FLASH EQ-LINEAR: ACCELERATING EQUIVARIANT LINEAR LAYERS VIA GROUP-WISE DISCRETE FOURIER TRANSFORM

Anonymous authors
Paper under double-blind review

ABSTRACT

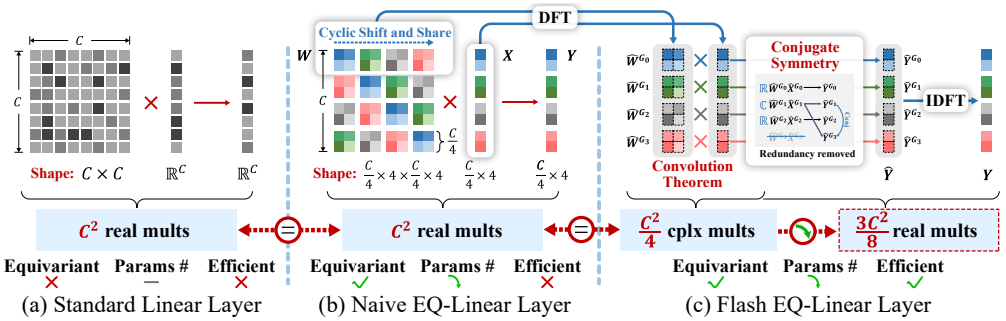
Equivariant networks embed geometric symmetries as structural priors through weight sharing, achieving remarkable parameter efficiency across vision tasks. However, this parameter efficiency does not translate into compute efficiency: most existing implementations unroll the structured weights into dense matrices and dispatch them to generic dense kernels, so an equivariant layer costs no fewer MACs than its non-equivariant counterpart. In this paper, we observe that the equivariant linear (EQ-Linear) layer—the most fundamental and frequently used module in modern equivariant architectures—is essentially a circular convolution along the group dimension composed with a linear transform along the channel dimension. Building on this observation, we propose **Flash EQ-Linear**, an exact acceleration algorithm that reduces the cost to $2(T-1)/T^2$ of the original dense formulation (T is the equivariant group size) by combining the Fourier convolution theorem along the group dimension with the conjugate symmetry of the real DFT. To translate these computational savings into wall-clock speedups, we further develop dedicated CUDA kernels for the $\mathrm{p4}$ group. At the operator level, Flash EQ-Linear achieves up to $\mathbf{2.1}\times$ forward speedup over PyTorch's highly optimized `F.linear`; at the network level, Flash EQ-ViT achieves up to $\mathbf{1.7}\times$ end-to-end speedup over both equivariant and non-equivariant baselines. As an operator-level acceleration algorithm, Flash EQ-Linear provides plug-and-play acceleration for diverse pretrained equivariant models, including EQ-ViT, EQ-Swin, EQ-VMamba, and EQ-INR, without retraining or architectural changes. Code is available at `https://anonymous.4open.science/r/FlashEQLinear-46D1`.

1 INTRODUCTION

Equivariant networks provide a principled way to explicitly embed geometric symmetry priors (e.g., translation, rotation, and reflection) into network architectures by sharing weights across a transformation group (Ravanbakhsh et al., 2017). Rather than learning such symmetries implicitly from augmented data, they guarantee by construction that a transformation of the input induces the corresponding transformation of the output, which improves robustness and generalization while using substantially fewer parameters. Equivariant variants now span the major vision architecture families: CNNs (Cohen & Welling, 2016a;b; Weiler et al., 2018; Kondor & Trivedi, 2018; Weiler & Cesa, 2019; Shen et al., 2020; 2021; He et al., 2021; Xie et al., 2022), Vision Transformers (EQ-ViTs) (Romero & Cordonnier, 2021; Xu et al., 2023; Fu et al., 2026), Visual Mamba (EQ-VMamba) (Zhao et al., 2026), and implicit neural representations (EQ-INR) (Xie et al., 2025). Each serves as a plug-and-play replacement for its standard counterpart, matching or exceeding its accuracy with far fewer parameters across image classification, segmentation, and restoration.

Despite the remarkable parameter efficiency and performance, equivariant networks generally suffer from a critical bottleneck: **the parameter efficiency has not translated well into compute efficiency**. The cause lies in how weight sharing is realized: the shared weights are replicated, permuted, and tiled along the group dimension into a dense weight tensor, which is then handed to a generic dense kernel. This naive replication-and-dense-multiplication strategy incurs substantial execution overhead, making equivariant networks *slower* than their non-equivariant counterparts (Gerken et al., 2022). While a few recent efforts (Nordström et al., 2026) have designed customized efficient ViT

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Standard Linear Layer (b) Naive EQ-Linear Layer (c) Flash EQ-Linear Layer

Figure 1: Illustration of the EQ-Linear under the p4 group. The weight matrix $\boldsymbol{W}$ is constructed by cyclically shifting and replicating a set of learnable parameters $\tilde{\boldsymbol{W}}$ along the group dimension.

architectures, their structural modifications cannot be easily transferred to more general equivariant network families, nor can they be directly applied to pre-trained equivariant networks for acceleration, which largely limits their generality. Moreover, current equivariant layers lack the highly optimized CUDA kernels (Chetlur et al., 2014) that standard network layers benefit from, which makes them hard to deploy in practice.

Ideally, an acceleration technique for equivariant models should target the operator level, specifically the fundamental equivariant linear (EQ-Linear) layer (Finzi et al., 2021; Xie et al., 2025). EQ-Linear is a core building block of EQ-ViTs, EQ-VMamba, and EQ-INR, and dominates their computational cost. In EQ-ViT, for instance, EQ-Linear layers appear in every attention projection and every FFN block, accounting for more than 50% of both FLOPs and inference latency (Table 3). EQ-Linear is thus the main computational bottleneck shared by these backbones, and accelerating it at the operator level benefits all of them at once, without redesigning or retraining any of them.

However, the current naive implementation of EQ-Linear suffers from dense matrix-multiply and repetitive memory movement (as shown in Fig. 1(b)). In particular, the underlying parameter sharing mechanism not only fails to deliver computational efficiency, but severely degrades the execution speed due to repeated memory access overhead. Although current workarounds typically pre-assemble the weight matrix at inference, allowing EQ-Linear to run exactly as fast as a vanilla linear layer, this merely hides the redundancy rather than translating parameter efficiency into compute efficiency.

Instead of treating the EQ-Linear weight matrix as an ordinary dense matrix, we observe that it exhibits a **Cyclic Shift and Share** structure: the shared parameters are cyclically shifted and replicated T times along the group axis. This structural regularity underlies our key insight: EQ-Linear performs a circular convolution along the group dimension composed with a linear transformation along the channel dimension, and a circular convolution costs far less than a dense matrix product.

This observation unlocks a significantly faster mathematical formulation for EQ-Linear. **(i)** By the convolution theorem of the discrete Fourier transform (DFT), the circulant convolution along the group dimension is exactly a pointwise multiplication in the Fourier domain (Oppenheim & Schafer, 2009), replacing NC^2 real multiplications with NC^2/T complex ones, where N is the token count, C the channel width, and T the group size. **(ii)** By the conjugate symmetry of the real DFT, about half of the frequency components are conjugates of the other half and need not be computed, while the zero and Nyquist (π) frequency components are purely real and require only real arithmetic. Together these reduce the computation of EQ-Linear to approximately $2(T-1)NC^2/T^2$ real multiplications.

Building on the above analysis, we propose **Flash EQ-Linear**, an exact acceleration algorithm that delivers an approximately **2**$\times$ speedup for p4 equivariant linear layers in the compute-bound regime. Unlike prior efforts that redesign equivariant architectures, Flash EQ-Linear accelerates the existing EQ-Linear operator itself with dedicated CUDA kernels, so it applies to pretrained p4-equivariant models without retraining. The main contributions of this paper can be summarized as follows:

- **Algorithmic.** We propose Flash EQ-Linear: an exact acceleration algorithm for the equivariant linear layer. It reformulates group convolution as pointwise multiplication in the frequency domain and further exploits the conjugate symmetry of the real DFT to remove redundant frequency-domain operations, reducing the theoretical cost to $2(T-1)/T^2$ of the original dense formulation[1].

[1]The stated ratio assumes an even group size T, as is standard in practice; for odd group size, it is $(2T-1)/T^2$.

- **Exactness and transferability.** The proposed algorithm is mathematically exact: relying only on the invertibility of the DFT and the convolution theorem, its outputs are identical to those of the naive EQ-Linear. Besides, it serves as a plug-and-play replacement for the EQ-Linear layers of pretrained equivariant networks across architecture families, including Transformers (EQ-ViT and EQ-Swin), state space models (EQ-VMamba), and implicit neural representations (EQ-INR), requiring no fine-tuning or architectural change.
- **Hardware-aware CUDA kernels.** We develop dedicated CUDA kernels for Flash EQ-Linear that support both forward and backward passes in FP32 and FP16, translating theoretical computational savings into practical speedups. The kernels fuse the group-wise transforms into the prologue and epilogue of a GEMM-like channel reduction, keeping every frequency-domain intermediate in registers and shared memory, and are released as an out-of-the-box library for the community.
- **Empirical validation.** At the operator level, Flash EQ-Linear achieves up to $\mathbf{2.1}\times$ forward speedup over PyTorch's `F.linear` in the compute-bound regime. At the network level, Flash EQ-ViT achieves up to $\mathbf{1.7}\times$ end-to-end speedup over standard ViT and the corresponding equivariant baselines. These results follow the trend predicted by our theoretical analysis.

2 Related Work

Group equivariant networks. Group equivariance was introduced to deep learning by G-CNN (Cohen & Welling, 2016a), which generalizes convolution by sharing weights across a transformation group. Subsequent work broadened the supported groups (Cohen & Welling, 2016b; Weiler & Cesa, 2019), refined the parameterization of group representations (Xie et al., 2022; 2025), and carried equivariance into modern architecture families, including ViTs (Romero & Cordonnier, 2021; Xu et al., 2023; Fu et al., 2026), VMamba (Zhao et al., 2026), and implicit neural representations (Xie et al., 2025). Most existing research, however, focuses on expressivity, generalization, and parameter efficiency, while overlooking the run-time efficiency of equivariant operators.

Exact acceleration. Unlike pruning, quantization, or distillation (Cheng et al., 2018; Han et al., 2016; Hinton et al., 2015), which trade accuracy for speed, exact acceleration leaves the computed function unchanged, achieving its gains through algebraic reformulation or hardware-aware optimization. For algebraic reformulation, Mathieu et al. (2013) computed CNN convolutions as pointwise products via the FFT (Cooley & Tukey, 1965), with optimized CUDA implementations by Vasilache et al. (2014). For hardware-aware optimization, FlashAttention (Dao et al., 2022) accelerates exact attention through IO-aware tiling to reduce data movement across the GPU memory hierarchy. Flash EQ-Linear combines both approaches: it applies the Fourier transform along the *group* dimension rather than the *spatial* dimension, and realizes the resulting reduction in fused CUDA kernels.

Acceleration of equivariant networks. Most closely related, Bökman et al. (2025) and Nordström et al. (2026) proposed customized equivariant ViT variants that achieve computational reductions by the isotypic decomposition. While this represents an important step, such solutions are architecture-specific and lack comprehensive system-level CUDA optimizations. More importantly, because they modify the underlying architecture, these methods invalidate existing model weights and require costly retraining from scratch. In contrast, Flash EQ-Linear provides a plug-and-play operator-level acceleration for pretrained equivariant networks, without architectural changes or retraining.

3 Flash EQ-Linear: Principle, Algorithm, and Implementations

In this section, we develop Flash EQ-Linear, an exact algorithm that exploits the discrete Fourier transform (DFT) to reduce the complexity of EQ-Linear. We first present the underlying acceleration principle based on the convolution theorem and the conjugate symmetry of the real DFT (Sec. 3.1), then derive the algorithm and analyze its properties (Sec. 3.2). Finally, we describe the mathematical specialization and CUDA kernel implementation for the $\mathrm{p4}$ group (Sec. 3.3).

