CS 577: NATURAL LANGUAGE PROCESSING

Abulhair Saparov

Lecture 15: Quantization

PREVIOUSLY: WORKING WITH LARGE MODELS

- Previously, we discussed how to train and run inference on models via parallelizing them across many GPUs.
- We also discussed how we can use large models even if we don't have access to large GPU clusters.
  - We looked into how to train models that are small enough for us to run inference (i.e., forward passes), but too large for us to fine-tune.
  - Parameter-efficient fine-tuning (PEFT)

PREVIOUSLY: WORKING WITH LARGE MODELS

- But what if the model is too large to fit in memory even for inference alone?
- Is there some way we can make the model smaller and retain accuracy?
  - Or minimize any loss to accuracy?
- Smaller models would be much cheaper.
  - Large models are expensive to train.

		Time (GPU hours)	Power Consumption (W)	Carbon Emitted (tCO_2eq)
Llama 2	7B	184320	400	31.22
	13B	368640	400	62.44
	34B	1038336	350	153.90
	70B	1720320	400	291.42
Total		3311616		539.00

[Touvron et al., 2023]

PREVIOUSLY: WORKING WITH LARGE MODELS

- tCO_2eq is a unit meaning "metric tons of CO_2 equivalent."
- This table only considers the power requirements of GPUs, and not the cost of running the CPUs, interconnects, datacenter cooling, etc.
- This table also only considers the cost of training.
  - For popular models, the cost of inference quickly outpaces the cost of training.

		Time (GPU hours)	Power Consumption (W)	Carbon Emitted (tCO_2eq)
Llama 2	7B	184320	400	31.22
	13B	368640	400	62.44
	34B	1038336	350	153.90
	70B	1720320	400	291.42
Total		3311616		539.00

[Touvron et al., 2023]

PREVIOUSLY: WORKING WITH LARGE MODELS

- Future AI data centers and clusters are projected to continue using more and more power,
- And therefore, producing more and more greenhouse gases.

Figure 1a: AI LLM data center share of total summer generating capacity

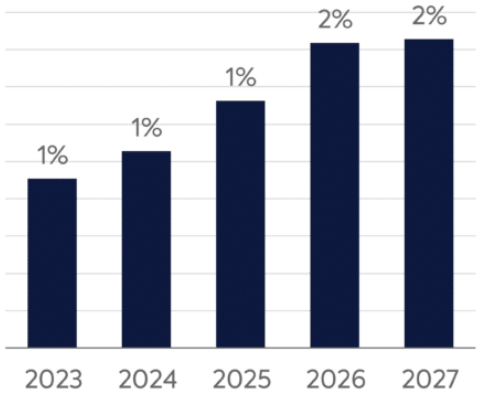
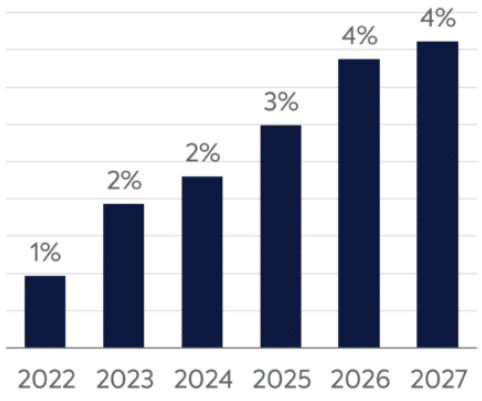


Figure 1b: AI LLM data center share of total energy sales



Source: Data derived from the EIA's Annual Energy Outlook 2023 and the authors' AI LLM demand projections.

[Jafari et al., Projecting the Electricity Demand Growth of Generative AI Large Language Models in the US, 2024]

PREVIOUSLY: WORKING WITH ELS

- Future AI data centers and clusters are projected re and more power,

and more greenh

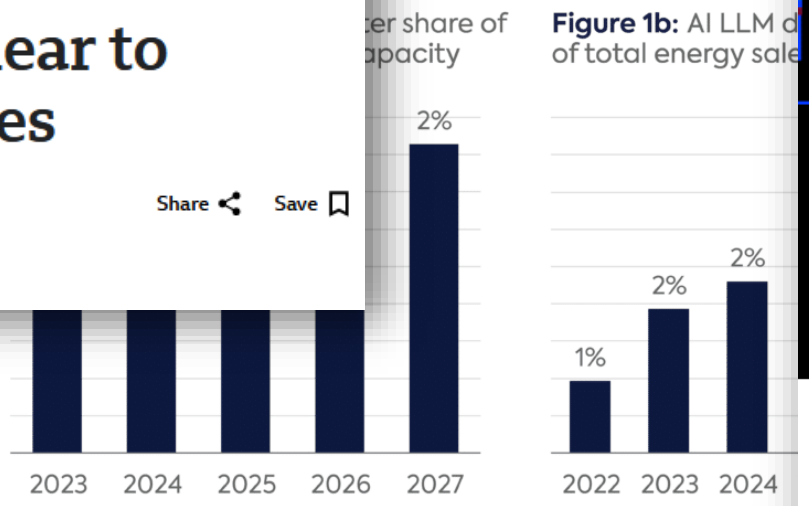
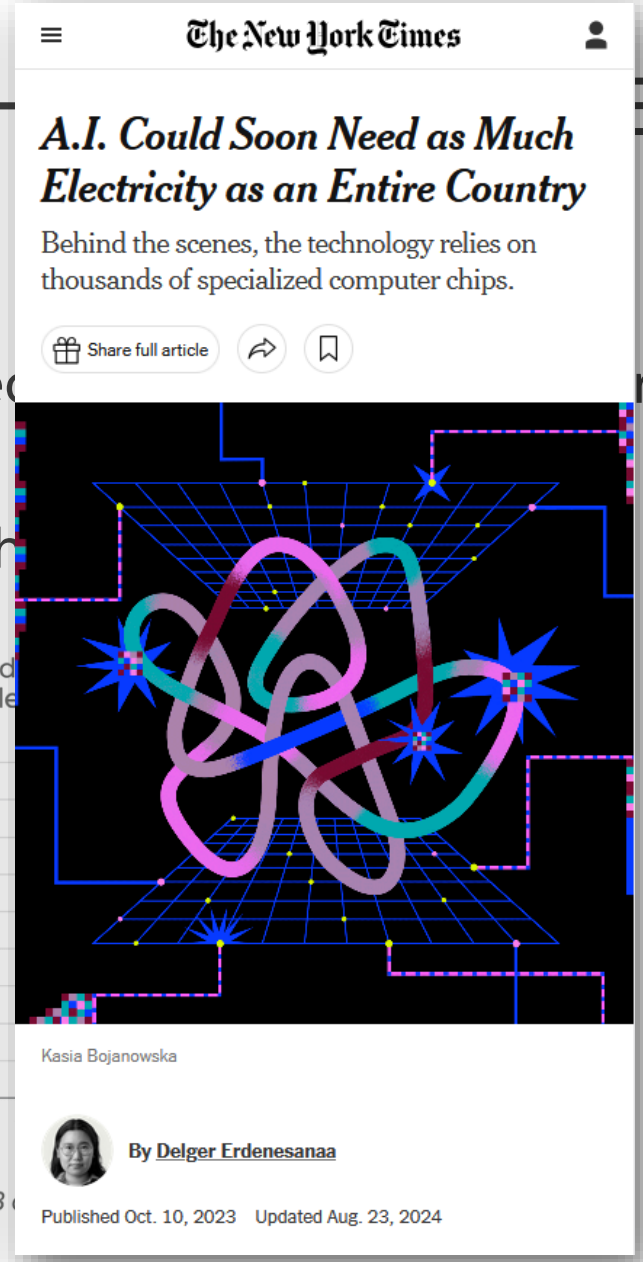


Figure 1b: AI LLM d
of total energy sale

Source: Data derived from the EIA's Annual Energy Outlook 2023
LLM demand projections.



[Jafari et al., Projecting the Electricity Demand Growth of Generative AI Large Language Models in the US, 2024]

MODEL COMPRESSION

- Effective model compression can potentially help to reduce the costs of training and inference.
- Compression can help to broaden access to large models.
- There are three broad categories of model compression techniques:
  - Quantization: Reduce the precision of the floating-point numbers in the model.
    - How can we reduce the precision without adversely affecting the model's accuracy?
    - It's not so simple, especially with very low precision.
  - Distillation: Use a larger model to train a small model.
  - Pruning: Remove parts of the model while minimizing any adverse effects on model performance.

MODEL COMPRESSION

- Why should we think it is even possible to compress models?
- Trained models may be overparameterized.
  - I.e., they have more parameters than they need to learn a task.
- Du and Lee (2018) showed that:

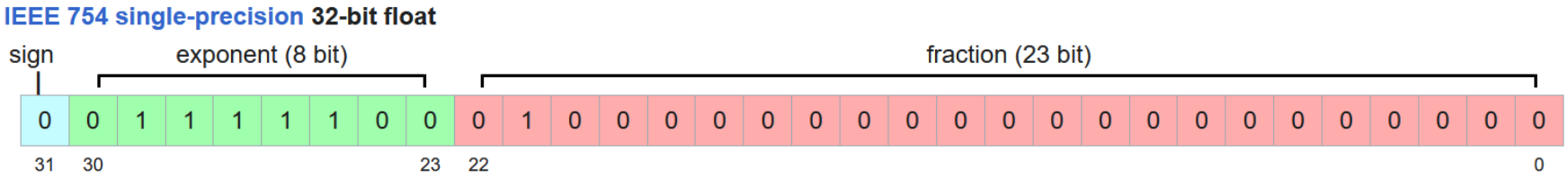
> For a k hidden node shallow network with quadratic activation and n training data points, we show as long as $k \geq \sqrt{2n}$, over-parametrization enables local search algorithms to find a *globally* optimal solution for general smooth and convex loss functions.

MODEL COMPRESSION

- But there are some caveats to the findings of Du and Lee (2018):
  - They only considered shallow networks.
  - They assumed the loss function is convex.
    - Loss functions for almost all large real-world models are not convex.
- Allen-Zhu et al. (2018) showed that if a network contains a subnetwork that is able to perform the target task,
  - And there is sufficient available training data,
  - Then the overparameterized network can learn the task without overfitting.
- So there is hope that large models might contain these smaller "subnetworks" that are able to perform tasks with similar accuracy.

QUANTIZATION

- The first model compression approach we will consider is quantization.
- We can reduce the size of the model by reducing the precision of each parameter.

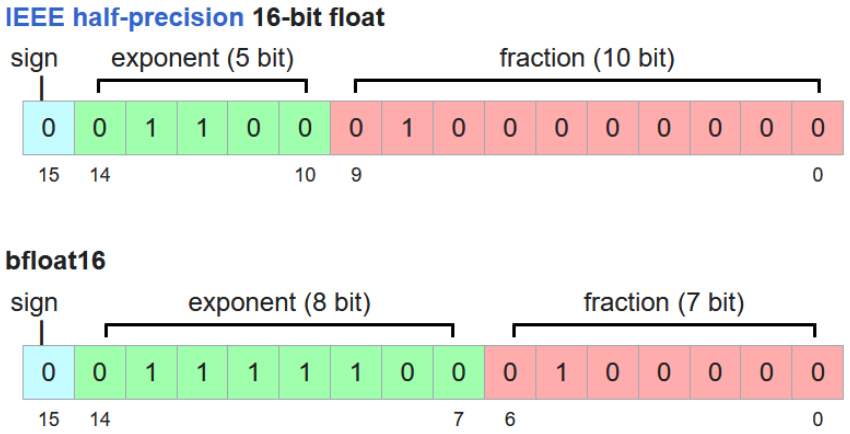


- This floating-point number represents $(-1)^{s} \cdot 2^{e-127} \cdot 1.f$
- Where s is the sign, e is the exponent (also called range), and f is the fraction (also called mantissa or precision) in binary.

