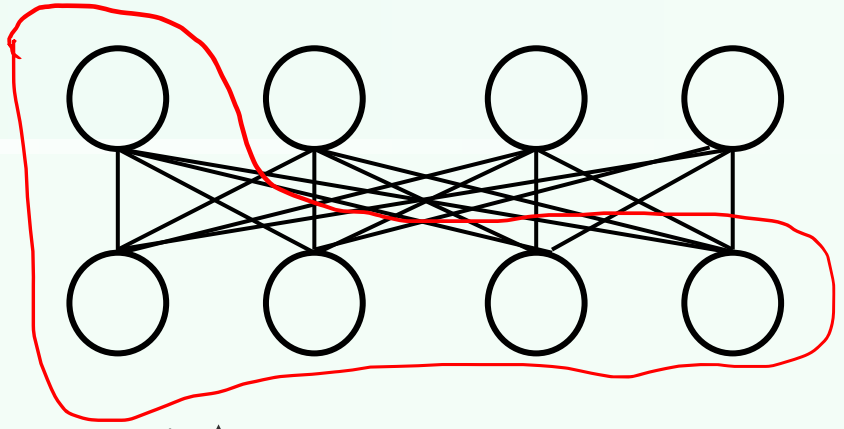
# 低精度训练

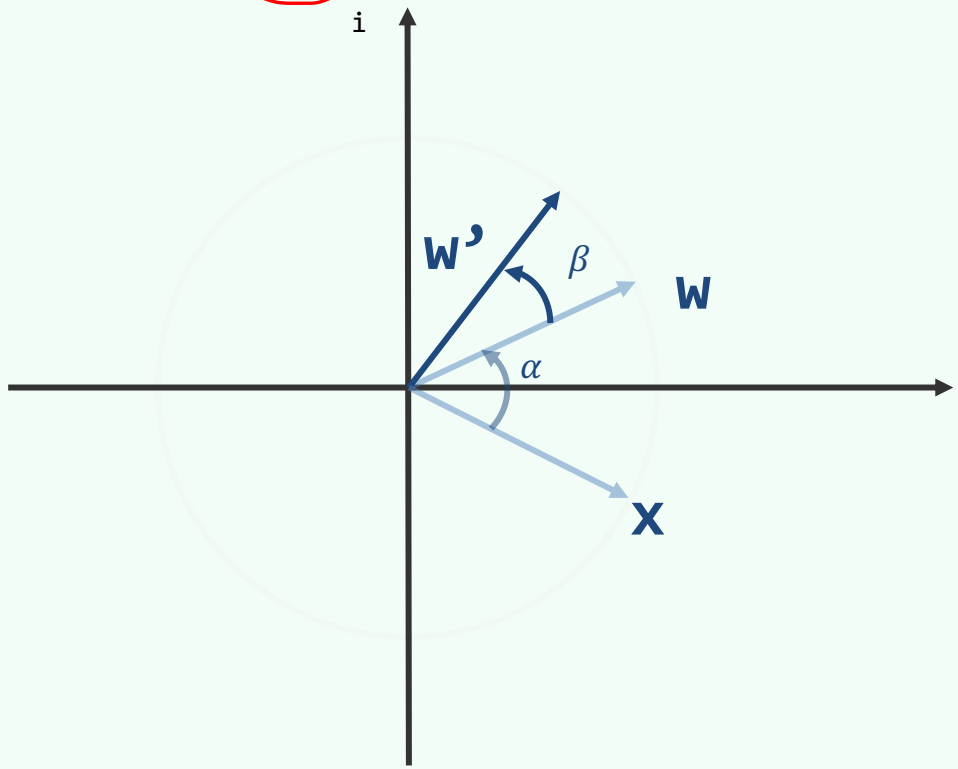
清华大学计算机系 陈键飞

《大模型计算》课程团队

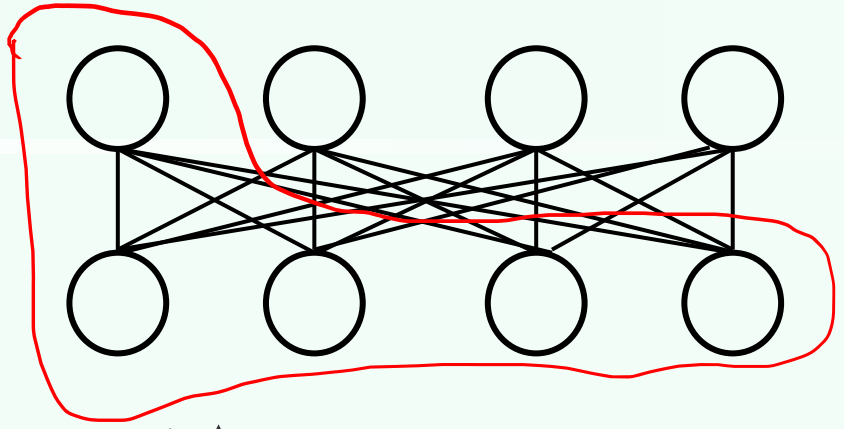
# 为什么神经网络对量化如此鲁棒?



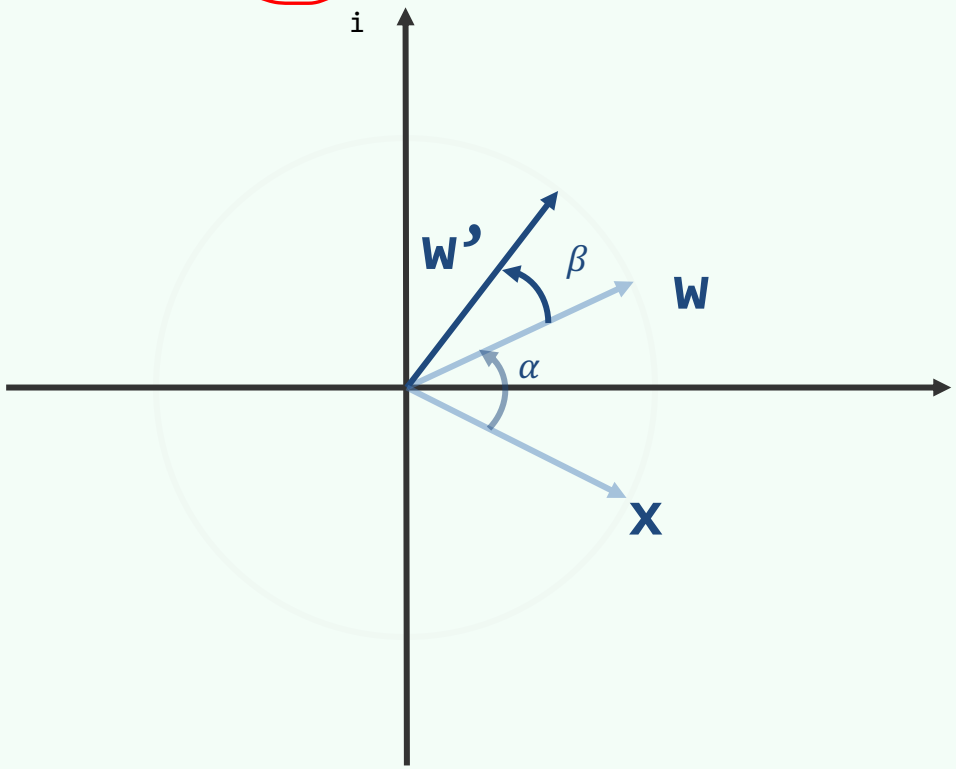
- 考虑某神经元w，不失一般性设$\|w\| = 1$
- 考察$< w, Q(w) >$
- 考察信噪比：信号：<w,w>=$\|w\|^2 =$
- 考察噪声<w,Q(w)-w>=$\sum_{i=1}^{N} w_i(Q(w_i) - w_i)$
- 注意到$Q(w_i) - w_i$是无偏的，Var[$Q(w_i) - w_i$]=O($s^2$)
- 由中心极限定理，Var[<w,Q(w)-w>] = $\sum_{i=1}^{N} w_i^2 s^2$
- Std[<w,Q(w)-w>] = s $\|w\|$
- 又注意到如果w是随机向量，那$\|w\|^2 = \sum_{i=1}^{N} s^2 = Ns^2$
- 所以信噪比= $\|w\|^2$/s $\|w\|$ = sqrt(N)
- 所以cossim(w, Q(w)) = 1 - 1/sqrt(N)



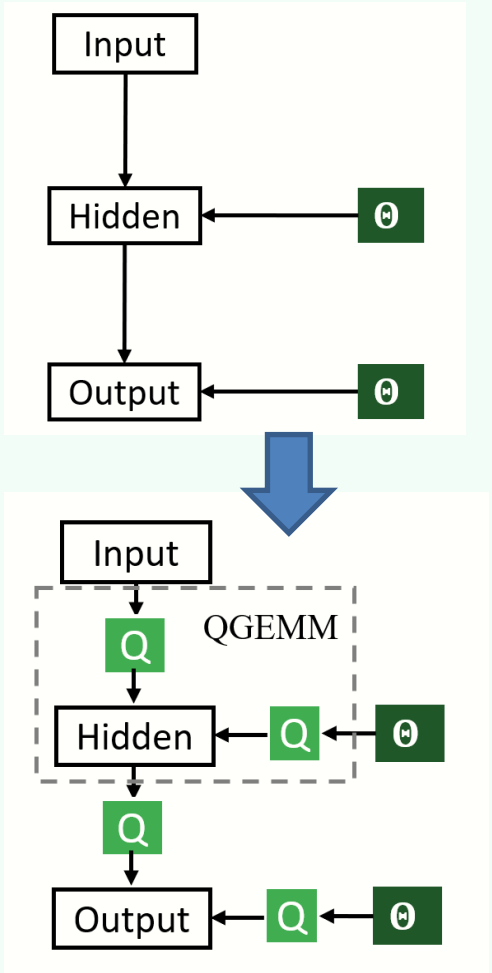
# 为什么神经网络对量化如此鲁棒?