Formulation setup. For clarity, we formulate our methodology under four simplifying conditions: (1) equal input and output channel widths, denoted by C; (2) an even group size T; (3) C divisible by T, with $C \gg T$; and (4) a total token count $N \gg T$. These assumptions reflect common settings and can be generalized to broader scenarios, such as asymmetric channel dimensions and odd T, essentially without altering our theoretical speedup derivations. Furthermore, since practical group

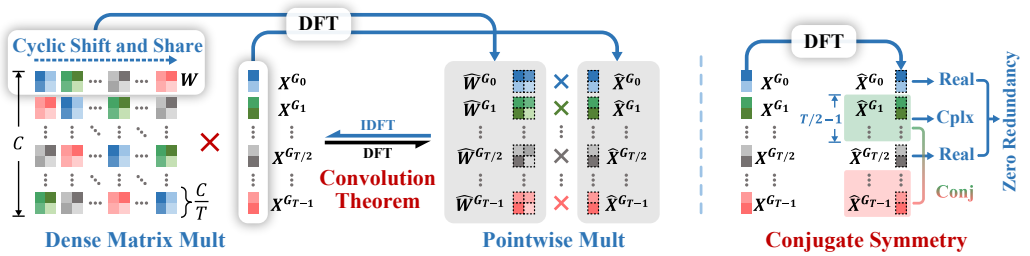


Figure 2: Acceleration via the real DFT. **Left:** The convolution theorem maps dense EQ-Linear computation to pointwise multiplications across T frequency bins. **Right:** Conjugate symmetry yields only $T/2+1$ non-redundant bins: frequency components at $k=0$ and $k=T/2$ are purely real, while the remaining components form conjugate pairs.

sizes are typically small (e.g., $T \in \{2,4,8\}$), any indivisibility can be easily resolved via negligible channel padding with minimal computational overhead.

3.1 Acceleration Principle

Formulation of EQ-Linear. As illustrated in Fig. 1, the weight matrix $\boldsymbol{W}$ of an equivariant linear layer has a group-circulant structure formed by cyclically shifting a set of learnable parameters $\tilde{\boldsymbol{W}} = [\tilde{\boldsymbol{W}}_0, \tilde{\boldsymbol{W}}_1, \ldots, \tilde{\boldsymbol{W}}_{T-1}]$ along the group dimension. Given a group-structured input $\boldsymbol{X} \in \mathbb{R}^{N \times \frac{C}{T} \times T}$ over a group $\mathcal{G} = \{G_t \,|\, t = 0, 1, \ldots, T-1\}$, EQ-Linear maps it to an output $\boldsymbol{Y} \in \mathbb{R}^{N \times \frac{C}{T} \times T}$ as

$$\begin{bmatrix} \boldsymbol{Y}_n^{G_0} \\ \boldsymbol{Y}_n^{G_1} \\ \vdots \\ \boldsymbol{Y}_n^{G_{T-1}} \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{W}}_0 & \tilde{\boldsymbol{W}}_{T-1} & \cdots & \tilde{\boldsymbol{W}}_1 \\ \tilde{\boldsymbol{W}}_1 & \tilde{\boldsymbol{W}}_0 & \cdots & \tilde{\boldsymbol{W}}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{\boldsymbol{W}}_{T-1} & \tilde{\boldsymbol{W}}_{T-2} & \cdots & \tilde{\boldsymbol{W}}_0 \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_n^{G_0} \\ \boldsymbol{X}_n^{G_1} \\ \vdots \\ \boldsymbol{X}_n^{G_{T-1}} \end{bmatrix} + \begin{bmatrix} \tilde{\boldsymbol{b}} \\ \tilde{\boldsymbol{b}} \\ \vdots \\ \tilde{\boldsymbol{b}} \end{bmatrix}, \quad \text{for } n = 0, \ldots, N-1. \tag{1}$$

Here, $\boldsymbol{X}_n^{G_t}, \boldsymbol{Y}_n^{G_t} \in \mathbb{R}^{\frac{C}{T}}$ denote the t-th group components[2] of $\boldsymbol{X}_n, \boldsymbol{Y}_n \in \mathbb{R}^{\frac{C}{T} \times T}$, respectively. The weight matrix $\boldsymbol{W} \in \mathbb{R}^{C \times C}$ and bias $\boldsymbol{b} \in \mathbb{R}^C$ are tiled from the learnable parameters $\tilde{\boldsymbol{W}} \in \mathbb{R}^{\frac{C}{T} \times C}$ and $\tilde{\boldsymbol{b}} \in \mathbb{R}^{\frac{C}{T}}$. As formulated in Eq. 1, the naive implementation ignores the group-circulant structure and expands $\tilde{\boldsymbol{W}}$ into a dense matrix, requiring NC^2 multiply-accumulate operations (MACs).

EQ-Linear as a group-circulant convolution. Our key observation is that the EQ-Linear in Eq. 1 essentially performs a circular convolution between each input token $\boldsymbol{X}_n$ and the parameters $\tilde{\boldsymbol{W}}$ along the group dimension, composed with matrix multiplications along the channel dimension:

$$\boldsymbol{Y}_n = \tilde{\boldsymbol{W}} \circledast_{\mathcal{G}} \boldsymbol{X}_n + \tilde{\boldsymbol{b}}, \quad \text{where } (\tilde{\boldsymbol{W}} \circledast_{\mathcal{G}} \boldsymbol{X}_n)^{G_t} = \sum_{s=0}^{T-1} \tilde{\boldsymbol{W}}_{(t-s) \bmod T} \cdot \boldsymbol{X}_n^{G_s}, \ \forall G_t \in \mathcal{G}. \tag{2}$$

Acceleration principle. This group-circulant structure unlocks a significantly faster mathematical formulation, leveraging two key properties of the DFT (as illustrated in Fig. 2):

(i) Convolution Theorem: The classical convolution theorem states that a circular convolution corresponds to pointwise scalar multiplication in the frequency domain. In our case, Eq. 2 defines a mixed operation: circular convolution along the group dimension and matrix multiplication along the channel dimension. Since these two linear operations act on orthogonal dimensions, they commute. Consequently, applying the group-wise DFT pushes through the channel-wise matrix multiplications. As illustrated in the Left part of Fig. 2, this strictly converts the dense EQ-Linear computation into T independent per-frequency matrix multiplications:

$$\mathcal{F}_{\mathcal{G}} \left(\tilde{\boldsymbol{W}} \circledast_{\mathcal{G}} \boldsymbol{X}_n \right)^{G_k} = \hat{\boldsymbol{W}}^{G_k} \cdot \hat{\boldsymbol{X}}_n^{G_k}, \quad \text{where } \hat{\boldsymbol{W}} = \mathcal{F}_{\mathcal{G}}(\tilde{\boldsymbol{W}}), \ \ \hat{\boldsymbol{X}}_n = \mathcal{F}_{\mathcal{G}}(\boldsymbol{X}_n), \tag{3}$$

for all $k \in \{0, \ldots, T-1\}$. Here, $\mathcal{F}_{\mathcal{G}}$ is formally defined as the 1D DFT applied strictly along the group dimension (e.g., $\mathcal{F}_{\mathcal{G}}(\boldsymbol{X}_n)^{G_k} = \sum_{t=0}^{T-1} \boldsymbol{X}_n^{G_t} e^{-\mathrm{i}2\pi kt/T}$). Each independent frequency-domain multiplication maps $\frac{C}{T}$ input channels to $\frac{C}{T}$ output channels.

[2] $G_t \in \mathcal{G}$ denotes both a group transformation (e.g., a rotation matrix) and an index along the group dimension.

Algorithm 1 Flash EQ-Linear (For even group size T)

Input: Real-valued input tokens $\boldsymbol{X} \in \mathbb{R}^{N \times \frac{C}{T} \times T}$, parameters $\tilde{\boldsymbol{W}} \in \mathbb{R}^{\frac{C}{T} \times \frac{C}{T} \times T}$, and bias $\tilde{\boldsymbol{b}} \in \mathbb{R}^{\frac{C}{T}}$.

1: **Step 1: Partial Group-wise DFT.** Compute DFT for non-redundant frequencies:
$\hat{\boldsymbol{X}}_n^{G_k} = \mathcal{F}_{\mathcal{G}}(\boldsymbol{X}_n)^{G_k}, \quad \hat{\boldsymbol{W}}^{G_k} = \mathcal{F}_{\mathcal{G}}(\tilde{\boldsymbol{W}})^{G_k}, \text{ for } k = 0, \ldots, T/2 \text{ and } n = 0, \ldots, N-1.$

2: **Step 2: Parallelizable per-frequency matrix multiplication.**

(a) Pure real frequencies (2 real matrix multiplications):
$\hat{\boldsymbol{Y}}_n^{G_k} = \hat{\boldsymbol{W}}^{G_k} \cdot \hat{\boldsymbol{X}}_n^{\ G_k}, \text{ for } k = 0, T/2 \text{ and } n = 0, \ldots, N-1.$

(b) Complex frequencies ($T/2-1$ complex matrix multiplications):
$\hat{\boldsymbol{Y}}_n^{G_k} = \hat{\boldsymbol{W}}^{G_k} \cdot \hat{\boldsymbol{X}}_n^{\ G_k}, \text{ for } k = 1, \ldots, T/2-1 \text{ and } n = 0, \ldots, N-1.$

3: **Step 3: Conjugate-symmetric recovery.** Recover $k = T/2+1, \ldots, T-1$ via $\hat{\boldsymbol{Y}}^{G_k} = \overline{\hat{\boldsymbol{Y}}^{G_{T-k}}}$.

4: **Step 4: Group-wise IDFT.** Apply inverse DFT to $\hat{\boldsymbol{Y}}$ and add bias $\tilde{\boldsymbol{b}}$.

Output: $\boldsymbol{Y} \in \mathbb{R}^{N \times \frac{C}{T} \times T}$

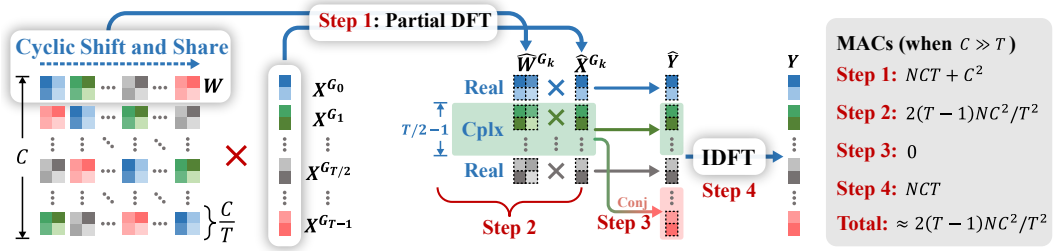


Figure 3: **Execution flow of Flash EQ-Linear.** The computation proceeds in four stages: (**Step 1**) partial group-wise DFT; (**Step 2**) independent frequency-domain matrix multiplications, using real arithmetic for zero/Nyquist frequencies and complex arithmetic for remaining pairs; (**Step 3**) zero-cost conjugate-symmetric recovery; and (**Step 4**) group-wise IDFT with bias addition.

(ii) Conjugate symmetry of real DFT: Since both $\boldsymbol{X}$ and $\tilde{\boldsymbol{W}}$ are real-valued, their Fourier coefficients exhibit conjugate symmetry:

$$\hat{\boldsymbol{X}}_n^{\ G_{T-k}} = \overline{\hat{\boldsymbol{X}}_n^{\ G_k}}, \quad \forall k \in \{1, \ldots, T-1\}, \tag{4}$$

and likewise for $\hat{\boldsymbol{W}}$. As illustrated in the Right part of Fig. 2, nearly half of the frequency components are redundant. Given our setup of an even group size T, only the first $T/2+1$ non-redundant components ($k = 0, \ldots, T/2$) need to be explicitly computed to achieve zero redundancy. More importantly, the zero frequency ($k = 0$) and the Nyquist frequency ($k = T/2$) components are purely real. This allows us to substitute complex matrix multiplications (4 real-valued MACs) with standard real matrix multiplications (1 real-valued MAC) at these frequencies.

3.2 Algorithm and Complexity Analysis

Building on the dual acceleration principle, we formally propose the **Flash EQ-Linear algorithm**. Specifically, the algorithm leverages the convolution theorem (Eq. 3) to decompose the dense matrix multiplication into a series of compact, independent frequency-domain matrix multiplications, and explicitly exploits conjugate symmetry (Eq. 4) to further eliminate redundant computations. The complete execution pipeline is detailed in Algorithm 1 and illustrated in Fig. 3, which map the theoretical derivations into an efficient four-step computational flow.

