TetraJet-v2: Accurate NVFP4 Training for Large Language Models with Oscillation Suppression and Outlier Control

Yuxiang Chen[1,2], Xiaoming Xu[1], Pengle Zhang[1], Michael Beyer[3], Martin Rapp[3], Jun Zhu[1], and Jianfei Chen*[1]

[1]Dept. of Comp. Sci. and Tech., Tsinghua University
[2]Zhili College, Tsinghua University
[3]Bosch AI Research, Renningen, Germany

chenyuxi22@mails.tsinghua.edu.cn; xiaomingxu@tsinghua.edu.cn; zhangpl21@mails.tsinghua.edu.cn; {michael.beyer2,martin.rapp}@de.bosch.com; {dcszj,jianfeic}@tsinghua.edu.cn

Abstract

Large Language Models (LLMs) training is prohibitively expensive, driving interest in low-precision fully-quantized training (FQT). While novel 4-bit formats like NVFP4 offer substantial efficiency gains, achieving near-lossless training at such low precision remains challenging. We introduce **TetraJet-v2**, an end-to-end 4-bit FQT method that leverages NVFP4 for activations, weights, and gradients in all linear layers. We identify two critical issues hindering low-precision LLM training: weight oscillation and outliers. To address these, we propose: 1) an unbiased double-block quantization method for NVFP4 linear layers, 2) OsciReset, an algorithm to suppress weight oscillation, and 3) OutControl, an algorithm to retain outlier accuracy. **TetraJet-v2** consistently outperforms prior FP4 training methods on pre-training LLMs across varying model sizes up to 370M and data sizes up to 200B tokens, reducing the performance gap to full-precision training by an average of 51.3 %.

1 Introduction

Large language models (LLMs) have drastically grown in size in recent years, reaching several hundred billions of parameters [1] and are trained on trillions of tokens [2]. At such scales, the cost of pre-training a single state-of-the-art model can exceed 100 million US dollars [3], which is prohibitively expensive. Leveraging low-precision computations has emerged as a powerful means to reduce the computational resources and memory requirements for training neural networks. This is achieved by performing both the forward and the backward pass with quantized tensors (i.e., activations, weights, and gradients), which can be computed more efficiently on modern hardware, resulting in higher throughput. Training using 16-bit floating-point formats such as FP16 [4] or BF16 [5], as well as 8-bit formats such as FP8 [6, 7] is increasingly mature and has reached commercial deployment [6]. These works have been enabled by advancements in computing hardware, such as support of FP8 by the NVIDIA Hopper architecture [8].

Recently, *microscaling* formats for 4-bit representations, such as MXFP4 [9] and NVFP4 [10], have been introduced into modern hardware, e.g., NVIDIA Blackwell architecture [11]. Unlike traditional low-precision formats that apply a single global scaling factor to an entire tensor, microscaling formats assign separate scaling factors to smaller groups of values (e.g., 32 for MXFP4 and 16 for NVFP4). This fine-grained quantization significantly reduces quantization error, especially for tensors containing outlier elements, with a slight increase in control complexity at runtime. Among these microscaling formats, NVFP4 stands out as the most accurate FP4 format, delivering up to 3× higher compute performance and 2× lower memory usage compared to MXFP8 [12], greatly improving training efficiency.

Prior work on post-training quantization has demonstrated that LLMs can maintain their original task performance even at low-precision (e.g., 4-bit [13]). However, achieving stable 4-bit training has proven more challenging. Early work on fully quantized 4-bit training incurred a large drop in model performance [14, 15, 16]. More recently, FP4 pre-training work adopts the modern data format MXFP4/NVFP4 for a more accurate representation, and combines several techniques to overcome outlier and instability problems in quantization training, such as Hadamard Rotation [17, 18], and differentiable gradient estimation [19]. In particular, NVIDIA's NVFP4 pre-training recipe [12] demonstrates the feasibility of large-scale FP4 training by leveraging 2D weight quantization and Hadamard transforms to preserve model accuracy. However, their approach is not fully 4-bit: a subset of Transformer blocks use BF16, and the model is kept in higher precision in the final training stage. While these techniques mitigate the

*Corresponding author

performance loss, the resulting model is no longer fully FP4. Still, *fully* FP4 training methods are yet not comparable to high-precision training in our experiments, and the challenging optimization problems lying in the FP4 training remains unsolved.

In this work, we find that the performance drop can mainly be attributed to two effects: 1) *Weight oscillation*, where the quantized weights oscillate between two quantization bins while there are only small changes in the corresponding high precision master weights, which impairs the model's performance on downstream tasks [20]. 2) *Outlier features*, where few channels in activations have massive magnitudes that lead to high dynamic ranges that FP4 cannot represent accurately, which are highly relevant to the model's final performance [21].

In this paper, we propose **TetraJet-v2**, a novel low-precision training method for LLMs. Compared to TetraJet [22], we switch the data format from MXFP4 to NVFP4 for a more accurate representation, and develop novel techniques to resolve oscillation and outlier problems on LLMs, while TetraJet's effectiveness was demonstrated only on Vision Transformers. We employ NVFP4 for activations, weights, and gradients of all linear layers in both the forward and backward pass to achieve highly efficient training that can fully leverage hardware support on modern GPUs. We introduce several methods to enable stable and accurate training of LLMs, achieving minimal impact on task performance compared to training the same model with high precision. In summary, our contributions are as follows:

- An **unbiased double-block quantization** method for NVFP4 linear layers. We adopt double-block scaling to satisfy the numerical needs of the scaling factors in NVFP4, and design a fully FP4 linear layer with unbiased gradient estimation.
- **OsciReset**, a novel algorithm for suppressing the oscillation of weights during training. To the best of our knowledge, we are the *first* to analyze and effectively address the weight oscillation problem in low-precision LLM training.
- **OutControl**, a novel algorithm that retains outlier accuracy during low-precision training. Differing from prior outlier selection methods, our selection for outlier channels works in both forward and backward, providing more enhancement than just selecting in the forward pass.
- Extensive evaluation on NVFP4 pre-training for LLMs. We pre-trained OLMo-2 [23] on the dataset OLMo-2-Mix-1124. With extensive pretraining and downstream evaluations across different model sizes (70M, 150M, 370M) and data scales (50B, 100B, 200B tokens), we demonstrate that our method consistently outperforms previous approaches when training with fully FP4 formats. Compared to the previous state-of-the-art method, our approach reduces the performance gap between FP4 and full-precision training by an average of 51.3%.

  We further perform empirical analyses to inspire future work, revealing that the forward propagation and MLP.ffn2 are most sensitive to quantization. Our results also suggest that the FP6 $\times$ FP4 training format, though not yet the focus of current mainstream hardware and algorithms, can offer substantial improvement.

