goto;

GOTO
AARHUS 2022

#GOTOaar

WHO AM I

Zenodia Charpy

Senior Deep Learning Data Scientist @ NVIDIA – NeMo| Megatron | bigNLP

My past experience:

Azure : senior data scientist & solution architect

+8 years experience as in-house data scientist & data science consultant

2GPT3, NLP AT BIG SCALE

Overview

The 2 GPT3s

The big 530B Megatron-Turing NLG GPT3 model

Motivation - Why Very Large Language Models

3 basic ingredients

Steps to follow - workflow

QnA – reach out and kick-start

The 2 GPT3

One ***English***, the other ***Swedish***

	English	Swedish
Data size	Terabytes of data	~40GB
Model size	530B	760M
Compute	***4480*** GPUs (560 nodes)	***8*** GPUs (1 nodes)
Time to train	~***20*** days	~***7*** days
Use cases	Mutiple downstream tasks with 0/1/fewshots	Limited to a handfull of cases

https://arxiv.org/pdf/2201.11990.pdf

	English	Swedish
Data size	Terabytes of data	~40GB
Model size	530B	760M
Compute	*4480* GPUs (560 nodes)	*16* GPUs (2 nodes)
Time to train	~*20* days	~*7* days
Use cases	Mutiple downstream tasks with 0/1/fewshots	Limited to a handfull of cases

530B MEGATRON TURING NLG - ENGLISH GPT3

VIRTUAL ASSISTANCE

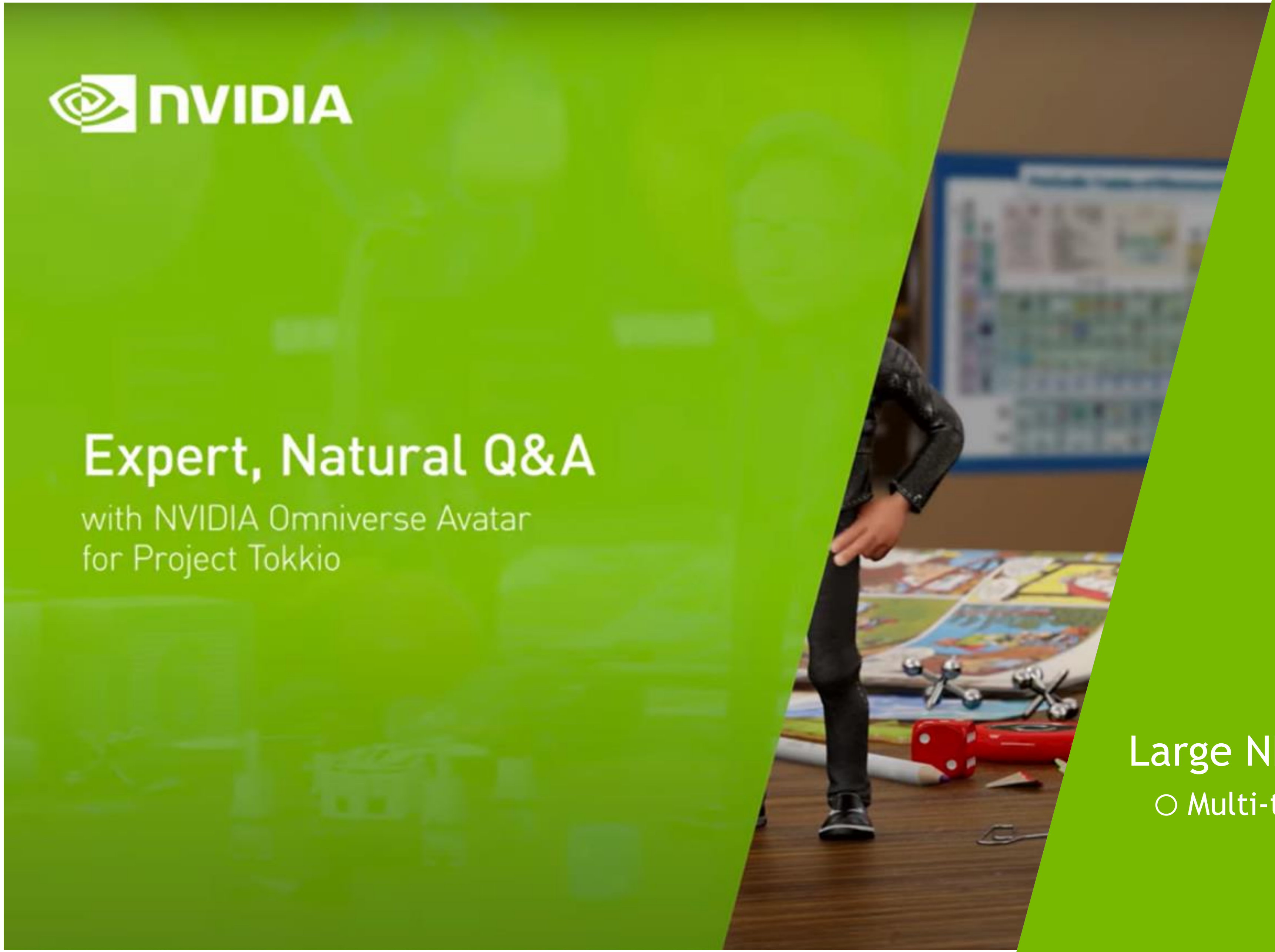

Shannon's

Large NLP models powers:

- Zero-shot intent and slot classification
- Context-based extractive Q&A
- Conversational phrasing of answer

TOY JENSEN – ASK ME ANYTHING


VENUS
NVIDIA



Large NLP models powers:

- Multi-turn Information Retrieval for Q&A

OTHER USE CASES

SOLVE RIDDLE

Context →

```
Here is a riddle:
All of us have one, but few get to choose
If you don't know mine, you are not my friend
When it is called, our attention is drawn
Even if we are gone, they are still around

I think the answer is
```

CODE GENERATION

Context →

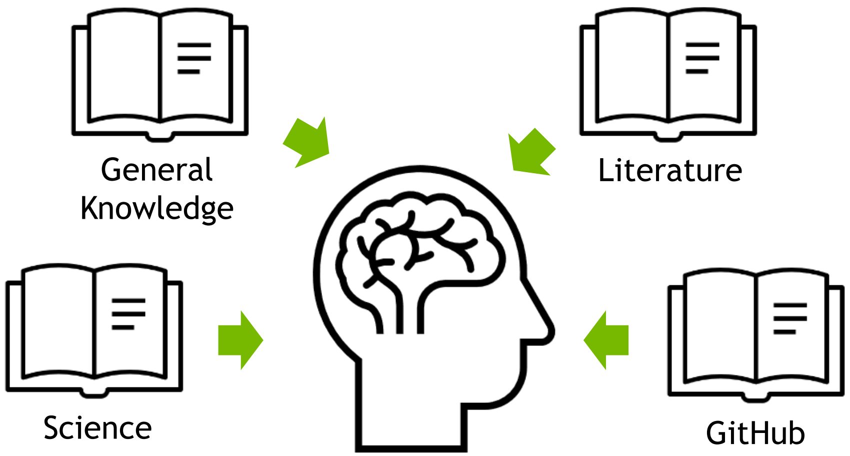
```
def find_3_or_7_divisible_fibs(n):
    """Find all the Fibonacci numbers below n that are divisible by 3 or divisible by
       7.
    """
```

WHY VERY LARGE LANGUAGE MODELS

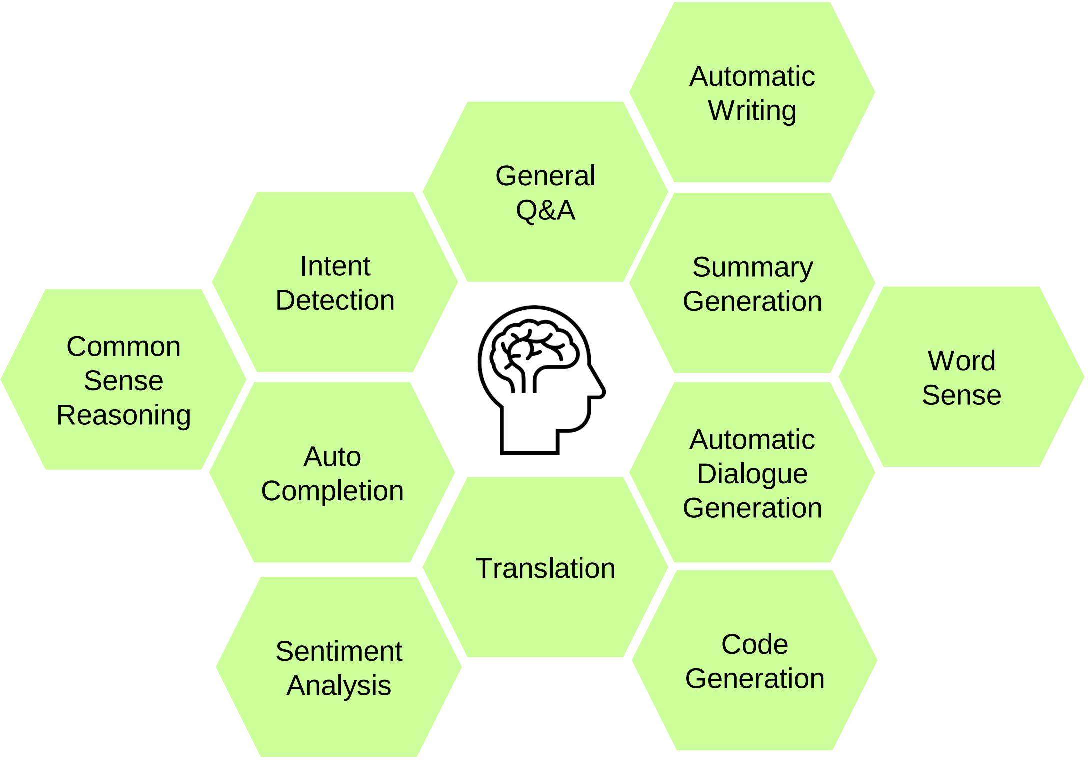
WHY ***PRE-TRAIN*** VERY LARGE NLP MODELS

