Hybrid Transformer Models with Performer and Regular Attention

Salim Bennouna (SB5044)
Chenxi Yang (CY2770)
Wuyan Shao (WS2763)
Joe Souaid (JS6637)

December 2025

Abstract

Vision Transformers achieve strong performance through exact self-attention, but their quadratic computational cost limits scalability as image resolution increases. Performer offers a linear-time approximation using random feature kernels, but fully replacing exact attention often leads to optimization difficulties or accuracy drops. This raises a natural question: can we use approximate attention selectively, rather than everywhere, to get better accuracy-efficiency trade-offs?

We systematically study hybrid Vision Transformer architectures that mix standard self-attention and Performer-based attention in the same model. We evaluate five architectural strategies that differ in where approximate attention is placed: fully approximate, fully exact, alternating layers, and asymmetric designs. We also compare two Performer variants (ReLU-based and softmax approximation) and analyze how performance depends on the number of random features m.

Our experiments on MNIST and CIFAR-10 show that hybrid architectures can match or even exceed standard ViT accuracy when approximate attention is placed carefully. Architectures that apply Performer layers early while preserving exact attention in later layers consistently achieve the best accuracy-efficiency balance and remain robust to kernel choice and feature dimensionality. Accuracy plateaus beyond moderate m values, meaning high-quality approximations don't require large feature expansions. We also find that Performer-based models converge more slowly during early training but remain stable and eventually reach baseline performance.

While wall-clock speedups are limited on our small-scale benchmarks, where implementation overhead and exact attention blocks offset the benefits of linear complexity, the theoretical advantages suggest hybrid architectures would be more effective on larger-scale tasks with longer sequences. Our results show that hybrid Performer-Attention ViTs provide a practical way to reduce computational cost without sacrificing accuracy, and highlight that *where* you place approximate attention matters as much as *how well* you approximate it.

1 Problem Statement

The problem we are tackling is the following (**Problem 2** in the projects list):

Implement small hybrid Transformer models with intertwined Performer and regular-attention layers, as well as models applying Performer/regular layers followed by regular/Performer layers. Apply two Performer variants: (a) Performer-ReLU and (b) Performer approximating the softmax kernel with positive random features. Compare these hybrid models with the regular Vision Transformer model. Use the following datasets to conduct the comparison: (a) MNIST, (b) CIFAR-10. Provide code of your implementation and compare also training and inference time of different variants. Analyze how the accuracy of the hybrid models depends on the number of kernel features m used, and determine which layer placement strategy is optimal.

Contents

2 Introduction

Since their introduction [**16**], Transformer-based architectures have become fundamental to modern deep learning, consistently achieving state-of-the-art results across natural language processing [**5**], computer vision, and scientific modeling. What makes Transformers so effective is their self-attention mechanism, which allows every element in an input sequence to interact with every other element. This is different from convolutional networks [**11**] that only look at local neighborhoods, or recurrent networks that process sequences one step at a time. Transformers capture long-range dependencies while being fully parallelizable.

However, this power comes at a cost. Standard self-attention has quadratic complexity with respect to sequence length [**15, 19**], which becomes a serious bottleneck in vision tasks. When we treat images as sequences of patch tokens, longer sequences (from higher resolution or smaller patches) quickly become impractical to process [**3**]. This has driven research into more efficient attention mechanisms that maintain the representational capacity of Transformers while reducing their computational burden.

One promising approach are Performers [**2**], which approximate self-attention in linear time using kernel methods and random feature maps. The key insight is to reformulate softmax attention as a kernel operation [**14**], then approximate it through the FAVOR+ mechanism. This brings both time and memory complexity down from quadratic to linear. The approximation quality can be tuned by adjusting the number of random features, letting us trade off accuracy for efficiency. Performers also support alternative kernels like ReLU-based attention for even greater speed.

