Half-Quadratic Quantization of Large Machine Learning Models

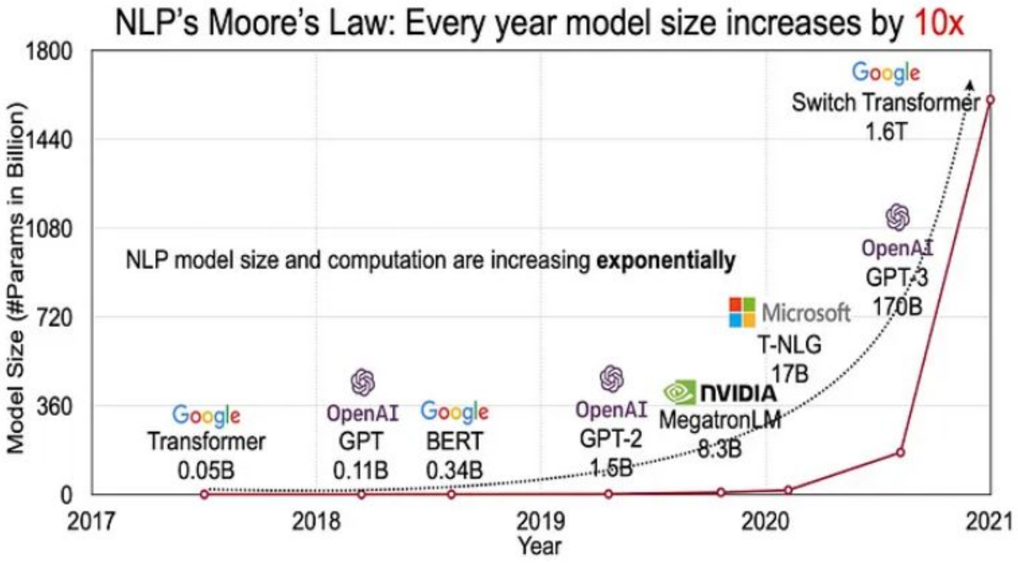
Dr. Hicham Badri

mobiuslabs

رياضيات المغرب
MATH MAROC

Introduction

- Machine learning models are getting very big!
- Running large models requires a lot of GPU memory:
  - Llama2-70B needs 140 GB of GPU memory, can't even fit in a single A100 80GB!
  - Training is even worse…



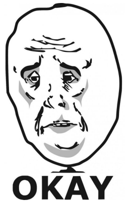


Image source: https://medium.com/@harishdatalab/unveiling-the-power-of-large-language-models-llms-e235c4eba8a9

What is Weight-Only Quantization?

- Weight-only quantization reduces the precision of the linear layer weights from floating-point to lower bits resulting in substantial GPU memory saving.
- Dequantization is performed on the fly for each layer at each forward pass:

out = matmul(x, W.t()) + bias —Quantized version→

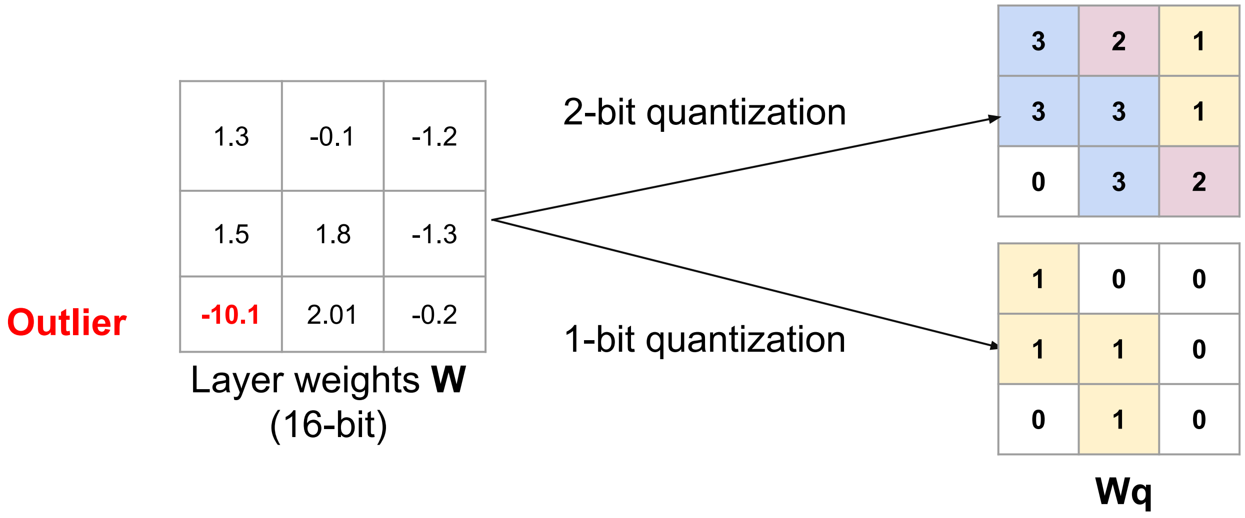
```
Wr = dequantize()
out = matmul(x, Wr.t()) + bias
del Wr #clear
```

