PyNvVideoCodec

Read Me

v | May 2025

Table of Contents

Chapter 1. Read Me

1.1. Release Notes

Key Features and Enhancements

This release of PyNvVideoCodec includes the following features and enhancements:

Decode Features

- **Seek and frame sampling:** Provides efficient and flexible methods for fetching video frames in various modes, including sequential, random, periodic, indexed, batched, and sliced, as well as at a specified target frame rate.
- **Decoder caching:** Optimizes decoding of short video clips through decoder caching and reconfiguration.
- **Threaded decoder:** Supports decoding on separate threads, delivering pre-decoded frames with near-zero latency, enabling high-performance video processing pipelines.
- **Video processing from buffer:** Supports video processing from memory buffers, reducing I/O overhead, enabling streaming applications.
- **Low latency decode:** Offers zero-latency decoding for video sequences that do not contain B-frames.
- **SEI extraction:** Supports the extraction of Supplemental Enhancement Information (SEI) messages, allowing access to additional information such as HDR information, timecodes, and custom user data.
- **Stream metadata access:** Enables access to stream metadata, including frame width, height, bit depth, and keyframe indices, to enhance content management.
- **GIL handling:** Improved multithreaded performance through better handling of Global Interpreter Lock (GIL) in C++ layer.
- **Multi-GPU decode:** Enables multi-GPU decoding to efficiently handle larger workloads.
- **Extended codec support:** Supports codecs H.264, HEVC, AV1, VP8, VP9, VC1, MPEG4, MPEG2, and MPEG1

- **4:2:2 decode:** Supports 4:2:2 decoding for both H.264 and HEVC formats on Blackwell GPUs (NV16, P210 and P216 surface formats).
- **Extended output formats:** Decode to various output formats including NV12, YUV420, YUV444, NV16, P010, P016 and RGB24(interleaved and planar)

Encode Features

- **Encoder reconfiguration:** Supports encoder reconfiguration, enabling dynamic updating of encoding parameters without recreating encoder instances.
- **SEI insertion:** Allows insertion of SEI messages during encoding.
- **GIL handling:** Improved multithreaded performance through better handling of Global Interpreter Lock (GIL) in C++ layer.
- **Multi-GPU encode:** Enables multi-GPU encoding to efficiently handle larger workloads.
- **Codec support:** Support encoding to codec H.264, HEVC, and AV1.
- **4:2:2 encode:** Supports 4:2:2 encoding for both H.264 and HEVC formats on Blackwell GPUs (NV16 and P210 surface formats).
- **Extended input formats:** Encode from various input formats including NV12, YV12, IYUV, YUV444, YUV420_10BIT, YUV444_10BIT, NV16, P210, ARGB, ABGR, ARGB10, and ABGR10.

Transcode Features

- **Segment-based transcode:** Enables transcoding of video segments based on timestamp ranges, ideal for content editing and partial processing.

Limitations and Known Issues

- PyNvVideoCodec uses the FFmpeg binaries for demuxing of audio and video content.

  NVIDIA will not update the FFmpeg binaries included in our release package as these binaries are available, maintained and updated by the FFmpeg open-source community.

  > ! **ATTENTION:** NVIDIA does not provide support for FFMPEG; therefore, it is the responsibility of end users and developers, to stay informed about any vulnerabilities or quality bugs reported against FFMPEG. Users are encouraged to refer to the official FFmpeg website and community forums for the latest updates, patches, and support related to FFmpeg binaries and act as they deem necessary.

Package Contents

This package contains the following:

1. Sample applications demonstrating usage of PyNvVideoCodec APIs for encoding, decoding and transcoding use cases.
   - [.\samples\]

2. Python Bindings
    - [.src\PyNvVideoCodec]
3. Video codec helper classes and utilities
    - [.src\VideoCodecSDKUtils]
4. FFmpeg libraries and source code
    - [.external\ffmpeg]
5. Documents
    - [.docs]
6. Benchmarks contains performance benchmarking scripts for testing various PyNvVideoCodec features including segmented transcoding, decoder caching, and frame sampling capabilities.
    - [.\benchmarks\]

The sample applications provided in the package are for demonstration purposes only and may not be fully tuned for quality and performance. Hence the users are advised to do their independent evaluation for quality and/or performance.