- 那么 $\approx 1 - N^{-1/2}$ $\approx 2N^{-\frac{1}{4}}$
- $cos(\alpha + \beta) = cos(\alpha)cos(\beta) - sin(\alpha)sin(\beta)$
- 设N=10000
- 则$cos(\alpha + \beta) \approx cos(\alpha) \pm 0.01\cos(\alpha) \pm 0.1\sin(\alpha)$
- 在$\alpha$较小的时候近似精确



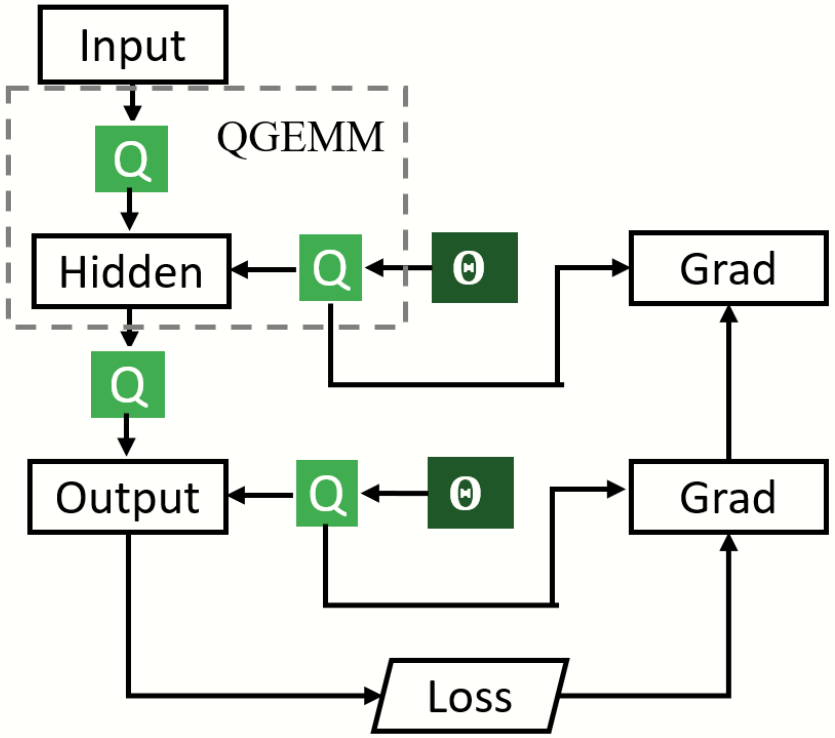
# 回顾：训练后量化不够准确（误差积累）




- Train-test gap
- 训练损失函数：全精度神经网络
- 测试损失函数：量化神经网络

- 思路：在训练时考虑低精度神经网络

# 量化感知训练




- 直接训练带有伪量化器的网络
- 回顾：反向传播
- 前向：$y = f(x_1, x_2, \ldots, x_N)$
- 反向：$dx_1, dx_2, \ldots, dx_N = \frac{\partial f}{\partial x_1} dy, \frac{\partial f}{\partial x_2} dy, \ldots, \frac{\partial f}{\partial x_N} dy$
- 此处：$x \in \mathbb{R}^D, y \in \mathbb{R}^C$
- $\frac{\partial f}{\partial x_1}$是$D \times C$维雅可比矩阵

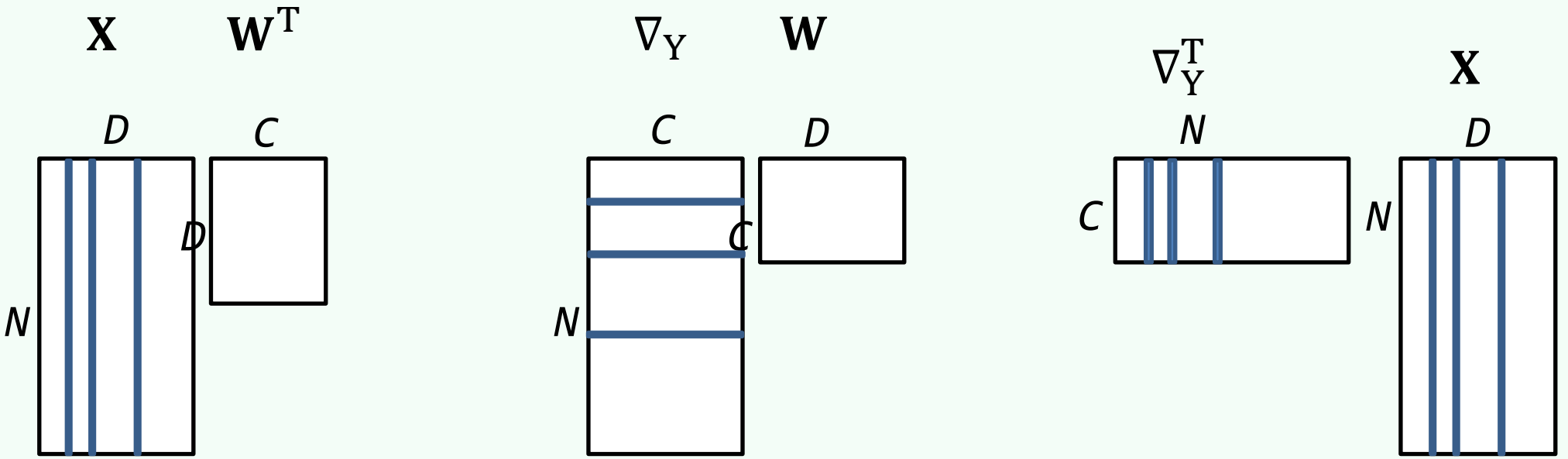
# 案例：量化线性层

**前向：** $W' = Q(W), X' = Q(X), Y = X'W'^T$

**反向：** $dX' = (dY)W', dW' = (dY^T)X'$

$$dX = dX' \circ \boldsymbol{Q'(X)},\ dW = dW' \circ \boldsymbol{Q'(W)}$$

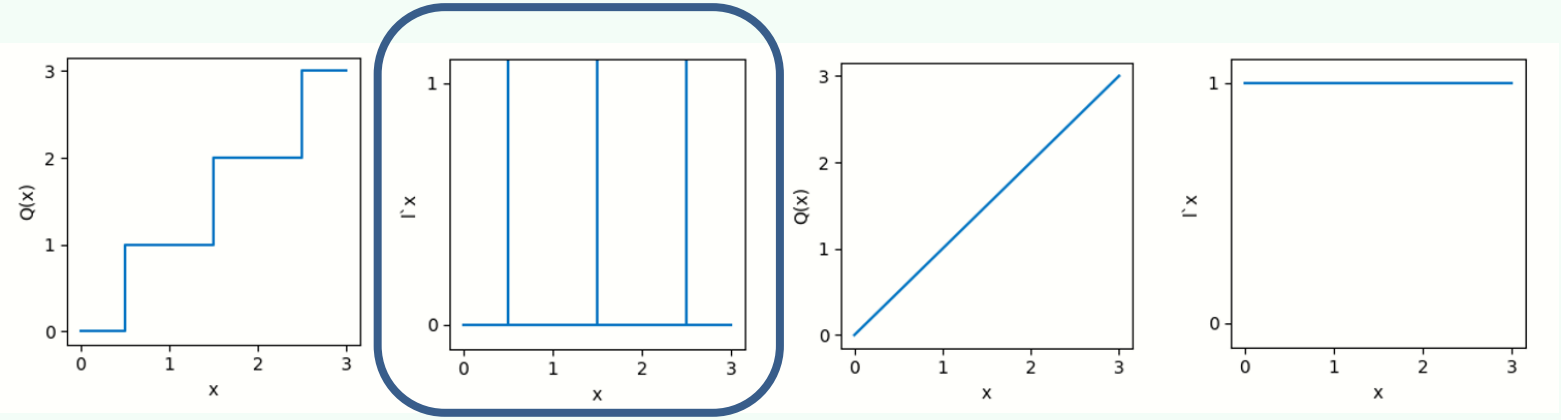
**问题：量化器的梯度是什么?**



# 量化器梯度

$$dX' \circ \boldsymbol{Q'(X)}$$

- $Q(X)$为阶梯函数
- 梯度为梳子函数（处处为0或∞），只有恰好卡在边沿的时候，会传递无穷大的梯度
- 样本效率很低




- 解决方案：直通估计器(straight-through estimator) $Q'(X) \approx 1,\ dX = dX'$
- 改进版：$X' = Q\big(\text{clamp}(X, min, max)\big), dX = dX' \circ \mathbb{I}(min \leq X \leq max)$
- 直觉1：当台阶数趋于无穷时，的确可以如此近似
- 直觉2：要让X’越大，X也需要越大

