for installation has changed from `nvidia-tensorrt` to just `tensorrt`.
- The C++ and Python API documentation in previous releases was included inside the tar file packaging. This release no longer bundles the documentation inside the tar file since the online documentation can be updated post release and avoids encountering mistakes found in stale documentation inside the packages.

Known Issues

Functional

- TensorRT compiled for CUDA 11.4 may fail to compile a graph when there are GEMM ops followed by a `gelu_erf` op.
- There is a known issue with huge graphs that cause out of memory errors with specific input shapes even though a larger input shape can be run.
- There are known issues reported by the Valgrind memory leak check tool when detecting potential memory leaks from TensorRT applications. The recommendation to suppress the issues is to provide a Valgrind suppression file with the following

contents when running the Valgrind memory leak check tool. Add the option `--keep-debuginfo=yes` to the Valgrind command line to suppress these errors.

```
{
  Memory leak errors with dlopen.
   Memcheck:Leak
   match-leak-kinds: definite
   ...
   fun:*dlopen*
   ...
}
{
    Memory leak errors with nvrtc
    Memcheck:Leak
    match-leak-kinds: definite
    fun:malloc
    obj:*libnvrtc.so*
    ...
}
```

- The Python sample yolov3_onnx has a known issue when installing the requirements with Python 3.10. The recommendation is to use a Python version < 3.10 when running the sample.
- The auto-tuner assumes that the number of indices returned by `INonZeroLayer` is half of the number of input elements. Thus, networks that depend on tighter assumptions for correctness may fail to build.
- SM 7.5 and earlier devices may not have INT8 implementations for all layers with Q/DQ nodes. In this case, you will encounter a `could not find any implementation` error while building your engine. To resolve this, remove the Q/DQ nodes, which quantize the failing layers.
- One of the deconvolution algorithms sourced from cuDNN exhibits non-deterministic execution. Disabling cuDNN tactics will prevent this algorithm from being chosen (refer to `IBuilderConfig::setTacticSources`).
- TensorRT in FP16 mode does not perform cast operations correctly when only the output types are set, but not the layer precisions.
- TensorRT does not preserve precision for operations that are imported from ONNX models in FP16 mode.
- There is a known functional issue (fails with a CUDA error during compilation) with networks using `ILoop` layers on the WSL platform.
- The tactic source cuBLASLt cannot be selected on SM 3.x devices for CUDA 10.x. If selected, it will fall back to using cuBLAS. *(not applicable for Jetson platforms)*
- Installing the `cuda-compat-11-4` package may interfere with CUDA enhanced compatibility and cause TensorRT to fail even when the driver is r465. The workaround is to remove the `cuda-compat-11-4` package or upgrade the driver to r470. *(not applicable for Jetson platforms)*
- TensorFlow 1.x is not supported for Python 3.9 or newer. Any Python samples that depend on TensorFlow 1.x cannot be run with Python 3.9 or newer.
- For some networks, using a batch size of 4096 may cause accuracy degradation on DLA.

- When using DLA, an elementwise, unary, or activation layer immediately followed by a scale layer may lead to accuracy degradation in INT8 mode. Note that this is a pre-existing issue also found in previous releases rather than a regression.
- When using DLA, INT8 convolutions followed by FP16 layers may cause accuracy degradation. In such cases, either change the convolution to FP16 or the subsequent layer to INT8.
- When using the algorithm selector API, the HWC1 and HWC4 DLA formats are both reported as `TensorFormat::kDLA_HWC4`.
- For transformer decoder based models (such as GPT2) with sequence length as dynamic, TensorRT 8.5 requires additional workspace (up to 2x) as compared to previous releases.
- For some QAT models, if convolution and pointwise fusion results in a multi-output layer with some output tensors quantized and others not, the building of the engine may fail with the following error message:

  ```
  [E] Error[2]: [optimizer.cpp::filterQDQFormats::4422] Error Code 2: Internal Error
   (Assertion !n->candidateRequirements.empty() failed. All of the candidates were removed,
   which points to the node being incorrectly marked as an int8 node.
  ```

  One workaround is to disable the `kJIT_CONVOLUTIONS` tactic source.
- For some QAT models, when FP16 is enabled and a foreign node is created, if a tensor is the output of the foreign node and also serves as input to another node inside the subgraph of the foreign node, TensorRT may report an error with the following message for the node:

  ```
  [W] [TRT] Skipping tactic 0x0000000000000000 due to Myelin error: Concat operation "XXX"
   has different types of operands.
  ```

  One workaround is insert a cast node between the tensor and the node inside the foreign node.