Train Once, Perform different tasks

Step 1: Train a ***Very Deep/Huge*** model

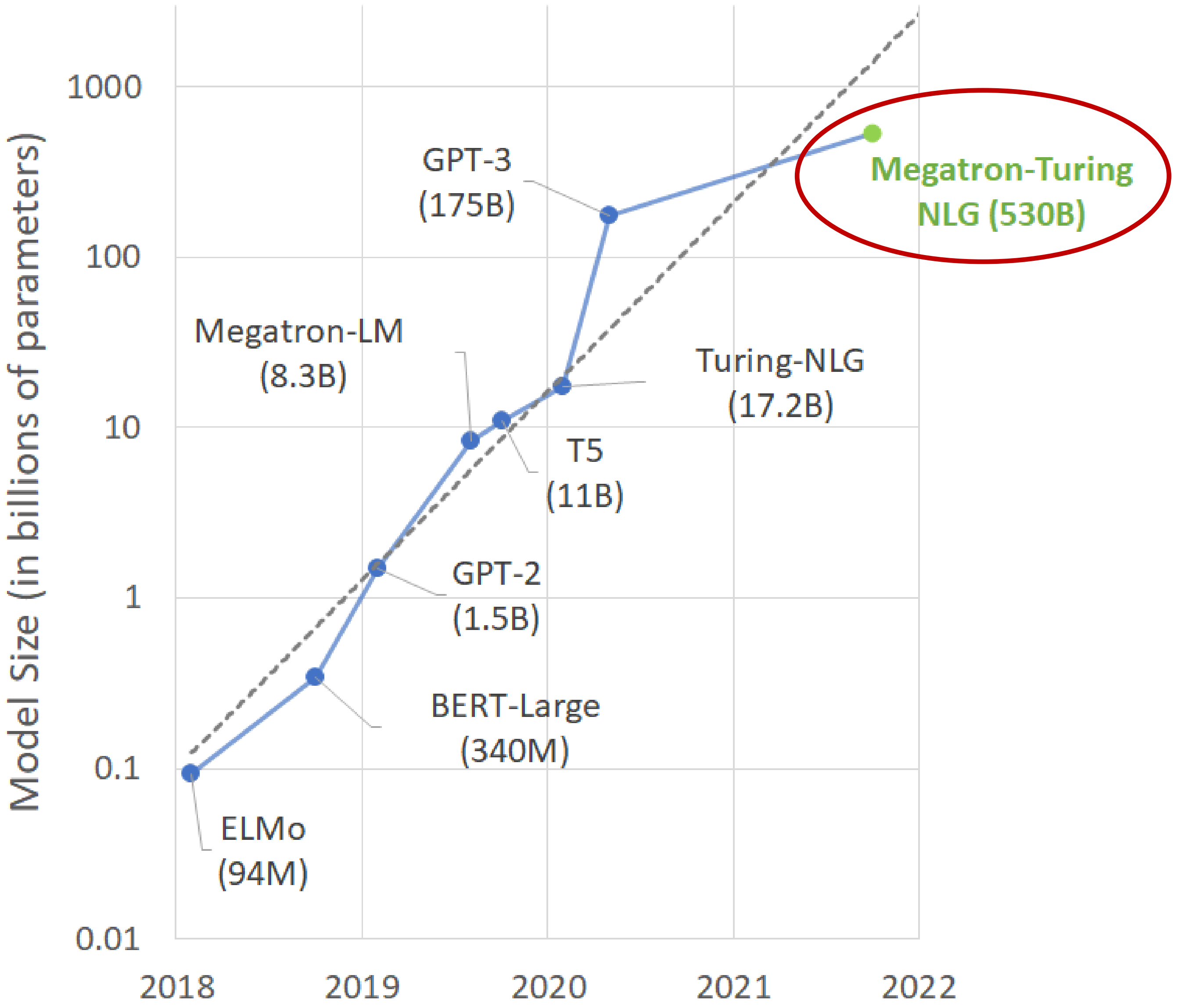


Step 2. Query different tasks with ***a few labelled*** data



Huge means Billions of parameters

MODEL SIZE GROWS EXPONENTIALLY



[Link to the article]

NVIDIA

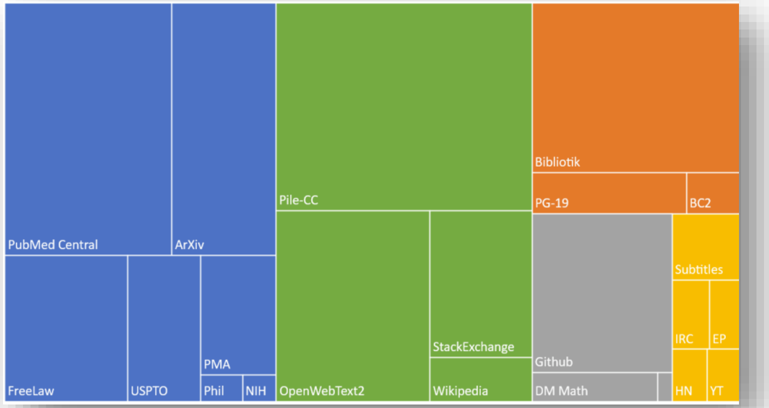
BUT THAT’S ENGLISH
- WHAT ABOUT LOCALIZED LANGUAGE ?

	English	Swedish
Data size	Terabytes of data	~40GB
Model size	530B	760M
Compute	*4480* GPUs (560 nodes)	*16* GPUs (2 nodes)
Time to train	~*20* days	~*7* days
Use cases	Mutiple downstream tasks with 0/1/fewshots	Limited to a handfull of cases

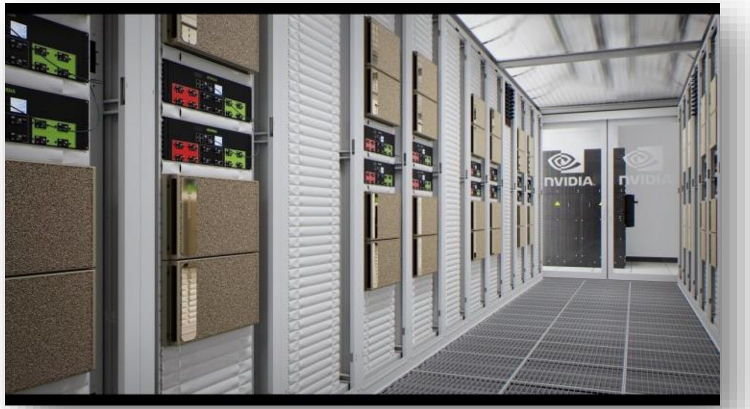
SECURE 3 BASIC INGREDIENTS

3 BASIC INGREDIENTS

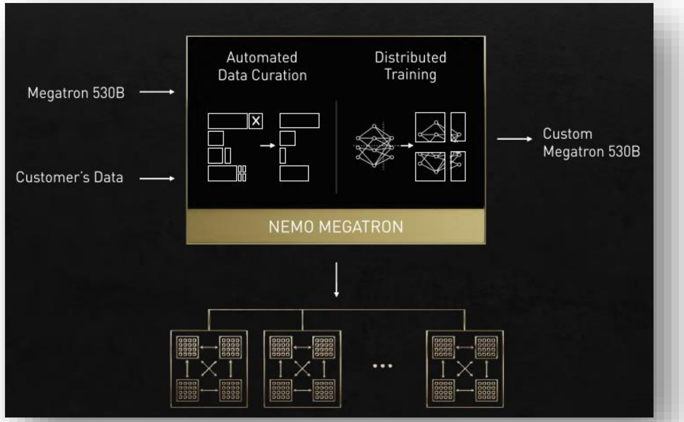
The Data



The Compute



The Framework



WHO HAS ALREADY BENEFITTED FROM THIS WORKFLOW ?

WHO HAS ALREADY USED THIS WORKFLOW ?

And trained + deployed their GPT3 models

Published in AI Sweden

Magnus Sahlgren

Jun 8 · 3 min read · Listen

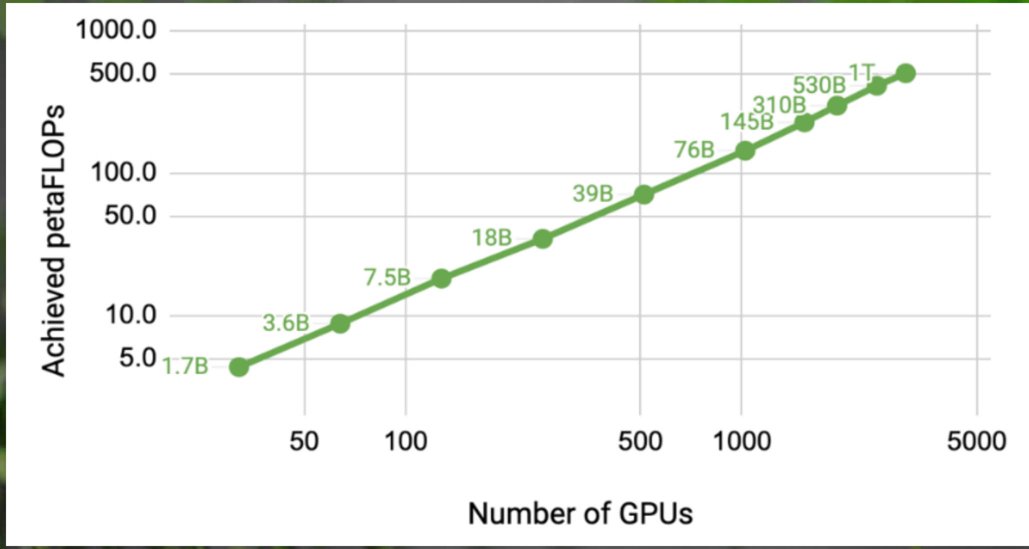
As the name suggest, **GPT-SW3** is the Swedish counterpart to GPT-3: a large-scale generative pretrained Transformer built on massive amounts of Swedish data with the explicit purpose of being able to generate Swedish text, and to thereby function in zero-shot and few-shot scenarios. The initiative to develop such a model for Swedish is led by the **NLU research group** at **AI Sweden**, and includes a number of collaborators, including **RISE**, the **WASP WARA on media and language**, and **Nvidia**. GPT-SW3 already exists in a preliminary version with 3.5 billion parameters, which was trained on a moderately sized corpus of approximately 100 GB of mostly web data. Our current goal is to train a 175 billion parameter model with close to 1 TB of data.