# Latent (master) weight and quantized weight

- 整个优化过程中，有两份模型权重
- Latent weight为FP32
- Quantized weight为低精度INT8/INT4/FP4…

- 前向传播：latent weight量化为quantized weight → 计算forward
- 反向传播：计算quantized weight的梯度 → straight-through到latent weight，更新权重

- Latent weight有记忆，能不断积累更新趋向，最终量变形成质变

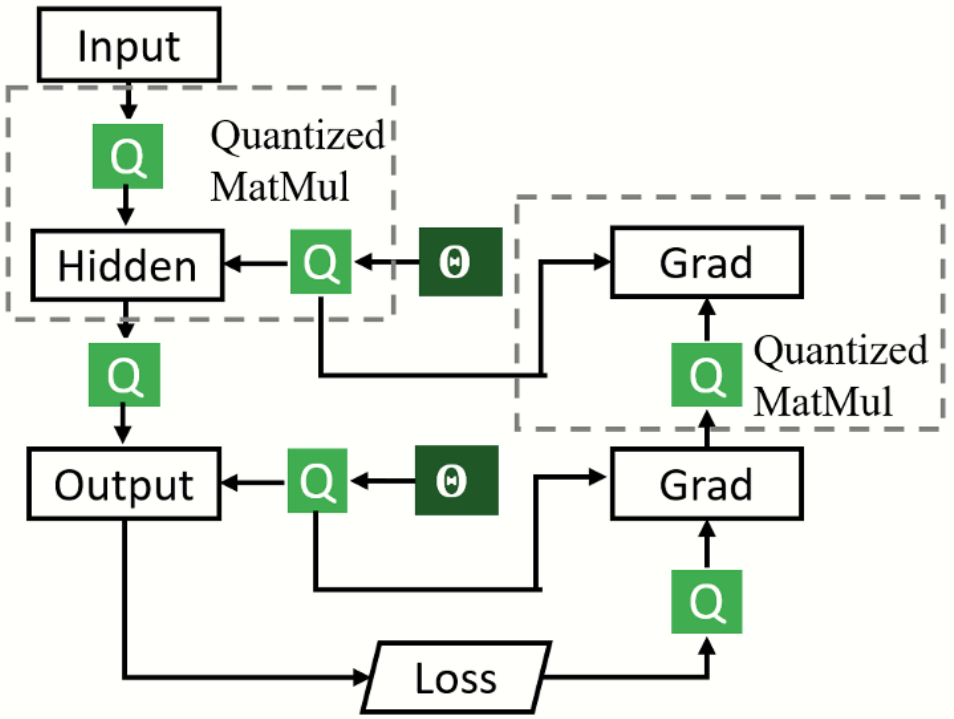
# 其他考虑

- 灾难遗忘
  - 在训练过程中，模型会逐渐记住新数据，忘记旧数据
- 训练数据集?
  - 常见情况：量化感知训练数据集弱于原始（闭源）数据集
  - 理想工作流：预训练-量化感知训练-（量化感知）指令微调-（量化感知）RL
  - 实际往往不得不：预训练-指令微调-RL-量化感知训练
- 从头开始 or 从收敛的全精度模型开始?
- 知识蒸馏?

# 样例

- BitNet b1.58: 仅权重3值量化
- NVFP4 QAT: 权重+激活 FP4量化

# 训练加速：全量化训练/低精度训练/混合精度训练/FP8训练……



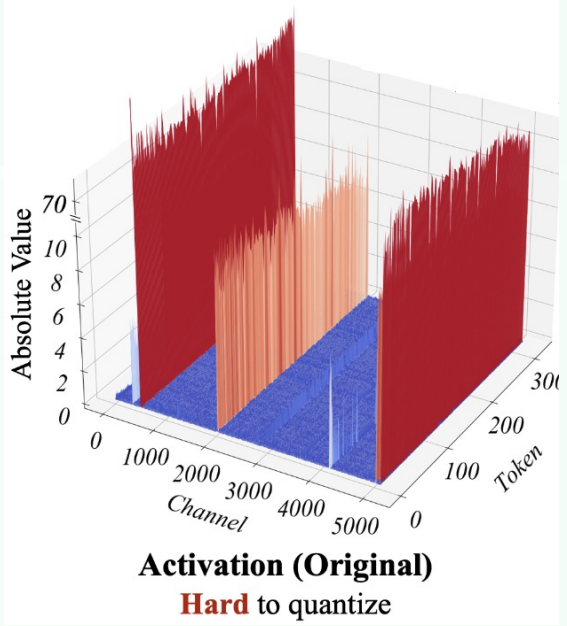

3 Matrix Multiplications:

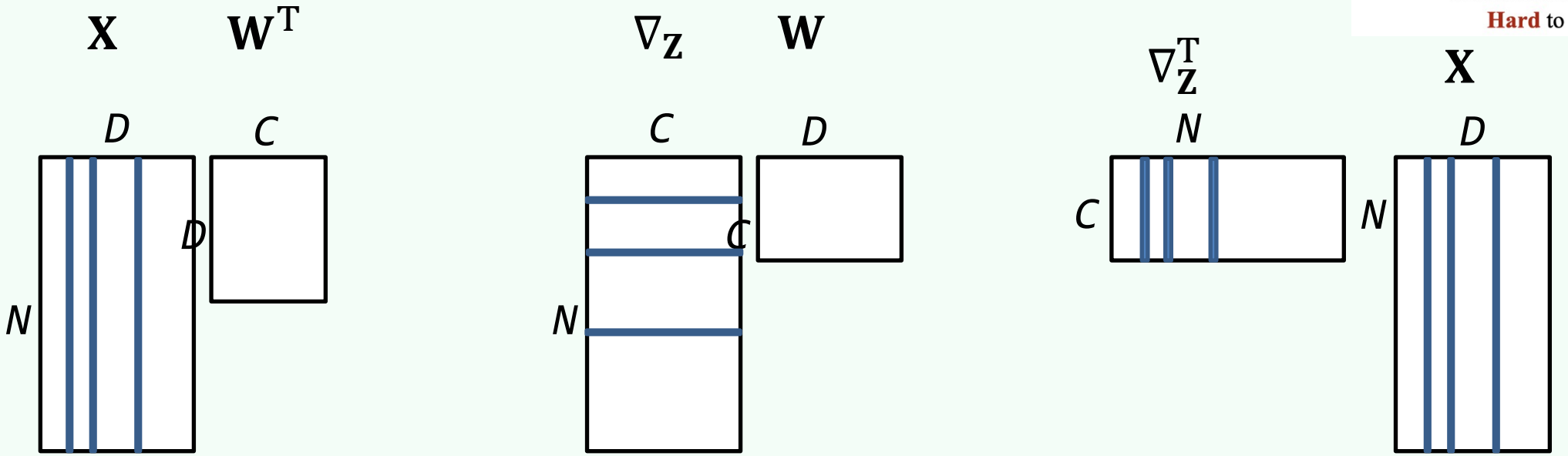
$$\mathbf{Z} = \mathbf{X}\mathbf{W}^\top, \quad \nabla_\mathbf{X} = \nabla_\mathbf{Z}\mathbf{W}, \quad \nabla_\mathbf{W} = \nabla_\mathbf{Z}^\top\mathbf{X},$$

| | | |
|---|---|---|
| X: | #token * #channel | input |
| W: | #channel * #channel | weight |
| Z: | #token * #channel | output |
| $\nabla_X$: | #token * #channel | act. grad. |
| $\nabla_W$: | #channel * #channel | wt. grad. |

# 线性层：离群值模式

$$\mathbf{Z} = \mathbf{X}\mathbf{W}^\top, \quad \nabla_\mathbf{X} = \nabla_\mathbf{Z}\mathbf{W}, \quad \nabla_\mathbf{W} = \nabla_\mathbf{Z}^\top\mathbf{X},$$



