NVIDIA

NVIDIA NeMo Framework

Fundamentals and Multitask

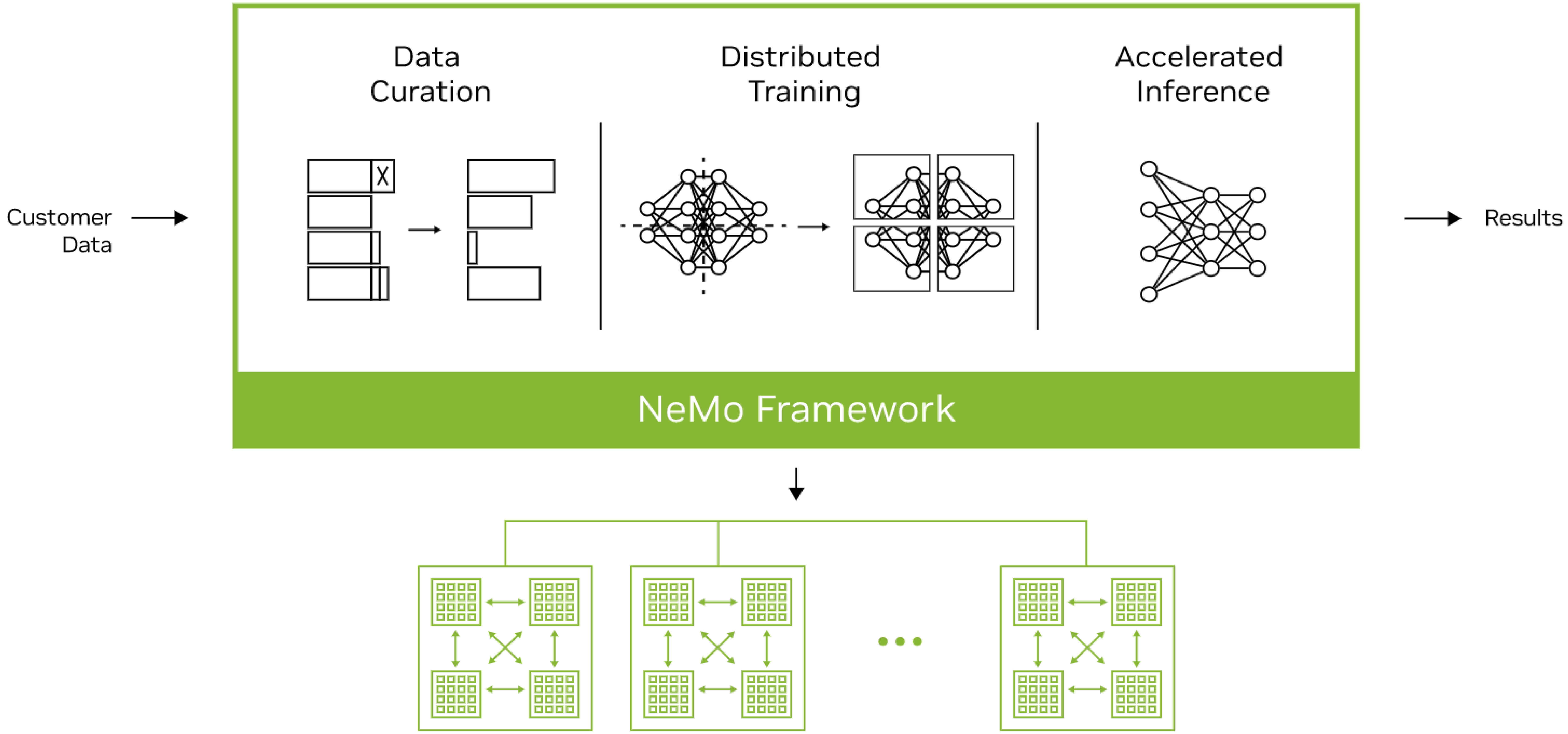
Agenda

- NeMo Framework
- Fundamentals of NeMo
- Multitask Prompt and P-tuning

NeMo Framework Fundamentals

Introduction

- NVIDIA NeMo™ framework is an end-to-end, cloud-native enterprise framework to build, customize, and deploy generative AI models with billions of parameters.



NeMo Framework Fundamentals

Models for different modalities

- The framework comes with extendable collections of pre-built modules and ready-to-use models for automatic speech recognition (ASR), natural language processing (NLP) and text synthesis (TTS).
- Built for speed, NeMo can utilize NVIDIA's Tensor Cores and scale out training to multiple GPUs and multiple nodes.

Language

- BERT
- GPT-3
- T5
- T5-MoE
- Inform

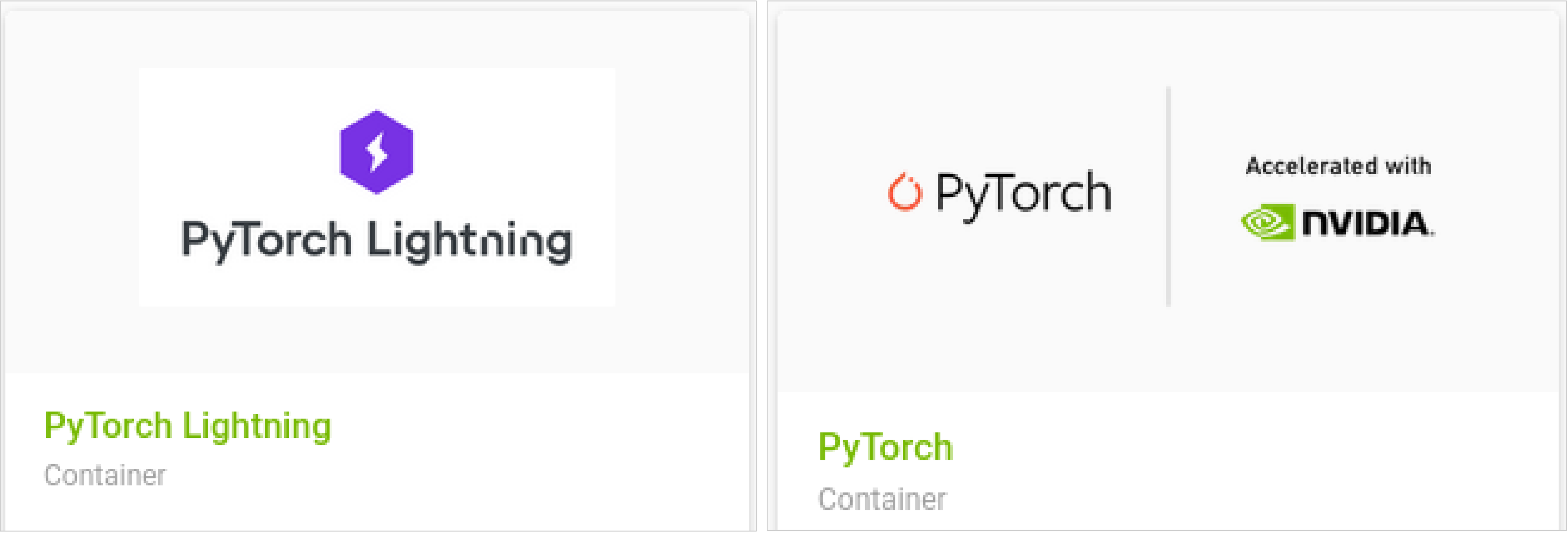
Multimodal

- Stable Diffusion v1.5
- VisionTransformers (ViT)
- CLIP
- Instruct-Pix2Pix
- Imagen

NeMo Framework Fundamentals

NeMo Models

- NeMo models leverage PyTorch Lightning Module and are compatible with the entire PyTorch ecosystem.



NeMo Framework Fundamentals

NeMo Models

- A "Model" is the neural network(s) as well as all the infrastructure supporting those network(s), wrapped into a singular, cohesive unit.
- As such, all NeMo models are constructed to contain the following:
  - Neural Network architecture
  - Dataset + Data Loaders
  - Preprocessing + Postprocessing
  - Optimizer + Schedulers
  - Any other supporting infrastructure

NeMo Framework Fundamentals

NeMo Models in Collections

```
import nemo.collections.asr as nemo_asr
import nemo.collections.nlp as nemo_nlp
import nemo.collections.tts as nemo_tts
```

```
asr_models = [model for model in dir(nemo_asr.models) if model.endswith("Model")]
```

```
nlp_models = [model for model in dir(nemo_nlp.models) if model.endswith("Model")]
```

```
tts_models = [model for model in dir(nemo_tts.models) if model.endswith("Model")]
```

```
citrinet = nemo_asr.models.EncDecCTCModelBPE.from_pretrained('stt_en_citrinet_512')
```

Fundamentals of NeMo

NeMo Models

Model Configuration using OmegaConf

- OmegaConf is an excellent library that is used throughout NeMo to enable yaml configuration management more easily.
- All NeMo models come packaged with their model configuration inside the cfg attribute
- The cfg is an essential tool to modify the behavior of the Model after it has been constructed.

```
from omegaconf import OmegaConf
import copy
cfg = copy.deepcopy(citrinet.cfg)
print(OmegaConf.to_yaml(cfg))
```

