

Optimizing LLMs

Start-to-finish LLM training

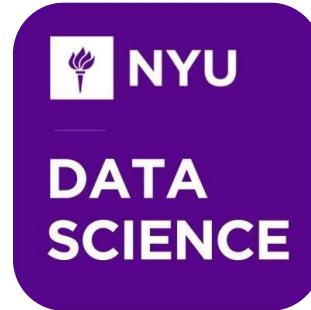


Shaan Khosla

Data Scientist, Researcher, Student

My name is **Shaan Khosla**

- Data Scientist at Nebula
- Various research with LLMs, topic models, and legal NLP
- Current Data Science masters student at NYU
- Writer of *Let's Talk Text* newsletter



Optimizing LLMs

Code:

github.com/shaankhosla/catalyst



Follow-up with me:

[linkedin.com/in/shaan-khosla](https://www.linkedin.com/in/shaan-khosla)



shaankhosla.substack.com for email newsletter



Massive thanks to Jon Krohn:

- [Slides](#)
- [Code](#)
- [Website](#)



Poll

What is your current level of experience with LLMs?

- Only used ChatGPT or OpenAI APIs
- Fine-tuned a smaller model, like a RoBERTA classification model, on a small dataset with one GPU
- Fine-tuned a large model, like LLaMA, with multiple GPUs and helped productionize it



Poll

What part of the talk are you most looking forward to?

- Hands-on coding walkthroughs
- Data augmentation techniques
- How to handle large datasets
- Fine-tuning optimizations
- Multi-GPU training tips
- More pictures of Mila



Schedule

1. Introduction to Large Language Models (35 mins)
2. Q&A (5 mins) & Break (5 mins)
3. Single GPU Training Techniques (60 mins)
4. Break (10 mins)
5. Multi-GPU Training Techniques (45 mins)
6. Summary and Q&A (20 mins)



Schedule

- 1. Introduction to Large Language Models (35 mins)**
2. Q&A (5 mins) & Break (5 mins)
3. Single GPU Training Techniques (60 mins)
4. Break (10 mins)
5. Multi-GPU Training Techniques (45 mins)
6. Summary and Q&A (20 mins)



How machines read

Word tokenization

- Large vocabulary size
- Unknown words lose meaning

Character tokenization

- Characters lack meaning
- Large number of tokens required represent text

Subword tokenization

- Byte-pair encoding
- Start with character level tokenization
- Recursively merge tokens together based on frequency of co-occurrence

Sample Data:

"This is tokenizing."

Character Level

[T] [h] [i] [s] [i] [s] [t] [o] [k] [e] [n] [i] [z] [i] [n] [g] [.]

Word Level

[This] [is] [tokenizing] [.]

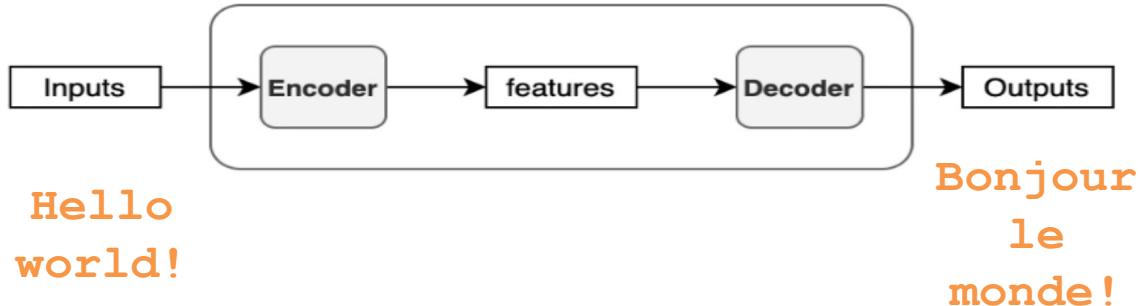
Subword Level

[This] [is] [token] [izing] [.]

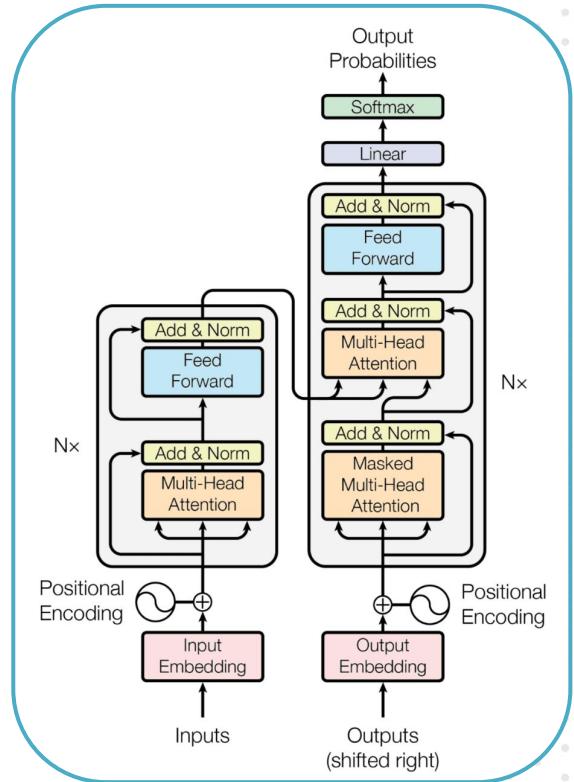
*Hands-on code demo:
Tokenization.ipynb*

