EVOLVING LAYER-SPECIFIC SCALAR FUNCTIONS FOR HARDWARE-AWARE TRANSFORMER ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision Transformers (ViTs) achieve state-of-the-art performance on challenging vision tasks, but their deployment on edge devices is hindered by the computational complexity and global reduction bottleneck imposed by layer normalization. Recent methods attempt to bypass this by replacing normalization layers with hardware-friendly scalar approximations. However, these homogeneous replacements do not optimally fit to all layers' behaviour and rely on expensive model retraining. In this work, we propose a highly efficient, hardware-aware framework that utilizes genetic programming (GP) to evolve heterogeneous, layer-specific scalar functions directly from pre-trained weights. Coupled with a novel post-training re-alignment strategy, our approach eliminates the need to retrain models from scratch entirely. Our evolved expressions accurately approximate the target normalization behaviours, capturing 90–93% of the variance (R^2) compared to only 70–76% for homogeneous baselines, allowing our modified architecture to recover 84.32% Top-1 ImageNet-1K accuracy on ViT-B and 85.74% on ViT-L in only 20 epochs. By retaining near-baseline accuracy while eliminating the global reduction bottleneck, our approach achieves a strict reduction in both arithmetic complexity and off-chip memory traffic compared to standard LayerNorm, removing a primary barrier to the efficient deployment of ViTs on edge accelerators.

1 INTRODUCTION

Vision Transformers (ViTs) have emerged as state-of-the-art architectures across a wide range of computer vision tasks, achieving remarkable success in image classification, object detection, and semantic segmentation (Cao et al., 2025; Carion et al., 2020; Dosovitskiy et al., 2020; Thisanke et al., 2023). By leveraging self-attention mechanisms to capture global context and long-range dependencies, ViTs consistently rival or outperform traditional convolutional neural networks, making them increasingly popular choices in modern computer vision tasks and research (Liu et al., 2021; Raghu et al., 2021). However, their high computational complexity and reliance on operations such as layer normalization and Softmax severely hinder their deployment on resource-constrained edge devices. These non-linear components introduce complex operations that incur considerable latency and energy consumption, making real-time inference on mobile platforms, FPGAs, and ASICs notoriously difficult (Marino et al., 2023; Sadeghi et al., 2024; Xiao et al., 2025; Zhao et al., 2025). Because linear layers are now routinely optimized using low-precision integer quantization, the performance bottleneck in AI hardware has fundamentally shifted toward memory bandwidth and data movement (Sun et al., 2025; Xu et al., 2025). As a result, these complex non-linear operations disrupt optimized dataflows and consume a disproportionate amount of on-chip resources, occupying up to 60% of memory usage and nearly 40% of total execution time (Chen et al., 2024; Tabani et al., 2021).

Among such non-linear operations, layer normalization (LayerNorm) is particularly costly relative to its arithmetic. Unlike batch normalization (BN) in convolutional networks, where the mean and variance can be calculated during training and folded into the preceding linear layers for inference, LayerNorm requires recalculating the mean and variance at the token level during inference (Chen et al., 2025; Kim et al., 2025). This process involves a reduction operation across the feature dimension, which we define as the global reduction bottleneck. This operation creates a cross-feature data dependency that prevents layer fusion and introduces inter-tile dependencies. Because the final normalization cannot occur until these statistics are computed sequentially, the system is forced into

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 multi-pass execution that repeatedly reads and writes intermediate data to off-chip memory, rapidly saturating memory bandwidth, driving up both inference latency and energy consumption (Bochem et al., 2025; Chen et al., 2024; 2025; Dong et al., 2024; Lee et al., 2024; Liu et al., 2023; Marino et al., 2023; Qiu et al., 2026; Sun et al., 2025; Wang et al., 2023; Yu et al., 2022; Zhao et al., 2025). Beyond this unavoidable data movement, the dynamic calculation of these token-dependent statistics imposes a high arithmetic stalling cost. Because the subsequent operations cannot proceed until the variance is computed and inverted, the primary computing units can be forced to idle, reducing end-to-end performance and system utilization on such hardware (Xiao et al., 2025; Xu et al., 2025).

To bypass these hardware and memory bottlenecks, recent works have explored various hardware-friendly approximations and normalization-free designs. Some approaches replace LayerNorm with BN to enable offline parameter folding (Chen et al., 2025), while others substitute complex non-linearities with integer-based polynomial fitting or Newton-Raphson iterations (Kim et al., 2025; Sun et al., 2025). A notable advancement by Zhu et al. (2025) proposes replacing LayerNorm with an element-wise operation called Dynamic Tanh (DyT), defined as $\text{DyT}(x) = \tanh(\alpha x)$. This is based on the observation that LayerNorm mappings follow tanh-like, S-shaped curves. By replacing global statistics with a scalar mapping, DyT effectively eliminates the reduction bottleneck while achieving near-identical performance in ViTs. However, this method adopts a single, homogeneous scalar function for all LayerNorm layers using only a learnable scaling parameter α for flexibility. This overlooks the fact that normalization layers operate differently depending on network depth, transitioning from near-linear mappings in early blocks to the S-shaped curves observed in deeper ones (Zhu et al., 2025). Additionally, this scalar replacement has primarily been evaluated through intensive from scratch training of 300 epochs or more (Zhu et al., 2025). While this establishes the replacement's viability, the requirement for extensive retraining presents a significant barrier to the rapid hardware adaptation of existing pre-trained models. These factors highlight the need for a versatile framework capable of discovering layer-specific functions and adapting them efficiently to pre-trained architectures without the overhead of from-scratch training.

We address these challenges with a two-fold hardware-aware adaptation strategy for pre-trained ViTs, summarized in Figure 1. First, we introduce a symbolic discovery framework using genetic programming (GP) (Koza, 1994) to evolve layer-specific scalar functions. While prior work (Zhu et al., 2025) demonstrated the viability of fixed scalar replacements, we automate this process via symbolic regression on the input-to-output mappings of pre-trained normalization layers. This allows our framework to discover heterogeneous expressions that capture the distinct normalization behaviours observed across the network. Second, we propose an efficient post-training re-alignment strategy that recovers the model's feature space in a fraction of the standard training duration by leveraging existing LayerNorm weights and biases.

Our main contributions are summarized as follows:

- We introduce a framework that utilizes genetic programming to evolve heterogeneous, layer-specific scalar functions. These achieve significantly higher functional alignment than homogeneous baselines across both ViT-B and ViT-L, capturing 90–93% of the variance (R^2) in the target LayerNorm mappings versus only 70–76% for DyT.
- We propose a computationally efficient post-training re-alignment strategy that leverages pre-trained weights to recover 84.32% Top-1 ImageNet-1K accuracy on ViT-B (and 85.74% on ViT-L) in only 20 epochs, completely avoiding the need for expensive retraining from scratch.
- We eliminate the LayerNorm global reduction bottleneck, restoring an on-chip streaming dataflow that roughly halves normalization memory access. This achieves a strict reduction in both arithmetic complexity and off-chip memory traffic compared to standard LayerNorm, removing a primary barrier to the efficient deployment of ViTs on bandwidth-bound edge accelerators.

The remainder of this paper is organized as follows. Section 2 details our extraction of normalization mappings, the GP symbolic discovery process, and the model re-alignment phase. Section 3 provides a comparative analysis of functional alignment, classification performance, and hardware efficiency. Finally, Section 4 discusses the implications and limitations of our findings.

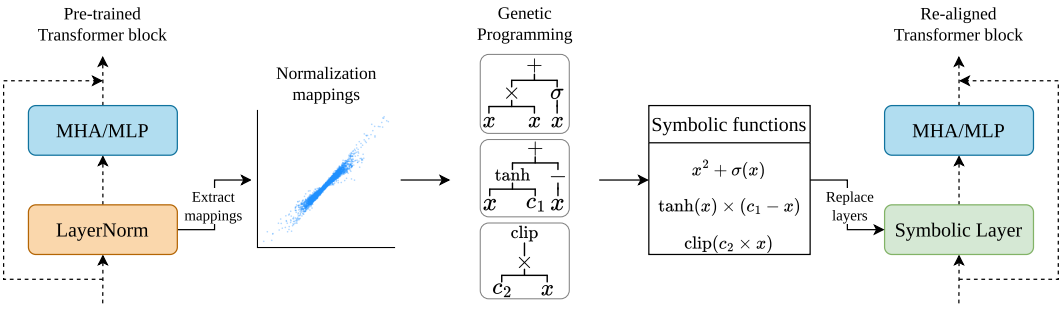

Figure 1: **Overview of the proposed symbolic discovery and re-alignment framework.** Normalization mappings are extracted from the LayerNorm operations within a pre-trained transformer block. Genetic programming uses these mappings to evolve layer-specific symbolic functions. We then replace the original LayerNorm operations with these discovered expressions, followed by a brief re-alignment phase to recover model performance.

2 METHODS

In this section, we describe our framework for replacing LayerNorm with evolved symbolic functions. We conduct our experiments using pre-trained ViT-B and ViT-L models (Dosovitskiy et al., 2020) as our baseline architectures, leveraging their established feature representations to drive both our symbolic discovery and model re-alignment. Our implementation uses the official weights pre-trained on ImageNet-1K (Deng et al., 2009), accessed via the `timm` library (Wightman, 2019). The anonymized source code is provided in the supplementary materials.

2.1 NORMALIZATION MAPPINGS EXTRACTION

To extract the LayerNorm mappings for our symbolic search, we first consider its standard formulation:

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \odot w + b, \tag{1}$$

where μ and σ^2 are the token-wise mean and variance, ϵ is a small constant for numerical stability, and w and b are the per-channel learnable affine weights and biases. While the arithmetic operations themselves are relatively lightweight, the dynamic calculation of μ and σ^2 introduces the complex cross-feature reduction dependency that bottlenecks edge execution. To ensure our symbolic search discovers a pure, element-wise replacement for this specific reduction operation, we isolate the mapping prior to the affine transformation. Following an approach similar to Zhu et al. (2025), we propagate a mini-batch of images through the pre-trained network. Given the captured layer output y, and the pre-trained weights w and biases b, we compute the target pre-affine mapping y_{pre} as:

$$y_{pre} = \frac{y - b}{w + \epsilon}. \tag{2}$$

Consistent with observations by Zhu et al. (2025), we find that normalization behaviour shifts significantly from non-linear mappings in earlier blocks to highly non-linear, S-shaped curves in deeper blocks (visualized across representative layers in Appendix A). To construct a representative dataset for symbolic regression, we randomly sample 50,000 data points across the feature and token dimensions per layer. This dataset is split into a 90/10 ratio for GP training and fitness validation, ensuring that the evolved solutions generalize well to unseen activations. A complete visualization of these extracted mappings for both architectures is provided in Appendix D.