Complexity analysis. Among the four steps of Algorithm 1, the per-frequency matrix multiplications (Step 2) constitute the primary computational bottleneck. Flash EQ-Linear reduces this dominant cost by exploiting the conjugate symmetry made explicit in Fig. 3: two self-conjugate frequencies $k = 0$ and $k = T/2$ are purely real and reduce to real matrix multiplications, whereas each of the remaining $T/2-1$ frequencies is complex and requires four real matrix multiplications. Step 2 therefore costs $2\frac{NC^2}{T^2} + 4(T/2-1)\frac{NC^2}{T^2} = \frac{2(T-1)NC^2}{T^2}$ real-valued MACs. We formalize the resulting efficiency gain in Theorem 1, with the full derivation deferred to Appendix A.1.

Theorem 1 (Complexity of Flash EQ-Linear) *Let $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^{N \times \frac{C}{T} \times T}$ be the input and output of an EQ-Linear layer with an even group size T. When $C \gg T$ and $N \gg T$, the number of real-valued*

MACs incurred by Flash EQ-Linear satisfies

$$\mathrm{MACs}_{\mathrm{Flash}} \approx 2(T-1)NC^2/T^2. \tag{5}$$

Compared with the NC^2 MACs of naive EQ-Linear, this yields a theoretical speedup of ${}^{T^2}/_{2(T-1)}$.

For the most ubiquitous transformation groups in practice, such as 90° rotations ($T = 4$) and mirror reflections ($T = 2$), this translates to concrete speedups of $\approx 2.67\times$ and $2\times$, respectively.

Properties. Beyond the asymptotic reduction in computational complexity, the proposed Flash EQ-Linear design inherently enjoys several highly desirable properties for practical deployment:

(i) Exactness and equivariance preservation. Flash EQ-Linear is exact by construction: the invertibility of DFT and the convolution theorem make it algebraically equivalent to naive EQ-Linear. It therefore preserves the rotation equivariance, as verified in Tables 10 and 11 in the Appendix.

(ii) Speedup scales with group size. As established in Theorem 1, the theoretical speedup ${}^{T^2}/_{2(T-1)}$ of Flash EQ-Linear grows with the group size T, from $2\times$ for $T = 2$ to $4.57\times$ for $T = 8$. Thus, EQ-Linear with finer groups (larger T) admit greater speedups, as verified in Table 12 in the Appendix.

(iii) Training-free and plug-and-play. Flash EQ-Linear can be used to directly replace naive EQ-Linear in pretrained equivariant networks in a plug-and-play manner, requiring no retraining, fine-tuning, or architectural changes. Existing models such as EQ-ViT and EQ-Swin can therefore adopt Flash EQ-Linear without modifying their training or inference pipelines (as shown in Table 4).

3.3 Implementation for the p4 Group

For the widely used p4 rotation group (Cohen & Welling, 2016a) ($T = 4$), specializing Alg. 1 yields a substantially simplified form that maps cleanly onto an efficient fused implementation.

Step 1: Group-wise DFT. For the p4 rotation group ($T = 4$), the group-wise discrete Fourier transform of $\boldsymbol{X}$ is specialized to the following explicit form:

$$\begin{bmatrix} \hat{\boldsymbol{X}}_n^{G_0} \\ \hat{\boldsymbol{X}}_n^{G_1} \\ \hat{\boldsymbol{X}}_n^{G_2} \\ \hat{\boldsymbol{X}}_n^{G_3} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_n^{G_0} \\ \boldsymbol{X}_n^{G_1} \\ \boldsymbol{X}_n^{G_2} \\ \boldsymbol{X}_n^{G_3} \end{bmatrix} \Longrightarrow \begin{bmatrix} \hat{\boldsymbol{X}}_n^{G_0} \\ \mathrm{Re}(\hat{\boldsymbol{X}}_n^{G_1}) \\ \mathrm{Im}(\hat{\boldsymbol{X}}_n^{G_1}) \\ \hat{\boldsymbol{X}}_n^{G_2} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_n^{G_0} \\ \boldsymbol{X}_n^{G_1} \\ \boldsymbol{X}_n^{G_2} \\ \boldsymbol{X}_n^{G_3} \end{bmatrix}. \tag{6}$$

The reduced DFT transform matrix in Eq. 6 contains only 0 and ± 1. Consequently, the group-wise DFT reduces entirely to additions and subtractions, requiring no nontrivial multiplications.

Step 2: Per-frequency Fourier-domain matrix multiplication. Using the four real-valued Fourier components derived in Step 1, the per-frequency multiplication specializes to

$$\begin{aligned} &\hat{\boldsymbol{Y}}_n^{G_0} = \hat{\boldsymbol{W}}^{G_0} \cdot \hat{\boldsymbol{X}}_n^{G_0}, \;\; \hat{\boldsymbol{Y}}_n^{G_2} = \hat{\boldsymbol{W}}^{G_2} \cdot \hat{\boldsymbol{X}}_n^{G_2}, \;\; \hat{\boldsymbol{Y}}_n^{G_3} = \overline{\hat{\boldsymbol{Y}}_n^{G_1}}, \\ &\mathrm{Re}(\hat{\boldsymbol{Y}}_n^{G_1}) = \mathrm{Re}(\hat{\boldsymbol{W}}^{G_1}) \cdot \mathrm{Re}(\hat{\boldsymbol{X}}_n^{G_1}) - \mathrm{Im}(\hat{\boldsymbol{W}}^{G_1}) \cdot \mathrm{Im}(\hat{\boldsymbol{X}}_n^{G_1}), \\ &\mathrm{Im}(\hat{\boldsymbol{Y}}_n^{G_1}) = \mathrm{Im}(\hat{\boldsymbol{W}}^{G_1}) \cdot \mathrm{Re}(\hat{\boldsymbol{X}}_n^{G_1}) + \mathrm{Re}(\hat{\boldsymbol{W}}^{G_1}) \cdot \mathrm{Im}(\hat{\boldsymbol{X}}_n^{G_1}). \end{aligned} \tag{7}$$

Step 2 thus reduces to six real-valued matrix multiplications, which can be executed in parallel. Since each multiplication maps $\frac{C}{T}$ input channels to $\frac{C}{T}$ output channels, the total cost is $\frac{\mathbf{6}}{\mathbf{16}}\boldsymbol{NC^2}$ MACs.

Step 3: Conjugate-symmetric recovery. Real-valued inputs and weights imply $\hat{\boldsymbol{Y}}^{G_3} = \overline{\hat{\boldsymbol{Y}}^{G_1}}$, allowing the inverse transform to incorporate G_3 implicitly without materializing it.

Step 4: Group-wise IDFT. Exploiting the conjugate symmetry from Step 3, the group-wise inverse DFT for the p4 rotation group can be expressed directly using four real-valued Fourier components:

$$\begin{bmatrix} \boldsymbol{Y}_n^{G_0} \\ \boldsymbol{Y}_n^{G_1} \\ \boldsymbol{Y}_n^{G_2} \\ \boldsymbol{Y}_n^{G_3} \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{Y}}_n^{G_0} \\ \hat{\boldsymbol{Y}}_n^{G_1} \\ \hat{\boldsymbol{Y}}_n^{G_2} \\ \hat{\boldsymbol{Y}}_n^{G_3} \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 2 & 0 & 1 \\ 1 & 0 & -2 & -1 \\ 1 & -2 & 0 & 1 \\ 1 & 0 & 2 & -1 \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{Y}}_n^{G_0} \\ \mathrm{Re}(\hat{\boldsymbol{Y}}_n^{G_1}) \\ \mathrm{Im}(\hat{\boldsymbol{Y}}_n^{G_1}) \\ \hat{\boldsymbol{Y}}_n^{G_2} \end{bmatrix}. \tag{8}$$

Eq. 8 contains only the dyadic coefficients $\{0, \pm\frac{1}{4}, \pm\frac{1}{2}\}$ and thus reduces to additions, subtractions, and power-of-two rescalings, incurring no additional MACs in the final implementation.

Table 1: **Forward Latency comparison** of Flash EQ-Linear against Standard Linear and Naive EQ-Linear baselines under different channel numbers ($C/T = 64 \to 2048$). *Speedup ratios* (green) are measured against the standard Linear (PyTorch's `F.linear`) baseline.

Method	#Param. (M)↓	FLOPs (G)↓	Forward Latency (ms)↓					
			64	128	256	512	1024	2048
Single-Precision Floating-Point (FP32)								
Standard Linear	0.066	0.134	0.11	0.43	1.52	6.05	24.48	104.96
Naive EQ-Linear	**0.016**	0.134	0.11	0.44	1.54	6.25	25.16	105.74
Flash EQ-Linear (PyTorch)	**0.016**	**0.052**	0.52	0.99	2.24	5.82	16.16	51.89
Flash EQ-Linear (CUDA)	**0.016**	**0.052**	**0.07**	**0.25**	**0.79**	**3.03**	**11.92**	**49.14**
Speedup/Reduction ↑	**75.0%**	**61.2%**	**1.6×**	**1.7×**	**1.9×**	**2.0×**	**2.1×**	**2.1×**
Half-Precision Floating-Point (FP16)								
Standard Linear	0.066	0.134	0.05	0.12	0.44	1.70	6.79	30.08
Naive EQ-Linear	**0.016**	0.134	0.05	0.12	0.46	1.73	6.90	30.59
Flash EQ-Linear (PyTorch)	**0.016**	**0.052**	0.58	0.61	1.20	2.68	6.73	18.67
Flash EQ-Linear (CUDA)	**0.016**	**0.052**	**0.04**	**0.09**	**0.31**	**1.04**	**3.23**	**14.07**
Speedup/Reduction ↑	**75.0%**	**61.2%**	**1.3×**	**1.3×**	**1.4×**	**1.6×**	**2.1×**	**2.1×**

Table 2: **Backward latency comparison** of Flash EQ-Linear against Standard Linear and Naive EQ-Linear baselines under different channel numbers ($C/T = 16 \to 2048$) under FP32.

Method	Backward Latency (ms)↓							
	16	32	64	128	256	512	1024	2048
Standard Linear	0.17	0.14	0.16	0.46	1.67	5.42	20.80	83.11
Naive EQ-Linear	0.34	0.33	0.33	0.53	1.80	5.68	22.35	89.34
Flash EQ-Linear (PyTorch)	2.88	3.18	4.17	8.43	15.63	30.62	67.49	170.92
Flash EQ-Linear (CUDA)	**0.14**	**0.13**	**0.13**	**0.39**	**1.29**	**4.70**	**17.79**	**70.91**
Speedup (vs. Standard) ↑	**1.2×**	**1.1×**	**1.3×**	**1.2×**	**1.3×**	**1.2×**	**1.2×**	**1.2×**
Speedup (vs. Naive) ↑	**2.5×**	**2.6×**	**2.6×**	**1.4×**	**1.4×**	**1.2×**	**1.3×**	**1.3×**

CUDA kernel design. A straightforward PyTorch implementation of Alg. 1 decomposes the computation into separate DFT, complex multiplication, and IDFT operators. The resulting kernel launches, layout transformations, and global-memory traffic can dominate runtime and diminish the algorithmic gains (Williams et al., 2009; Dao et al., 2022). To translate the theoretical speedups into practical speedups, we develop dedicated CUDA kernels of Flash EQ-Linear for the $\mathrm{p4}$ rotation group, supporting both forward and backward passes in FP32 and FP16.

Our implementation fuses all four steps of Alg. 1 into a single GEMM-like dataflow, eliminating costly intermediate round trips to global memory. For $\mathrm{p4}$ with $T = 4$, the DFT and IDFT are fully unrolled into additions, subtractions, and fixed rescalings, with intermediate Fourier-domain fragments retained in registers or shared memory. Weights are stored in frequency-domain planes, while input activations remain in their native layout, avoiding auxiliary packed tensors and additional GPU memory allocation. We further provide shape-specialized kernel variants that tune tile sizes, shared-memory staging, persistence, Tensor Core usage, and vectorized memory access across different $(B, L, \frac{C}{T}, \frac{\bar{C}}{T})$ regimes. Further implementation details are provided in Appendix A.3.