- **Challenges:**
  - quality loss
  - processing time (both in the quantization and dequantization steps)

- Getting better quality quantization typically requires using *calibration data:*
  - Heavier processing time resulting in long quantization time.
  - Potential bias introduced by the calibration data.

The Outlier Problem

Outlier weights create problems with lower-bit quantization:



How Do Outliers Affect The Quantized Model Prediction?

**Model : ** Llama 2-7B Chat - 4-bit quantized
Prompt: How can I build a car?

```
Unterscheidung zwischen "Build a car" und "Buy a car"

Build a car: This phrase is used to indicate that you are going to construct a car from scratch, using
raw materials and tools. It requires a lot of time, effort, and expertise in mechanical engineering,
metalworking, and other related fields.

Buy a car: This phrase is used to indicate that you are going to purchase a pre-owned or new car from a
dealership or private seller. It is a more straightforward and less time-consuming process than
building a car from scratch.

In summary, building a car is a complex and time-consuming process that requires specialized knowledge
and skills, while buying a car is a more straightforward process that involves purchasing a pre-owned
or new car from a dealership or private seller.
```

Half-Quadratic Quantization

Linear Weight-Only Quantization Formulation

$$Q_{z,s}(W) = \text{round}(W/s + z) = W_q$$ **Quantize()**

$$Q_{z,s}^{-1}(W_q) = s(W_q - z) = W_r$$ **Dequantize()**

Parameters
z: zero-point
s: scaling

$$\underset{W_q}{\text{argmin}} \, ||Q_{z,s}^{-1}(W_q)X - WX||_2^2$$

Standard formulation uses the L2-norm which models **the activation quantization error** as a Gaussian noise.

X: Layer-wise calibration data

Popular formulation GPTQ, etc.

Note:
We normally work with a reshaped version:
W.view([group_size, -1])

$$\underset{z,s}{\text{argmin}} \, \phi\left(Q_{z,s}^{-1}(W_q) - W\right)$$

This formulation better reflects the distribution of the **outliers in the weights** by using a non-convex function like the L_p(<1) norm.

- No calibration data needed
- We only optimize for the quant parameters **z,s** not **Wq**
- Requires a solver

Proposed formulation (HQQ)

HQQ: Half-Quadratic Solution

Weight quantization residual

$$\underset{z,s}{\operatorname{argmin}}\, \phi(\boxed{W - Q_{z,s}^{-1}(Q_{z,s}(W))})$$

Introduce an intermediate variable ***We.***
We only optimize for ***z*** and ***We***, ***s*** is fixed.

$$\underset{z,W_e}{\operatorname{argmin}}\, \phi(W_e) + \frac{\beta}{2}||W_e - \boxed{(W - Q_z^{-1}(Q_z(W))}||_2^2$$

Half-Quadratic splitting results in sub-problems solved iteratively

$$(\mathrm{sp}_1) \quad W_e^{(t+1)} \leftarrow \underset{W_e}{\operatorname{argmin}}\, \phi(W_e) + \frac{\beta^{(t)}}{2}||W_e - \boxed{(W - Q_z^{-1}(Q_z(W))}||_2^2$$

$$(\mathrm{sp}_2) \quad z^{(t+1)} \leftarrow \underset{z}{\operatorname{argmin}}\, \frac{1}{2}||Q_z^{-1}(Q_z(W)) - (W - W_e^{(t+1)})||_2^2$$

$$\beta^{(t+1)} \leftarrow \kappa\beta^{(t)},$$

HQQ - Sub-Problems - SP1

Weight quantization residual

$$(\mathrm{sp}_1) \quad W_e^{(t+1)} \leftarrow \underset{W_e}{\operatorname{argmin}}\, \phi(W_e) + \frac{\beta^{(t)}}{2} ||W_e - (\boxed{W - Q_z^{-1}(Q_z(W))}||_2^2$$