2.2 HARDWARE-AWARE SYMBOLIC SEARCH

To discover optimal scalar replacements for each normalization layer, we utilize a symbolic discovery framework based on GP. We employ the `Kozax` (De Vries et al., 2025) library, a fast and flexible GP framework built in JAX, to conduct an independent evolutionary search for each normalization layer, run separately for ViT-B (25 layers) and ViT-L (49 layers). By fitting unique expressions to the

individual mappings captured in Section 2.1, our approach discovers specialized replacements that accommodate the varying functional behaviours observed across the network depths.

In our GP implementation, candidate solutions are represented as parse trees with a maximum initial depth of 4 and a strict complexity limit of 20 nodes to ensure hardware friendliness. The operator set $\mathcal{O}$ consists of addition ($+$), multiplication ($\times$), the hyperbolic tangent ($\tanh$), the sigmoid function (σ), negation (neg), and value clipping to ± 5.0 (clip). To optimize directly for hardware efficiency, we assign an estimated floating-point operations (FLOPs) cost to each operator, which serves as the complexity metric during the evolutionary search. The full derivation of these per-operator hardware costs is detailed in Appendix F. Finally, to improve search efficiency, constants within the trees are refined using gradient-based optimization for 10 steps per evaluation.

The primary objective of the search is to minimize the error between the evolved function $f(x)$ and the target mapping y_{pre}. However, our preliminary experiments revealed that optimizing for mean squared error (MSE) alone sometimes favoured near-linear functions that accurately captured the central distribution but exhibited limitless growth at the activation tails. When integrated into the ViT-B architecture, these unconstrained functions led to immediate activation explosion and gradient instability during fine-tuning. To address this, we developed a composite fitness function $\mathcal{F}$ designed to enforce numerical stability via a "pull-to-zero" penalty:

$$\mathcal{F} = \text{MSE}(y_{pre}, f(x)) + \gamma \cdot \tfrac{1}{2} \sum_{x \in \{-\delta, \delta\}} f(x)^2, \tag{3}$$

where $\gamma = 0.005$ is a weighting factor determined through empirical tuning to balance fitting accuracy with model stability. The anchor δ is dynamically set to $2 \cdot \max |x_{train}|$ for each layer, forcing the GP to favour expressions that revert toward zero at out-of-distribution extremes. This constraint effectively mimics the regularizing properties of standard LayerNorm, ensuring that the evolved scalar functions behave similarly during the model re-alignment.

For each search, the framework utilizes the NSGA-II algorithm (Deb et al., 2002) to perform multi-objective optimization, treating both the composite fitness $\mathcal{F}$ and functional complexity (estimated FLOPs) as competing objectives. We set a population size of 500 evolved over 50 generations. To account for the stochastic nature of GP, we perform five independent evolutionary searches for every normalization layer using varying random seeds. For each layer-seed combination, we maintain a complete Pareto front of solutions. To identify the optimal balance between functional fitness and computational cost, we apply the Kneedle algorithm (Satopaa et al., 2011) to locate the “knee” point of the Pareto front. This process yields five distinct sets of symbolic replacements for the normalization layers in the network, providing the foundation for the model re-alignment phase detailed in Section 2.3.

2.3 Model re-alignment

The final stage of our framework involves restoring the classification performance of the pre-trained models. We replace all LayerNorm layers with corresponding element-wise modules: either the symbolic solutions obtained with our GP framework (see Section 2.2) or DyT layers. To ensure minimal disruption to the network's established feature representations, these custom modules inherit the pre-trained affine weights (w) and biases (b) from the original layers. Preserving these parameters is essential for functional consistency, as our GP expressions operate as fixed approximations of the model's pre-affine mappings. For the DyT modules, we initialize the learnable α parameter at 0.5 to maintain consistency with Zhu et al. (2025).

We evaluate each method on the ImageNet-1K training set using three fine-tuning variants: -A (affine-only), -F (full fine-tuning), and -D (full fine-tuning with distillation). For the affine-only variants (GP-A and DyT-A), the backbone is frozen, optimizing only the learnable affine weights and biases (w, b). For the full fine-tuning variants (GP-F and DyT-F), the entire network is unfrozen to allow global feature re-alignment. Finally, the distillation variants (GP-D and DyT-D) also unfreeze the entire network, employing logit-based distillation (Hinton et al., 2015) where the student model minimizes the Kullback-Leibler (KL) divergence against its own pre-trained, LayerNorm-based teacher alongside standard cross-entropy loss. To provide a rigorous baseline, we also fine-tune the original architecture (LN) under identical conditions.

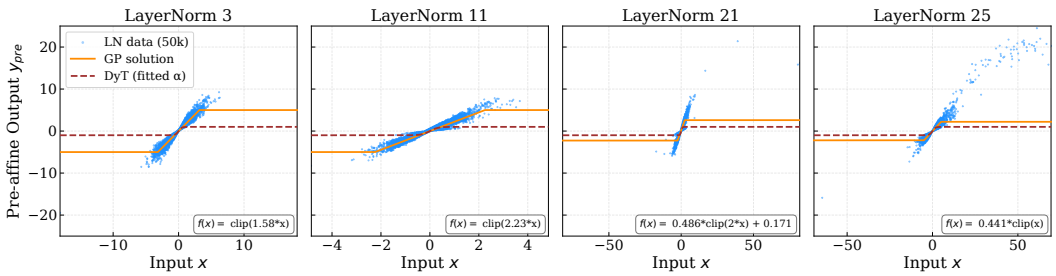

Figure 2: **Functional alignment of discovered symbolic expressions on ViT-B.** The evolved GP solutions (orange lines) and the optimized DyT baseline (dashed red lines, utilizing a least-squares optimized α) are overlaid onto the 50,000-point LayerNorm mappings (blue scatter). The GP framework successfully discovers symbolic functions that match both the shape and amplitude of the extracted LayerNorm mappings across all network depths. The explicitly discovered mathematical functions for each layer are displayed in the text boxes.

To ensure statistical robustness against the stochastic nature of both symbolic search and re-alignment, we report results averaged over five independent runs. GP variants utilize the five distinct sets of evolved functions (Section 2.2), while DyT and LN vary the training initialization seed. All models are trained for 20 epochs using AdamW without warmup, applying a cosine learning rate schedule for most variants; the complete per-variant, per-architecture schedule is detailed in Appendix C. We performed independent grid searches per variant to optimize the peak learning rate, weight decay, and stochastic depth. The general experimental configuration is detailed in Appendix B. Notably, we observed a significant disparity in training stability across both architectures: GP and LN trained stably using a single global learning rate, whereas the DyT-F and DyT-D variants were highly prone to divergence, necessitating differential learning rates across the backbone, affine, and α parameters. For the distillation variants, the total loss $\mathcal{L}$ is the weighted sum of the standard cross-entropy loss ($\mathcal{L}_{CE}$) and the KL-divergence ($\mathcal{L}_{KL}$):

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_{CE} + \lambda\tau^2\mathcal{L}_{KL}, \tag{4}$$

where the loss balancing coefficient $\lambda = 0.5$ and temperature $\tau = 4.0$ were empirically optimized in preliminary experiments.

3 Results

3.1 Symbolic Alignment

We first evaluate the capacity of our GP framework to capture the diverse functional behaviours of pre-trained LayerNorm layers. Figure 2 visualizes the discovered symbolic expressions overlaid on the target mappings for four representative layers across ViT-B's network depth. As observed in Section 2.1, the normalization behaviour shifts significantly from near-linear mappings in earlier blocks to highly non-linear, S-shaped curves in deeper blocks. Our framework successfully discovers heterogeneous expressions that adapt to these varying behaviours. For instance, in earlier layers, the

Table 1: **Functional alignment quality.** Comparison of GP-discovered expressions against a Dynamic Tanh (DyT) baseline across the 50,000-point LayerNorm mappings, using pre-trained ViT-B and ViT-L architectures. Metrics include mean squared error (MSE) and Coefficient of Determination (R^2). GP results represent the mean $\pm$ std across five independent seeds. The DyT baseline utilizes a per-layer least-squares optimized α parameter.

	ViT-B		ViT-L	
Method	**MSE (↓)**	R^2 (↑)	**MSE (↓)**	R^2 (↑)
Dynamic Tanh (Optimized α)	0.300	0.702	0.244	0.756
GP Symbolic Discovery (Ours)	$\mathbf{0.100 \pm 0.004}$	$\mathbf{0.900 \pm 0.004}$	$\mathbf{0.073 \pm 0.001}$	$\mathbf{0.927 \pm 0.001}$

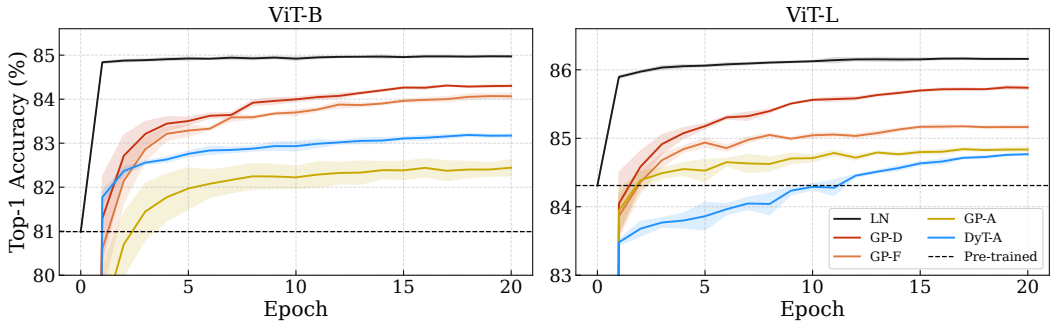

Figure 3: **ImageNet-1K validation performance recovery dynamics. (Left)** ViT-B fine-tuning trajectory comparing the LayerNorm (LN) baseline against GP-A, GP-F, GP-D, and DyT-A. **(Right)** The same comparison for ViT-L. Shaded areas represent ± 1 standard deviation across five independent seeds; black dashed lines denote each architecture's pre-trained baseline (80.99% for ViT-B, 84.31% for ViT-L). DyT-F and DyT-D variants are omitted for visual clarity.

discovered solutions are nearly linear, often using the `clip` operator to manage activation outliers and mimic the regularizing properties of LayerNorm. When tackling the pronounced S-curves of deeper blocks, our complexity-aware selection criteria actively penalizes expensive transcendental functions. As a result, the GP adapts by scaling and shifting those same `clip` operators, constructing highly efficient piecewise-linear approximations that strictly bound computational cost. A comprehensive visualization of the LayerNorm mappings and symbolic expressions for both architectures is provided in Appendix D.

To quantify the quality of this alignment, we report the mean squared error (MSE) and Coefficient of Determination (R^2) in Table 1, across all normalization layers for both architectures. We compare our results against a DyT baseline where the α parameter was independently optimized for every layer using least-squares regression. Our GP solutions achieve a substantially better alignment with the original LayerNorm mappings at both scales, explaining approximately 90% of the variance at ViT-B ($R^2 = 0.900 \pm 0.004$) and 92.7% at ViT-L ($R^2 = 0.927 \pm 0.001$), compared to only 70.2% and 75.6%, respectively, for the optimized DyT baseline. As shown in Figure 2, this significant gap in R^2 arises because the DyT function is strictly bounded to $[-1, 1]$. It is therefore fundamentally insufficient to capture the unconstrained amplitudes and varied normalization behaviours observed across either architecture. Furthermore, the low inter-seed standard deviation in our GP results demonstrates that the symbolic discovery process is robust and consistently identifies high-quality functional approximations regardless of the evolutionary initialization.