2 Related Works

Low-Precision Training Prior work on neural network quantization focused mainly on optimizations that improve inference efficiency of models after training. To this end, post-training quantization (PTQ) and quantization-aware training (QAT) methods only quantize the forward pass. In contrast, fully quantized training (FQT) aims at improving the computational efficiency during training by quantizing activations, weights, and gradients in the forward and backward pass. The limited dynamic range of 4-bit data types leads to increasing errors and thus numerical instabilities such that multiple methods are leveraged simultaneously to reduce the loss gap between full precision training and FQT. Low-precision training recipes typically use a combination of fine-grained quantization with microscaling data formats, stochastic rounding for unbiased gradient estimates, and outlier compensation strategies such as random Hadamard transforms or clamping to reduce their effect on quantization [9, 12, 17, 18, 19]. Nevertheless, accuracy has still not reached performance levels of high-precision training.

Weight Oscillation Problem Prior work on weight oscillation focuses on convolutional and Transformer-based neural networks in the vision domain and mainly targets QAT, i.e., when fine-tuning

a full-precision checkpoint [20, 24, 22]. In [20] two methods are introduced that address the oscillation during QAT. First, a regularization term that aims at dampening oscillating weights, effectively encouraging them to be closer to the center of a quantization bin. Second, oscillating weights are frozen during QAT. For pre-training this approach is not ideal as these weights will no longer be updated, which limits their contribution toward optimizing the training loss. Recent work on training vision Transformers with `MXFP4` introduced the two methods Q-EMA and Q-Ramping, which aim at limiting the magnitude and frequency of updates for oscillating weights [22]. Using an exponential moving average (EMA) for weight update steps has been shown to reduce oscillation of weights and the impact thereof on the model's task performance. Further, by adapting the update frequency and using a higher gradient accumulation step, Q-Ramping can effectively reduce the oscillation problem. In this work we expand upon [22] and introduce a novel method that addresses weight oscillation during low-precision (pre-)training of LLMs that requires little tuning of hyperparameters.

Outlier Control Several works tried to mitigate outliers with architectural changes, such as gated attention [25] or attention sinks [26]. However, these methods cannot fully mitigate outliers, and their quantization remains an open challenge to be addressed at the algorithm level. The introduction of highly fine-grained block-wise quantization addresses outlier features to some extent, since large values only affect the quantization of a smaller subset of values. In [27], matrix multiplications with outliers features are handled separately in the forward pass. Products involving outliers are computed with higher precision (e.g., `FP16`), products for remaining values are performed with quantized low-precision 8-bit integer multiplication kernels. Several recent works use random or learned rotations to transform features into a more favorable space that can be quantized easily [13, 28]. However, with the transition from PTQ to FQT, the challenge of accurately quantizing outliers has shifted from post-training deployment-related optimizations to pre-training. Recent work on `NVFP4` pre-training of LLMs integrates prior methods that have previously mainly been used for PTQ optimization, such as Hadamard transforms for end-to-end low-precision pre-training [12, 17, 18].

3 TetraJet-v2's NVFP4 Linear Layer Design

3.1 Preliminary

NVFP4 Format Each floating-point (`FP`) number consists of a sign-bit, exponent-bits, and mantissa-bits. An `FP` format is written as `E`x`M`y if it has x exponent-bits, and y mantissa-bits. The most common 4-bit floating-point (`FP4`) is `E2M1`, whose values are $\texttt{FP4Values} = \{0, \pm 0.5, \pm 1, \pm 1.5, \pm 2, \pm 3, \pm 4, \pm 6\}$.

To represent a matrix in `FP4` precision, we need additional scaling factors to scale the numerical range to $[-6, 6]$. The `MXFP4` and `NVFP4` formats achieve this by partitioning each matrix into groups of 32 and 16 elements, respectively, and they use an 8-bit scaling factor for each group to scale the elements. Specifically, `MXFP4` uses an unsigned `E8M0` scaling factor, while `NVFP4` utilizes the more precise `E4M3` scaling factor. Given the finer-grained group shape and the more precise scaling format of `NVFP4`, we select `NVFP4` as our training format.

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

3.2 Double-Block Quantization to NVFP4 formats

However, the range of the 8-bit `E4M3` scaling factor of `NVFP4` is only $[-448, 448]$. This is too small to represent large values, so values X_i should be scaled into $[-448 \times 6, 448 \times 6]$ before quantization with the NVFP4-quantizer. Following the numerical limitation of `NVFP4`, we add a 1×128-*outer-block* outside the 1×16-*inner-block*. For an outer-block $\{X_i\}_{i=0}^{127}$, we need to scale values to $[-448 \times 6, +448 \times 6]$ with a global scale factor S_{global}, and then cast to `NVFP4` with group quantization for each inner-block:

$$S_{\text{global}} = \frac{\max_{i=0}^{127} |X_i|}{448 \times 6}, \qquad S_{\text{block}k} = \frac{\max_{i=16k}^{16k+15} |X_i / S_{\text{global}}|}{6},$$
$$P_i = \text{round}_{\text{FP4}} \left(\frac{X_i}{S_{\text{global}} \times S_{\text{block}\lfloor i/16 \rfloor}} \right), \quad X_i \approx P_i \times S_{\text{global}} \times S_{\text{block}\lfloor i/16 \rfloor}.$$

In this way, we can not only satisfy the numerical needs but also avoid computing the maximum absolute value on a full tensor. This is more hardware-friendly and also more accurate compared to [12], where a per-tensor second quantization scaling is used. In Sec. 6.2.1, we demonstrate that our finer outer-block scaling yields better performance through experiments.

3.3 NVFP4 Linear Layers with Unbiased Gradients

The forward/backward pass of a linear layer with input $\mathbf{X}$ and weight $\mathbf{W}$ can be demonstrated as:

$$\mathbf{Y} = \mathbf{X} \times \mathbf{W}^\top, \quad \mathrm{d}\mathbf{X} = \mathrm{d}\mathbf{Y} \times \mathbf{W}, \quad \mathrm{d}\mathbf{W} = \mathrm{d}\mathbf{Y}^\top \times \mathbf{X}$$

To accelerate training, we need to compute all three matrix multiplications (`MM`s) in linear layers with `FP4`. To achieve this, our method quantizes the six input matrices of the three `MM`s to `NVFP4`, following TetraJet's design for `MXFP4` linear layer [22], which can be formulated as:

$$\begin{aligned}
\mathbf{Y} &= \widehat{\mathbf{X}} \times \widehat{\mathbf{W}^\top}, \quad \widehat{\mathbf{X}} := Q_D^{(1)}(\mathbf{X}), \quad \widehat{\mathbf{W}^\top} := Q_D^{(2)}(\mathbf{W}^\top) \\
\mathrm{d}\mathbf{X} &= Q_S^{(3)}(\mathrm{d}\mathbf{Y}) \times Q_S^{(4)}\left(\widehat{\mathbf{W}}\right), \\
\mathrm{d}\mathbf{W} &= Q_S^{(5)}\left(\mathrm{d}\mathbf{Y}^\top\right) \times Q_S^{(6)}\left(\widehat{\mathbf{X}}\right)
\end{aligned}$$

where $\mathbf{X} \in \mathbb{R}^{N \times D}, \mathbf{W} \in \mathbb{R}^{C \times D}, \mathbf{Y} \in \mathbb{R}^{N \times C}$, $\widehat{\mathbf{W}} := \widehat{\mathbf{W}^\top}^\top$, and $\mathrm{d}\mathbf{X}, \mathrm{d}\mathbf{Y}, \mathrm{d}\mathbf{W}$ are the input/output/weight gradient matrices respectively referring to $\nabla_{\mathbf{X}}\mathcal{L}, \nabla_{\mathbf{Y}}\mathcal{L}, \nabla_{\mathbf{W}}\mathcal{L}$ ($\mathcal{L}$ is the loss function), and $Q_{D/S}$ represents the deterministic/stochastic rounding. To meet the acceleration need of `MM`s, the group shape of $Q^{(1)}, Q^{(3)}, Q^{(5)}$ should be 1×16 and $Q^{(2)}, Q^{(4)}, Q^{(6)}$ should 16×1.