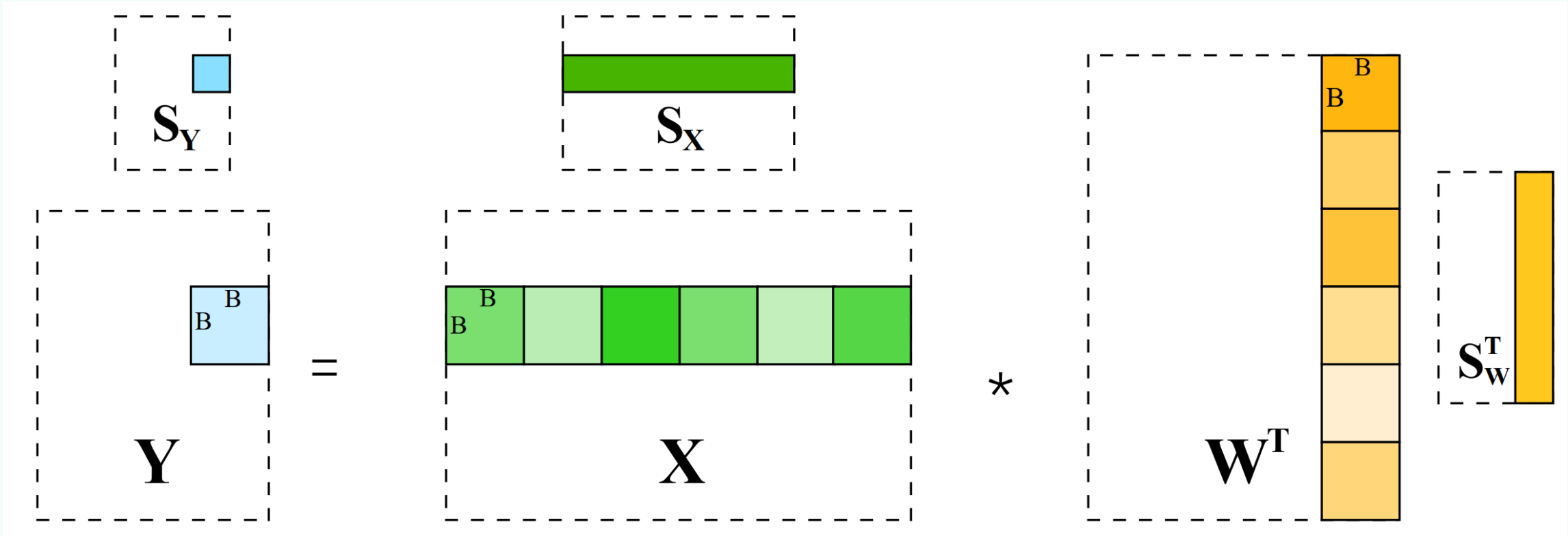

N: #tokens, D: input dimensionality, C: output dimensionality

# 难点

- 从头开始训练（training from scratch）：不能卡在局部最优
- 梯度也需要量化
- 低精度矩阵乘法收益难以掩盖量化开销

# 案例：DeepSeek FP8

- 激活：1*128
- 权重：128*128
- 梯度：1*128（？）



# 随机舍入

- 每个值均为整数，但它们的期望可以是小数
- $p(x=1)=0.8, p(x=0)=0.2, E[x]=0.8$
- 随机舍入：

**Quantization** To quantize a high-precision (e.g., `BF16`) matrix to `NVFP4`, we need to compute the 8-bit scaling factor $S$ for each group and 4-bit `E2M1` value $P_i$ for each group element $X_i$. For any group of high-precision values $\{X_i\}_{i=0}^{15}$, we map each $X_i$ to a `FP4` values as follows, where $\text{round}_{\text{FP4}}$ can be deterministic or stochastic.

$$P_i = \text{round}_{\text{FP4}}\left(X_i/S\right), \quad X_i \approx P_i \cdot S$$

We adopt *Deterministic Rounding* (Round-To-Nearest, RTN) in forward to minimize quantization error:

$$\text{round}_{\text{FP4},D}(x) = \arg\min_{q \in \texttt{FP4Values}}\{|x-q|\}$$

We use *Stochastic Rounding* [29] in backward to ensure unbiased estimation of gradients. For any $-6 \le x \le 6$ we can find two consecutive `FP4` values $q_1, q_2 \in \texttt{FP4Values}$, such that $q_1 \le x < q_2$. We round $x$ to either $q_1$ or $q_2$ with probability:

$$\text{round}_{\text{FP4},S}(x) = \begin{cases} q_1, & x+\xi \le \frac{q_1+q_2}{2} \\ q_2, & \text{otherwise} \end{cases}, \quad \xi \sim \text{Uniform}\left(-\frac{q_2-q_1}{2}, \frac{q_2-q_1}{2}\right)$$

Stochastic rounding is unbiased: $\mathbb{E}_\xi[\text{round}_{\text{FP4},S}(x)] = x$.

# Theoretical Results

SGD: $\boldsymbol{\Theta}_t \leftarrow \boldsymbol{\Theta}_{t-1} - \eta \hat{\nabla}_{\boldsymbol{\Theta}}$

**Theorem 1 (unbiased gradient)**

FQT gradient is an **unbiased** gradient estimator of the QAT gradient.

$$\mathbf{E}\left[\hat{\nabla}_{\boldsymbol{\Theta}^{(l)}}\right] = \nabla_{\boldsymbol{\Theta}^{(l)}}$$

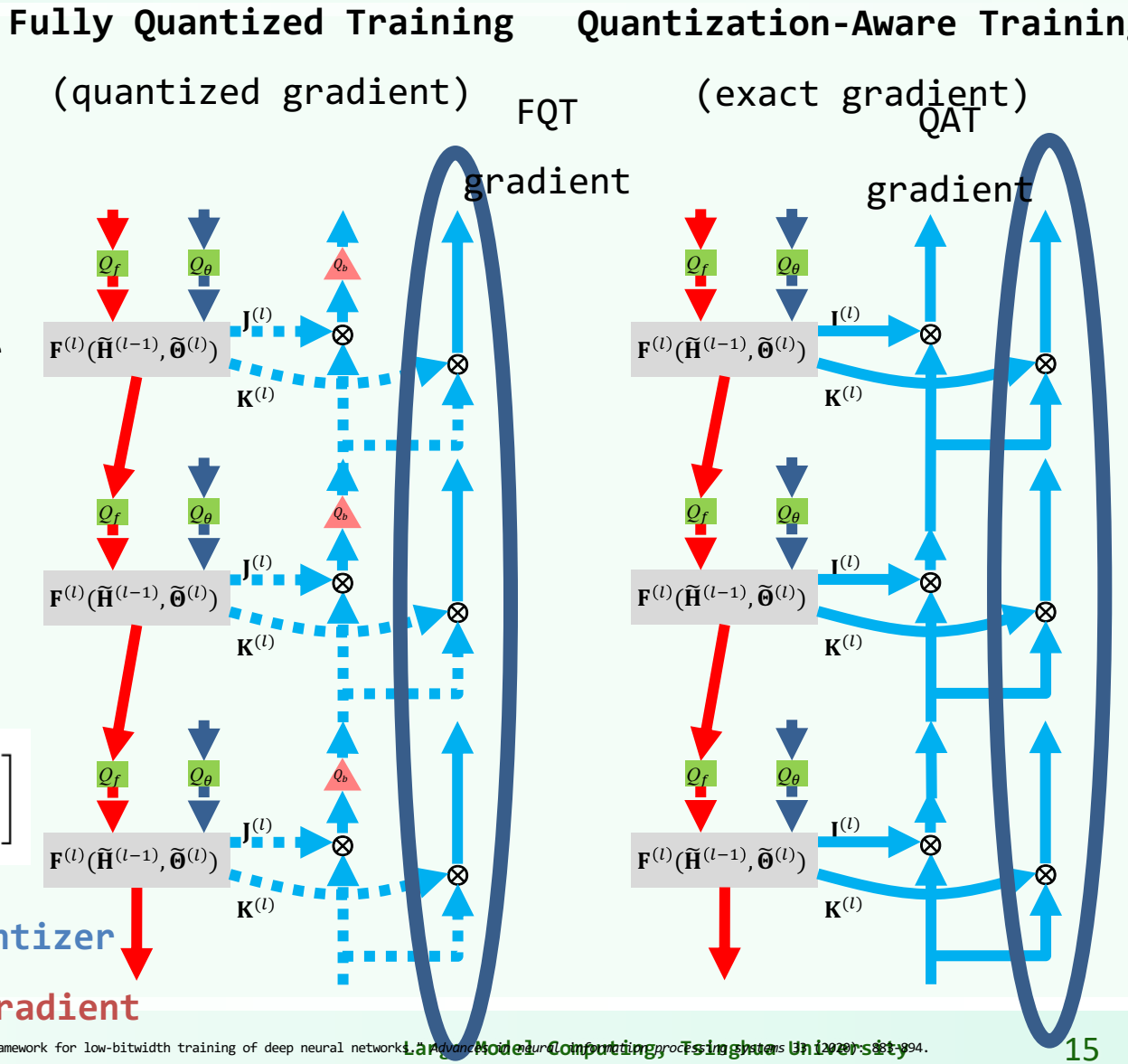
**FQT converges to a stationary point of QAT**