NeMo Models

Model Config Content

```yaml
train_ds:
  manifest_filepath: null
  sample_rate: 16000
  batch_size: 32
  trim_silence: true
  max_duration: 16.7
  shuffle: true
  is_tarred: false
  tarred_audio_filepaths: null
validation_ds:
  manifest_filepath: null
  sample_rate: 16000
  batch_size: 32
  shuffle: false
test_ds:
  manifest_filepath:
  - /home/smajumdar/PycharmProjects/nemo-eval/nemo_eval/librispeech/manifests/dev
  sample_rate: 16000
  batch_size: 32
  shuffle: false
  num_workers: 12
  pin_memory: true
model_defaults:
  repeat: 5
  dropout: 0.0
  separable: true
  se: true
  se_context_size: -1
tokenizer:
  dir: /home/smajumdar/PycharmProjects/nemo-eval/nemo_beta_eval/asrset/manifests/
unigram_v1024/
  type: bpe
preprocessor:
  _target_: nemo.collections.asr.modules.AudioToMelSpectrogramPreprocessor
  sample_rate: 16000
  normalize: per_feature
  window_size: 0.025
  window_stride: 0.01
  window: hann
  features: 80
  n_fft: 512
  frame_splicing: 1
  dither: 1.0e-05
  pad_to: 16
  stft_conv: false
spec_augment:
  _target_: nemo.collections.asr.modules.SpectrogramAugmentation
  freq_masks: 2
  time_masks: 10
  freq_width: 27
```

```yaml
encoder:
  _target_: nemo.collections.asr.modules.ConvASREncoder
  feat_in: 80
  activation: relu
  conv_mask: true
  jasper:
  - filters: 512
    repeat: 1
    kernel:
    - 5
    stride:
    - 1
    dilation:
    - 1
```

```yaml
decoder:
  _target_: nemo.collections.asr.modules.ConvASRDecoder
  feat_in: 640
  num_classes: 1024
  vocabulary:
  - <unk>
  - s
  - ▁the
  - t
  - ▁a
```

NeMo Models

Model Config Content


```yaml
optim:
  name: novograd
  lr: 0.05
  betas:
  - 0.8
  - 0.25
  weight_decay: 0.001
  sched:
    name: CosineAnnealing
    warmup_steps: 1000
    warmup_ratio: null
    min_lr: 1.0e-09
    last_epoch: -1
target: nemo.collections.asr.models.ctc_bpe_models.EncDecCTCModelBPE
nemo_version: 1.17.0
decoding:
  strategy: greedy
  preserve_alignments: null
  compute_timestamps: null
  word_seperator: ' '
  ctc_timestamp_type: all
  batch_dim_index: 0
  greedy:
    preserve_alignments: false
    compute_timestamps: false
    preserve_frame_confidence: false
    confidence_method_cfg: null
  beam:
    beam_size: 4
    search_type: default
    preserve_alignments: false
    compute_timestamps: false
    return_best_hypothesis: true
    beam_alpha: 1.0
    beam_beta: 0.0
    kenlm_path: null
    flashlight_cfg:
      lexicon_path: null
      beam_size_token: 16
      beam_threshold: 20.0
      unk_weight: -.inf
      sil_weight: 0.0
      unit_lm: false
    pyctcdecode_cfg:
      beam_prune_logp: -10.0
      token_min_logp: -5.0
      prune_history: false
      hotwords: null
      hotword_weight: 10.0
  confidence_cfg:
```

NeMo Models

Components of the Model

- NeMo has special utilities for a few components. They are:
  - setup_training_data
  - setup_validation_data and setup_multi_validation_data
  - setup_test_data and setup_multi_test_data
  - setup_optimization

NeMo Models

Optimizer & Scheduler Config

- **Optim structure**

```yaml
optim:
    name: novograd
    lr: 0.01

    # optimizer arguments
    betas: [0.8, 0.25]
    weight_decay: 0.001

    # scheduler setup
    sched:
      name: CosineAnnealing

      # Optional arguments
      max_steps: -1 # computed at runtime or explicitly set here
      monitor: val_loss
      reduce_on_plateau: false

      # scheduler config override
      warmup_steps: 1000
      warmup_ratio: null
      min_lr: 1e-9
```

NeMo Models

Saving and restoring models

save_to

```
citrinet.save_to('citrinet_512.nemo')
```

restore_from

```
temp_cn = nemo_asr.models.EncDecCTCModelBPE.restore_from('citrinet_512.nemo')
```

load_from_checkpoint

```
citrinet = nemo_asr.models.EncDecCTCModelBPE.load_from_checkpoint(<path to checkpoint>)
```

NeMo Models

NeMo with Hydra

- Hydra is used throughout NeMo as a way to enable rapid prototyping using predefined config files.
- While all NeMo models come with at least 1 default config file, one might want to switch configs without changing code.
- This is easily achieved by the following commands:

```
--config-path:
--config-name:
```

NeMo Models

NeMo with Hydra: Overriding config from the command line

- Hydra allows users to provide command line overrides to any part of the config.
- There are three cases to consider:
  - Override existing value in config
  - Add new value in config
  - Remove old value in config

Overriding existing values in config: change the optimizer from novograd to adam

```
$ python <script>.py \
  --config-path="dir to config" \
  --config-name="name of config" \
  model.optim.name="adam" \
  model.optim.betas=[0.9,0.999]
```

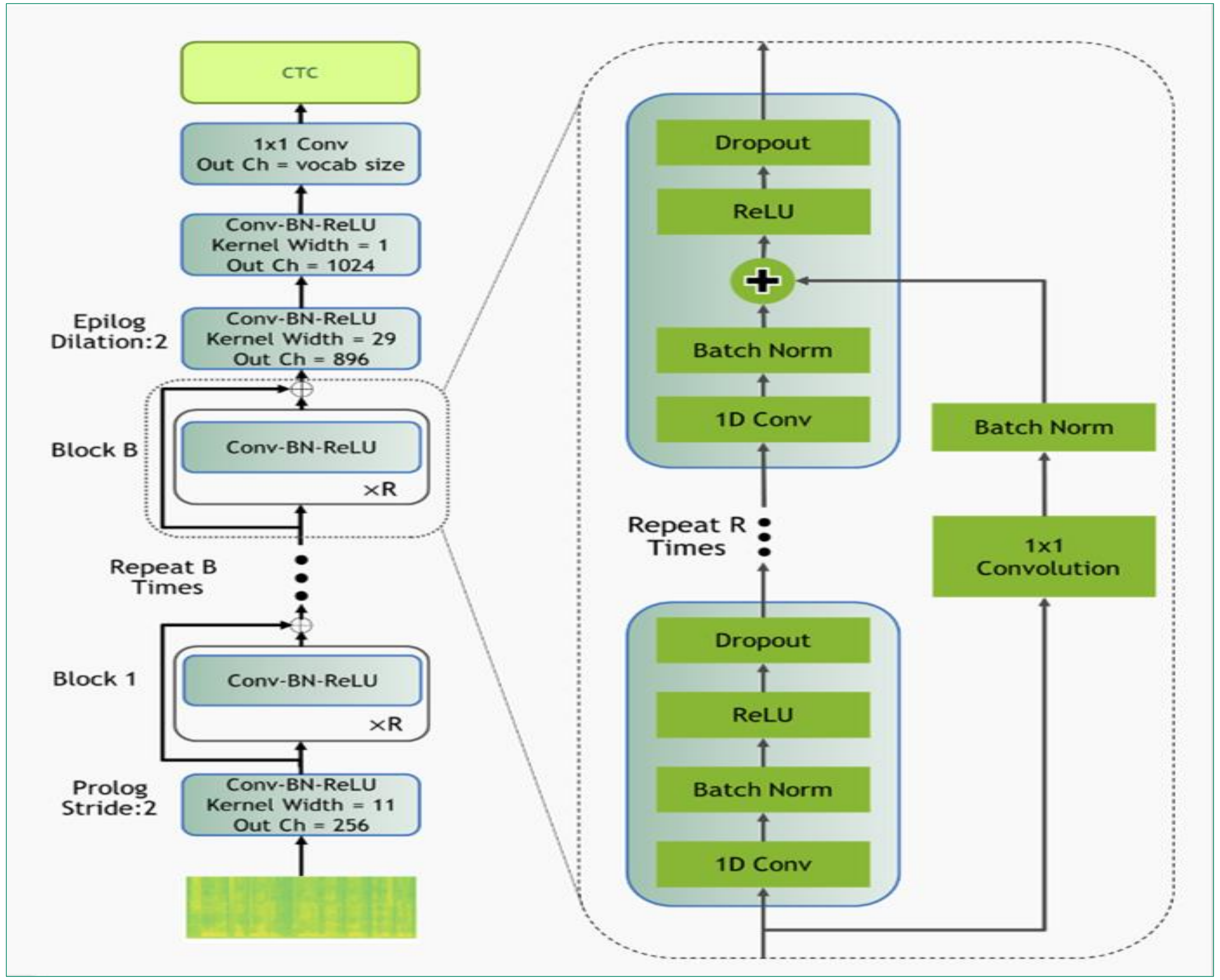
NeMo Examples

ASR Models

- NeMo supports multiple Speech Recognition models such as:
  - Jasper
  - QuartzNet
  - Citrinet
  - ContextNet
  - Conformer-CTC
  - Conformer-Transducer
  - Fast-Conformer
  - Cache-aware Streaming Conformer
  - LSTM-Transducer
  - LSTM-CTC
  - Squeezeformer-CTC
  - Hybrid-Transducer-CTC. All of these can be trained on various datasets

ASR Models

Jasper ("Just Another Speech Recognizer")