Our basic linear layer design has three major differences compared to NVIDIA's training recipe [12]:

1. **The alignment of tensors in forward and backward**: In the computation of $\mathrm{d}\mathbf{W}$, the correct gradient should be computed upon $\widehat{\mathbf{X}}$ rather than $\mathbf{X}$ to align with the forward. Therefore, we choose to unbiasedly estimate $\widehat{\mathbf{X}}$ while [12] just estimate $\mathbf{X}$ in backward.
2. **The unbiasedness from stochastic rounding**: We adopt stochastic rounding to achieve unbiased estimation in backward, while [12] choose deterministic rounding for $\mathbf{X}$ in the backward, which may lead to bias in gradient expectation.
3. **The block shape of weights**: We choose a more accurate 1×16 shape, while NVIDIA [12] adopts 16×16 as weight group shape. Our non-square block shape also needs alignment for $\widehat{\mathbf{W}}$ in forward and backward, so $\widehat{\mathbf{W}}$ is unbiasedly estimated in the computation of $\mathrm{d}\mathbf{X}$.

Our design ensures the unbiasedness of gradients estimation according to a similar analysis in TetraJet [22], theoretically ensuring the convergence of SGD [30]. This results in better performance in the `NVFP4` training compared to previous work. We show the superiority of our design in Sec. 6.2.

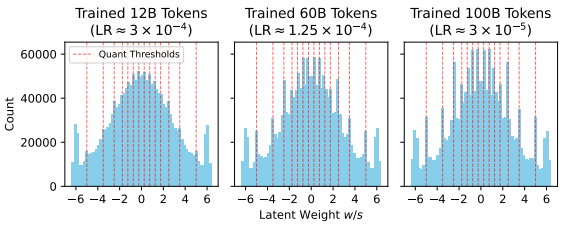

Figure 1: The distribution of latent weight w/s in `OLMo2-150M blocks.11.att_proj` in `NVFP4` training without oscillation suppression.

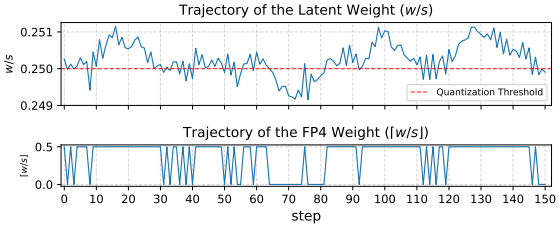

Figure 2: Optimization trajectory of one oscillating weight in `OLMo2-150M blocks.11.att_proj` near the end of `NVFP4` training.

4 OsciReset: Oscillation Suppression for Weight Optimization

4.1 Oscillation Phenomenon in LLMs Pre-training

Although the unbiased gradient makes `NVFP4` training feasible, the optimization of low-precision weights is not the same as that of high-precision weights. In the final stage of low-precision training, the learning rate (LR) drops to near zero, so the model can quickly descend to a local minimum for convergence. However, we found that when the master weights converge as the LR is approaching zero, there are still drastic changes among the quantized weights. This phenomenon is *weight oscillation*, which widely exists in quantized training.

To demonstrate this phenomenon, we plot the distribution of *latent weight* w/s for linear weight parameters during `NVFP4` training, where s is w's quantization scaling factor. According to the distribution in Fig. 1, there is a growing number of latent weights lying close to the quantization decision threshold. It means that more and more weights are prone to switching quantization values. This trend explains the drastic changes the quantized model exhibits at the end of training.

For example, $q_1 = 0, q_2 = 0.5$ are two `FP4` values. The latent weights near the threshold $\frac{q_1+q_2}{2} = 0.25$ would be sensitive to small perturbations. Assume $w/s \in (0.25, 0.25 + \varepsilon)$, where ε is a small number. The corresponding quantized weight is $\lceil w/s \rfloor = 0.5$. Even a tiny gradient step could make w/s jump to $w/s \in (0.25 - \varepsilon, 0.25)$, resulting in a giant leap to $\lceil w/s \rfloor = 0$, and vice versa. This process can frequently repeat for the weights near the threshold. In Fig. 2, we can see a trajectory of an oscillating weight — the latent value w/s is always near and frequently crossing the threshold $\frac{q_1+q_2}{2} = 0.25$, so the quantized values $\lceil w/s \rfloor$ switch frequently between $q_1 = 0$ and $q_2 = 0.5$.

If we want to reach a low-bit model ultimately with low-precision training, the oscillation phenomenon needs to be addressed. The oscillating weights cannot converge well to a determined quantized value, becoming a potential detriment to the global optimization. Therefore, we aim to identify the specific elements responsible for oscillations during LLM training, and apply additional mechanisms to suppress oscillations, thereby improving the overall optimization performance.

4.2 Identifying Oscillating Weights

Following the previous oscillation detection method [22] for Vision Transformers, we identify weight oscillation through tracking the optimization trajectory of each weight element. During our tracking in an interval $t = 0, 1, 2, \ldots, T_0$, we record the optimization distance of master weights and quantized weights, respectively, for each weight element:

$$\text{dist}_M(w) = \sum_{t=1}^{T_0} |w^{(t)} - w^{(t-1)}|, \quad \text{dist}_Q(w) = \sum_{t=1}^{T_0} |Q(w^{(t)}) - Q(w^{(t-1)})|$$

The risk of oscillation can be defined as $\text{OsciRisk}(w) = \text{dist}_Q(w)/\text{dist}_M(w)$. A larger $\text{OsciRisk}(w)$ indicates that the weight element w oscillates because its quantized weight is switching frequently ($\text{dist}_Q(w)$ is relatively large) but its master weight is moving in small steps ($\text{dist}_M(w)$ is relatively small). We represent the calculation process in Alg. 1.