3.2 Performance Recovery

To evaluate the effectiveness of our symbolic replacements, we measure the performance recovery during a 20 epoch re-alignment phase, where the model is fine-tuned after replacing its normalization layers. This process reveals a significant disparity in how different functional approximations impact the pre-trained feature space. The initial replacement of LayerNorm with element-wise functions causes a distinct "shock" to the model's performance: at epoch 0, before any fine-tuning, all models with replaced LayerNorm layers show a significant drop in performance from the pre-trained baseline. However, the difference in this performance drop already shows a major advantage of our heterogeneous GP solutions compared to the homogeneous DyT approximations. Before any tuning has occurred, the GP variants already maintain a Top-1 accuracy of approximately 51.4% at ViT-B and 22.3% at ViT-L, suggesting that our discovered symbolic functions successfully preserve a meaningful portion of the pre-trained feature representations, though this retention is markedly weaker at the larger scale. In contrast, the homogeneous DyT approximations cause the model to break entirely, collapsing to chance level (0.10% and 0.11%, respectively) at initialization.

The remainder of the performance recovery process demonstrates both our heterogeneous layer-specific approach and our efficient post-training strategy. As illustrated in the zoomed view in Figure 3 (Right), most GP and DyT variants surpass their respective pre-trained baseline (80.99% for ViT-B, 84.31% for ViT-L) within the first epoch of fine-tuning, demonstrating that the model can typically

Table 2: **Main fine-tuning results on the ImageNet-1K validation set.** Classification performance of our evolved symbolic normalizations (GP) compared against standard LayerNorm and Dynamic Tanh (DyT) baselines, using pre-trained ViT-B and ViT-L architectures. All variants are fine-tuned for 20 epochs and evaluated across 5 independent runs.

	ViT-B		ViT-L	
Method	**Top-1 (%)**	**Top-5 (%)**	**Top-1 (%)**	**Top-5 (%)**
LN (Fine-tuning)	84.99 ± 0.02	97.43 ± 0.02	86.17 ± 0.01	97.91 ± 0.01
Affine-Only Fine-tuning				
DyT-A	$\mathbf{83.19 \pm 0.03}$	$\mathbf{96.74 \pm 0.02}$	84.77 ± 0.01	97.46 ± 0.02
GP-A (Ours)	82.48 ± 0.20	96.43 ± 0.06	$\mathbf{84.85 \pm 0.02}$	97.46 ± 0.02
Full Fine-tuning				
DyT-F	82.81 ± 0.02	96.60 ± 0.02	84.23 ± 0.05	97.19 ± 0.02
GP-F (Ours)	$\mathbf{84.07 \pm 0.06}$	$\mathbf{97.10 \pm 0.03}$	$\mathbf{85.19 \pm 0.02}$	$\mathbf{97.64 \pm 0.01}$
Knowledge Distillation				
DyT-D	83.36 ± 0.04	96.79 ± 0.05	84.85 ± 0.03	97.41 ± 0.03
GP-D (Ours)	$\mathbf{84.32 \pm 0.01}$	$\mathbf{97.15 \pm 0.01}$	$\mathbf{85.74 \pm 0.02}$	$\mathbf{97.77 \pm 0.02}$

be successfully re-aligned using only a fraction of the 300 epochs required by previous methods that train from scratch; the one exception is DyT-F at ViT-L, which never fully recovers to the pre-trained baseline across the full 20-epoch schedule. Specifically, the GP-D and GP-F variants show the most rapid and complete recovery, whereas the affine-only variants (GP-A and DyT-A) level off at lower accuracies. It is important to note that the full fine-tuning of the DyT variants (DyT-F and DyT-D) resulted in significantly slower convergence compared to our GP solutions. For visual clarity, these variants are omitted from Figure 3 but are analysed in full in Appendix E.

The peak quantitative results for this re-alignment are summarized in Table 2. Our best-performing variant, GP-D, reaches a peak Top-1 accuracy of 84.32% at ViT-B and 85.74% at ViT-L, narrowing the gap to the standard LayerNorm baseline to just 0.67% and 0.43%, respectively. Notably, the Top-5 gaps are smaller still (0.28% and 0.14%), suggesting that our symbolic replacements have a minimal impact on the model's overall ranking capabilities. The one setting where DyT leads is affine-only fine-tuning at ViT-B; in every other comparison across both architectures, our GP solutions match or exceed the fixed DyT baseline. Furthermore, these results validate our broader post-training strategy; both our DyT-A and DyT-D fine-tuning variants surpass the 82.5% accuracy reported in the original study (Zhu et al., 2025), which required training from scratch for over 300 epochs. This demonstrates that a short re-alignment phase is a viable and efficient alternative to training normalization-free architectures from initialization.

3.3 Hardware efficiency

Edge AI accelerators have limited on-chip memory, requiring model weights and intermediate tensors to be partitioned into tiles and streamed from off-chip memory (Chen et al., 2017; Lee et al., 2026). Because off-chip transactions dominate latency and energy on bandwidth-bound hardware (Bochem et al., 2025; Qiu et al., 2026), dataflows that keep tiles on-chip and avoid intermediate write-backs are strongly preferred. However, computational dependencies spanning tile boundaries, such as LayerNorm's global reduction, foreclose this streaming execution by forcing off-chip round-trips. While breaking this dependency typically degrades accuracy, the central question is whether a replacement can bypass the traditional efficiency-accuracy trade-off entirely, securing strict hardware savings without sacrificing performance.

To characterize each method along both axes, we instantiate this analysis for ViT-B and report the best-performing variants of each method (LN, DyT-D, and GP-D), denoted LN, DyT, and GP throughout this section. Because intra-method variants differ only in parameter values and not underlying architecture, selecting the highest accuracy point fairly reflects each method's reachable performance at identical hardware cost. On the efficiency side, we analytically quantify computational cost in floating-point operations (FLOPs) and memory traffic in read bytes per forward pass under FP32 storage. We adopt an analytical rather than runtime-based comparison because our GP functions

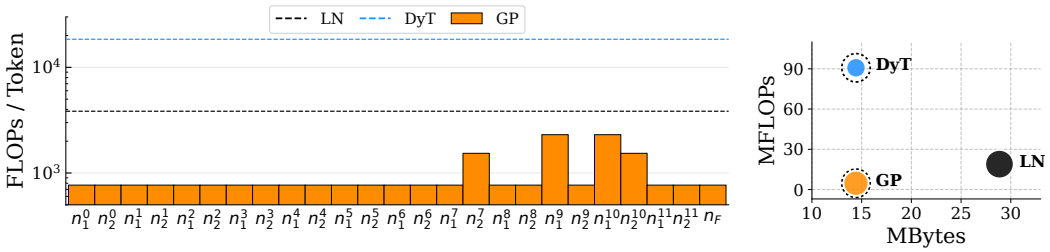

Figure 4: **Computational and memory complexity of normalization replacements in ViT-B. (Left)** Per-token FLOP counts for each of the 25 normalization layers, where n_j^i denotes the j-th normalization within block i and n_F denotes the final normalization layer. Our hardware-aware GP search explicitly restricts FLOP usage, producing layer-specific expressions that are uniformly cheaper than both the DyT and LN baselines. **(Right)** Aggregate complexity across all 25 normalization layers for a single forward pass (sequence length $\ell = 197$, feature dimension $d = 768$): total FLOPs (MFLOPs) versus total memory read access (MB). Bubble area is proportional to Top-1 accuracy; dashed circles around the GP and DyT markers show LN's bubble size.

lack the kernel-level optimization available for primitives like LayerNorm and $\tanh$; thus, runtime measurements would reflect implementation maturity rather than the intrinsic cost. FLOP counts for LayerNorm follow directly from its arithmetic definition. For DyT and our evolved expressions, transcendental functions ($\tanh(x)$ and $\sigma(x)$) are decomposed into exponential forms and evaluated via Horner's method (Higham, 2002). This approach is FLOP-optimal under the standard arithmetic model (Pan, 1966), setting the polynomial order to the minimum satisfying IEEE-754 FP32 unit-roundoff accuracy. We count each non-transcendental operation as a single FLOP, and evaluate read bytes only, as output write behaviour is structurally identical across methods.

Under this accounting, normalization requires $5d + 3$ FLOPs per token for standard LayerNorm and $24d$ for DyT, where d is the feature dimension. Both counts exclude the shared per-channel affine transform (γ, β). The corresponding cost of our evolved GP expressions varies layer by layer, with full derivations and per-layer FLOP and memory budgets are provided in Appendix F.

3.3.1 Analysis

Figure 4 (Left) reveals the structural payoff of layer-specific evolution. While our FLOP-aware selection tightly bounds the maximum compute, the per-layer cost still dynamically adapts to the heterogeneity of the underlying normalization mappings: near-linear mappings in early blocks are replaced by absolute minimal single-FLOP expressions, while deeper layers with S-shaped responses require slightly more operations. Concretely, all 25 layers fall below both the LN baseline of $5d + 3$ FLOPs per token and the DyT baseline, with the vast majority (21 layers) requiring only $1d$ FLOPs. The few marginally more complex layers (requiring $2d$ or $3d$ FLOPs) naturally appear in the deeper network blocks, reflecting the GP's ability to allocate a slightly higher complexity budget exactly where the underlying mapping demands it. A homogeneous replacement cannot exhibit this property by construction: it must commit to a single FLOP budget per layer and therefore cannot exploit this adaptive, layer-specific allocation that the heterogeneous mappings invite.

Figure 4 (Right) summarizes the aggregate hardware footprint across all 25 normalization layers in a single forward pass. Both GP and DyT roughly halve the memory access of LN by replacing its two-pass reduction with a single element-wise pass. Because each output element now depends only on the corresponding input element, normalization can be fused with the producing or consuming matmul, restoring the on-chip streaming dataflow that the LayerNorm reduction forecloses. Crucially, the GP framework secures this memory benefit without the arithmetic penalty associated with homogeneous replacements. While DyT incurs roughly a $4.8\times$ higher FLOP count than standard LN, our GP expressions demand approximately $4\times$ fewer total FLOPs than LN. This structural efficiency is paired with stronger performance recovery (GP: 84.32%, LN: 84.99%, DyT: 83.36%). The three methods therefore do not lie on a common Pareto curve: rather than shifting along an efficiency frontier, the GP framework circumvents the trade-off entirely, achieving a simultaneous reduction in both off-chip memory traffic and arithmetic complexity without a meaningful drop in model performance.