Link to the article

GPT-3 på svenska – de bygger en "superintelligens" som pratar svenska

2022-06-09 06:00 Av: Simon Campanello 0 kommentarer

Applicable to any language

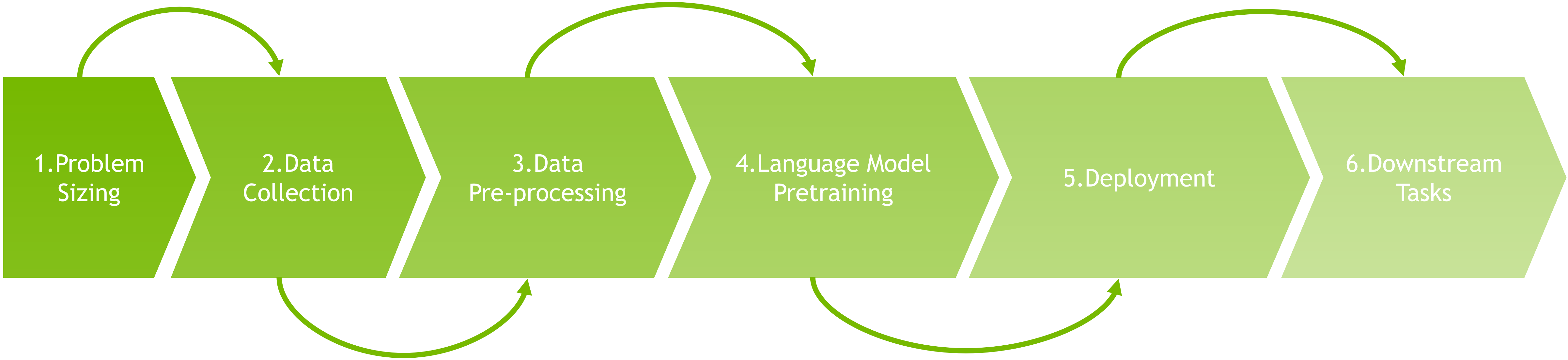


Scalable

KICK START YOUR JOURNEY - THE WORKFLOWS

THE WORKFLOW - START THE JOURNEY

Generic workflow – works for small to large to very large NLP models



1.Problem sizing

THE ITERATION TIME

Problem sizing – reducing years to weeks/days

GPT-3 Model Parameter Count	Estimated Time to Train			
	5 x SuperPod 100 nodes (days)	3 x SuperPod 60 nodes (days)	1 SuperPod 20 nodes (days)(weeks)	4 x DGX A100 (weeks)(years)
0.5B	0.21	0.21	0.62	0.44
18B	5	8	23	16
175B	43	72	31	3.0

Training

NVIDIA DGX A100 SuperPOD

Link to python function to estimate training time based on the equation from the paper below

Deepak Narayanan et al. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM.

2.Collect data – part 1/3
- Training data

DATA COLLECTION

Training data: Quantity vs. Quality

Common Crawl

OSCAR

Open **S**uper-large **C**rawled **A**ggregated co**r**pus

https://oscar-corpus.com/

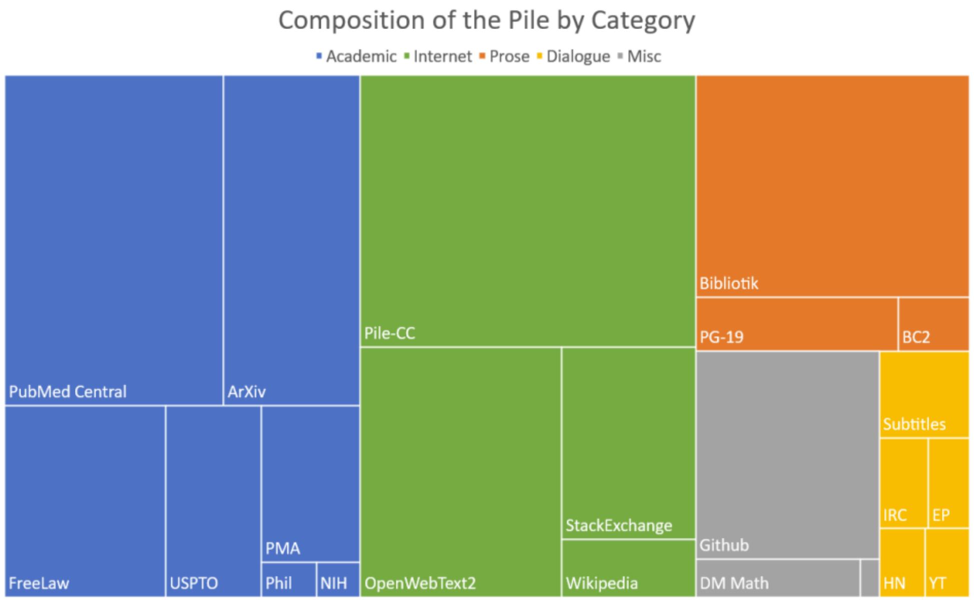


Figure 1: Treemap of Pile components by effective size.

The Pile

An 800GB Dataset of Diverse Text for Language Modeling

https://pile.eleuther.ai/

WEIGHING AGAINST QUALITY VS QUANTITY

Weighing mechanism is specific to the datasets used for training

Dataset	Tokens (billions)	Weights (%)	Epochs
Books3	25.7	14.3	1.5
OpenWebText2	14.8	19.3	3.6
Stack Exchange	11.6	5.7	1.4
PubMed Abstracts	4.4	2.9	1.8
Wikipedia	4.2	4.8	3.2
Gutenberg (PG-19)	2.7	0.9	0.9
BookCorpus2	1.5	1.0	1.8
NIH ExPorter	0.3	0.2	1.8
Pile-CC	49.8	9.4	0.5
ArXiv	20.8	1.4	0.2
GitHub	24.3	1.6	0.2
CC-2020-50	68.7	13.0	0.5
CC-2021-04	82.6	15.7	0.5
RealNews	21.9	9.0	1.1
CC-Stories	5.3	0.9	0.5

https://developer.nvidia.com/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/

2.Collect data – part 2/3
- Data for downstream tasks

DATA COLLECTION

Annotating and creating data for downstream tasks

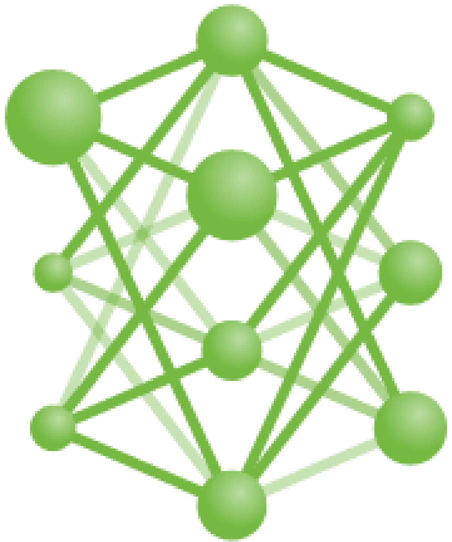
Train the GPT model unsupervised

WIKIPEDIA
L'enciclopedia libera

PubMed

Lemur

Common Crawl



Evaluate

labeled data

NVIDIA

DATA + ***LABEL*** COLLECTION

Annotation requires Tools and Expertise

UNIVERSITY OF GOTHENBURG

SPRÅKBANKENTEXT

Svenska

NEWS AND EVENTS | BLOG | RESEARCH | TOOLS | RESOURCES | FAQ | ABOUT US | CONTACT US

Home / Tools / Sparv / Annotations by Sparv

Sparv
- Sparv Pipeline
- Sparv's user manual
- **Annotations by Sparv**
- Web service (API)
- Web Sparv

Annotations by Sparv

This page aims at giving an overview of the analyses available within the Sparv Pipeline and the Sparv plugins developed by Språkbanken.

annotations are the names of the annotations as they are used in the corpus config file in the `export.annotations` section (read more about this in the Sparv user manual). Please observe that the annotations usually have shorter names in the corpus exports.

annotators are the names of the annotation functions (including their module names) which are used for procuding the annotations. They can be run directly with the `sparv run-rule [annoterare]` command. Though in most cases this is not necessary due to the fact that the annotation functions needed to produce all annotations listed in the corpus config file are executed automatically when running `sparv run`.

Analyses for contemporary Swedish

The following analyses are available in Sparv for contemporary Swedish. For various reasons not all of these are used by us at Språkbanken for analysing the corpora in Korp.

- Sentence segmentation with PunktSentenceTokenizer
  - **description**: Texts are split into sentences.
  - **model**: punkt-nltk-svenska.pickle
  - **method**: The model is build with NLTK's PunktTrainer and trained on StorSUC. The segmentation is done with NLTK's PunktSentenceTokenizer.
  - **annotations**:
    - `segment.sentence`: sentence segments
  - **annotators**: `segment.sentence`

Stockholm-Umeå corpus 3.0

The Stockholm-Umeå Corpus (SUC) is a collection of Swedish texts from the 1990's, consisting of one million words in total. The corpus is balanced, meaning that it contains various text types and stylistic levels. The texts are annotated with part-of-speech tags, morphological analysis and lemma (all that can be considered *gold standard data*), as well as some structural and functional information.

Version 1.0 was developed in co-operation between Gunnel Källgren at Stockholm University and Eva Ejerhed at Umeå University and was made available in 1997 by the department of linguistics at Stockholm University. *Version 2.0* was made available in 2006 by Sofia Gustafson-Capková and Britt Hartmann at the department of linguistics at Stockholm University. It contains the same texts as SUC 1.0 but is extended with some annotation. Additionally, SUC 2.0 contains bonus materials. TigerSUC is SUC 2.0 converted to TIGER-XML by Martin Volk. StorSUC is additional SUC material of four million words. *Version 3.0* is available since 2012. It contains improved annotations, and unannotated texts with seven million words. (For the TigerXML-version, Suc2c, Suc2d, and the DTDs we still refer to version 2.0.)