Algorithm 1: UpdateOscillationStats(θ, t_0)

Input: θ: model parameters; t_0: step in a period.
Output: $\text{dist}_M, \text{dist}_Q$: updated oscillation statistics.
Data: $\text{dist}_M, \text{dist}_Q$: accumulated statistics; w': weight snapshots.

if $t_0 = 0$ **then**
  for i-th Element w_i **in** Quantized Parameters of θ **do**
    Initialize statistics: $\text{dist}_M(w_i) \leftarrow 0$, $\text{dist}_Q(w_i) \leftarrow 0$;
    Build a record for w_i: $w'_i \leftarrow w_i$
else
  for i-th Element w_i **in** Quantized Parameters of θ **do**
    Update statistics:
      $\text{dist}_M(w_i) \leftarrow \text{dist}_M(w_i) + |w_i - w'_i|$;
      $\text{dist}_Q(w_i) \leftarrow \text{dist}_Q(w_i) + |Q(w_i) - Q(w'_i)|$;
    Update record: $w'_i \leftarrow w_i$;
return $\text{dist}_M, \text{dist}_Q$

Algorithm 2: OscillationSuppress(θ, dist_M, dist_Q, τ_{osci})

Input: θ: model parameters;
  τ_{osci}: oscillation risk threshold;
  $\text{dist}_M, \text{dist}_Q$: oscillation statistics.
Output: Modified parameters θ

for i-th Element w_i **in** Quantized Parameters of θ **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` scale_i `is the scaling factor.`
    $w_i \leftarrow w_{\text{FP4},i} \cdot \text{scale}_i$
return θ;

4.3 Reducing Weight Oscillation by Resetting Master Weights

After identifying those special elements prone to oscillation, we would like to stabilize them for improving convergence. This is a challenging task, because suppressing oscillation would likely harm global optimization, which is why previous methods for Vision Transformers cannot generalize well to LLM training (as shown in Sec. 6.2: Ablation Study). We need a method to effectively stabilize the oscillation, and meanwhile, to maintain a good optimization for all the parameters.

Our novel stabilization method solves this problem by simply resetting the oscillating weight to the center of the quantization bin where it is lying, which is Alg. 2.

We adopt this algorithm when the learning rate decays to a relatively low value and oscillation problems emerge. The algorithm has the following effects:

- The quantized weights are not affected right after the master weights are reset, ensuring *no performance degradation* from the resetting.
- If there is not such resetting, oscillating elements would keep fluctuating between two fixed bins, so *resetting* can provide more chances to be optimized to other values. This is better than *freezing*, which results in these parameters never being updated in the future.

In this way, the oscillating weights, which are detrimental to optimization, can be *reset* to follow a new optimization trajectory less prone to oscillation, resulting in improved accuracy.

Differing from previous methods, our method is a novel and simple technique to reduce the negative impact of weight oscillation effectively. Previous research on oscillation focuses only on Vision Transformers [20, 24, 22], and some of the work can only be applied to fine-tuning on a full-precision checkpoint,

Algorithm 3: Trainer with Periodic Oscillation Suppression through Resetting (**OsciReset**)

Input: θ: initialized model parameters; $\mathcal{D}$: batched data; τ_{osci}: oscillation risk threshold; $T_{\max}$: total number of training steps; T_{start}: suppression start step; T_{period}: suppression period length; T_{accu}: steps for oscillation detection.

Output: Updated parameters θ_T

```
for t ← 1 to T_max do
    Compute batch loss: L(θ, D_i);
    Compute gradients: ∇_θ L;
    Update parameters: θ ← OptimizerStep(θ, ∇_θ);
    if t ≥ T_start then
        if t mod T_period ≤ T_accu then
            dist_M, dist_Q ← UpdateOscillationStats(θ, t mod T_period);
        end
        if t mod T_period = T_accu + 1 then
            OscillationSuppress(θ, dist_M, dist_Q, τ_osci);
        end
return θ.
```

meanwhile requiring careful tuning on the hyperparameters, while our method can be directly useful for pre-training LLMs from scratch, and needs little tuning on hyperparameters.

Combining the suppression algorithm, the refined trainer **OsciReset** can be implemented as Alg. 3.

5 OutControl: Outlier Control for Activation and Gradients

5.1 Random Hadamard Transformation for Quantized Backpropagation

Large *outliers* are widely present in the activations and gradients of LLMs, but are harmful for quantization. Due to the presence of outliers which makes the scaling factor large, small numbers in the quantization group would be scaled down near zero, and cannot be represented accurately through low-precision formats like `FP4`.

Hadamard Matrix A common way to reduce outliers is performing orthogonal rotation on the tensors with the Hadamard matrix [31]. The Hadamard matrix $H_d (d = 2^k, k \in \mathbb{Z}_+)$ is an orthogonal matrix defined by

$$H_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad H_{2d} = \frac{1}{\sqrt{2}} H_2 \otimes H_d = \frac{1}{\sqrt{2}} \begin{bmatrix} H_d & H_d \\ H_d & -H_d \end{bmatrix}$$

Intuitively, a Hadamard rotation makes the distribution of a vector more even, mitigating large outliers. In practice, we use a block Hadamard $\mathbf{H}_n = \text{diag}\{H_d, H_d, \cdots, H_d\} \in \mathbb{R}^{n \times n}$ $(d \in \{16, 32, \cdots\}, d|n)$ instead of a full Hadamard matrix to avoid heavy computation in the Hadamard Transformation [16].

Random Hadamard Transformation To control the quantization variance in the backward pass, we follow [17] and apply a Random Hadamard Transformation (RHT). The RHT performs a random sign flipping before Hadamard Transformation: $\mathbf{A}' \leftarrow \mathbf{A}\mathbf{S}_n\mathbf{H}_n$, where $\mathbf{S}_n \in \text{diag}\{\pm 1\}^n$ is randomly generated. Applying RHT simultaneously to both operands of a matrix multiplication (`MM`) results in an *equivalent* result since $\mathbf{H}_n$ and $\mathbf{S}_n$ are orthogonal: $\mathbf{A}\mathbf{B}^T = (\mathbf{A}\mathbf{S}_n\mathbf{H}_n)(\mathbf{B}\mathbf{S}_n\mathbf{H}_n)^T$. Therefore, we can alleviate matrices' outliers through RHT before low-precision `MM` as: $\mathbf{A}\mathbf{B}^T \approx Q(\mathbf{A}\mathbf{S}_n\mathbf{H}_n)Q(\mathbf{B}\mathbf{S}_n\mathbf{H}_n)^T$.

As described in Sec. 3.3, there are one `MM` in the forward and two `MM`s in the backward of a linear layer. Practically, we apply RHT for the two `MM`s in backward, the computation of d$\mathbf{X}$ and d$\mathbf{W}$. This design is based on our empirical findings (Sec. 6.2). We do not adopt the Hadamard transformation in the forward pass like [18] because we observe that Hadamard transformation to the forward pass is harmful for the optimization. We believe that the additional error from RHT exceeds the outlier reduction effect in the forward. Moreover, while [12] suggests RHT brings benefits only for d$\mathbf{W}$, our experiments yield a

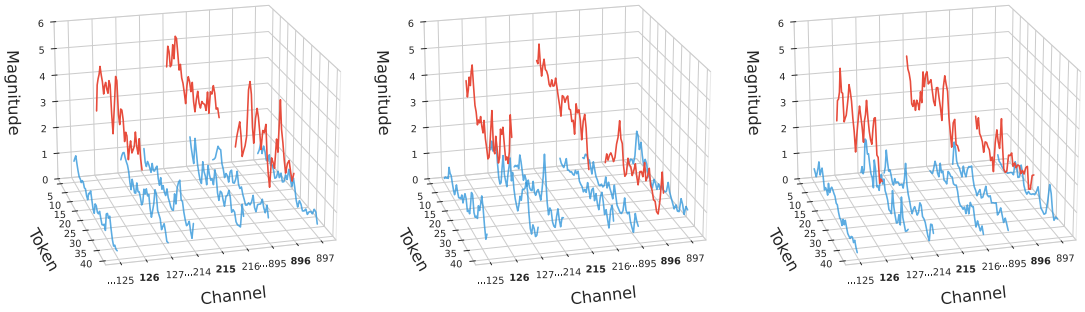

Figure 3: Activation magnitudes of MLP input at layer 10 for different `GSM8K` samples across `OLMo2-370M` training checkpoints at different steps. Outliers consistently appear in specific channels.

different insight that RHT would improve both computation of d$\mathbf{X}$ and d$\mathbf{W}$. Therefore, we apply RHT to all `MM`s in backward passes in `NVFP4` linear layers.