Most existing work treats linear attention [**9, 18**] as a drop-in replacement for standard attention throughout the entire model. But we think this might be missing something important. Transformer layers at different depths seem to play different roles, early layers typically learn local, low-level patterns, while deeper layers capture more global, semantic relationships [**13**]. If that's the case, using the same attention mechanism everywhere might not be the best strategy.

In this work, we explore hybrid Vision Transformer [**6**] architectures that combine both standard multi-head self-attention and Performer-based attention in the same model. Instead of viewing Performer as a complete substitute for exact attention, we investigate where it's most effective within the layer stack. Through systematic experiments on MNIST [**12**] and CIFAR-10 [**10**], we compare different hybrid configurations, kernel variants, and numbers of random features, examining their impact on accuracy, training time, inference speed, and scalability.

Our main contributions are:

- We design and test five hybrid Vision Transformer architectures that mix standard and Performer attention in different ways: All-Standard (baseline), All-Performer, Intertwined (alternating layers), Performer-First (early layers only), and Standard-First (late layers only).
- We run experiments on MNIST and CIFAR-10 to understand the accuracy-efficiency trade-offs, testing both ReLU-based and softmax-approximating Performer variants.
- We systematically vary the number of random features m to see how it affects accuracy, providing practical guidance on how to balance approximation quality against computational cost.
- We identify which strategies work best for placing approximate attention in Vision Transformers, showing that hybrid designs can match the accuracy of standard models while being significantly more efficient.

Organization:
The remainder of this report is organized as follows. **Section 3** reviews the theoretical background on Transformers, Vision Transformers, Performers, and the motivation for hybrid attention architectures. **Section 4** describes our experimental setup, including datasets, model architectures, and the training and evaluation protocol. **Section 5** details how to reproduce our experiments, tables, and figures from the provided codebase and notebooks. **Section 6** and **7** then report and analyze the results, covering accuracy trends, learning dynamics, and computational efficiency on both small-scale benchmarks and ImageNet-scale inputs.

3 Background and Related Concepts

This section covers the theoretical background needed to understand our hybrid architectures. We start by reviewing the self-attention mechanism and Transformer encoders (**Section 3.1**), then discuss Vision Transformers (**Section 3.2**) and efficient attention through Performers (**Section 3.3**). Finally, we explain the motivation behind hybrid attention designs (**Section 3.4**).

3.1 Transformers and Self-Attention

The Transformer architecture [16] is built around the self-attention mechanism, which lets the model dynamically weight interactions between all elements in an input sequence. Given an input sequence

$$X \in \mathbb{R}^{L\times d},$$

where L is the sequence length and d is the embedding dimension, self-attention works by projecting X into queries, keys, and values:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V,$$

using learnable weight matrices $W_Q, W_K, W_V \in \mathbb{R}^{d\times d}$. The scaled dot-product attention is then computed as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V. \tag{1}$$

To make the model more expressive, Transformers use multi-head self-attention, which divides the embedding space into multiple subspaces and runs attention separately in each head [17]. Each encoder layer then combines self-attention with a position-wise feed-forward network, along with residual connections [7] and layer normalization [1] for stable training in deep networks.

The main problem with standard self-attention is its computational cost: it requires $\mathcal{O}(L^2d)$ time and $\mathcal{O}(L^2)$ memory. For vision tasks where L is the number of image patches, this quadratic scaling quickly becomes prohibitive. There's also evidence that different Transformer layers learn different levels of abstraction, which suggests that using the same attention mechanism at every depth might not be optimal.

3.2 Vision Transformers

Vision Transformer (ViT), introduced by Dosovitskiy et al. [6], adapts the Transformer architecture to image classification by treating an image as a sequence of tokens. Given an image of size $H \times W$, we split it into non-overlapping patches of size $P \times P$, giving us

$$N = \frac{HW}{P^2}$$

patches total. Each patch is flattened and linearly projected into a d-dimensional embedding.

Since Transformers don't inherently understand spatial structure, we add learnable positional embeddings to the patch embeddings. We also prepend a special learnable classification token to the sequence, whose final representation is used to make the image-level prediction. This sequence of tokens then goes through a stack of Transformer encoder layers.