4 DISCUSSION

The global reduction bottleneck of LayerNorm remains a primary barrier to deploying ViTs on edge devices. While high-bandwidth GPUs can efficiently absorb the cost of this cross-feature data dependency, modern edge accelerators are primarily bottlenecked by memory bandwidth rather than arithmetic throughput (Kim et al., 2024; Moon et al., 2025). We address this bottleneck by utilizing genetic programming to evolve heterogeneous, layer-specific scalar functions directly from pre-trained network activations. By breaking the cross-feature dependency that ties LayerNorm to its global reduction, our approach enables optimizations such as token-wise tiling and layer fusion, leading to a $2\times$ reduction in off-chip memory traffic alongside a strict reduction in arithmetic complexity, all without the catastrophic performance collapse seen in homogeneous scalar replacements.

Compared to homogeneous scalar replacements (Zhu et al., 2025), our evolved expressions achieve superior functional alignment across all network depths (see Table 1). This alignment allows both architectures to retain 22–51% Top-1 accuracy immediately upon replacement, whereas homogeneous functions collapse to chance level. By evolving functions that follow the varying behaviours across the network, moving from linear to S-shaped mappings, we preserve the established feature representations. This preservation provides a strong foundation for the 20 epoch fine-tuning period, explaining why our full fine-tuning (GP-F) and distillation variants (GP-D) achieve higher final accuracies than their DyT counterparts at both scales, even at ViT-L, where initial retention is markedly lower. At ViT-B, DyT-A slightly outperforms our affine-only variant when the backbone is frozen, suggesting that DyT uses its optimizable scaling parameter (α) to compensate for its poorer functional fit. However, when the rest of the model is allowed to adapt, our symbolic expressions are more effective because they more closely preserve the original LayerNorm behaviour. This allows the network to easily adjust its weights to the new scalar operations.

Because our symbolic expressions retain near-baseline accuracy, we can fully realize the hardware benefits of breaking the global reduction bottleneck. Unlike prior element-wise substitutions that trade an increase in on-chip compute for reduced data movement, our layer-specific expressions strictly reduce both metrics by construction. Specifically, our GP solutions maintain lower arithmetic complexity than both standard LayerNorm and the DyT baseline, while reaching a significantly higher accuracy ceiling than DyT. Rather than trading arithmetic cost for memory traffic, our methodology reduces both: maintaining DyT's memory advantages while strictly reducing computational cost. Crucially, the heterogeneity of these expressions does not require unique hardware circuits for every layer. A single reconfigurable activation unit can evaluate the necessary symbolic primitives on a shared datapath, allowing simpler expressions to bypass expensive transcendental pipelines for lower latency and energy. The resulting overhead for storing these layer-specific operation sequences and coefficients is negligible relative to the memory and arithmetic budgets they control.

Alongside these results, we note several limitations and avenues for future work. Our GP framework minimizes MSE on just 50,000 activation points. While computationally fast, this small dataset may limit how well solutions generalize to the true distribution of LayerNorm behaviours. Scaling the data could improve functional alignment and reduce performance loss during replacement. Furthermore, future work could evaluate model accuracy directly within the GP evolutionary loop. Despite heavy computational costs, this directly optimizes the true objective and captures cross-layer interactions. ViT-L illustrates this gap: its stronger per-layer alignment does not yield higher retention.

Additionally, our validation spans two model scales but a single architecture family (ViT) and dataset (ImageNet-1K). Zhu et al. (2025) demonstrated that homogeneous scalar replacements are viable across various transformer architectures. However, whether our heterogeneous GP framework similarly generalizes to other vision architectures or NLP domains remains an open question. Beyond generalization, our layer-specific approach presents an opportunity for hardware optimization. Because we evolve a unique equation for every layer, the deployed model must store 25 instruction sets at ViT-B and 49 at ViT-L. Future work could explore structural sharing, where layers exhibiting similar normalization mappings are constrained to reuse the same symbolic sub-trees. This would drastically reduce the instruction memory required on the target edge device.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods, and to clean and reformat experimental data. The methods themselves, including the symbolic search formulation, the composite fitness function, and the post-training re-alignment procedure, were conceived by the authors; generative AI assistance was limited to their implementation in code. We have not used generative AI tools to develop theoretical models or conceptual frameworks, to formulate mathematical claims or assist in their derivation, to formulate or refine hypotheses, to design or provide feedback on research methodology or experiments, or to interpret results, and the generation of synthetic data sets and translation are not applicable to this work.

Additionally, we used generative AI tools to create and edit software code, to create and modify figures, to draft and edit parts of this paper for readability and formatting, to summarize existing literature, and to format references.

We have reviewed all AI-assisted work. All AI-implemented code was reviewed and tested by the authors against the intended method specification, all reported numerical results were verified against the original experiment logs, and all AI-assisted text was reviewed and revised by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

By successfully eliminating the memory bottlenecks that restrict ViT deployment, our framework allows for the execution of highly capable vision models directly on local, resource-constrained edge devices. While this local processing improves latency, reduces cloud reliance, and lowers the energy footprint of AI systems, it inherently introduces dual-use risks. Specifically, unconstrained edge vision could facilitate pervasive surveillance systems, raising privacy concerns, or be adapted for autonomous military applications. We encourage the research community to prioritize transparent, ethical frameworks when deploying hardware-aware vision models in real-world environments.

REPRODUCIBILITY STATEMENT

We have taken several steps to ensure that our results can be reproduced. The complete framework, covering normalization mapping extraction, the genetic programming search, and the model re-alignment procedure, is described in Section 2, with the general experimental configuration given in Appendix B and the per-variant, per-architecture hyperparameters in Appendix C. Appendix D lists the exact symbolic expressions discovered for every normalization layer of both architectures, allowing the modified models to be constructed directly without repeating the evolutionary search. All reported results are averaged over five independent runs, with the resulting variability shown as error shading in Figure 3 and as standard deviations in Table 1 and Table 2. Finally, we provide an anonymized archive of all code and scripts required to reproduce our main experimental results, together with the instructions and settings used, as supplementary material.

REFERENCES

Severin Bochem, Victor J.B. Jung, Arpan Suravi Prasad, Francesco Conti, and Luca Benini. Distributed inference with minimal off-chip traffic for transformers on low-power MCUs. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pp. 1–7, 2025. doi: 10.23919/DATE64628.2025.10992712.

Jinfeng Cao, Bo Peng, Mingzhong Gao, Haichun Hao, Xinfang Li, and Hongwei Mou. Object detection based on cnn and vision-transformer: A survey. *IET Computer Vision*, 19(1):e70028, 2025.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pp. 213–229. Springer, 2020.

Chunyun Chen, Lantian Li, and Mohamed M Sabry Aly. Vita: A highly efficient dataflow and architecture for vision transformers. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6. IEEE, 2024.

Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017. doi: 10.1109/JSSC.2016.2616357.

Ziyang Chen, Ming Hao, Xinye Cao, Jingwei Zhang, Chaoyao Shen, Guoqing Li, and Meng Zhang. Hardware-friendly and efficient vision transformer for deployment on low-power embedded device. *Journal of Low Power Electronics and Applications*, 16(1):1, 2025.

Sigur De Vries, Sander Wessel Keemink, and Marcel Antonius Johannes van Gerven. Kozax: flexible and scalable genetic programming in jax. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 603–606, 2025.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

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

Peiyan Dong, Jinming Zhuang, Zhuoping Yang, Shixin Ji, Yanyu Li, Dongkuan Xu, Heng Huang, Jingtong Hu, Alex K Jones, Yiyu Shi, et al. Eq-vit: Algorithm-hardware co-design for end-to-end acceleration of real-time vision transformer inference on versal acap architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(11):3949–3960, 2024.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, 2nd edition, 2002. ISBN 0898715210.

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

Raehyeong Kim, Dayoung Lee, Jinyeol Kim, Joungmin Park, and Seung Eun Lee. Hardware accelerator for approximation-based softmax and layer normalization in transformers. *Electronics*, 14(12):2337, 2025.

Sangyeob Kim, Sangjin Kim, Wooyoung Jo, Soyeon Kim, Seongyon Hong, and Hoi-Jun Yoo. C-transformer: A 2.6-18.1μJ/Token homogeneous DNN-transformer/spiking-transformer processor with big-little network and implicit weight generation for large language models. In *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 67, pp. 368–370, 2024. doi: 10.1109/ISSCC49657.2024.10454330.

John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, 1994.

Jemin Lee, Yongin Kwon, Sihyeong Park, Misun Yu, Jeman Park, and Hwanjun Song. Q-hyvit: Post-training quantization of hybrid vision transformers with bridge block reconstruction for iot systems. *IEEE Internet of Things Journal*, 11(22):36384–36396, 2024.

Kyungmi Lee, Gaurab Das, Donghyeon Han, and Anantha P. Chandrakasan. Securing DNN acceleration from off-chip memory vulnerabilities with low-overhead authenticated encryption. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 34(3):953–966, 2026. doi: 10.1109/TVLSI.2025.3650411.

Xinyu Liu, Houwen Peng, Ningxin Zheng, Yuqing Yang, Han Hu, and Yixuan Yuan. Efficientvit: Memory efficient vision transformer with cascaded group attention. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14420–14430. Ieee, 2023.

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.

Kyle Marino, Pengmiao Zhang, and Viktor K Prasanna. Me-vit: A single-load memory-efficient fpga accelerator for vision transformers. In *2023 IEEE 30th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pp. 213–223. IEEE, 2023.

Seunghyun Moon, Mao Li, Gregory K. Chen, Phil C. Knag, Ram Kumar Krishnamurthy, and Mingoo Seok. T-REX: A 68-to-567μs/Token 0.41-to-3.95μJ/Token transformer accelerator with reduced external memory access and enhanced hardware utilization in 16nm FinFET. In *2025 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 68, pp. 406–408, 2025. doi: 10.1109/ISSCC49661.2025.10904793.

Jean-Michel Muller, Nicolas Brunie, Florent De Dinechin, Claude-Pierre Jeannerod, Mioara Joldes, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, and Serge Torres. *Handbook of Floating-Point Arithmetic*, volume 1. Springer, 2018.

V Ya Pan. Methods of computing values of polynomials. *Russian Mathematical Surveys*, 21(1): 105–136, 1966.

Yikan Qiu, Guoxiang Li, Meng Wu, Yifan Jia, Le Ye, and Yufei Ma. Quartet: A digital compute-in-memory versatile AI accelerator with heterogeneous tensor engines and off-chip-less dataflow. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 73(1):370–383, 2026. doi: 10.1109/TCSI.2025.3598287.

Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? *Advances in Neural Information Processing Systems*, 34:12116–12128, 2021.

Mohammad Erfan Sadeghi, Arash Fayyazi, Seyedarmin Azizi, and Massoud Pedram. Peano-vit: Power-efficient approximations of non-linearities in vision transformers. In *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 1–6, 2024.

Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a" kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pp. 166–171. IEEE, 2011.

Tianyi Sun, Tuo Ma, Jiali Liu, Zhiwei Li, Qingjiang Li, Yinan Wang, Haijun Liu, and Sen Liu. Integer quantization of nonlinear operations towards hardware-friendly vits. In *2025 32nd IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 1–4. IEEE, 2025.