5.2 Precision Retaining for Activation Outliers

Besides adopting RHT in the backward pass to improve the gradient calculation, we observe that RHT performs poorly in the forward pass compared to without RHT as described above. To overcome this limitation and further boost performance, we introduce a method based on retaining the precision of certain outliers in activations.

We note that the most obvious outlier pattern in activations is *structural*. This means that a small number of activation channels *consistently* exhibit much larger variance (higher norms) than others across different inputs and training steps, as shown in Fig. 3. Therefore, we can leverage this fixed structural pattern to statically select these persistent outlier channels and retain them in higher precision (e.g., `FP8`, `BF16`). Moreover, for the backward pass, we further combine it with the RHT technique to further control outliers beyond selected channels, ensuring additional gradient accuracy.

This method is inspired by `LLM.int8()` [27], a PTQ technique that identifies and retains activation outlier channels during inference. However, unlike `LLM.int8()`, our method applies to the fully low-precision training, simultaneously improving the accuracy of forward and backward passes.

In detail, we select p% activation channels $\mathbf{X}$ (the index set is $\mathcal{A}$) and use $\bar{\mathcal{A}}$ to represent the complement indices. Typically, we choose $p = 5\%$ in `BF16` formats or $p = 10\%$ in `FP8` formats. The forward and backward passes are then computed as follows:

$$
\begin{aligned}
\mathbf{Y} &= Q_D^{(1)}\left(\mathbf{X}_{:,\bar{\mathcal{A}}} \quad \mathbf{0}\right) \times Q_D^{(2)}(\mathbf{W}^\top) + \underline{\left(\mathbf{0} \quad \mathbf{X}_{:,\mathcal{A}}\right) \times Q_D^{(2)}(\mathbf{W}^\top)}, \\
\mathrm{d}\mathbf{X} &= Q_S^{(3)}\left(\mathrm{d}\mathbf{Y}\mathbf{S}_C\mathbf{H}_C\right) \quad \times Q_S^{(4)}\left(\mathbf{H}_C^\top\mathbf{S}_C^\top\widehat{\mathbf{W}}\right), \\
\mathrm{d}\mathbf{W}_{:,\bar{\mathcal{A}}} &= Q_S^{(5)}\left(\mathrm{d}\mathbf{Y}^\top\mathbf{S}_N\mathbf{H}_N\right) \times Q_S^{(6)}\left(\mathbf{H}_N^\top\mathbf{S}_N^\top\widehat{\mathbf{X}}_{:,\bar{\mathcal{A}}}\right), \\
\mathrm{d}\mathbf{W}_{:,\mathcal{A}} &= \underline{\mathrm{d}\mathbf{Y}^\top \times \widehat{\mathbf{X}}_{:,\mathcal{A}}}
\end{aligned}
$$

where $\mathbf{X} \in \mathbb{R}^{N\times D}, \mathbf{W} \in \mathbb{R}^{C\times D}, \mathbf{Y} \in \mathbb{R}^{N\times C}$. Here we denote $\widehat{\mathbf{X}} := Q_D^{(1)}\left(\mathbf{X}_{\bar{\mathcal{A}}} \quad \mathbf{0}\right) + \left(\mathbf{0} \quad \mathbf{X}_{\mathcal{A}}\right), \widehat{\mathbf{W}} := Q_D^{(2)}(\mathbf{W}^\top)^\top$, and underline the high-precision multiplication with red color.

As mentioned above, our outlier selection for $\mathbf{X}$ is offline, since the outlier channel pattern remains consistent across different inputs and training steps throughout the training, which echoes the observations in [32]. This consistency enables us to statically select outlier channels at the starting stage of training, removing the need for dynamic adjustments for each input batch or training step. Besides, this consistency stabilizes the model structure by fixing the selected channels and aligns the model for inference with the training model. Compared to previous outlier selection methods like [19], which require input-specific dynamic adjustments, our static selection method incurs minimal computational overhead in maintaining the outlier structure.

Table 1: Training and validation perplexity of OLMo2 pre-training with different FP4 training methods. The 70M/150M/370M models are trained from scratch with 50B/100B/200B tokens.

	Train PPL			Validation PPL		
Methods \ OLMo2-Size	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**

Table 2: Performance on downstream tasks of OLMo-2 370M trained for 200B tokens with different FP4 training methods.

Methods	ARC-e	ARC-c	BoolQ	CQA	Hella.	MMLU	OBQA	PIQA	SIQA	Wino.	Avg.↑	Wiki.	Pile
	acc↑	acc_n↑	acc↑	acc_n↑	acc_n↑	acc_n↑	acc_n↑	acc_n↑	acc_n↑	acc↑		PPL↓	PPL↓
BF16	55.96	28.43	54.49	36.60	43.30	25.85	33.40	68.66	42.78	51.86	44.13	16.86	12.21
Quartet	53.04	27.43	56.54	34.23	39.89	24.12	**32.20**	67.25	42.32	50.91	42.79	19.03	13.53
NVIDIA	52.11	26.76	56.48	34.72	39.81	25.10	31.20	65.99	**44.11**	**51.46**	42.77	19.06	13.52
TetraJet-v2-base(ours)	54.56	**27.75**	**59.32**	35.38	40.53	23.75	31.00	**67.57**	42.89	51.32	43.41	18.52	13.16
TetraJet-v2-full (ours)	**54.74**	24.75	59.23	**37.18**	**41.11**	**26.08**	31.60	66.59	43.65	51.06	**43.60**	**18.06**	**12.81**

6 Experiments

6.1 Pre-Training LLMs with NVFP4

Model and Data Settings Based on the official open-source code base of OLMo[1], we pretrained OLMo2 models [23] with 70M, 150M, and 370M parameters[2], from scratch for 50B, 100B, and 200B tokens respectively, on the open-sourced OLMo-2-Mix-1124 dataset[3]. We adopt the AdamW optimizer with a cosine-decay learning-rate scheduler with warm-up for all pre-training. The details of the models and training recipes are listed in Appendix A.1.

NVFP4 Algorithm Settings Following prior work on low-precision training [22, 16], we focus on evaluating the quantization algorithm for the forward and backward process on the *linear* layers. We quantized activation, weight, and gradients to achieve *Fully Quantized Training* (FQT). We quantize *all* linear layers in all Transformer [33] blocks for each quantized training method for a fair comparison.