4 Experiments

In this section, we evaluate the practical speedups of Flash EQ-Linear through operator-level and network-level runtime benchmarks. Complete experimental results, including validation of numerical accuracy and equivariance preservation, are provided in Appendix A.4.

4.1 Operator-Level Acceleration

Experimental settings. We benchmark Flash EQ-Linear against two baselines: standard Linear (PyTorch's `F.linear` (Paszke et al., 2019)) and Naive EQ-Linear, which explicitly unrolls the equivariant weights. To quantify the benefits of the CUDA optimizations, we compare the PyTorch and CUDA implementations of Flash EQ-Linear. All methods receive the same input feature map

Table 3: **Inference latency comparison** of Flash EQ-ViT against standard non-EQ and naive EQ baselines on ImageNet-100. *Speedup ratios* (green) are measured against standard ViT baselines.

Method	#Param. (M)↓	Top-1 (%)↑	All Linear Layers			Total Network		
			FLOPs (G)↓	Latency (ms)↓	Latency Ratio	FLOPs (G)↓	Latency (ms)↓	Throughput (imgs/s)↑
Single-Precision Floating-Point (FP32)								
ViT-S	33.8	76.5	13.1	0.352	58.4%	14.1	0.603	1658
Naive EQ-ViT-S	**8.4**	**78.2**	13.1	0.355	58.7%	14.1	0.604	1656
Flash EQ-ViT-S	**8.4**	**78.2**	**5.0**	**0.183**	**42.6%**	**6.0**	**0.429**	**2332** (1.4×)
ViT-B	85.9	77.1	33.5	0.761	67.1%	35.1	1.135	881
Naive EQ-ViT-B	**21.4**	**80.1**	33.5	0.757	67.1%	35.1	1.128	886
Flash EQ-ViT-B	**21.4**	**80.1**	**12.6**	**0.400**	**52.3%**	**14.3**	**0.764**	**1309** (1.5×)
ViT-L	303.4	78.4	119.0	2.672	71.9%	123.1	3.715	269
Naive EQ-ViT-L	**75.8**	**80.6**	119.0	2.760	73.9%	123.1	3.735	268
Flash EQ-ViT-L	**75.8**	**80.6**	**44.9**	**1.336**	**56.7%**	**49.0**	**2.356**	**424** (1.6×)
ViT-H	631.0	81.3	248.0	5.586	77.7%	254.7	7.192	139
Naive EQ-ViT-H	**157.7**	**81.5**	248.0	5.627	78.4%	254.7	7.182	139
Flash EQ-ViT-H	**157.7**	**81.5**	**93.4**	**2.719**	**63.2%**	**100.1**	**4.304**	**232** (1.7×)
Half-Precision Floating-Point (FP16)								
ViT-L	303.4	78.4	119.0	0.742	59.0%	123.1	1.258	795
Naive EQ-ViT-L	**75.8**	**80.6**	119.0	0.752	59.6%	123.1	1.262	792
Flash EQ-ViT-L	**75.8**	**80.6**	**44.9**	**0.497**	**49.5%**	**49.0**	**1.004**	**996** (1.3×)
ViT-H	631.0	81.3	248.0	1.541	64.8%	254.7	2.379	420
Naive EQ-ViT-H	**157.7**	**81.5**	248.0	1.557	65.1%	254.7	2.392	418
Flash EQ-ViT-H	**157.7**	**81.5**	**93.4**	**0.976**	**54.2%**	**100.1**	**1.801**	**555** (1.3×)

$\boldsymbol{X} \in \mathbb{R}^{B\times L\times \frac{C}{T}\times T}$, with batch size $B=32$, sequence length $L=1024$, and group size $T=4$. We vary the per-group channel width C/T from 64 to 2048. For a fair comparison, the standard Linear applies a dense $C \to C$ matrix projection. Parameters and FLOPs are reported at $C/T = 64$. We measure forward and backward latency on a single NVIDIA RTX 4090 GPU, averaging over 200 runs.

Experimental results. Tables 1 and 2 report forward and backward latency across channel widths, respectively. **(i) Forward latency.** As the channel width C/T increases from 64 to 2048, Flash EQ-Linear accelerates the forward pass by $\mathbf{1.6\times}$–$\mathbf{2.1\times}$ in FP32 and $\mathbf{1.3\times}$–$\mathbf{2.1\times}$ in FP16. At small channel widths, performance is dominated by memory traffic, layout transformations, and kernel-launch overhead. As the workload becomes compute-bound, the speedup increases toward the theoretical upper bound of $\mathbf{2.67\times}$. Flash EQ-Linear also reduces the computational cost from 0.134 G to 0.052 G FLOPs, a $\mathbf{2.6\times}$ reduction consistent with our theoretical complexity analysis. **(ii) Backward latency.** In FP32, Flash EQ-Linear consistently accelerates the backward pass over Standard Linear, although by a smaller margin than that in the forward pass because gradient computation incurs additional arithmetic and memory traffic. Comparing the two realizations of Flash EQ-Linear, Flash EQ-Linear (CUDA) consistently outperforms its PyTorch counterpart, especially at smaller channel widths and during backpropagation, highlighting the importance of CUDA optimization. **Overall.** These results confirm that the theoretical reduction in computation translates into substantial wall-clock acceleration. In the compute-bound regime, Flash EQ-Linear achieves up to a $\mathbf{2.1\times}$ forward speedup over a highly optimized non-equivariant baseline.

4.2 Network-Level Acceleration

Experimental settings. We first integrate Flash EQ-Linear into EQ-ViT (Fu et al., 2026) by replacing every naive EQ-Linear layer while leaving the remaining architecture unchanged. We compare Flash EQ-ViT against the standard non-equivariant ViT (Dosovitskiy et al., 2021) and its naive equivariant counterpart across multiple model scales (Tiny/Small/Base/Large/Huge). Within each scale, all methods share the same total channel width, batch size 128, input resolution 224×224, and group size $T = 4$. We report Top-1 accuracy on ImageNet-100, a 100-class subset of ImageNet-1K (Deng et al., 2009), together with the parameter count, FLOPs, cumulative linear-layer latency, the linear-layer-to-network latency ratio, end-to-end latency, and inference throughput (imgs/s). To

Table 4: **Plug-and-play speedups** achieved by Flash EQ-Linear over naive equivariant implementations of Swin Transformer, VMamba, SwinIR-LIIF, and SwinIR-LTE in FP32.

Method	#Param. (M)	Top-1 (%) / PSNR (dB)	FLOPs (G)↓		Latency (ms)↓		Throughput (imgs/s)↑		
			Naive	Flash	Naive	Flash	Naive	Flash	Speedup
EQ-Swin-H (Fu et al., 2026)	164.0	88.3	230.3	**90.8**	7.396	**4.694**	135	**213**	**1.6**×
EQ-VMamba-H (Zhao et al., 2026)	161.9	89.2	227.3	**101.4**	7.735	**5.441**	129	**184**	**1.4**×
EQ-INR-LIIF (Xie et al., 2025)	7.8	33.54	163.6	**93.8**	9.542	**7.572**	105	**132**	**1.3**×
EQ-INR-LTE (Xie et al., 2025)	7.7	33.67	157.2	**89.2**	10.446	**8.243**	96	**121**	**1.3**×

Table 5: Comparison of Flash EQ-ViT with Quick ViTs on ImageNet-100 under FP32.

Method	#Param. (M)↓	FLOPs (G)↓	Top-1 (%)↑	Latency (ms)↓	Throughput (imgs/s)↑
Naive EQ-ViT-T	5.4	9.2	**77.3**	0.359	2785
Quick ViT-$\mathcal{D}_8$-T (Nordström et al., 2026)	2.8	1.7	71.6	0.304	3294
Quick ViT-$\mathcal{H}_8$-T (Nordström et al., 2026)	4.3	2.3	76.0	0.305	3276
Quick ViT-$\mathcal{I}_8$-T (Nordström et al., 2026)	4.5	2.3	76.9	0.307	3257
Flash EQ-ViT-T	5.4	4.0	**77.3**	**0.249**	**4012**

evaluate the plug-and-play applicability, we additionally integrate Flash EQ-Linear into pretrained EQ-Swin-H (Fu et al., 2026), EQ-VMamba-H (Zhao et al., 2026), and EQ-INR-LIIF/LTE (Xie et al., 2025) for $2\times$ super-resolution, without retraining or fine-tuning. We further compare Flash EQ-ViT with the $\mathcal{D}_8$, $\mathcal{I}_8$, and $\mathcal{H}_8$ variants of Quick ViTs (Nordström et al., 2026) on ImageNet-100 in FP32 at comparable parameter counts. All measurements are obtained on a single NVIDIA RTX 4090 GPU and averaged over 200 runs.

Experimental results. Tables 3–5 report the inference efficiency of Flash EQ-Linear across different equivariant architectures. **(i) EQ-ViT speedup.** Flash EQ-ViT achieves end-to-end speedups of $\mathbf{1.4}\times$–$\mathbf{1.7}\times$ in FP32 and approximately $\mathbf{1.3}\times$ in FP16 over standard ViT baselines, while preserving the Top-1 accuracy of its naive equivariant counterpart at every scale. The gains grow with model scale, since wider models devote a larger fraction of their runtime to linear layers (58.4% for ViT-S versus 77.7% for ViT-H) and their larger channel widths push the operator further into the compute-bound regime. **(ii) Plug-and-play speedup.** Beyond ViT, Flash EQ-Linear yields end-to-end speedups of $\mathbf{1.6}\times$ on EQ-Swin-H, $\mathbf{1.4}\times$ on EQ-VMamba-H, and $\mathbf{1.3}\times$ on EQ-INR-LIIF/LTE, respectively, with accuracy and PSNR unchanged. **(iii) Comparison with architecture-level acceleration.** At the Tiny scale, Flash EQ-ViT is $\mathbf{1.22}\times$ faster than the fastest Quick ViT variant at comparable or higher Top-1 accuracy, which we attribute mainly to implementation (fused kernels versus a PyTorch implementation). Notably, Quick ViTs obtain efficiency through architectural redesign and require training from scratch, whereas Flash EQ-Linear accelerates pretrained equivariant models with no retraining, no architectural change, and no accuracy trade-off.

5 Conclusion

In this paper, we presented Flash EQ-Linear, an exact acceleration algorithm with dedicated CUDA kernels that reduces the cost of an EQ-Linear layer from NC^2 to $2(T-1)NC^2/T^2$ MACs. At the operator level, Flash EQ-Linear achieves up to $\mathbf{2.1}\times$ forward speedup over PyTorch's `F.linear`; at the network level, Flash EQ-ViT achieves up to $\mathbf{1.7}\times$ end-to-end speedup over both equivariant and non-equivariant baselines, with no retraining, no architectural change, and no accuracy trade-off. Crucially, while equivariant architectures like EQ-ViT and EQ-VMamba already excel in highly symmetric domains such as remote sensing and medical imaging, their deployment has historically been bottlenecked by slow execution. By resolving this bottleneck, Flash EQ-Linear unlocks the application potential of equivariant networks in real-world, latency-sensitive scenarios, demonstrating that equivariance can simultaneously deliver parameter efficiency and hardware-aware acceleration.

Limitations and future work. Our current CUDA implementation focuses on the widely used $\mathrm{p4}$ rotation group. A natural extension is to generalize the kernels to other cyclic groups, such as the reflection group. Moreover, while this work targets the fundamental equivariant linear layer, the same algorithmic principle may extend to other equivariant operators, such as equivariant convolutions, enabling more comprehensive acceleration of equivariant networks.

Ethics Statement

This work proposes an exact operator-level acceleration algorithm and its CUDA implementation. It involves no human subjects and releases no new data. All experiments use ImageNet-100, a public subset of ImageNet-1K, and publicly released architectures, under their respective terms of use.

Use of Large Language Models

An LLM was used to polish and condense author-written text and to proofread the manuscript for inconsistencies in notation and reported numbers. It was not used to generate the research idea, design the algorithm, or run the experiments. The authors reviewed every suggested edit and take full responsibility for this paper.