This takes the form of a **Proximal Operator.**
A first-order closed-form solution is given via the *generalized thresholding operator* [Badri et al., 2016]

$$W_e^{(t+1)} \leftarrow \mathrm{shrink}_{l_p}\left(W - Q_z^{-1}(Q_z(W)), \beta\right)$$

$$\mathrm{shrink}_{l_p}(x, \beta) = \mathrm{sign}(x)\mathrm{relu}(|x| - \frac{|x|^{p-1}}{\beta})$$

This is basically shrinking the pointwise quantization error

HQQ - Sub-Problems - SP2

$$z^{(t+1)} \leftarrow \underset{z}{\operatorname{argmin}} \frac{1}{2}||z - \left(W_q^{(t+1)} - \frac{(W - W_e^{(t+1)})}{s}\right)||_2^2$$

$$W_q^{(t+1)} = \operatorname{round}(W/s + z^{(t)})$$

Re-estimate the zero-point by taking into account the shrinked quantization error.

SP2 has a closed-form solution!

$$z^{(t+1)} \leftarrow \langle W_q^{(t+1)} - \frac{(W - W_e^{(t+1)})}{s} \rangle$$

This is basically averaging the re-estimated zero-point along the group-size dimension

HQQ Solver Implementation

Putting it all together:

```
shrink_op = lambda x, beta,p=lp_norm: torch.sign(x)*torch.nn.functional.relu(torch.abs(x) - (1./beta)*torch.pow(torch.abs(x), p-1))

for i in range(iters):
    W_q    = torch.round(W_f*scale + zero).clamp(min_max[0], min_max[1])
    W_r    = (W_q - zero)/scale
    W_e    = shrink_op(W_f - W_r, beta)
    zero   = torch.mean(W_q - (W_f - W_e)*scale, axis=axis, keepdim=True)
    beta *= kappa
```

*

quantize()

dequantize()

Sp1

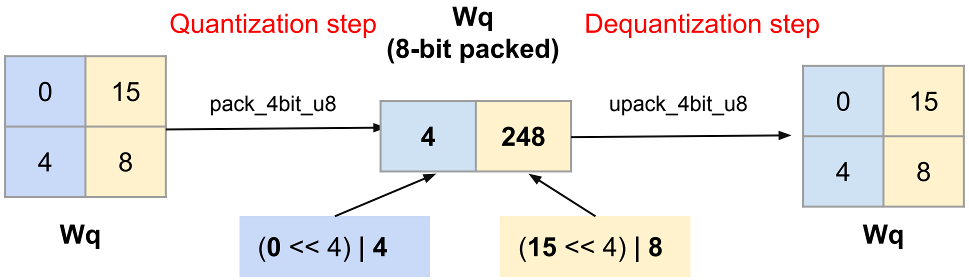
Sp2

This simple solution is over **100x** faster compared to Pytorch's Autograd (for p=1) !

* The scale is inverted in the implementation due to fp16 stability issues

Bitpacking

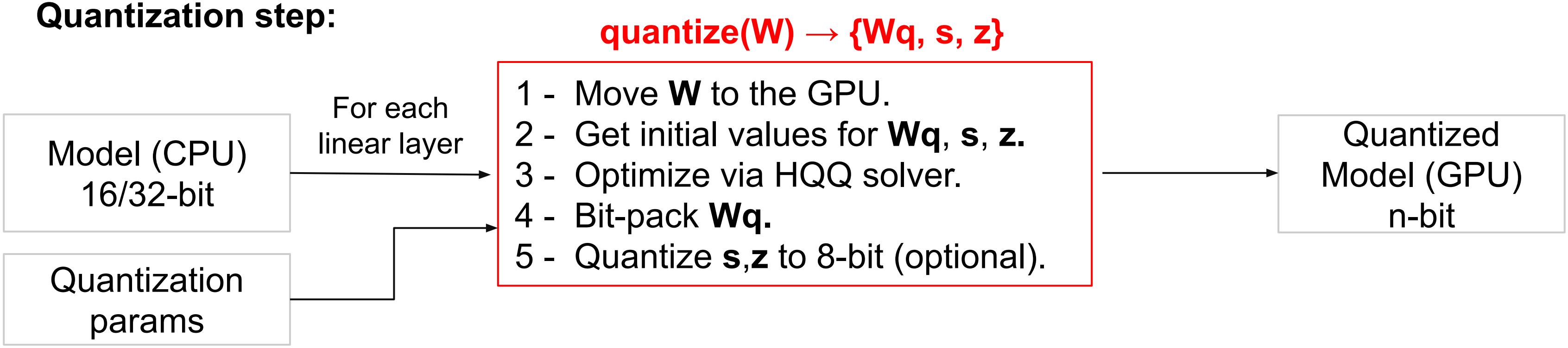
- There’s no native type (so far) for 2,3,4 bits in Pytorch. How do we actually store the n-bit quantized weights ?
  - **4-bit**: bitpack 2x4-bit in 1 uint8
  - **3-bit**: bitpack 10x3-bit in 1 int32
  - **2-bit**: bitpack 4x2-bit in 1 uint8