Performance

- There is a ~12% performance drop on NVIDIA Ampere architecture GPUs for the BERT network on Windows systems.
- There is a known performance issue when running instance normalization layers on Arm Server Base System Architecture (SBSA).
- There is an up to 22% performance drop for Jasper networks compared to TensorRT 8.2 when running in FP32 precision on NVIDIA Volta or NVIDIA Turing GPUs with CUDA 10.2. This performance drop can be avoided if CUDA 11.x is used instead.
- There is an up to 5% performance drop for the InceptionV4 network compared to TensorRT 8.2 when running in FP32 precision on NVIDIA Volta GPUs with CUDA 10.2. This performance drop can be avoided if CUDA 11.x is used instead.
- There is an up to 27% performance drop for BART compared to TensorRT 8.2 when running with both FP16 and INT8 precisions enabled on T4. This performance drop can be fixed by disabling the INT8 precision flag.
- There is an up to 10% performance drop for the SegResNet network compared to TensorRT 8.2 when running in FP16 precision on NVIDIA Ampere architecture GPUs

due to a cuDNN regression in the `InstanceNormalization` plug-in. This will be fixed in a future TensorRT release. You can work around the regression by reverting the cuDNN version to cuDNN 8.2.1.

- There is a performance drop when offloading a SoftMax layer to DLA on NVIDIA Orin as compared to when running the layer on a GPU, with a larger drop for larger batch sizes. As an example, FP16 AlexNet with batch size 16 shows 32% drop when the network runs on DLA as compared to when the last SoftMax layer runs on a GPU.
- There is an up to 20% performance variation between different engines built from the same network for some LSTM networks due to unstable tactic selections.
- Due to the difference in DLA hardware specification between NVIDIA Orin and Xavier, a relative increase in latency is expected when running DLA FP16 operations involving convolution (which includes deconvolution, fully-connected, and concat) on NVIDIA Orin as compared to running on Xavier. At the same DLA clocks and memory bandwidth, INT8 convolution operations on NVIDIA Orin are expected to be about 4x faster than on Xavier, whereas FP16 convolution operations on NVIDIA Orin are expected to be about 40% slower than on Xavier.
- There is a known issue with DLA clocks that requires users to reboot the system after changing the nvpmodel power mode or otherwise experience a performance drop. Refer to the L4T board support package Release Notes for details.
- For transformer-based networks such as BERT and GPT, TensorRT can consume CPU memory up to 10 times the model size during compilation.
- There is an up to 17% performance regression for DeepASR networks at BS=1 on NVIDIA Turing GPUs.
- There is an up to 7.5% performance regression compared to TensorRT 8.0.1.6 on NVIDIA Jetson AGX Xavier™ for ResNeXt networks in FP16 mode.
- There is an up to 10-11% performance regression on Xavier compared to TensorRT 7.2.3 for ResNet-152 with batch size 2 in FP16.
- There is an up to 40% regression compared to TensorRT 7.2.3 for DenseNet with CUDA 11.3 on P100 and V100. The regression does not exist with CUDA 11.0. *(not applicable for Jetson platforms)*
- On Xavier, DLA automatically upgrades INT8 LeakyRelu layers to FP16 to preserve accuracy. Thus, latency may be worse compared to an equivalent network using a different activation like ReLU. To mitigate this, you can disable LeakyReLU layers from running on DLA.
- There is an up to 126% performance drop when running some ConvNets on DLA in parallel to the other DLA and the iGPU on Xavier platforms, compared to running on DLA alone.
- There is an up to 5% performance drop for networks using sparsity in FP16 precision.
- There is an up to 5% performance drop for Megatron networks in FP32 precision at `batch-size = 1` between CUDA 11.8 and CUDA 10.2 on Volta GPUs. This performance drop does not happen on Turing or later GPUs.

