# 2025 동계 세미나

Low bit post-training quantization

***Sogang University***
*Vision & Display Systems Lab, Dept. of Electronic Engineering*

***Presented By***
*양진철*

# Outline

- Intro

- Papers
  - AdaLog: Post-Training Quantization for Vision Transformers with Adaptive Logarithm Quantizer (ECCV 2024)
  - SVDQuant: Absorbing Outliers by Low-Rank Components for 4-Bit Diffusion Models (ICLR 2025)

# Intro

- What is quantization?
  - 모델 최적화를 위한 motivation
    - Performance ↑ → Model size ↑
      - 컴퓨터 비전에서 모델들은 모델 사이즈를 크게 가지면서 성능을 향상
        - →모델 학습의 시간, latency 및 비용 증가
    - Edge device
      - Edge device의 부족한 메모리 용량
    - Applications such as real-time intelligent
      - health care monitoring, autonomous driving, …
  - Method for optimizing models
    - Quantization, Pruning, Knowledge Distillation, Efficient Network Design
  - Quantization은 파라미터의 값(weight, activation)의 표현 정밀도를 낮추는 과정
    - Floating point (FP32) value → INT value
  - Basic equations

Quantization : $x_q = \mathrm{clamp}(\left\lfloor \frac{x}{s} \right\rceil + z, 0, 2^b - 1)$

Dequantization : $\hat{x} = s \cdot (x_q - z)$

scale factor $s = \dfrac{\beta - \alpha}{2^b - 1}$

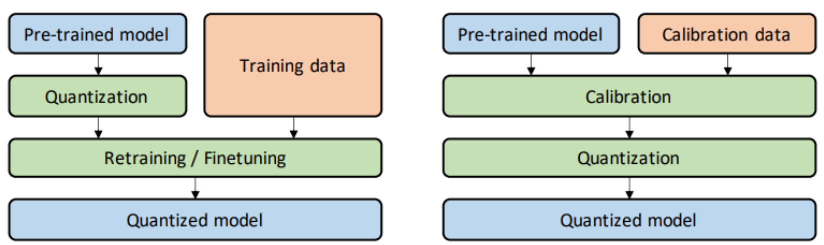
Zero-point $z = \left\lfloor -\dfrac{\min(x)}{s} \right\rceil$

Sogang University 서강대학교

VDS LAB

# Intro

- What is quantization?
  - Fine-tuning methods : PTQ vs QAT
    - Post-Training Quantization (PTQ)
      - Fine-tuning 없이 pre-trained model에서 모든 weight, activation quantization 파라미터를 quantization하는 방식
      - Inference에서 quantization하는 방법
      - QAT와 비교하여 낮은 accuracy
    - Quantization-Aware Training (QAT)
      - Fine-tuning을 하면서 loss를 최소로 하는 최적의 파라미터 찾는 방식
      - Loss를 최소로 하는 최적의 파라미터 찾기 위해 fine-tuning에 많은 시간과 비용을 들이는 단점 존재
      - PTQ와 비교하여 높은 accuracy 달성




< Left : QAT, Right : PTQ >

< Overview of QAT and PTQ >

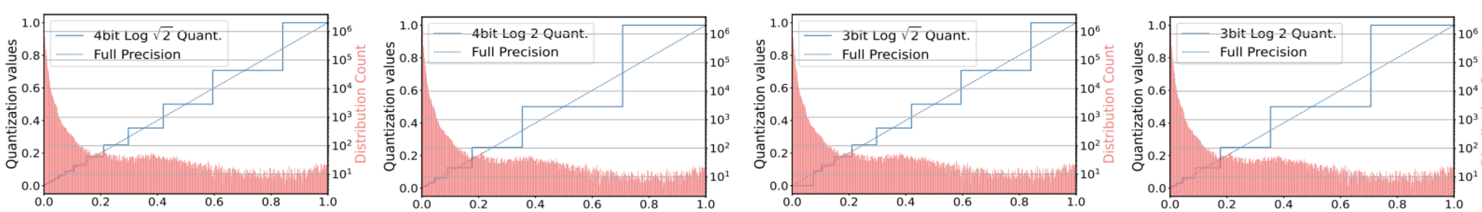
**AdaLog: Post-Training Quantization for Vision Transformers with Adaptive Logarithm Quantizer (ECCV 2024)**

# AdaLog[1)]

- Keyword
  - PTQ, low-bit quantization, imge classification, vit-based models

- Introduction
  - Image classification task에서 quantization의 한계
    - 기존 방법들이 low-bit에서는 큰 정확도 하락

- Analysis
  - 1) Inflexible Logarithm Base.
    - 기존의 log2, log√2 와 같은 고정된 log 기반 quantization 방법에서의 문제
  - 2) Excessively sparse partition of hyperparameter search space.
    - 기존의 grid search 기반 방법에서의 문제

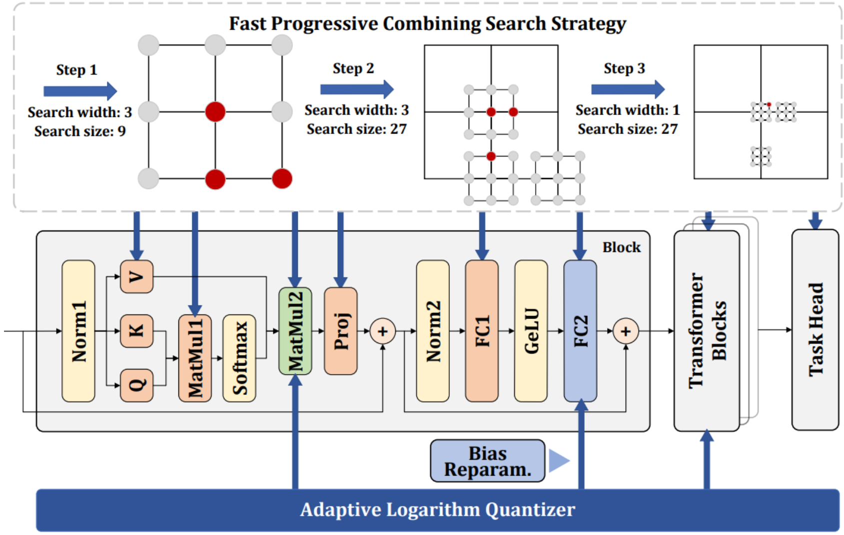



(a) 4-bit log√2 Quantizer (b) 4-bit log2 Quantizer (c) 3-bit log√2 Quantizer (d) 3-bit log2 Quantizer