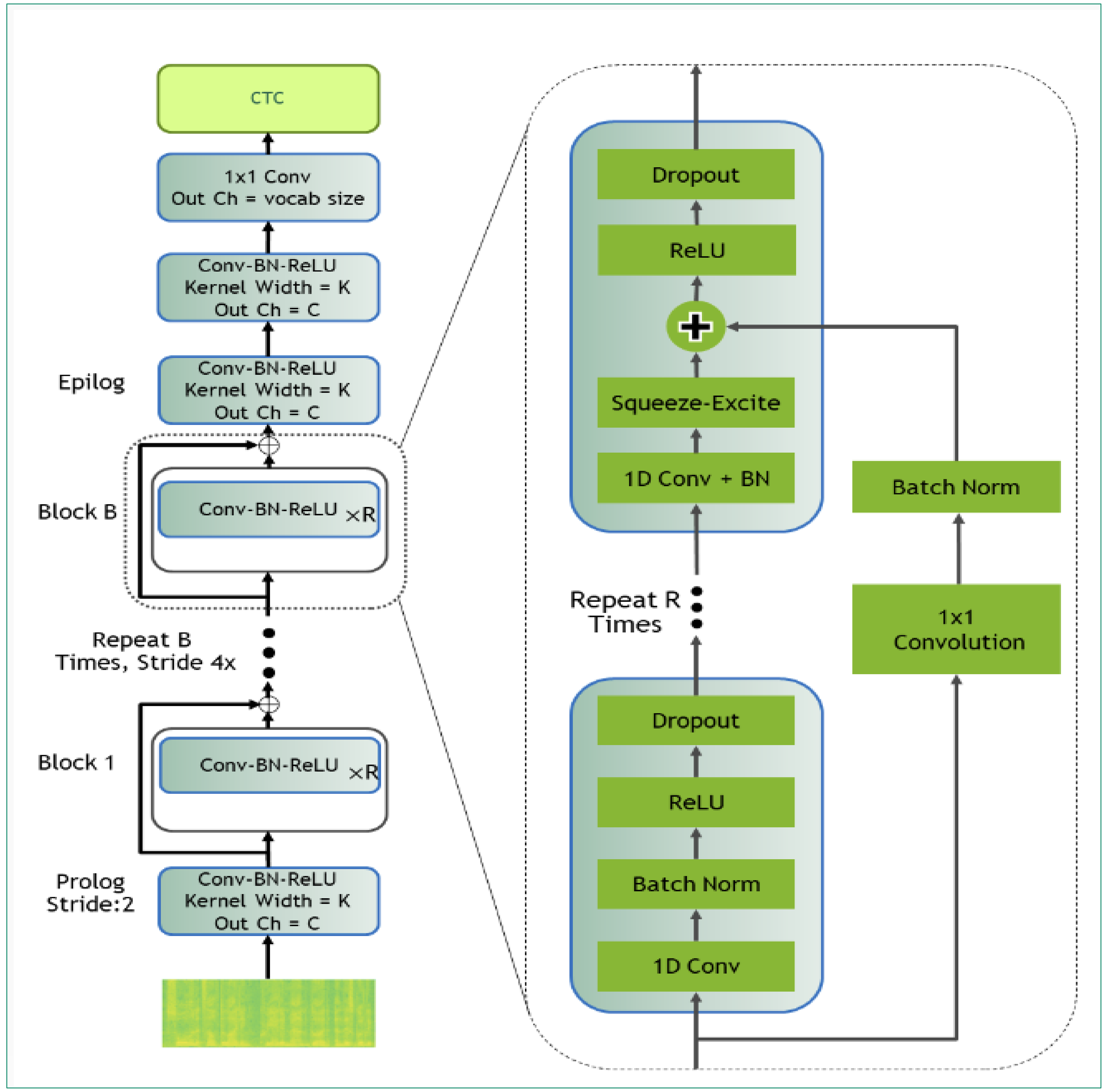
Jasper Architecture

- The Jasper family of models are denoted as **Jasper_[BxR]**
- Jasper models can be instantiated using the **EncDecCTCModel** class.

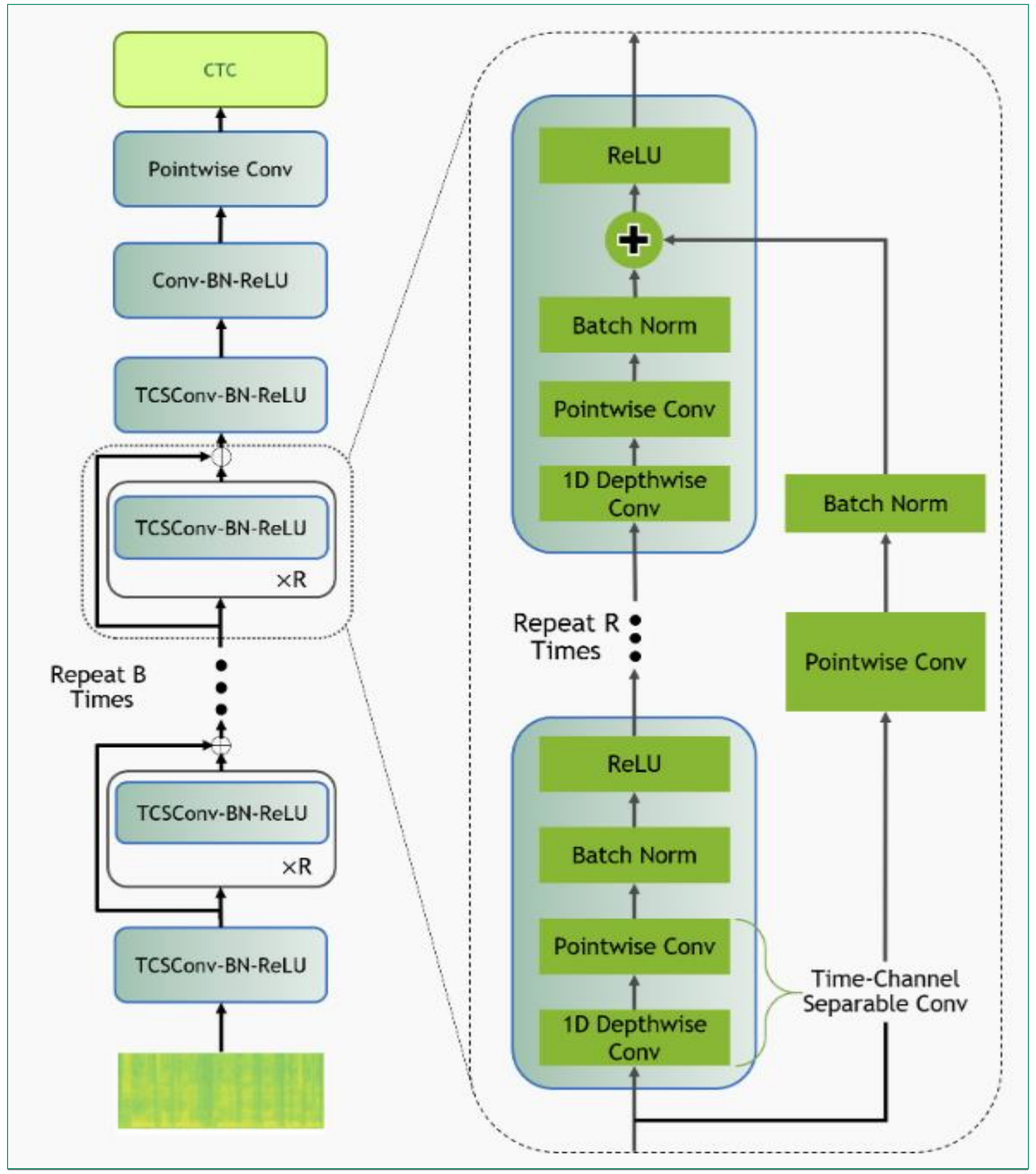
Source: https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/stable/asr/models.html

