NVIDIA NeMo

Best Practices | NVIDIA Docs

DU-09886-001_v1.0.0rc1 | July 2020

Table of Contents

Chapter 1. Overview

The NVIDIA NeMo Toolkit is available on GitHub as open source as well as a Docker container on NGC. This guide assumes that the user has already installed NeMo by following the Quick Start instructions in the *NVIDIA NeMo User Guide*.

The conversational AI pipeline consists of three major stages:

- Automatic Speech Recognition (ASR)
- Natural Language Processing (NLP) or Natural Language Understanding (NLU)
- Text-to-Speech (TTS) or voice synthesis

As you talk to a computer, the ASR phase converts the audio signal into text, the NLP stage interprets the question and generates a smart response, and finally the TTS phase converts the text into speech signals to generate audio for the user. The toolkit enables development and training of deep learning models involved in conversational AI and easily chain them together.

Chapter 2. Why NeMo?

Deep learning model development for Conversational AI is complex. It involves defining, building, and training several models in specific domains; experimenting several times to get high accuracy, fine tuning on multiple tasks and domain specific data, ensuring training performance and making sure the models are ready for deployment to inference applications.

Neural modules are logical blocks of AI applications which take some *typed* inputs and produce certain *typed* outputs. By separating a model into its essential components in a building block manner, NeMo helps researchers develop state-of-the-art accuracy models for domain specific data faster and easier.

Collections of modules for core tasks as well as specific to speech recognition, natural language, speech synthesis help develop modular, flexible, and reusable pipelines.

A neural module's inputs/outputs have a neural type, that describes the semantics, the axis order and meaning, and the dimensions of the input/output tensors. This typing allows neural modules to be safely chained together to build models for applications.

NeMo can be used to train new models or perform transfer learning on existing pre-trained models. Pre-trained weights per module (such as encoder, decoder) help accelerate model training for domain specific data.

ASR, NLP and TTS pre-trained models are trained on multiple datasets (including some languages such as Mandarin) and optimized for high accuracy. They can be used for transfer learning as well.

NeMo supports developing models that work with Mandarin Chinese data. Tutorials help users train or fine tune models for conversational AI with the Mandarin Chinese language. The export method provided in NeMo makes it easy to transform a trained model into inference ready format for deployment.

A key area of development in the toolkit is interoperability with other tools used by speech researchers. Data layer for Kaldi compatibility is one such example.

Chapter 3. NeMo, PyTorch Lightning, And Hydra

Conversational AI architectures are typically very large and require a lot of data and compute for training. NeMo uses Pytorch Lightning for easy and performant multi-GPU/multi-node mixed precision training.

Pytorch Lightning is a high-performance PyTorch wrapper that organizes PyTorch code, scales model training, and reduces boilerplate. PyTorch Lightning has two main components, the `LightningModule` and the Trainer. The `LightningModule` is used to organize PyTorch code so that deep learning experiments can be easily understood and reproduced. The Pytorch Lightning Trainer is then able to take the `LightningModule` and automate everything needed for deep learning training.

NeMo models are `LightningModules` that come equipped with all supporting infrastructure for training and reproducibility. This includes the deep learning model architecture, data preprocessing, optimizer, check-pointing and experiment logging. NeMo models, like `LightningModules`, are also PyTorch modules and are fully compatible with the broader PyTorch ecosystem. Any NeMo model can be taken and plugged into any PyTorch workflow.

Configuring conversational AI applications is difficult due to the need to bring together many different Python libraries into one end-to-end system. NeMo uses Hydra for configuring both NeMo models and the PyTorch Lightning Trainer. Hydra is a flexible solution that makes it easy to configure all of these libraries from a configuration file or from the command-line.

Every NeMo model has an example configuration file and a corresponding script that contains all configurations needed for training to state-of-the-art accuracy. NeMo models have the same look and feel so that it is easy to do conversational AI research across multiple domains.

Chapter 4. Using Optimized Pretrained Models With NeMo

The NVIDIA GPU Cloud (NGC) is a software repository that has containers and models optimized for deep learning. NGC hosts many conversational AI models developed with NeMo that have been trained to state-of-the-art accuracy on large datasets. NeMo models on NGC can be automatically downloaded and used for transfer learning tasks.