[en.wikipedia.org/wiki/Bfloat16_floating-point_format]

QUANTIZATION

- There are smaller floating-point formats:

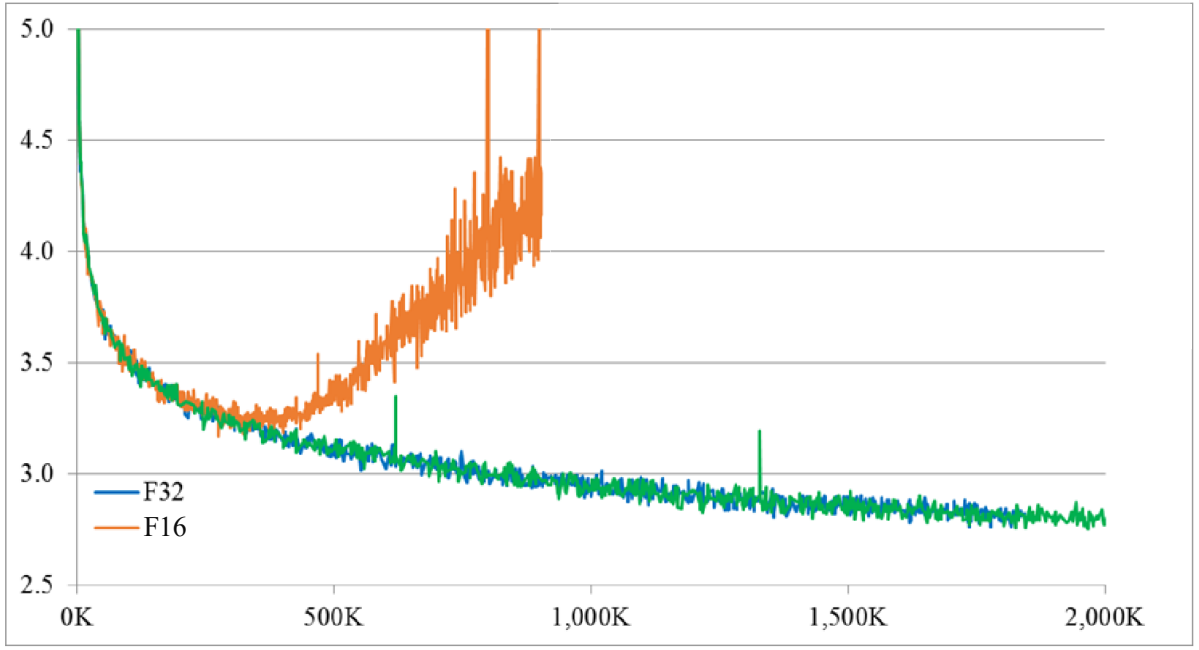


- `bf16` or `bfloat16` ("Brain float-16"; Google Brain, 2018) allocates more bits to the exponent (same number of bits as `fp32`) and fewer bits to the fraction.
  - A wider range of numbers can be represented in `bf16` compared to `fp16`.
  - At the expense of the precision.

[en.wikipedia.org/wiki/Bfloat16_floating-point_format]

QUANTIZATION DURING TRAINING

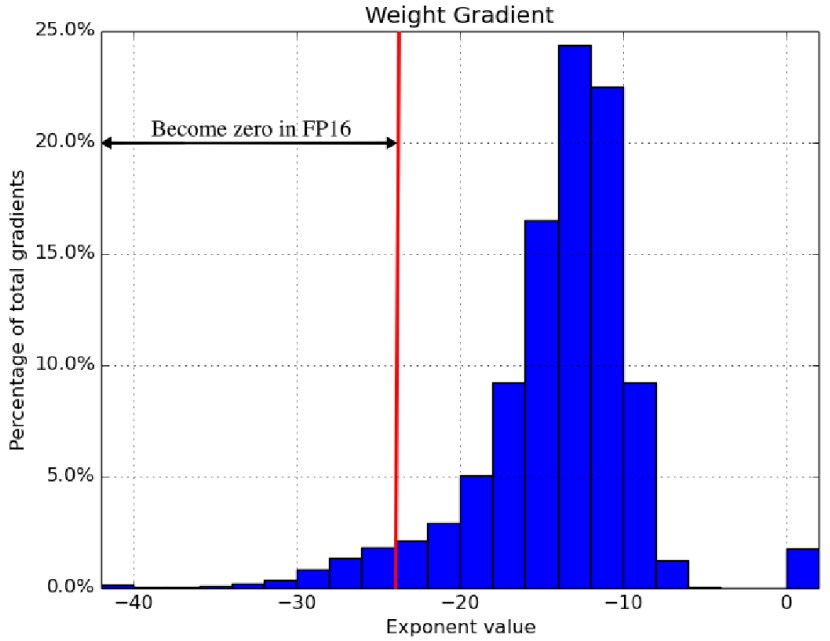
- Training is very sensitive to floating-point precision.
- Simply reducing the precision of all parameters from `fp32` to `fp16` leads to instability during training.



[Narang et al., 2018]

QUANTIZATION DURING TRAINING

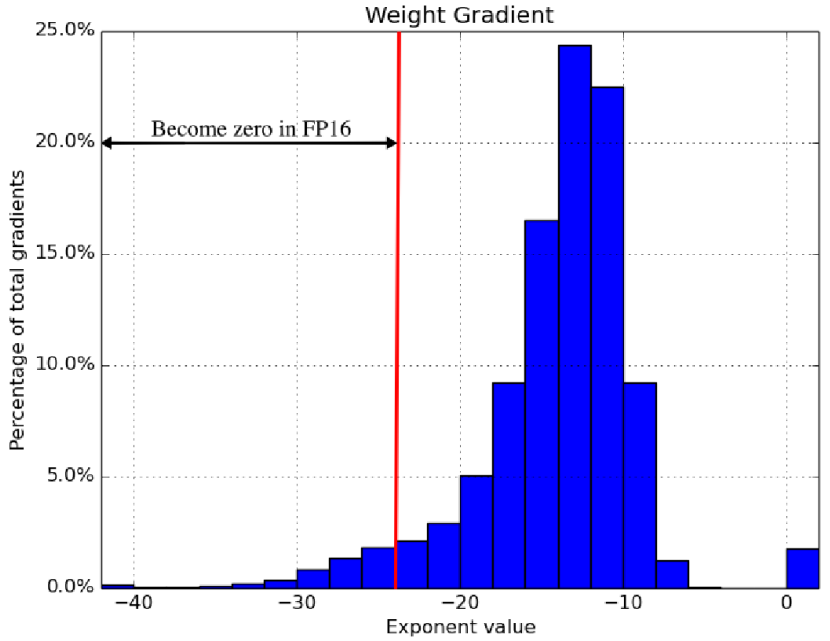
- Why do quantized models diverge during training?
- Inspect the gradients of the parameters during `fp32` training, and plot a histogram of their magnitudes:



[Narang et al., 2018]

QUANTIZATION DURING TRAINING

- Possible reason for training instability when using fp16:
  - Small gradients are important for training.
  - When naively switching from `fp32` to `fp16`, they are rounded to zero.



[Narang et al., 2018]

QUANTIZATION DURING TRAINING

- Possible reason for training instability when using fp16:
  - Small gradients are important for training.
  - When naively switching from `fp32` to `fp16`, they are rounded to zero.
- This is especially problematic for larger models since they require a smaller learning rate for training.
- Is there a way we can avoid underflow?

QUANTIZATION DURING TRAINING

- What if we keep the weights in high precision (`fp32`) but perform and forward and backward pass in lower precision (`fp16`)?
- But we still have the problem where the gradients computed in the backward pass will be rounded to zero.
- What if we scale the gradients by a constant factor?

$$W_{new} = W - \gamma \frac{\partial L}{\partial W}$$

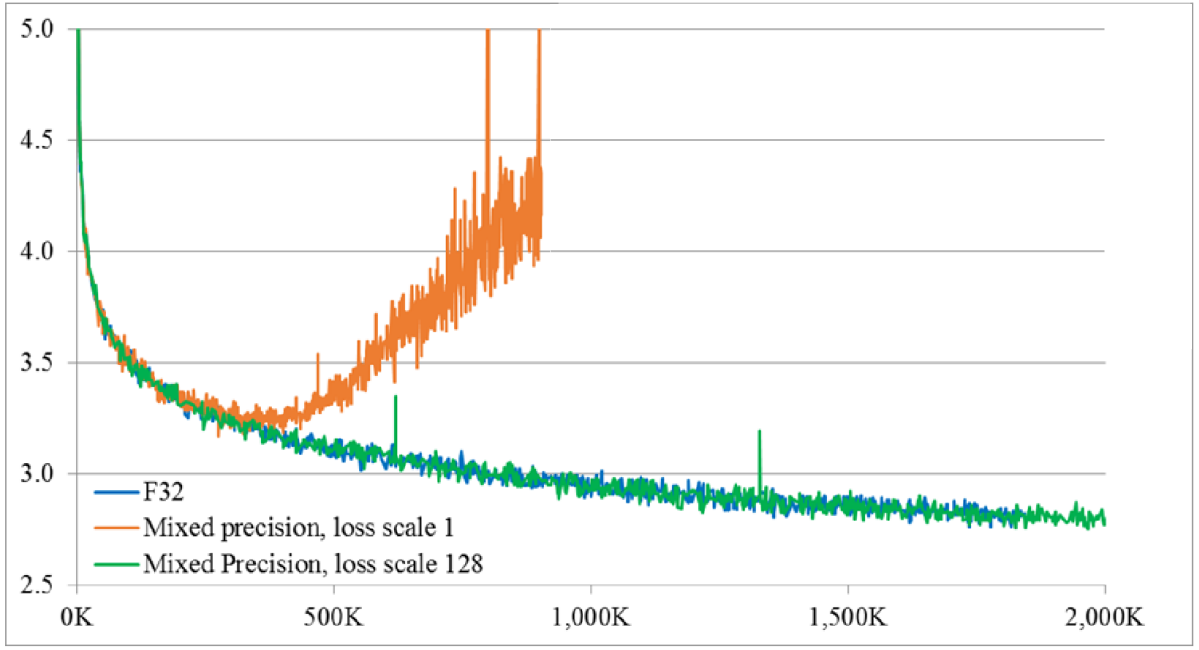
$$= W - \frac{\gamma}{S} \frac{\partial}{\partial W} (SL)$$

We compute the red portion in `fp16` before converting to `fp32`.

  - This is possible due to the linearity of the derivative operator.
  - This has the effect of shifting the magnitude of the gradients further away from 0, where there is a much lower risk of underflow.

MIXED PRECISION TRAINING

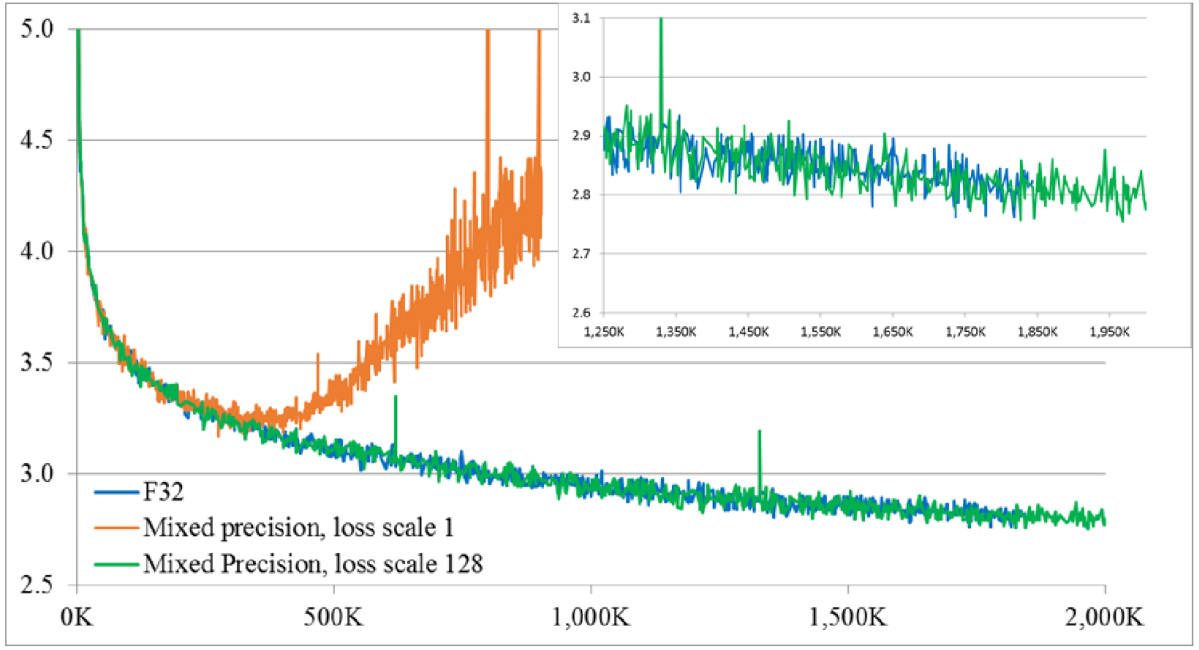
- This approach is called mixed-precision training.