ASR Models

Citrinet



Citrinet Arhictecture

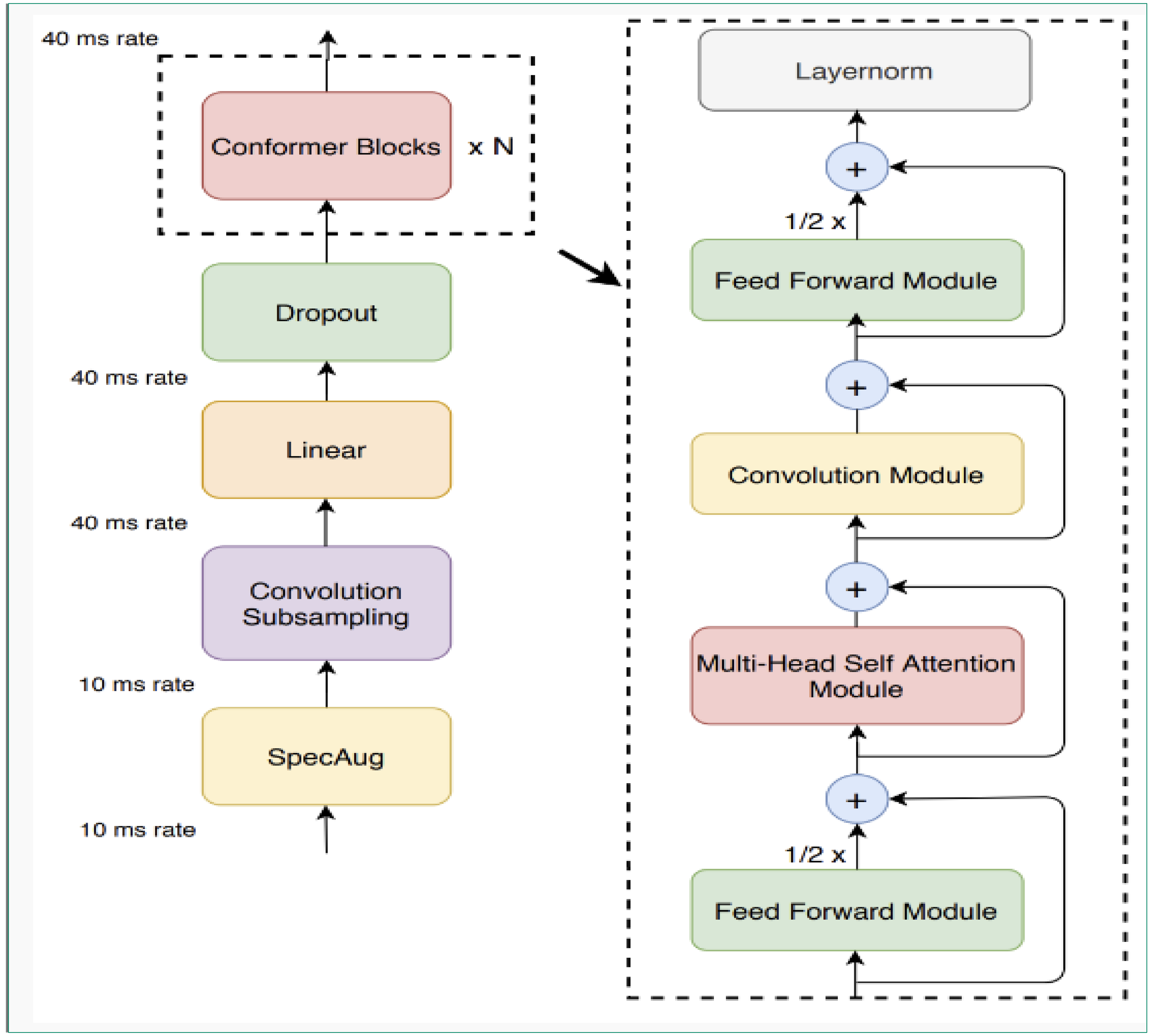


QuartzNet

Source: https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/stable/asr/models.html

ASR Models

Conformer-CTC



Overall architecture of Conformer-CTC encoder

Source: https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/stable/asr/models.html

NeMo Examples

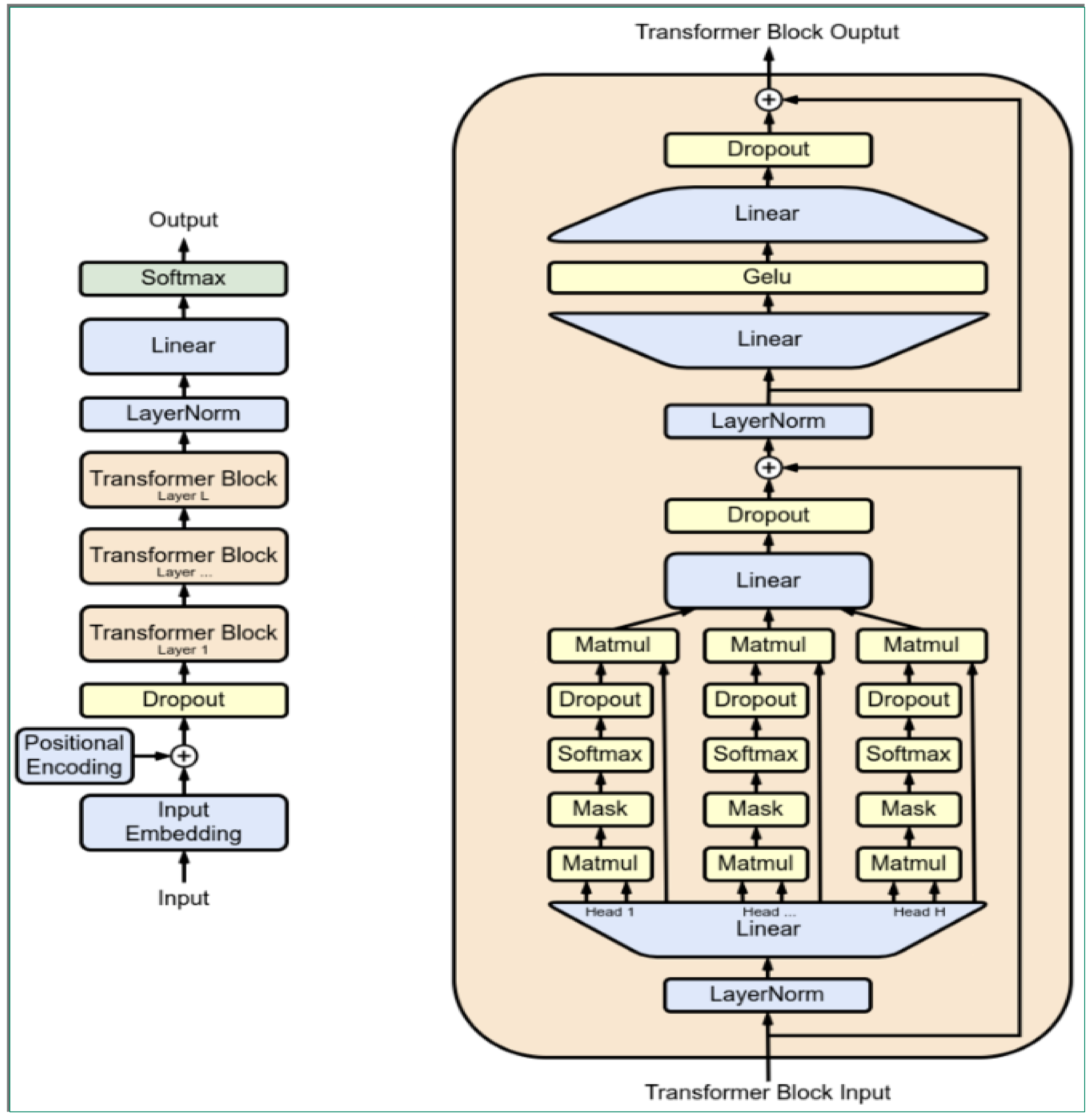
NLP

- **NeMo supports a wide variety of tasks in NLP**:
  - Punctuation And Capitalization Models
  - Token Classification (Named Entity Recognition) Model
  - Joint Intent and Slot Classification
  - Text Classification model
  - BERT
  - Language Modeling
  - Prompt Learning
  - Question Answering
  - Dialogue tasks
  - Dialogue tasks
  - GLUE Benchmark
  - Information Retrieval
  - Entity Linking
  - Model NLP
  - Machine Translation Models

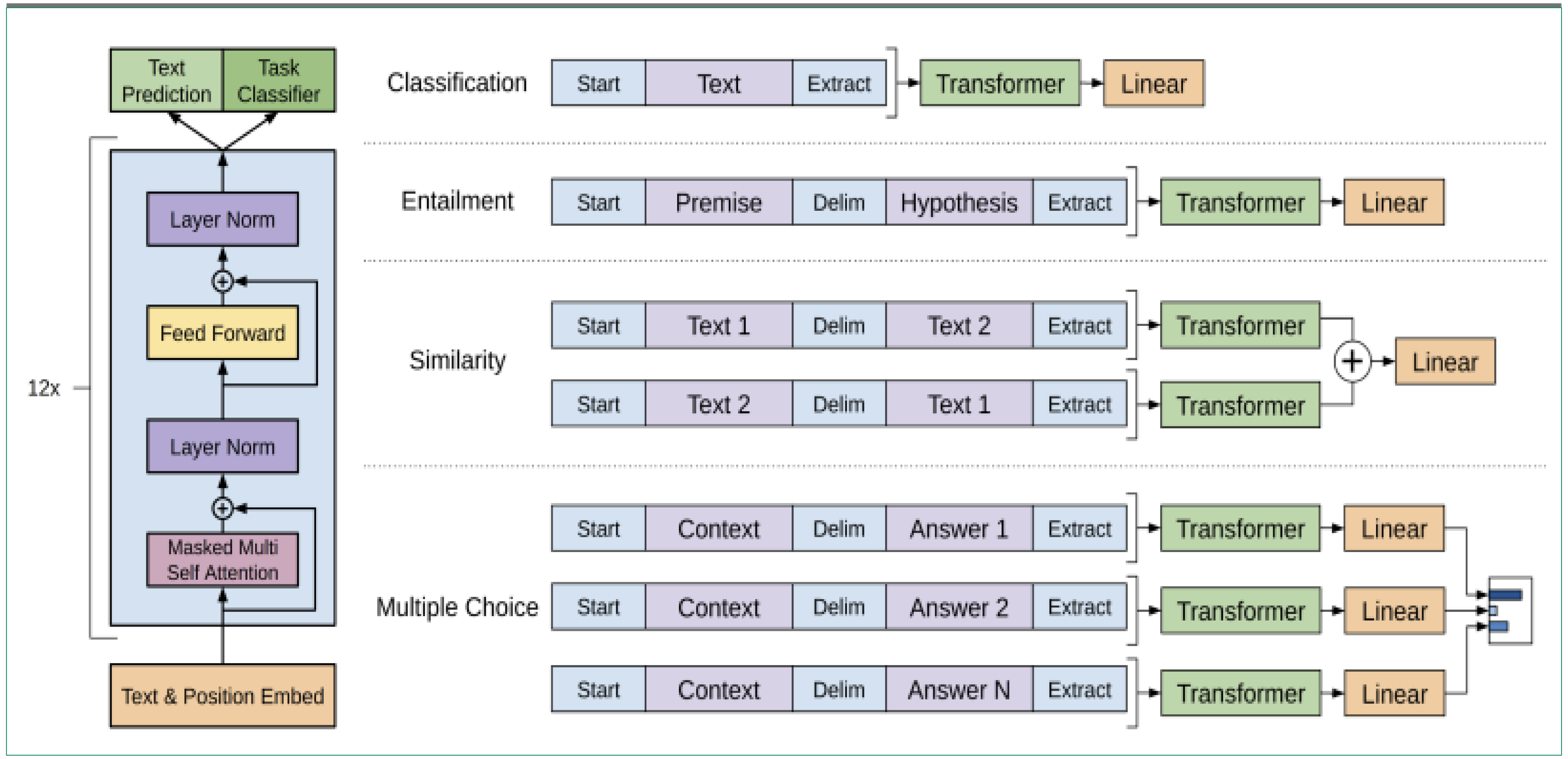
- **List of tokenizers**:
  - WordPiece tokenizer,
  - SentencePiece tokenizer or
  - simple tokenizers like Word tokenizer.