Reproducibility Statement

Appendix A.1 derives Theorem 1, Appendix A.3 details the CUDA kernels, and Appendix A.4 gives the timing protocol, model configurations, and exactness checks. Anonymized code is available at `https://anonymous.4open.science/r/FlashEQLinear-46D1`.

References

Georg Bökman, David Nordström, and Fredrik Kahl. Flopping for FLOPs: Leveraging equivariance for computational efficiency. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 4823–4838. PMLR, 2025.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *IEEE Signal Processing Magazine*, 35(1):126–136, 2018.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

Taco S. Cohen and Max Welling. Group equivariant convolutional networks. In *International Conference on Machine Learning*, pp. 2990–2999, 2016a.

Taco S Cohen and Max Welling. Steerable cnns. *arXiv preprint arXiv:1612.08498*, 2016b.

James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359, 2022.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Marc Finzi, Max Welling, and Andrew Gordon Wilson. A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 3318–3328. PMLR, 2021.

Jiahong Fu, Qi Xie, Deyu Meng, and Zongben Xu. Vanilla group equivariant vision transformer: Simple and effective. *arXiv preprint arXiv:2602.08047*, 2026.

Jan Gerken, Oscar Carlsson, Hampus Linander, Fredrik Ohlsson, Christoffer Petersson, and Daniel Persson. Equivariance versus augmentation for spherical images. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 7404–7421. PMLR, 2022.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016.

Lingshen He, Yuxuan Chen, Yiming Dong, Yisen Wang, Zhouchen Lin, et al. Efficient equivariant network. *Advances in Neural Information Processing Systems*, 34:5290–5302, 2021.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Wen-mei W. Hwu, David B. Kirk, and Izzat El Hajj. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, Cambridge, MA, fourth edition, 2022. ISBN 978-0-323-91231-0.

Risi Kondor and Shubhendu Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. In *International Conference on Machine Learning*, pp. 2747–2755. PMLR, 2018.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

David Nordström, Johan Edstedt, Fredrik Kahl, and Georg Bökman. Octic vision transformers: Quicker vits through equivariance. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2026.

Alan V Oppenheim and Ronald W Schafer. *Discrete-Time Signal Processing*. Pearson, 3 edition, 2009.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pp. 8024–8035, 2019.

Siamak Ravanbakhsh, Jeff Schneider, and Barnabás Póczos. Equivariance through parameter-sharing. In *International Conference on Machine Learning*, pp. 2892–2901. PMLR, 2017.

David W. Romero and Jean-Baptiste Cordonnier. Group equivariant stand-alone self-attention for vision. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=JkfYjnOEo6M`.

Jean-Pierre Serre. *Linear Representations of Finite Groups*, volume 42 of *Graduate Texts in Mathematics*. Springer, 1977.

Zhengyang Shen, Lingshen He, Zhouchen Lin, and Jinwen Ma. Pdo-econvs: Partial differential operator based equivariant convolutions. In *International Conference on Machine Learning*, pp. 8697–8706. PMLR, 2020.

Zhengyang Shen, Tiancheng Shen, Zhouchen Lin, and Jinwen Ma. Pdo-es2cnns: Partial differential operator based equivariant spherical cnns. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 9585–9593, 2021.

Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A gpu performance evaluation. *arXiv preprint arXiv:1412.7580*, 2014.

Maurice Weiler and Gabriele Cesa. General E(2)-equivariant steerable CNNs. In *Advances in Neural Information Processing Systems*, 2019.

Maurice Weiler, Fred A Hamprecht, and Martin Storath. Learning steerable filters for rotation equivariant cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 849–858, 2018.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

Qi Xie, Qian Zhao, Zongben Xu, and Deyu Meng. Fourier series expansion based filter parametrization for equivariant convolutions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4537–4551, 2022.

Qi Xie, Jiahong Fu, Zongben Xu, and Deyu Meng. Rotation equivariant arbitrary-scale image super-resolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025.

Renjun Xu, Kaifan Yang, Ke Liu, and Fengxiang He. $E(2)$-equivariant vision transformer. In *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*, volume 216 of *Proceedings of Machine Learning Research*, pp. 2356–2366. PMLR, 2023. URL `https://proceedings.mlr.press/v216/xu23b.html`.

Zhongchen Zhao, Qi Xie, Keyu Huang, Lei Zhang, Deyu Meng, and Zongben Xu. Rotation equivariant mamba for vision tasks. *arXiv preprint arXiv:2603.09138*, 2026.

A APPENDIX

This appendix provides a detailed complexity analysis of Flash EQ-Linear (Sec. A.1), its relation to isotypic-basis equivariant architectures (Sec. A.2), CUDA kernel implementation details (Sec. A.3), and additional experimental results (Sec. A.4). The additional experiments cover extended network-level benchmarks, numerical equivalence and equivariance verification, the scaling of the operator-level speedup with the group size ($T \in \{2, 4, 8\}$), and comparisons with Quick ViTs.

A.1 DETAILED COMPLEXITY ANALYSIS OF FLASH EQ-LINEAR

This section derives the per-step MAC counts in Theorem 1. Following Sec. 3.2, we consider an EQ-Linear layer applied to N tokens, with equal input and output channel widths, an even group size T, and T dividing C. We count real-valued multiply–accumulate operations, excluding standalone additions, subtractions, and fixed power-of-two rescalings.

MACs of Step 1: Group-wise DFT. The group-wise DFT is a complex matrix multiplication along the group dimension, $\hat{\boldsymbol{X}} = \boldsymbol{X} \cdot \boldsymbol{F}$, where $\boldsymbol{F} \in \mathbb{C}^{T\times T}$ is the DFT matrix. Computing the $T/2+1$ non-redundant frequency components therefore amounts to multiplying $\boldsymbol{X} \in \mathbb{R}^{N\times \frac{C}{T}\times T}$ by the corresponding columns of $\boldsymbol{F}$:

$$\hat{\boldsymbol{X}}_{n,k} = \boldsymbol{X}_n \cdot \boldsymbol{F}_k, \quad \text{for } k = 0, \ldots, \frac{T}{2}, \text{ and } n = 0, \ldots, N-1, \tag{9}$$

The DFT coefficients at the zero and Nyquist frequencies ($k = 0, T/2$) are real, so these two components involve real-by-real MACs, while the remaining $T/2-1$ non-redundant components involve real-by-complex MACs. Counting these as one and two real-valued MACs, respectively, gives an upper bound of $2NC + 2\frac{NC}{T}T(T/2-1) = NCT$ MACs. Similarly, transforming $\tilde{\boldsymbol{W}} \in \mathbb{R}^{\frac{C}{T}\times\frac{C}{T}\times T}$ requires C^2 MACs. Step 1 therefore costs at most $NTC + C^2$ MACs.

MACs of Step 2: Per-frequency matrix multiplication (dominant). At each of the $T/2+1$ non-redundant frequencies, Step 2 applies a channel-wise linear projection to each token:

$$\hat{\boldsymbol{Y}}_n^{G_k} = \hat{\boldsymbol{W}}^{G_k} \cdot \hat{\boldsymbol{X}}_n^{G_k}, \quad \text{for } k = 0, \ldots, \frac{T}{2}, \text{ and } n = 0, \ldots, N-1, \tag{10}$$

which maps $\frac{C}{T}$ input channels to $\frac{C}{T}$ output channels, and therefore performs $\frac{NC^2}{T^2}$ multiplications at that frequency. The self-conjugate frequencies $k = 0$ and $k = T/2$ are purely real and each require one real multiplication, whereas the remaining $T/2-1$ are complex and need four real multiplications[3]. Step 2 therefore costs $\frac{2NC^2}{T^2} + 4(T/2-1)\frac{NC^2}{T^2} = \frac{2(T-1)NC^2}{T^2}$ real-valued MACs, which dominates the overall cost of Flash EQ-Linear.

MACs of Step 3: Conjugate-symmetric recovery. The remaining frequency components of $\hat{\boldsymbol{Y}}$ are recovered by conjugate copying according to Eq. 4. This step therefore incurs zero MACs.

MACs of Step 4: Group-wise IDFT. Similar to Step 1, the group-wise IDFT can be written as $\boldsymbol{Y}_n = \hat{\boldsymbol{Y}}_n \cdot \boldsymbol{F}^{-1}$, where $\boldsymbol{F}^{-1} \in \mathbb{C}^{T\times T}$ is the inverse DFT matrix. A direct evaluation involves NCT complex multiplications, corresponding to at most $4NCT$ real-valued MACs. Fortunately, the conjugate symmetry $\hat{\boldsymbol{Y}}_n^{G_{T-k}} = \overline{\hat{\boldsymbol{Y}}_n^{G_k}}$ allows each pair of conjugate terms to be combined into a real-valued expression. For each output group index t, $k = 1, \ldots, T/2-1$, and $n = 0, \ldots, N-1$,

$$\hat{\boldsymbol{Y}}_n^{G_k}\boldsymbol{F}_{t,k}^{-1} + \hat{\boldsymbol{Y}}_n^{G_{T-k}}\boldsymbol{F}_{t,T-k}^{-1} = \hat{\boldsymbol{Y}}_n^{G_k}\boldsymbol{F}_{t,k}^{-1} + \overline{\hat{\boldsymbol{Y}}_n^{G_k}\boldsymbol{F}_{t,k}^{-1}} = 2\,\mathrm{Re}(\hat{\boldsymbol{Y}}_n^{G_k})\mathrm{Re}(\boldsymbol{F}_{t,k}^{-1}) - 2\,\mathrm{Im}(\hat{\boldsymbol{Y}}_n^{G_k})\mathrm{Im}(\boldsymbol{F}_{t,k}^{-1}), \tag{11}$$

Each pair thus requires two real-valued MACs per output scalar instead of eight. The two self-conjugate frequencies, $k = 0$ and $k = T/2$, each contribute one real-valued MAC. Across all NC output scalars, Step 4 therefore requires at most $2NC + 2(T/2-1)NC = NCT$ MACs.

Total MACs. Combining the four steps, the total computational cost of Flash EQ-Linear is

$$\mathrm{MACs}_{\text{Flash}} = \underbrace{NCT + C^2}_{\text{group-wise DFT (Step 1)}} + \underbrace{\frac{2(T-1)NC^2}{T^2}}_{\text{Complex matmul (Step 2)}} + \underbrace{NCT}_{\text{group-wise IDFT (Step 4)}}. \tag{12}$$

[3] Computing a complex product directly requires 4 real multiplications: $(a+b\mathrm{i})(c+d\mathrm{i}) = (ac-bd)+(ad+bc)\mathrm{i}$.

In typical equivariant networks, where $N \gg T$ and $C \gg T$, the dominant term is

$$\mathrm{MACs}_{\mathrm{Flash}} \approx \frac{2(T-1)NC^2}{T^2}, \tag{13}$$

while the group-wise DFT and IDFT contribute only lower-order overhead.

For the common settings $T = 2$ and $T = 4$, the DFT and IDFT simplify to additions, subtractions, and fixed power-of-two rescalings. Under our counting convention, Steps 1 and 4 therefore incur no additional MACs, leaving Step 2 as the sole contribution. The resulting counts are $NC^2/2$ and $3NC^2/8$, respectively.

Speedup analysis. Under equal input and output widths, both naive EQ-Linear and a standard non-equivariant linear layer of the same total width compute a dense $C \times C$ projection and therefore require NC^2 MACs. When per-frequency matrix multiplication dominates the computational cost, Flash EQ-Linear yields a theoretical speedup of

$$\mathrm{Speedup} = \frac{\mathrm{MACs}_{\mathrm{Naive}}}{\mathrm{MACs}_{\mathrm{Flash}}} \approx \frac{NC^2}{2(T-1)NC^2/T^2} = \frac{T^2}{2(T-1)}, \tag{14}$$

which gives $16/6 \approx \mathbf{2.67}\times$ for the $\mathrm{p4}$ rotation group ($T = 4$). This proves Theorem 1.