Hamid Tabani, Ajay Balasubramaniam, Shabbir Marzban, Elahe Arani, and Bahram Zonooz. Improving the efficiency of transformers for resource-constrained devices. In *2021 24th Euromicro Conference on Digital System Design (DSD)*, pp. 449–456. IEEE, 2021.

Ping-Tak Peter Tang. Table-driven implementation of the exponential function in IEEE floating-point arithmetic. *ACM Trans. Math. Softw.*, 15(2):144–157, June 1989. ISSN 0098-3500. doi: 10.1145/63522.214389. URL https://doi.org/10.1145/63522.214389.

Hans Thisanke, Chamli Deshan, Kavindu Chamith, Sachith Seneviratne, Rajith Vidanaarachchi, and Damayanthi Herath. Semantic segmentation using vision transformers: A survey. *Engineering Applications of Artificial Intelligence*, 126:106669, 2023.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pp. 10347–10357. PMLR, 2021.

Wenxun Wang, Shuchang Zhou, Wenyu Sun, Peiqin Sun, and Yongpan Liu. Sole: Hardware-software co-design of softmax and layernorm for efficient transformer inference. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2023.

Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

Can Xiao, Jianyi Cheng, and Yiren Zhao. Refining datapath for microscaling vits. In *2025 35th International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 263–272. IEEE, 2025.

Bin Xu, Ayan Banerjee, and Sandeep Gupta. Hardware acceleration for neural networks: A comprehensive survey. *arXiv preprint arXiv:2512.23914*, 2025.

Joonsang Yu, Junki Park, Seongmin Park, Minsoo Kim, Sihwa Lee, Dong Hyun Lee, and Jungwook Choi. Nn-lut: Neural approximation of non-linear operations for efficient transformer inference. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 577–582, 2022.

Zhixiong Zhao, Haomin Li, Fangxin Liu, Yuncheng Lu, Zongwu Wang, Tao Yang, Li Jiang, and Haibing Guan. Quark: Quantization-enabled circuit sharing for transformer acceleration by exploiting common patterns in nonlinear operations. In *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2025.

Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without normalization. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 14901–14911, 2025.

A Normalization mappings

Figure 5 visualizes the underlying distribution of the pre-affine normalization mappings across four representative layers. Generated using 5 million sampled data points, these plots clearly illustrate the network's transition from near-linear behaviour in early blocks to highly non-linear, S-shaped curves in deeper blocks.

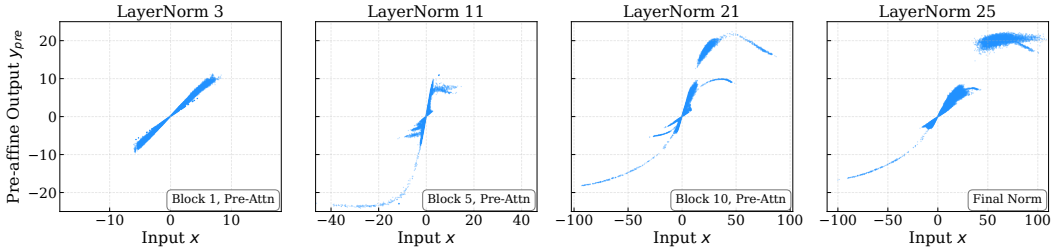

Figure 5: **Evolution of LayerNorm mappings across network depth in a pre-trained ViT-B.** Scatter plots visualize the pre-affine output y_{pre} against the input x across four representative layers. Early normalization blocks (e.g., LayerNorm 3 and 11) exhibit largely linear transformations, whereas deeper blocks (e.g., LayerNorm 21 and 25) display highly non-linear, S-shaped characteristics. This distinct structural variation across the network motivates our layer-specific symbolic discovery approach to accurately capture the diverse behaviours across the network.

B Experimental settings

All re-alignment experiments are conducted on the ImageNet-1K dataset using pre-trained ViT-B/16 (`vit_base_patch16_224`) and ViT-L/16 (`vit_large_patch16_224`) architectures. The following settings were held constant across all variants to ensure a fair comparison:

- **Hardware and precision:** ViT-B experiments were performed on a single NVIDIA A100 (80GB) GPU. ViT-L experiments were trained on 2 GPUs via data-parallel training, using either NVIDIA A100 (80GB) or H100 (80GB) GPUs. We utilized Automatic Mixed Precision (AMP) throughout to optimize memory usage and training speed.
- **Optimization:** We used the AdamW optimizer with default momentum parameters $(\beta_1, \beta_2) = (0.9, 0.999)$. The global batch size was set to 512. Each variant was fine-tuned for exactly 20 epochs without a warmup phase, using the learning rate schedule listed for that variant in Table 3 (ViT-B) or Table 4 (ViT-L).
- **Data augmentation:** We adopted the standard augmentation strategy used in Touvron et al. (2021). Specifically, we applied RandAugment (`rand-m9-mstd0.5-inc1`), Color Jitter (0.4), and Random Erasing (0.25). Consistent with a short fine-tuning regime, we did not employ Mixup, Cutmix, or Label Smoothing.
- **Evaluation:** Validation performance was measured using a Center Crop with a crop ratio of 0.875.
- **Distillation configuration:** For all distillation variants (GP-D and DyT-D), we utilized logit-based distillation with a temperature $\tau = 4.0$ and a loss balancing coefficient $\lambda = 0.5$.
- **Compute time and total resources:** Depending on the specific architectural variant and cluster node allocation, a standard 20-epoch fine-tuning run takes approximately 4–9 hours for ViT-B and 6–16 hours for ViT-L. Accounting for the evaluation of all variants, hyperparameter sweeps, preliminary trials, and final logging across both architectures, we estimate the total compute required for this research project to be approximately 3,150 GPU hours.

C HYPERPARAMETERS

To ensure a fair comparison, we performed an independent grid search for each variant to identify the optimal learning rate, weight decay, and stochastic depth rate. The final configurations for ViT-B and ViT-L are summarized in Table 3 and Table 4, respectively. As described in Section 2.3, the DyT-F and DyT-D variants were highly sensitive to hyperparameters at both scales. For these variants, independent learning rates were applied to the backbone, affine parameters, and the learnable α scalars. In contrast, the LN and GP variants were successfully optimized using a single global learning rate for all trainable parameters.

Table 3: **Optimal hyperparameters for ImageNet-1K re-alignment (ViT-B).** Parameters were determined via independent grid searches for each variant to ensure a fair performance comparison. For variants where the backbone is frozen (-A), the backbone learning rate is denoted with a hyphen (-). The LR_{affine} row corresponds to the learning rate for the affine transformation parameters (weights and biases), which were inherited from the original pre-trained normalization layers. For DyT variants, LR_α denotes the specific learning rate for the learnable scaling parameter.

Parameter	LN	GP-A	DyT-A	GP-F	DyT-F	GP-D	DyT-D
LR_{backbone}	1×10^{-6}	-	-	1×10^{-5}	2×10^{-5}	1×10^{-5}	3×10^{-5}
LR_{affine}	1×10^{-6}	1×10^{-3}	8×10^{-3}	1×10^{-5}	1×10^{-4}	1×10^{-5}	1×10^{-4}
LR_α	-	-	8×10^{-3}	-	5×10^{-5}	-	5×10^{-5}
LR Schedule	Cosine	Constant	Cosine	Cosine	Cosine	Cosine	Cosine
Weight Decay	0.05	0.0	0.0	0.0	0.0	0.05	0.0
Stoch. Depth	0.2	0.0	0.0	0.0	0.0	0.2	0.0

D COMPLETE LAYERNORM MAPPINGS AND DISCOVERED SYMBOLIC SOLUTIONS

This section provides a comprehensive overview of the layer-specific symbolic solutions discovered by our GP framework, for both architectures evaluated in this work.

Table 4: **Optimal hyperparameters for ImageNet-1K re-alignment (ViT-L).** Parameter definitions and notation follow Table 3.

Parameter	**LN**	**GP-A**	**DyT-A**	**GP-F**	**DyT-F**	**GP-D**	**DyT-D**
LR_{backbone}	1×10^{-6}	-	-	1×10^{-5}	2×10^{-5}	1×10^{-5}	3×10^{-5}
LR_{affine}	1×10^{-6}	2×10^{-3}	8×10^{-3}	1×10^{-5}	1×10^{-4}	1×10^{-5}	1×10^{-4}
LR_{α}	-	-	8×10^{-3}	-	5×10^{-5}	-	5×10^{-5}
LR Schedule	Cosine	Cosine	Cosine	Cosine	Cosine	Cosine	Cosine
Weight Decay	0.05	0.0	0.0	0.05	0.05	0.05	0.05
Stoch. Depth	0.2	0.0	0.0	0.2	0.2	0.2	0.2

D.1 VIT-B

Figure 6 visualizes the complete set of 25 normalization mappings extracted from the pre-trained ViT-B architecture, overlaid with the functional approximations discovered by the GP search. Correspondingly, Table 5 lists the exact mathematical formulations of these symbolic expressions. These solutions are derived from a single representative evolutionary search seed, where the specific expression for each layer was selected using the Kneedle algorithm to locate the optimal balance between functional fitness and computational cost on that layer's respective Pareto front.

Table 5: **Optimal GP-evolved symbolic expressions per layer.** For each LayerNorm layer in the pre-trained ViT-B/16 architecture, we show the discovered symbolic solutions obtained from a single representative evolutionary seed (out of five independent runs). Solutions were selected by applying the Kneedle algorithm to locate the optimal balance of fitness and hardware cost on the Pareto front. **Complexity** represents the relative rank of the expression's structural depth and number of operations within that layer's search space. **Train/Val Fitness** denotes the score achieved via the GP fitness function. **FLOPs** reports the per-token cost at hidden dimension d under the unified $\exp$ primitive (Appendix F), where each $\tanh(x)$ counts as 23 FLOPs, each $\sigma(x)$ as 22, and each `clip` as zero (reducing to a comparison and a select instead of floating operation).