[Narang et al., 2018]

MIXED PRECISION TRAINING

- This approach is called mixed-precision training.
- With an appropriate value for S, it can perform very similar to full-precison training.



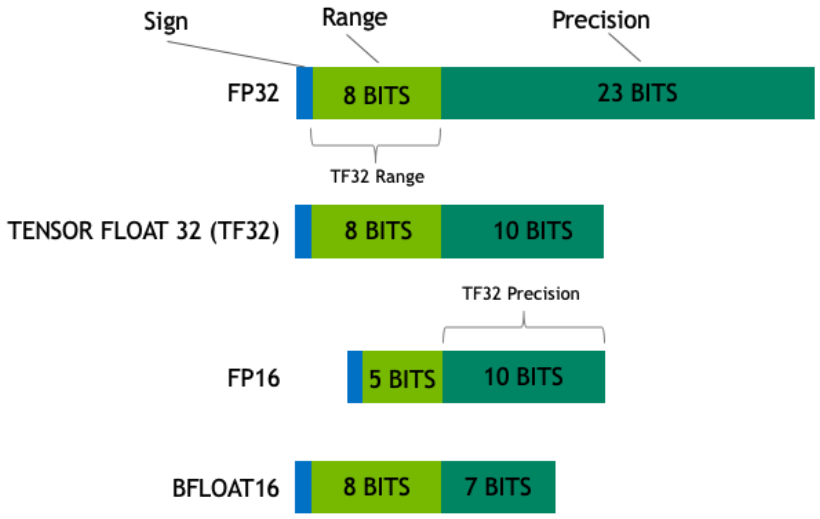
[Narang et al., 2018]

MIXED PRECISION TRAINING

- This approach is called mixed-precision training.
- With an appropriate value for S, it can perform very similar to full-precison training.
- We can reduce the memory requirement of training by *almost half*:
  - Only the model parameters require high-precision.
  - Everything else (the activations, gradients) can be stored in half precision.

WE DON'T NEED 32-BITS PER PARAMETER

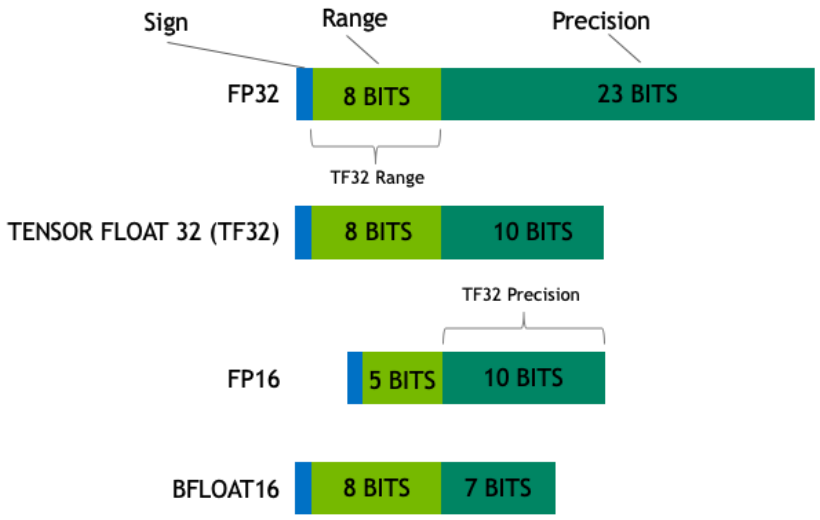
- Is there a middle-ground between `fp32` and `fp16` that would maintain stability during training?
- What if we used the same number of exponent bits as `fp32` (8 bits) and the same number of fraction bits as `fp16` (10 bits)?



[Kharya, TensorFloat-32 in the A100 GPU Accelerates AI Training, HPC up to 20x, 2020]

TF32

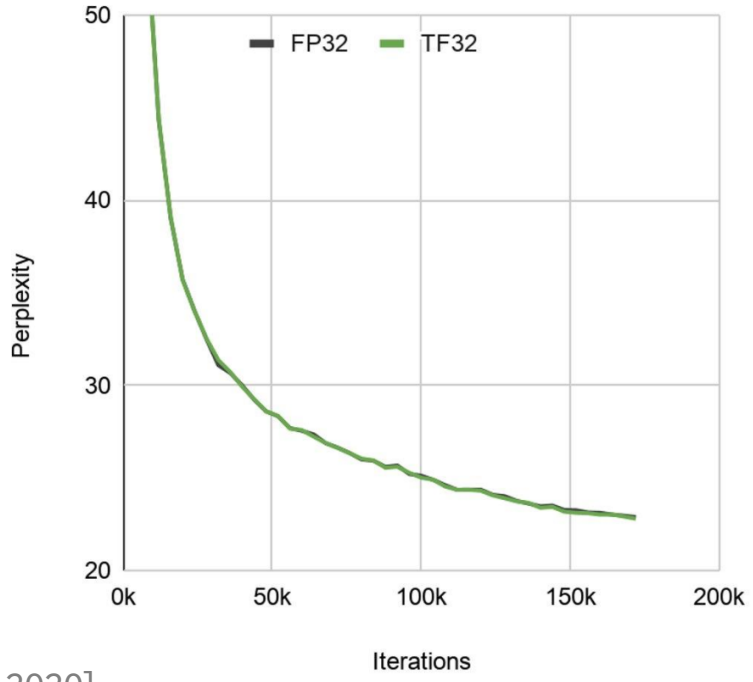
- The proposed format is called TensorFloat-32 or `tf32` (Nvidia, 2020).
- Note that only 19 bits are useful, and the remaining 13 bits are padding.
- So there are no memory savings as compared to `fp32`, but arithmetic operations can be much faster.



[Kharya, TensorFloat-32 in the A100 GPU Accelerates AI Training, HPC up to 20x, 2020]

TRAINING WITH TF32 VS FP32

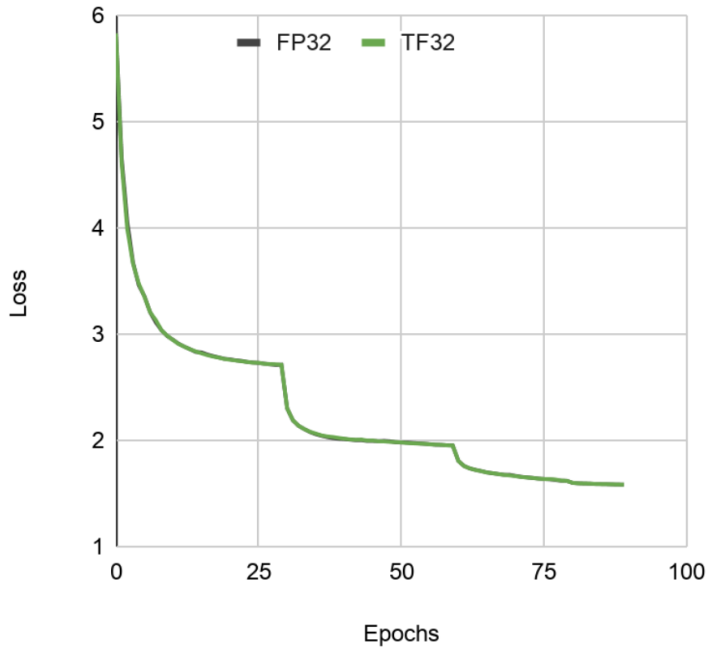
- Experiments demonstrate that training/using models with `tf32` is similar to `fp32`.
- For example, training a Transformer-XL model:

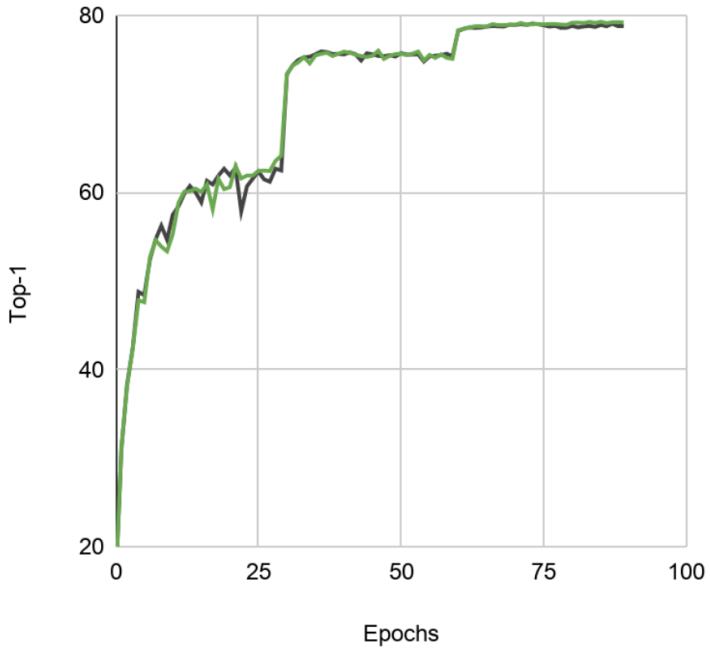


[Stosic, Training Neural Networks with Tensor Cores, 2020]

TRAINING WITH TF32 VS FP32

- Experiments demonstrate that training/using models with `tf32` is similar to `fp32`.
- For example, training `ResNeXt101`:





[Stosic, Training Neural Networks with Tensor Cores, 2020]

QUANTIZED MODELS ARE FASTER

- In addition to reducing memory footprint,
- Quantization can significantly improve the speed of model operations.
- E.g., on an P100 GPU:
  - `fp64` (“`double`” in C/C++/Java): 5.3 TFLOPs/s
  - `fp32` (“`float`” in C/C++/Java): 10.6 TFLOPs/s
  - `fp16`: 21.2 TFLOPs/s

[en.wikipedia.org/wiki/Hopper_(microarchitecture)]

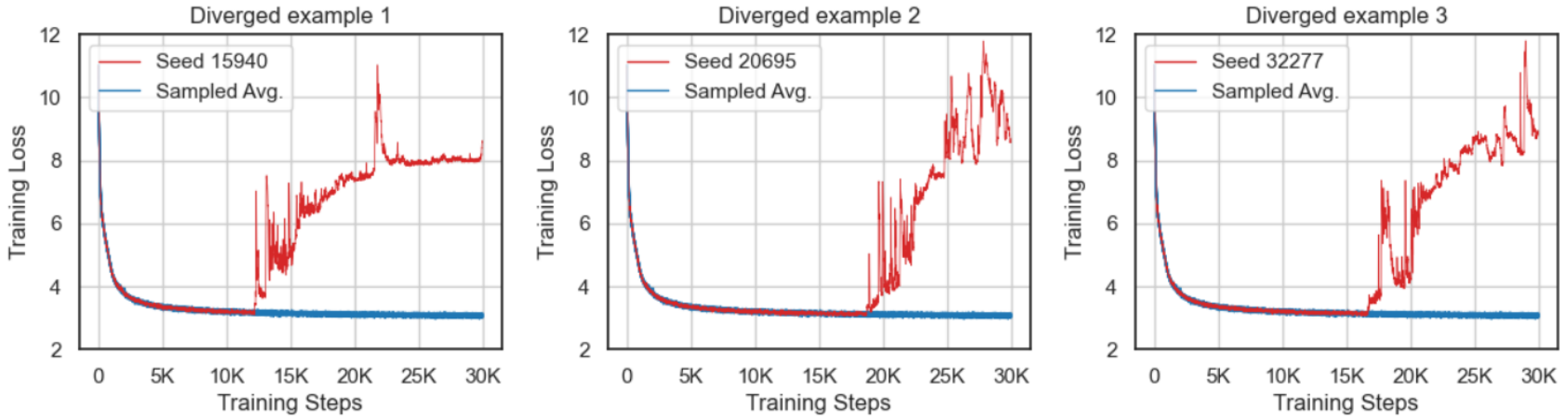
QUANTIZED MODELS ARE FASTER

- In addition to reducing memory footprint,
- Quantization can significantly improve the speed of model operations.
- Newer GPUs have specialized hardware for reduced-precision arithmetic.
- E.g., on an H200 GPU:
  - `fp64` ("`double`" in C/C++/Java): 34 TFLOPs/s
  - `fp32` ("`float`" in C/C++/Java): 67 TFLOPs/s
  - `tf32`: 495 TFLOPs/s
  - `fp16`: 990 TFLOPs/s
  - `bfloat16`: 990 TFLOPs/s
  - `int8` ("`char`" in C/C++, "`byte`" in Java): 1980 TOPs/s