NLP Transformer

GPT (Generative Pre-trained Transformer) Architecture



Original GPT architecture



Improved GPT architecture by OpenAI

Source: https://en.wikipedia.org/wiki/GPT-2

Source:https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf

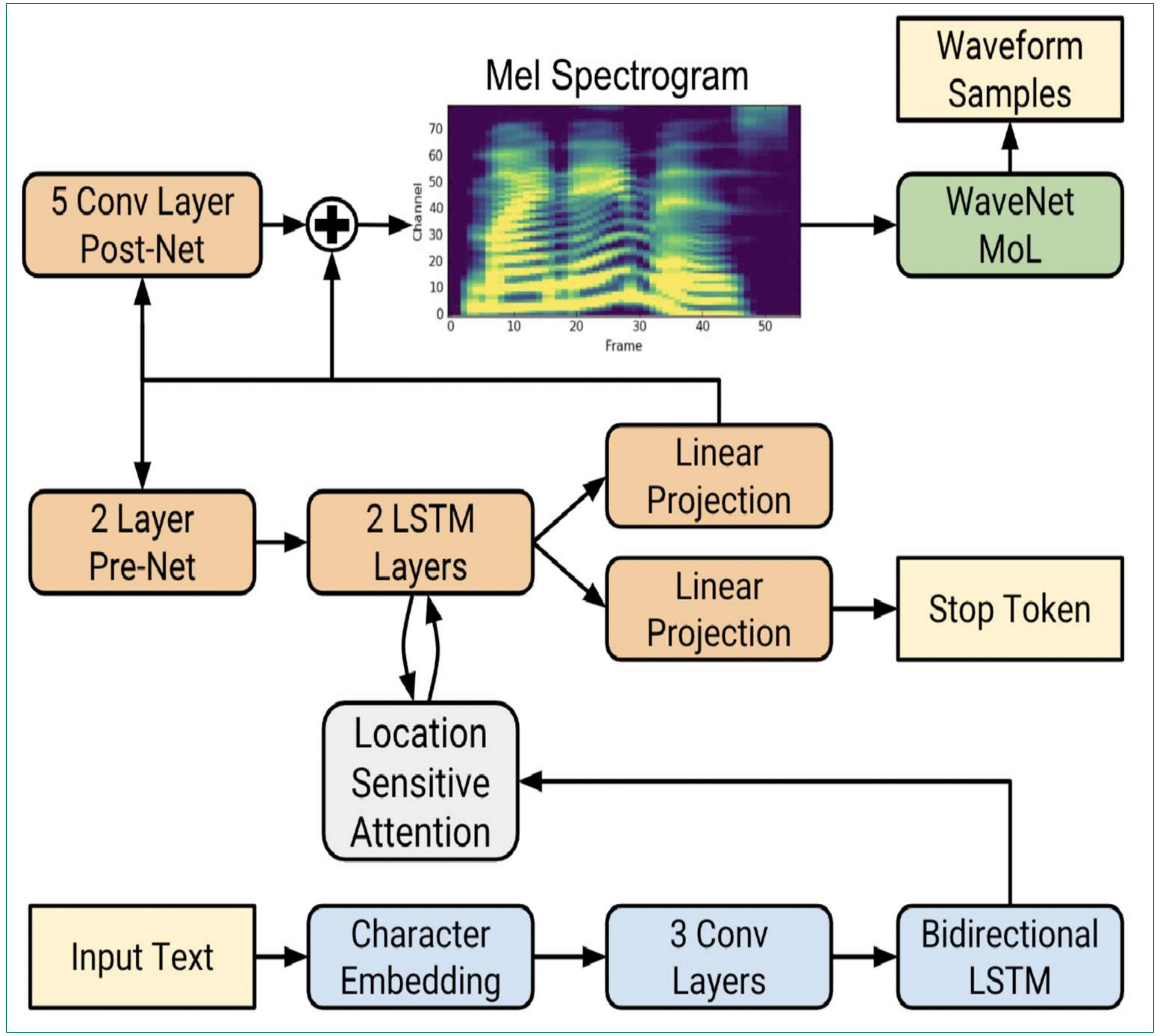
NeMo Examples

TTS Models

- NeMo supports Text To Speech (TTS, aka Speech Synthesis) via a two-step inference procedure.
- First, a model is used to generate a **mel spectrogram** from the text. Second, a model is used to generate audio from a mel spectrogram.
- Supported Models:
  - Mel Spectrogram Generators:
    - Tacotron2
    - FastPitch
    - Talknet
    - And more…
  - Audio Generators (Vocoders):
    - WaveGlow
    - HiFiGAN
    - UnivNet
    - And more...