#	Layer	Train Fitness (↓)	Val Fitness (↓)	Expression	FLOPs
1	`blocks.0.norm1`	0.2445	0.2362	$\text{clip}(1.54x)$	$1d$
2	`blocks.0.norm2`	0.3327	0.2732	$\text{clip}(1.62x)$	$1d$
3	`blocks.1.norm1`	0.1808	0.1739	$\text{clip}(1.58x)$	$1d$
4	`blocks.1.norm2`	0.1941	0.1912	$\text{clip}(2.08x)$	$1d$
5	`blocks.2.norm1`	0.2046	0.2029	$\text{clip}(2.1x)$	$1d$
6	`blocks.2.norm2`	0.2095	0.2020	$\text{clip}(2x)$	$1d$
7	`blocks.3.norm1`	0.2277	0.2248	$\text{clip}(2.17x)$	$1d$
8	`blocks.3.norm2`	0.2094	0.2093	$\text{clip}(2.09x)$	$1d$
9	`blocks.4.norm1`	0.2109	0.2040	$\text{clip}(2.17x)$	$1d$
10	`blocks.4.norm2`	0.2330	0.2190	$\text{clip}(2.15x)$	$1d$
11	`blocks.5.norm1`	0.2125	0.2165	$\text{clip}(2.23x)$	$1d$
12	`blocks.5.norm2`	0.2050	0.2659	$\text{clip}(2.06x)$	$1d$
13	`blocks.6.norm1`	0.2021	0.2071	$\text{clip}(2.07x)$	$1d$
14	`blocks.6.norm2`	0.2108	0.2012	$\text{clip}(1.9x)$	$1d$
15	`blocks.7.norm1`	0.2177	0.2132	$\text{clip}(1.84x)$	$1d$
16	`blocks.7.norm2`	0.1530	0.1376	$0.545 \cdot \text{clip}(2.98x)$	$2d$
17	`blocks.8.norm1`	0.1982	0.2306	$\text{clip}(1.54x)$	$1d$
18	`blocks.8.norm2`	0.1989	0.2071	$\text{clip}(-1.38 \cdot (-x))$	$1d$
19	`blocks.9.norm1`	0.0788	0.1030	$0.525 \cdot \text{clip}(2.42x + 0.409)$	$3d$
20	`blocks.9.norm2`	0.1751	0.1673	$\text{clip}(x) + 0.161$	$1d$
21	`blocks.10.norm1`	0.0920	0.1073	$0.486 \cdot \text{clip}(2x) + 0.171$	$3d$
22	`blocks.10.norm2`	0.0798	0.1538	$-0.0237x + \text{clip}(x)$	$2d$
23	`blocks.11.norm1`	0.1443	0.1356	$0.542 \cdot \text{clip}(x)$	$1d$
24	`blocks.11.norm2`	0.2714	0.2260	$0.455 \cdot \text{clip}(x)$	$1d$
25	`norm`	0.3560	0.6113	$0.441 \cdot \text{clip}(x)$	$1d$
	Total				**31d**

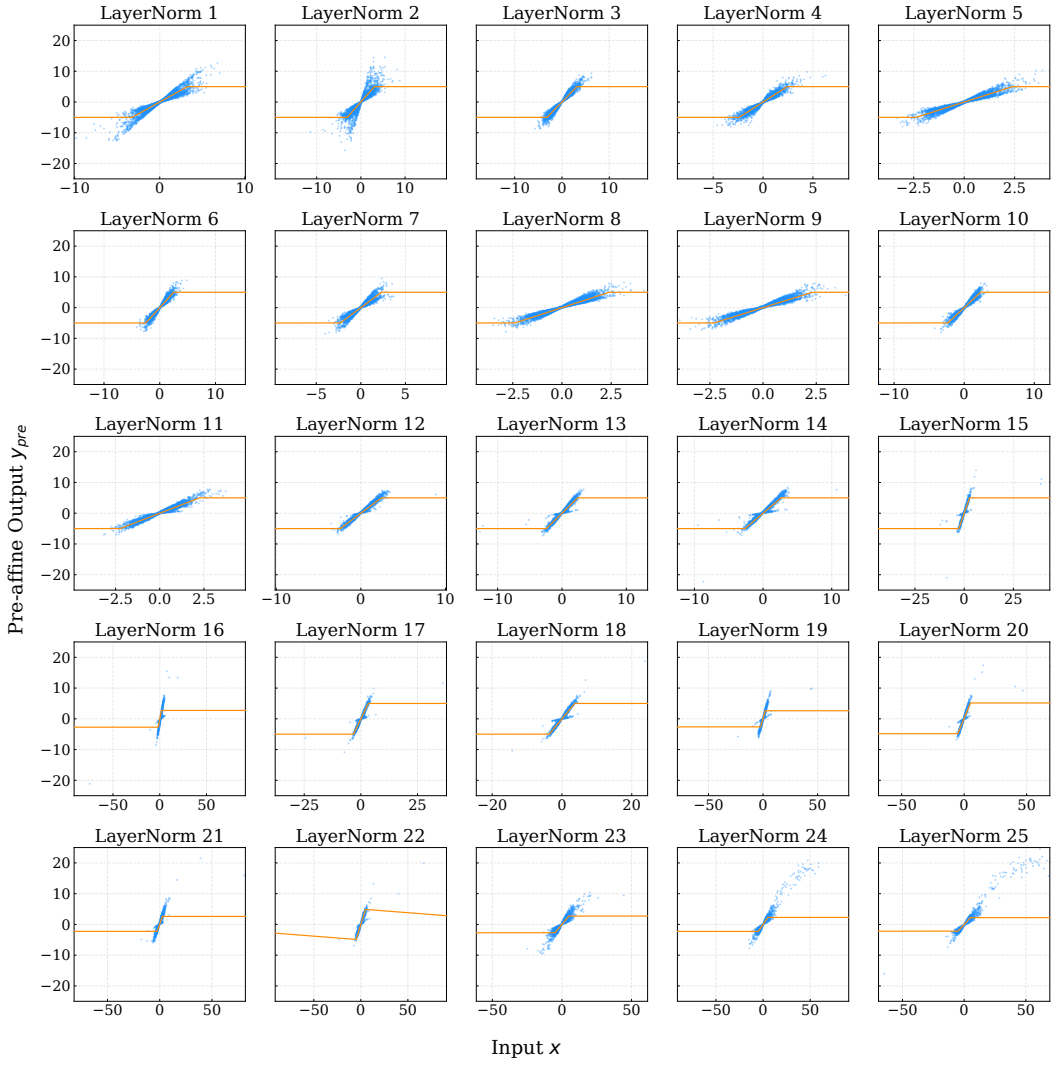

Figure 6: **Comprehensive functional alignment across all 25 normalization layers.** The evolved GP solutions (orange lines) are overlaid onto the target 50,000-point LayerNorm mappings (blue scatter) extracted from the pre-trained ViT-B architecture. The visualization demonstrates the framework's ability to seamlessly adapt to the structural transition from near-linear behaviour in early layers to highly non-linear, S-shaped characteristics in deeper blocks.

D.2 ViT-L

Figure 7 provides the analogous visualization for the 49 normalization layers of ViT-L, mirroring the ViT-B figure above, with the corresponding symbolic expressions listed in Table 6.

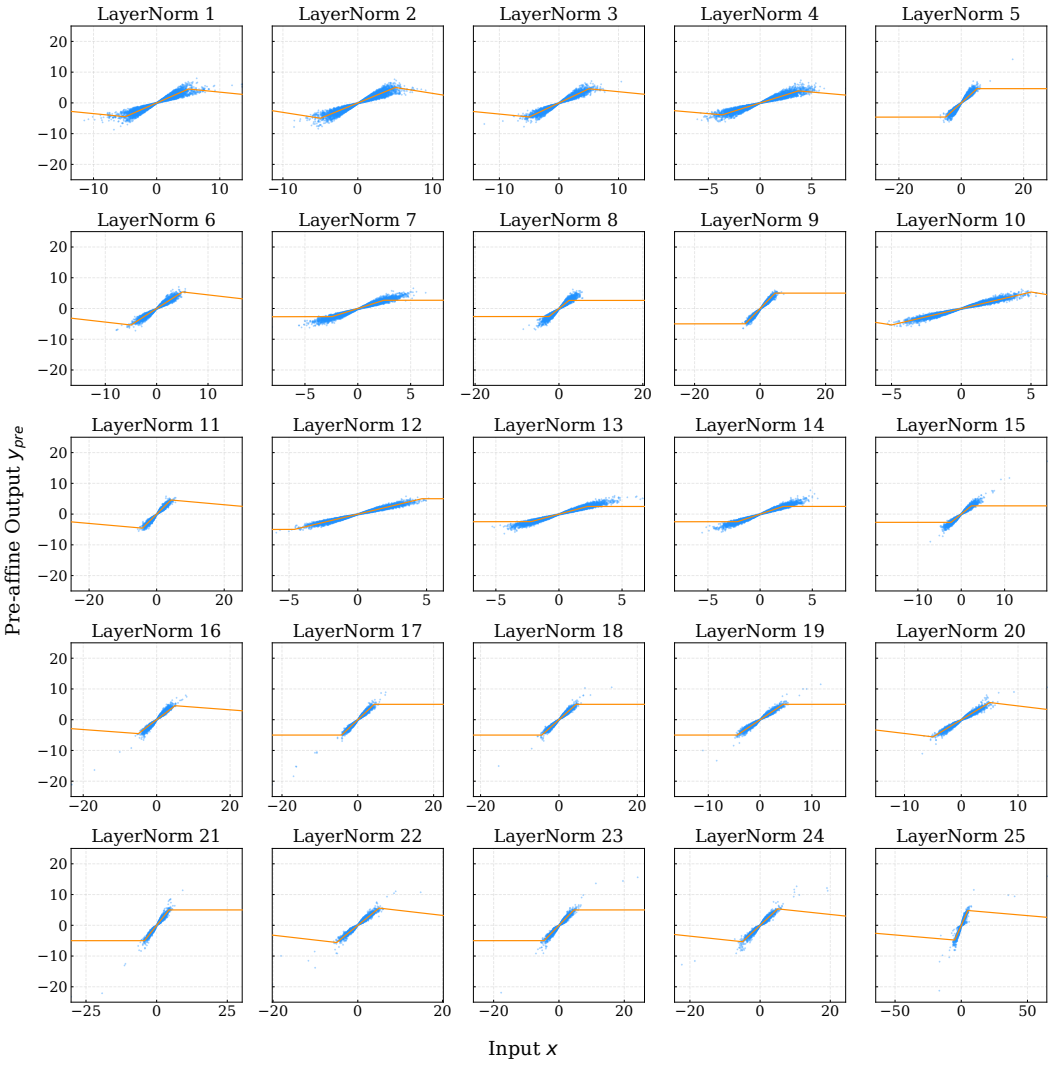

Figure 7: **Comprehensive functional alignment across all 49 normalization layers of ViT-L (1/2).** The evolved GP solutions (orange lines) are overlaid onto the target 50,000-point LayerNorm mappings (blue scatter) extracted from the pre-trained ViT-L architecture, mirroring Figure 6.