[en.wikipedia.org/wiki/Hopper_(microarchitecture)]

MIXED PRECISION TRAINING

- But mixed-precision training isn't completely free of instability.
- Lee et al., 2024, attempted to train GPT-2 for 188 different random initial seed values.
  - They used mixed-precision training with `bf16` and `tf32` formats.
  - They found 18 of the 188 seeds diverged (about 5%).
  - Some examples of diverged training runs:

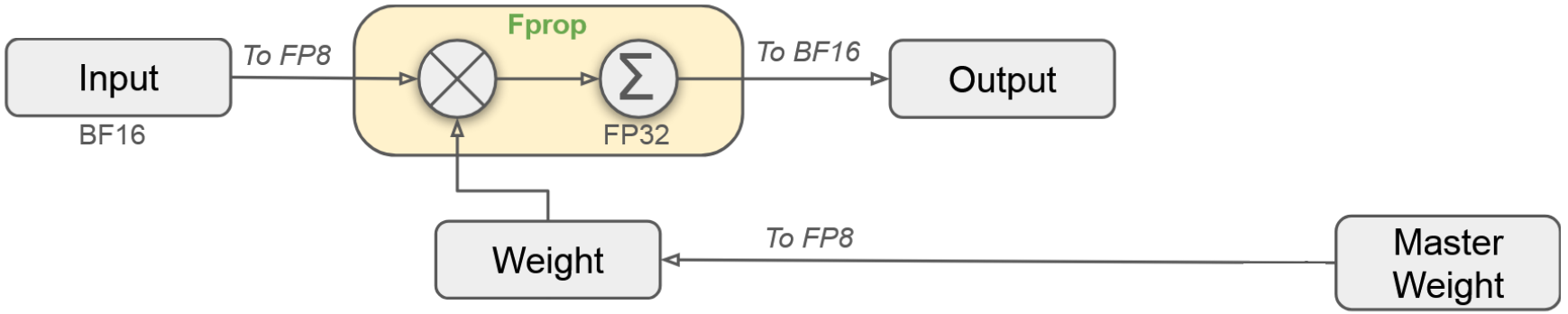


MIXED PRECISION TRAINING

- But mixed-precision training isn't completely free of instability.
- Lee et al., 2024, attempted to train GPT-2 for 188 different random initial seed values.
  - They used mixed-precision training with `bf16` and `tf32` formats.
  - They found 18 of the 188 seeds diverged (about 5%).
  - However, they used early stopping to be able to experiment with a larger number seeds.
  - So they estimate the true divergence rate is closer to 10% of seeds.

MIXED PRECISION TRAINING

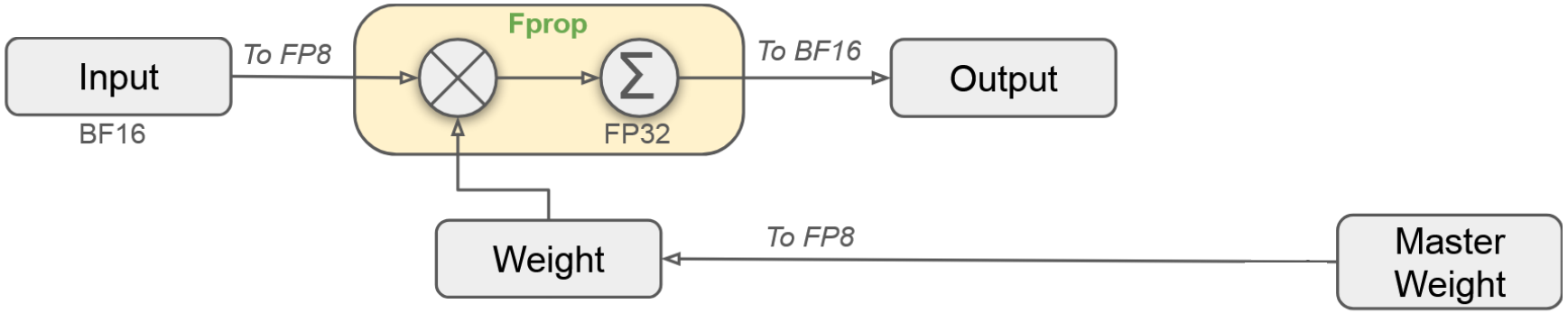
- DeepSeek-AI, 2024, performs mixed-precision training where weights are stored in `fp32` and matrix products are computed in `fp8`.
- Only one matrix product is shown here ("Fprop" means forward pass).



$Y = XA$
where X is the input,
Y is the output,
and A are the weights.

MIXED PRECISION TRAINING

- DeepSeek-AI, 2024, performs mixed-precision training where weights are stored in `fp32` and matrix products are computed in `fp8`.
- Activations are in `bf16` but are converted to `fp8` for matrix multiplications.



$Y = XA$
where X is the input,
Y is the output,
and A are the weights.

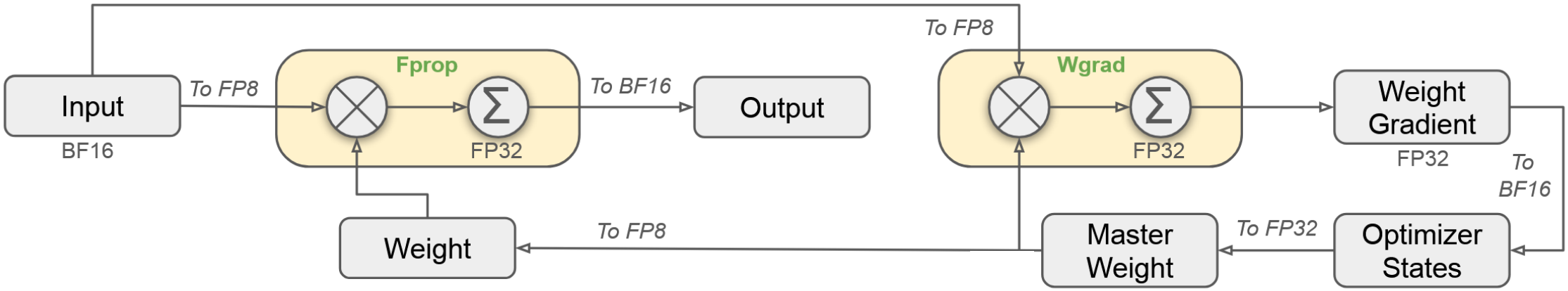
Recall: In the backward pass,
we need to compute

$$\frac{\partial L}{\partial A} = \frac{\partial L}{\partial Y}\frac{\partial Y}{\partial A} = X^{\mathrm{T}}\frac{\partial L}{\partial Y} \quad (\text{“Wgrad”})$$

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y}\frac{\partial Y}{\partial X} = \frac{\partial L}{\partial Y}A^{\mathrm{T}} \quad (\text{“Dgrad”})$$

MIXED PRECISION TRAINING

- DeepSeek-AI, 2024, performs mixed-precision training where weights are stored in `fp32` and matrix products are computed in `fp8`.



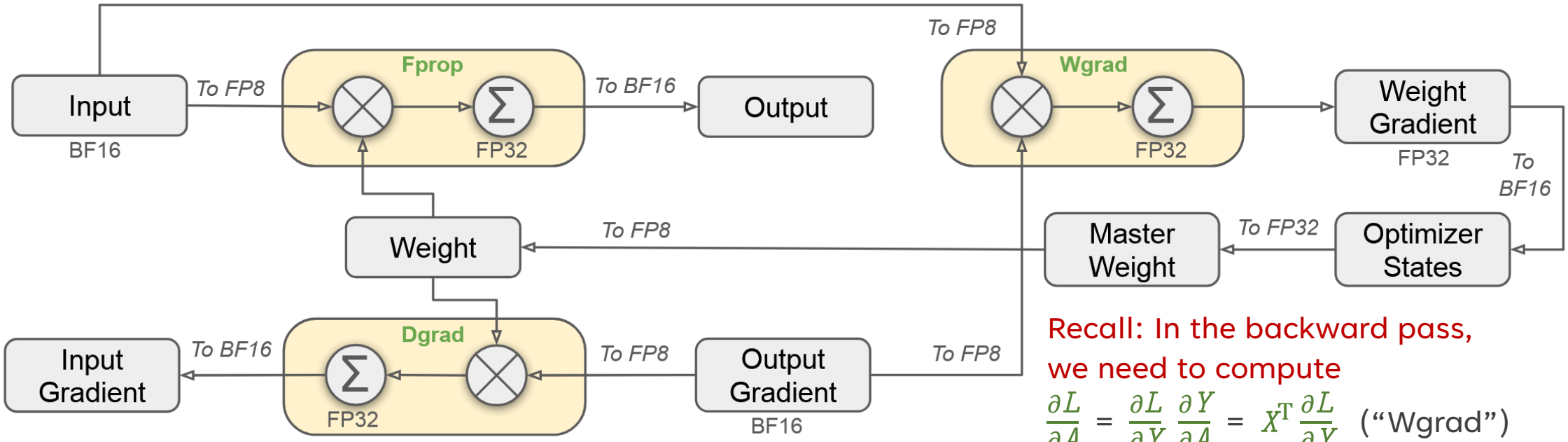
Recall: In the backward pass, we need to compute

$$\frac{\partial L}{\partial A} = \frac{\partial L}{\partial Y}\frac{\partial Y}{\partial A} = X^{\mathrm{T}}\frac{\partial L}{\partial Y} \quad (\text{“Wgrad”})$$

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y}\frac{\partial Y}{\partial X} = \frac{\partial L}{\partial Y}A^{\mathrm{T}} \quad (\text{“Dgrad”})$$

MIXED PRECISION TRAINING

- DeepSeek-AI, 2024, performs mixed-precision training where weights are stored in `fp32` and matrix products are computed in `fp8`.



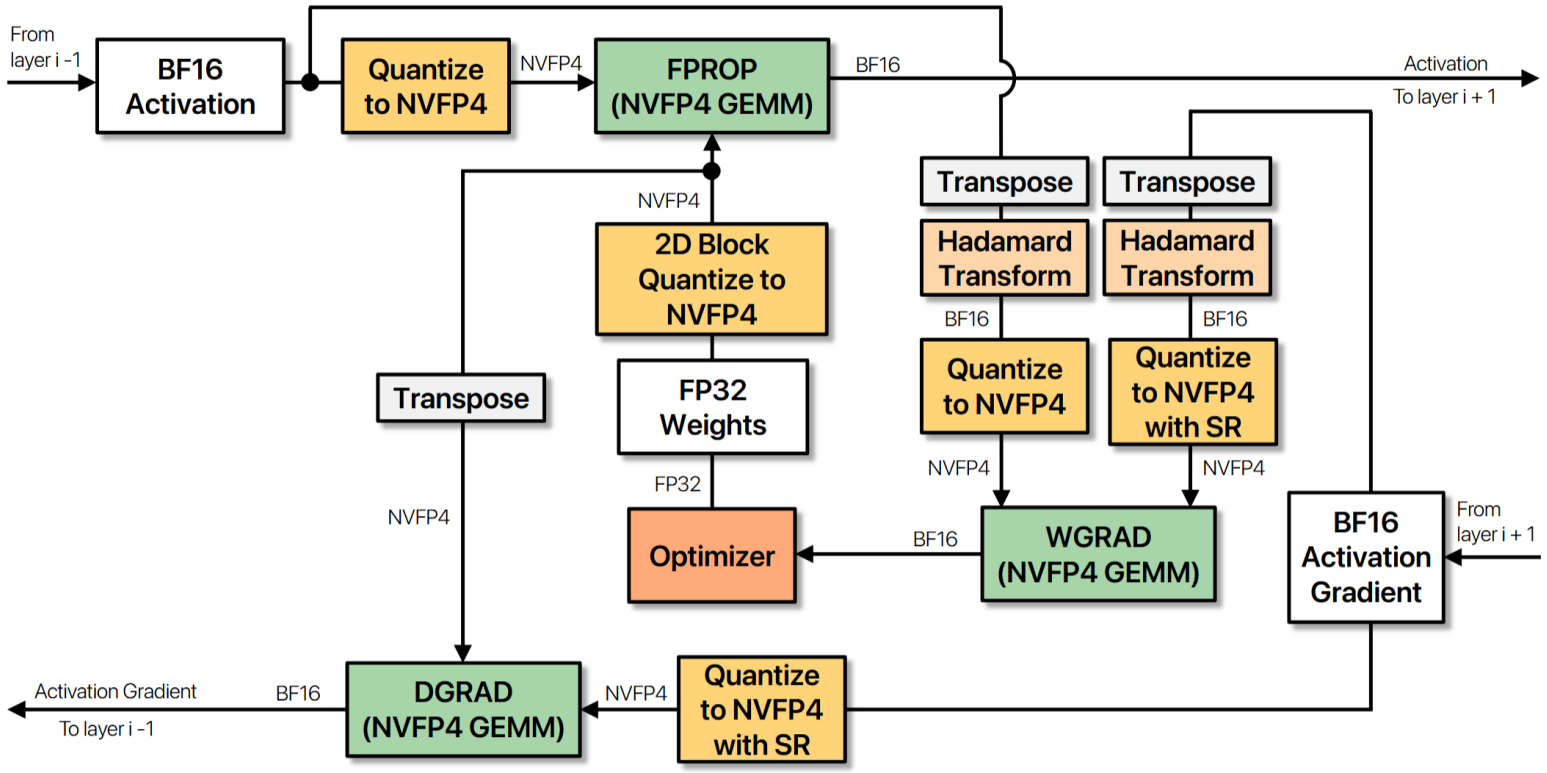
Recall: In the backward pass, we need to compute