- H100 performance for some ConvNets in TF32 precision is not fully optimized. This will be improved in future TensorRT versions.
- There is an up to 6% performance drop for ResNeXt-50 QAT networks in INT8, FP16, and FP32 precision at `batch-size = 1` compared to TensorRT 8.4 on NVIDIA Volta GPUs.
- H100 performance for some Transformers in FP16 precision is not fully optimized. This will be improved in future TensorRT versions.
- H100 performance for some ConvNets containering depthwise convolutions (like QuartzNets and EfficientDet-D0) in INT8 precision is not fully optimized. This will be improved in future TensorRT versions.
- H100 performance for some LSTMs in FP16 precision is not fully optimized. This will be improved in future TensorRT versions.
- H100 performance for some 3DUnets is not fully optimized. This will be improved in future TensorRT versions.
- There is an up to 6% performance drop for OpenRoadNet networks in TF32 precision compared to TensorRT 8.4 on NVIDIA Ampere architecture GPUs.
- There is an up to 6% performance drop for T5 networks in FP32 precision compared to TensorRT 8.4 on NVIDIA Volta GPUs due to a functionality fix.
- There is an up to 5% performance drop for UNet networks in INT8 precision with explicit quantization on CUDA 11.x compared to CUDA 10.2 on Turing GPUs.
- There is an up to 6% performance drop for WaveRNN networks in FP16 precision compared to TensorRT 8.4 on CUDA 11.8 on Volta GPUs. Downgrading CUDA to CUDA 11.6 fixes the issue.
- There is an up to 13% performance drop for Megatron networks in FP16 precision on Tesla T4 GPUs when `disableExternalTacticSourcesForCore0805` is enabled.
- There is an up to 17% performance drop for LSTM on Windows in FP16 precision compared to TensorRT 8.4 on Volta GPUs.
- There is an up to 7% performance drop for Artifact Reduction networks involving Deconvolution ops in INT8 precision compared to TensorRT 8.4 on Volta GPUs.
- With the `kFASTER_DYNAMIC_SHAPES_0805` preview feature enabled on the GPT style decoder models, there can be an up to 20% performance regression for odd sequence lengths only compared to TensorRT without the use of the preview feature.

3.3. TensorRT Release 8.4.3

These are the TensorRT 8.4.3 Release Notes and is applicable to x86 Linux and Windows users. This release incorporates Arm® based CPU cores for Server Base System Architecture (SBSA) users on Linux only. This release includes several fixes from the previous TensorRT releases as well as the following additional changes.

These Release Notes are applicable to workstation, server, and NVIDIA JetPack™ users unless appended specifically with *(not applicable for Jetson platforms)*.

For previously released TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Deprecated API Lifetime

- APIs deprecated before TensorRT 8.0 will be removed in TensorRT 9.0.
- APIs deprecated in TensorRT 8.0 will be retained until at least 8/2022.
- APIs deprecated in TensorRT 8.2 will be retained until at least 11/2022.
- APIs deprecated in TensorRT 8.4 will be retained until at least 2/2023.

Refer to the API documentation (C++, Python) for how to update your code to remove the use of deprecated features.

Compatibility

- TensorRT 8.4.3 has been tested with the following:
  - cuDNN 8.4.1
  - TensorFlow 1.15.5
  - PyTorch 1.9.0
  - ONNX 1.9.0
- This TensorRT release supports NVIDIA CUDA®:
  - 11.7 update 1
  - 11.6 update 2
  - 11.5 update 2
  - 11.4 update 4
  - 11.3 update 1
  - 11.2 update 2
  - 11.1 update 1
  - 11.0 update 1
  - 10.2
- It is suggested that you use TensorRT with a software stack that has been tested; including cuDNN and cuBLAS versions as documented in Features For Platforms And Software. Other semantically compatible releases of cuDNN and cuBLAS can be used; however, other versions may have performance improvements as well as regressions. In rare cases, functional regressions might also be observed.

Limitations

- There are two modes of DLA softmax where the mode is chosen automatically based on the shape of the input tensor, where:
  - the first mode triggers when all non-batch, non-axis dimensions are 1, and
  - the second mode triggers in other cases if valid.

  The second of the two modes is supported only for DLA 3.9.0 and later. It involves approximations that may result in errors of a small degree. Also, batch size greater than 1 is supported only for DLA 3.9.0 and later. Refer to DLA Supported Layers for more information.
- On QNX, networks that are segmented into a large number of DLA loadables may fail during inference.
- You may encounter an error such as, "*`Unable to load library: nvinfer_builder_resource.dll`*", if using Python 3.9.10 on Windows. You can workaround this issue by downgrading to an earlier version of Python 3.9.
- Under some conditions, `RNNv2Layer` can require a larger workspace size in TensorRT 8.0 than TensorRT 7.2 in order to run all supported tactics. Consider increasing the workspace size to work around this issue.
- The builder may require up to 60% more memory to build an engine.
- CUDA graph capture will capture `inputConsumed` and profiler events only when using the build for 11.x and >= 11.1 driver (455 or later).
- There is an up to 10% performance regression compared to TensorRT 7.2.3 in NVIDIA JetPack 4.5 for ResNet-like networks on NVIDIA DLA on Xavier platforms when the dynamic ranges of the inputs of the ElementWise `ADD` layers are different. This is due to a fix for a bug in DLA where it ignored the dynamic range of the second input of the ElementWise `ADD` layers and caused some accuracy issues. NVIDIA Orin platforms are not affected by this.