**Theorem 2 (gradient variance)**

$$\mathrm{Var}\left[\hat{\nabla}_{\boldsymbol{\Theta}}\right] = \mathrm{Var}\left[\nabla_{\boldsymbol{\Theta}}\right] + \sum_{l=1}^{L} \mathbb{E}\left[\sum_{k=1}^{l} \mathrm{Var}\left[\mathrm{vec}(Q_b(\hat{\nabla}_{\mathbf{H}^{(l)}}))\boldsymbol{\gamma}^{(k,l)} \,\middle|\, \hat{\nabla}_{\mathbf{H}^{(l)}}\right]\right]$$

Minibatch sampling

Impact of the ***l*-th layer quantizer** to the ***k*-th layer parameter gradient**




Chen, Jianfei, et al. "A statistical framework for low-bitwidth training of deep neural networks." *Advances in Neural Information Processing Systems* 33 (2020): 883-894.

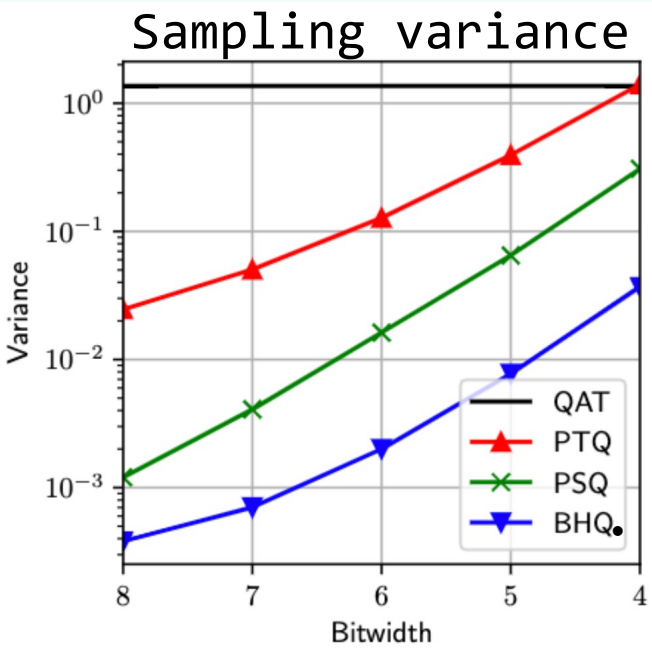
# Convergence Behavior

**Theorem 2 (gradient variance)**

$$\mathrm{Var}\left[\hat{\nabla}_{\boldsymbol{\Theta}}\right] = \mathrm{Var}\left[\nabla_{\boldsymbol{\Theta}}\right] + \sum_{l=1}^{L}\mathbb{E}\left[\sum_{k=1}^{l}\mathrm{Var}\left[\mathrm{vec}(Q_b(\hat{\nabla}_{\mathbf{H}^{(l)}}))\boldsymbol{\gamma}^{(k,l)} \,\middle|\, \hat{\nabla}_{\mathbf{H}^{(l)}}\right]\right]$$

Minibatch sampling

Impact of the *l*-th layer quantizer to the *k*-th layer parameter gradient $= O\left(\frac{1}{B^2}\right)$ where $B=2^b-1$ is the number of quant. bins




If sampling variance dominates

- Less bits for free

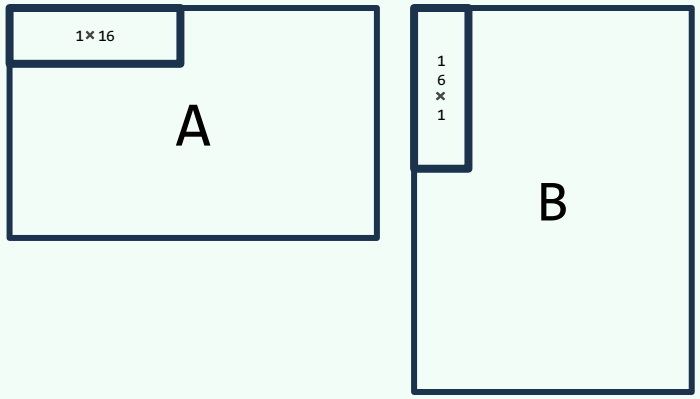
Otherwise

1 less bit ≈ 4x larger variance ≈ 4x slower convergence

Chen, Jianfei, et al. "A statistical framework for low-bitwidth training of deep neural networks." *Advances in Neural Information Processing Systems* 33 (2020): 884-894.

# 问题：量化块朝向

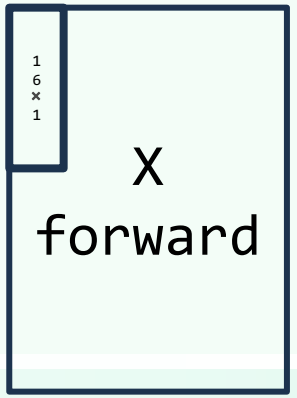
$$\mathbf{C} = \mathbf{AB}$$

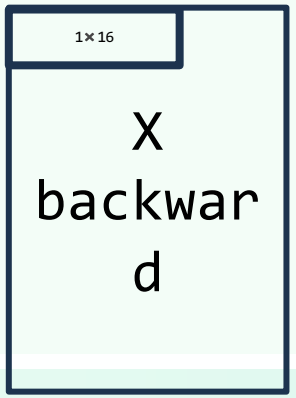



$$\mathbf{Z} = \mathbf{X}\mathbf{W}^\top, \quad \nabla_{\mathbf{X}} = \nabla_{\mathbf{Z}}\mathbf{W}, \quad \nabla_{\mathbf{W}} = \nabla_{\mathbf{Z}}^\top\mathbf{X},$$

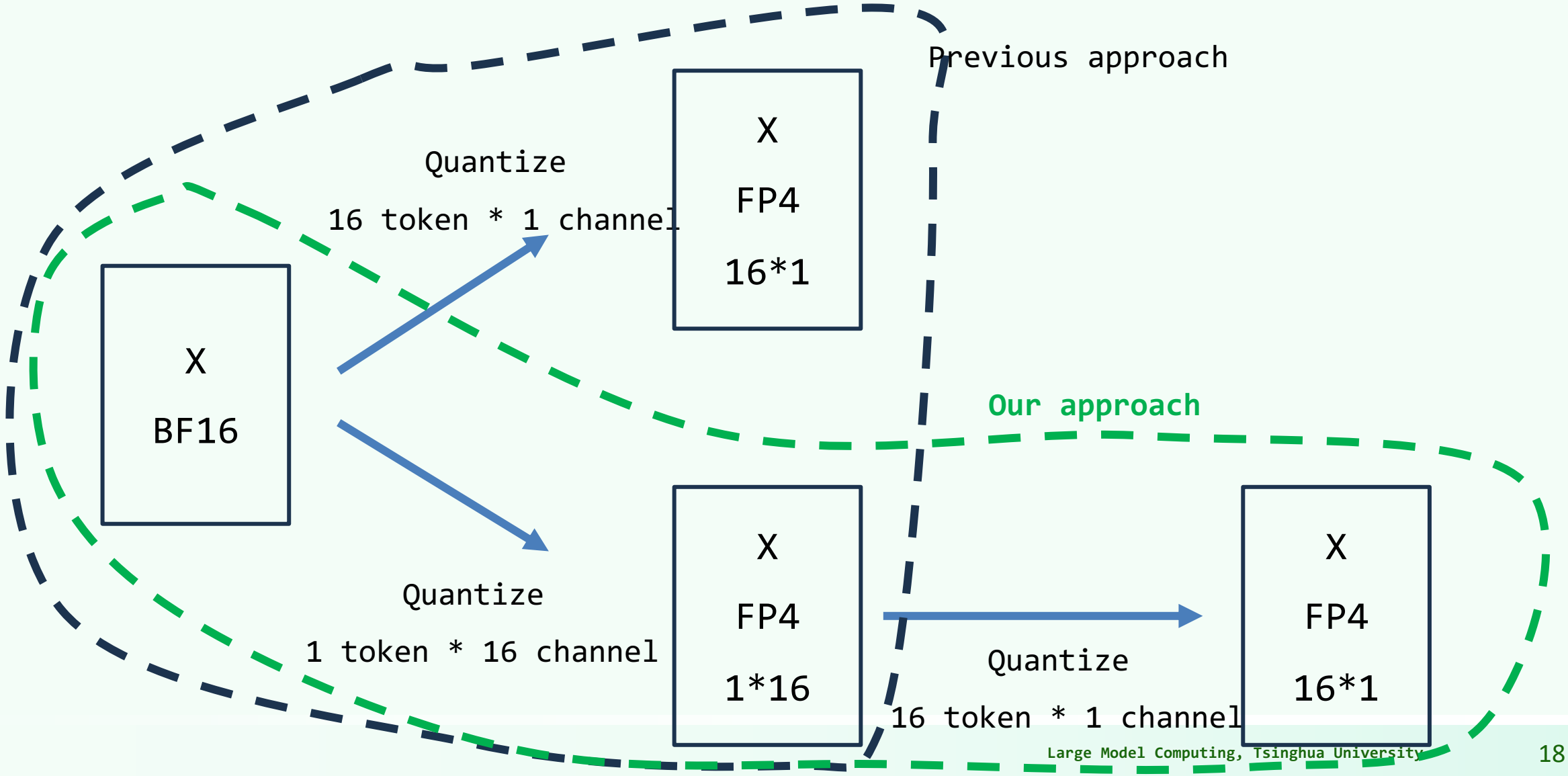
- Activation X
- Forward pass: 1 token * 16 channels
- Backward pass: 16 tokens * 1 channel
- Weight W
- Forward pass: 16 input * 1 output
- Backward pass: 1 input * 16 output