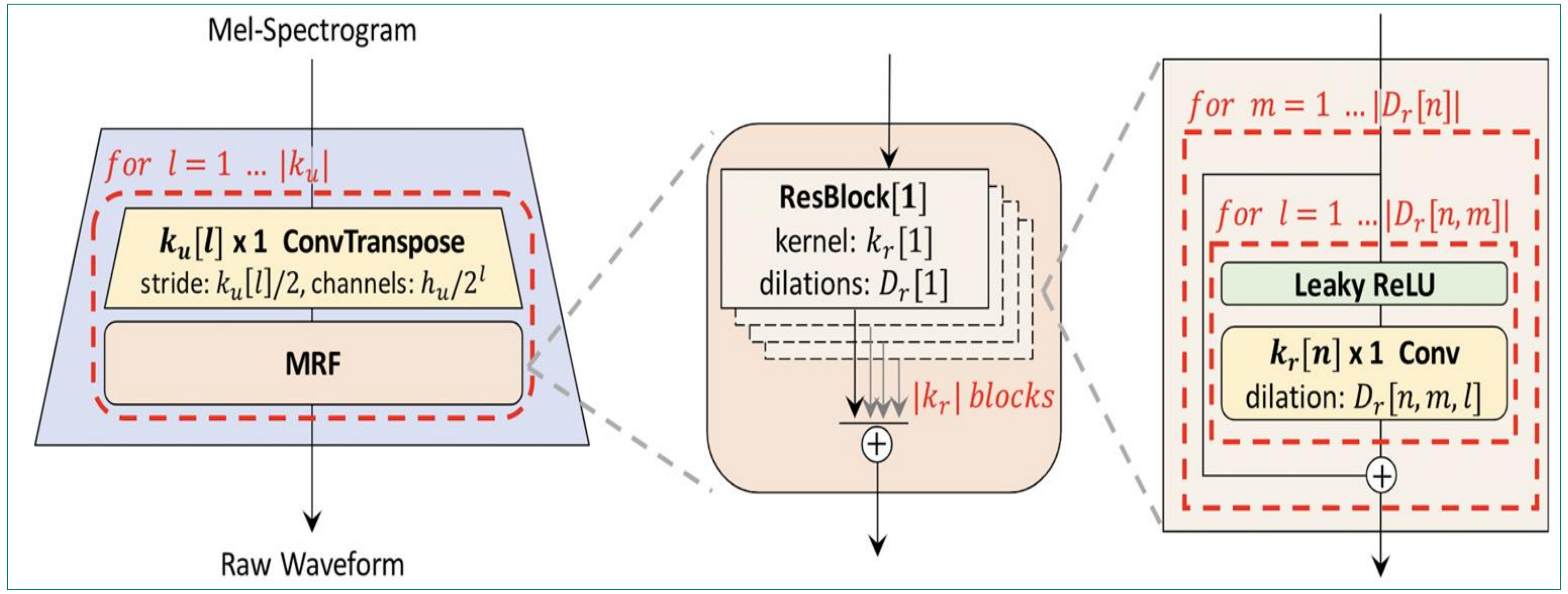
TTS Models

Tacotron 2 & HiFi-GAN Architecture



Tacotron 2

Source: https://en.wikipedia.org/wiki/GPT-2



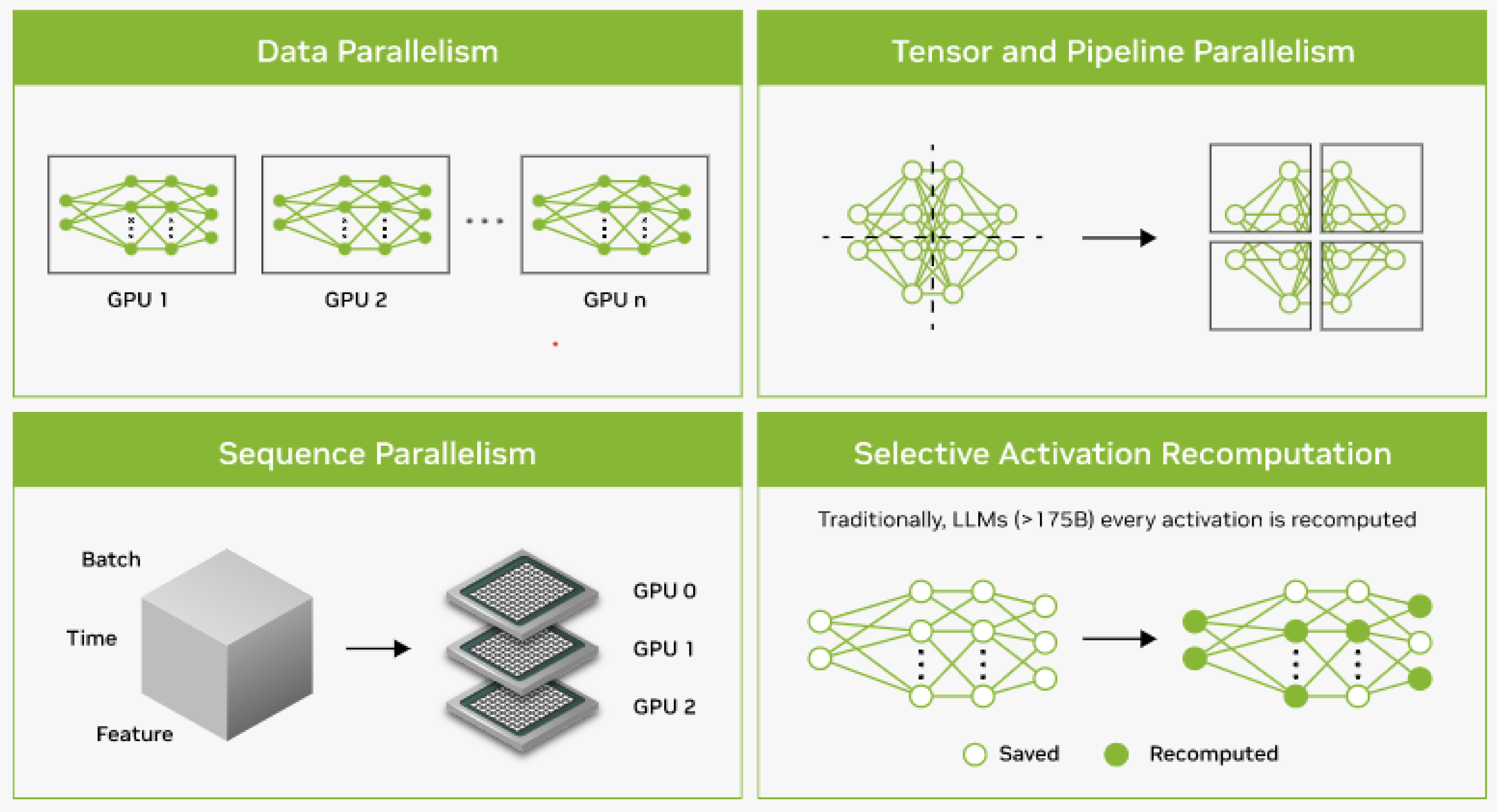
HiFi-GAN

Source:https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf

Multitask Prompt and P-tuning

NeMo Megatron

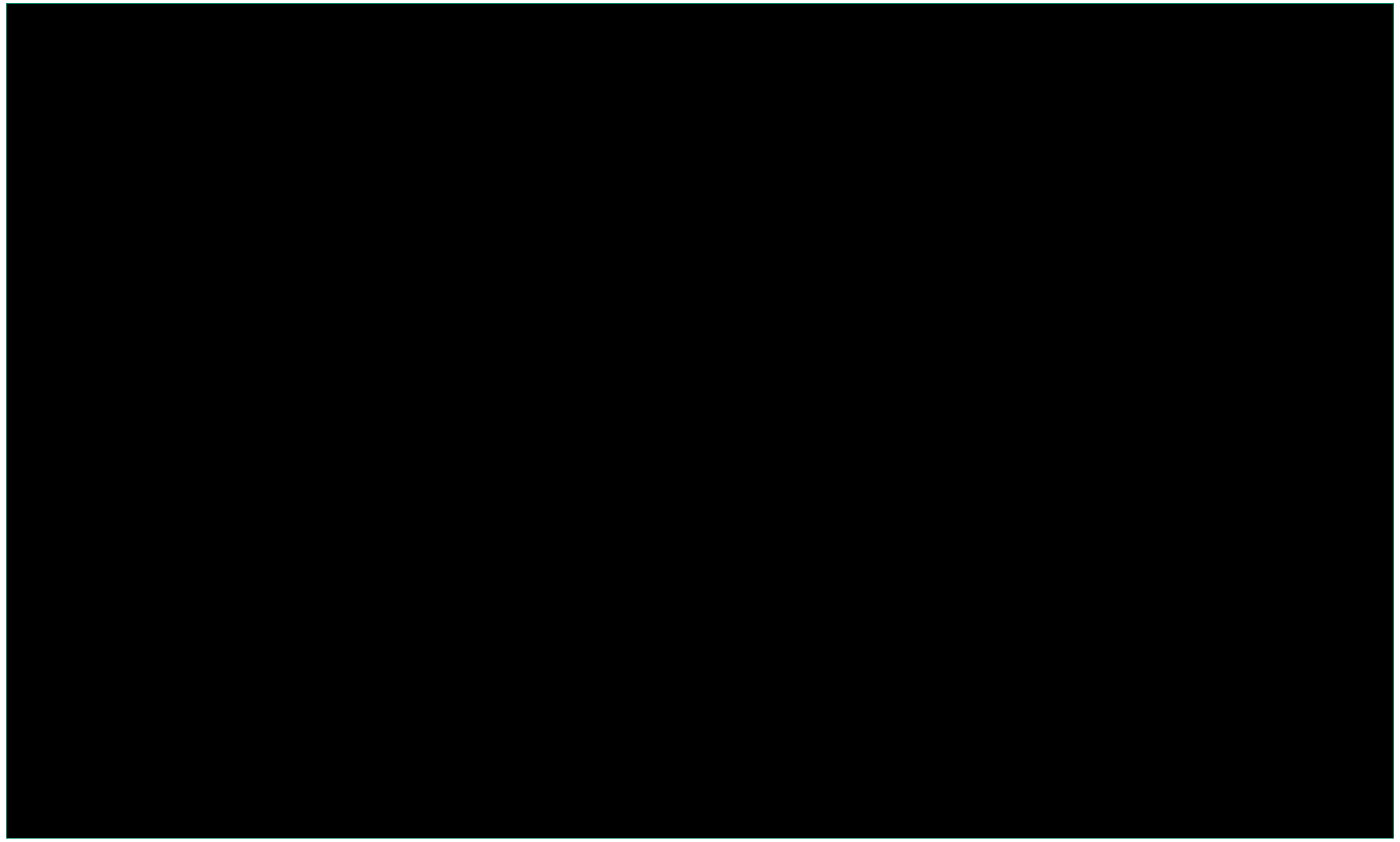
Parallelism



- The NeMo Megatron delivers high levels of training efficiency, making training of large-scale foundation models possible, using 3D parallelism techniques
- NeMo Megatron supports 4 types of parallelisms:
  - Distributed Data parallelism
  - Tensor Parallelism
  - Pipeline Parallelism
  - Sequence Parallelism