Fixed Issues

- When parsing networks with ONNX operand `expand` on scalar input. TensorRT would error out. This issue has been fixed in this release.
- The custom `ClipPlugin` used in the uff_custom_plugin sample had an issue with a plugin parameter not being serialized, leading to a failure when the plugin needed to be deserialized. This issue has been fixed with proper serialization/deserialization.
- When working with transformer based networks with multiple dynamic dimensions, if the network had shuffle operations which caused one or more dimensions to be a coalesced dimension (combination of multiple dynamic dimensions) and if this shuffle was further used in a reduction operation such as MatrixMultiply layer, it can potentially lead to corruption of results. This issue has been fixed in this release.

- When working with recurrent networks containing Loops and Fill layers, it was possible that the engine may have failed to build. This issue has been fixed in this release.
- In some rare cases when converting a MatrixMultiply layer to a Convolution layer for optimization purposes, the shapes may fail to inference. This issue has been fixed in this release.
- In some cases, Tensor memory was not zero initialized for vectorized dimensions. This resulted in NaN in the output tensor during engine execution. This issue has been fixed in this release.
- For the HuggingFace demos, the T5-3B model had only been verified on A100, and was not expected to work on A10, T4, and so on. This issue has been fixed in this release.
- Certain spatial dimensions may have caused crashes during DLA optimization for models using single-channel inputs. This issue has been fixed in this release.
- Under certain conditions on WSL2, an `INetwork` with Convolution layers that can be horizontally fused before a Concat layer may have created an internal error causing the application to crash while building the engine. This issue has been fixed in this release.
- For some networks using sparsity, TensorRT may have produced inaccurate results. This issue has been fixed in this release.

Announcements

- CUDA 11.7 added a feature called Lazy loading, however, this feature is not supported by TensorRT 8.4 because the CUDA 11.x binaries were built with CUDA Toolkit 11.6.

Known Issues

Functional

- When performing an L2_Normalization in float16 precision, there is undefined behavior occurring from a fusion. This fusion can be disabled by marking the input to the L2_Normalization as a network output.
- When performing PTQ with TensorRT with tensors rank > 4, some layers may cause an assertion about invalid Region Dims. This can be worked around by fusing the index layers into the 4th dimension to have the tensor have a rank 4.
- SM75 and earlier devices may not have INT8 implementations for all layers with Q/DQ nodes. In this case, you will encounter a `could not find any implementation` error while building your engine. To resolve this, remove the Q/DQ nodes which quantize the failing layers.
- When the TensorRT static library is used to build engines and the NVPTXCompiler static library is also used outside of the TensorRT core library at the same time, it is possible to trigger a crash of the process in rare cases.