Overview of pre-trained language models

- History
 - Bag of Words
 - TF-IDF
 - Topic models such as LDA
 - Word2Vec, GloVe
 - RNNs, LSTMs, GRUs
- Attention Is All You Need (Vaswani et al., 2017)



Hands-on code demo:
GPT4All-inference.ipynb



Source: [The Illustrated Transformer](#)

Autoregressive vs. autoencoding

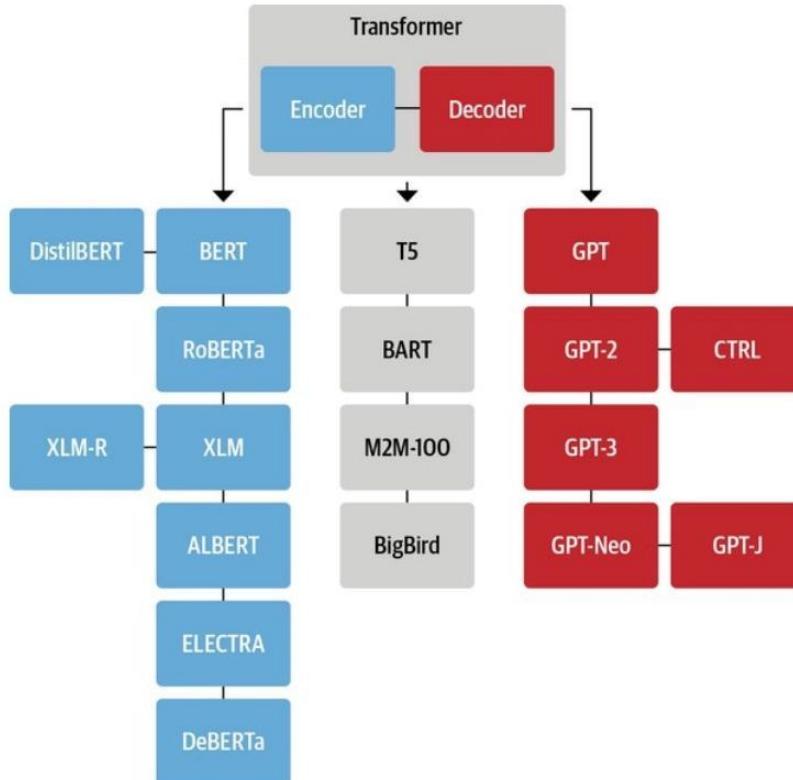
Autoregressive

- “**After practice she went to the ____**”
- Typically used in text-generation tasks
 - Summarization, question answering, translation
- Like doing a live freestyle rap

Autoencoding

- “**Steve loved his previous ___, he had a great team and ___ his work.**”
 - Sentiment analysis, classification, semantic search
- Reconstruct original data from corrupted input

Autoregressive vs. autoencoding pt.2



- Unsupervised training let's us train on large quantities of text data, the learning is arbitrarily defined
- The same architecture can be used for autoregressive and autoencoding models

Figure 3-8. An overview of some of the most prominent transformer architectures

Source: [Aman](#)

Ways to use LLMs

1. Using a pre-trained model without any fine-tuning
 - Using BERT to generate embeddings and creating a semantic search engine
2. Fine-tuning a model to accomplish a specific task
 - Fine-tuning GPT-2 to summarize letters to shareholder financial documents
3. Prompting:
 - Using GitHub co-pilot to generate code
 - ChatGPT



Key libraries

- 😊 **Transformers**
 - Many, many open-source LLMs ready to go
 - Task ready code that works easily with pre-trained models
 - Easy inference
 - Automatic optimizations
- **PyTorch Lightning**
 - Simplifies model training yet enables flexibility
 - Easy distributed training
 - Easy to restructure code into Lightning Module
 - Automatic optimizations
 - Works easily with HuggingFace models
- **DeepSpeed**
 - Deep learning optimization library from Microsoft
 - Enables training and inference of billions/trillions of parameters



Latest Open-Source LLMs

- [LLaMA](#): 65-billion parameter LLM from Meta
- [Alpaca](#): LLaMA 7B fine-tuned on 52k GPT-3.5
- [Vicuna](#): 70k ShareGPT convos
- [Dolly 2.0](#): commercial use allowed
- [Falcon 40B](#): ranked #1 on HuggingFace LLM leaderboard
- [LongNet](#): Transformer with 1,000,000,000 tokens