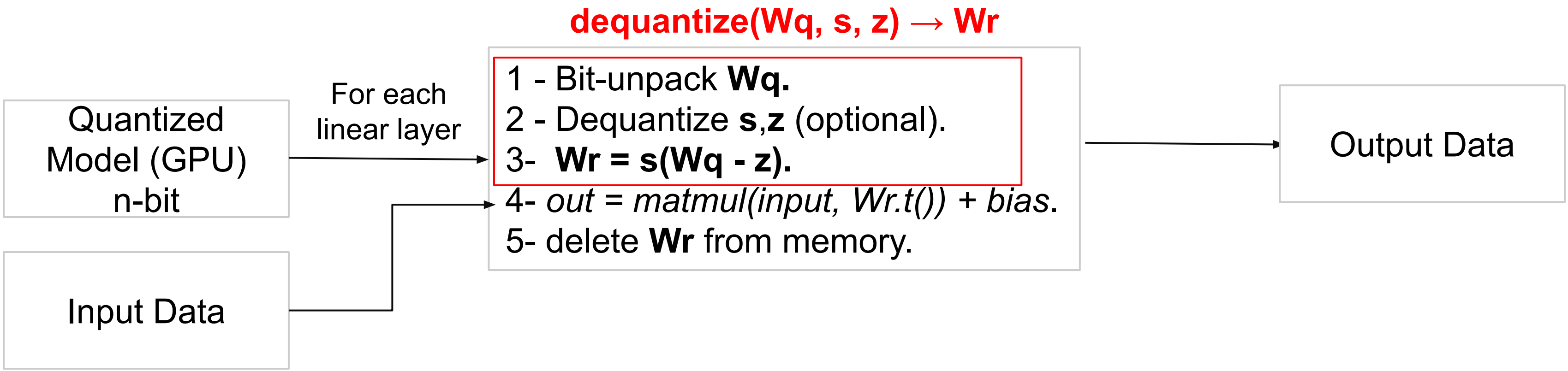

```
def pack_4bit_u8(W_q): #uint8 > uint8/2
        W_q = W_q.to(torch.uint8)
        _step = int(len(W_q)/2)
        return (W_q[:_step] << 4) | W_q[_step:]

def unpack_4bit_u8_cat(W_q): #uint8/2 > uint8
        return torch.cat([(W_q & 0b11110000) >> 4, W_q & 0b00001111], axis=0)
```

HQQ Summary

- **Quantization step:**



- **Inference step:**



HQQ + LoRA for Training

- HQQ + LoRA allows fine-tuning large models with frozen quantized weights.
- Rewrite the backward pass to support dequantization during LoRA training.

```python
class HQQMatmulNoCacheDeq(torch.autograd.Function):

        @staticmethod
        def forward(x, dequantize, bias):
                out = torch.matmul(x, dequantize().t())
                if(bias!=None): out += bias
                return out

        @staticmethod
        def setup_context(ctx, inputs, outputs):
                x, dequantize, bias = inputs
                ctx.save_for_backward(x, bias)
                ctx.dequantize = dequantize

        @staticmethod
        def backward(ctx, grad_output):
                x, bias    = ctx.saved_tensors
                dtype_out  = grad_output.dtype
                grad_input = grad_weight = grad_bias = None

                if ctx.needs_input_grad[0]:
                        grad_input = torch.matmul(grad_output, ctx.dequantize())

                if bias is not None and ctx.needs_input_grad[2]:
                        grad_bias = grad_output.sum(0)

                return grad_input, grad_weight, grad_bias
```