Pretrained models are the quickest way to get started with conversational AI on your own data. NeMo has many example scripts and Jupyter Notebook tutorials showing step-by-step how to fine-tune pretrained NeMo models on your own domain-specific datasets.

The table below shows all pretrained models available to use. For BERT based models, the model weights provided are ready for downstream NLU tasks. For speech models, it can be helpful to start with a pretrained model and then continue pretraining on your own domain-specific data. Jasper and QuartzNet base model pretrained weights have been known to be very efficient when used as base models. For an easy to follow guide on transfer learning and building domain specific ASR models, you can follow this blog. All pre-trained NeMo models can be found on the NGC NeMo Collection. Everything needed to quickly get started with NeMo ASR, NLP, and TTS models is there.

Pre-trained models are packaged as a `.nemo` file and contain the PyTorch checkpoint along with everything needed to use the model. NeMo models are trained to state-of-the-art accuracy and trained on multiple datasets so that they are robust to small differences in data. NeMo contains a large variety of models such as speaker identification and Megatron BERT and the best models in speech and language are constantly being added as they become available. NeMo is the premier toolkit for conversational AI model building and training.

Below you can find a description of the models available in NeMo as well as links to tutorials to run training/fine tuning workflow.

Table 1. Tutorials

Domain	Title	GitHub URL
NeMo	Simple Application with NeMo	Voice swap app
NeMo	Exploring NeMo Fundamentals	NeMo primer
NeMo Models	Exploring NeMo Model Construction	NeMo models
ASR	ASR with NeMo	ASR with NeMo

ASR	Speech Commands	Speech commands
ASR	Speaker Recognition and Verification	Speaker Recognition and Verification
ASR	Online Noise Augmentation	Online noise augmentation
NLP	Using Pretrained Language Models for Downstream Tasks	Pretrained language models for downstream tasks
NLP	Exploring NeMo NLP Tokenizers	NLP tokenizers
NLP	Text Classification (Sentiment Analysis) with BERT	Text Classification (Sentiment Analysis)
NLP	Question answering with SQuAD	Question answering Squad
NLP	Token Classification (Named Entity Recognition)	Token classification: named entity recognition
NLP	Joint Intent Classification and Slot Filling	Joint Intent and Slot Classification
NLP	GLUE Benchmark	GLUE benchmark
NLP	Punctuation and Capitialization	Punctuation and capitalization
NLP	Named Entity Recognition - BioMegatron	Named Entity Recognition - BioMegatron
NLP	Relation Extraction - BioMegatron	Relation Extraction - BioMegatron
TTS	Speech Synthesis	TTS inference

Chapter 5. ASR Guidance

This section is to help guide your decision making by answering our most asked ASR questions.

Q: Is there a way to add domain specific vocabulary in NeMo? If so, how do I do that?

A: QuartzNet and Jasper models are character-based. So pretrained models we provide for these two output lowercase English letters and `'`. Users can re-retrain them on vocabulary with upper case letters and punctuation symbols.

Q: When training, there are "Reference" lines and "Decoded" lines that are printed out. It seems like the reference line should be the "truth" line and the decoded line should be what the ASR is transcribing. Why do I see that even the reference lines do not appear to be correct?

A: Because our pre-trained models can only output lowercase letters and apostrophe, everything else is dropped. So the model will transcribe `10` as `ten`. The best way forward is to prepare the training data first by transforming everything to lowercase and convert the numbers from digit representation to word representation using a simple library such asinflect. Then, add the uppercase letters and punctuation back using the NLP punctuation model. Here is an example of how this is incorporated: NeMo voice swap demo.

Q: What languages are supported in NeMo currently?

A: Along with English, Mandarin Chinese is supported. A pre-trained model for Mandarin, QuartzNet15x5Base-Zh, is provided that works for that language. For more information, see NeMo Speech Models.