$$\frac{\partial L}{\partial A} = \frac{\partial L}{\partial Y}\frac{\partial Y}{\partial A} = X^{\mathrm{T}}\frac{\partial L}{\partial Y} \quad (\text{“Wgrad”})$$

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y}\frac{\partial Y}{\partial X} = \frac{\partial L}{\partial Y}A^{\mathrm{T}} \quad (\text{“Dgrad”})$$

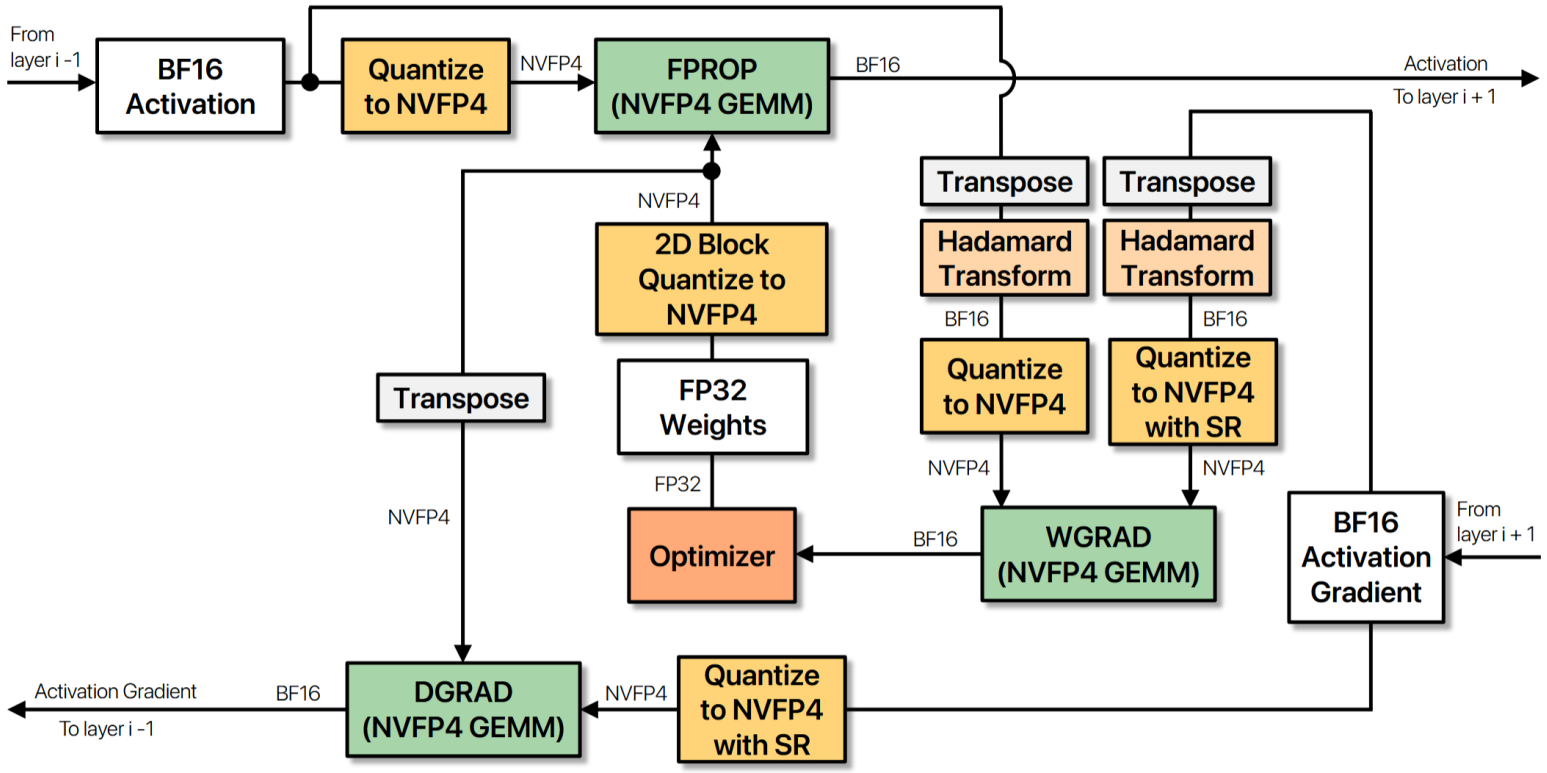
MIXED PRECISION TRAINING

- NVIDIA (2025) recently proposed a mixed-precision training method where weights are stored in fp32 and matrix products are computed in 4-bits.



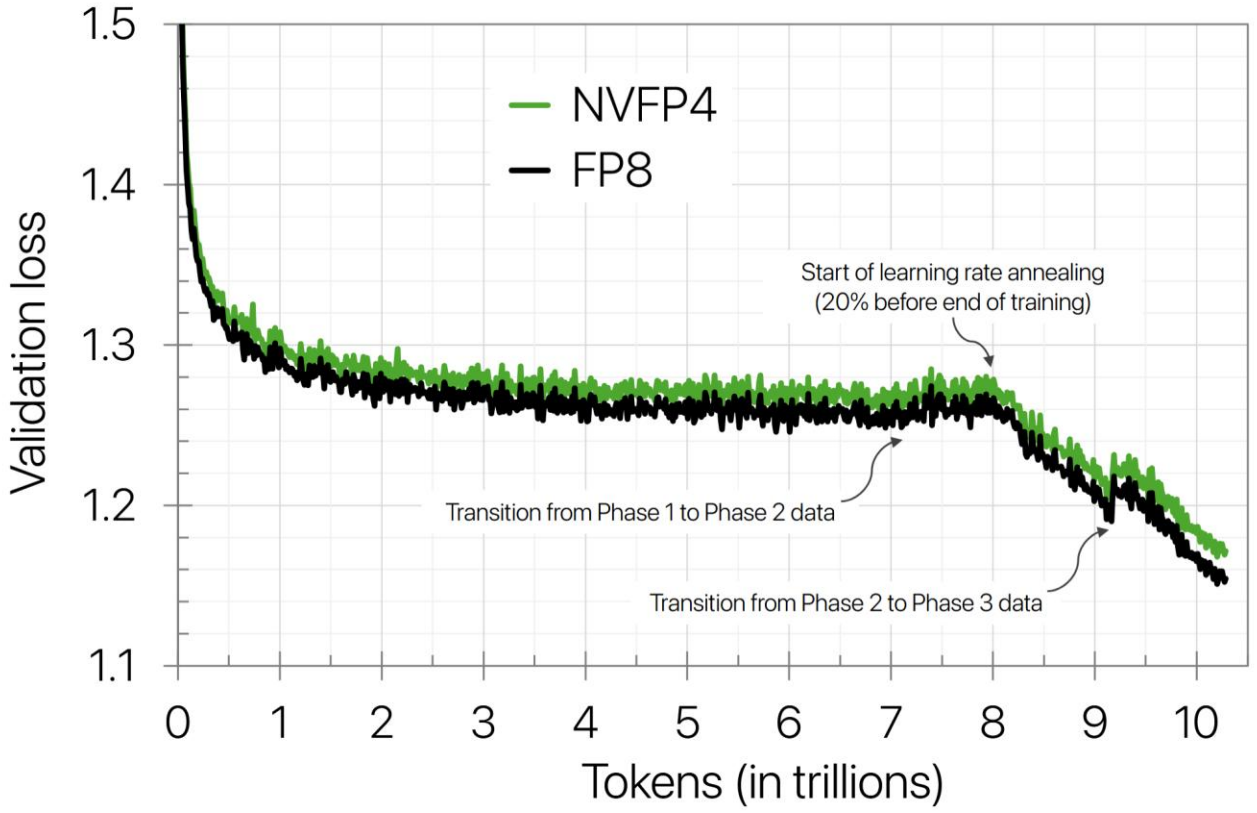
MIXED PRECISION TRAINING

- They introduce a new 4-bit floating-point format called `nvfp4`.



MIXED PRECISION TRAINING

- They introduce a new 4-bit floating-point format called `nvfp4`.



EVEN SMALLER NUMBER FORMATS

- Smaller floating-point layouts are possible:
  - `fp8` or `float8`: 1 bit sign, 4 bit exponent, 3 bit fraction
  - `fp4` or `float4`: 1 bit sign, 2 bit exponent, 1 bit fraction
- Numbers can be converted into integers:
  - `int8`
  - `int4`
  - `int3`
  - `int2`
  - `int1` (binary)

HARDWARE SUPPORT FOR QUANTIZATION

- GPUs don’t support arbitrary floating-point or integer formats.
- We can imagine how fast `int4` or `fp4` operations would be, but most GPUs simply don’t provide hardware acceleration support.
- Many software frameworks still don’t support many small number formats.
  - PyTorch does not currently support `int3` or `fp4` (as of March 2025),
  - And has limited support for `int4` and `fp8`.
- When we perform arithmetic operations using these number formats on GPUs without hardware support, we first must convert them into a supported number format.

HARDWARE SUPPORT FOR QUANTIZATION

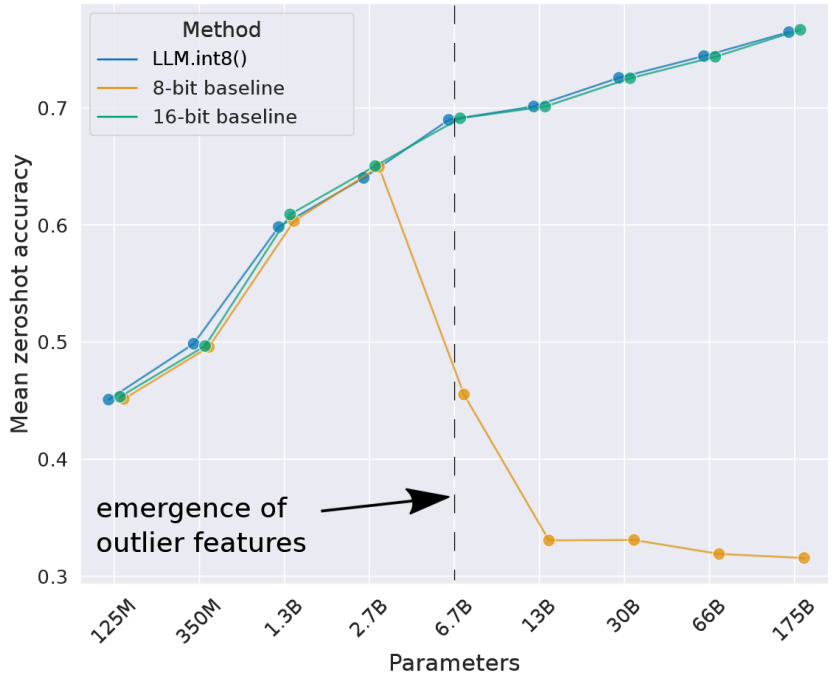
- We can still use these smaller number formats, since they still provide reduced memory footprint.
  - The speed of the operation will be the same as the supported number format.
  - Number formats smaller than `fp16` cause too much instability during training,
    - But inference is much more robust to quantization.
    - Thus smaller number formats are currently only used for inference.
- Let’s see how to quantize models into these smaller formats for inference.