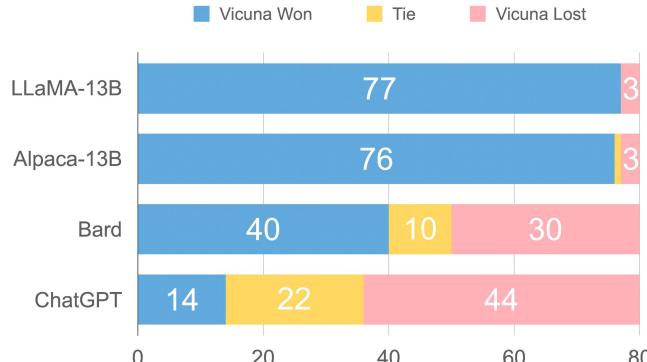
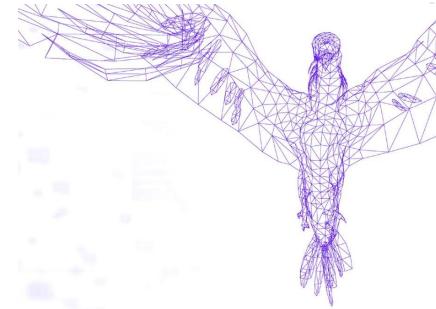


Figure 1: Trend of Transformer sequence lengths over time.

Data Augmentation Using LLMs

- Back translation with translation models



- Synonym replacement using [MASK]



- Text generation with generative models

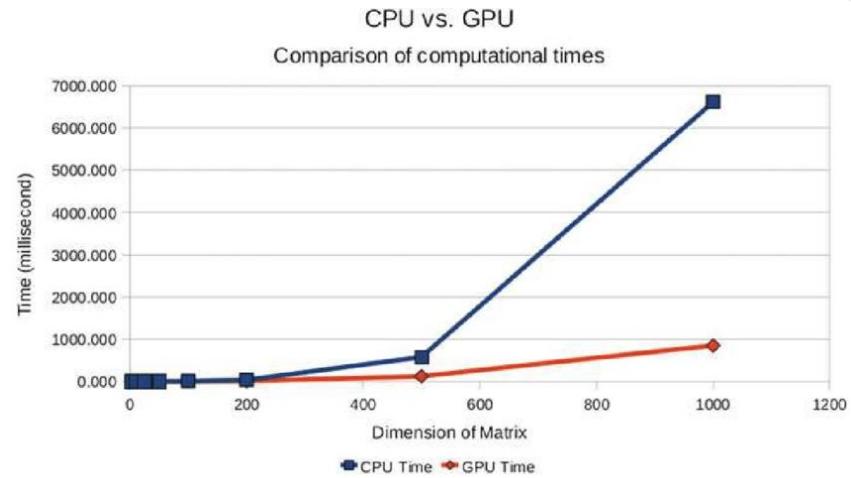
If only to avoid making this type of film in the future. This film is interesting as an experiment but tells no cogent **plot or any other interesting story**.

Overall, this is an entertaining film but is in most cases not good.

Hardware

Matrix multiplication chips

- CPUs
 - Useful for inference
- GPUs
 - Used for training + inference
- TPUs
 - Google Tensor Processing Unit
- AWS
 - Trainium
 - Inferentia



Source: [Paper](#)

Schedule

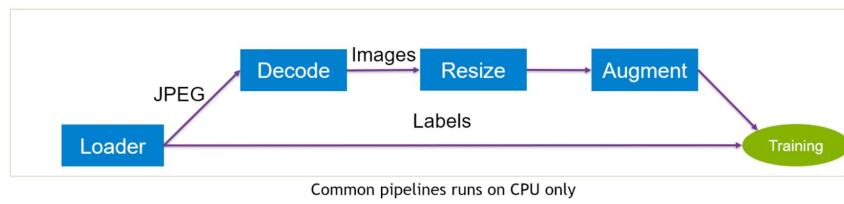
1. Introduction to Large Language Models (35 mins)
2. Q&A (5 mins) & Break (5 mins)
- 3. Single GPU Training Techniques (60 mins)**
4. Break (10 mins)
5. Multi-GPU Training Techniques (45 mins)
6. Summary and Q&A (20 mins)



Managing large datasets

- GPU memory > RAM > Disk
- Can't cache entire dataset into memory
- Data loading from disk is often the bottleneck
- PyTorch Datasets & DataLoader
 - Return a single datapoint on request
 - Handles:
 - Parallel loading using multiple workers
 - Intelligently prefetching data
 - Easily add data transformations, shuffling, batching
 - Decouples data code from modeling code