< Histogram of post-Softmax activations >

# AdaLog[1)]

- Method
  - Adalog quantization
    - Adaptive Logarithm Base Quantizer
      - Power-law probability 분포를 잘 처리하기 위한 방법
      - Post-Softmax, Post-GELU
    - Fast Progressive Combining Search
      - 빠르게 하이퍼파라미터를 최적화하기 위한 방법
      - QKV, Proj, FC1, FC2, MatMul1 및 MatMul2




< Illustration on the framework >



Sogang University
VDS LAB

# AdaLog[1)]

- Method
  - Adaptive Logarithm Base Quantizer
    - 최적의 로그 밑수를 적응적으로 탐색하는 방법
    - Log2 quantizer

$$\begin{cases} \text{Quantization: } A^Z = \text{clamp}\left(\left\lfloor -\log_2 \frac{A}{s} \right\rceil, 0, 2^{bit} - 1\right) \\ \text{Dequantization: } \hat{A} = s \cdot 2^{-A^Z} \end{cases}$$

Log2 quantizer는 하드웨어 친화적이지만 low-bit quantization에서는 에러 증가

    - Log√2 quantizer

$$\begin{cases} \text{Quantization: } A^Z = \text{clamp}\left(\left\lfloor -2\log_2 \frac{A}{s} \right\rceil, 0, 2^{bit} - 1\right) \\ \text{Dequantization: } \hat{A} = \tilde{S} \cdot 2^{\left\lfloor -\frac{A^Z}{2} \right\rfloor} ;\ \tilde{S} = s \cdot (p[x^Z] \cdot (\sqrt{2}-1)+1) \end{cases}$$

Log√2 quantizer는 Log2 quantizer보다 에러가 적지만 하드웨어 비친화적

    - **Adaptive Logarithm Base Quantizer**

$$\begin{cases} \text{Quantization: } A^Z = \text{clamp}\left(\left\lfloor -\log_b \frac{A}{s} \right\rceil, 0, 2^{bit} - 1\right) \\ \qquad = \text{clamp}\left(\left\lfloor \frac{-\log_2 \frac{A}{s}}{-\log_2 b} \right\rceil, 0, 2^{bit} - 1\right) \\ \text{Dequantization: } \hat{A} = s \cdot b^{-A^Z} \end{cases}$$

$b^{-A^Z}$ 연산의 bit shift로 가속화 진행 불가 (하드웨어 친화적이지 않음)

서강대학교 SOGANG UNIVERSITY



VDS LAB

1) Wu, Zhuguanyu, et al. "Adalog: Post-training quantization for vision transformers with adaptive logarithm quantizer." European Conference on Computer Vision. Springer, Cham, 2025.

# AdaLog[1)]

- Method
  - Adaptive Logarithm Base Quantizer
    - 밑수 b를 사용하는 경우의 문제점 해결 방안
      - 유리수로 근사화 (log2 b using a rational number, i.e., $\log_2 b \approx q/r$)

**Adaptive Logarithm Base Quantizer**

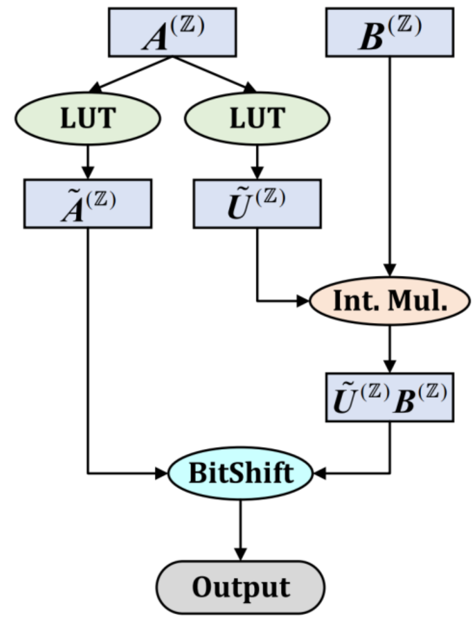
$$\begin{cases} \text{Quantization: } A^Z = \text{clamp}\left(\left\lfloor -\log_b \frac{A}{s} \right\rceil, 0, 2^{bit}-1\right) \\ \text{Dequantization: } \hat{A} = s \cdot b^{-A^Z} \end{cases}$$

유리수로 근사화

$$= s \cdot \left(2^{-\tilde{A}^Z} \circ 2^{-\tilde{U}}\right) \quad ; \tilde{U} = \frac{(q \cdot A^Z) \bmod r}{r}, \ \tilde{A}^Z = \left\lfloor \frac{q \cdot A^Z}{r} \right\rfloor \quad \text{LUT}$$

하드웨어 친화적

부동소수점 행렬: $2^{-\tilde{U}}$ → $\tilde{U}^Z = \left\lfloor \frac{2^{-\tilde{U}}}{s_{table}} \right\rceil$ ; $s_{table} = \frac{1}{2 \cdot (2^{bit}-1)}$




< standard linear integer multiplication >

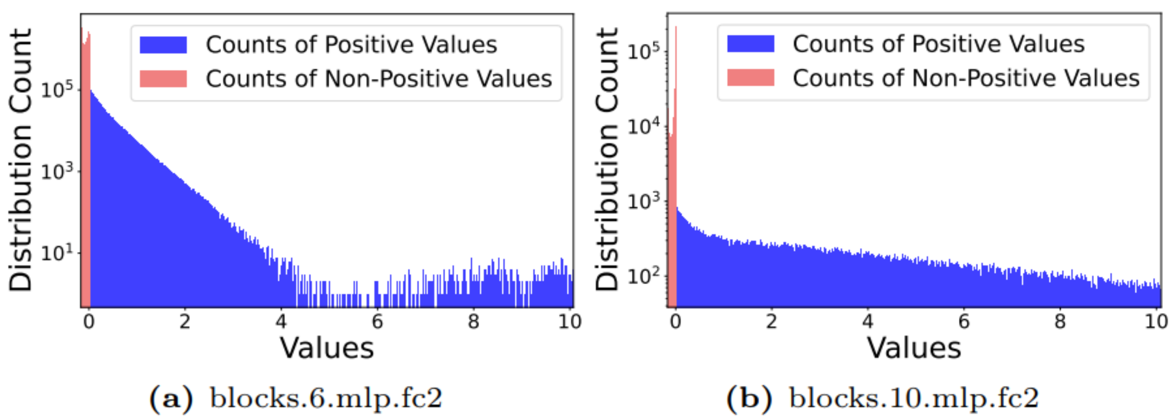
  - Application of Adaptive Logarithm Base Quantizer in MatMul2