NeMo Megatron

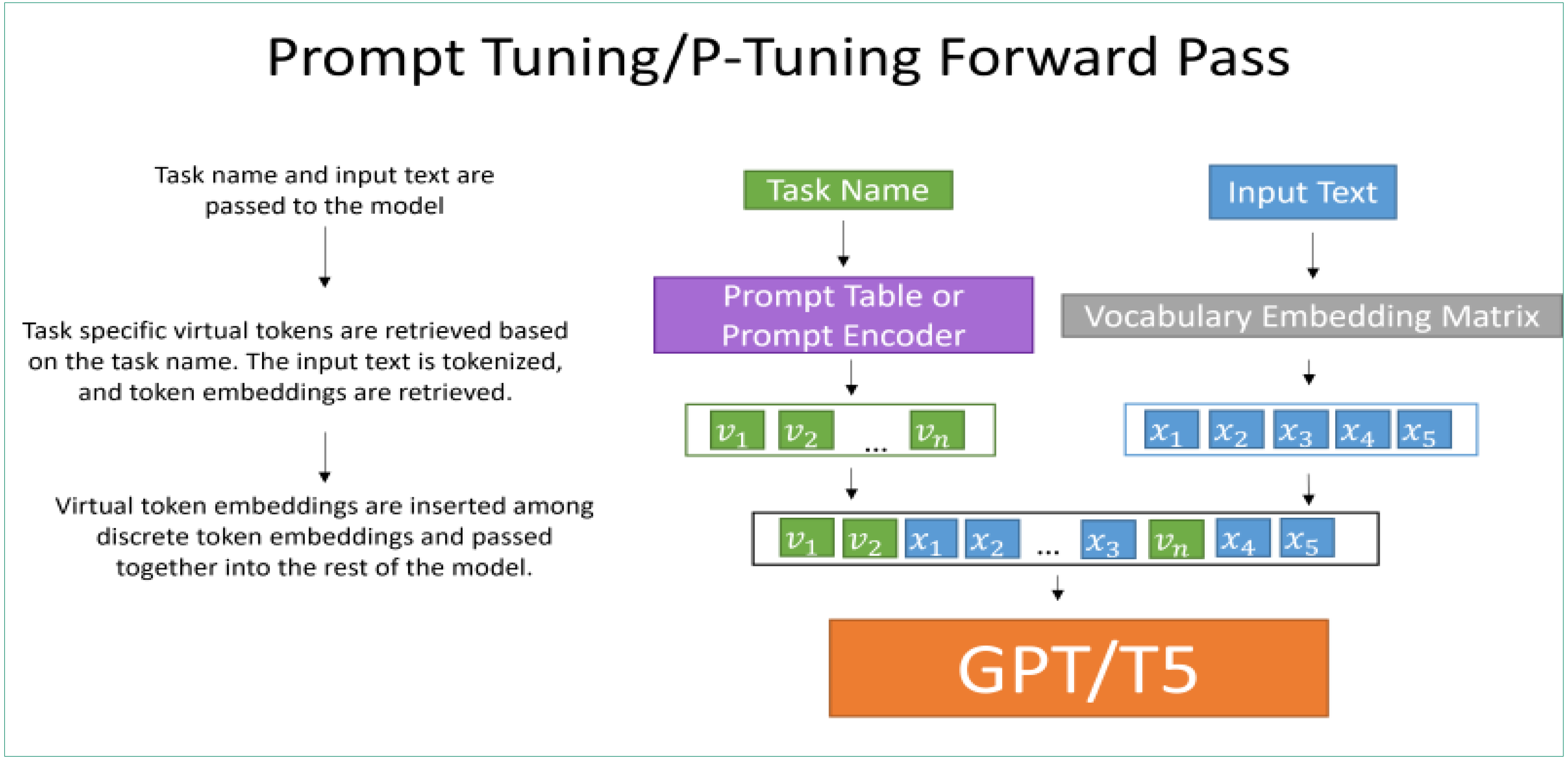
Parallelism nomenclature



- When reading and modifying NeMo Megatron code you will encounter the following terms.
  - Local and global ranks
  - Data parallel ranks
  - Tensor model parallel ranks
  - Tensor and pipeline ranks

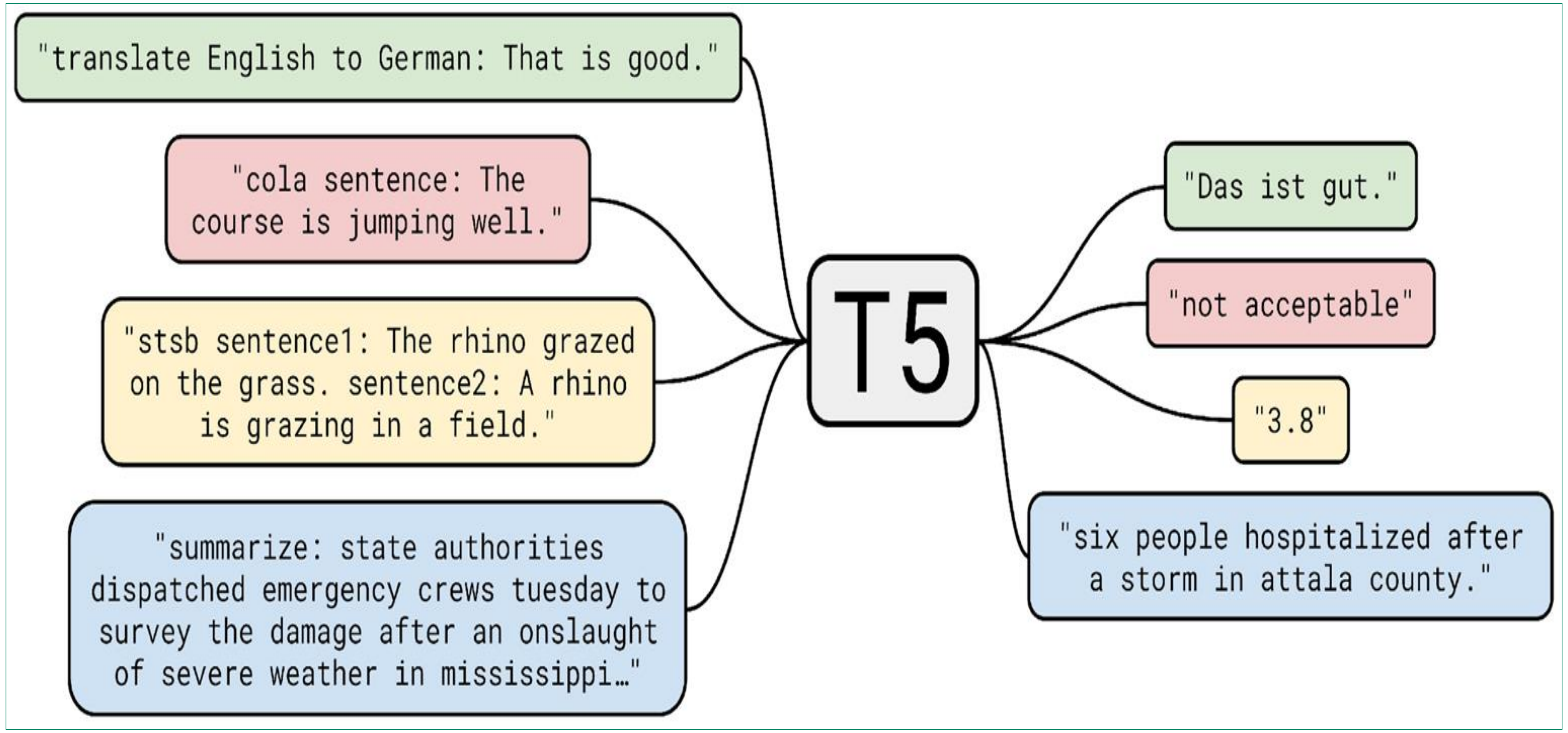
NeMo Megatron

Prompt Learning



NeMo Megatron

T5 Model (Text-to-Text Transfer Transformer)



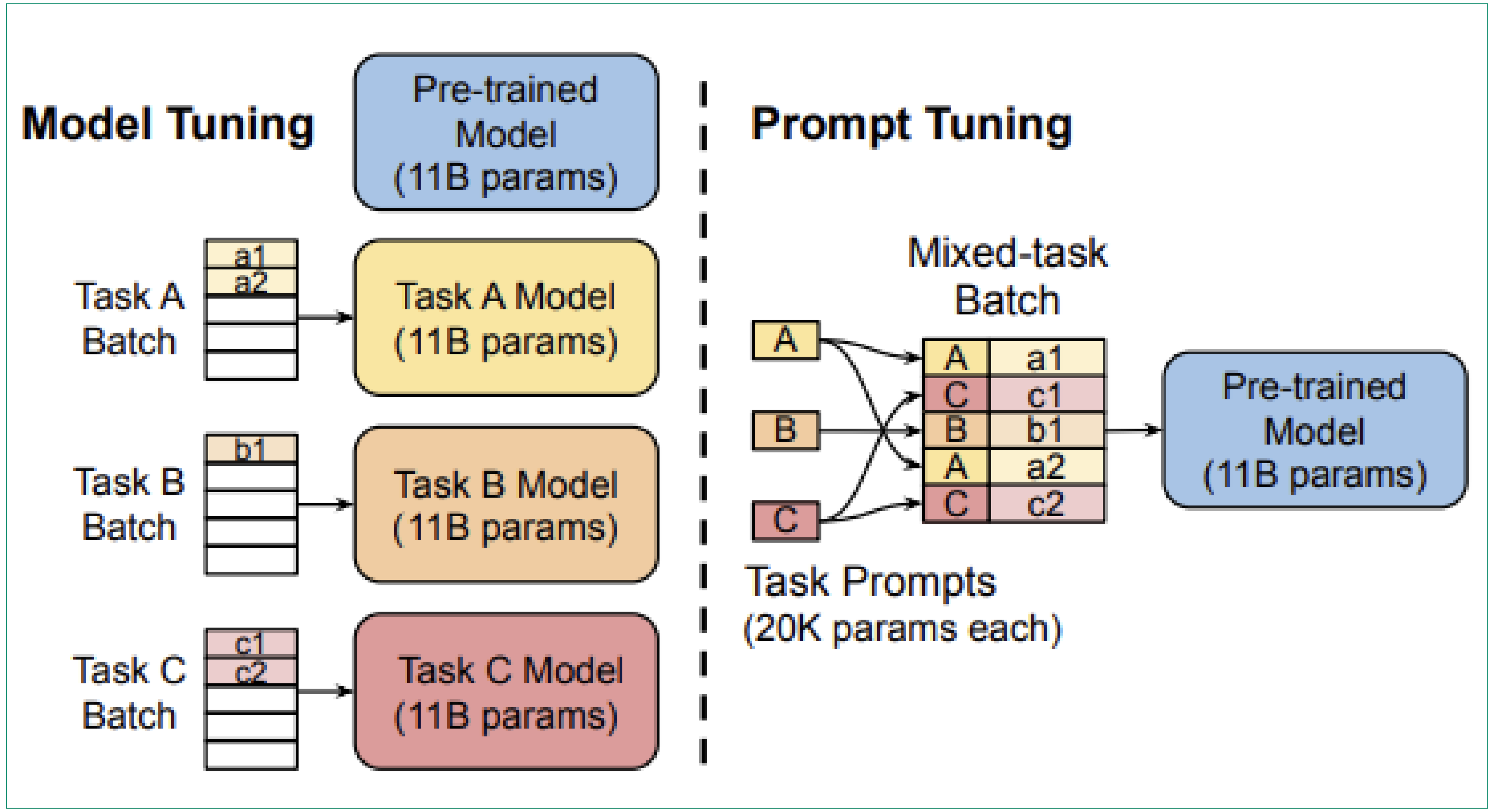
T5 Model

- T5 is all about reframing all NLP tasks into a unified text-to-text-format where the input and output are always text strings
- The T5 model, pre-trained on C4 dataset, achieves state-of-the-art results on many NLP benchmarks.