Example of tools for Swedish: https://spraakbanken.gu.se/en/resources/suc3

2. Collect data – part 3/3
- Legal/ethical/responsibleAI concerns

DATA COLLECTION

Establish Censoring Criteria

1. Data source licence implications
2. Data characteristics and coverage
3. Sufficiency of data pre-processing pipeline
4. Addressing issues of dataset bias
5. Addressing issues of dataset fairness
6. Addressing to ethical concerns
7. Addressing to legal concernss
8. Blacklisting & toxicity removal
9. Composition and weighing
10. …

3.Data preprocessing – part 1/3
- Filter / clean / deduplicate

DATA PRE-PROCESSING

Filtering + Deduplicate + Sentence Boundary

root@a0ca60d991fa: /works| × | data_cleaning_demo.ipynb ●

Code | Python 3

Step 1 - detect and filter undesired langauge in the raw text corpus

```
[1]: import random
     from langdetect import detect
     f=open('./bland_langs.txt','r')
     lines=f.readlines()
     len(lines)
```

```
[1]: 1824
```

```
[ ]: rn=random.randint(0,len(lines)-1)
     print("sentence = ", lines[rn])
     lang=detect(lines[rn])
     print("detected language is = ", lang)
```

step 2 - deduplicate based on similarity threshold

sanity check, fetch previously labelled two documents with groundtruth (i.e duplicated or not)

check against threhold, set to 0.85 and determine these two documents are duplicated or not !

```
[ ]: import itertools
     from lsh import cache, minhash # https://github.com/mattilyra/lsh

     # a pure python shingling function that will be used in comparing
     # LSH to true Jaccard similarities
     def shingles(text, char_ngram=5):
         return set(text[head:head + char_ngram] for head in range(0, len(text) - char_ngram))
```

Demo: Data pre-processing toy example for the Swedish language

Languages filtering + Deduplication + Sentence boundary (filter short sentences/single sentence document)

Link to [data cleaning example notebook]

NVIDIA

3.Data preprocessing – part 2/3 – Train your own tokenizer

TRAIN OWN GPT TOKENIZER

Train a GPT tokenizer on your own raw corpus

root@a0ca60d991fa: /works|

```
root@a0ca60d991fa:/workspace/SVdata/gtc# python trainGPTTokenizer.py --infile ./sv10thousand.txt --bpe_path ./32k/ --vocab_size 32000
```

trainGPTTokenizer.py

```python
    parser.add_argument('--pretrained_gpt_dir', default=None, type=str, help='path to pretrained GPT vocab and merge files, default None')
    parser.add_argument('--vocab_size', default=None, type=int,
                        help='specify the vocab_size when training HF GPTBPE for own language usually 16k/32k/48k/64k')
    args = parser.parse_args()
    tokenizer = Tokenizer(models.BPE())
    if args.load_pretrained :
        if args.pretrained_gpt_dir is not None :
            print("loading gpt2bpe english vocab and merge \n")
            vocab_file=args.pretrained_gpt_dir+'/gpt2-vocab.json'
            merge_file=args.pretrained_gpt_dir+'/gpt2-merges.txt'
            tokenizer.model = models.BPE.from_file(vocab_file, merge_file)
        else:
            print("please provide path to the pretrained gpt vocab and merge file !")

    tokenizer.pre_tokenizer = pre_tokenizers.ByteLevel()
    tokenizer.decoder = ByteLevelDecoder()

    print("include minimal special token end of text ")
    special_tokens= ["<|endoftext|>"]

    # Set the training hyperparameters
    trainer = trainers.BpeTrainer(
        vocab_size=args.vocab_size,
        special_tokens=special_tokens,
        initial_alphabet=pre_tokenizers.ByteLevel.alphabet()
    )
    # Train it with either files or an iterator:
    tokenizer.train([args.infile], trainer=trainer)
    print("Trained vocab size: {}".format(tokenizer.get_vocab_size()))
    # You will see the generated files in the output.
    print("saving trained BPE model to : ", args.bpe_path)
    tokenizer.model.save(args.bpe_path)
    print("model saved ! \n\n\n")
    print("testing ...\n\n\n")
    test_txt="Har någon funderat på varför man inte får inomhusteperaturens kurva synlig i grafen? Är det någon som frågat Thermia? Skulle det inte vara väsentligt att kunna kolla historiken på den då man skall ställa in
```

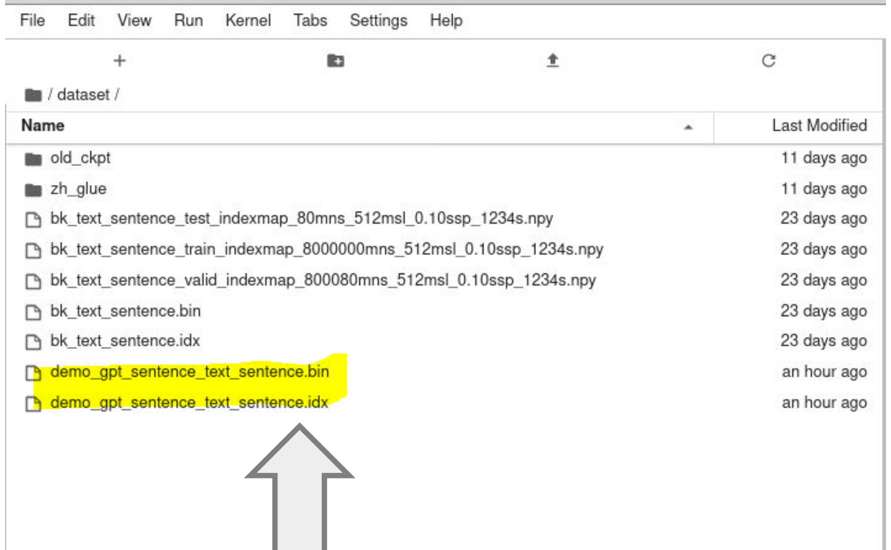
Demo: Train a GPTBPE Tokenizer on a toy Swedish data

Link to train GPTBPE Tokenizer example notebook

3.Data preprocessing - 3/3
- Optimal data loading

PREPROCESS TO BINARY FILES

Improve data loading performance during training time



mmap :
process data into xxx.idx + xxx.bin for optimized memory mapping to improve performance

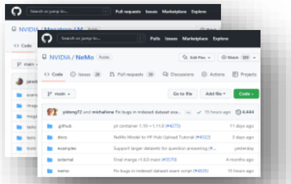
4. Training considerations

1. Problem Sizing | 2. Data Collection | 3. Data Pre-processing | **4. Language Model Pretraining** | 5. Deployment | 6. Downstream Tasks

CONSIDERATION FOR TRAINING LAUNCHES

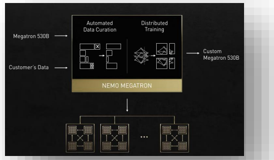
Manual vs automatic : obtain optimal training/inference configurations

DIY(Megatron-LM)



1. Manual runs to obtain optimal training config
2. Manually develop inference route for pretrained models
3. ***Manually*** debug with limited support

Enterprise ready(NeMo Megatron)



1. Get ***automatic*** recommandation of optimal training config from the HP tool
2. Support by default training and Inference per model size in the template
3. Get ***enteprise-grade expertise support***