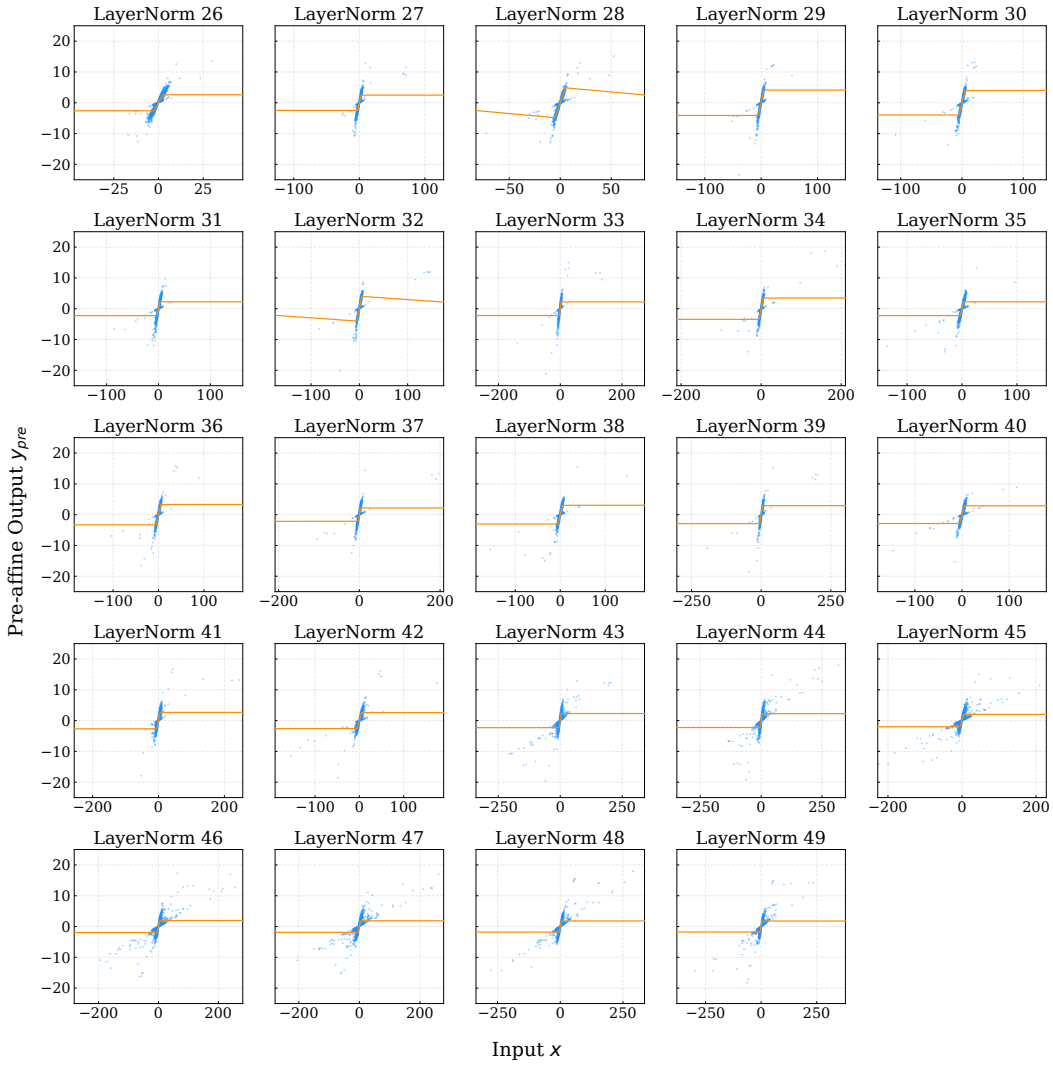

Figure 7: **(continued)** Remaining normalization layers of ViT-L.

Table 6: **Optimal GP-evolved symbolic expressions per layer (ViT-L).** Definitions and notation follow Table 5.

#	Layer	Train Fitness (↓)	Val Fitness (↓)	Expression	FLOPs
1	`blocks.0.norm1`	0.0696	0.0695	$-0.205x + 1.11 \cdot \text{clip}(x)$	$3d$
2	`blocks.0.norm2`	0.0657	0.0862	$-0.386x + 1.39 \cdot \text{clip}(x)$	$3d$
3	`blocks.1.norm1`	0.0448	0.0490	$-0.191x + 1.11 \cdot \text{clip}(x)$	$3d$
4	`blocks.1.norm2`	0.0513	0.0735	$-0.297x + \text{clip}(1.36x)$	$3d$
5	`blocks.2.norm1`	0.1659	0.1598	$0.926 \cdot \text{clip}(x)$	$1d$
6	`blocks.2.norm2`	0.0332	0.1022	$-0.192x + 1.27 \cdot \text{clip}(x)$	$3d$
7	`blocks.3.norm1`	0.0773	0.0771	$0.536 \cdot \text{clip}(2.04x)$	$2d$
8	`blocks.3.norm2`	0.0837	0.0852	$0.525 \cdot \text{clip}(2.12x)$	$2d$
9	`blocks.4.norm1`	0.1533	0.1485	$\text{clip}(-1.1 \cdot (-x))$	$1d$
10	`blocks.4.norm2`	0.0233	0.0225	$-0.737x + 1.81 \cdot \text{clip}(x)$	$3d$
11	`blocks.5.norm1`	0.0305	0.1004	$-0.0979x + \text{clip}(1.19x)$	$3d$
12	`blocks.5.norm2`	0.1472	0.1468	$\text{clip}(1.08x)$	$1d$
13	`blocks.6.norm1`	0.0673	0.0693	$0.496 \cdot \text{clip}(2.24x)$	$2d$
14	`blocks.6.norm2`	0.0692	0.0639	$0.499 \cdot \text{clip}(2.29x)$	$2d$
15	`blocks.7.norm1`	0.0778	0.1092	$0.536 \cdot \text{clip}(2.13x)$	$2d$
16	`blocks.7.norm2`	0.0815	0.1414	$-0.0896x + \text{clip}(x)$	$2d$
17	`blocks.8.norm1`	0.1644	0.1483	$\text{clip}(1.17x)$	$1d$
18	`blocks.8.norm2`	0.1546	0.1666	$\text{clip}(1.14x)$	$1d$
19	`blocks.9.norm1`	0.1505	0.1583	$\text{clip}(1.13x)$	$1d$
20	`blocks.9.norm2`	0.0241	0.0969	$-0.222x + 1.34 \cdot \text{clip}(x)$	$3d$
21	`blocks.10.norm1`	0.1524	0.2704	$\text{clip}(1.16x)$	$1d$
22	`blocks.10.norm2`	0.0346	0.0950	$-0.159x + 1.28 \cdot \text{clip}(x)$	$3d$
23	`blocks.11.norm1`	0.1655	0.1526	$\text{clip}(1.1x)$	$1d$
24	`blocks.11.norm2`	0.0329	0.0663	$-0.121x + 1.19 \cdot \text{clip}(x)$	$3d$
25	`blocks.12.norm1`	0.0538	0.0774	$-0.0371x + \text{clip}(x)$	$2d$
26	`blocks.12.norm2`	0.0976	0.0928	$0.519 \cdot \text{clip}(1.96x)$	$2d$
27	`blocks.13.norm1`	0.1059	0.1158	$0.496 \cdot \text{clip}(2.01x)$	$2d$
28	`blocks.13.norm2`	0.0668	0.1072	$-0.03x + \text{clip}(x)$	$2d$
29	`blocks.14.norm1`	0.1728	0.1763	$0.824 \cdot \text{clip}(x)$	$1d$
30	`blocks.14.norm2`	0.1616	0.1558	$0.796 \cdot \text{clip}(x)$	$1d$
31	`blocks.15.norm1`	0.1105	0.1291	$0.45 \cdot \text{clip}(1.93x)$	$2d$
32	`blocks.15.norm2`	0.0790	0.0914	$-0.011x + 0.813 \cdot \text{clip}(x)$	$3d$
33	`blocks.16.norm1`	0.1132	0.1896	$0.442 \cdot \text{clip}(1.78x)$	$2d$
34	`blocks.16.norm2`	0.1389	0.1339	$0.692 \cdot \text{clip}(x)$	$1d$
35	`blocks.17.norm1`	0.1028	0.1153	$0.448 \cdot \text{clip}(1.62x)$	$2d$
36	`blocks.17.norm2`	0.1279	0.1390	$0.66 \cdot \text{clip}(x)$	$1d$
37	`blocks.18.norm1`	0.1008	0.1057	$0.435 \cdot \text{clip}(1.57x)$	$2d$
38	`blocks.18.norm2`	0.1125	0.1867	$0.608 \cdot \text{clip}(x)$	$1d$
39	`blocks.19.norm1`	0.1297	0.1083	$0.584 \cdot \text{clip}(x)$	$1d$
40	`blocks.19.norm2`	0.1011	0.1109	$0.575 \cdot \text{clip}(x)$	$1d$
41	`blocks.20.norm1`	0.1243	0.2006	$0.534 \cdot \text{clip}(x)$	$1d$
42	`blocks.20.norm2`	0.1121	0.1720	$0.518 \cdot \text{clip}(x)$	$1d$
43	`blocks.21.norm1`	0.1881	0.2277	$0.459 \cdot \text{clip}(x)$	$1d$
44	`blocks.21.norm2`	0.2204	0.1585	$0.451 \cdot \text{clip}(x)$	$1d$
45	`blocks.22.norm1`	0.1838	0.2024	$0.401 \cdot \text{clip}(x)$	$1d$
46	`blocks.22.norm2`	0.2229	0.1768	$-0.391 \cdot (-\text{clip}(x))$	$1d$
47	`blocks.23.norm1`	0.1905	0.1588	$0.375 \cdot \text{clip}(x)$	$1d$
48	`blocks.23.norm2`	0.1833	0.1258	$-0.362 \cdot (-\text{clip}(x))$	$1d$
49	`norm`	0.1769	0.2045	$0.358 \cdot \text{clip}(x)$	$1d$
	Total				**85d**

E COMPLETE FINE-TUNING TRAJECTORIES

As noted in Section 3.2, the full fine-tuning (DyT-F) and distillation (DyT-D) variants of the Dynamic Tanh baseline exhibited slower convergence at both architectures. Figure 8 provides the complete trajectories with these baselines included. The visualization confirms both methods recover performance at a significantly slower rate and ultimately plateau relatively low compared to the higher performance recovery results. This is most severe for DyT-F at ViT-L, whose mean trajectory remains below the pre-trained baseline for the entire schedule.

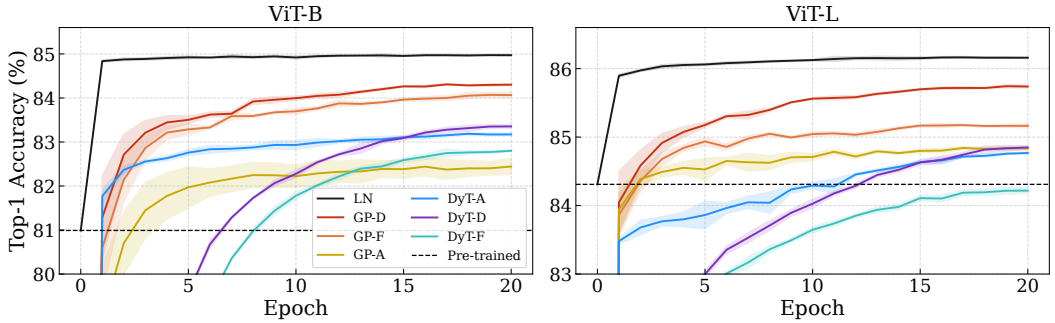

Figure 8: **ImageNet-1K validation performance recovery dynamics. (Left)** ViT-B fine-tuning trajectory comparing the LayerNorm (LN) baseline against GP-A, GP-F, GP-D, DyT-A, DyT-F, and DyT-D. **(Right)** The same comparison for ViT-L. Shaded areas represent ± 1 standard deviation across five independent seeds; black dashed lines denote each architecture's pre-trained baseline (80.99% for ViT-B, 84.31% for ViT-L).