INT8 QUANTIZATION

- `int8` operations are significantly faster than floating-point for GPUs that provide acceleration support.
- But naïvely quantizing the weights/activations of a trained model leads to significant approximation errors.
- For example, if we evaluate OPT on a wide range of datasets: WinoGrande, HellaSwag, PIQA, and LAMBADA:
  - Interestingly, only larger models are strongly affected by quantization error.

Absolute maximum (absmax) quantization:

$$X_{\texttt{int8}} = \texttt{round}\left(\frac{127\, X_{\texttt{fp16}}}{\max |X_{\texttt{fp16}}|}\right)$$

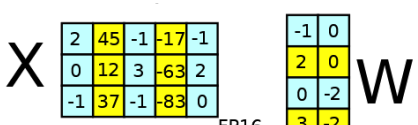


[Dettmers et al., 2022]

INT8 QUANTIZATION

- Dettmers et al., 2022, recognized that outliers are the issue.
- Larger models were more likely to contain weights/activations that have very large magnitude.
  - And these outliers were somehow important to the model's functioning.
- They proposed to separate the rows/columns of matrices that contain outliers.
  - Perform arithmetic operations on the outliers in `fp16`.
  - Perform arithmetic operations on the other values in `int8`.
  - Combine the outputs and convert the result back into `fp16`.

INT8 QUANTIZATION



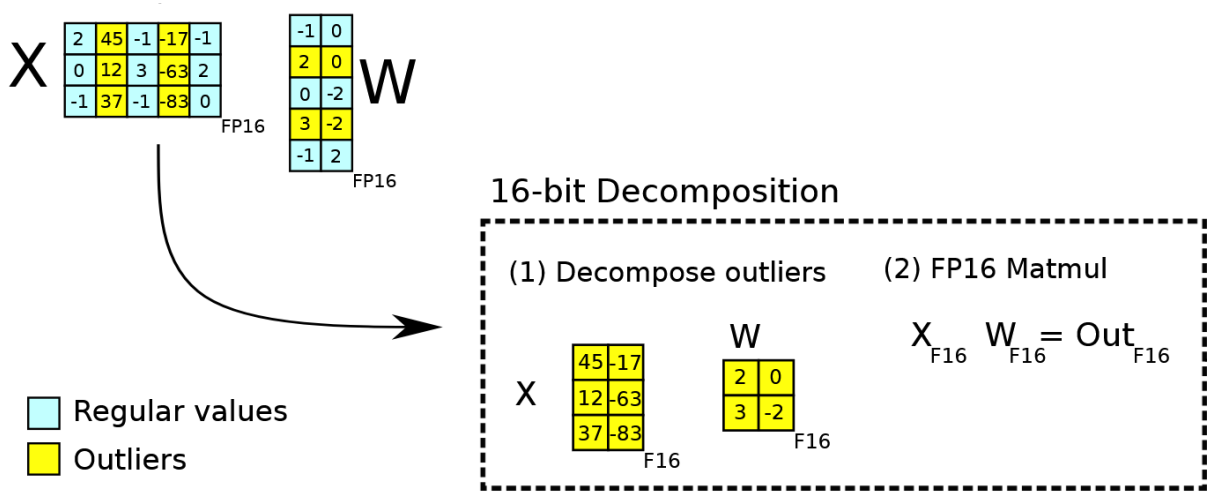
W

Regular values

Outliers

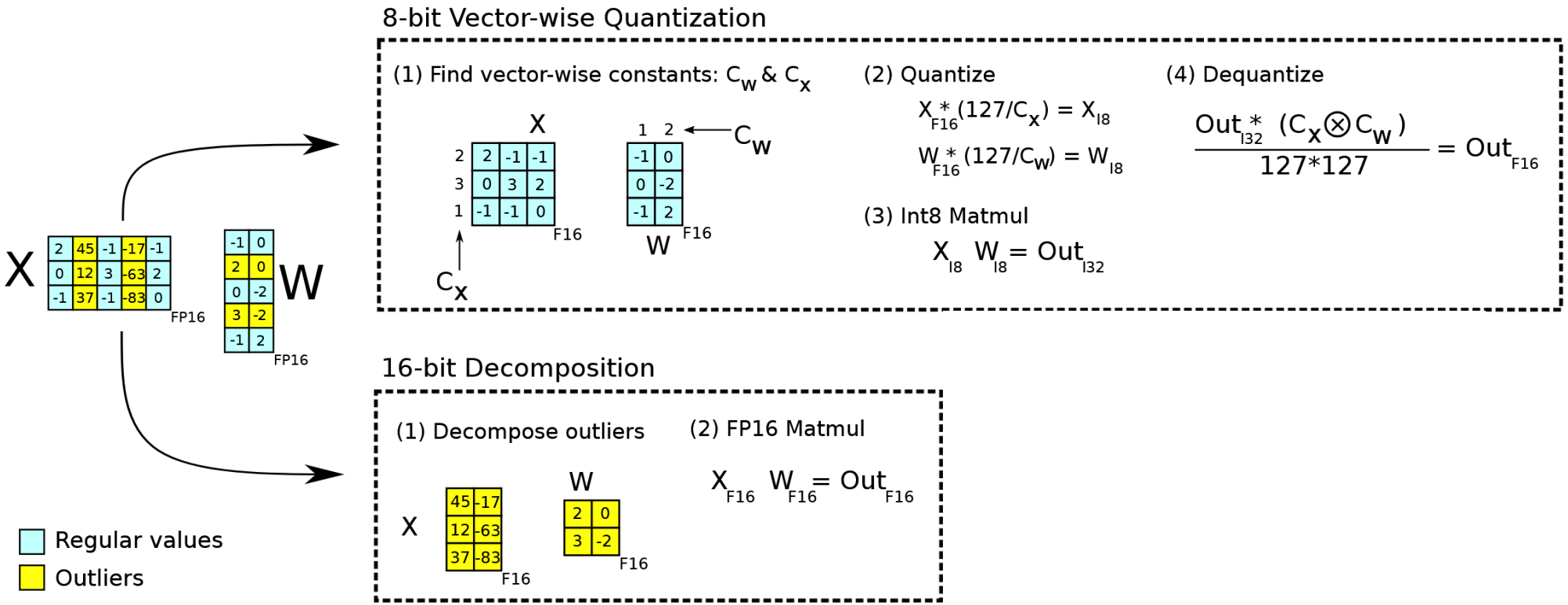
[Dettmers et al., 2022]

INT8 QUANTIZATION



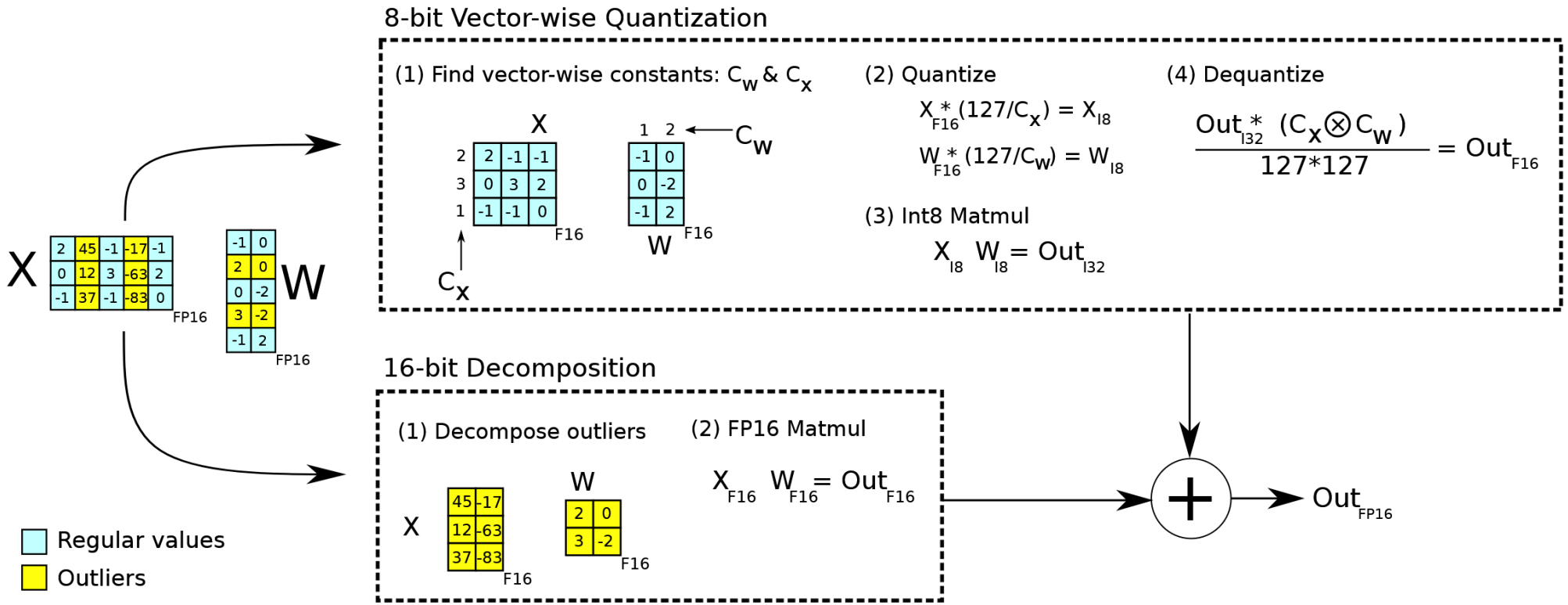
[Dettmers et al., 2022]

INT8 QUANTIZATION



[Dettmers et al., 2022]

INT8 QUANTIZATION



[Dettmers et al., 2022]

INT8 QUANTIZATION

- The conversion to and from `fp16`/`int8` requires additional overhead computation.
  - Models smaller than 6.7B parameters are slower due to this overhead.
- But this quantization approach enables running inference with 175B parameter models on 80GB of VRAM.
  - (there are single GPUs with this much memory)
  - Or you can use 8 consumer GPUs (GeForce RTX 3090).
  - Inference speed is doubled in comparison to `fp16` inference.