A.2 Relation to Isotypic-Basis Equivariant Architectures

Shared algebraic foundation. The acceleration principle of Sec. 3.1 is an instance of a classical fact from the representation theory of finite groups: by Schur's lemma, an equivariant linear map between (multiples of) the regular representation becomes block-diagonal once features are expressed in the isotypic basis, i.e., grouped by irreducible representations (Serre, 1977). For cyclic groups, this change of basis is exactly the group-wise DFT. For $\mathrm{p4}$, the real transform in Eq. 6 decomposes the regular representation of the rotation group into the trivial representation (first row), the sign representation (last row), and the two-dimensional rotation representation (middle rows). Since the latter is of complex type, its equivariant maps form complex $\frac{C}{T} \times \frac{C}{T}$ matrices, which is precisely the structure of Eq. 7 and accounts for the $\frac{6}{16}NC^2$ MACs of Theorem 1. The same principle underlies Bökman et al. (2025), who decompose features into symmetric and antisymmetric components under horizontal mirroring ($2\times$ fewer FLOPs), and Nordström et al. (2026), who extend it to the octic group D_8, whose two-dimensional irreducible representation is of real type ($5.33\times$ fewer FLOPs; see also the comparison with Quick ViTs in Appendix A.4). In all three cases, the attainable reduction is determined by the irreducible decomposition of the group rather than by the specific architecture.

Network-level versus layer-level use. The works differ in *where* the change of basis is applied. Bökman et al. (2025) and Nordström et al. (2026) apply it at the network level: features are kept in the isotypic basis throughout the network, pointwise nonlinearities are evaluated by transforming to the spatial basis and back (Bökman et al., 2025), and the resulting architectures are trained from scratch. Flash EQ-Linear applies it at the layer level: each EQ-Linear layer keeps the regular-representation interface and changes basis internally, so that the weights, inputs, and outputs of existing networks remain unchanged. The two routes are therefore complementary rather than competing: the network-level route avoids per-layer transforms at the cost of redesign and retraining, whereas the layer-level route accelerates existing models without either, and the fused-kernel techniques developed here could equally benefit the linear layers of isotypic-basis architectures.

A.3 Detailed CUDA Kernel Implementations

A direct PyTorch realization of Alg. 1 issues the group-wise DFT, the frequency-domain matrix multiplication, and the group-wise IDFT as three separate operators. Each operator launches its own kernel and writes a full-size tensor to GPU device memory, so the activations are read and written three times per call. This overhead is most severe for the $\mathrm{p4}$ group: its DFT and IDFT involve only additions, subtractions, and dyadic scalings (Eqs. 6 and 8), so their runtime is dominated by memory access rather than arithmetic and can offset the savings of Step 2. We therefore fuse Steps 1–4 into a single tiled matrix-multiplication kernel, in which the DFT and IDFT are applied as a change of basis to each entry when it is loaded and stored, rather than executed as standalone operators.

GPU terminology. We follow the standard CUDA memory hierarchy (Hwu et al., 2022). *Device memory* is the off-chip GPU DRAM that stores all PyTorch tensors; it is visible to every kernel but

slow to access. *Shared memory* is a small on-chip scratchpad allocated per CUDA block and shared by the threads of that block; it is unrelated to host–device unified memory. *Registers* store the local scalar variables of a CUDA thread and are the fastest storage on the GPU. *Tiling* partitions a matrix multiplication into fixed-size sub-matrices (*tiles*) that are copied from device memory into shared memory, so that the inner loop accesses only on-chip memory. A *kernel launch* is one host-side invocation of a CUDA kernel over a grid of threads; its fixed cost is significant when the workload per launch is small.

Fused forward kernel. We flatten the batch and token dimensions into $N = BL$ rows and write $C' = C/T$ for the per-group widths. Each CUDA thread computes the four group components of one output entry $\boldsymbol{Y}_{i,j} \in \mathbb{R}^4$ (row i, output channel j) by traversing the input channels tile by tile, where each tile $\mathcal{P}$ is a group of input channels that is loaded into shared memory at once. The kernel consists of three stages that correspond to the steps of Alg. 1.

(i) Input transform (Step 1). When a tile of $\boldsymbol{X}$ is loaded, Eq. 6 is applied to each entry, producing the four real coordinates $\hat{x}_{ic}^{G_0}$, $\hat{x}_{\mathrm{Re},ic}^{G_1}$, $\hat{x}_{ic}^{G_2}$, and $\hat{x}_{\mathrm{Im},ic}^{G_1}$, i.e., the entries of $\hat{\boldsymbol{X}}^{G_0}$, $\hat{\boldsymbol{X}}_{\mathrm{Re}}^{G_1}$, $\hat{\boldsymbol{X}}^{G_2}$, and $\hat{\boldsymbol{X}}_{\mathrm{Im}}^{G_1}$. Since the transform matrix contains only 0 and ± 1, this stage reduces to four add/subtract expressions; neither an FFT routine nor a complex tensor is used.

(ii) Frequency-domain accumulation (Step 2). With $\hat{w}_{jc}^{G_k}$ denoting the entries of the cached weight planes, four accumulators held in registers collect the products of Eq. 7 over all tiles:

$$
\begin{aligned}
a_{ij}^{G_0} &= \sum_{\mathcal{P}} \sum_{c \in \mathcal{P}} \hat{x}_{ic}^{G_0}\, \hat{w}_{jc}^{G_0}, \qquad\qquad a_{ij}^{G_2} = \sum_{\mathcal{P}} \sum_{c \in \mathcal{P}} \hat{x}_{ic}^{G_2}\, \hat{w}_{jc}^{G_2}, \\
a_{\mathrm{Re},ij}^{G_1} &= \sum_{\mathcal{P}} \sum_{c \in \mathcal{P}} \big(\hat{x}_{\mathrm{Re},ic}^{G_1}\, \hat{w}_{\mathrm{Re},jc}^{G_1} - \hat{x}_{\mathrm{Im},ic}^{G_1}\, \hat{w}_{\mathrm{Im},jc}^{G_1} \big), \\
a_{\mathrm{Im},ij}^{G_1} &= \sum_{\mathcal{P}} \sum_{c \in \mathcal{P}} \big(\hat{x}_{\mathrm{Re},ic}^{G_1}\, \hat{w}_{\mathrm{Im},jc}^{G_1} + \hat{x}_{\mathrm{Im},ic}^{G_1}\, \hat{w}_{\mathrm{Re},jc}^{G_1} \big).
\end{aligned}
\tag{15}
$$

Eq. 15 is Eq. 7 written entry-wise. All six real products are accumulated in a single pass over the input channels and share one load of each tile, instead of being executed as six separate GEMM calls on intermediate tensors.

(iii) Output transform (Steps 3–4). The redundant component $\hat{\boldsymbol{Y}}^{G_3}$ is never formed. The inverse transform of Eq. 8, which uses only the constants $\{0, \pm\frac{1}{4}, \pm\frac{1}{2}\}$, is applied to the four accumulators, and the result plus the bias $\tilde{b}_j$ is written directly into $\boldsymbol{Y}_{i,j}$.

Consequently, neither the unrolled matrix $\boldsymbol{W}$ nor the full intermediate tensors $\hat{\boldsymbol{X}}$ and $\hat{\boldsymbol{Y}}$ are materialized in global memory; only $\boldsymbol{X}$, the cached weight planes, and $\boldsymbol{Y}$ reside there. The kernel preserves the mathematical computation in Sec. 3.2 while reorganizing its execution and memory access.

Memory layout and weight caching. Activations keep the external layout $[B, L, C', T]$ with the group dimension contiguous, so the four group components of each entry are adjacent in memory and are read in a single access before being transformed. The four weight planes are computed once from $\tilde{\boldsymbol{W}}$ by Eq. 6 and cached in device memory, so Step 1 on the weight side is amortized over all rows and calls; the cache is invalidated whenever the parameters are updated in place. No packed copy of the activations is allocated. Under FP16, tiles are held in half precision, whereas the accumulators are kept in FP32 to limit round-off error.

Shape-specialized backends. Different shapes (B, L, C', C') expose different bottlenecks. At small widths, kernel-launch overhead and partially filled tiles dominate the runtime, whereas at large widths arithmetic throughput dominates, so no single tile size is fastest across the whole range. We therefore implement four backends (Table 6) that compute the same quantities as the fused kernel above but differ in multiplication routine, launch strategy, and tile staging, together with per-shape adjustments such as tile padding for narrow output widths C'. Each backend is compiled for a grid of tile sizes and thread-grid configurations, so the number of specialized kernels exceeds the backends listed.

Forward and backward passes. The forward pass uses the full backend family described above. The backward pass expresses the output gradient $\nabla \boldsymbol{Y}$ in the same four frequency coordinates, forms the transposed products of Eq. 7, and maps the results back through the fixed transforms, yielding $\nabla \boldsymbol{X}$, $\nabla \tilde{\boldsymbol{W}}$, and $\nabla \tilde{\boldsymbol{b}}$ in the original parameterization rather than in any backend-specific layout. Because these gradients do not depend on which forward backend was used, a single fused backward kernel

Table 6: **Shape-specialized backends of Flash EQ-Linear** for the $\mathrm{p4}$ group. All backends compute Eqs. 6, 15, and 8 and agree up to floating-point round-off; they differ in which routine executes the matrix multiplication, how kernels are launched, and how tiles are staged.

Backend	Multiplication routine	Specialization	Target regime
Tiled	Custom tiled kernel	One kernel launch per call; tiles in shared memory, accumulators in registers	Small C', short sequences
Tiled-persistent	Custom tiled kernel	Kernel instances launched once and reused over a queue of output tiles, removing per-call launch overhead	Small to medium C'
Tiled-static	Custom tiled kernel	Shapes fixed at compile time to remove boundary checks; group components loaded in vectorized form through two staging buffers	Medium to large C'
CUTLASS	CUTLASS GEMM templates	Tensor layouts expressed with CuTe; tiles staged by asynchronous copies	Large C'

Table 7: Configuration details, parameter counts, and total FLOPs of the non-equivariant ViT and Swin Transformer baselines used in our experiments. Their equivariant counterparts use the same total channel widths at each model scale.

Model	Scale	Patch Size	Embedding Dimension	Depth/Depths	Num. Heads	#Param. (M)↓	FLOPs (G)↓
ViT	Tiny	16	384	12	3	21.7	9.2
	Small	16	480	12	6	33.8	14.1
	Base	16	768	12	12	85.9	35.1
	Large	16	1024	24	16	303.4	123.1
	Huge	16	1280	32	16	631.0	254.7
Swin	Tiny	4	$[96, 192, 384, 768]$	$[2, 2, 6, 2]$	$[3, 6, 12, 24]$	27.6	8.99
	Small	4	$[96, 192, 384, 768]$	$[2, 2, 18, 2]$	$[3, 6, 12, 24]$	48.9	17.5
	Base	4	$[128, 256, 512, 1024]$	$[2, 2, 18, 2]$	$[4, 8, 16, 32]$	86.8	30.9
	Large	4	$[192, 384, 768, 1536]$	$[2, 2, 18, 2]$	$[6, 12, 24, 48]$	195.1	69.0
	Huge	4	$[352, 704, 1408, 2816]$	$[2, 2, 18, 2]$	$[11, 22, 44, 88]$	655.1	230.3

per precision is correct for every forward backend, and we share one such kernel across all of them instead of specializing the backward pass per shape. This choice is deliberate: forward latency dominates the inference setting that Flash EQ-Linear targets, and a single backward kernel allows the entire gradient path to be verified against the naive implementation in one comparison (Table 10). Even without specialization, the shared backward kernel is faster than both baselines at every channel width (Table 2); per-shape specialization of the backward pass is left for future work.

A.4 More Experimental Results

Runtime measurement protocol. All timings use CUDA events with synchronization before and after each timed loop, exclude input generation, host–device transfers, and model initialization. **(i) Operator benchmarks** are averaged over 200 iterations after 5 warm-up iterations. Naive EQ-Linear re-expands its weights on every forward call, and backward timing excludes the forward pass and optimizer updates. **(ii) Network benchmarks** run in evaluation mode without gradient tracking and are averaged over 200 forward passes after 20 warm-up iterations. Linear-layer latency is profiled separately with CUDA-event hooks, which are removed for end-to-end timing, so the reported latency ratio is a profiling-based estimate. We report per-image latency (batch latency divided by B) and the corresponding throughput. FP16 runs cast parameters and inputs to half precision without automatic mixed precision.