The dequantized weights should **not** be cached in the context. We pass the function instead

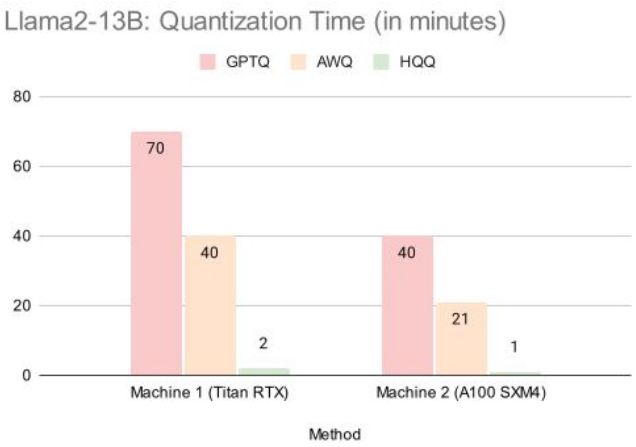
The secret to avoid

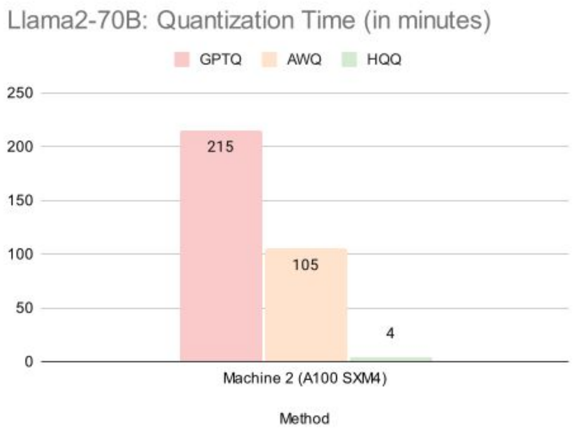
`CUDA out of memory.`

during training

Benchmarks

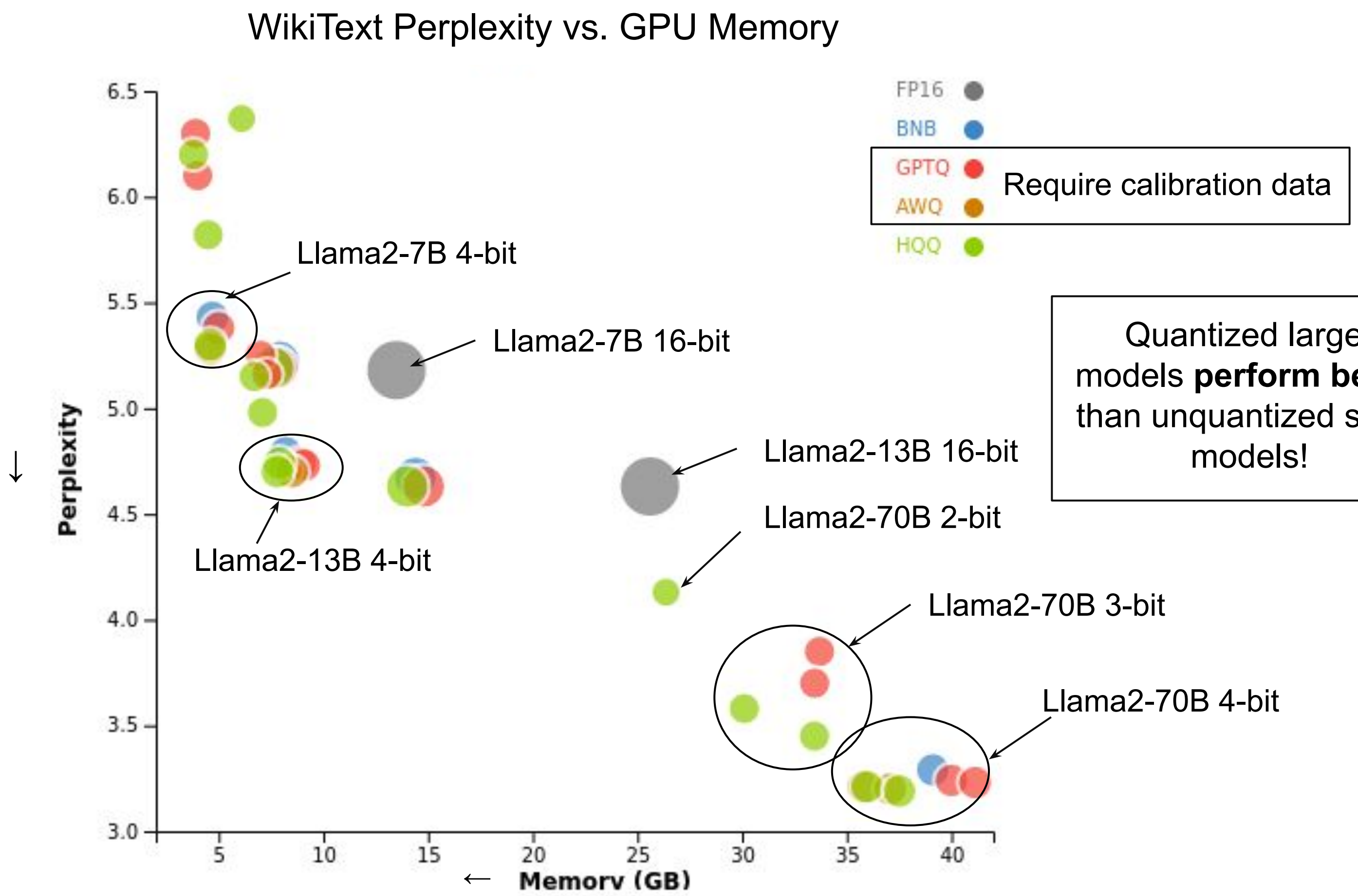
HQQ: Quantization Speed