Control

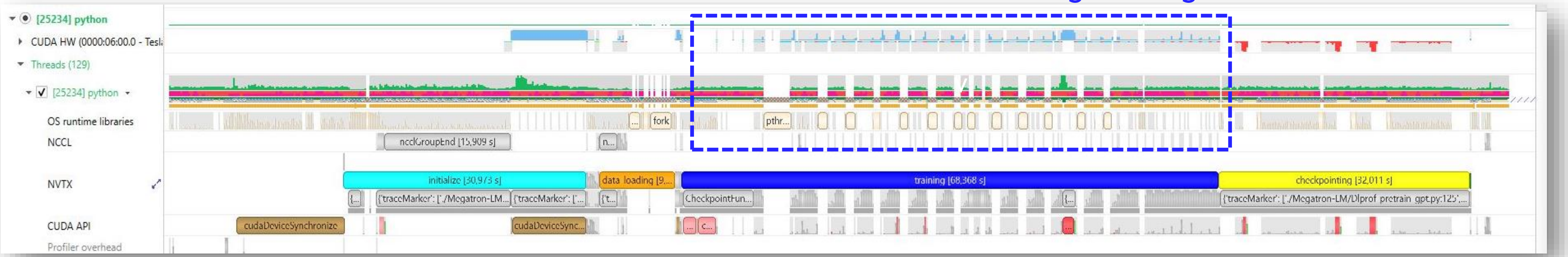
Manual work

DIY– MANUALLY TUNE TO ENSURE PERF VIA PROFILING

Profile training runs | improve iteratively

Naive config

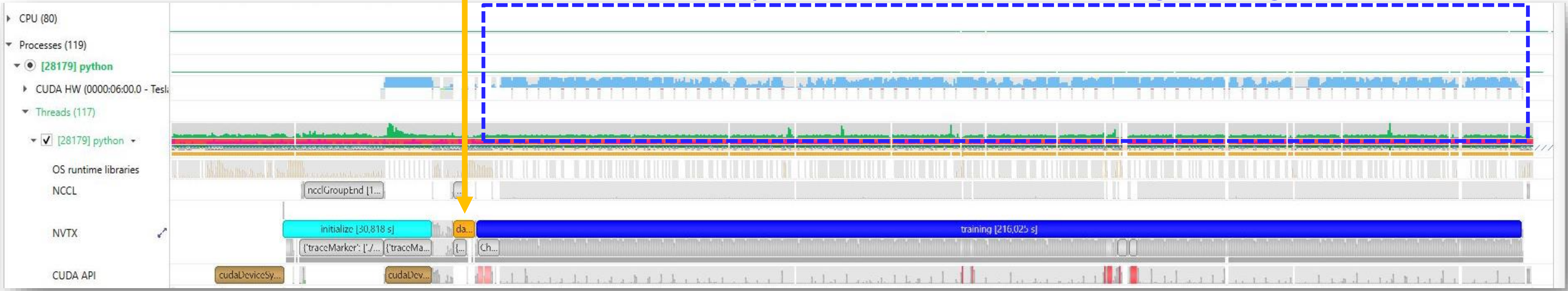
Low GPU utilization during training



Reduce data loading time

Improved config

Improved GPUs utilization during training



Learn more about NVIDIA Nsight Systems

NVIDIA

DIY - OBTAIN GOOD CONFIG FOR MODEL PRETRAINING

Training configuration performance obtained via manual experiments

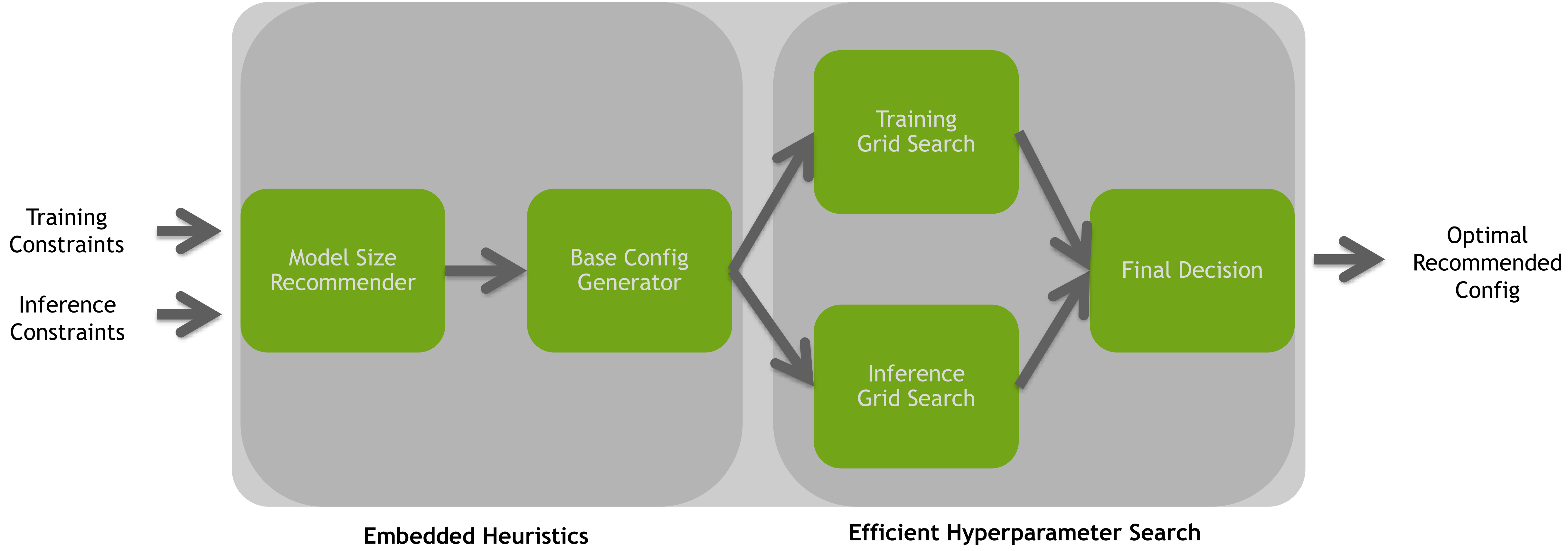
Find good training config

Teraflop util %

based on 100		Activation	**4**						312	A100 FP16 TFLOPS
		No Activation	**3**						112	V100 PCIe FP16 TFLOPS
elapsed time per									125	V100 SXM
Speed ms	N Gpus	Number of Nodes	**Model**	Micro	DP Size	TP Size	PP Size	PP	NVIDIA	Lower bound utilisation
31579	8	2	**18**	4	1	4	4	128	156.514	50.16%
44438	8	2	**18**	6	1	4	4	128	166.835	53.47%
31532	8	2	**18**	4	1	4	8	128	156.747	50.24%
44688	8	2	**18**	6	1	4	8	128	165.902	53.17%
62078	8	2	**18**	4	1	4	8	256	159.236	51.04%
15889	8	2	**18**	8	1	4	8	32	155.533	49.85%
16355	8	2	**18**	4	1	4	8	64	151.102	48.43%
31191	8	4	**39**	4	1	4	8	128	167.118	53.56%
45146	8	4	**39**	6	1	4	8	128	173.191	55.51%
87563	8	4	**39**	6	1	4	8	256	178.588	57.24%
173505	8	4	**39**	6	1	4	8	512	180.257	57.77%
43119	8	8	**52**	4	1	4	16	256	160.871	51.56%
43143	8	8	**52**	4	1	4	16	256	160.782	51.53%
43120	8	8	**52**	4	1	4	16	256	160.868	51.56%
43154	8	8	**52**	4	1	4	16	256	160.741	51.52%
62427	8	8	**52**	6	1	4	16	256	166.673	53.42%
80920	8	8	**52**	8	1	4	16	256	171.444	54.95%
121531	8	8	**52**	6	1	4	16	512	171.231	54.88%
158302	8	8	**52**	8	1	4	16	512	175.275	56.18%
43112	8	8	**52**	4	1	4	32	256	160.897	51.57%

ENTERPRISE READY - PRETRAINING WITH RECOMMENDED CONFIG

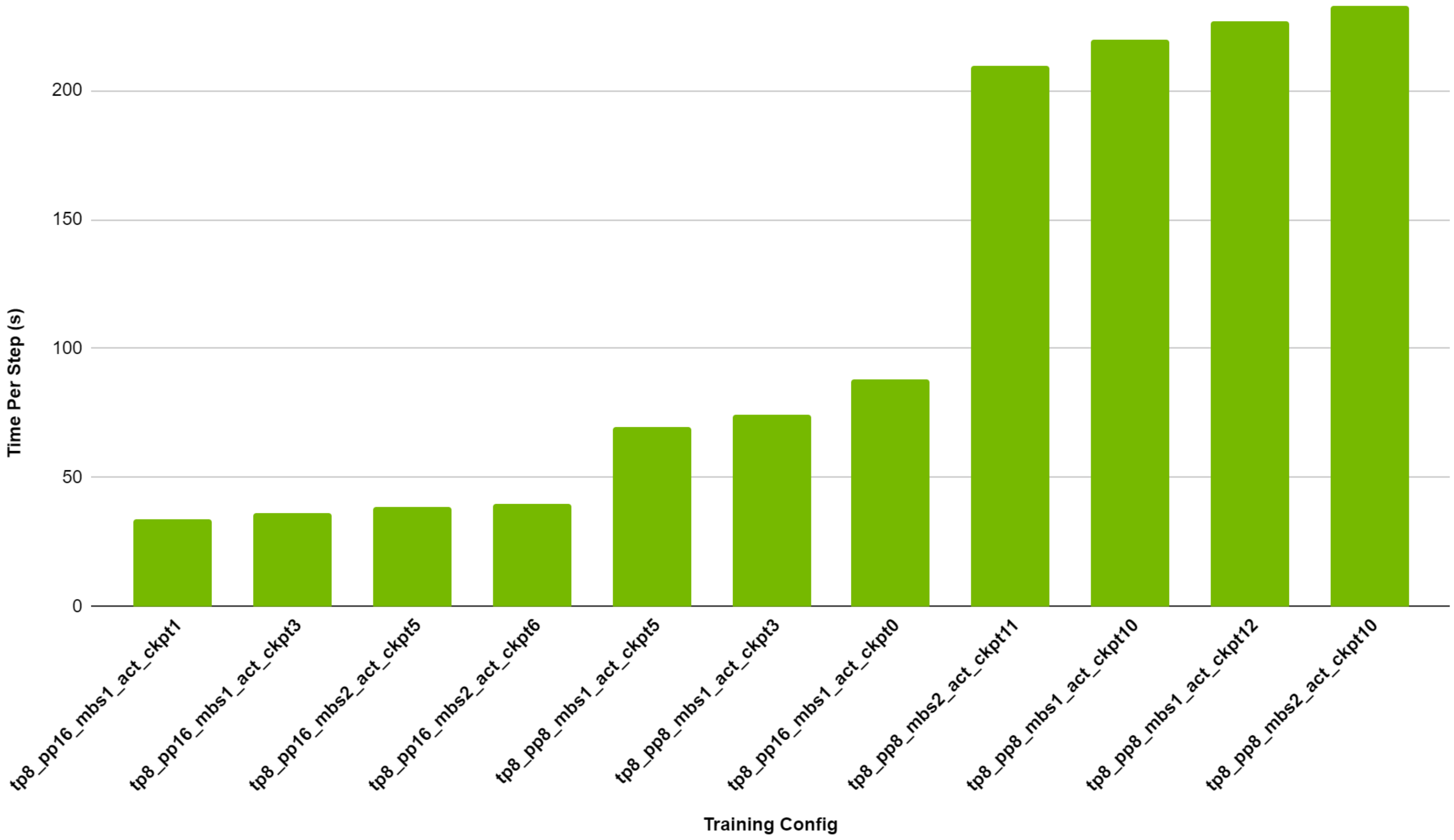
NVIDIA **NeMo Megatron** makes training easy with recommended config & perf guarantee



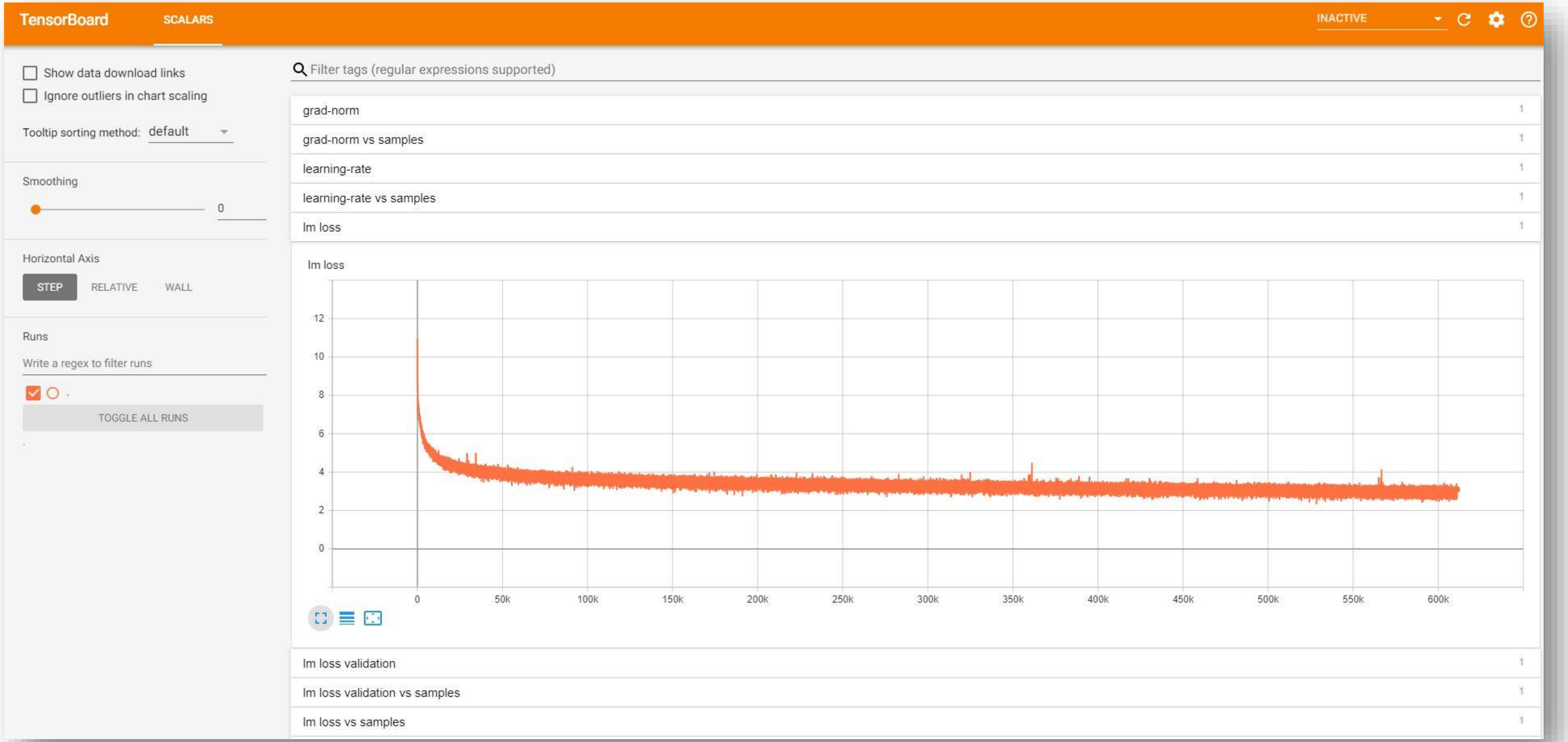
1.Problem Sizing | 2.Data Collection | 3.Data Pre-processing | **4.Language Model Pretraining** | 5.Deployment | 6.Downstream Tasks

ENTERPRISE READY - PRETRAINING WITH RECOMMENDED CONFIG

NVIDIA **NeMo Megatron** makes training easy with ready-made template config & perf guarantee



MONITOR TENSORBOARD



NVIDIA

5. Deploy & Serve
- Text generation demo

localhost:2345/lab?