Configuration details of ViT and Swin. Table 7 lists the architectural configurations of the ViT and Swin Transformer variants (Dosovitskiy et al., 2021; Liu et al., 2021) used in Sec. 4.2. Each family spans five scales of increasing capacity (Tiny / Small / Base / Large / Huge). ViT scales the embedding

Table 8: **Inference efficiency comparison** of Flash EQ-ViT against standard non-EQ and naive EQ baselines on ImageNet-100. *Speedup ratios* (green) are measured against standard baselines.

Method	#Param. (M)↓	Top-1 (%)↑	All Linear Layers			Total Network		
			FLOPs (G)↓	Latency (ms)↓	Latency Ratio	FLOPs (G)↓	Latency (ms)↓	Throughput (imgs/s)↑
Single-Precision Floating-Point (FP32)								
ViT-T	21.7	76.5	8.4	0.218	60.8%	9.2	0.358	2796
Naive EQ-ViT-T	**5.4**	**77.3**	8.4	0.217	60.3%	9.2	0.359	2785
Flash EQ-ViT-T	**5.4**	**77.3**	**3.2**	**0.115**	**46.2%**	**4.0**	**0.249**	**4012** (1.4×)
ViT-S	33.8	76.5	13.1	0.352	58.4%	14.1	0.603	1658
Naive EQ-ViT-S	**8.4**	**78.2**	13.1	0.355	58.7%	14.1	0.604	1656
Flash EQ-ViT-S	**8.4**	**78.2**	**5.0**	**0.183**	**42.6%**	**6.0**	**0.429**	**2332** (1.4×)
ViT-B	85.9	77.1	33.5	0.761	67.1%	35.1	1.135	881
Naive EQ-ViT-B	**21.4**	**80.1**	33.5	0.757	67.1%	35.1	1.128	886
Flash EQ-ViT-B	**21.4**	**80.1**	**12.6**	**0.400**	**52.3%**	**14.3**	**0.764**	**1309** (1.5×)
ViT-L	303.4	78.4	119.0	2.672	71.9%	123.1	3.715	269
Naive EQ-ViT-L	**75.8**	**80.6**	119.0	2.760	73.9%	123.1	3.735	268
Flash EQ-ViT-L	**75.8**	**80.6**	**44.9**	**1.336**	**56.7%**	**49.0**	**2.356**	**424** (1.6×)
ViT-H	631.0	81.3	248.0	5.586	77.7%	254.7	7.192	139
Naive EQ-ViT-H	**157.7**	**81.5**	248.0	5.627	78.4%	254.7	7.182	139
Flash EQ-ViT-H	**157.7**	**81.5**	**93.4**	**2.719**	**63.2%**	**100.1**	**4.304**	**232** (1.7×)
Half-Precision Floating-Point (FP16)								
ViT-T	21.7	76.5	8.4	0.074	57.4%	9.2	0.128	7794
Naive EQ-ViT-T	**5.4**	**77.3**	8.4	0.075	57.3%	9.2	0.130	7685
Flash EQ-ViT-T	**5.4**	**77.3**	**3.2**	**0.058**	**51.3%**	**4.0**	**0.113**	**8848** (1.1×)
ViT-S	33.8	76.5	13.1	0.105	52.6%	14.1	0.199	5030
Naive EQ-ViT-S	**8.4**	**78.2**	13.1	0.109	54.5%	14.1	0.200	5010
Flash EQ-ViT-S	**8.4**	**78.2**	**5.0**	**0.084**	**48.4%**	**6.0**	**0.174**	**5750** (1.1×)
ViT-B	85.9	77.1	33.5	0.215	57.6%	35.1	0.372	2688
Naive EQ-ViT-B	**21.4**	**80.1**	33.5	0.218	58.3%	35.1	0.374	2676
Flash EQ-ViT-B	**21.4**	**80.1**	**12.6**	**0.147**	**49.1%**	**14.3**	**0.300**	**3330** (1.2×)
ViT-L	303.4	78.4	119.0	0.742	59.0%	123.1	1.258	795
Naive EQ-ViT-L	**75.8**	**80.6**	119.0	0.752	59.6%	123.1	1.262	792
Flash EQ-ViT-L	**75.8**	**80.6**	**44.9**	**0.497**	**49.5%**	**49.0**	**1.004**	**996** (1.3×)
ViT-H	631.0	81.3	248.0	1.541	64.8%	254.7	2.379	420
Naive EQ-ViT-H	**157.7**	**81.5**	248.0	1.557	65.1%	254.7	2.392	418
Flash EQ-ViT-H	**157.7**	**81.5**	**93.4**	**0.976**	**54.2%**	**100.1**	**1.801**	**555** (1.3×)

dimension C and the number of attention heads, from 21.7M parameters at ViT-T to 631.0M at ViT-H; Swin retains its four-stage hierarchy and scales the per-stage embedding dimensions, depths, and head counts, from 27.6M at Swin-T to 655.1M at Swin-H. The Naive EQ and Flash EQ variants share these configurations exactly but operate on the $\mathrm{p4}$ rotation group ($T = 4$): the total channel width C is matched to the non-equivariant baseline at every scale, so the per-group width is C/T. This matching is what makes the comparison like-for-like—at a given scale the three variants differ only in how the same $C \times C$ projection is computed, not in its size.

Flash EQ-ViT and Flash EQ-Swin. Tables 8 and 9 report the complete results for ViT and for Swin respectively, each covering all five scales in both FP32 and FP16. **(i)** Flash EQ-Linear reduces the cumulative linear-layer latency by $\sim \mathbf{2}\times$ over the Naive EQ baseline at every scale in FP32, and by $1.1\times$–$1.6\times$ in FP16, yielding $\mathbf{1.1}\times$–$\mathbf{1.7}\times$ end-to-end speedup across the twenty configurations. The FP16 margin is smaller because half precision roughly doubles arithmetic throughput, leaving the same layer closer to the memory-bound regime; the effect is strongest for Swin, whose windowed attention uses narrower linear layers than ViT at matched total width. **(ii)** The end-to-end speedup tracks the fraction of runtime the linear layers occupy rather than model size as such, as established in Sec. 4.2. That fraction generally grows with scale and the measured speedups follow it in both directions, with ViT-T and ViT-S both at $\mathbf{1.4}\times$ while ViT-H reaches $\mathbf{1.7}\times$ as its linear layers come to dominate the runtime. Across all scales and both families, Flash EQ-ViT/Swin reproduces the

Table 9: **Inference efficiency comparison** of Flash EQ-Swin against standard non-EQ and naive EQ baselines on ImageNet-100. *Speedup ratios* (green) are measured against standard baselines.

Method	#Param. (M)↓	Top-1 (%)↑	All Linear Layers			Total Network		
			FLOPs (G)↓	Latency (ms)↓	Latency Ratio	FLOPs (G)↓	Latency (ms)↓	Throughput (imgs/s)↑
Single-Precision Floating-Point (FP32)								
Swin-T	27.6	87.1	8.68	0.291	46.1%	8.99	0.631	1585
Naive EQ-Swin-T	**6.9**	**87.3**	8.68	0.300	47.4%	8.99	0.633	1580
Flash EQ-Swin-T	**6.9**	**87.3**	**3.55**	**0.142**	**29.0%**	**3.85**	**0.490**	**2041** (1.3×)
Swin-S	48.9	87.6	17.0	0.513	49.3%	17.5	1.040	962
Naive EQ-Swin-S	**12.3**	**87.7**	17.0	0.527	50.3%	17.5	1.047	956
Flash EQ-Swin-S	**12.3**	**87.7**	**6.72**	**0.276**	**34.9%**	**7.20**	**0.791**	**1265** (1.3×)
Swin-B	86.8	87.8	30.2	0.812	53.2%	30.9	1.527	655
Naive EQ-Swin-B	**21.8**	**88.6**	30.2	0.806	52.6%	30.9	1.534	652
Flash EQ-Swin-B	**21.8**	**88.6**	**11.9**	**0.442**	**38.4%**	**12.5**	**1.151**	**868** (1.3×)
Swin-L	195.1	87.5	68.0	1.661	59.4%	69.0	2.799	357
Naive EQ-Swin-L	**48.9**	**88.2**	68.0	1.672	59.7%	69.0	2.800	357
Flash EQ-Swin-L	**48.9**	**88.2**	**26.6**	**0.875**	**44.1%**	**27.6**	**1.983**	**504** (1.4×)
Swin-H	655.1	87.2	228.6	5.239	71.0%	230.3	7.381	135
Naive EQ-Swin-H	**164.0**	**88.3**	228.6	5.273	71.3%	230.3	7.396	135
Flash EQ-Swin-H	**164.0**	**88.3**	**89.1**	**2.639**	**56.2%**	**90.8**	**4.694**	**213** (1.6×)
Half-Precision Floating-Point (FP16)								
Swin-T	27.6	87.1	8.68	0.088	37.8%	8.99	0.232	4315
Naive EQ-Swin-T	**6.9**	**87.3**	8.68	0.091	39.1%	8.99	0.233	4292
Flash EQ-Swin-T	**6.9**	**87.3**	**3.55**	**0.085**	**38.5%**	**3.85**	**0.220**	**4554** (1.1×)
Swin-S	48.9	87.6	17.0	0.168	46.0%	17.5	0.366	2733
Naive EQ-Swin-S	**12.3**	**87.7**	17.0	0.177	48.2%	17.5	0.367	2727
Flash EQ-Swin-S	**12.3**	**87.7**	**6.72**	**0.150**	**45.0%**	**7.20**	**0.332**	**3009** (1.1×)
Swin-B	86.8	87.8	30.2	0.238	44.9%	30.9	0.529	1889
Naive EQ-Swin-B	**21.8**	**88.6**	30.2	0.245	45.7%	30.9	0.535	1869
Flash EQ-Swin-B	**21.8**	**88.6**	**11.9**	**0.185**	**39.4%**	**12.5**	**0.470**	**2129** (1.1×)
Swin-L	195.1	87.5	68.0	0.463	48.3%	69.0	0.958	1044
Naive EQ-Swin-L	**48.9**	**88.2**	68.0	0.469	48.8%	69.0	0.961	1041
Flash EQ-Swin-L	**48.9**	**88.2**	**26.6**	**0.317**	**38.8%**	**27.6**	**0.817**	**1224** (1.2×)
Swin-H	655.1	87.2	228.6	1.432	59.3%	230.3	2.414	414
Naive EQ-Swin-H	**164.0**	**88.3**	228.6	1.445	59.8%	230.3	2.418	414
Flash EQ-Swin-H	**164.0**	**88.3**	**89.1**	**0.958**	**49.0%**	**90.8**	**1.955**	**512** (1.2×)

Table 10: **Validation of numerical equivalence** between Flash EQ-Linear and Naive EQ-Linear in FP32. The relative L_2 error (Rel L_2) is the primary metric.

Tensor	Dtype	Shape	Rel L_2 ↓	Mean Abs ↓	p99 Rel ↓	Match
Forward output $\boldsymbol{Y}$	FP32	B×L×C/T×T	2.6×10^{-7}	1.1×10^{-7}	1.7×10^{-6}	✓
Backward input grad $\nabla\boldsymbol{X}$	FP32	B×L×C/T×T	2.2×10^{-7}	9.4×10^{-8}	1.1×10^{-5}	✓
Backward weight grad $\nabla\tilde{\boldsymbol{W}}$	FP32	C/T×C/T×T	5.9×10^{-7}	2.9×10^{-5}	3.2×10^{-5}	✓
Backward bias grad $\nabla\tilde{\boldsymbol{b}}$	FP32	C/T	1.5×10^{-7}	7.2×10^{-6}	4.5×10^{-6}	✓

ImageNet-100 Top-1 accuracy of its Naive EQ counterpart exactly, as guaranteed by the exactness established in Sec. 3.2.

Exactness and equivariance verification. We provide empirical support for the two properties established in Sec. 3.2: numerical equivalence to Naive EQ-Linear, and preservation of rotation equivariance. **(i) Numerical equivalence.** We compare Flash EQ-Linear with Naive EQ-Linear on random inputs of shape $(B, L, {}^{C}/_{T}, T) = (1, 1024, 64, 4)$ over a complete training step, i.e., for $\boldsymbol{Y}$, $\nabla\boldsymbol{X}$, $\nabla\tilde{\boldsymbol{W}}$ and $\nabla\tilde{\boldsymbol{b}}$, using the naive results as the reference. The primary metric is the relative L_2 error, i.e., the L_2 norm of the difference divided by that of the reference. We also report the mean absolute error and the 99-th percentile of the element-wise relative error, computed over

Table 11: **Equivariance verification** of Flash EQ-Linear on the p4 rotation group in FP32.