$$\therefore \text{Dequantization: } \hat{A} \cdot \hat{B} = s_A \cdot \left(2^{-\tilde{A}^Z} \circ 2^{-\tilde{U}}\right) \cdot s_B \cdot B^Z$$

$$= s_A \cdot s_B \cdot s_{table} \cdot \left[\left(\tilde{U}^Z B^Z\right) \gg \tilde{A}^Z\right]$$

# AdaLog[1)]

- Method
  - Adaptive Logarithm Base Quantizer for Post-GELU Layers
    - Post-Softmax (MatMul2)와 유사한 Post-GELU (FC2) layers의 power-law distribution
      - 문제점
        - ✓ Data distribution이 서로 다른 layer 사이에서 큰 변동 존재
        - ✓ 값의 대부분이 -0.17 ~ 0 에 집중
      - Adaptive Logarithm Base Quantizer 변형
        - ✓ 양수 값만 처리하므로 이를 해결하기 위해 Bias Reparameterization 기법 사용




< Illustration on the distribution of post-GeLU activations >

# AdaLog[1)]

- Method
  - Adaptive Logarithm Base Quantizer for Post-GELU Layers
    - Post-GELU linear layer FC2 수식 재구성

$$Y = W \cdot X + b \qquad X = (-0.17, 0]$$

$$= W \cdot \underbrace{(X + 0.17 \cdot 1_{mxn})}_{X'\text{: 양수}} + (b - 0.17 \cdot W \cdot 1_m)$$

$$X' \begin{cases} \text{Quantization: } X'^{Z} = \text{clamp}\left(\left\lfloor -\log_b \frac{X'}{s} \right\rceil, 0, 2^{bit} - 1\right) \\ \text{Dequantization: } \hat{X}' = s \cdot b^{-X'^{Z}} \approx X + 0.17 \cdot 1_{mxn} \end{cases}$$

$$b_{\text{rep}} = b - 0.17 \cdot \widehat{W} \cdot 1_m$$

$$\therefore \text{Dequantization: } \widehat{W} \cdot \hat{X} = s_X \cdot (2^{-\tilde{X}^{Z}} \circ 2^{-\tilde{U}}) \cdot s_W \cdot W^{Z} + b_{\text{rep}}$$
$$= s_X \cdot s_W \cdot s_{table} \cdot [(\tilde{U}^{Z} W^{Z}) \gg \tilde{X}^{Z}] + b_{\text{rep}}$$

# AdaLog[1)]

- Method
  - Fast Progressive Combining Search
    - 두 가지 종류의 하이퍼파라미터를 빠르게 결정하기 위한 방법
      - Uniform quantizer and AdaLog quantizer
    - 기존 방법과의 차이점
      - Brute-force search: 가능한 모든 하이퍼파라미터 조합 탐색
        - ✓ Complexity of brute-force search is O(nm) ; n and m are the number of candidates
      - Alternating search: 한 하이퍼파라미터를 고정한 상태에서 다른 하이퍼파라미터를 탐색
        - ✓ Complexity of alternating search is O(n+m)
        - ✓ Local minimum으로 인한 성능 하락 존재
      - Beam Search: 탐색 공간에서 최적의 하이퍼파라미터를 찾기 위해 상위 k개의 후보만 유지하며 탐색하는 방법 기반으로 설계
        - ✓ 모든 조합을 찾는 Brute-force에 비해 낮은 complexity
        - ✓ Local minimum 방지

# AdaLog[1)]

- Method
  - Fast Progressive Combining Search
    - Initialization step: 넓은 범위에서 A와 B의 후보 값을 설정하여 초기 후보 집합 $C_0$ 생성
    - Progressive searching step: 각 반복 단계에서 후보를 대략적 탐색, 해당 후보 주변에서 탐색 세분화
    - Final step: 최적의 a*, b* 하이퍼파라미터를 선택하여 quantization loss 최소화

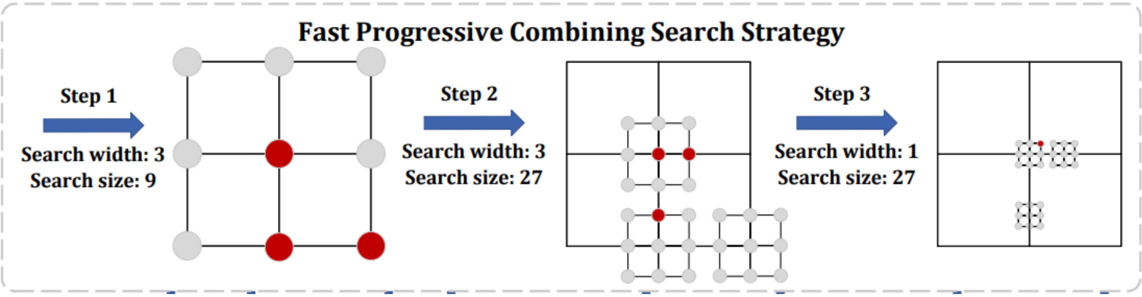
**Algorithm 1** Fast Progressive Combing Searching.

**Input:** Coefficients $x, y, z_1, z_2, k, p$; a pretrained full-precision model; a set of calibration data $\mathcal{D}_{calib}$; and the $l$-th layer to be quantized $\phi_l$.
**Output:** Quantization hyperparameters $a^*, b^*$.