DATA PIPELINE - BEFORE & AFTER

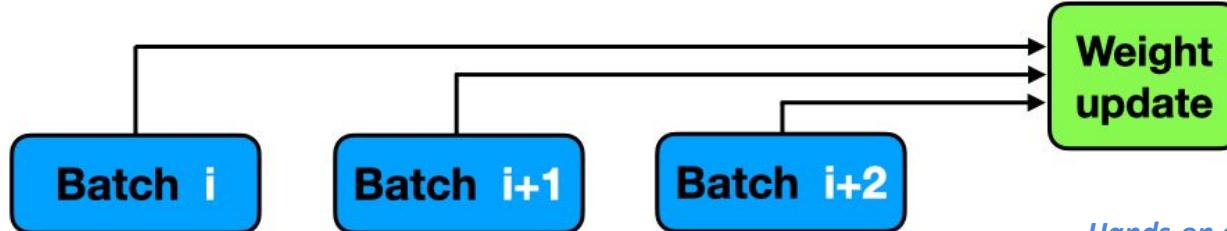


Hands-on code demo:
Streaming_Datasets.ipynb

Source: [Arctic Wiki](#)

Gradient Accumulation

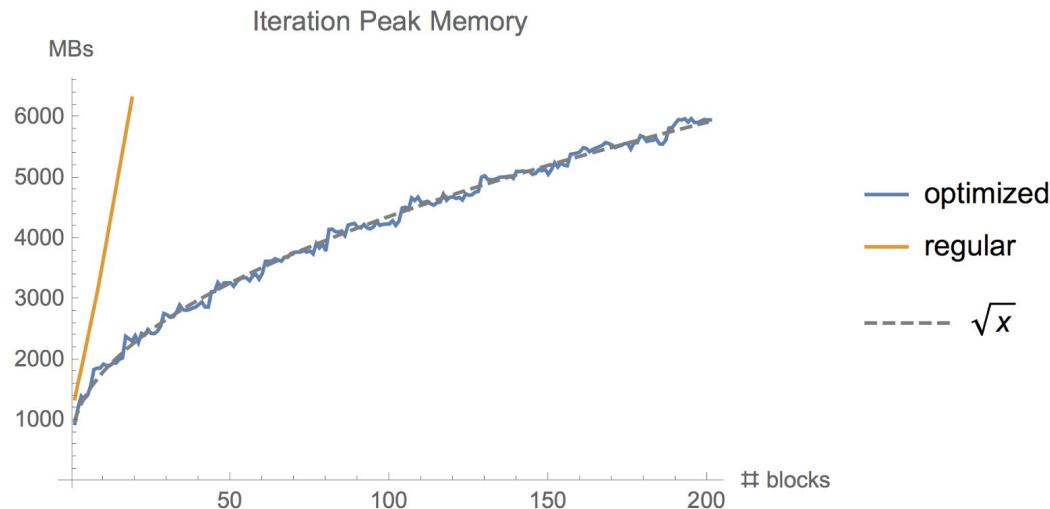
- Batch size is limited by GPU memory
- Maximize GPU usage
 - Split batch into N microbatches
 - Forward pass each microbatch and accumulate gradients over N batches
 - Perform backprop with accumulated gradients



Hands-on code demo:
Single GPU Optimizations.ipynb

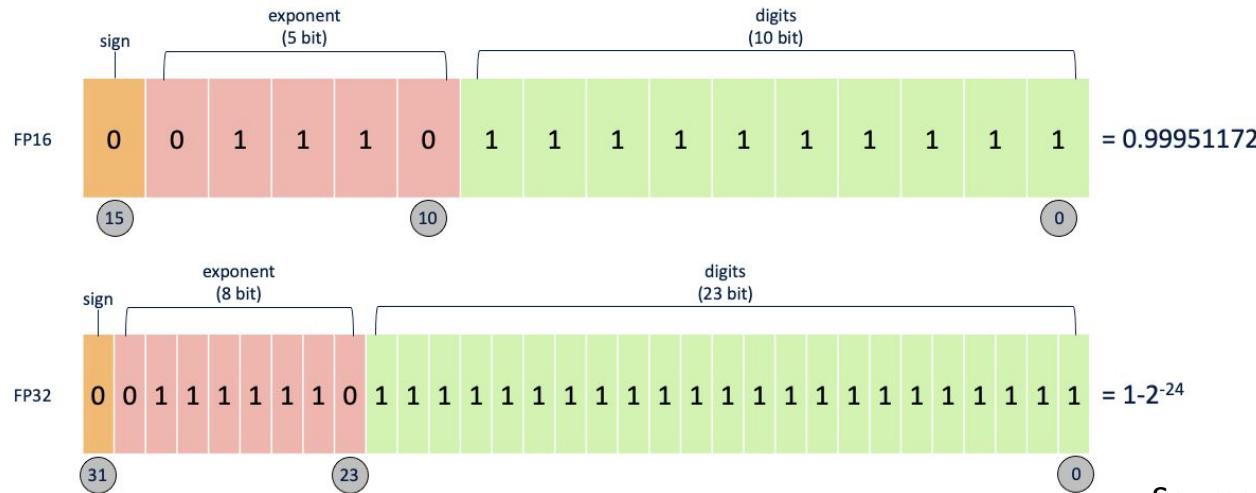
Gradient Checkpointing

- Normally all the activations from the forward pass are saved to compute the gradients in the backward pass
 - Compute efficient, but memory inefficient
- Recompute nodes during backpropagation
 - Memory efficient, but increases compute