F HARDWARE-CENTRIC COST ANALYSIS

This appendix derives the worst-case FLOP cost of all normalization operations referenced in Section 3.3, under IEEE-754 FP32 unit-roundoff accuracy $u = 2^{-24} \approx 5.96 \times 10^{-8}$. We adopt the convention that each multiplication, addition, subtraction, division, and floating-point round-to-integer counts as exactly one FLOP. Transcendental functions are decomposed into a single unified primitive, the exponential, so that the FLOP bound for $\exp$ propagates to both the sigmoid and the hyperbolic tangent. The resulting per-token costs underpin the per-layer breakdown in Figure 4 (Left) and the aggregate comparison in Figure 4 (Right).

F.1 POLYNOMIAL EVALUATION VIA HORNER'S METHOD

A polynomial of degree N evaluated naively as $\sum_{k=0}^{N} a_k x^k$ requires $N(N+1)/2$ multiplications and N additions. Horner's method (Higham, 2002) rewrites the polynomial in nested form,

$$P(x) = a_0 + x\big(a_1 + x(a_2 + \cdots + x\, a_N)\cdots\big), \tag{5}$$

reducing the cost to exactly N multiplications and N additions, that is, $2N$ FLOPs. This count is FLOP-optimal for generic polynomial evaluation under the standard arithmetic model (Pan, 1966), and we adopt it as the basis for all polynomial cost estimates that follow.

F.2 EXPONENTIAL FUNCTION AS A UNIFIED PRIMITIVE

We evaluate $\exp(x)$ via standard range reduction followed by a Maclaurin truncation (Muller et al., 2018). The cost decomposes into three stages: a range reduction that maps the input into a bounded interval, a polynomial approximation evaluated on that interval, and a reconstruction step that recovers the original argument.

Range reduction. We decompose the input as

$$x = k \ln 2 + r, \qquad k = \operatorname{round}\big(x \cdot \log_2 e\big) \in \mathbb{Z}, \qquad |r| \le \tfrac{\ln 2}{2} \approx 0.3466, \tag{6}$$

which gives the identity $e^x = 2^k \cdot e^r$. Since $\log_2 e = 1/\ln 2$, the integer k is the nearest integer to $x/\ln 2$, ensuring $|r| \le \ln 2/2$. The reduction itself costs four discrete operations: one multiplication ($x \cdot \log_2 e$), one round-to-integer, one multiplication ($k \cdot \ln 2$), and one subtraction ($r = x - k\ln 2$).

Truncation bound. The Maclaurin truncation $E_N(r) = \sum_{k=0}^{N} r^k/k!$ has Lagrange remainder

$$\left|e^r - E_N(r)\right| \;\le\; \frac{e^{|r|}\,|r|^{N+1}}{(N+1)!} \;\le\; \frac{\sqrt{2}\,|r|^{N+1}}{(N+1)!}, \tag{7}$$

where the second inequality uses $e^{|r|} \le e^{\ln 2/2} = \sqrt{2}$. Evaluating (7) at the boundary $|r| = \ln 2/2$ gives:

N	degree	$\sqrt{2}\,(\ln 2/2)^{N+1}/(N+1)!$
6	6	$1.69 \times 10^{-7} \;>\; u$
7	7	$7.30 \times 10^{-9} \;<\; u$

Therefore $N = 7$ is the minimum polynomial order satisfying FP32 unit-roundoff accuracy, and Horner evaluation (5) of degree 7 costs $2 \cdot 7 = 14$ FLOPs.

Reconstruction. The factor 2^k is applied by adding k to the FP32 exponent field, a single integer-arithmetic operation that we count as one FLOP.

Total cost.

$$\mathrm{FLOPs}_{\exp} \;=\; \underbrace{4}_{\text{range red.}} + \underbrace{14}_{\text{Horner } N=7} + \underbrace{1}_{\text{reconstr.}} \;=\; \mathbf{19}. \tag{8}$$

Remark on production implementations. Numerous strategies exist for evaluating the exponential function in practice, ranging from straight Maclaurin or Chebyshev truncations to minimax (Remez) polynomials, table-driven hybrids that combine small look-up tables with low-degree polynomial corrections, and multi-phase schemes that fall back to extended precision for hard inputs (Muller et al., 2018; Tang, 1989). Each strategy trades polynomial degree, table size, and accuracy differently, and production libraries typically adopt a higher-degree minimax polynomial to obtain the required accuracy with a tight worst-case error bound. Our reported $N = 7$ is the *mathematical* minimum derived from the Maclaurin truncation bound alone for FP32 unit-roundoff accuracy, and we adopt it as the FLOP baseline to obtain a tight lower bound on arithmetic cost.

F.3 Transcendental functions via the unified exp primitive

The sigmoid and hyperbolic tangent are both expressed as direct compositions of a single $\exp$ evaluation, allowing the FLOP cost derived in Appendix F.2 to propagate to both. We treat $\exp$ as a unified primitive of 19 FLOPs and account separately for the algebraic operations that compose it.

F.3.1 Sigmoid

The sigmoid is defined as $\sigma(x) = 1/(1 + e^{-x})$. Its evaluation decomposes into one negation, one exponential, one addition, and one division:

$$\mathrm{FLOPs}_{\sigma} \;=\; \underbrace{1}_{-x} + \underbrace{19}_{\exp} + \underbrace{1}_{+1} + \underbrace{1}_{1/(\cdot)} \;=\; \mathbf{22}. \tag{9}$$

F.3.2 Hyperbolic tangent

Multiplying numerator and denominator of $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$ by e^x yields the algebraically equivalent form

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}, \tag{10}$$

which requires only a single $\exp$ evaluation. The cost decomposes into one doubling, one exponential, one subtraction, one addition, and one division:

$$\mathrm{FLOPs}_{\tanh} \;=\; \underbrace{1}_{2x} + \underbrace{19}_{\exp} + \underbrace{1}_{-1} + \underbrace{1}_{+1} + \underbrace{1}_{\div} \;=\; \mathbf{23}. \tag{11}$$

F.4 Per-token and per-layer FLOP cost of normalization methods

We now apply the FLOP convention established in Appendices F.1–F.3 to the three normalization methods compared in Section 3.3: standard LayerNorm, DyT, and our GP-evolved expressions. All counts exclude the per-channel affine transform (γ, β), which is shared across all methods and therefore does not affect the comparison.

F.4.1 LayerNorm

The per-token FLOP cost of LayerNorm follows directly from its arithmetic definition $y_i = (x_i - \mu)/\sqrt{\sigma^2 + \epsilon}$, with $\mu = (1/d)\sum_i x_i$ and $\sigma^2 = (1/d)\sum_i (x_i - \mu)^2$. Table 7 enumerates the operations and the resulting FLOP count. The two reduction passes for μ and σ^2 couple all d feature dimensions, creating the global cross-feature dependency that motivates the analysis in Section 3.3.

Table 7: Per-token FLOP count for LayerNorm.

Step	**Operations**	**FLOPs**
$\sum_i x_i$	$d-1$ adds	$d-1$
$\mu = (\cdot)/d$	1 div	1
$x'_i = x_i - \mu$	d subs	d
$(x'_i)^2$	d muls	d
$\sum_i(\cdot)$	$d-1$ adds	$d-1$
$\sigma^2 = (\cdot)/d$	1 div	1
$(\cdot) + \epsilon$	1 add	1
$\sqrt{(\cdot)}$	1 sqrt	1
$1/(\cdot)$	1 div	1
$x'_i \cdot (\cdot)$	d muls	d
Total		$\mathbf{5d+3}$

F.4.2 DyT

DyT is defined as $\mathrm{DyT}(x) = \tanh(\alpha x)$, with α a learnable scalar shared across the feature dimension. Each token therefore incurs d multiplications for the input scaling and d tanh evaluations, yielding $d + 23d = 24d$ FLOPs per token under the unified $\exp$ primitive of Appendix F.3.

F.4.3 GP-evolved expressions

The per-layer FLOP cost of our GP solutions is reported in the rightmost column of Table 5 (Appendix D), computed under the same convention. Each $\tanh$ contributes 23 FLOPs and each σ contributes 22 FLOPs; `clip` operations are neglected because they reduce to a comparison and a select on hardware that supports conditional moves, neither of which we count as FLOPs. Aggregating across all 25 normalization layers in ViT-B, the GP expressions require $31d$ FLOPs per token, compared to $125d + 75$ for LayerNorm and $600d$ for DyT.

We note one deliberate asymmetry between this deployment cost and the complexity metric used during the evolutionary search. Assigning `clip` its true cost of zero during the search makes redundant `clip` nesting complexity-neutral, and we observed the search exploiting this to stack such operations without any effect on the resulting function. We therefore charge `clip` a nominal cost of one FLOP during the search only, purely as a regularizer against this degenerate behaviour. All reported hardware costs, including those in Table 5 and Figure 4, use the true zero cost.

F.5 Memory access analysis

Beyond arithmetic cost, normalization replacements differ in the volume of off-chip memory traffic they generate, which on bandwidth-bound hardware often dominates inference latency and energy. We quantify this traffic in bytes per token assuming FP32 storage (4 bytes per element) and a feature dimension d, counting only read traffic since the methods we compare differ structurally in how many times the input vector must be re-read from memory but not in their output write behaviour.

LayerNorm. The reduction operations for μ and σ^2 couple all d feature dimensions, forcing two separate passes over the input vector. Pass 1 reads x to compute the global statistics, and pass 2 reads x again to apply the normalization. The cross-feature dependency in pass 1 forbids fusion with pass 2, since the per-element normalization requires μ and σ^2 to be available in full before any output element can be produced. The resulting read cost is

$$\mathrm{Bytes}_{\mathrm{LN}}^{\mathrm{read}} \;=\; 2 \cdot 4d \;=\; 8d \;\text{ bytes per token.} \tag{12}$$

DyT and GP. Both replacements are element-wise scalar mappings, requiring a single read of x to produce each output element. The single-pass execution fuses naturally with adjacent operations and incurs only

$$\mathrm{Bytes}_{\mathrm{DyT}}^{\mathrm{read}} \;=\; \mathrm{Bytes}_{\mathrm{GP}}^{\mathrm{read}} \;=\; 4d \;\text{ bytes per token,} \tag{13}$$

half the read traffic of LayerNorm.

Aggregate. Across all 25 normalization layers in ViT-B with sequence length $\ell = 197$ and feature dimension $d = 768$, the total read access per forward pass is $\mathrm{Bytes}_{\mathrm{LN}} = 25 \cdot \ell \cdot 8d \approx 29.0$ MB for LayerNorm and $\mathrm{Bytes}_{\mathrm{DyT}} = \mathrm{Bytes}_{\mathrm{GP}} = 25 \cdot \ell \cdot 4d \approx 14.5$ MB for DyT and GP. These are the values used for Figure 4 (Right).