Method	NMAE ↓	Max Abs ↓	Mean Abs ↓	p99 Rel ↓	Equivariant
Standard Linear	1.4×10^{0}	4.1×10^{0}	6.5×10^{-1}	6.4×10^{1}	✗
Naive EQ-Linear	1.3×10^{-7}	7.3×10^{-7}	6.0×10^{-8}	7.5×10^{-6}	✓
Flash EQ-Linear	$\mathbf{4.0\times10^{-8}}$	$\mathbf{3.2\times10^{-7}}$	$\mathbf{1.9\times10^{-8}}$	$\mathbf{2.4\times10^{-6}}$	✓

Table 12: **Forward latency scaling with the group size T.** Flash EQ-Linear (PyTorch) is compared against Standard Linear and Naive EQ-Linear baselines for $T \in \{2, 4, 8\}$ under different channel numbers ($C/T = 64 \to 2048$), measured in FP32 with $B = 32$ and $L = 1024$. #Param. and FLOPs are reported at a fixed total width $C = 2048$. *Speedup ratios* (green; gray if <1) are measured against the standard Linear (PyTorch's `F.linear`) baseline.

Method	#Param. (M)↓	FLOPs (G)↓	Forward Latency (ms)↓					
			64	128	256	512	1024	2048
Group size $T=2$ (theoretical speedup $T^2/2(T-1) = \mathbf{2.00\times}$)								
Standard Linear	4.194	8.590	**0.04**	**0.11**	**0.46**	**1.68**	6.73	27.86
Naive EQ-Linear	**2.097**	8.590	0.08	0.12	0.51	1.69	6.83	28.18
Flash EQ-Linear (PyTorch)	**2.097**	**4.301**	0.17	0.34	0.80	1.98	**5.58**	**17.25**
Speedup/Reduction ↑	**50.0%**	**49.9%**	0.2×	0.3×	0.6×	0.8×	**1.2×**	**1.6×**
Group size $T=4$ (theoretical speedup $T^2/2(T-1) = \mathbf{2.67\times}$)								
Standard Linear	4.194	8.590	**0.11**	**0.43**	**1.52**	6.05	24.48	104.96
Naive EQ-Linear	**1.049**	8.590	**0.11**	0.44	1.54	6.25	25.16	105.74
Flash EQ-Linear (PyTorch)	**1.049**	**3.233**	0.52	0.99	2.24	**5.82**	**16.16**	**51.89**
Speedup/Reduction ↑	**75.0%**	**62.4%**	0.2×	0.4×	0.7×	**1.0×**	**1.5×**	**2.0×**
Group size $T=8$ (theoretical speedup $T^2/2(T-1) = \mathbf{4.57\times}$)								
Standard Linear	4.194	8.590	**0.54**	1.90	7.25	28.95	120.23	570.75
Naive EQ-Linear	**0.524**	8.590	0.55	**1.89**	7.22	29.19	120.99	574.21
Flash EQ-Linear (PyTorch)	**0.524**	**1.906**	1.01	3.12	**7.01**	**16.91**	**44.94**	**138.08**
Speedup/Reduction ↑	**87.5%**	**77.8%**	0.5×	0.6×	**1.0×**	**1.7×**	**2.7×**	**4.1×**

entries whose reference magnitude exceeds 10^{-1} for $\boldsymbol{Y}$ and 10^{-3} for the gradients to exclude near-zero references. A tensor is marked as matching if its relative L_2 error is below 10^{-5}. As shown in Table 10, the relative L_2 error lies between 10^{-7} and 10^{-6} for every tensor, since the two implementations accumulate the same quantity in a different order. **(ii) Equivariance preservation.** Under the p4 rotation group, rotating the input cyclically permutes its group dimension, so an equivariant layer f must satisfy $f(G(\boldsymbol{X})) = G(f(\boldsymbol{X}))$ for every $G \in \mathcal{G}$. Following Xie et al. (2025), we measure the discrepancy between the two sides with the normalized mean absolute error, $\mathrm{NMAE} = \mathrm{mean}\,|f(G(\boldsymbol{X})) - G(f(\boldsymbol{X}))| \,/\, \mathrm{mean}\,|G(f(\boldsymbol{X}))|$: values of order 1 indicate that the two sides are unrelated, whereas values near the FP32 unit round-off ($2^{-24} \approx 6\times10^{-8}$) indicate equivariance up to rounding. We additionally report the maximum and mean absolute discrepancy and the 99-th percentile of the relative discrepancy over entries with $|G(f(\boldsymbol{X}))| > 10^{-1}$, and mark a layer as equivariant if its NMAE is below 10^{-5}, as in (i). In Table 11, Standard Linear yields an NMAE of $\mathbf{1.4}$, confirming that an unconstrained layer does not commute with the group action. In contrast, Flash EQ-Linear yields $\mathbf{4.0\times10^{-8}}$, comparable to Naive EQ-Linear ($\mathbf{1.3\times10^{-7}}$) and more than seven orders of magnitude below Standard Linear, showing that equivariance is preserved up to floating-point rounding.

Scaling with the group size T. Theorem 1 predicts that the speedup of Flash EQ-Linear grows with the group size T. To verify this, Table 12 extends the operator-level comparison of Table 1 from the p4 group ($T = 4$) to $T \in \{2, 4, 8\}$, using the same PyTorch implementation strategy for every T. Standard Linear and Naive EQ-Linear both compute a dense $C \times C$ projection with NC^2 MACs, whereas Flash EQ-Linear requires only $2(T-1)NC^2/T^2$ MACs: two real matrix multiplications at the zero and Nyquist frequencies and $T/2-1$ complex ones, each mapping C/T input channels to C/T output channels. The theoretical speedup $T^2/2(T-1)$ therefore rises from $2.00\times$ ($T=2$) to $2.67\times$ ($T=4$) and $4.57\times$ ($T=8$). At a fixed total width $C = 2048$, the resulting FLOPs reductions (49.9%,

Table 13: Comparison of standard ViT, naive EQ-ViT, Quick ViTs, and Flash EQ-ViT on ImageNet-100 in FP32. Within each scale, the best Top-1 accuracy, latency, and throughput are shown in **bold** and the second best are underlined; parameter counts and FLOPs are reported as budgets and are not ranked. *Speedup ratios* (green) are measured against standard ViT.

Method	#Param. (M)↓	FLOPs (G)↓	Top-1 (%)↑	Latency (ms)↓	Throughput (imgs/s)↑
ViT-T (Dosovitskiy et al., 2021)	21.7	9.2	76.5	0.358	2796
Naive EQ-ViT-T (Fu et al., 2026)	5.4	9.2	**77.3**	0.359	2785
Quick ViT-$\mathcal{D}_8$-T (Nordström et al., 2026)	2.8	1.7	71.6	0.304	3294 (1.2×)
Quick ViT-$\mathcal{H}_8$-T (Nordström et al., 2026)	4.3	2.3	76.0	0.305	3276 (1.2×)
Quick ViT-$\mathcal{I}_8$-T (Nordström et al., 2026)	4.5	2.3	76.9	0.307	3257 (1.2×)
Flash EQ-ViT-T	5.4	4.0	**77.3**	**0.249**	**4012** (1.4×)
ViT-S (Dosovitskiy et al., 2021)	33.8	14.1	76.5	0.603	1658
Naive EQ-ViT-S (Fu et al., 2026)	8.4	14.1	78.2	0.604	1656
Quick ViT-$\mathcal{D}_8$-S (Nordström et al., 2026)	4.3	2.6	76.3	0.445	2249 (1.4×)
Quick ViT-$\mathcal{H}_8$-S (Nordström et al., 2026)	6.7	3.5	78.3	0.451	2220 (1.3×)
Quick ViT-$\mathcal{I}_8$-S (Nordström et al., 2026)	6.9	3.6	**78.7**	0.454	2203 (1.3×)
Flash EQ-ViT-S	8.4	6.0	78.2	**0.429**	**2332** (1.4×)
ViT-B (Dosovitskiy et al., 2021)	85.9	35.1	77.1	1.135	881
Naive EQ-ViT-B (Fu et al., 2026)	21.4	35.1	**80.1**	1.128	886
Quick ViT-$\mathcal{D}_8$-B (Nordström et al., 2026)	10.8	6.5	74.7	**0.720**	**1389** (1.6×)
Quick ViT-$\mathcal{H}_8$-B (Nordström et al., 2026)	29.5	13.3	77.3	0.811	1233 (1.4×)
Quick ViT-$\mathcal{I}_8$-B (Nordström et al., 2026)	29.9	13.5	77.0	0.815	1227 (1.4×)
Flash EQ-ViT-B	21.4	14.3	**80.1**	0.764	1309 (1.5×)

62.4%, and 77.8%) lie within 0.4 percentage points of the ideal $1-2(T-1)/T^2$, since the group-wise DFT and IDFT contribute only lower-order cost. The measured latency follows the same trend. At $C/T = 2048$, Flash EQ-Linear (PyTorch) is $1.6\times$, $2.0\times$, and $4.1\times$ faster than `F.linear` for $T = 2, 4, 8$, respectively, and for every $C/T \geq 128$ the speedup increases monotonically with T (e.g., $1.2\times \to 1.5\times \to 2.7\times$ at $C/T = 1024$). The channel width at which Flash EQ-Linear first outperforms `F.linear` also halves with each doubling of T ($C/T = 1024$, 512, and 256): at a fixed C/T, the dense cost grows as T^2, whereas the memory traffic, layout transformations, and kernel-launch overhead of the unfused implementation grow only linearly in T. The numerical accuracy is likewise unaffected by T. In FP32, the Rel L_2 error remains below 10^{-6} for every T, the magnitude expected from floating-point round-off as in the numerical-equivalence test above.

Comparison with Quick ViTs. Quick ViTs (Nordström et al., 2026) build equivariance into the architecture by parameterizing features in the isotypic basis of the octic group D_8, which comprises 90° rotations and reflections. Because this group is twice as large as ours ($|\mathrm{D}_8| = 8$ versus $T = 4$), its equivariant linear layers admit $5.33\times$ fewer FLOPs and $8\times$ fewer parameters than a dense layer—roughly twice the arithmetic reduction available to our formulation ($2.67\times$). These larger theoretical savings make Quick ViTs a useful comparison for assessing how effectively our fused implementation translates arithmetic reductions into practical acceleration. We compare Flash EQ-ViT with three Quick ViT configurations, denoted $\mathcal{D}_8$, $\mathcal{I}_8$, and $\mathcal{H}_8$. The $\mathcal{D}_8$ variant retains octic-equivariant blocks throughout the backbone, whereas the hybrid $\mathcal{I}_8$ and $\mathcal{H}_8$ variants combine equivariant and standard blocks through tokenwise invariantization and relaxation of equivariance, respectively. We adjust the proportion of equivariant and standard blocks in $\mathcal{I}_8$ and $\mathcal{H}_8$ to obtain comparable parameter budgets and train them on ImageNet-100 from scratch. All measurements are obtained under identical experimental settings on a single NVIDIA RTX 4090 GPU and averaged over 200 runs.

As shown in Table 13, Flash EQ-ViT is generally faster than $\mathcal{H}_8$ and $\mathcal{I}_8$ at comparable accuracy, and substantially more accurate than $\mathcal{D}_8$. Quick ViTs are equivariant to the larger octic group D_8 and thus admit a larger arithmetic reduction ($5.33\times$ vs. $2.67\times$). Even so, at the Tiny scale Flash EQ-ViT is $\mathbf{1.22}\times$ faster than the fastest Quick ViT variant while being more accurate, which we attribute to our fused CUDA kernels. As width grows, the larger arithmetic reduction of D_8 increasingly offsets this kernel-level advantage: at the Base scale, Quick ViT-$\mathcal{D}_8$ is $1.06\times$ faster but 5.4 points less accurate (Table 13). Notably, Quick ViTs are new architectures trained from scratch, whereas Flash EQ-Linear accelerates existing models with no retraining and no accuracy change.