Automatic Mixed Precision

- 32-bits are typically used to store the:
 - Weights
 - Activations
 - Gradients
- Instead, use FP16 for some training values
- Makes training faster and preserves memory



Source: [Jonathan Davis](#)

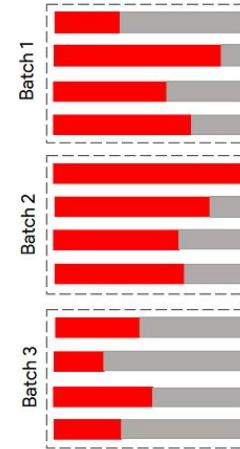
Dynamic Padding & Uniform Length Batching

- All inputs in a batch need to be the same length
- Sequences are padded to the length of the maximum training sample
- [PAD] tokens are still included in all the operations

Training Data

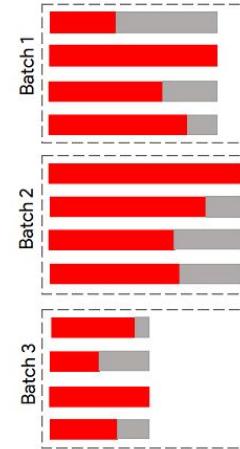


Normal Padding
Until biggest dataset



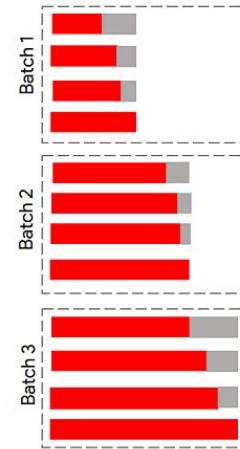
Dataset Size = 12

Dynamic Padding
Until biggest in Batch



Bach Size = 4

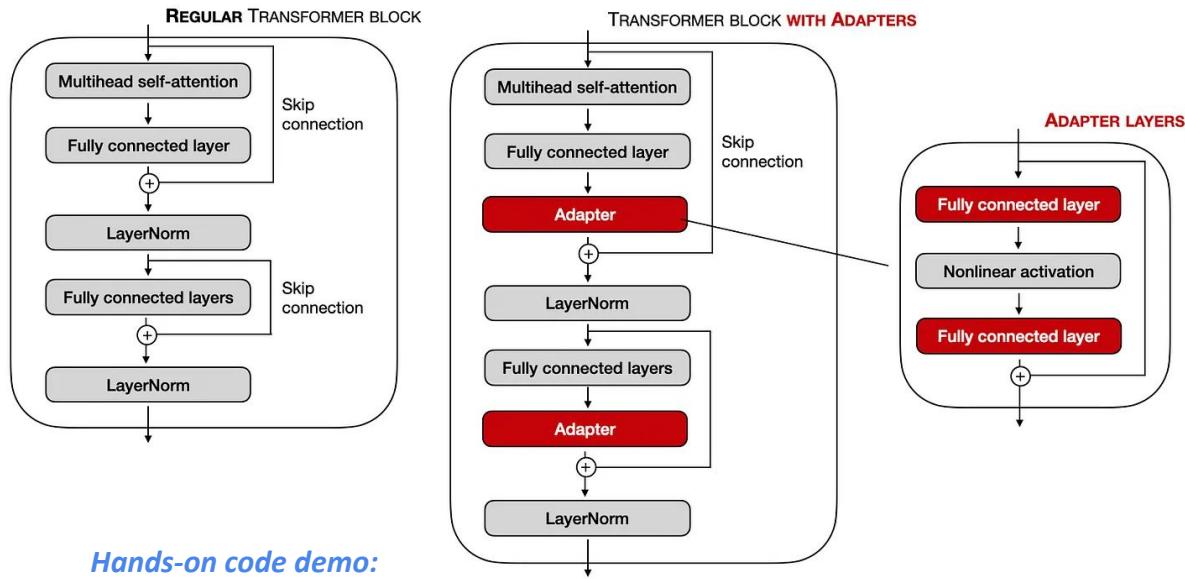
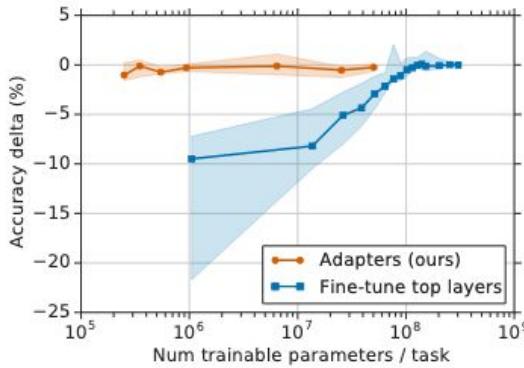
Dynamic Padding
With Sorted dataset



Source: [Sajjad Ayoubi](#)

PEFT with Low-Rank Adaptation

- Reduced computational costs
- Faster training times
- Lower hardware requirements
- Less overfitting
- Less storage