\# The initialization step:
1: Generate the raw input $\boldsymbol{X}_l$ and output $\boldsymbol{O}_l$ by $\phi_l$ based on $\mathcal{D}_{calib}$, and compute the percentiles $pct_0$, $pct_{0.1}$, $pct_{0.9}$ and $pct_1$ by [14].
2: Compute the uniform partition of the first and second hyperparameters as $\mathcal{A} = \{pct_{0.1} + i \cdot \tau_A | i = 0, \cdots, x\}$ and $\mathcal{B} = \{pct_{0.9} + j \cdot \tau_B | j = 0, \cdots, y\}$ with the intervals $\tau_A = (pct_0 - pct_{0.1})/x$ and $\tau_B = (pct_1 - pct_{0.9})/y$.
3: Generate the candidate set $\mathcal{C}_0$ as the Cartesian product of $\mathcal{A}$ and $\mathcal{B}$: $\mathcal{C}_0 = \mathcal{A} \times \mathcal{B}$.
\# The progressive searching step:
4: **for** $i = 0, \cdots, p$ **do**
\# The coarse searching step:
5: Construct the subset $\mathcal{C}' \subset \mathcal{C}_i$ by selecting the partitions that have the top-$k$ smallest quantization loss.
\# The expanding step:
6: Update the intervals for fine partitions: $\tau_A := \tau_A/(2 \cdot z_1)$, $\tau_B := \tau_B/(2 \cdot z_2)$.
7: Update the candidate set with fine partitions: $\mathcal{C}_{i+1} = \{(a + i \cdot \tau_A, b + j \cdot \tau_B) | (a, b) \in \mathcal{C}'; i = -z_1, \cdots, z_1; j = -z_2, \cdots, z_2\}$.
8: **end for**
9: The optimal hyperparameter $(a^*, b^*) \in \mathcal{C}_p$ is the one that has the smallest quantization loss.

# AdaLog[1)]

- Method
  - Fast Progressive Combining Search




– Step 1. Initialization
  - 대략적인 search를 통해 넓은 범위에서 초기 최적 후보 선택
  - Search width: a*, b* 두 하이퍼파라미터 각각에 대해 3개의 후보 값 고려 (3)
  - Search size: 두 하이퍼파라미터의 모든 조합 탐색 (9)

– Step 2. Progressive searching
  - 선택된 유망한 후보들 주변에서 탐색 범위를 세분화하여 더 정밀한 탐색 수행
  - Search width: 각 하이퍼파라미터에서 선택된 후보 주변에서 다시 3개의 후보 값을 추가 탐색 (3)
  - Search size: 후보 주변에 추가된 새로운 후보 공간 탐색 (27)

– Step 3. Final
  - 가장 최적의 후보를 중심으로 quantization loss가 최소가되는 하이퍼파라미터 결정

# AdaLog[1)]

- Experimental Results
  - ImageNet dataset에서 다양한 모델에서 Image classification task 실험 결과

| Model | Full Prec. | Method | W3/A3 | W4/A4 | W6/A6 |
|---|---|---|---|---|---|
| ViT-S/224 | 81.39 | PTQ4ViT | 0.10 | 42.57 | 78.63 |
| | | APQ-ViT | - | 47.95 | 79.10 |
| | | RepQ-ViT | 0.10 | 65.05 | 80.43 |
| | | **AdaLog (Ours)** | **13.88** | **72.75** | **80.91** |
| ViT-B/224 | 84.54 | PTQ4ViT | 0.10 | 30.69 | 81.65 |
| | | APQ-ViT | - | 41.41 | 82.21 |
| | | RepQ-ViT | 0.10 | 68.48 | 83.62 |
| | | **AdaLog (Ours)** | **37.91** | **79.68** | **84.80** |
| DeiT-T/224 | 72.21 | PTQ4ViT | 3.50 | 36.96 | 69.68 |
| | | APQ-ViT | - | 47.94 | 70.49 |
| | | RepQ-ViT | 0.10 | 57.43 | 70.76 |
| | | **AdaLog (Ours)** | **31.56** | **63.52** | **71.38** |
| DeiT-S/224 | 79.85 | PTQ4ViT | 0.10 | 34.08 | 76.28 |
| | | APQ-ViT | - | 43.55 | 77.76 |
| | | RepQ-ViT | 0.10 | 69.03 | 78.90 |
| | | **AdaLog (Ours)** | **24.47** | **72.06** | **79.39** |
| DeiT-B/224 | 81.80 | PTQ4ViT | 31.06 | 64.39 | 80.25 |
| | | APQ-ViT | - | 67.48 | 80.42 |
| | | RepQ-ViT | 0.10 | 75.61 | 81.27 |
| | | **AdaLog (Ours)** | **57.45** | **78.03** | **81.55** |
| Swin-S/224 | 83.23 | PTQ4ViT | 28.69 | 76.09 | 82.38 |
| | | APQ-ViT | - | 77.15 | 82.67 |
| | | RepQ-ViT | 0.10 | 79.45 | 82.79 |
| | | **AdaLog (Ours)** | **64.41** | **80.77** | **83.19** |
| Swin-B/224 | 85.27 | PTQ4ViT | 20.13 | 74.02 | 84.01 |
| | | APQ-ViT | - | 76.48 | 84.18 |
| | | RepQ-ViT | 0.10 | 78.32 | 84.57 |
| | | **AdaLog (Ours)** | **69.75** | **82.47** | **85.09** |

1) Wu, Zhuguanyu, et al. "Adalog: Post-training quantization for vision transformers with adaptive logarithm quantizer." European Conference on Computer Vision. Springer, Cham, 2025.
2) Li, Zhikai, et al. "Repq-vit: Scale reparameterization for post-training quantization of vision transformers." *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023.
3) Li, Yuhang, Xin Dong, and Wei Wang. "Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks." *arXiv preprint arXiv:1909.13144* (2019).