Chapter 6. Data Augmentation

Data augmentation in ASR is invaluable. It comes at the cost of increased training time if samples are augmented during training time. To save training time, it is recommended to pre-process the dataset offline for a one time preprocessing cost and then train the dataset on this augmented training set.

For example, processing a single sample involves:

- Speed perturbation
- Time stretch perturbation (sample level)
- Noise perturbation
- Impulse perturbation
- Time stretch augmentation (batch level, neural module)

A simple tutorial guides users on how to use these utilities provided in NeMo: GitHub: NeMo.

Chapter 7. Speech Data Explorer

Speech data explorer is a Dash-based tool for interactive exploration of ASR/TTS datasets.

Speech data explorer it helps find out:

- Dataset statistics. For example, alphabet, vocabulary, and duration-based histograms.
- Navigation across dataset. For example, sorting and filtering.
- Inspection of individual utterances. For example, waveform, spectrogram, and audio player.
- Errors analysis. For example, word error rate, character error rate, word match rate, mean word accuracy, and diff.

In order to use the tool, it needs to be installed separately. Perform the steps here to install speech data explorer.

Chapter 8. Using Kaldi Formatted Data

The Kaldi Speech Recognition Toolkit project began in 2009 at Johns Hopkins University. It is a toolkit written in C++. If researchers have used Kaldi and have datasets that are formatted to be used with the toolkit; they can use NeMo to develop models based on that data.

To load Kaldi-formatted data, you can simply use `KaldiFeatureDataLayer` instead of `AudioToTextDataLayer`. The `KaldiFeatureDataLayer` takes in the argument `kaldi_dir` instead of a `manifest_filepath`. The `manifest_filepath` argument should be set to the directory that contains the files `feats.scp` and `text`.

Chapter 9. Using Speech Command Recognition Task For ASR Models

Speech Command Recognition is the task of classifying an input audio pattern into a set of discrete classes. It is a subset of ASR, sometimes referred to as Key Word Spotting, in which a model is constantly analyzing speech patterns to detect certain `action` classes.

Upon detection of these *commands*, a specific action can be taken. An example Jupyter notebook provided in NeMo shows how to train a QuartzNet model with a modified decoder head trained on a speech commands dataset.

Note: It is preferred that you use absolute paths to `data_dir` when preprocessing the dataset.

Chapter 10. NLP Fine-Tuning BERT

BERT, or Bidirectional Encoder Representations from Transformers, is a neural approach to pre-train language representations which obtains near state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks, including the GLUE benchmark and SQuAD Question & Answering dataset.

BERT model checkpoints (BERT-large-uncased and BERT-base-uncased) are provided can be used for either fine tuning BERT on your custom dataset, or fine tuning downstream tasks, including GLUE benchmark tasks, Question & Answering tasks, Joint Intent & Slot detection, Punctuation and Capitalization, Named Entity Recognition, and Speech Recognition post processing model to correct mistakes.

Note: Almost all NLP examples also support RoBERTa and ALBERT models for downstream fine-tuning tasks (see the list of all supported models by calling `nemo.collections.nlp.modules.common.lm_utils.get_pretrained_lm_models_list()`. The user needs to specify the name of the model desired while running the example scripts.

Chapter 11. BioMegatron Medical BERT

BioMegatron is a large language model (Megatron-LM) trained on larger domain text corpus (PubMed abstract + full-text-commercial). It achieves state-of-the-art results for certain tasks such as Relationship Extraction, Named Entity Recognition and Question &nAnswering. Follow these tutorials to learn how to train and fine tune BioMegatron; pretrained models are provided on NGC:

- Relation_Extraction-BioMegatron.ipynb
- Token_Classification-BioMegatron.ipynb

Chapter 12. Efficient Training With NeMo

12.1. Using Mixed Precision

Mixed precision accelerates training speed while protecting against noticeable loss. Tensor Cores is a specific hardware unit that comes starting with the Volta and Turing architectures to accelerate large matrix to matrix multiply-add operations by operating them on half precision inputs and returning the result in full precision.