*Hands-on code demo:
PEFT_LoRA.ipynb*

Source: [Sebastian Raschka](#)

Schedule

1. Introduction to Large Language Models (35 mins)
2. Q&A (5 mins) & Break (5 mins)
3. Single GPU Training Techniques (60 mins)
4. Break (10 mins)
5. **Multi-GPU Training Techniques (45 mins)**
6. Summary and Q&A (20 mins)



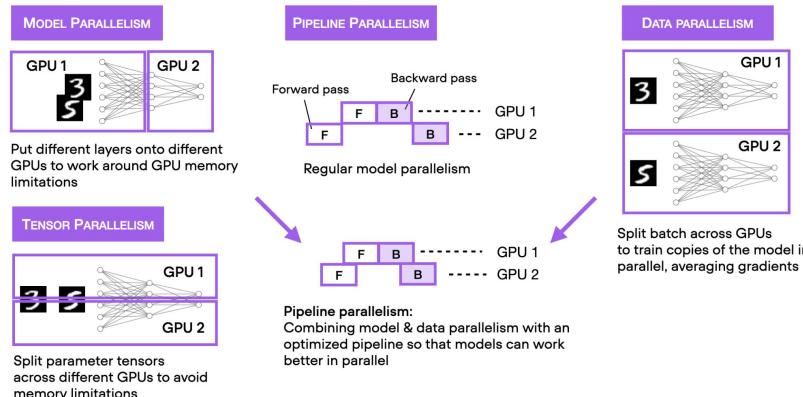
Parallelism: Use more GPUs!

Data Parallelism

- Each GPU receives a small batch and computes the forward and backward passes
- Gradients are averaged across nodes

Model Parallelism

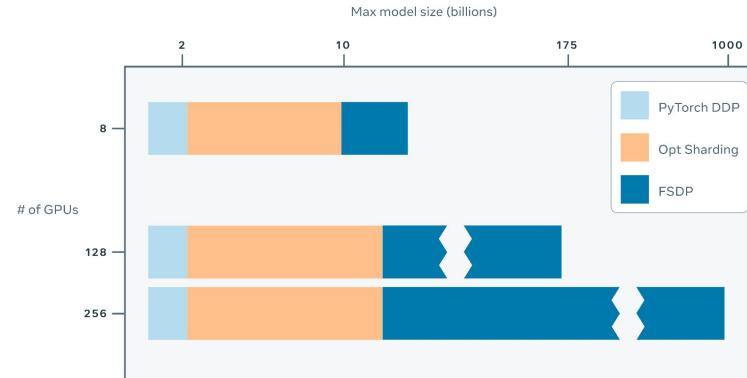
- But what if we can't fit even one copy of the model on a device?
 - GPT-3: 175B parameters
 - $175\text{B} \times 4 \approx 700\text{GB}$
- Put different layers of the model on different devices



Source: [Sebastian Raschka](#)

Fully sharded data parallel (FSDP)

- Shards an AI model's parameters across data parallel workers
- Utilizes both data parallelism and tensor parallelism
- Can offload training computation to CPUs
- Benefits
 - Improves memory efficiency
 - Improves computational efficiency
 - Identical results to distributed data parallel (DDP)

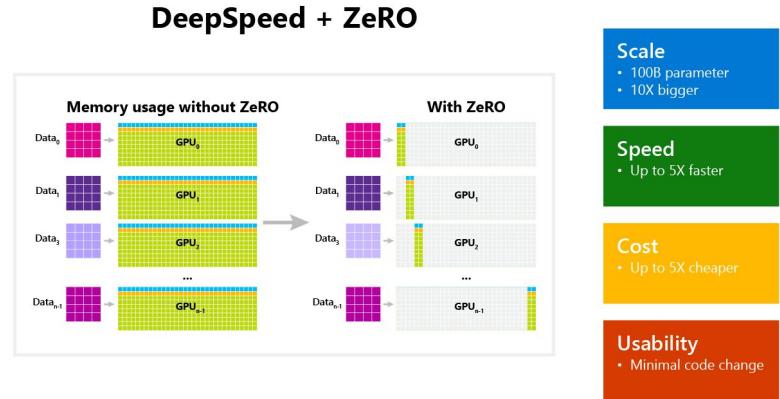


*Hands-on code demo:
code*

Source: [Meta Engineering](#)

DeepSpeed ZeRO

- Easily integrated into Transformers or Lightning
- **Stage 1**
 - Shards optimizer states
 - Train only
- **Stage 2**
 - Shards gradients
 - Train only
- **Stage 3**
 - Shards model parameters
 - Train and inference
- CPU offloading
 - Offload data and compute to CPU/Disk



Hands-on code demo:
T5 Inference.ipynb

Source: [Microsoft](#)

Poll

How much deployment experience do you have?