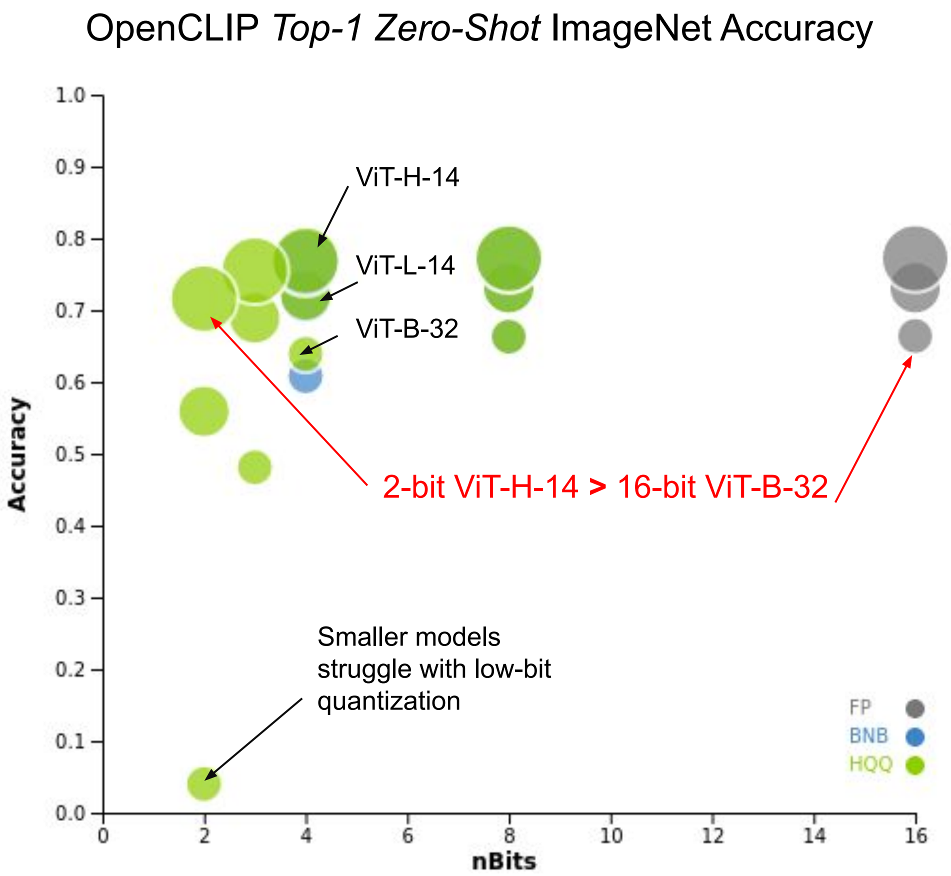
- **1 minute** for Llama2-7B (22-38x faster than GPTQ)
- **1 minute** for Llama2-13B (30-40x faster than GPTQ)
- **4 minutes** for Llama2-70B (54x faster than GPTQ)

HQQ: Llama2 Benchmark



Quantized larger models **perform better** than unquantized small models!