1.2. System Requirements

Table 1. System Requirements

Operating System	▶ Windows 10 or higher ▶ Ubuntu 18.04 or higher
GPU	▶ Turing ▶ Ampere ▶ Ada ▶ Hopper ▶ Blackwell
Drivers	**Pre-Blackwell GPUs:** ▶ NVIDIA Windows display driver 531.61 or newer ▶ NVIDIA Linux display driver 530.41.03 or newer **Blackwell GPUs and onwards:**

	▶ NVIDIA Windows display driver 576.52 or newer ▶ NVIDIA Linux display driver 570.153.02 or newer Get most recent NVIDIA Display Driver
Python	▶ Python 3.10 ▶ Python 3.10 Dev (required in Ubuntu only)
CMake	▶ 3.21 and onwards
Visual Studio(Windows only)	▶ Visual Studio
CUDA Toolkit	Latest CUDA Toolkit
Python modules to run Sample applications	PyCUDA and PyTorch

Windows Subsystem for Linux (WSL) Configuration Requirements

- Add the directory /usr/lib/wsl/lib to PATH environment variable, in case it is not added by default. This is required to include path for the WSL libraries.
- Plus all the requirements under System Requirements

1.3. Installing PyNvVideoCodec Python Module

! **ATTENTION:** This project will download and install additional third-party open source software projects - DLPack. Review the license terms of these open source projects before use.

The Python module can be installed using following ways.

Installing from PyPI

1. The ready-to-use Python WHL's (Wheel) of the PyNvVideoCodec for Windows and Linux OSes are hosted on PyPI.
2. Open the bash/shell prompt and run:

   ```
   $>pip install PyNvVideoCodec
   ```

3. This is the recommended way.

Upon installation of the wheel, the sample applications and benchmark scripts are placed in the Python site-packages directory. The specific location of site-packages may vary depending on the operating system and Python environment. The path can be identified by running:

```
import site; print(site.getsitepackages())
```

Building and Installing from Source on NVIDIA NGC

The package containing PyNvVideCodec Python module's source code, all dependencies, Python sample applications, and documents is hosted on NVIDIA NGC.

Follow these steps:

1. Download the zip file of the latest package from NVIDIA NGC .
2. Open the bash/shell prompt from the same directory where zip was downloaded and run the following command, replacing "PyNvVideoCodec.zip" with the actual name of the downloaded zip file:

   ```
   $>pip install "PyNvVideoCodec.zip"
   ```

3. You can access documents and Python sample applications from the package.

Use this method if you need any customization on PyNvVideoCodec Python module e.g. enabling NVTX markers for profiling

Follow these steps to build customized version:

1. Unzip the source package to a directory.
2. Do the necessary modifications to the source.
3. On the same directory where `setup.py` is located, run the following commands:

   ```
   $>pip install .
   ```

1.4. Running Sample Applications

PyNvVideoCodec includes several sample applications that demonstrate key features and capabilities. These samples provide practical examples of how to implement video processing workflows using the API.

Prerequisites

Before running the samples, ensure you have:

- Installed PyNvVideoCodec following the installation instructions
- NVIDIA GPU with appropriate drivers installed
- Required dependencies installed (as listed in the Read Me)

Decoder Sample Applications

Decode.py - Basic video decoding sample

```
python Decode.py -g 0 -i input_video.mp4 -o output_frames_dir -d 1
```

Parameter	Type	Description
-g, --gpu_id	int	Ordinal of GPU to use (default: 0)
-i, --input	string	Path to input video file
-o, --output	string	Path to output directory for decoded frames
-d	int	Output type: 0 for host memory, 1 for device memory
-lm	int	Enable zero latency for All-Intra / IPPP streams. Do not use this flag if the stream contains B-frames

DecodePerf.py - Measures decoder performance

```
python DecodePerf.py -g 0 -i input_video.mp4 -d 1 -n 1
```

Parameter	Type	Description
-g, --gpu_id	int	Ordinal of GPU to use (default: 0)
-i, --input	string	Path to input video file
-d	int	Output type: 0 for host memory, 1 for device memory
-n	int	Number of processes to launch (typically twice the number of NVDECs for full throughput)
-f	int	Number of frames to decode