# AdaLog[1)]

- Experimental results
  - Ablation studies
    - Effect of the main components

| AdaLog | FPCS | ViT-S (81.39) | | DeiT-T (72.21) | | Swin-S (81.80) | |
|---|---|---|---|---|---|---|---|
| | | W3/A3 | W4/A4 | W3/A3 | W4/A4 | W3/A3 | W4/A4 |
| | | 3.51 | 62.20 | 22.73 | 58.01 | 44.65 | 78.40 |
| ✓ | | 11.40 | 72.01 | 28.41 | 62.87 | 61.50 | 80.46 |
| | ✓ | 3.77 | 63.14 | 24.80 | 59.93 | 44.61 | 78.79 |
| ✓ | ✓ | **13.88** | **72.75** | **31.56** | **63.52** | **64.41** | **80.77** |

    - On the Efficiency of AdaLog
      - AdaLog는 quantized LUT를 사용하여 RepQ-ViT[2)] (Log√2 quantizer) 보다 효율적
        - ✓ FixOP[3)]: 8bit weight와 8bit activation 값 사이의 하나의 연산

| Model | Bits | Method | Prec. | FixOPs | Model Size |
|---|---|---|---|---|---|
| DeiT-T FixOPs: 20.1B Size: 21.9MB | 4/4 | RepQ-ViT | 57.43 | 0.613B | 3.4MB |
| | 4/4 | AdaLog | **63.52** | **0.539B** | 3.4MB |
| | 3/3 | RepQ-ViT | 0.10 | 0.444B | 2.7MB |
| | 3/3 | AdaLog | **31.56** | **0.391B** | 2.7MB |

# AdaLog[1)]

- Experimental results
  - Ablation studies
    - On the Efficiency of FPCS

| Model | Method | Top-1 Acc. (%) | Complexity | GPU Min. |
|---|---|---|---|---|
| DeiT-T/224 (W3A3) | Alternating [35] | 28.41 | $O(n)$ | 3.3 |
| | Brute Force [31] | 32.04 | $O(n^2)$ | 183 |
| | FPCS (Ours) | 31.56 | $O(pn)$ | 4.1 |
| DeiT-S/224 (W3A3) | Alternating [35] | 22.17 | $O(n)$ | 5.7 |
| | Brute Force [31] | 29.38 | $O(n^2)$ | 312 |
| | FPCS (Ours) | 28.51 | $O(pn)$ | 6.5 |

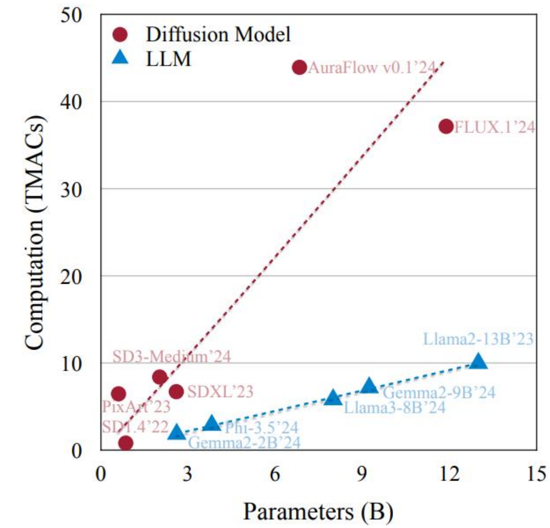
    - Results on the post-GELU quantizers

| Method | Rep. | ViT-S | ViT-B | DeiT-T | DeiT-S | DeiT-B | Swin-S | Swin-B |
|---|---|---|---|---|---|---|---|---|
| Full-Precision | - | 81.39 | 84.54 | 72.21 | 79.85 | 81.80 | 83.23 | 85.27 |
| Uniform [2] | × | 63.14 | 78.08 | 59.93 | 69.23 | 76.02 | 78.79 | 80.67 |
| T-Uniform [5] | × | 65.29 | <u>78.76</u> | 60.96 | 69.78 | 76.69 | <u>80.51</u> | <u>80.93</u> |
| Log2 [4] | ✓ | 39.83 | 71.27 | 59.33 | 66.30 | 68.53 | 80.36 | 78.95 |
| Log$\sqrt{2}$ [2] | ✓ | <u>72.44</u> | 46.16 | <u>62.91</u> | <u>70.60</u> | <u>77.15</u> | 75.91 | 24.50 |
| **AdaLog** | ✓ | **72.75** | **79.68** | **63.52** | **72.06** | **78.03** | **80.77** | **82.47** |

**SVDQuant: Absorbing Outliers by Low-Rank Components for 4-Bit Diffusion Models (ICLR 2025)**

1) Li, Muyang, et al. "Svdqunat: Absorbing outliers by low-rank components for 4-bit diffusion models." *arXiv preprint arXiv:2411.05007* (2024).

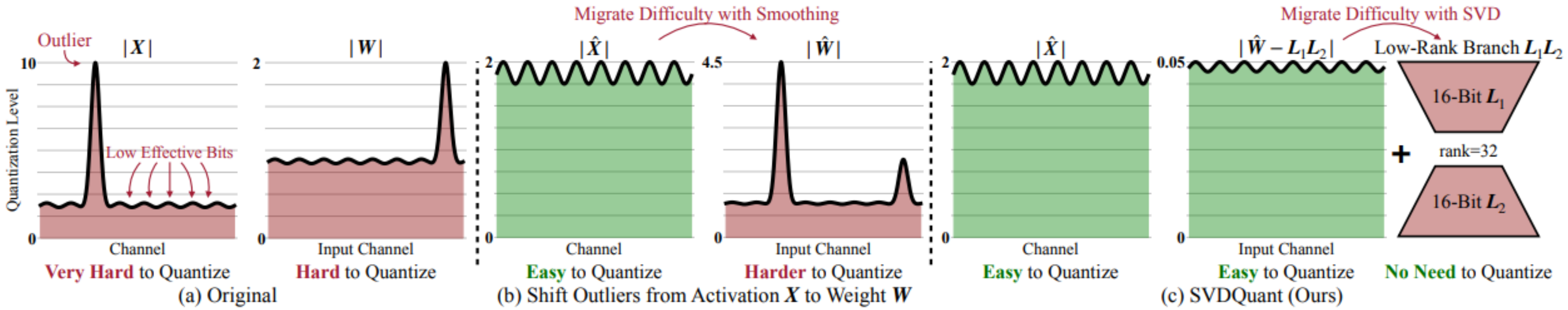
# SVDQuant[1)]