MORE QUANTIZATION

- More recent quantization methods are able to further reduce the precision to `int4`, `int3`, and `int2` (!).
- For example, SpQR (Sparse-Quantized Representation; Dettmers et al., 2023), not only considers outliers in the weight/activation matrices.
  - They more directly consider the quantization error on real inputs.
  - For each weight matrix W in the model,
  - They use a small set of calibration inputs X to measure a sensitivity parameter for each weight w_{ij} in W:

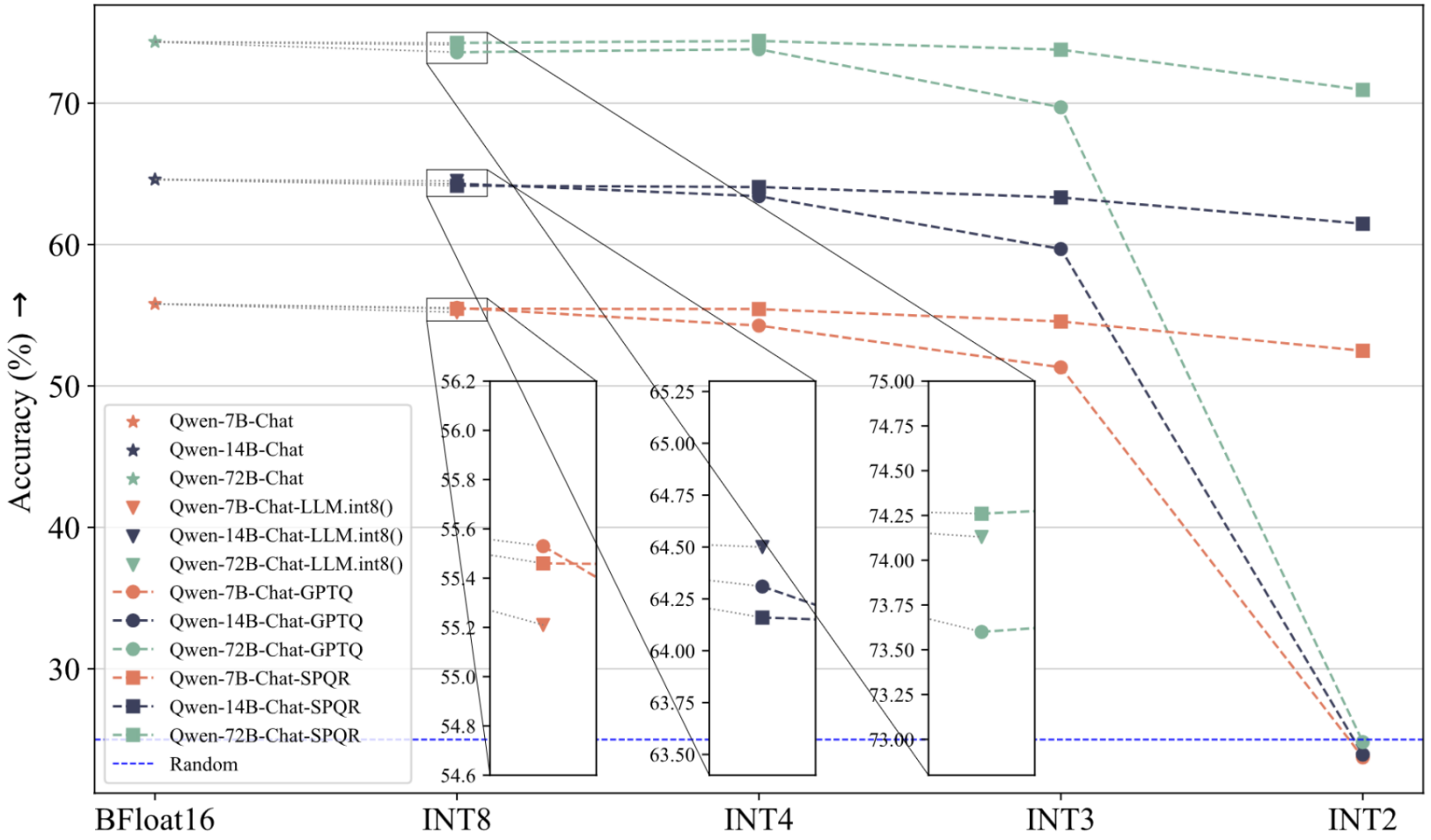
$$s_{ij} = \min_{W'} \|WX - W'X\|_2^2$$

  such that w_{ij}' = quantize(w_{ij}) and the other parameters of W' are unconstrained.

MORE QUANTIZATION

- They then specially consider weights in the matrix that are very sensitive to quantization (i.e., they have very high s_{ij}).
- They use higher precision to represent those weights, and lower precision for all other weights.
- With this approach, they are able to quantize models into `int3` and `int2` (binary).
- Jin et al., 2024, evaluated SpQR along with other quantization methods on a number of benchmarks.
  - MMLU (Massive Multitask Language Understanding; Hendrycks et al., 2021): Multiple-choice questions from broad set of categories.
  - C-EVAL (Huang et al., 2023): Multiple-choice examples from Chinese standardized exams.

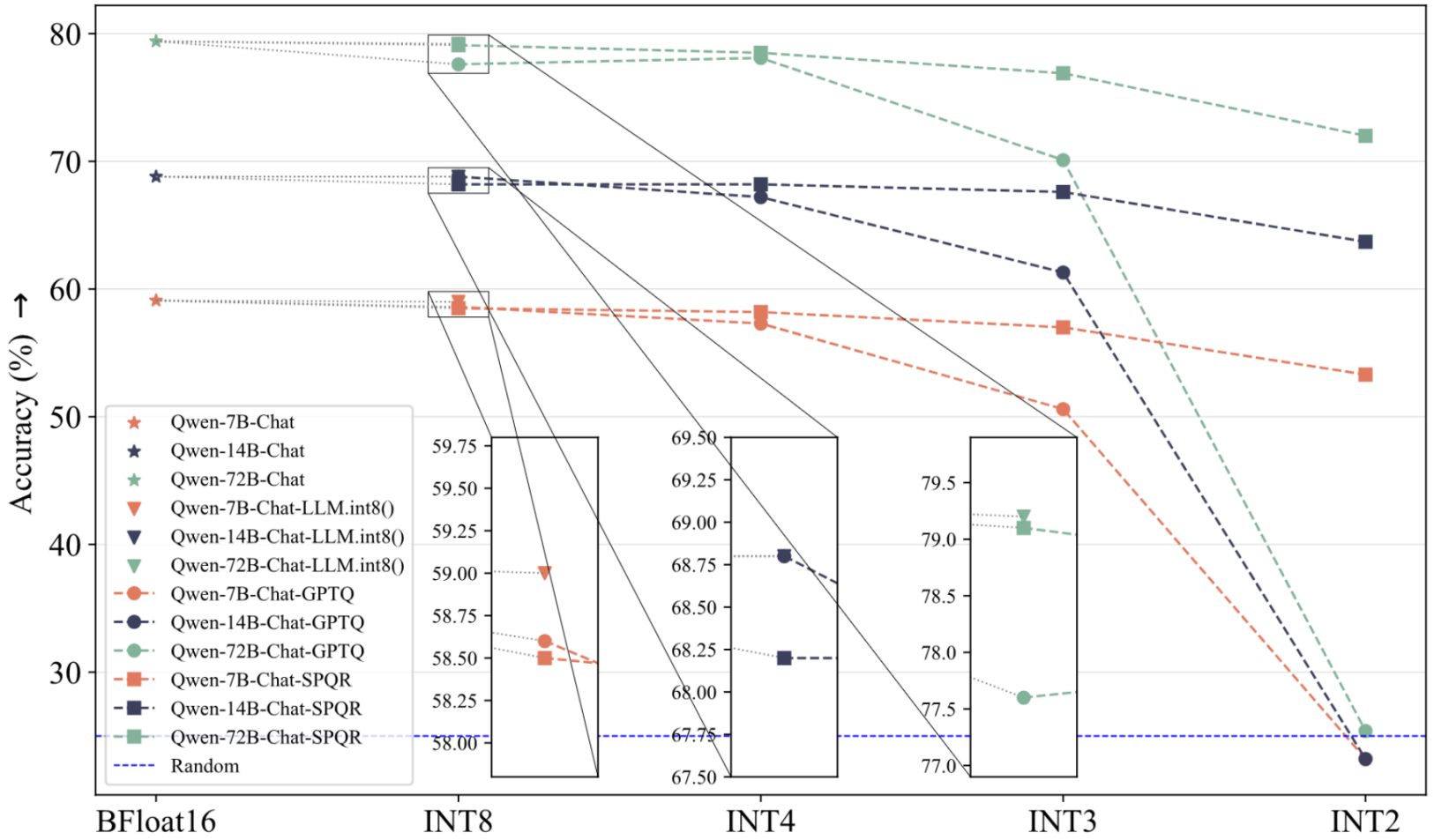
MORE QUANTIZATION



(a) MMLU

[Jin et al., 2024]

MORE QUANTIZATION



(b) C-EVAL

[Jin et al., 2024]

TWO BROAD APPROACHES TO QUANTIZATION

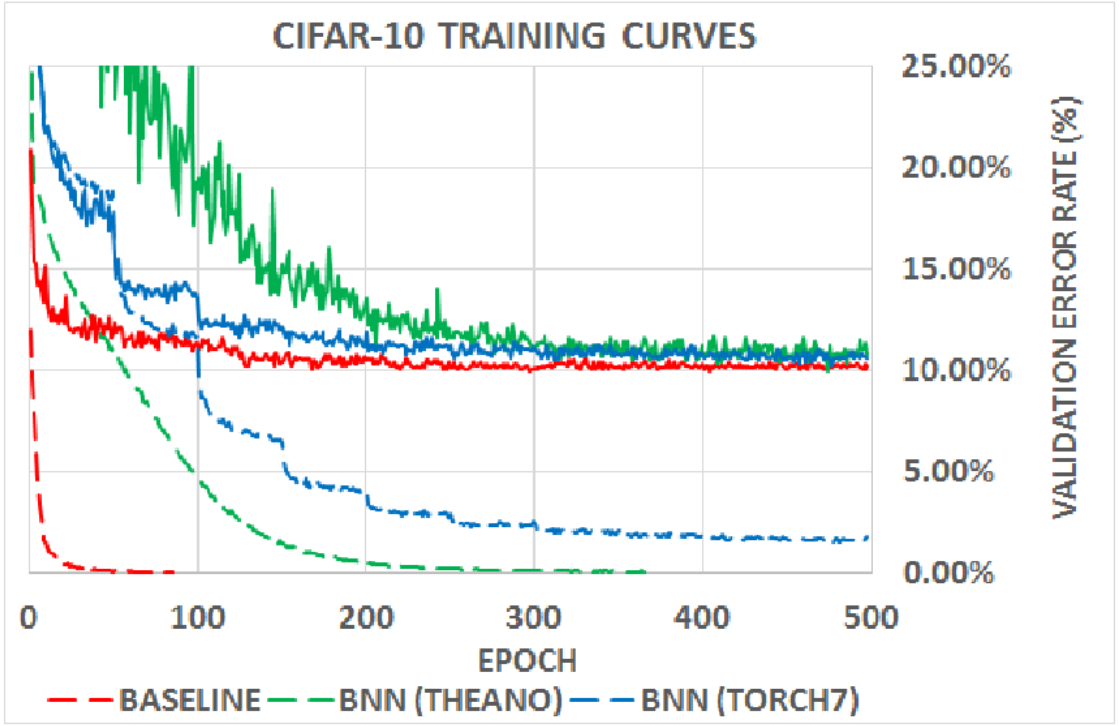
- The methods described thusfar involve training a model with high precision (or mixed precision) and then quantizing the model for inference afterwards.
- These approaches (including GPTQ and SpQR) are classified as post-training quantization (PTQ) methods.
  - Generally, while quantized models are evaluated on a handful of benchmarks,
  - Their abilities and generalization behavior, relative to the unquantized model, are currently not well understood.
- What if we modified the training procedure to make the model more robust to quantization?
- This other general approach is called quantization-aware training (QAT).

BINARIZED NEURAL NETWORKS

- One early approach to QAT is called binarized neural networks (Courbariaux et al., 2016).
  - All activations and weights are +1 or -1.
  - Backprop is also discretized.
  - They wrote a GPU kernel ("XNOR kernel") to optimize binary matrix multiplication.
- They tested their method by training a convolutional neural network on the CIFAR-10 dataset.

BINARIZED NEURAL NETWORKS

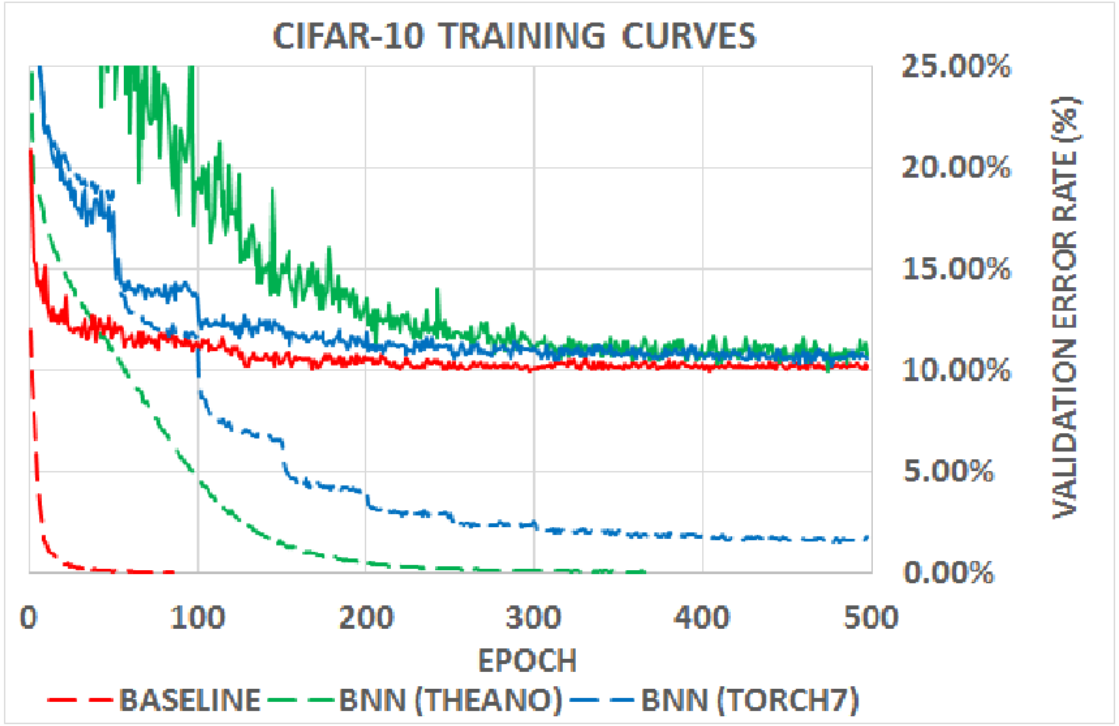
- The dashed lines show the training loss.
- The solid lines show the test error rate.