DecodeMultiprocessing.py - Demonstrates decoder in a multiprocessing setup

```
python DecodeMultiprocessing.py -g 0 -i input_video.mp4 -n 3 -f 100
```

Parameter	Type	Description
-g, --gpu_id	int	Ordinal of GPU to use (default: 0)
-i, --raw_file_path	string	Path to input video file
-n, --number	int	Number of processes to launch
-f, --frame_count	int	Number of frames to decode

DecodeSEIMsgExtraction.py - Demonstrates extracting SEI messages during decoding

```
python DecodeSEIMsgExtraction.py -g 0 -i input_video.mp4 -o output.yuv -d 1
```

Parameter	Type	Description
-g	int	Ordinal of GPU to use (default: 0)
-i	string	Path to input video file
-o	string	Path to output YUV file

Parameter	Type	Description
-d	int	Output type: 0 for host memory, 1 for device memory

DemuxFromByteArray.py - Demonstrates demuxing from byte array

```
python DemuxFromByteArray.py -i input.ts -o output.yuv -d 1
```

Parameter	Type	Description
-i	string	Path to input video file (typically TS format)
-o	string	Path to output YUV file
-d	int	Output type: 0 for host memory, 1 for device memory
-g	int	Ordinal of GPU to use (default: 0)

Encoder Sample Applications

Encode.py - Basic video encoding sample

```
python Encode.py -g 0 -i input.yuv -o output.h264 -s 1920x1080 -if nv12 -c h264 -
json encode_config.json
```

Parameter	Type	Description
-g, --gpu_id	int	Ordinal of GPU to use (default: 0)
-i, --raw_file_path	string	Path to input raw video file
-o, --encoded_file_path	string	Path to output encoded video
-s, --size	string	Input resolution in format WxH (e.g., 1920x1080)
-if, --format	string	Input pixel format (NV12, ARGB, ABGR, YUV444, YUV420, P010, YUV444_16BIT, NV16, P210)
-c, --codec	string	Output codec (h264, hevc, av1)
-json	string	JSON config file with encoding parameters

EncodeFromCPUBuffer.py - Demonstrates encoding from host memory buffers

```
python EncodeFromCPUBuffer.py -g 0 -i input.yuv -o output.h264 -s 848x464 -if nv12 -
c h264 -json encode_config.json
```

Parameter	Type	Description
-g, --gpu_id	int	Ordinal of GPU to use (default: 0)
-i, --raw_file_path	string	Path to input raw video file
-o, --encoded_file_path	string	Path to output encoded video
-s, --size	string	Input resolution in format WxH (e.g., 1920x1080)
-if, --format	string	Input pixel format (NV12, ARGB, ABGR, YUV444, YUV420, P010, YUV444_16BIT, NV16, P210)
-c, --codec	string	Output codec (h264, hevc, av1)
-json	string	JSON config file with encoding parameters

EncodeSEIMsgInsertion.py - Demonstrates inserting SEI messages during encoding

```
python EncodeSEIMsgInsertion.py -i input.yuv -o output.hevc -g 0 -if NV12 -c hevc -
s 1920x1080
```

Parameter	Type	Description
-g, --gpu_id	int	Ordinal of GPU to use (default: 0)
-i, --raw_file_path	string	Path to input raw video file
-o, --encoded_file_path	string	Path to output encoded video
-s, --size	string	Input resolution in format WxH (e.g., 1920x1080)
-if, --format	string	Input pixel format (NV12, ARGB, ABGR, YUV444, YUV420, P010, YUV444_16BIT)
-c, --codec	string	Output codec (h264, hevc, av1)

EncodeReconfigure.py - Demonstrates encoder reconfiguration at runtime

```
python EncodeReconfigure.py -i input.yuv -o output.h264 -s 848x464 -if nv12 -c h264
 -json encode_config_lowlatency.json
```

Parameter	Type	Description
-g, --gpu_id	int	Ordinal of GPU to use (default: 0)
-i, --raw_file_path	string	Path to input raw video file
-o, --encoded_file_path	string	Path to output encoded video
-s, --size	string	Input resolution in format WxH (e.g., 1920x1080)
-if, --format	string	Input pixel format (NV12, ARGB, ABGR, YUV444, YUV420, P010, YUV444_16BIT)
-c, --codec	string	Output codec (h264, hevc, av1)
-json	string	JSON config file with encoding parameters