$$C_{ij} = \sum_k A_{ik}B_{kj} = \sum_{\substack{\text{each block } \boldsymbol{a} \text{ with scale } S_a \\ \text{each block } \boldsymbol{b} \text{ with scale } S_b}} S_a S_b \langle \boldsymbol{a}, \boldsymbol{b} \rangle$$



# 二次量化


Previous approach
X
FP4
16*1
Quantize
16 token * 1 channel
X
BF16
Our approach
X
FP4
1*16
Quantize
1 token * 16 channel
Quantize
16 token * 1 channel
X
FP4
16*1

# What's wrong with the previous approach?

$$\mathbf{Y} = Q_D^{(1)}(\mathbf{X}) \times Q_D^{(2)}(\mathbf{W}^\top)$$

Previous approach (microscaling paper)

$$\nabla_{\mathbf{X}}\mathcal{L} = Q_D^{(3)}(\nabla_{\mathbf{Y}}\mathcal{L}) \times Q_D^{(4)}(\mathbf{W})$$
$$\nabla_{\mathbf{W}}\mathcal{L} = Q_D^{(5)}\left((\nabla_{\mathbf{Y}}\mathcal{L})^\top\right) \times Q_D^{(6)}(\mathbf{X})$$

Our approach

$$\mathbf{Y} = Q_D^{(1)}(\mathbf{X}) \times Q_D^{(2)}(\mathbf{W}^\top)$$
$$\nabla_{\mathbf{X}}\mathcal{L} = Q_S^{(3)}(\nabla_{\mathbf{Y}}\mathcal{L}) \times Q_S^{(4)}\left(Q_D^{(2)}(\mathbf{W}^\top)^\top\right)$$
$$\nabla_{\mathbf{W}}\mathcal{L} = Q_S^{(5)}\left((\nabla_{\mathbf{Y}}\mathcal{L})^\top\right) \times Q_S^{(6)}\left(Q_D^{(1)}(\mathbf{X})\right)$$

Computes forward pass for a model with

- 1*16 quantized activation
- 1*16 quantized weight

Computes gradient for **another** model with

- 16*1 quantized activation
- 16*1 quantized weight

You are *not* training the model you actually use…

# What's the bottleneck?

$$\mathbf{Y} = Q_D^{(1)}(\mathbf{X}) \times Q_D^{(2)}(\mathbf{W}^\top)$$

$$\nabla_{\mathbf{X}}\mathcal{L} = Q_S^{(3)}(\nabla_{\mathbf{Y}}\mathcal{L}) \times Q_S^{(4)}\left(Q_D^{(2)}(\mathbf{W}^\top)^\top\right)$$

$$\nabla_{\mathbf{W}}\mathcal{L} = Q_S^{(5)}\left((\nabla_{\mathbf{Y}}\mathcal{L})^\top\right) \times Q_S^{(6)}\left(Q_D^{(1)}(\mathbf{X})\right)$$

| | DeiT-T | DeiT-S |
|---|---|---|
| Full Precision | 63.73 | 73.33 |
| Q1 | 61.50 | 71.66 |
| Q2 | 62.77 | 72.45 |
| Q3 | 63.46 | 72.97 |
| Q4 | 63.37 | 72.79 |
| Q5 | 63.81 | 73.25 |
| Q6 | 63.78 | 73.13 |
| All Quantizers | 59.75 | 71.03 |

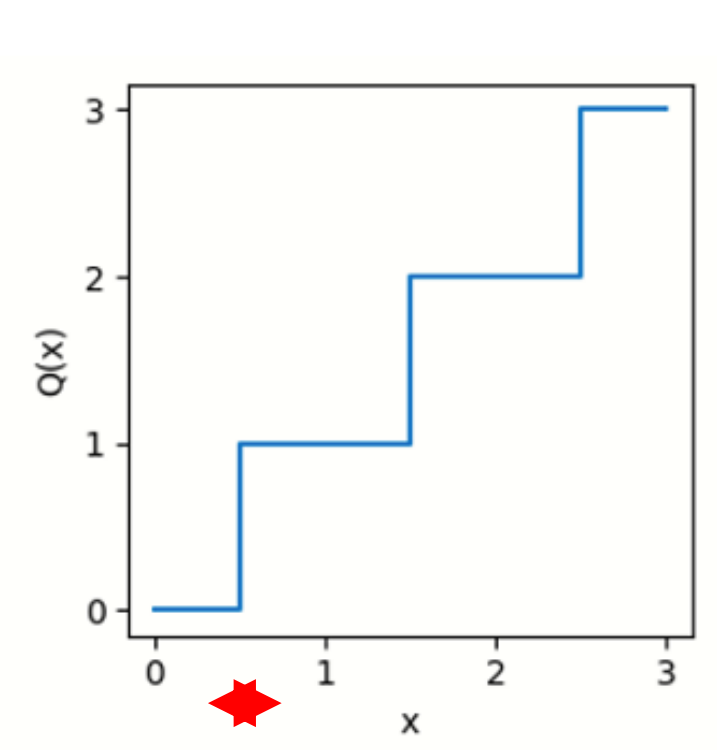
Forward quantizers are the bottleneck…

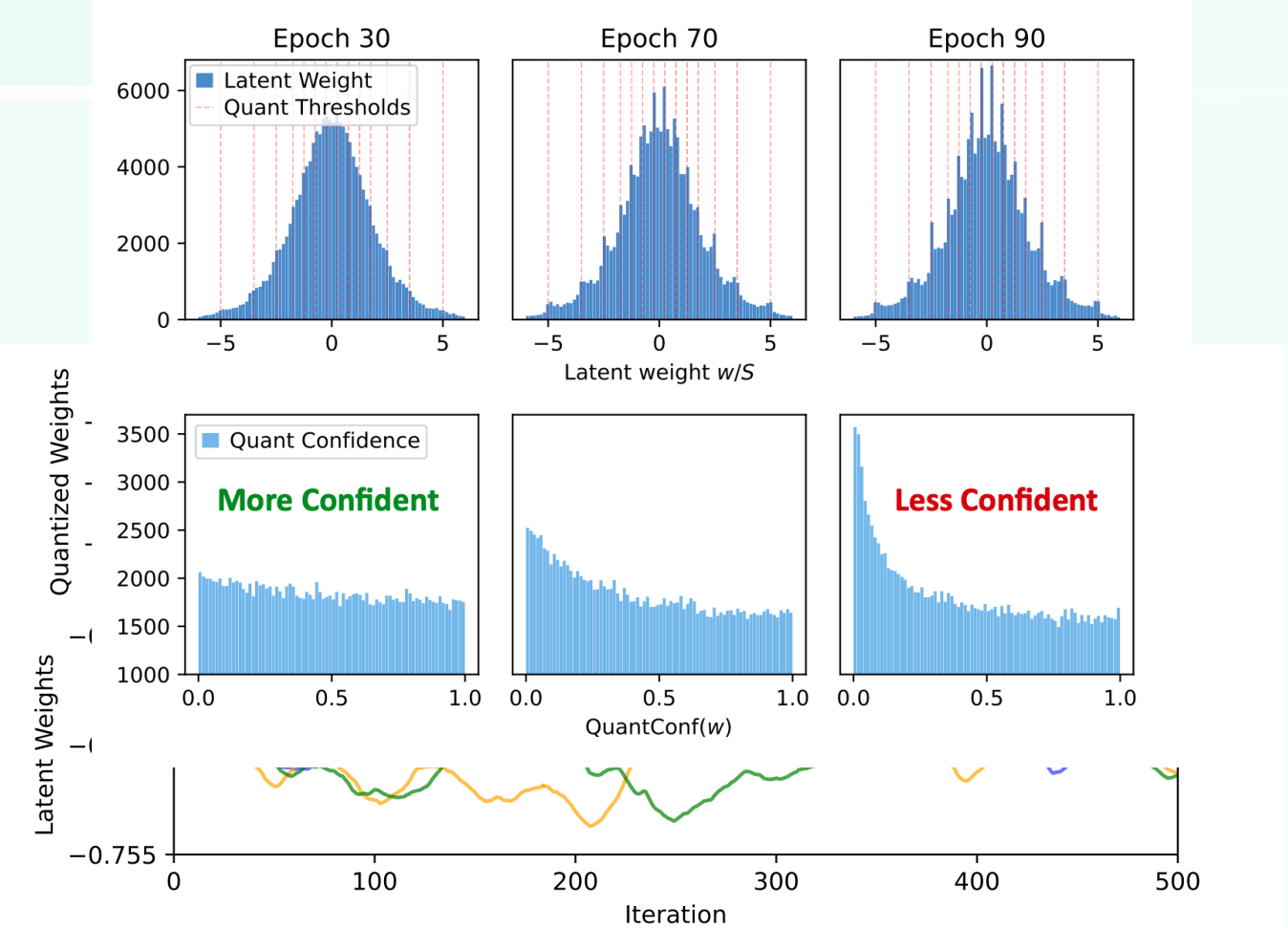
Intuition:

- You only compute the forward pass **once** during inference

But you can compute the backward pass **many times** during training

# Oscillation problem




The weight learns to swing around the critical point

The quantized model keeps oscillating

the optimization will not converge…

# Dilemma

Keep doing something bad

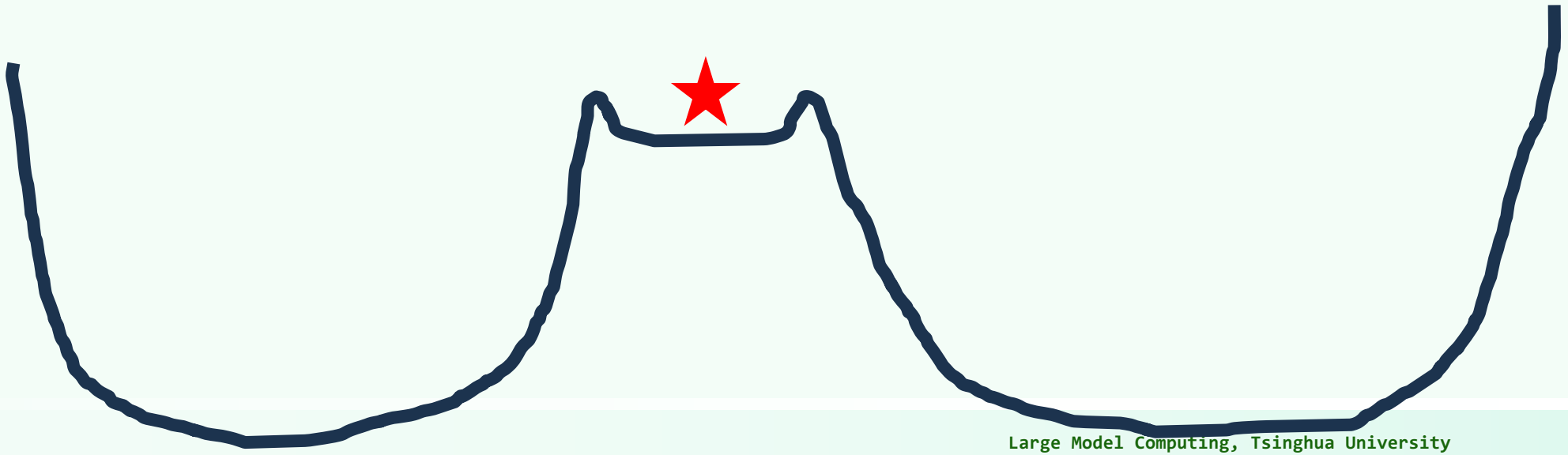
Oscillating in between

Keep doing something good

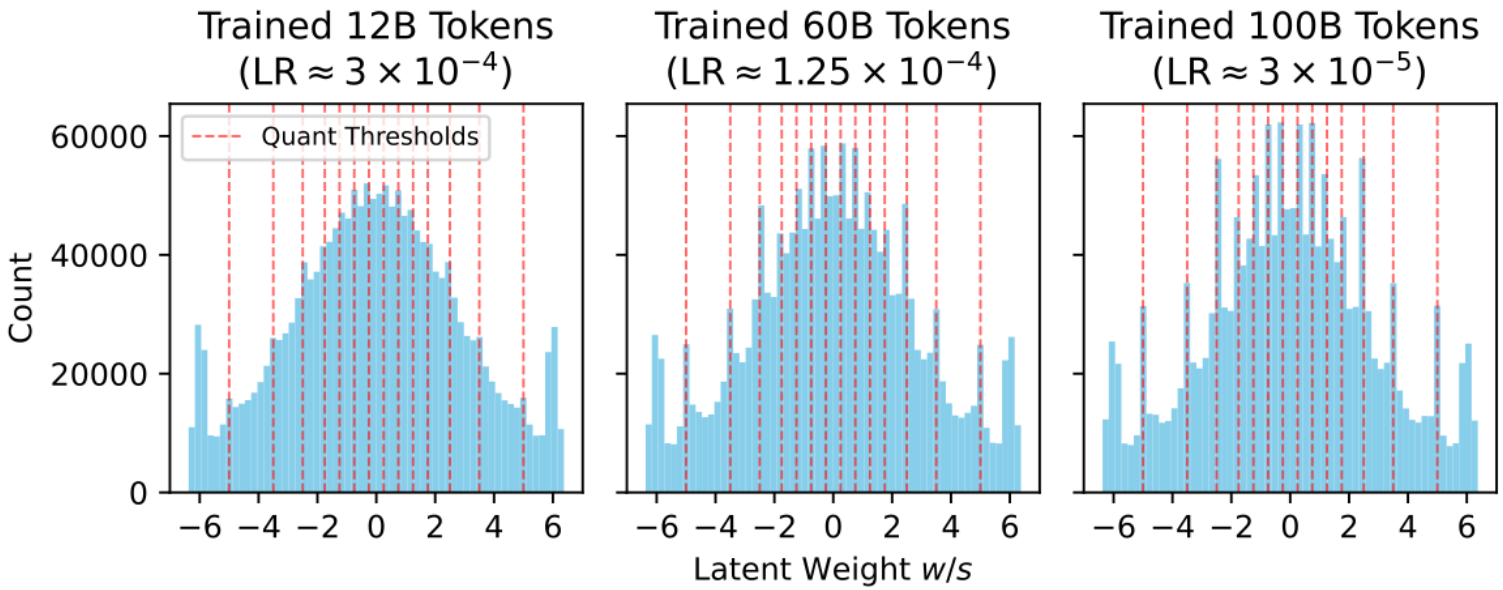
**Supervillain**

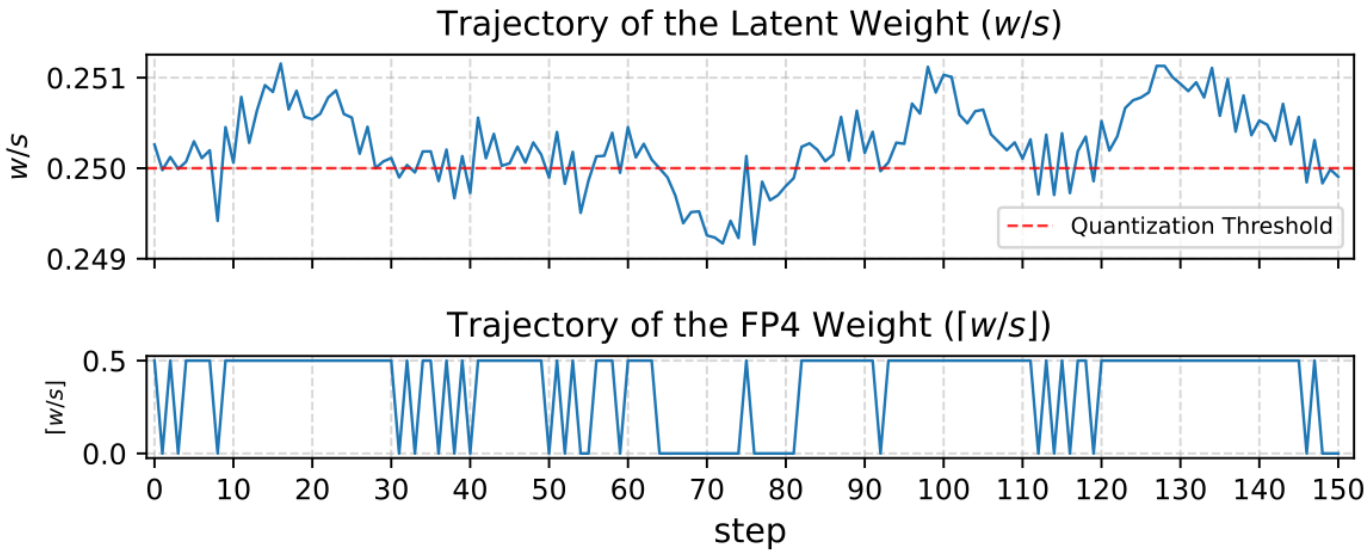
**Mediocre**

**Superhero**



# LLM训练的震荡


Trained 12B Tokens
(LR ≈ 3 × 10⁻⁴)
Trained 60B Tokens
(LR ≈ 1.25 × 10⁻⁴)
Trained 100B Tokens
(LR ≈ 3 × 10⁻⁵)
Quant Thresholds
Count
60000
40000
20000
0
−6 −4 −2 0 2 4 6
Latent Weight w/s