- No model I've worked on has made it to production
- Trained a model and handed it off to MLE
- Built service/deployed model



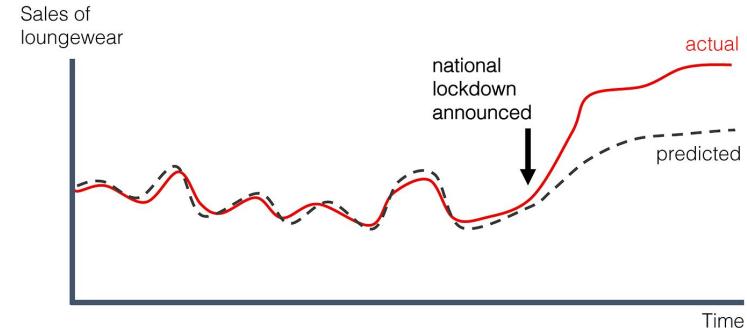
LLM Deployment Options

Tools like HuggingFace and PyTorch make deployment easy:

1. **Batch**: offline inference
2. **Real-time**: more complex MLOps
3. **Edge**: e.g., in user's browser, phone, or watch
 - a. Rare for LLMs yet applicable for other ML

Monitoring LLMs can be challenging:

- **Drift Detection**
 - Data Drift
 - Label Drift
 - Prediction Drift
 - Concept Drift
- **Detection algorithms**
 - Kolmogorov-Smirnov test
 - Population Stability Index
 - Kullback-Leibler divergence



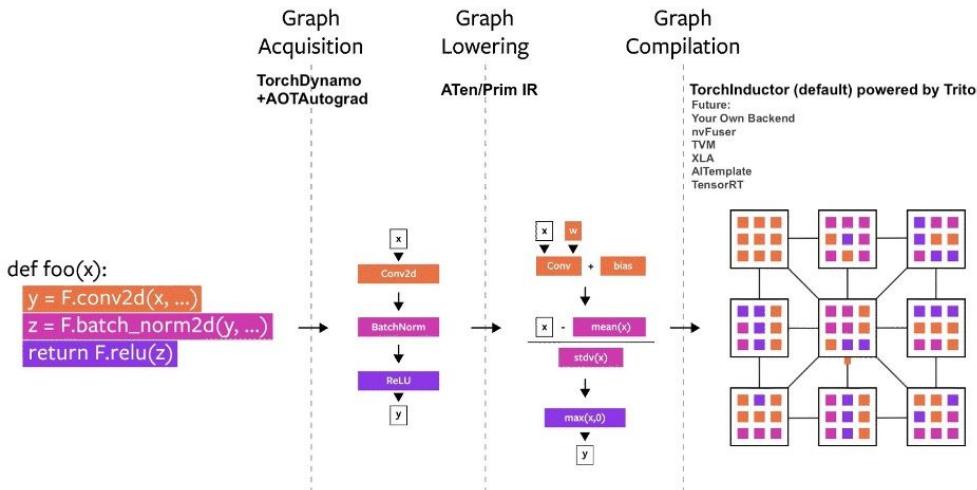
Major LLM Challenges

- **Large size**
 - Using vendor (OpenAI) API for fine-tuning or inference
 - Using easy (HuggingFace inference endpoint) LLM hosting
 - Relatively advanced MLOps
- Very fast moving, complex web of **model options**
 - Great options are out there
 - Often when a model is trained, there's a *better* model
- LLMs are often very **exposed** to users
 - Selective generation calculation
 - Moderation APIs
 - False/"hallucinated" information
 - Harmful
- **Malicious** attacks
 - Prompt injection



Inference Optimizations

- Vectorization & Parallelization
- `torch.compile()`
- Quantization
- Pruning

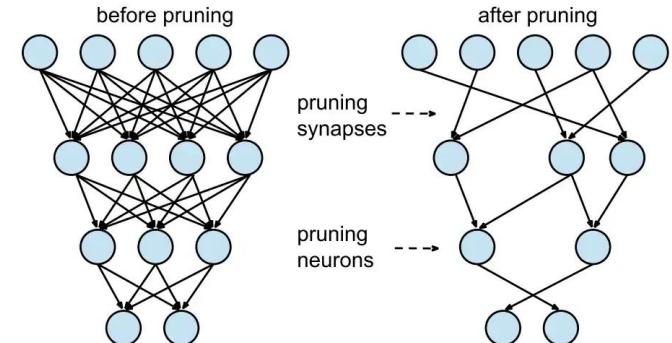
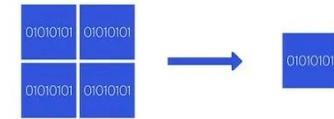


Hands-on code demo:
Inference_Optimizations.ipynb
Quantization.ipynb
4_Bit_Quantization_QLoRA.ipynb

Quantization

Floating point → Integer
3452.3194 → 3452

32 bit → 8 bit



Source: [PyTorch](#)

SparseGPT

- GPT-175B
 - 320GB of storage
 - Five A100 GPUs with 80GB of memory each for inference
- Magnitude pruning: 10% removal before accuracy hit
- Remove 50% w/o accuracy impact
- Layer pruning -> retain outputs

OPT	Sparsity	125M	350M	1.3B	2.7B	6.7B	13B	30B	66B	175B
dense	0%	27.66	22.01	14.63	12.46	10.86	10.12	9.56	9.33	8.34
Magnitude	50%	193.	97.80	1.7e4	265.	969.	1.2e5	168.	4.2e4	4.3e4
SparseGPT	50%	36.89	31.60	17.45	13.44	11.56	11.12	9.77	9.33	8.21
SparseGPT	4:8	44.75	38.82	20.02	14.97	12.54	11.72	10.28	9.66	8.45
SparseGPT	2:4	59.17	50.21	24.04	17.14	14.16	12.91	10.88	10.10	8.73

Table 1: OPT perplexity results on raw-WikiText2.