Neural networks which usually use massive matrix multiplications can be significantly sped up with mixed precision and Tensor Cores. However, some neural network layers are numerically more sensitive than others. Apex AMP is an NVIDIA library that maximizes the benefit of mixed precision and Tensor Cores usage for a given network.

12.2. Multi-GPU Training

This section is to help guide your decision making by answering our most asked multi-GPU training questions.

Q: Why is multi-GPU training preferred over other types of training?

A: Multi-GPU training can reduce the total training time by distributing the workload onto multiple compute instances. This is particularly important for large neural networks which would otherwise take weeks to train until convergence. Since NeMo supports multi-GPU training, no code change is needed to move from single to multi-GPU training, only a slight change in your launch command is required.

Q: What are the advantages of mixed precision training?

A: Mixed precision accelerates training speed while protecting against noticeable loss in precision. Tensor Cores is a specific hardware unit that comes starting with the Volta and Turing architectures to accelerate large matrix multiply-add operations by operating on half precision inputs and returning the result in full precision in order to prevent loss in precision. Neural networks which usually use massive matrix multiplications can be significantly sped up

with mixed precision and Tensor Cores. However, some neural network layers are numerically more sensitive than others. Apex AMP is a NVIDIA library that maximizes the benefit of mixed precision and Tensor Core usage for a given network.

Q: What is the difference between multi-GPU and multi-node training?

A: Multi-node is an abstraction of multi-GPU training, which requires a distributed compute cluster, where each node can have multiple GPUs. Multi-node training is needed to scale training beyond a single node to large amounts of GPUs.

From the framework perspective, nothing changes from moving to multi-node training. However, a master address and port needs to be set up for inter-node communication. Multi-GPU training will then be launched on each node with passed information. You might also consider the underlying inter-node network topology and type to achieve full performance, such as HPC-style hardware such as NVLink, InfiniBand networking, or Ethernet.

Chapter 13. Exporting Models

Deep learning model development for Conversational AI is complex. It involves defining, building, and training several models in specific domains; experimenting several times to get high accuracy, fine tuning on multiple tasks and domain specific data, ensuring training performance and making sure the models are ready for deployment to inference applications.

Modules fine-tuned or trained in NeMo can be exported for efficient deployment in a variety of formats. The `nemo.core.neural_factory.NeuralModuleFactory` contains the API

```
deployment_export(module, output: str, d_format: nemo.core.neural_factory.DeploymentFormat,
 input_example=None, output_example=None)
```

which handles the export.

Currently, the `nemo.core.neural_factory.NeuralModuleFactory` API provides several deployment format targets:

- `ONNX`
- `TORCHSCRIPT`
- `TRTONNX`
- `PYTORCH`
- `JARVIS`

The recommended format to export your modules is `ONNX`. However, not all modules may support export to all formats. `PYTORCH` target will work for all modules and will simply save the module's weights in a PyTorch checkpoint.

For core ASR models such as Jasper and QuartzNet, all 4 formats will work. Example scripts for export to NVIDIA Jarvis ASR service can be found under the `scripts` folder in the NeMo repository.

QuartzNetModel.ipynb is an example of a NeMo model. For ASR, specifically, a NeMo model is a kind of neural module which contains other neural modules inside it. The NeMo model can have other neural modules inside and their mode, and topology of connections can depend on the mode in which the NeMo model is used (training or evaluation). Exporting to `.nemo` file greatly simplifies the experience of deployment with Jarvis.

Chapter 14. Recommendations For Optimization And FAQs

This section is to help guide your decision making by answering our most asked NeMo questions.

Q: Are there areas where performance can be increased?

A: You should try using mixed precision for improved performance. Note that typically when using mixed precision, memory consumption is decreased and larger batch sizes could be used to further improve the performance.

When fine-tuning ASR models on your data, it is almost always possible to take advantage of NeMo's pre-trained modules. Even if you have a different target vocabulary, or even a different language; you can still try starting with pre-trained weights from Jasper or QuartzNet `encoder` and only adjust the `decoder` for your needs.

Q: What is the recommended sampling rate for ASR?