EncodePerf.py - Measures encoder performance

```
python EncodePerf.py -g 0 -i input.yuv -s 1920x1080 -if NV12 -c h264 -n 3 -f 100
```

Parameter	Type	Description
-g, --gpu_id	int	Ordinal of GPU to use (default: 0)
-i, --raw_file_path	string	Path to input raw video file
-s, --size	string	Input resolution in format WxH (e.g., 1920x1080)
-if, --format	string	Input pixel format (NV12, ARGB, ABGR, YUV444, YUV420, P010, YUV444_16BIT)
-c, --codec	string	Output codec (h264, hevc, av1)
-json	string	JSON config file with encoding parameters
-n, --number	int	Number of processes to launch
-f, --frame_count	int	Number of frames to encode

EncodeMultiprocessing.py - Demonstrates encoding in a multiprocessing setup

```
python EncodeMultiprocessing.py -g 0 -i input.yuv -s 1920x1080 -if NV12 -c h264 -
n 3 -f 100
```

Parameter	Type	Description
-g, --gpu_id	int	Ordinal of GPU to use (default: 0)
-i, --raw_file_path	string	Path to input raw video file
-s, --size	string	Input resolution in format WxH (e.g., 1920x1080)
-if, --format	string	Input pixel format (NV12, ARGB, ABGR, YUV444, YUV420, P010, YUV444_16BIT)
-c, --codec	string	Output codec (h264, hevc, av1)
-json	string	JSON config file with encoding parameters
-n, --number	int	Number of processes to launch
-f, --frame_count	int	Number of frames to encode

Transcoding and Advanced Sample Applications

EnsembleApp.py - Demonstrates various frame retrieval methods and segmented transcoding

```
python EnsembleApp.py -i input.mp4 -t timeListInSeconds.txt -d 1 -g 0 -json
 transcode_config.json -segments segments.txt -o 1 3 4 -so segmented_output.mp4
```

Parameter	Type	Description
-i	string	Input video file
-t	string	Path to text file with time values (in seconds), one per line
-d	int	Output type: 0 for host memory, 1 for device memory
-g	int	GPU ID (default: 0)
-segments	string	Path to segment file with start and end times separated by space on each line
-json	string	Config file with transcoding parameters (must include "bf" field)
-o	int list	Operations to perform: 1=Batch frame comparison, 2=Frame slicing, 3=Timestamp extraction, 4=Keyframe extraction, 5=Segment generation
-so	string	Base output file name template for segmented transcode

SubsamplingAndReconfigure.py - Demonstrates frame subsampling and decoder reconfiguration

```
python SubsamplingAndReconfigure.py -i fileLists.txt -fps 5 -c 1 -d 1 -v 1 -g 0
```

Parameter	Type	Description
-i	string	Text file containing video file paths, one per line
-fps	int	Desired output frame rate in frames per second
-c	int	CUDA options: 0 for default CUDA initializations, 1 to enable CUDA stream and context
-d	int	Output type: 0 for host memory, 1 for device memory
-v	int	Frame verification: 0 to skip verification, 1 to verify frames against golden YUV files
-g	int	Ordinal of GPU to use (default: 0)

ObjectDetection.py - Demonstrates integration with AI model for object detection

```
python ObjectDetection.py -i test_video.mp4 -d -c 0.5
```

Parameter	Type	Description
-i	string	Input video file path
-d	flag	Display the output (no value needed)
-o	string	Output file to dump detection results
-c	float	Confidence threshold for detections (default: 0.5)

JSON Encoder Configuration Parameters

Many of the samples accept a JSON configuration file for encoder parameters. Here are the key parameters:

Parameter	Type	Valid Values	Default
codec	string	"h264", "hevc", "av1"	"h264"
bitrate	integer	> 0	10000000
qp	integer or list	0-51	[30,30,30]
gop	integer	> 0	varies by settings
tuning_info	string	"high_quality", "low_latency", "ultra_low_latency", "lossless"	"high_quality"
preset	string	"P1" to "P7"	"P4"
rc	string	"cbr", "constqp", "vbr"	"cbr"
bf	integer	>= 0	varies by preset