LLM pre-training and scaling laws

▼ Pre-training Large Language models

- Foundation model vs Train your own model
- model architectures and pre-training objectives

Autoencoding models (architecture - encoder only)

Good use cases: BERT, ROBERTA

- sentiment analysis
- NER - name entity recognition
- word classification

Autoregressive Models (architecture - decoder only)

Causal language Modeling (CLM)

- mask token, then forecast the next token, this means unidirectional context

Good Use cases: GBT, BLOOM

- text generation

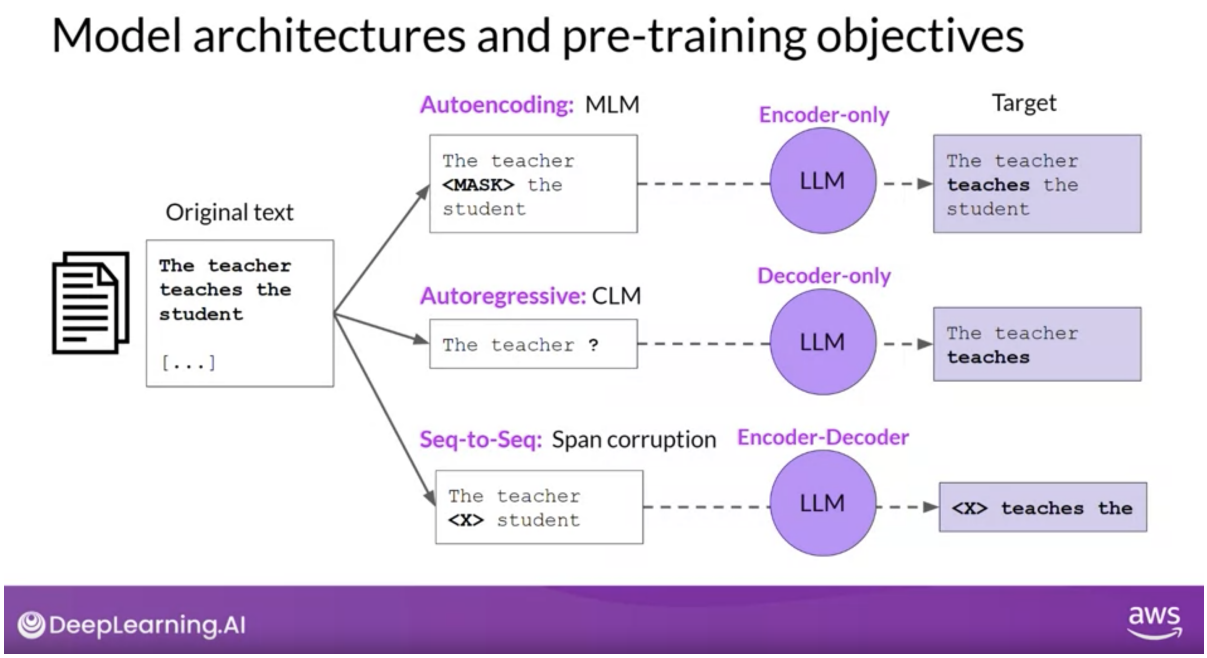
Sequence-to-Sequence Models (architecture - encoder-decoder)

- mask and replace tokens
- objective: reconstruct span

Good use cases: T5, BART

- translation

- text summarization
- question answering

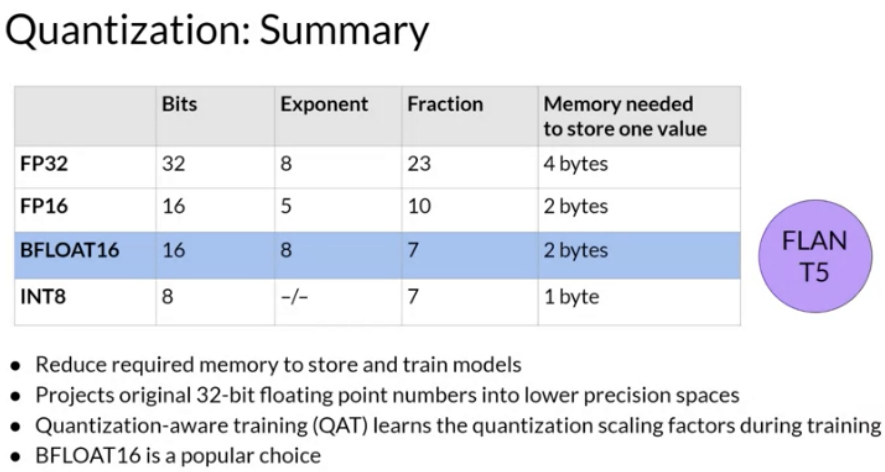


▼ Computational challenges of training LLMs

- Common problem: Out of Memory
- Approximate GPU RAM needed to store 1B parameters
  - 1 parameter = 4 bytes (32-bit float)

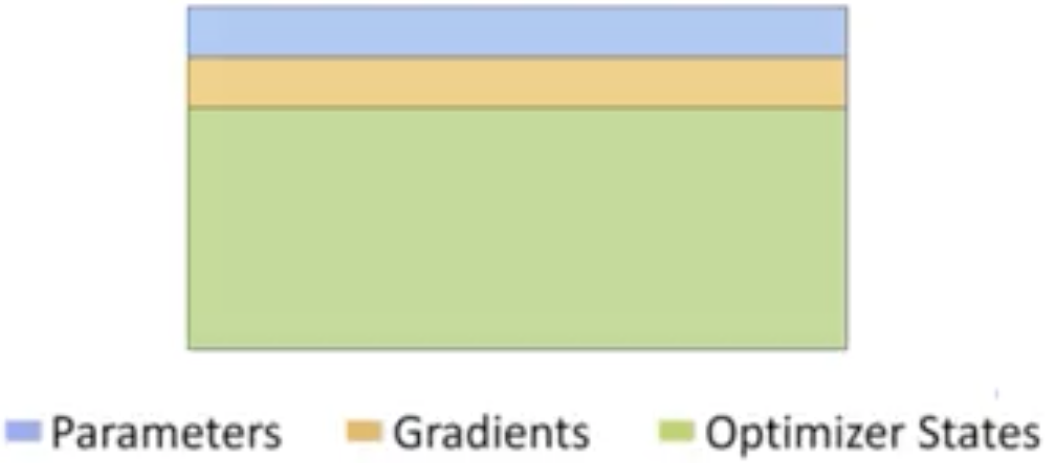
  model parameters (weights), Adam optimizers (2 states), Gradients, activations and temp memory —> 24 bytes per paramaters
  - **eg, 24G RAM for training**
- **Quantization** — the goal of quantization is to reduce the memory required to store and train models by reducing the precision off the model weights.

- reduce from 32-bit floating point to, in the order of reducing precision
  - FP16 in most cases are acceptable
  - **BFLOAT16** is even better in speeding up calculation.
  - INT8



▼ Distributed Data Parallel (DDP) in Pytorch

- Use dataloader to distribute data to each GPU, then synchronize gradients, then update the model - full model copy data on each GPU, waste memory
- Fully Sharded Data Parallel, ZeRO — subset of parameters for each GPU, no repeat



ZeRO stage 1: shard only optimizer states

ZeRO stage 2: shard optimizer states, gradients

ZeRO stage 3: shard optimizer states, gradients, and parameters