We compared our method with recent state-of-the-art approaches with their default settings: the MXFP4 training method Quartet [18], and NVIDIA's NVFP4 training recipe [12]. For our methods, we trained with two different recipes for a clearer comparison (We list the detailed hyperparameters of our algorithms in Appendix A.2): (1) TetraJet-v2-base: combines our NVFP4 layer with RHT in backpropagation; (2) TetraJet-v2-full: adds oscillation suppression algorithm **OsciReset** and the outlier accuracy retaining algorithm **OutControl** over TetraJet-v2-base.

Results We present the training perplexity (averaged on the last 50 steps) and the validation perplexity on the C4 dataset [34] in Tab. 1. We observe that TetraJet-v2-full reaches the lowest validation perplexity, further closing the gap to high-precision training. For the downstream evaluation, we reported the results in Tab. 2 on the following benchmarks: ARC [35], BoolQ [36], Commonsense QA (CQA) [37], Hellaswag [38], MMLU [39], Open Book QA (OBQA) [40], PIQA [41], SocialIQA (SIQA) [42], and Winogrande [43]. We also report the validation perplexity on Wikitext-103 [44] and Pile [45]. TetraJet-v2-full reaches the highest average performance across all benchmarks.

[1] https://github.com/allenai/OLMo

[2] The number of parameters here is excluding embeddings.

[3] https://huggingface.co/datasets/allenai/olmo-mix-1124

Table 3: Ablation study on `OLMo2-150M` model for `NVFP4` linear layer design. We reported training PPL averaged on the last 50 steps. (a) Ablation on block-size settings trained for `50B` tokens based on TetraJet-v2-`base`. (b) Ablation on the unbiasedness of gradients trained for `50B` tokens. (StoQ.: stochastic quantization in d$\mathbf{W}$ compute) (c) Ablation on Hadamard Transform trained for `100B` tokens. (Fwd.: adopt HT in forward; dX/dW: adopt RHT in dX/dW computation of backpropagation)

(a)

Outer-Block Size	PPL
Per-Tensor	31.75
Per-Row	31.58
1×128	**31.49**
Weight-Inner-Block Size	**PPL**
16×16	31.74
1×16	**31.49**

(b)

Method	PPL
NVIDIA	32.42
w/o $\widehat{\mathbf{X}}$ align and StoQ.	31.80
w/o $\widehat{\mathbf{X}}$ align	31.78
TetraJet-v2-base	**31.49**

(c)

Fwd.	dX	dW	PPL
✓	✗	✗	28.89
✗	✗	✗	28.67
✗	✗	✓	28.47
✗	✓	✓	**28.42**

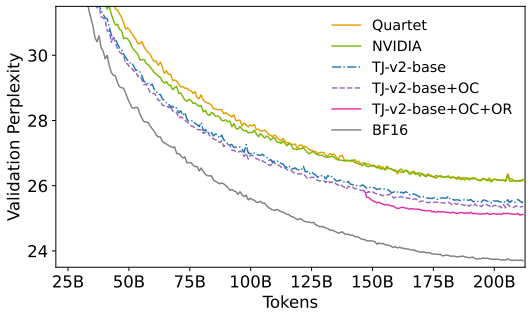

Figure 4: Validation loss curve of `OLMo2-370M` with about `200B` tokens for comparing different methods. (TJ: training method TetraJet-v2; OC: OutControl; OR: OsciReset)

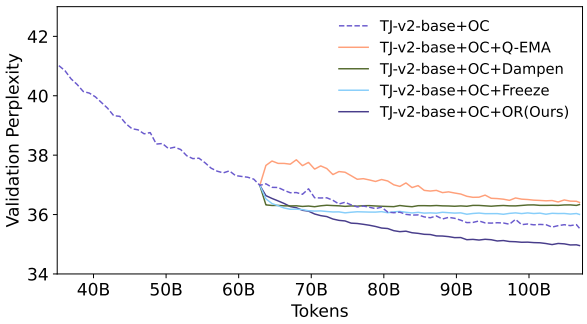

Figure 5: Validation loss curve of `OLMo2-150M` with about `100B` tokens for comparing different oscillation suppressing techniques. We set $T_{\text{start}} \approx$ `65B` tokens for all methods to begin suppressing oscillation.

6.2 Ablation Studies

6.2.1 Basic NVFP4 Linear Layer Setting

The Scaling Style of NVFP4 Quantization As described in Sec. 3, we set an outer-block scaling outside `NVFP4`'s micro-block to satisfy the numerical constraints, and our finer outer-block is more accurate; we adopt 1×16 group shape for $\mathbf{W}$ rather than 16×16 in [12]. In Tab. 3a, we reveal that both of our settings are better. We find that the per-tensor outer scaling would be detrimental, while the 1×128 outer scaling or per-row scaling would be similarly more accurate. Furthermore, 1×16 group shape outperforms 16×16 for weight quantization.

The Unbiasedness of Gradient Estimation In Tab. 3b, we demonstrate the effectiveness of our more refined designs in unbiased backpropagation of `NVFP4` linear layer compared to [12]. We surpass [12] through the alignment of $\widehat{\mathbf{X}}$ in forward/backward and the stochastic rounding, as described in Sec. 3.

The Effect of Hadamard Transform We find that applying Hadamard Transformation (HT) for forward is harmful, while for the backward is beneficial. Moreover, while [12] suggests RHT brings benefits only for d$\mathbf{W}$, we see in Tab. 3c that under our unbiased linear framework, the Random Hadamard Transformation (RHT) would improve both computations of d$\mathbf{X}$ and d$\mathbf{W}$.

In conclusion, our `NVFP4` linear surpasses NVIDIA's `NVFP4` design [12] with a more refined scaling method and unbiased gradient estimation as suggested in Sec. 3.

6.2.2 Improvement of Oscillation Suppression and Outlier Control

The Combination of Methods We show in the validation loss curve in Fig. 4 that our methods OsciReset and OutControl can be finely combined. From the figure, we see that **OutControl** can improve accuracy throughout the training process, and the oscillation suppression algorithm **OsciReset** could effectively alleviate oscillation at $T_{\text{start}} \approx$ `150B` tokens to bring additional enhancement.

Ablations on Oscillation Suppression and Outlier Selection The difficulty of suppressing oscillation in LLMs lies in simultaneously controlling the oscillating weights and maintaining the effect of global optimization. We tested several prior methods that work on Vision Transformers: (1) "Q-EMA" [22] that adopts EMA to smooth weight quantization; (2) "Freeze" [20] that tracks weight oscillation frequency and freezes frequently oscillating weights to the running average; (3) "Dampen" [20] that adds a penalty term $\mathcal{L}_{\text{dampen}} = \lambda \|\mathbf{W} - Q(\mathbf{W})\|^2$ into loss to encourage weights to be away from quantization thresholds. In Fig. 5, we found that they all present severe detriment to the final performance, while our method OsciReset consistently maintains a substantial improvement to the optimization.

Finally, we validate the effectiveness of the outlier selection method OutControl. In Tab. 4, we show that our static choice of the channels is effective and the method improves both the forward and backward.

6.3 Further Studies and Discussions

The Effectiveness of Oscillation Suppression To prove our method OsciReset truly reduced the oscillation, we compute the metric OsciRisk(w) defined in Sec. 4.2 to identify the proportion of oscillation weights throughout the training process. In Fig. 6, there would be a drastic increase of oscillation weights if we do not suppress them, and we can observe a substantial decay to the oscillation when applying OsciReset, which confirms the effect of our method.