- TensorRT should only allow up to a total of 16 I/O tensors for a single subnetwork offloaded to DLA. However, there is a leak in the logic that incorrectly allows > 16 I/O tensors. You may need to manually specify the per layer device to avoid the creation of subnetworks with over 16 I/O tensors, for successful engine construction. This restriction will be properly reinstated in a future release.
- One of the deconvolution algorithms sourced from cuDNN exhibits non-deterministic execution. Disabling cuDNN tactics will prevent this algorithm from being chosen (refer to `IBuilderConfig::setTacticSources`).
- Due to ABI compatibility issues, static builds are not supported on SBSA platforms.
- TensorRT in FP16 mode does not perform cast operations correctly when only the output types are set, but not the layer precisions.
- TensorRT does not preserve precision for operations that are imported from ONNX models in FP16 mode.
- There is a known issue when `ProfilingVerbosity` is set to `kDETAILED`, the `enqueueV2()` call may take up to 2ms compared to `ProfilingVerbosity=kNONE` or `kLAYER_NAMES_ONLY`.
- Under certain conditions on WSL2, an `INetwork` with Convolution layers that can be horizontally fused before a Concat layer may create an internal error causing the application to crash while building the engine. As a workaround, build your network on Linux instead of WSL2.
- There is a known functional issue (fails with a CUDA error during compilation) with networks using `ILoop` layers on the WSL platform.
- The tactic source cuBLASLt cannot be selected on SM 3.x devices for CUDA 10.x. If selected, it will fall back to using cuBLAS. *(not applicable for Jetson platforms)*
- Installing the `cuda-compat-11-4` package may interfere with CUDA enhanced compatibility and cause TensorRT to fail even when the driver is r465. The workaround is to remove the `cuda-compat-11-4` package or upgrade the driver to r470. *(not applicable for Jetson platforms)*
- TensorFlow 1.x is not supported for Python 3.9 or newer. Any Python samples that depend on TensorFlow 1.x cannot be run with Python 3.9 or newer.
- The Debian and RPM packages for the Python bindings, UFF, GraphSurgeon, and ONNX-GraphSurgeon wheels do not install their dependencies automatically; when installing them, ensure you install the dependencies manually using `pip`, or install the wheels instead.
- You may see the following error:

```
"Could not load library libcudnn_ops_infer.so.8. Error: libcublas.so.11: cannot
 open shared
        object file: No such file or directory"
```

after installing TensorRT from the network repo. cuDNN depends on the RPM dependency `libcublas.so.11()(64bit)`, however, this dependency installs cuBLAS from CUDA 11.0 rather than cuBLAS from the latest CUDA release. The library

search path will not be set up correctly and cuDNN will be unable to find the cuBLAS libraries. The workaround is to install the latest `libcublas-11-x` package manually.

- There is a known issue on Windows with the Python sample uff_ssd when converting the frozen TensorFlow graph into UFF. You can generate the UFF model on Linux or in a container and copy it over to work around this issue. Once generated, copy the UFF file to `\path\to\samples\python\uff_ssd\models\ssd_inception_v2_coco_2017_11_17\frozen_inference_graph.uff`.
- For some networks, using a batch size of 4096 may cause accuracy degradation on DLA.

Performance

- There is a known performance issue when running instance normalization layers on Arm Server Base System Architecture (SBSA).
- There is an up to 22% performance drop for Jasper networks compared to TensorRT 8.2 when running in FP32 precision on NVIDIA Volta or NVIDIA Turing GPUs with CUDA 10.2. This performance drop can be avoided if CUDA 11.x is used instead.
- There is an up to 5% performance drop for the InceptionV4 network compared to TensorRT 8.2 when running in FP32 precision on NVIDIA Volta GPUs with CUDA 10.2. This performance drop can be avoided if CUDA 11.x is used instead.
- There is an up to 27% performance drop for BART compared to TensorRT 8.2 when running with both FP16 and INT8 precisions enabled on T4. This performance drop can be fixed by disabling the INT8 precision flag.
- There is an up to 5% performance drop for the ShuffleNet network compared to TensorRT 8.2 when running in INT8 precision on NVIDIA Ampere Architecture GPUs. This will be fixed in a future TensorRT release.
- There is an up to 10% performance drop for the SegResNet network compared to TensorRT 8.2 when running in FP16 precision on NVIDIA Ampere Architecture GPUs due to a cuDNN regression in the `InstanceNormalization` plug-in. This will be fixed in a future TensorRT release. You can work around the regression by reverting the cuDNN version to cuDNN 8.2.1.
- There is an up to 10% performance difference for the WaveRNN network between different operating systems when running in FP16 precision on NVIDIA Ampere Architecture GPUs. This will be fixed in a future TensorRT release.
- There is a performance drop when offloading a SoftMax layer to DLA on NVIDIA Orin as compared to when running the layer on a GPU, with a larger drop for larger batch sizes. As an example, FP16 AlexNet with batch size 16 shows 32% drop when the network runs on DLA as compared to when the last SoftMax layer runs on a GPU.
- There is an up to 7% performance regression for the 3D-UNet networks compared to TensorRT 8.4 EA when running in INT8 precision on NVIDIA Orin due to a functionality fix.