Source: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer by Raffel et al., 2020

NeMo Megatron

Prompt-Tuning



Source: The Power of Scale for Parameter-Efficient Prompt Tuning by Lester et. al's EMNLP 2021 paper (https://arxiv.org/pdf/2104.08691.pdf)

NeMo Megatron

P-Tuning

In p-tuning, an LSTM model is used to predict virtual token embeddings.

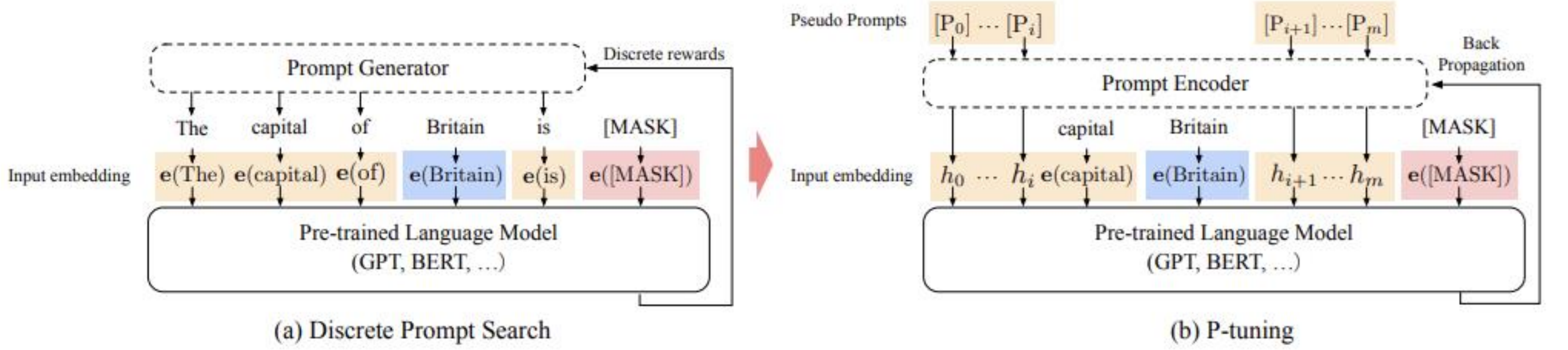


Figure 2. An example of prompt search for "The capital of Britain is [MASK]". Given the context (blue zone, "Britain") and target (red zone, "[MASK]"), the orange zone refer to the prompt tokens. In (a), the prompt generator only receives discrete rewards; on the contrary, in (b) the pseudo prompts and prompt encoder can be optimized in a differentiable way. Sometimes, adding few task-related anchor tokens (such as "capital" in (b)) will bring further improvement.

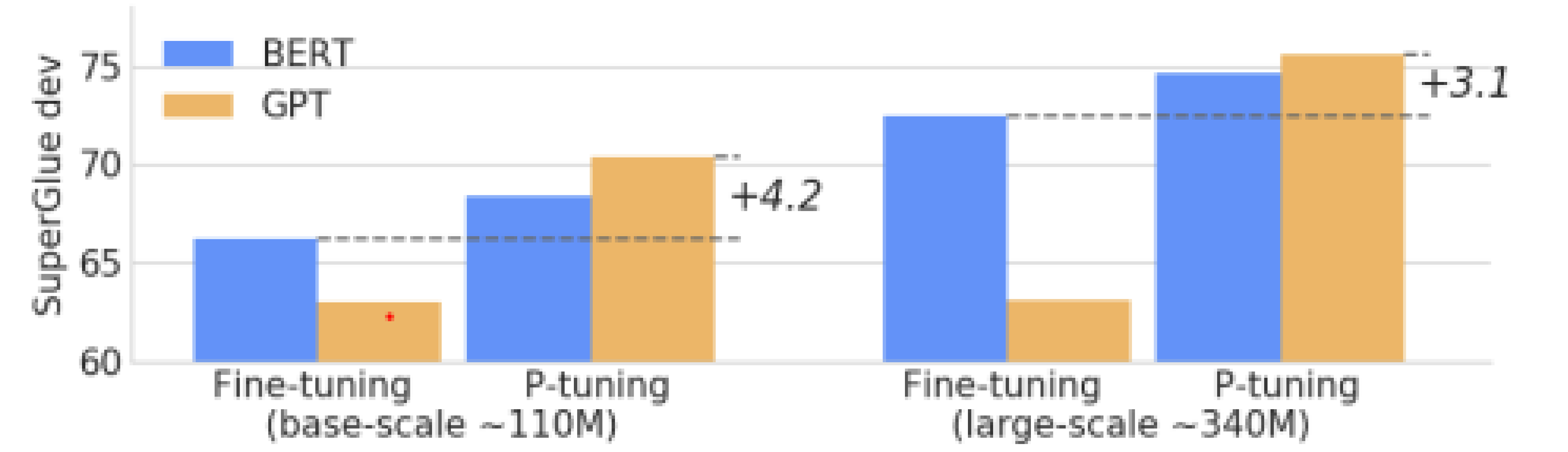


Figure 1. Average scores on 7 dev datasets of SuperGlue. GPTs can be better than similar-sized BERTs on NLU with P-tuning.

Source: GPT Understands, Too by Liu et al (https://arxiv.org/abs/2103.10385)

NeMo Megatron

Prompt tuning/P-Tuning

- A single pretrained GPT model can use both p-tuning and prompt-tuning.
- While you must decide to use either p-tuning or prompt-tuning for each task you want your model to perform, you can p-tune your model on a set of tasks A, then prompt tune your same model on a different set of tasks B, then finally run inference on tasks from both A and B at the same time.
- During prompt-tuning or p-tuning, tasks tuned at the same time must use the same number of virtual tokens.
- During inference, tasks using differing amounts of virtual tokens can be run at the same time.

NeMo Megatron

Dataset Preprocessing

```
[
  {"taskname": "squad", "context": [CONTEXT_PARAGRAPH_TEXT1], "question": [QUESTION_TEXT1], "answer": [ANSWER_TEXT1]},
  {"taskname": "squad", "context": [CONTEXT_PARAGRAPH_TEXT2], "question": [QUESTION_TEXT2], "answer": [ANSWER_TEXT2]},
  {"taskname": "intent_and_slot", "utterance": [UTTERANCE_TEXT1], "label": [INTENT_TEXT1][SLOT_TEXT1]},
  {"taskname": "intent_and_slot", "utterance": [UTTERANCE_TEXT2], "label": [INTENT_TEXT2][SLOT_TEXT2]},
  {"taskname": "sentiment", "sentence": [SENTENCE_TEXT1], "label": [SENTIMENT_LABEL1]},
  {"taskname": "sentiment", "sentence": [SENTENCE_TEXT2], "label": [SENTIMENT_LABEL2]},
]
```

NeMo Megatron

SQUAD Dataset Prompt Formatting

```
{"taskname": "squad", "context": "Super Bowl 50 was an American football ga... numerals 50.",
"question": "What does AFC stand for?", "answer": "American Football Conference“ }.
```

```
config.model.task_templates = [

    {
        "taskname": "squad",
        "prompt_template": "<|VIRTUAL_PROMPT_0|> Context: {context}\n\nQuestion:
{question}\n\nAnswer:{answer}",
        "total_virtual_tokens": 15,
        "virtual_token_splits": [15],
        "truncate_field": "context",
        "answer_only_loss": True,
        "answer_field": "answer",
    },

]
```

NeMo Megatron

Setting The Pre-Trained GPT Model

Check what GPT .nemo models we have available on NGC

```
from nemo.collections.nlp.models.language_modeling.megatron_gpt_model import MegatronGPTModel
MegatronGPTModel.list_available_models()
```

Direct use of 345M parameter GPT model

```
gpt_model = MegatronGPTModel.from_pretrained(model_name="megatron_gpt_345m", trainer=trainer).cuda()
```

Download the model from NGC and set GPT model path on prompt learning config

```
gpt_file_name = "megatron_gpt_345m.nemo"
!wget  -nc --content-disposition
https://api.ngc.nvidia.com/v2/models/nvidia/nemo/megatron_gpt_345m/versions/1/files/megatron_gpt_345m.nemo -O {NEMO_DIR}/{gpt_file_name}
config.model.language_model_path = gpt_file_name
```

NeMo Megatron

Setting P-Tuning Specific Params

Set the model.virtual_prompt_style hyperparameter:

```
from nemo.collections.nlp.modules.common import VirtualPromptStyle
config.model.virtual_prompt_style = VirtualPromptStyle.P_TUNING
```

We can set the two p-tuning specific parameters:

- p_tuning.dropout
- p_tuning.num_layers

```
config.model.p_tuning.dropout = 0.0
config.model.p_tuning.num_layers = 2
config.model.global_batch_size = 2
config.model.micro_batch_size = 1
```

NeMo Megatron

Setting Prompt-Tuning Specific Params

- Prompt tuning specific parameters
  - **prompt_tuning.new_prompt_init_methods**
  - **prompt_tuning.new_prompt_init_text**

- Each of the above hyperparameters are a list of strings.
- `new_prompt_init_methods` would look like ["text", "random", "text", "text"]
- `new_prompt_init_text` might look like ["some text I want to use", None, "some other text", "task text goes here"] for those four new tasks

NeMo Megatron

Hands-On

- How to build a PyTorch Lightning Trainer
- Setting up a NeMo Experiment
- Sample P-Tuning and inferencing session

Details on these are covered in the Hands-On Jupyter Notebooks

Q & A

NVIDIA

THANK YOU