File Edit View Run Kernel Tabs Settings Help

root@3c633147b13b: /works | Step3_ViewGenerated_text.i | Step2_760MGPTGenerateTe

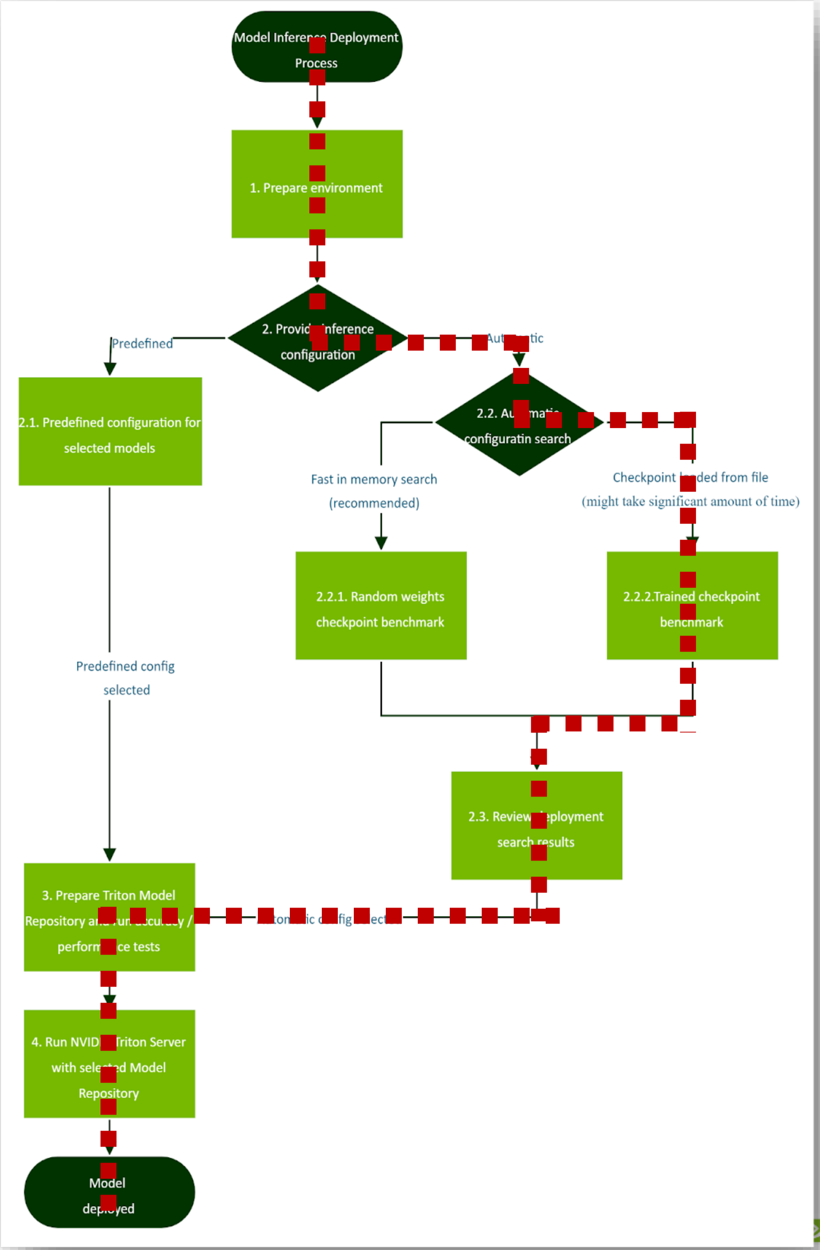
/ SVdata / gpt2bpe /

Name
- 32k
- 48k
- 56k
- gpt2-merges.txt
- gpt2-vocab.json
- SVGPT_32k_text_d...
- SVGPT_32k_text_d...

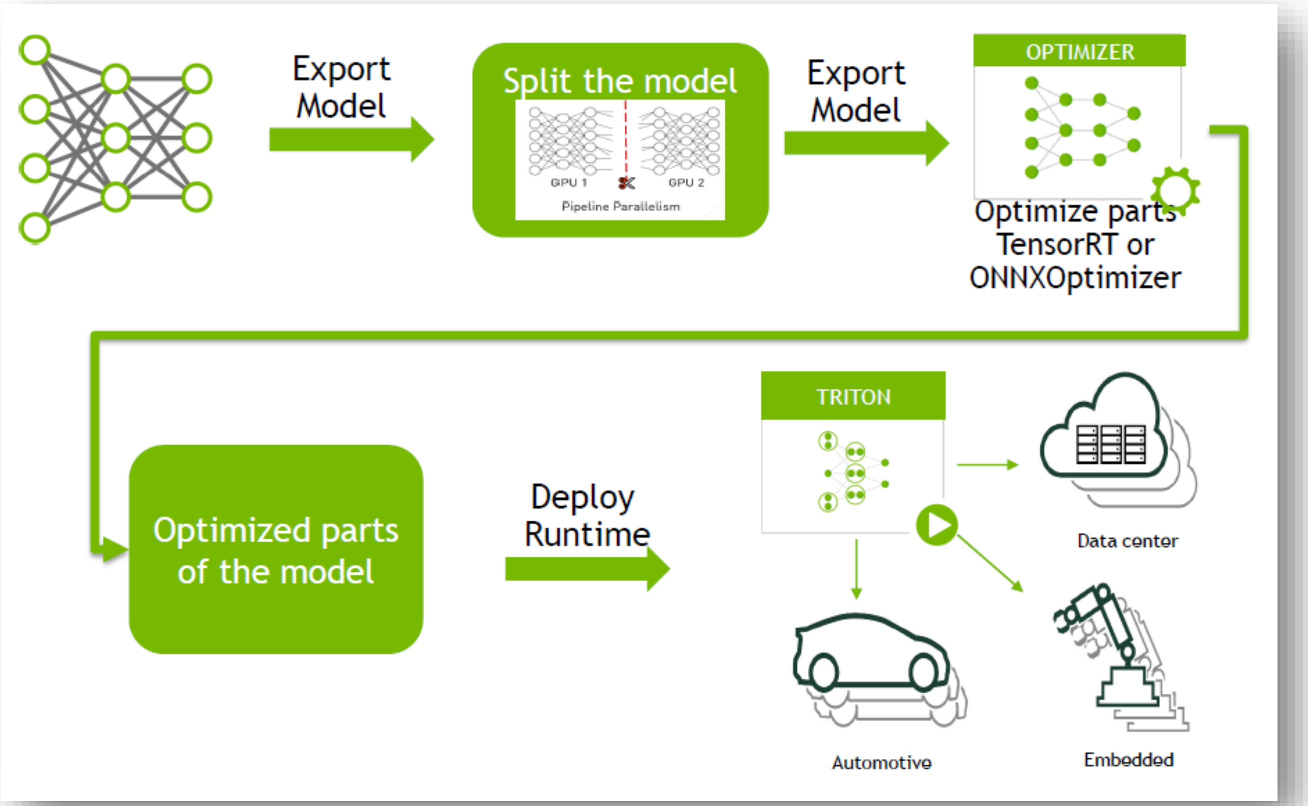
```
  use_contiguous_buffers_in_ddp .................. False
  use_cpu_initialization ......................... None
  use_one_sent_docs .............................. False
  virtual_pipeline_model_parallel_size ........... None
  vocab_extra_ids ................................ 0
  vocab_file ..................................... /workspace/SVdata/gpt2bpe/32k/vocab.json
  weight_decay ................................... 0.01
  world_size ..................................... 1
-------------------- end of arguments ---------------------
setting number of micro-batches to constant 1
> building GPT2BPETokenizer tokenizer ...
 > padded vocab (size: 32000) with 0 dummy tokens (new size: 32000)
> initializing torch distributed ...
> initializing tensor model parallel with size 1
> initializing pipeline model parallel with size 1
> setting random seeds to 1 ...
> initializing model parallel cuda seeds on global rank 0, model parallel rank 0, and data parallel rank 0 with model parallel seed: 2719 and data parallel seed: 1
> compiling dataset index builder ...
make: Entering directory '/workspace/megatron/data'
make: Nothing to be done for 'default'.
make: Leaving directory '/workspace/megatron/data'
>>> done with dataset index builder. Compilation time: 0.038 seconds
WARNING: constraints for invoking optimized fused softmax kernel are not met. We default back to unfused kernel invocations.
> compiling and loading fused kernels ...
Detected CUDA files, patching ldflags
Emitting ninja build file /workspace/megatron/fused_kernels/build/build.ninja...
Building extension module fused_mix_prec_layer_norm_cuda...
Allowing ninja to set a default number of workers... (overridable by setting the environment variable MAX_JOBS=N)
ninja: no work to do.
Loading extension module fused_mix_prec_layer_norm_cuda...
>>> done with compiling and loading fused kernels. Compilation time: 1.028 seconds
building GPT model ...
 > number of parameters on (tensor, pipeline) model parallel rank (0, 0): 729897984
 loading checkpoint from /workspace/tools/gpt3_iter2mil/ at iteration 2000000
 checkpoint version 3.0
  successfully loaded checkpoint from /workspace/tools/gpt3_iter2mil/ at iteration 2000000
Avg s/batch: 0.9077534675598145
root@3c633147b13b:/workspace#
```

5.Deploy & Serve

Enterprise ready(NeMo Megatron) - support by default *__automtaic__* deployment route



DIY(Megatron-LM) – *__Manually__* develop deployment route



1.Problem Sizing → 2.Data Collection → 3.Data Pre-processing → 4.Language Model Pretraining → 5.Deployment → 6.Downstream Tasks