- There is an up to 20% performance variation between different engines built from the same network for some LSTM networks when running on Windows due to unstable tactic selections.
- Some networks may see a small increase in deserialization time.
- Due to the difference in DLA hardware specification between NVIDIA Orin and Xavier, a relative increase in latency is expected when running DLA FP16 operations involving convolution (which includes deconvolution, fully-connected, and concat) on NVIDIA Orin as compared to running on Xavier. At the same DLA clocks and memory bandwidth, INT8 convolution operations on NVIDIA Orin are expected to be about 4x faster than on Xavier, whereas FP16 convolution operations on NVIDIA Orin are expected to be about 40% slower than on Xavier.
- There is a known issue with DLA clocks that requires users to reboot the system after changing the nvpmodel power mode or otherwise experience a performance drop. Refer to the L4T board support package Release Notes for details.
- For transformer-based networks such as BERT and GPT, TensorRT can consume CPU memory up to 10 times the model size during compilation.
- There is an up to 17% performance regression for DeepASR networks at BS=1 on NVIDIA Turing GPUs.
- There is an up to 7.5% performance regression compared to TensorRT 8.0.1.6 on NVIDIA Jetson AGX Xavier™ for ResNeXt networks in FP16 mode.
- There is a performance regression compared to TensorRT 7.1 for some networks dominated by FullyConnected with activation and bias operations:
  - up to 12% in FP32 mode. This will be fixed in a future release.
  - up to 10% in FP16 mode on NVIDIA Maxwell® and NVIDIA Pascal GPUs.
- There is an up to 10-11% performance regression on Xavier compared to TensorRT 7.2.3 for ResNet-152 with batch size 2 in FP16.
- There is an up to 40% regression compared to TensorRT 7.2.3 for DenseNet with CUDA 11.3 on P100 and V100. The regression does not exist with CUDA 11.0. *(not applicable for Jetson platforms)*
- On Xavier, DLA automatically upgrades INT8 LeakyRelu layers to FP16 to preserve accuracy. Thus, latency may be worse compared to an equivalent network using a different activation like ReLU. To mitigate this, you can disable LeakyReLU layers from running on DLA.
- There is an up to 126% performance drop when running some ConvNets on DLA in parallel to the other DLA and the iGPU on Xavier platforms, compared to running on DLA alone.
- There is an up to 5% performance drop for networks using sparsity in FP16 precision.

3.4. TensorRT Release 8.2.5

These are the TensorRT 8.2.5 Release Notes and are applicable to x86 Linux and Windows users. This release incorporates ARM® based CPU cores for Server Base System Architecture (SBSA) users on Linux only. This release includes several fixes from the previous TensorRT release as well as the following additional changes.

These Release Notes are also applicable to workstation, server, and NVIDIA JetPack™ users unless appended specifically with *(not applicable for Jetson platforms)*.

For previously released TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Deprecated API Lifetime

- APIs deprecated before TensorRT 8.0 will be removed in TensorRT 9.0.
- APIs deprecated in TensorRT 8.0 will be retained until at least 8/2022.
- APIs deprecated in TensorRT 8.2 will be retained until at least 11/2022.

Refer to the API documentation (C++, Python) for how to update your code to remove the use of deprecated features.

Compatibility

- TensorRT 8.2.5 has been tested with the following:
  - cuDNN 8.2.1
  - TensorFlow 1.15.5
  - PyTorch 1.9.0
  - ONNX 1.9.0
- This TensorRT release supports NVIDIA CUDA®:
  - 11.5 update 2
  - 11.4 update 3
  - 11.3 update 1
  - 11.2 update 2
  - 11.1 update 1
  - 11.0 update 1
  - 10.2
- It is suggested that you use TensorRT with a software stack that has been tested; including cuDNN and cuBLAS versions as documented in Features For Platforms And Software. Other semantically compatible releases of cuDNN and cuBLAS can be used,

however, other versions may have performance improvements as well as regressions. In rare cases, functional regressions might also be observed.

Fixed Issues

- There is a fast configuration for the SoftMax kernel which was not enabled previously when porting it from cuDNN. This performance regression has been fixed in this release.
- The Scale kernel had previously incorrectly supported strides (due to concat and slice elision). This issue has been fixed with this release.