- Keyword
  - PTQ, low-bit quantization, diffusion models
- Introduction
  - Diffusion model의 inference time 증가
    - Moore's law slows down
    - 고품질 이미지를 생성하는데 모델이 커지면서 메모리 요구 사항이 크게 증가하여 inference time 증가
  - Diffusion model의 quantization 한계
    - 기존 방법들은 outlier로 인해 low-bit에서는 큰 정확도 하락을 확인
- Analysis
  - 1) Quantize activations
    - Weight만 quantization하는 방식은 GPU에서 가속화 불가능
    - Weight와 activation을 동일한 bit로 quantization 진행
  - 2) Memory access overhead

# SVDQuant[1)]

- Method
  - SVDQuant
    - Outlier Migration
      - Activation과 weight의 outlier migration
    - Low-rank branch via SVD decomposition
      - Quantization을 용이하기 위해 low-rank branch를 통해 outlier migration 보정
    - LoRunner: Kernel fusion
      - Low-rank branch 실행 시 추가적인 메모리 비용 발생
      - 메모리 접근 최소화 및 속도 향상을 위한 LoRunner 설계




< Overview of SVDQuant >

1) Li, Muyang, et al. "Svdqunat: Absorbing outliers by low-rank components for 4-bit diffusion models." *arXiv preprint arXiv:2411.05007* (2024).
2) Xiao, Guangxuan, et al. "Smoothquant: Accurate and efficient post-training quantization for large language models." *International Conference on Machine Learning*. PMLR, 2023.

# SVDQuant[1)]

- Method
  - Migrate outliers from activation to weight
    - Quantization 시 activatio과 weigh에 outlier가 존재하여 양자화 오류가 크게 증가

Error decomposition

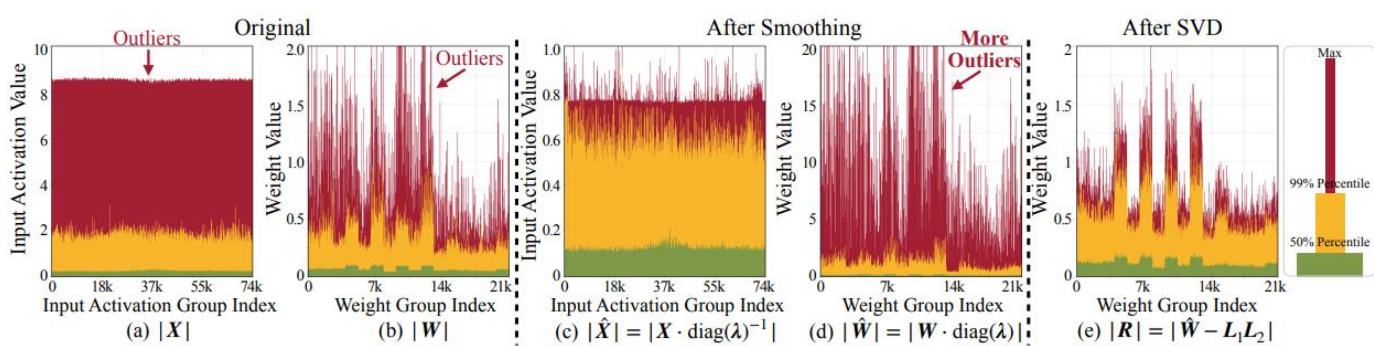
$$E(X,W) = \|XW - Q(X)Q(W)\|_F$$

$$E(X,W) \le \|X\|_F \|W - Q(W)\|_F + \|X - Q(X)\|_F (\|W\|_F + \|W - Q(W)\|_F)$$

    - Smoothquant[2)] 방법을 사용하여 activation에서 outlier를 제거하기 위해 크기를 줄이고 weight를 조정
      - Activation $X$와 weight $X$를 채널별 smoothing 계수 $\lambda$를 사용해 scaling

$$\hat{X} = X \cdot \mathrm{diag}(\lambda)^{-1},\ \widehat{W} = W \cdot \mathrm{diag}(\lambda) \qquad ;\ \text{scaling } \lambda = \max(|X|)^{\alpha} / \max(|W|)^{1-\alpha}$$

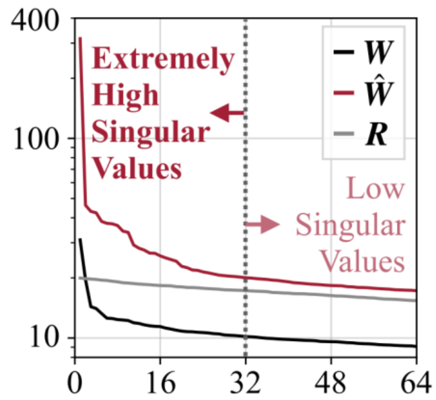
      - Smoothed activation은 크기가 줄어들고 outlier가 감소하여 quantization error 감소
      - Smoothed weight는 크기와 outlier가 증가하여 quantization error 증가
    - 따라서, total quantization error 감소가 제한적




< Example value distribution of inputs and weights in PixArt-Σ >

1) Li, Muyang, et al. "Svdqunat: Absorbing outliers by low-rank components for 4-bit diffusion models." *arXiv preprint arXiv:2411.05007* (2024).

# SVDQuant[1)]




- Method
  - Absorb magnified weight outliers with a low-rank branch
    - Smoothed weight는 크기와 outlier가 증가
    - 16-bit low-rank branch 추가하여 weight의 outlier 흡수

$$\widehat{W} = L_1 L_2 + R; \quad L_1, L_2\text{: low-rank, } R\text{: residual}$$

$$XW = \hat{X}\widehat{W} = \hat{X}L_1L_2 + \hat{X}R \approx \underbrace{\hat{X}L_1L_2}_{\text{16-bit low-rank branch}} + \underbrace{Q(\hat{X})Q(R)}_{\text{4-bit residual}} \quad ; L_1, L_2\text{: low-rank, } R\text{: residual}$$