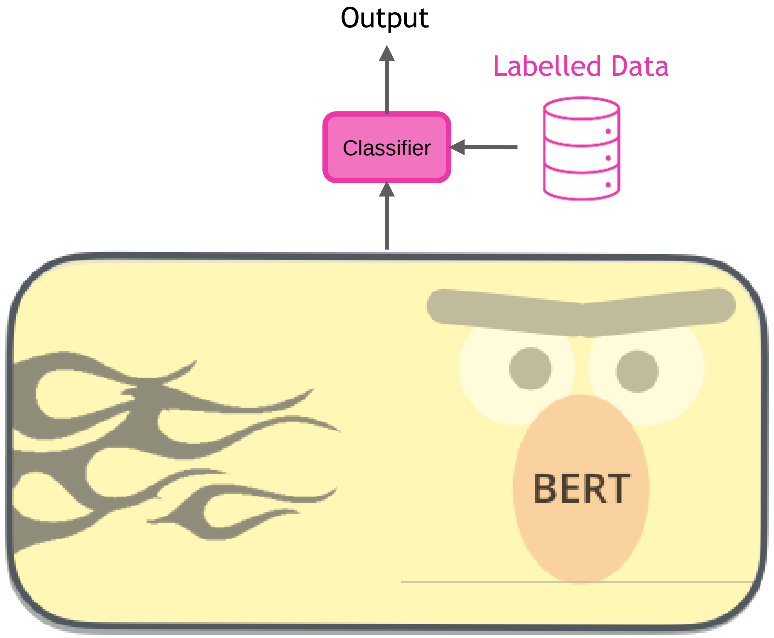
P-tuning for downstream tasks

NLP APPROACH (CIRCA 2019)

Fine tune per specific task

Step 1: Pre-training a Encoder

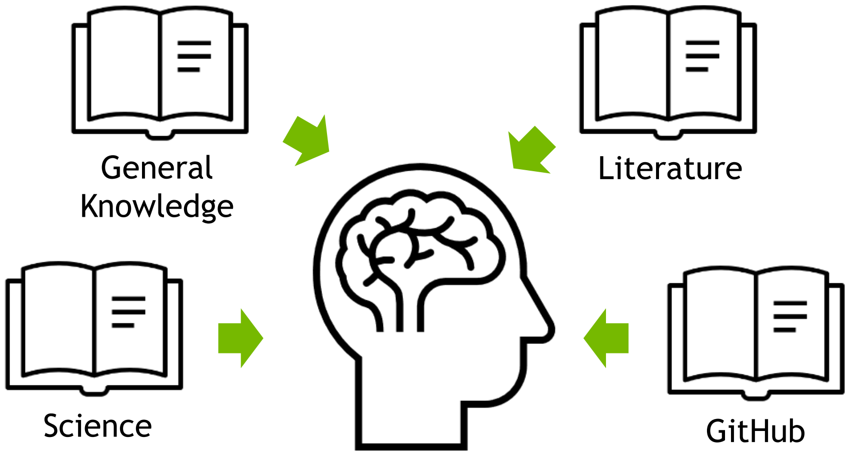
Step 2. Fine tune for a specific task



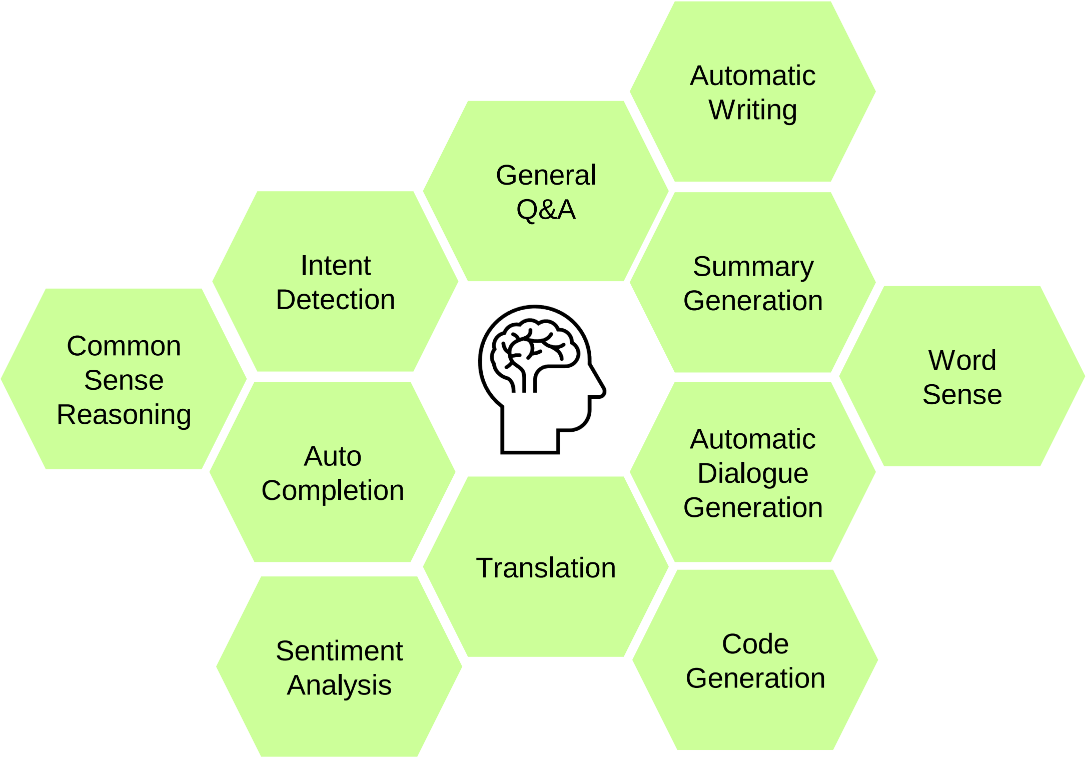
NEW NLP APPROACH (CIRCA 2021)

Train Once, Perform different tasks

Step 1: Train a ***Very Deep/Huge*** model



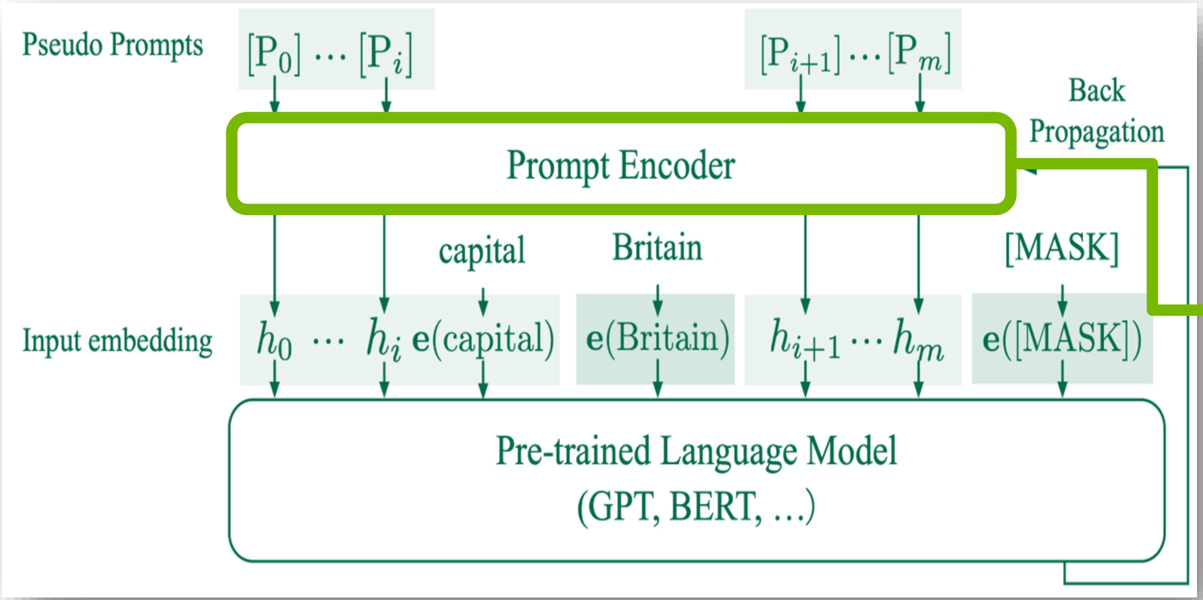
Step 2. Query different tasks with ***a few labelled*** data



Huge means Billions of parameters

P-TUNING METHOD

Vanilla P-tuning



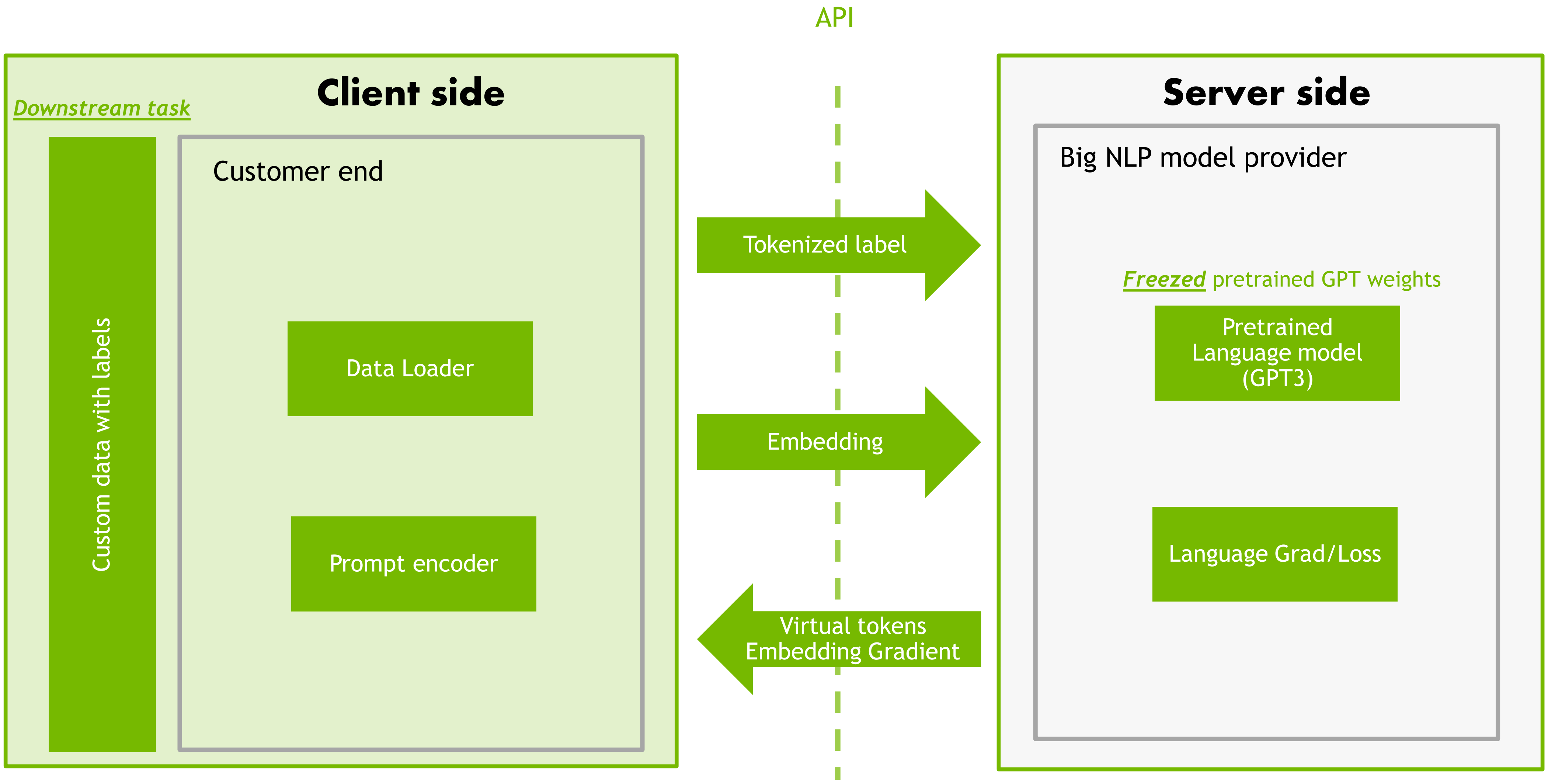
Liu, Zheng et al, GPT Understands, Too 2020