Source: [Paper](#)

SpQR: Sparse-Quantized Representation

- 4x compression while being **lossless**
- After quantizing, 1% of model's weights result in 75% error
- Outlier weight detection
 - Measure inputs/outputs of uncompressed/quantized models
 - Identify weights whose quantization changes output
 - Retain 16-bit representation for outliers

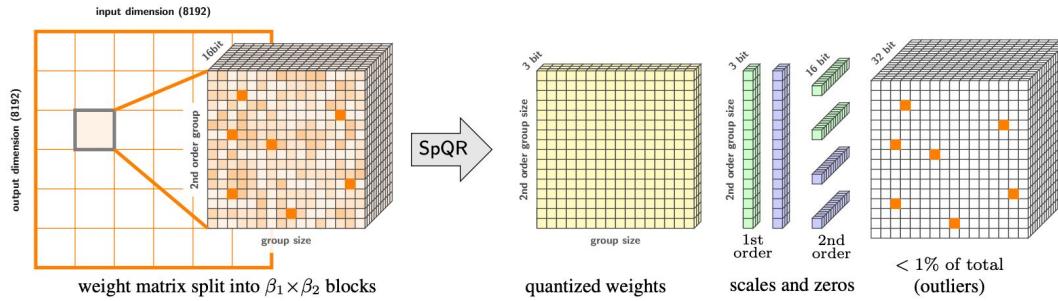
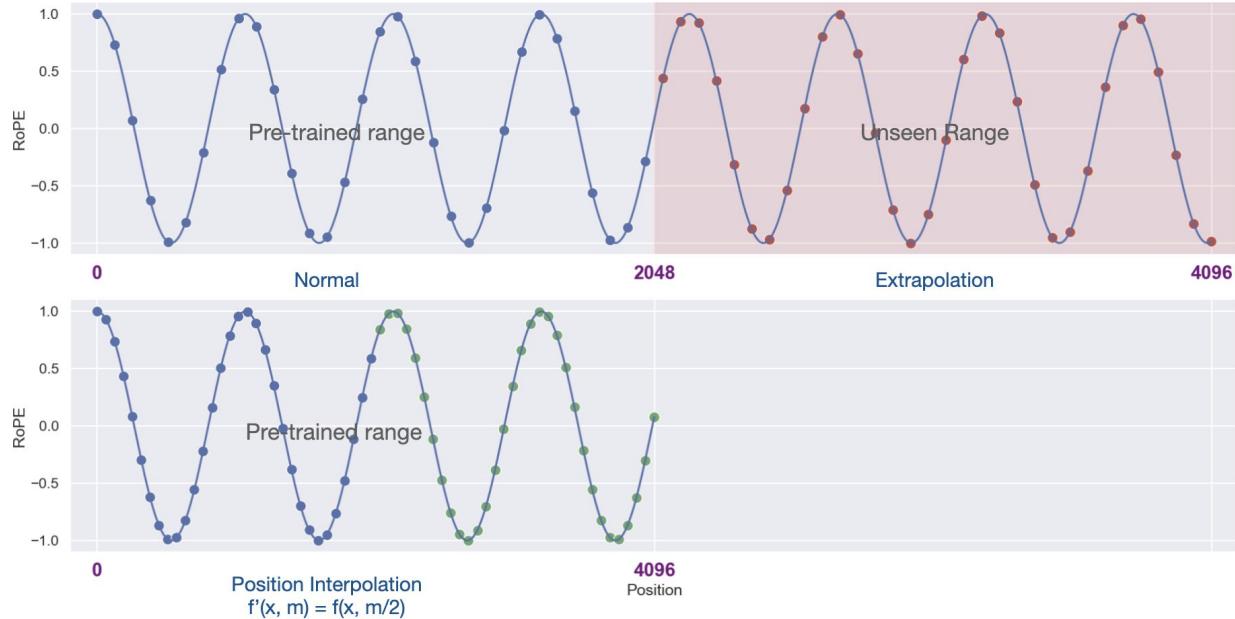


Figure 3: A high-level overview of the SpQR representation for a single weight tensor. The right side of the image depicts all stored data types and their dimensions.

Source: [Paper](#)

Position Interpolation: 32x longer context



Hands-on code demo:
Rope Scaling.ipynb

Source: [Paper](#)