※ Low-rank branch가 weight의 주요 정보를 보존 residual의 크기와 outlier를 감소

- $\hat{X}$는 outlier에서 자유롭기 때문에 $\|R\|_F$ 와 $\|R - Q(R)\|_F$ 최적화

$$E(\hat{X}, R) = \left\|\hat{X}\widehat{W} - \left(\hat{X}L_1L_2 + Q(\hat{X})Q(R)\right)\right\|_F = \left\|\hat{X}R - Q(\hat{X})Q(R)\right\|_F$$
$$\leq \|X\|_F \boxed{\|R - Q(R)\|_F} + \|X - Q(X)\|_F \boxed{\left(\|R\|_F + \|R - Q(R)\|_F\right)}$$

$$c = \sqrt{\frac{\log(\text{size}(R))\pi}{\text{size}(R)}}$$

- Quantization error bound

$R$이 정규조건을 만족한다면

$E[\max(|R|)] \leq c \cdot E[\|R\|_F]$ ⟶ $\boldsymbol{E[\|R - Q(R)\|_F] \leq \frac{c\sqrt{\text{size}(R)}}{q_{\max}} \cdot E[\|R\|_F]}$; $\text{size}(R) = \textit{the number of elements in } R$

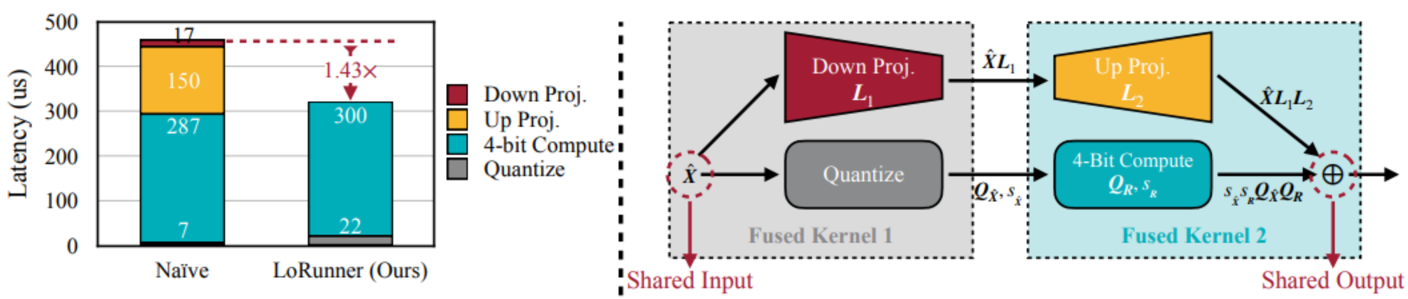
$\|R - Q(R)\|_F$가 $\|R\|_F$에 의해 제한 ⟶ $\boldsymbol{\|R\|_F = \left\|\widehat{W} - L_1L_2\right\|_F}$**이므로 최적의** $\boldsymbol{L_1, L_2}$ **탐색**

SVD를 통해 해결 ⟶ $\widehat{W} = USV$로 SVD 수행 ⟶ 최적의 $\boldsymbol{L_1 = U_{:,1:r}}$, $\boldsymbol{L_2 = V_{1:r,:}}$

- Low-rank branch를 반복적으로 업데이트하고 $R$을 조정함으로써 quantization error 감소

# SVDQuant[1)]

- Method
  - LORUNNER: FUSING LOW-RANK AND LOW-BIT BRANCH KERNELS
    - Low-rank branch는 계산 비용이 적지만 memory access bottleneck으로 인해 50% 추가 latency 발생
      - 입력 및 출력 데이터 크기가 줄어들지 않아 memory acces가 높은 비용
      - Diffusion transformer block에서 QKV projection은 출력 크기가 L2 cache를 초과
        - ✓ DRAM으로의 추가적인 load 및 store operation 발생
    - LoRunner kenel fusion
      - Shared input: Down projection과 quantize 커널은 동일한 input 공유
      - Shared output: Up projection과 4bit compute 커널은 동일한 output 공유
      - Low-rank branch와 activation을 공유하여 추가 메모리 접근을 제거하고 커널 호출 횟수를 절반으로 줄여 5~10%의 추가 latency만 발생



(a) Latency Breakdown on QKV projection
(b) LoRunner Kernel Fusion

# SVDQuant[1)]

- Experimental Results
  - MJHQ-30K, Densely Captioned Images (DCI) dataset에서의 정량적 평가