	Name	Type	Params
0	model	MegatronGPTModel	3.6 B
1	word_embeddings	VocabParallelEmbedding	154 M
2	prompt_table	PromptTable	0
3	prompt_encoder	PromptEncoder	132 M

132 M	Trainable params
3.6 B	Non-trainable params
3.7 B	Total params
7,383.245	Total estimated model params size (MB)

BRING YOUR OWN DATA VIA P-TUNING

Well-engineered architecture in NeMo (future release in NeMo Megatron)

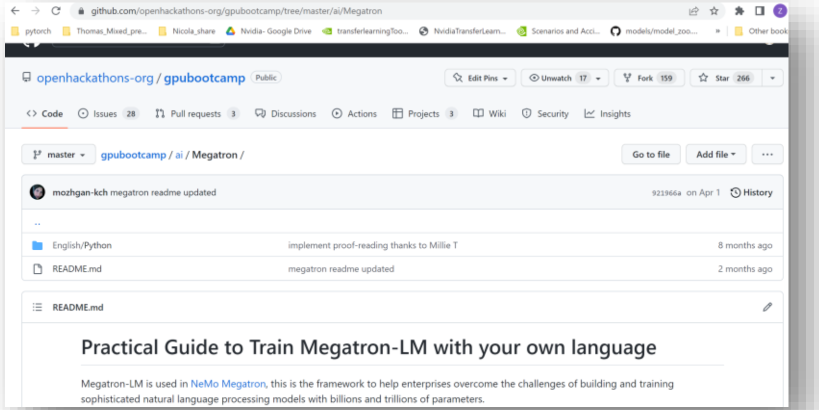


BUT HOW DO I TRY-OUT/KICK-START FOR REAL ?

MULTIPLE WAYS TO TRY-OUT/KICK-START

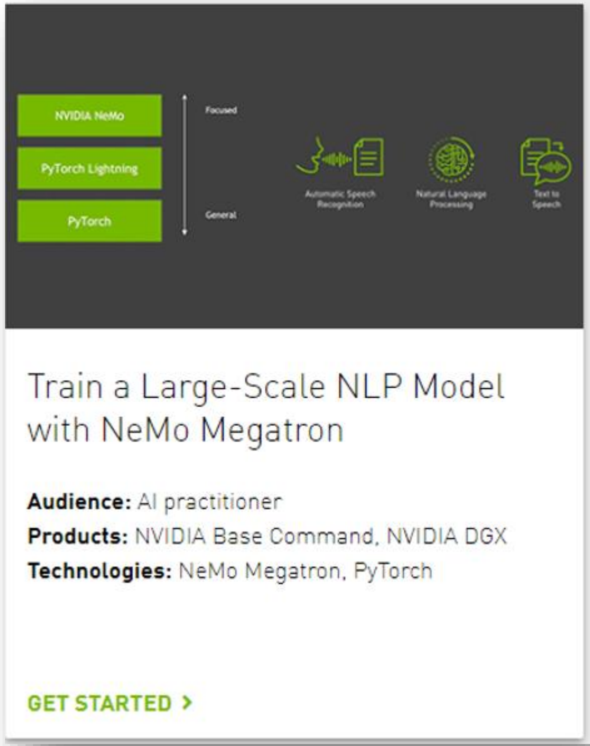
Contact NVIDIA

Request a bootcamp



https://github.com/openhackathons-org/gpubootcamp/tree/master/ai/Megatron

Register LaunchPad



https://www.nvidia.com/en-us/data-center/launchpad/

Join NeMo Megatron EA

NeMo Megatron Early Access

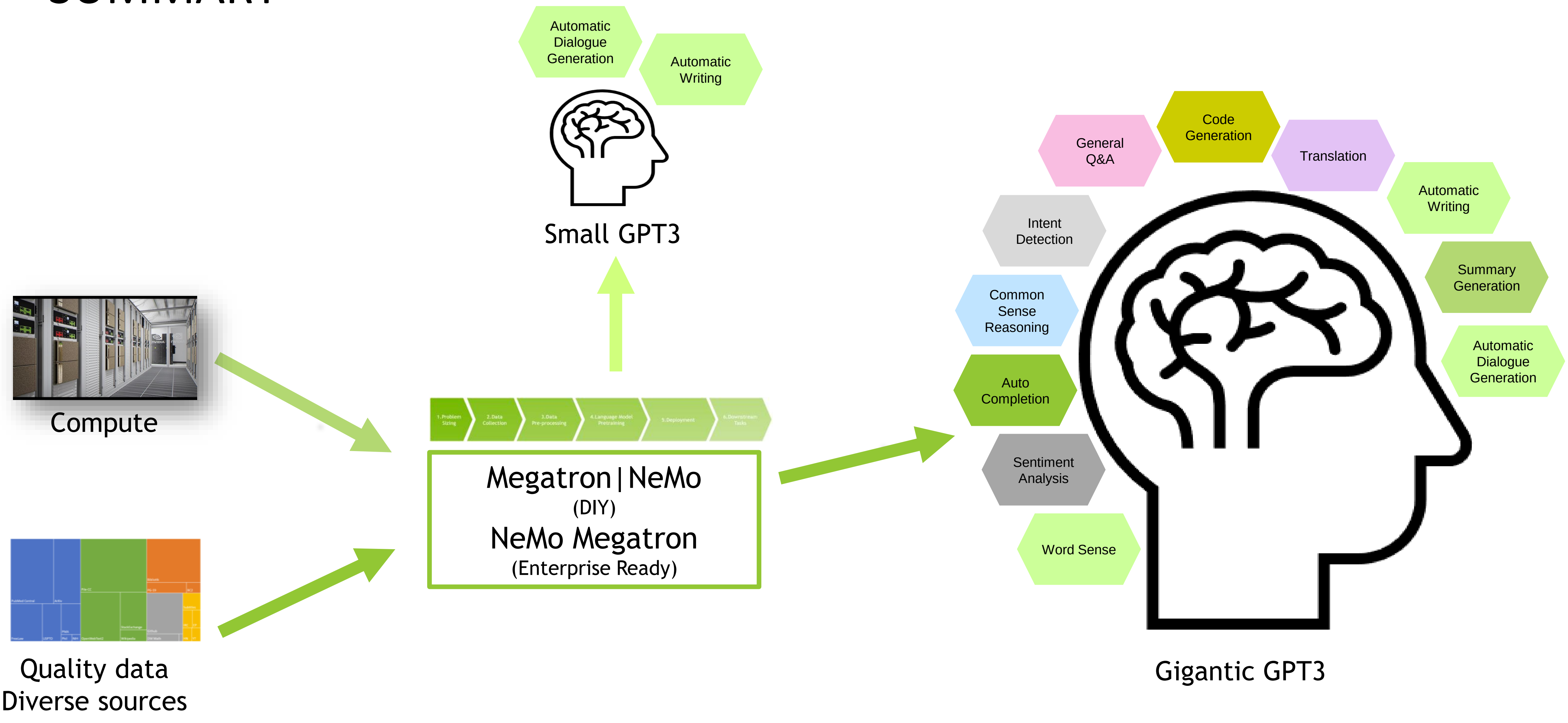
NeMo Megatron is a new capability in the NeMo framework that allows developers to effectively train and scale language models to billions of parameters. With NeMo Megatron, you can train different variants of GPT-3 models and scale them to multiple nodes on superpods. This deep learning software stack is optimized for NVIDIA DGX A100 SuperPODs using NVIDIA InfiniBand to provide efficient on-premises compute for training and inferring complex workloads.

Early access to NeMo Megatron is limited to enterprises that want to train and deploy GPT-3 style models on NVIDIA A100 SuperPOD to perform zero-shot tasks such as answering deep domain questions, translating languages, comprehending and summarizing complex documents, etc.

https://developer.nvidia.com/nemo-megatron-early-access

SUMMARY

SUMMARY


Automatic Dialogue Generation
Automatic Writing
Small GPT3
Compute
Quality data
Diverse sources
1.Problem Sizing
2.Data Collection
3.Data Pre-processing
4.Language Model Pretraining
5.Deployment
6.Downstream Tasks
Megatron | NeMo
(DIY)
NeMo Megatron
(Enterprise Ready)
General Q&A
Code Generation
Translation
Automatic Writing
Intent Detection
Summary Generation
Common Sense Reasoning
Automatic Dialogue Generation
Auto Completion
Sentiment Analysis
Word Sense
Gigantic GPT3

REACH OUT

Rasmus Bisgaard
rbisgaard@nvidia.com

Zenodia Charpy
zcharpy@nvidia.com

goto;

DON'T FORGET TO
RATE THE SESSIONS

#GOTOaar

Rate a minimum of **5 sessions** and claim your **reward** at the Registration Desk at the Trifork Hall