Known Issues

Functional

- TensorRT attempts to catch GPU memory allocation failure and avoid profiling tactics whose memory requirements would trigger Out of Memory. However, GPU memory allocation failure cannot be handled by CUDA gracefully on some platforms and would lead to an unrecoverable application status. If this happens, consider lowering the specified workspace size if a large size is set, or using the `IAlgorithmSelector` interface to avoid tactics that require a lot of GPU memory.
- TensorRT may experience some instabilities when running networks containing TopK layers on T4 under Azure VM.
- Under certain conditions on WSL2, an `INetwork` with Convolution layers that can be horizontally fused before a Concat layer may create an internal error causing the application to crash while building the engine. As a workaround, build your network on Linux instead of WSL2.
- When running ONNX models with dynamic shapes, there is a potential accuracy issue if the dimension names of the inputs that are expected to be the same are not. For example, if a model has two 2D inputs of which the dimension semantics are both `batch` and `seqlen`, and in the ONNX model, the dimension name of the two inputs are different, there is a potential accuracy issue when running with dynamic shapes. Ensure you the dimension semantics match when exporting ONNX models from frameworks.
- There is a known functional issue (fails with a CUDA error during compilation) with networks using `ILoop` layers on the WSL platform.
- The tactic source cuBLASLt cannot be selected on SM 3.x devices for CUDA 10.x. If selected, it will fallback to using cuBLAS. *(not applicable for Jetson platforms)*
- For some networks with large amounts of weights and activation data, DLA may fail compiling a subgraph, and that subgraph will fallback to GPU.
- Under some conditions, `RNNv2Layer` can require a larger workspace size in TensorRT 8.0 than TensorRT 7.2 in order to run all supported tactics. Consider increasing the workspace size to work around this issue.

- CUDA graph capture will capture `inputConsumed` and profiler events only when using the build for 11.x and >= 11.1 driver (455 or above).
- On integrated GPUs, a memory tracking issue in TensorRT 8.0 that was artificially restricting the amount of available memory has been fixed. A side effect is that the TensorRT optimizer is able to choose layer implementations that use more memory, which can cause the OOM Killer to trigger for networks where it previously didn't. To work around this problem, use the `IAlgorithmSelector` interface to avoid layer implementations that require a lot of memory, or use the layer precision API to reduce precision of large tensors and use `STRICT_TYPES`, or reduce the size of the input tensors to the builder by reducing batch or other higher dimensions.
- TensorRT bundles a version of `libnvptxcompiler_static.a` inside `libnvinfer_static.a`. If an application links with a different version of PTXJIT than the version used to build TensorRT, it may lead to symbol conflicts or undesired behavior.
- Installing the `cuda-compat-11-4` package may interfere with CUDA enhanced compatibility and cause TensorRT to fail even when the driver is r465. The workaround is to remove the `cuda-compat-11-4` package or upgrade the driver to r470. *(not applicable for Jetson platforms)*
- TensorFlow 1.x is not supported for Python 3.9. Any Python samples that depend on TensorFlow 1.x cannot be run with Python 3.9.
- TensorRT has limited support for fusing `IConstantLayer` and `IShuffleLayer`. In explicit-quantization mode, the weights of Convolutions and Fully-Connected layers must be fused. Therefore, if a weights-shuffle is not supported, it may lead to failure to quantize the layer.
- For DLA networks where a convolution layer consumes an NHWC network input, the compute precision of the convolution layer must match the data type of the input tensor.
- Hybrid precision is not supported with the Pooling layer. Data type of input and output tensors should be the same as the layer precision.
- When running the Python engine_refit_mnist, network_api_pytorch_mnist, or onnx_packnet samples, you may encounter `Illegal instruction (core dumped)` when using the CPU version of PyTorch on Jetson TX2. The workaround is to install a GPU enabled version of PyTorch as per the instructions in the sample READMEs.
- Intermittent accuracy issues are observed in sample_mnist with INT8 precision on WSL2.
- The Debian and RPM packages for the Python bindings, UFF, GraphSurgeon, and ONNX-GraphSurgeon wheels do not install their dependencies automatically; when installing them, ensure you install the dependencies manually using `pip`, or install the wheels instead.
- You may see the following error:

```
"Could not load library libcudnn_ops_infer.so.8. Error: libcublas.so.11: cannot
 open shared
```

```
        object file: No such file or directory"
```

after installing TensorRT from the network repo. cuDNN depends on the RPM dependency `libcublas.so.11()(64bit)`, however, this dependency installs cuBLAS from CUDA 11.0 rather than cuBLAS from the latest CUDA release. The library search path will not be set up correctly and cuDNN will be unable to find the cuBLAS libraries. The workaround is to install the latest `libcublas-11-x` package manually.

- There is a known issue on Windows with the Python sample uff_ssd when converting the frozen TensorFlow graph into UFF. You can generate the UFF model on Linux or in a container and copy it over to work around this issue. Once generated, copy the UFF file to `\path\to\samples\python\uff_ssd\models\ssd_inception_v2_coco_2017_11_17\frozen_inference_graph.uff`.