Loss Decomposition After developing our accurate training method TetraJet-v2-full, with weight oscillation reduction and outlier controlling, we would like to know which quantizer harms most to the final accuracy, that is, which component of the computation in the forward/backward pass of the linear layer is most sensitive to the quantization. Also, we want to discover which component in the Transformer block is the most sensitive.

In Tab. 5, we find that in the forward pass, the activation would take the most responsibility for the accuracy loss after we adopt weight oscillation. When we only quantize one of the computations of $\mathrm{d}\mathbf{X}$ or $\mathrm{d}\mathbf{W}$ in backpropagation, it seems lossless. However, if we combine both quantization for gradients, the error would accumulate, leading to a larger accuracy loss. Finally, we conclude that the backward pass is similarly important as the weight quantization, and the activation quantization would still be the most crucial bottleneck.

In Tab. 6, linear modules in MLP yields higher sensitivity than those in Attentions. And retaining `mlp.ffn2` in high-precision brings more improvement than others, which means that this layer, just after the SwishGLU in the transformer block, is more sensitive to quantization. This may attribute to the larger outliers brought by SwishGLU. We hope the conclusions could inspire future works on low-precision training on how to further improve the performance. We left the decomposition experiments on varying model/data size to future work.

Precision Switching in Final Stage Currently, it is quite difficult to achieve lossless with fully `FP4` $\times$ `FP4` quantization training for all linear layers. We want to see how much we can improve the accuracy by introducing some more bits (e.g., `FP6` $\times$ `FP4`, `FP6` $\times$ `FP6`), although these formats are not yet the focus of current mainstream hardware and algorithms. Here, we simulated a fully `FP6` $\times$ `FP4` linear layer as follows:

$$\mathbf{Y} = \widehat{\mathbf{X}} \times \widehat{\mathbf{W}^\top}, \quad \widehat{\mathbf{X}} := Q_D^{\mathbf{FP6}}(\mathbf{X}), \quad \widehat{\mathbf{W}^\top} := Q_D^{\mathbf{FP4}}(\mathbf{W}^\top)$$
$$\mathrm{d}\mathbf{X} = Q_S^{\mathbf{FP6}}(\mathrm{d}\mathbf{Y}) \times Q_S^{\mathbf{FP4}}\left(\widehat{\mathbf{W}}\right), \quad \mathrm{d}\mathbf{W} = Q_S^{\mathbf{FP4}}\left(\mathrm{d}\mathbf{Y}^\top\right) \times Q_S^{\mathbf{FP6}}\left(\widehat{\mathbf{X}}\right)$$

In Fig. 7, we find that a substantial improvement can be achieved if we just switch to `FP6` $\times$ `FP4` in the last `50B` tokens of `OLMo2-370M` training; also, switching to `FP6` $\times$ `FP6` could bring considerable improvement. This result may be inspiring for future study.

Table 4: Ablation on `150M` model for OutControl (Fwd. / Bwd.: apply OutControl in fwd./bwd.).

Fwd.	Bwd.	PPL
✗	✗	31.49
✓	✗	31.41
✓	✓	**31.28**
Selection Style		PPL
None		31.49
Random		31.47
Largest norms		**31.28**

Table 5: Loss decomposition analysis for quantizers on `OLMo2-370M` for `50B` tokens (Only `fwd.X`: only quantize X in forward to FP4).

Settings	PPL	ΔPPL
BF16	24.06	0
Only `fwd.X`	24.50	+0.43
Only `fwd.W`	24.28	+0.22
Only `bwd.dX`	24.13	+0.07
Only `bwd.dW`	24.05	−0.01
Only `bwd.dX+dW`	24.27	+0.21
Quant All	24.89	+0.83

Table 6: Loss decomposition analysis for modules on `OLMo2-370M` for `50B` tokens (w/o `qkv`: only retain module qkv in `BF16`).

Settings	PPL	ΔPPL
Quant All	24.89	0
w/o `qkv`	24.74	−0.15
w/o `attn.out`	24.73	−0.16
w/o `mlp.ffn1`	24.60	−0.29
w/o `mlp.ffn2`	24.54	−0.35
All `BF16`	24.06	−0.83

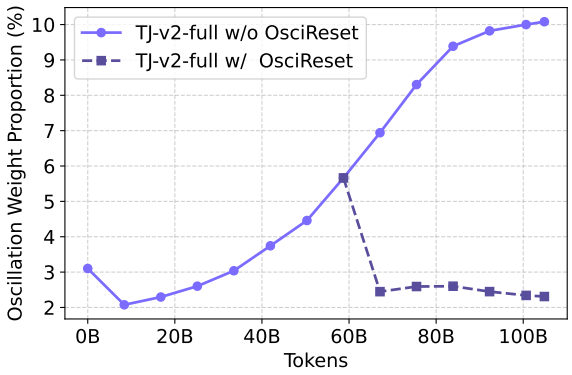

Figure 6: The change of the oscillation proportion with/without OsciReset on `OLMo2-150M`. We count a weight w as oscillating if OsciRisk(w) > 16.

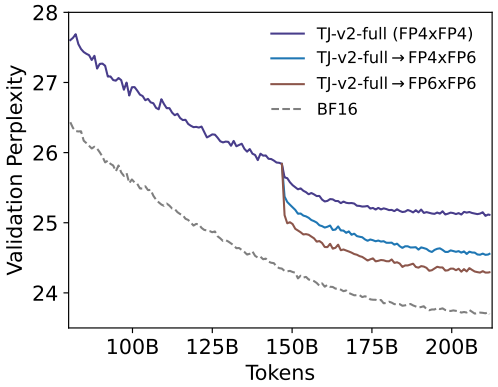

Figure 7: Validation PPL for the precision switch on `OLMo2-370M`. (We start suppressing oscillation at $T_{\text{start}} \approx$ `150B`, so the curve drops for `TJ-v2-full`)

7 Conclusions and Limitations

In this work, we propose **TetraJet-v2**, an end-to-end fully-quantized `FP4` training recipe for LLMs. Our experiments demonstrate that our unbiased double-block quantization, weight oscillation reduction, and outlier precision control methods enable stable and accurate 4-bit training that consistently outperforms the previous state of the art. We analyze and effectively address the problem of oscillating weights during ultra-low-precision LLM training. Results show that combining our proposed methods offers significant advantages for bridging the performance gap to full-precision training.

Due to limited compute resources, our experiments focus on model sizes up to `370M` parameters and data scales up to `200B` tokens. Furthermore, due to the unavailability of hardware that supports low-precision `NVFP4` computations, we cannot evaluate the effective speedups and improvements in compute efficiency that our methods enable. We hope that future work will implement our proposed algorithms on practical low-precision hardware, enabling more efficient training of larger-scale LLMs.

References

[1] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025.

[2] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[3] Nestor Maslej, Loredana Fattorini, Raymond Perrault, Yolanda Gil, Vanessa Parli, Njenga Kariuki, Emily Capstick, Anka Reuel, Erik Brynjolfsson, John Etchemendy, et al. Artificial intelligence index report 2025. *arXiv preprint arXiv:2504.07139*, 2025.