A: The released models are based on 16 KHz audio, therefore, ensure you use models with 16 KHz audio. Reduced performance should be expected for any audio that is up-sampled from a sampling frequency less than 16 KHz data.

Q: How do we use this toolkit for audio with different types of compression and frequency than the training domain for ASR?

A: You have to match the compression and frequency.

Q: How do you replace the 6-gram out of the ASR model with a custom language model? What is the language format supported in NeMo?

A: NeMo's Beam Search decoder with Levenberg-Marquardt (LM) neural module supports the KenLM language model.

- This Jupyter notebook tutorial demonstrates the usage of the language model.
- Instructions on using KenLM

- You should retrain the KenLM language model on your own dataset. To do so, there are two options:
  - Refer to KenLM's documentation.
  - Refer to the build_6-gram_OpenSLR_lm.sh script and adjust to use your own dataset.
- If you want to use a different language model, other than KenLM, you will need to implement a corresponding decoder module.
- Transformer-XL example is present in OS2S. It would need to be updated to work with NeMo. Here is the code.

Q: How do I use text-to-speech (TTS)?

A:

- Obtain speech data ideally at 22050 Hz or alternatively at a higher sample rate and then down sample to 22050 Hz.
  - If less than 22050 Hz and above 16000 Hz:
    - Retrain WaveGlow on your own dataset.
    - Tweak the spectrogram generation parameters, namely the `window_size` and the `window_stride` for their fourier transforms.
  - For below 16000 Hz, look into obtaining new data.
- In terms of bitrate/quantization, the general advice is the higher the better. We have not experimented enough to state how much this impacts quality.
- For the amount of data, again the more the better, and the more diverse in terms of phonemes the better. Aim for around 20 hours of speech after filtering for silences and non-speech audio.
- Most open speech datasets are in ~10 second format so training Tacotron 2 on audio on the order of 10s - 20s per sample is known to work. Additionally, the longer the speech samples, the more difficult it will be to train Tacotron 2.
- Audio files should be clean. There should be little background noise or music. Data recorded from a studio mic is likely to be easier to train compared to data captured using a phone.
- To ensure pronunciation of words are accurate; the technical challenge is related to the dataset, text to phonetic spelling is required, use phonetic alphabet (notation) that has the name correctly pronounced.
- Here are some example parameters you can use to train Tacotron 2:
  - Tacotron 2 requires a single speaker dataset
  - Use AMP level O0
  - Trim long silences in the beginning and end
  - `optimizer="adam"`
  - `beta1 = 0.9`

- beta2 = 0.999
- lr=0.001 (constant)
- amp_opt_level="O0"
- weight_decay=1e-6
- batch_size=48 (per GPU)
- trim_silence=True

Chapter 15. Resources

Ensure you are familiar with the following resources for NeMo.

Developer blogs

- How to Build Domain Specific Automatic Speech Recognition Models on GPUs
- Develop Smaller Speech Recognition Models with NVIDIA's NeMo Framework
- Neural Modules for Fast Development of Speech and Language Models

Domain specific, transfer learning, Docker container with Jupyter Notebooks

Domain Specific NeMo ASR Application

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.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

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

ARM

ARM, AMBA and ARM Powered are registered trademarks of ARM Limited. Cortex, MPCore and Mali are trademarks of ARM Limited. All other brands or product names are the property of their respective holders. "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.

OpenCL

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

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

Trademarks

NVIDIA, the NVIDIA logo, and cuBLAS, CUDA, CUDA Toolkit, cuDNN, DALI, DIGITS, DGX, DGX-1, DGX-2, DGX Station, DLProf, GPU, JetPack, Jetson, Kepler, Maxwell, NCCL, NeMo, Nsight Compute, Nsight Systems, NVCaffe, NVIDIA Ampere GPU architecture, NVIDIA Deep Learning SDK, NVIDIA Developer Program, NVIDIA GPU Cloud, NVLink, NVSHMEM, PerfWorks, Pascal, SDK Manager, T4, Tegra, TensorRT, TensorRT Inference Server, Tesla, TF-TRT, Transfer Learning Toolkit, Triton Inference Server, Turing, and Volta are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

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