Performance

- There is an up to 7.5% performance regression compared to TensorRT 8.0.1.6 on NVIDIA Jetson AGX Xavier™ for ResNeXt networks in FP16 mode.
- There is a performance regression compared to TensorRT 7.1 for some networks dominated by FullyConnected with activation and bias operations:
  - up to 12% in FP32 mode. This will be fixed in a future release.
  - up to 10% in FP16 mode on NVIDIA Maxwell® and Pascal GPUs.
- There is an up to 8% performance regression compared to TensorRT 7.1 for some networks with heavy FullyConnected operation like VGG16 on NVIDIA Jetson Nano™.
- There is an up to 10-11% performance regression on Xavier:
  - compared to TensorRT 7.2.3 for ResNet-152 with batch size 2 in FP16.
  - compared to TensorRT 6 for ResNeXt networks with small batch (1 or 2) in FP32.
- For networks that use deconv with large kernel size, the engine build time can increase a lot for this layer on Xavier. It can also lead to `the launch timed out and was terminated` error message on Jetson Nano/TX1.
- There is an up to 40% regression compared to TensorRT 7.2.3 for DenseNet with CUDA 11.3 on P100 and V100. The regression does not exist with CUDA 11.0. *(not applicable for Jetson platforms)*
- There is an up to 10% performance regression compared to TensorRT 7.2.3 in JetPack 4.5 for ResNet-like networks on NVIDIA DLA when the dynamic ranges of the inputs of the ElementWise `ADD` layers are different. This is due to a fix for a bug in DLA where it ignored the dynamic range of the second input of the ElementWise `ADD` layers and caused some accuracy issues.
- DLA automatically upgrades INT8 LeakyRelu layers to FP16 to preserve accuracy. Thus, latency may be worse compared to an equivalent network using a different activation like ReLU. To mitigate this, you can disable LeakyReLU layers from running on DLA.
- The builder may require up to 60% more memory to build an engine.

- There is an up to 126% performance drop when running some ConvNets on DLA in parallel to the other DLA and the iGPU on Xavier platforms, compared to running on DLA alone.
- There is an up to 21% performance drop compared to TensorRT 8.0 for SSD-Inception2 networks on NVIDIA Volta GPUs.
- There is an up to 5% performance drop for networks using sparsity in FP16 precision.
- There is an up to 25% performance drop for networks using the `InstanceNorm` plugin. This issue is being investigated.
- The engine building time for the networks using 3D convolution, like 3d_unet, is up to 500% longer compared to TensorRT 8.0 due to many fast kernels being added in, which enlarges the profiling time.

3.5. TensorRT Release 8.0.3

This is the TensorRT 8.0.3 release notes. This is a bug fix release supporting Linux x86 and Windows users.

These release notes are applicable to workstation, server, and JetPack users unless appended specifically with *(not applicable for Jetson platforms)*.

This release includes several fixes from the previous TensorRT 8.x.x release as well as the following additional changes. For previous TensorRT documentation, refer to the NVIDIA TensorRT Archived Documentation.

Fixed Issues

- Fixed an invalid fusion assertion problem in the fusion optimization pass.
- Fixed other miscellaneous issues seen in proprietary networks.
- Fixed a CUDA 11.4 NVRTC issue during kernel generation on Windows.

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.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Arm

Arm, AMBA and Arm Powered are registered trademarks of Arm Limited. Cortex, MPCore and Mali are trademarks of Arm Limited. "Arm" is used to represent Arm Holdings plc; its operating company Arm Limited; and the regional subsidiaries Arm Inc.; Arm KK; Arm Korea Limited.; Arm Taiwan Limited; Arm France SAS; Arm Consulting (Shanghai) Co. Ltd.; Arm Germany GmbH; Arm Embedded Technologies Pvt. Ltd.; Arm Norway, AS and Arm Sweden AB.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

Blackberry/QNX

Copyright © 2020 BlackBerry Limited. All rights reserved.

Trademarks, including but not limited to BLACKBERRY, EMBLEM Design, QNX, AVIAGE, MOMENTICS, NEUTRINO and QNX CAR are the trademarks or registered trademarks of BlackBerry Limited, used under license, and the exclusive rights to such trademarks are expressly reserved.

Google

Android, Android TV, Google Play and the Google Play logo are trademarks of Google, Inc.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com