[Courbariaux et al., 2016]

BINARIZED NEURAL NETWORKS

- Training is considerably slower with binarized neural networks,
- But the final accuracy is not significantly lower than the baseline.



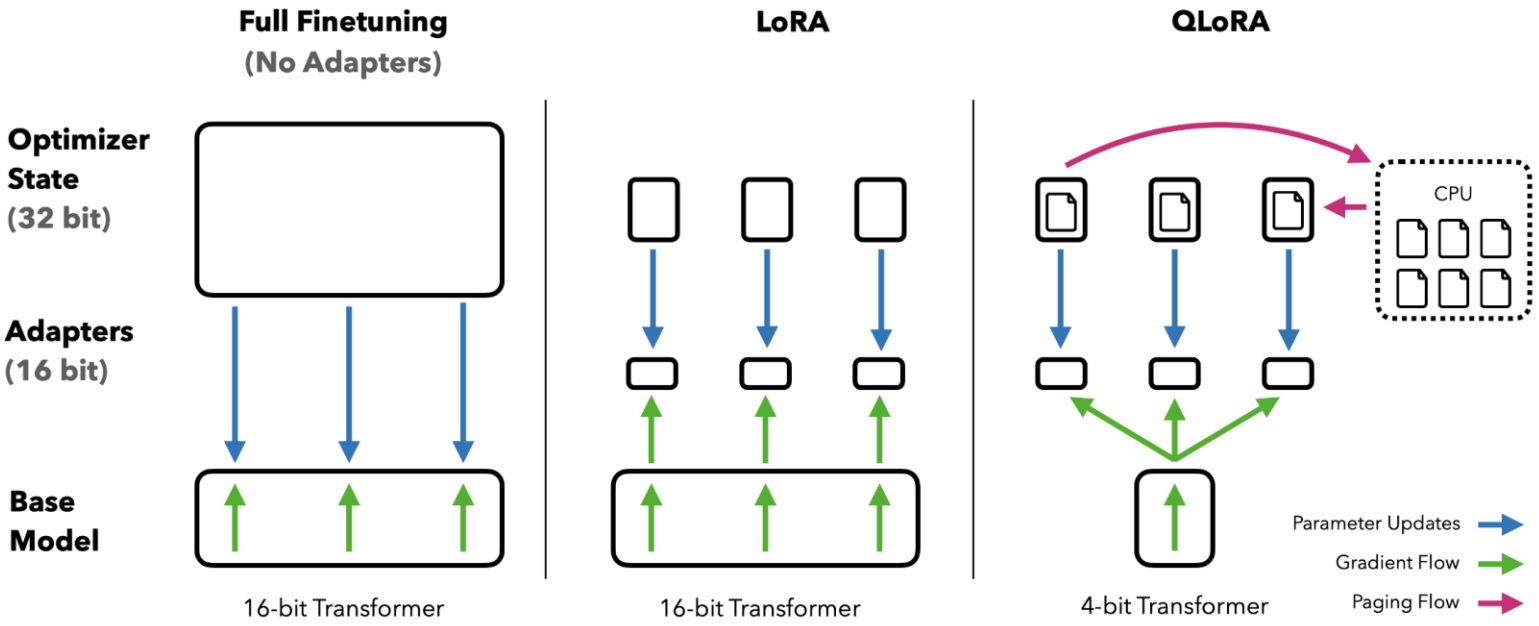
[Courbariaux et al., 2016]

QUANTIZATION-AWARE TRAINING

- Liu and Oğuz (2023) propose an approach to first pretrain an unquantized model.
  - Next, initialize a quantized model from the first model's weights.
  - Generate many input examples and perform forward passes on the unquantized models to obtain the logits for each example.
  - Fine-tune the quantized model to minimize the difference between its predicted logits and the logits from the unquantized model.
- QAT methods generally tend to be more computationally intensive.
  - Thus, PTQ methods are more widely used today.
  - But research is ongoing.

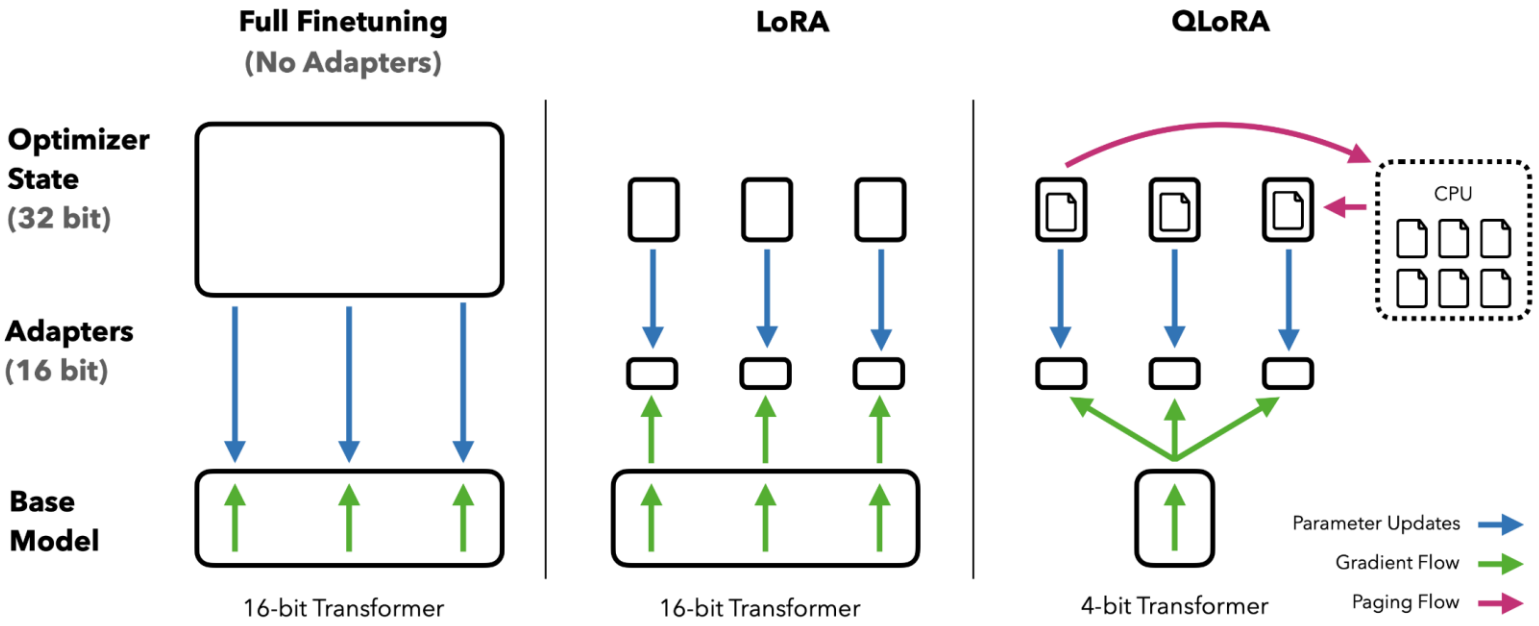
QLORA

- Quantization can be combined with parameter-efficient fine-tuning methods such as LORA.
- In QLoRA (Dettmers et al., 2023), the model parameters are quantized to 4 bits.



QLORA

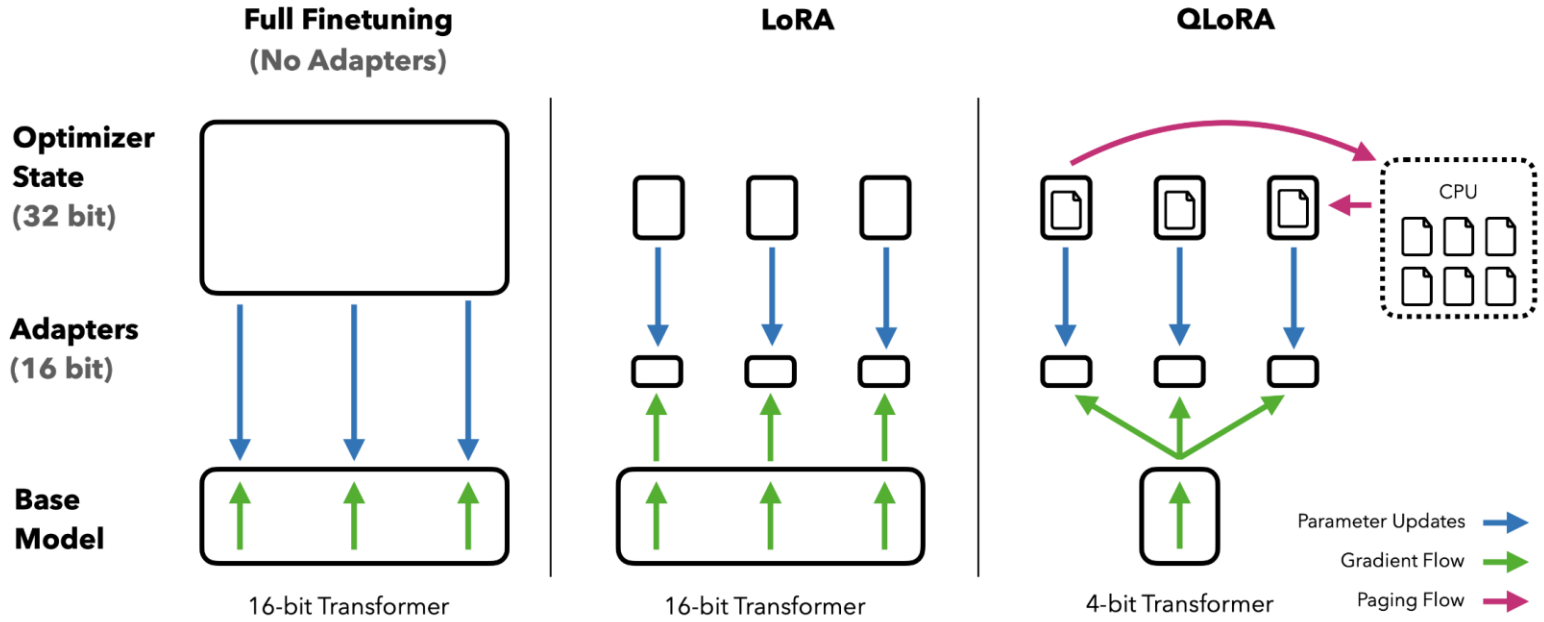
- They propose a new number format called `NormalFloat` which is specifically designed to store normally-distributed values.
- The adapters are stored using `bf16`.



[Dettmers et al., 2023]

QLORA

- They utilize “GPU memory paging” to move memory from GPU memory to system memory and back to more gracefully handle spikes in GPU memory usage.
- Enables fine-tuning a 65B parameter model on a single 48GB GPU.



[Dettmers et al., 2023]

QUANTIZATION SUMMARY

- We have discussed how to use quantization to speed up both inference and training of large models,
  - And to greatly reduce their memory footprints.
- For inference, we have looked at both post-training quantization (PTQ) and quantization-aware training (QAT) approaches.
- Next time, we will continue looking into other model compression techniques.
  - Distillation
    - Can we use a large model to teach a smaller model?
    - Is there an upper limit to the performance of the smaller model?
  - Pruning
    - Can we remove parts of the model and maintain accuracy?

QUESTIONS?