HQQ: How About Vision Models?



Quantized larger models **perform better** than unquantized small models for vision models as well!

HQQ - Adaptive Quantization

- Adaptive Quantization with HQQ allows setting different quantization settings to different layers.
- Mixture of experts models (MoE) can use lower bits for the experts and higher bits for the attention modules with minimal memory increase.

Running Mixtral 8x7B on a single 24GB GPU

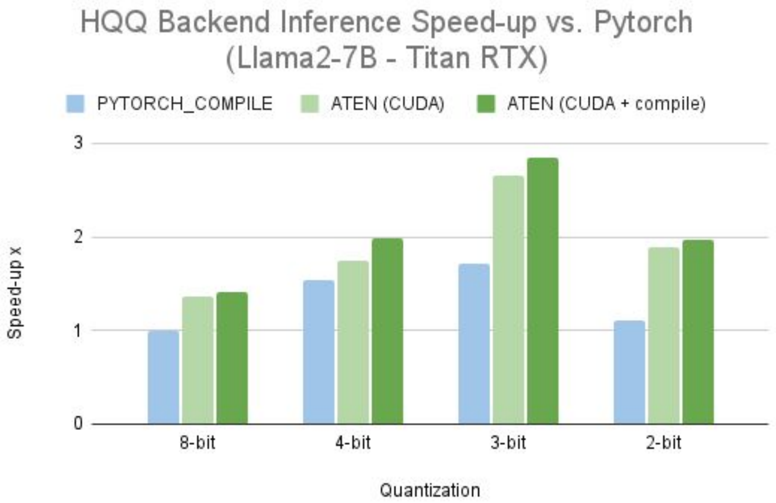
Method	**Perplexity**	**Runtime Memory**
HQQ 2-bit (attn + experts)	5.903	~20 GB
HQQ 4-bit (attn) HQQ 2-bit (experts)	4.686	~20 GB

Speeding-Up HQQ

- Dequantization speed:
  - Earlier HQQ version relied on *Torch Dynamo* to speed-up inference.
  - Torch compilation takes a long time and generated kernels are not cached.
  - Custom Triton kernels could make inference faster but some kernels might run slower on older gpus.
  - Custom CUDA dequantization kernels work quite well!

Llama2-7B forward time with different kernels (Titan RTX)

Backend	Time (sec)
PYTORCH_COMPILE	0.304
AutoAWQ - GEMV	0.499
AutoAWQ - GEMM	0.572
HQQLinearTritonSavable	1.987



HQQ+

- HQQ+ is an improved version of HQQ that leverages calibration data when available:
  - Uses LoRA for calibration on top of an HQQ quantized model.
  - Offloads the scale/zero-point to the CPU to achieve true n-bit quantization on the GPU.

Llama2-7B HQQ+ performance on Wikitext (with PYTORCH_COMPILE backend)

Method	Memory	Forward Time (sec / A100)	Perplexity
Quip# (2-bit)	2.72 GB	0.353	8.54
HQQ (2-bit_g16)	3.7 GB	0.155	7.31
HQQ+ (2-bit_g128)	2.90 GB	0.170	**7.29**
HQQ+ (2-bit_g64)	2.90 GB	0.180	**6.66**
HQQ+ (2-bit_g32)	2.90 GB	0.201	**6.10**
HQQ+ (2-bit_g16)	2.90 GB	0.248	**5.61**

HQQ - Resources

- **Blog**: https://mobiusml.github.io/hqq_blog/
- **Code** (Apache 2.0): https://github.com/mobiusml/hqq
- **Discussions**: https://github.com/mobiusml/hqq/issues
- **Quantized models**: https://huggingface.co/mobiuslabsgmbh
- **Oobabooga integration :** https://github.com/oobabooga/text-generation-webui
- **Mixtral-Offloading**: https://github.com/dvmazur/mixtral-offloading

Thank you for your attention!
Q&A