Trajectory of the Latent Weight (w/s)
w/s
0.251
0.250
0.249
Quantization Threshold
Trajectory of the FP4 Weight (⌈w/s⌋)
⌈w/s⌋
0.5
0.0
0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150
step

# 抑制震荡

❖ 将震荡最频繁的元素的latent weight强制重制回中央

**Algorithm 2:** OscillationSuppress($\theta$, $\text{dist}_M$, $\text{dist}_Q$, $\tau_{\text{osci}}$)

**Input:** $\theta$: model parameters;
$\tau_{\text{osci}}$: oscillation risk threshold;
$\text{dist}_M, \text{dist}_Q$: oscillation statistics.

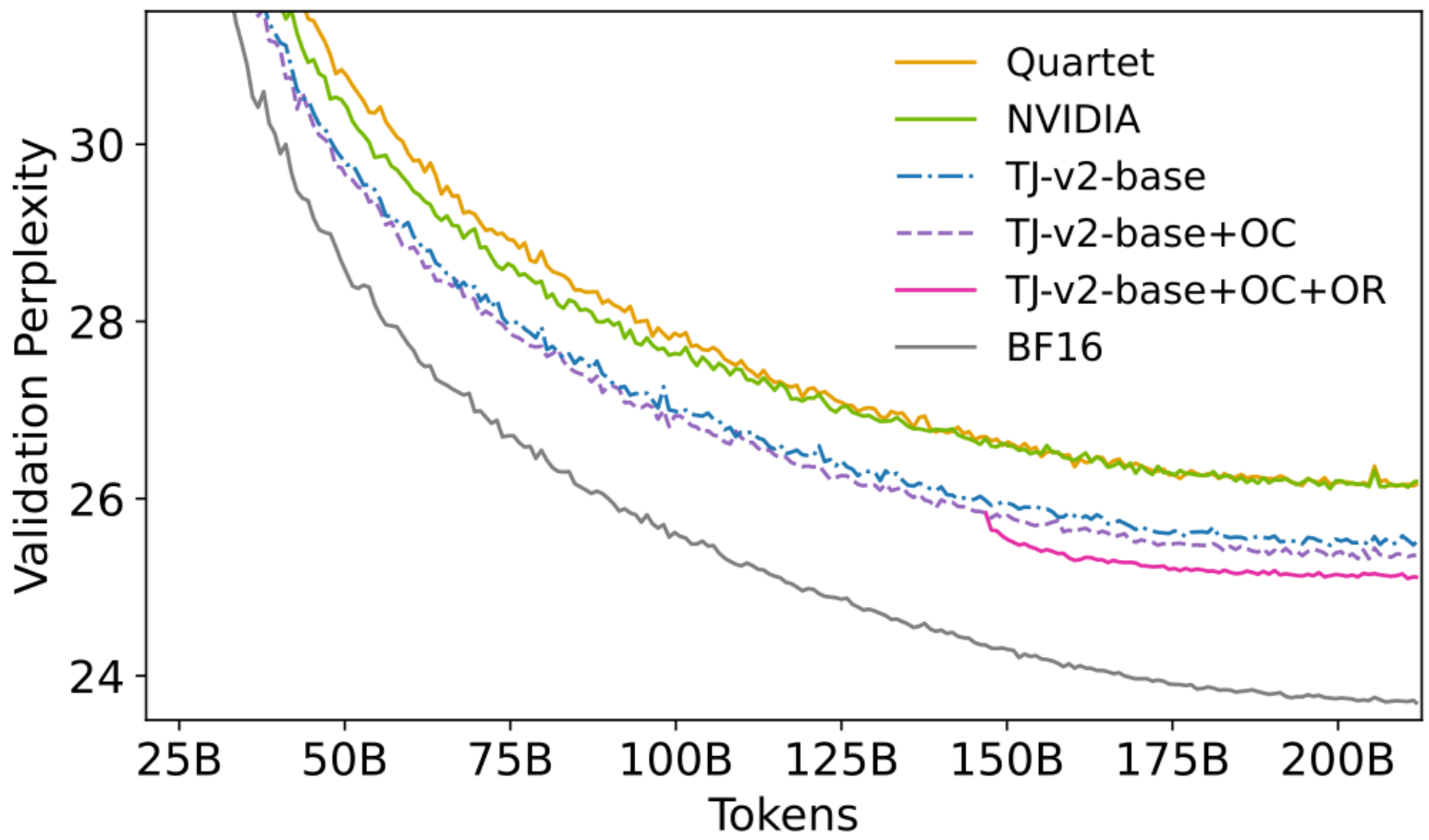
**Output:** Modified parameters $\theta$

**for** $i$-th Element $w_i$ **in** Quantized Parameters of $\theta$ **do**
  Calculate Osci-Risk: $\text{OsciRisk}(w_i) \leftarrow \text{dist}_Q(w_i)/\text{dist}_M(w_i)$;
  **if** $\text{OsciRisk}(w_i) \geq \tau_{\text{osci}}$ **then**
    // Reset it to the center of the bin,
    // where $w_{\text{FP4},i}$ is the FP4 value and $\text{scale}_i$ is the scaling factor.
    $w_i \leftarrow w_{\text{FP4},i} \cdot \text{scale}_i$
**return** $\theta$;

# 结果

| Methods \ OLMo2-Size | Train PPL | | | Validation PPL | | |
|---|---|---|---|---|---|---|
| | 70M | 150M | 370M | 70M | 150M | 370M |
| BF16 | 35.95 | 26.38 | 18.70 | 45.27 | 33.49 | 23.70 |
| Quartet | 40.77 | 29.25 | 20.76 | 51.23 | 36.89 | 26.16 |
| NVIDIA | 40.50 | 29.18 | 20.75 | 50.94 | 36.73 | 26.20 |
| TetraJet-v2-base (ours) | 39.26 | 28.39 | 20.23 | 49.33 | 35.88 | 25.50 |
| TetraJet-v2-full (ours) | **38.08** | **27.58** | **19.89** | **47.75** | **34.95** | **25.11** |

# 结果


Validation Perplexity
30
28
26
24
25B
50B
75B
100B
125B
150B
175B
200B
Tokens
Quartet
NVIDIA
TJ-v2-base
TJ-v2-base+OC
TJ-v2-base+OC+OR
BF16

# 量化案例：前馈网络

清华大学计算机系 陈键飞

《大模型计算》课程团队

# 前馈网络

```
residual = x
x = norm(x)
gate = gate_proj(x)
up = up_proj(x)
z = act_fn(gate) * up
down = down_proj(z)
y = residual + down
```

**利用FP数据类型**

**量化数值相乘后直接是FP32**

**无需反量化**

```
residual = x
x = norm(x)
qx = Q(x)

gate = gate_proj(qx)
up = up_proj(qx)

z = act_fn(gate) * up
qz = Q(z)

down = down_proj(z)
y = residual + down
```

**norm_Q算子**

**act_Q算子**

# 静态量化 vs 动态量化

- 量化算子分为两步：计算max；缩放&取整&类型转换

```
max = compute_abs_max(x)
scaled_x = x / max * range  # 127, 448…
q_x = cast_to_lp(round(scaled_x))
```

- 如果是per-tensor量化，需要扫描两次显存 ±±
- 静态量化：预先在校准集计算好max，推理时无需再算
- 动态量化：现场计算max
- 口诀：粗粒度推理用静态，其他都用动态
- 总结：裸写=扫三遍，+算子融合=扫两遍，+静态量化或细粒度量化=扫一遍
- 理论上来说，如果实现得理想，充分融合后访存甚至能比不做量化小点……

# 谢谢

清华大学计算机系 陈键飞

《大模型计算》课程团队