[4] Sharan Narang, Gregory Diamos, Erich Elsen, Paulius Micikevicius, Jonah Alben, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. In *Int. Conf. on Learning Representation*, 2017.

[5] Dhiraj D. Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, Jiyan Yang, Jongsoo Park, Alexander Heinecke, Evangelos Georganas, Sudarshan Srinivasan, Abhisek Kundu, Misha Smelyanskiy, Bharat Kaul, and Pradeep Dubey. A study of bfloat16 for deep learning training. *CoRR*, abs/1905.12322, 2019.

[6] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[7] Houwen Peng, Kan Wu, Yixuan Wei, Guoshuai Zhao, Yuxiang Yang, Ze Liu, Yifan Xiong, Ziyue Yang, Bolin Ni, Jingcheng Hu, et al. Fp8-lm: Training fp8 large language models. *arXiv preprint arXiv:2310.18313*, 2023.

[8] Jack Choquette. NVIDIA Hopper H100 GPU: Scaling performance. *IEEE Micro*, 43(3):9–17, 2023.

[9] Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, et al. Microscaling data formats for deep learning. *arXiv preprint arXiv:2310.10537*, 2023.

[10] Eduardo Alvarez, Omri Almog, Eric Chung, Simon Layton, Dusan Stosic, Ronny Krashinsky, and Kyle Aubrey. Introducing NVFP4 for efficient and accurate low-precision inference. Technical report, NVIDIA, 2025.

[11] Ajay Tirumala and Raymond Wong. NVIDIA Blackwell platform: Advancing generative AI and accelerated computing. 2024.

[12] NVIDIA, Felix Abecassis, Anjulie Agrusa, Dong Ahn, Jonah Alben, Stefania Alborghetti, Michael Andersch, Sivakumar Arayandi, Alexis Bjorlin, Aaron Blakeman, Evan Briones, et al. Pretraining large language models with nvfp4. *arXiv preprint arXiv:2509.25149*, 2025.

[13] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 100213–100240. Curran Associates, Inc., 2024.

[14] Xiao Sun, Naigang Wang, Chia-Yu Chen, Jiamin Ni, Ankur Agrawal, Xiaodong Cui, Swagath Venkataramani, Kaoutar El Maghraoui, Vijayalakshmi Viji Srinivasan, and Kailash Gopalakrishnan. Ultra-low precision 4-bit training of deep neural networks. *Advances in Neural Information Processing Systems*, 33:1796–1807, 2020.

[15] Brian Chmiel, Ron Banner, Elad Hoffer, Hilla Ben-Yaacov, and Daniel Soudry. Accurate neural training with 4-bit matrix multiplications at standard formats. In *The Eleventh International Conference on Learning Representations*, 2023.

[16] Haocheng Xi, Changhao Li, Jianfei Chen, and Jun Zhu. Training transformers with 4-bit integers. *Advances in Neural Information Processing Systems*, 36:49146–49168, 2023.

[17] Albert Tseng, Tao Yu, and Youngsuk Park. Training llms with mxfp4. In Yingzhen Li, Stephan Mandt, Shipra Agrawal, and Emtiyaz Khan, editors, *Proceedings of The 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pages 1630–1638. PMLR, 03–05 May 2025.

[18] Roberto L. Castro, Andrei Panferov, Rush Tabesh, Jiale Chen, Oliver Sieberling, Mahdi Nikdan, Saleh Ashkboos, and Dan Alistarh. Quartet: Native FP4 training can be optimal for large language models. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*, 2025.

[19] Ruizhe Wang, Yeyun Gong, Xiao Liu, Guoshuai Zhao, Ziyue Yang, Baining Guo, Zheng-Jun Zha, and Peng Cheng. Optimizing large language model training using FP4 quantization. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 62937–62957. PMLR, 13–19 Jul 2025.

[20] Markus Nagel, Marios Fournarakis, Yelysei Bondarenko, and Tijmen Blankevoort. Overcoming oscillations in quantization-aware training. In *International Conference on Machine Learning*, pages 16318–16330. PMLR, 2022.

[21] Mingjie Sun, Xinlei Chen, J. Zico Kolter, and Zhuang Liu. Massive activations in large language models. *arXiv preprint arXiv:2402.17762*, 2024.

[22] Yuxiang Chen, Haocheng Xi, Jun Zhu, and Jianfei Chen. Oscillation-reduced MXFP4 training for vision transformers. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 9400–9414. PMLR, 13–19 Jul 2025.

[23] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 olmo 2 furious, 2024.

[24] Shih-Yang Liu, Zechun Liu, and Kwang-Ting Cheng. Oscillation-free quantization for low-bit vision transformers. In *International Conference on Machine Learning*, pages 21813–21824. PMLR, 2023.

[25] Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, et al. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. *arXiv preprint arXiv:2505.06708*, 2025.

[26] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024.

[27] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.

[28] Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: LLM quantization with learned rotations. In *The Thirteenth International Conference on Learning Representations*, 2025.

[29] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

[30] Jianfei Chen, Yu Gai, Zhewei Yao, Michael W Mahoney, and Joseph E Gonzalez. A statistical framework for low-bitwidth training of deep neural networks. *Advances in neural information processing systems*, 33:883–894, 2020.

[31] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[32] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[34] Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2021.

[35] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[36] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

[37] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, 2019.

[38] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019.

[39] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[40] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, 2018.

[41] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[42] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

[43] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740, 2020.

[44] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

[45] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020.

A Hyperparameters for Models and Training Recipes

A.1 Hyperparameters for Model

Table 7: Model Hyperparameters in Sec. 6.1: Pre-Training LLMs with NVFP4

Model	OLMo2-70M	OLMo2-150M	OLMo2-370M
# Parameters (non-embedding)	72,368,640	147,886,848	371,262,464
# Parameters (all)	123,748,864	224,957,184	474,022,912
Hidden Size	512	768	1024
# Heads	8	12	16
# Layers	8	12	16
# MLP Hidden Size	2048	3072	8192
Vocab Size	100278		
Sequence Length (N)	4096		
Batch Size (BS)	1024		
Learning Rate (LR)	4×10^{-4}		
Learning Rate Scheduler	Cosine Decay with Warm-up		
Weight Decay	0.1		
# Trained Steps	12500	25500	50500
# Warm-up Tokens	1 B	8 B	8 B
# Trained Tokens (=Steps×BS×N)	52 B	107 B	212 B

A.2 Hyperparameters for NVFP4 Training Techniques

Table 8: Hyperparameters for NVFP4 Training Techniques

Technique	Hyperparameters	OLMo2-70M	OLMo2-150M	OLMo2-370M
RHT	Hadamard Block Size	32		
OsciReset	$T_{\max}$: total number of training steps	12500	25500	50500
	T_{start}: suppression start step	8000	15000	35000
	T_{period}: suppression period length	200		
	T_{accu}: accumulated steps	50		
	τ_{osci}: oscillation risk threshold	8		
OutControl	Outlier Compute Type	FP8		
	Outlier Selected Ratio	10%		