| Backbone | Model | Precision | Method | MJHQ Quality FID (↓) | MJHQ Quality IR (↑) | MJHQ Similarity LPIPS (↓) | MJHQ Similarity PSNR(↑) | sDCI Quality FID (↓) | sDCI Quality IR (↑) | sDCI Similarity LPIPS (↓) | sDCI Similarity PSNR (↑) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| DiT | FLUX.1 -dev (50 Steps) | BF16 | – | 20.3 | 0.953 | – | – | 24.8 | 1.02 | – | – |
| | | INT W8A8 | Ours | 20.4 | 0.948 | 0.089 | 27.0 | 24.7 | 1.02 | 0.106 | 24.9 |
| | | W4A16 | NF4 | 20.6 | 0.910 | 0.272 | 19.5 | 24.9 | 0.986 | 0.292 | 18.2 |
| | | INT W4A4 | Ours | **20.0** | 0.924 | 0.259 | 20.0 | **24.6** | 0.992 | 0.275 | **18.8** |
| | | FP W4A4 | Ours | 20.9 | **0.932** | **0.245** | **20.2** | 25.6 | **0.998** | **0.269** | 18.7 |
| | FLUX.1 -schnell (4 Steps) | BF16 | – | 19.2 | 0.938 | – | – | 20.8 | 0.932 | – | – |
| | | INT W8A8 | Ours | 19.2 | 0.966 | 0.120 | 22.9 | 20.7 | 0.975 | 0.133 | 21.3 |
| | | W4A16 | NF4 | 18.9 | 0.943 | **0.257** | **18.2** | 20.7 | 0.953 | **0.263** | **17.1** |
| | | INT W4A4 | Ours | **18.1** | **0.965** | 0.292 | 17.5 | **19.8** | **0.986** | 0.298 | 16.4 |
| | | FP W4A4 | Ours | 20.1 | 0.957 | 0.281 | 17.4 | 21.7 | 0.971 | 0.280 | 16.6 |
| | PixArt-Σ (20 Steps) | FP16 | – | 16.6 | 0.944 | – | – | 24.8 | 0.966 | | |
| | | INT W8A8 | ViDiT-Q | **15.7** | 0.944 | 0.137 | 22.5 | **23.5** | **0.974** | 0.163 | 20.4 |
| | | INT W8A8 | Ours | 16.3 | **0.955** | **0.109** | **23.7** | 24.2 | 0.969 | **0.129** | **21.8** |
| | | INT W4A8 | ViDiT-Q | 37.3 | 0.573 | 0.611 | 12.0 | 40.6 | 0.600 | 0.629 | 11.2 |
| | | INT W4A4 | ViDiT-Q | 412 | -2.27 | 0.854 | 6.44 | 425 | -2.28 | 0.838 | 6.70 |
| | | INT W4A4 | Ours | 20.1 | 0.898 | 0.394 | 16.2 | 25.1 | 0.922 | 0.434 | 14.9 |
| | | FP W4A4 | Ours | **18.3** | **0.946** | **0.326** | **17.4** | **23.7** | **0.978** | **0.357** | **16.1** |
| UNet | SDXL -Turbo (4 Steps) | FP16 | – | 24.3 | 0.845 | – | – | 24.7 | 0.705 | – | – |
| | | INT W8A8 | MixDQ | **24.1** | 0.834 | 0.147 | 21.7 | 25.0 | 0.690 | 0.157 | 21.6 |
| | | INT W8A8 | Ours | 24.3 | **0.845** | **0.100** | **24.0** | **24.8** | **0.701** | **0.110** | **23.7** |
| | | INT W4A8 | MixDQ | 27.7 | 0.708 | 0.402 | 15.7 | 25.9 | 0.610 | 0.415 | 15.7 |
| | | INT W4A4 | MixDQ | 353 | -2.26 | 0.685 | 11.0 | 373 | -2.28 | 0.686 | 11.3 |
| | | INT W4A4 | Ours | 24.2 | 0.796 | 0.279 | 17.7 | 25.7 | 0.657 | 0.289 | 17.6 |
| | | FP W4A4 | Ours | **24.1** | **0.822** | **0.250** | **18.5** | **24.7** | **0.699** | **0.261** | **18.4** |
| | SDXL (30 Steps) | FP16 | – | 16.6 | 0.729 | – | – | 22.5 | 0.573 | – | – |
| | | INT W8A8 | TensorRT | 20.2 | 0.591 | 0.247 | 22.0 | 25.4 | 0.453 | 0.265 | 21.7 |
| | | INT W8A8 | Ours | **16.6** | **0.718** | **0.119** | **26.4** | **22.4** | **0.574** | **0.129** | **25.9** |
| | | INT W4A4 | Ours | 21.4 | 0.591 | 0.306 | 20.4 | 26.8 | 0.470 | 0.320 | 20.3 |
| | | FP W4A4 | Ours | **19.0** | **0.607** | **0.294** | **21.0** | **25.4** | **0.480** | **0.312** | **20.7** |

1) Li, Muyang, et al. "Svdquant: Absorbing outliers by low-rank components for 4-bit diffusion models." *arXiv preprint arXiv:2411.05007* (2024).

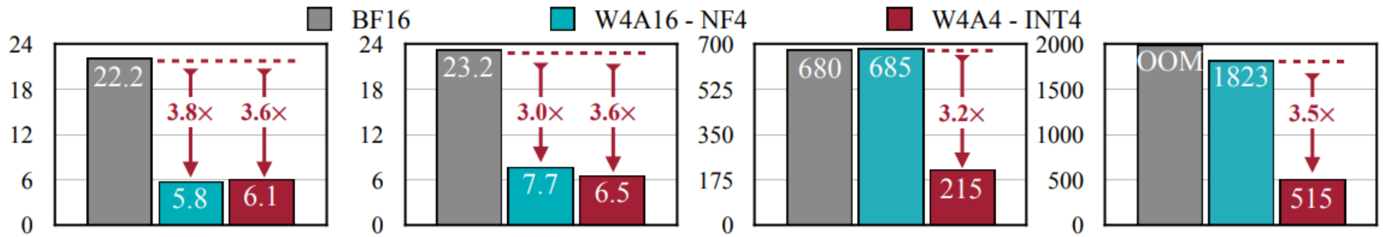
# SVDQuant[1)]

- Experimental Results
  - MJHQ-30K dataset에서의 정성적 평가


(a) **FLUX.1-dev** (b) **FLUX.1-schnell**

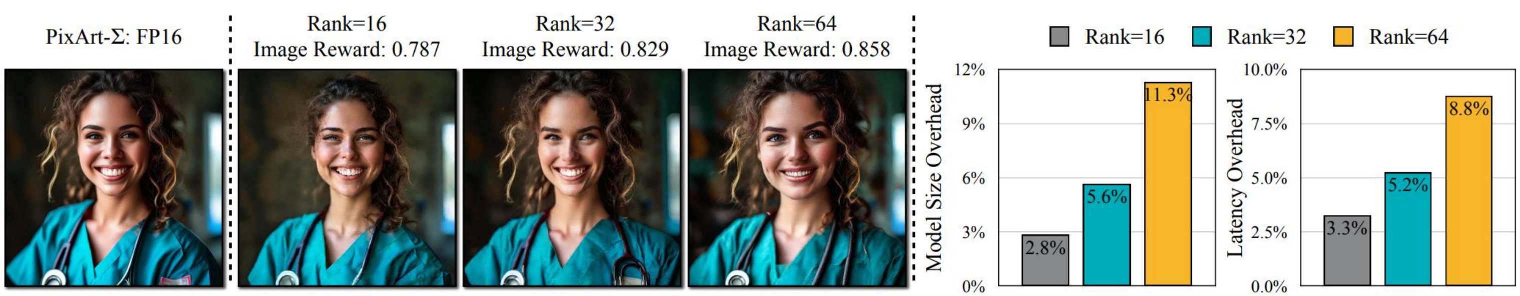
# SVDQuant[1)]

- Experimental Results
  - FLUX.1 model에서의 memory save & speedup
    - 전체 size 3.6x 감소와 low-rank branch로 인한 0.3 GiB overhead
    - Inference engine LoRunner로 1.2x memory fooprint 절약
    - 3.2x, 3.5x speedup




(a) Model Size (GiB) (b) Inference Memory (GiB) (c) Single Step Latency on Desktop 4090 (ms) (d) Single Step Latency on Laptop 4090 (ms)

  - Trade-off of increasing rank
    - The results of different rank r in SVDQuant on PixArt-Σ




Prompt: *award winning photography of a beautiful medic smiling*

# Thank you