This design lets every image region interact with every other region, but it comes at a cost: computation scales quadratically with the number of patches, $\mathcal{O}(N^2d)$. One important thing to note is that ViT's architecture is quite modular. While the standard ViT uses the same attention mechanism in every layer, there's nothing in the architecture that requires this. This modularity opens the door to hybrid attention designs, where different layers use different attention mechanisms to better balance expressiveness and efficiency.

3.3 Efficient Attention and Performer

Performers tackle the scalability problem of self-attention by approximating it with kernel methods and random feature maps. The key insight is the *Fast Attention Via positive Orthogonal Random features* (FAVOR+) mechanism, which approximates the softmax attention kernel as

$$\text{softmax}(QK^\top) \approx \phi(Q)\phi(K)^\top, \tag{2}$$

where $\phi(\cdot)$ is a randomized feature map satisfying

$$\exp(x^\top y) \approx \phi(x)^\top \phi(y).$$

In general, ϕ is constructed using random projections $\{\omega_i\}_{i=1}^m$ along with scalar functions $f_1, \ldots, f_m$ (usually trigonometric or exponential) and a normalization term $h(x)$ for proper scaling:

$$\phi(x) = \frac{h(x)}{\sqrt{m}}\left[f_1(\omega_1^\top x), \ldots, f_m(\omega_m^\top x)\right]. \tag{3}$$

This construction works for both softmax approximation and more general attention mechanisms.

With this representation, we can compute attention without ever forming the full attention matrix:

$$\text{Attention}(Q, K, V) \approx D_Q^{-1}\, \phi(Q)\big(\phi(K)^\top V\big), \tag{4}$$

where $D_Q = \text{diag}(\phi(Q)\mathbf{1})$ is a diagonal normalization matrix that ensures each query's attention weights sum to one. This reduces time complexity to $\mathcal{O}(Lmd)$ and memory to $\mathcal{O}(Lm)$. The number of random features m controls the accuracy-efficiency trade-off: larger m gives better approximations but costs more computation. As m increases, the approximation error typically decreases at a rate of $\mathcal{O}(1/\sqrt{m})$, giving us a principled way to balance accuracy and speed. Performers also support ReLU-based attention, which sacrifices some expressiveness for even better efficiency by using simpler activation functions.

3.4 Hybrid Attention Architectures

Standard Vision Transformers use the same attention mechanism in every layer. But there's evidence that early layers tend to focus on local or low-level features, while deeper layers capture more global, semantic relationships. This suggests that different layers might benefit from different trade-offs between exact and approximate attention.

Hybrid attention architectures take advantage of this by mixing exact and approximate attention mechanisms in the same model. Instead of replacing all standard attention with Performer, we can selectively use Performer layers where they work best, potentially reducing computational cost without sacrificing too much accuracy. The goal is to find configurations that give us better accuracy-efficiency trade-offs than using either standard attention or Performer uniformly throughout the network. Understanding how the placement of approximate attention affects both performance and efficiency is a key goal of this work, which we explore through systematic experiments described in **Section 4**.

4 Methodology

This section describes our experimental approach for evaluating hybrid Vision Transformer architectures. Our goal is to isolate how attention type, placement, kernel choice, and approximation accuracy affect both performance and computational efficiency. To do this fairly, all our models use the same ViT backbone and training setup, differing only in which attention mechanisms they use in each layer.

We've organized this section into two parts: experimental setup (**Sections 4.1–4.2**) covers the datasets and preprocessing, while model architecture and evaluation (**Sections 4.3.1–4.3.4**) describes the model components, hybrid designs, training procedures, and evaluation metrics.

4.1 Datasets

We evaluate our hybrid Vision Transformer architectures on two standard image classification benchmarks: MNIST and CIFAR-10. These datasets provide complementary levels of difficulty and are well-understood.

MNIST: MNIST consists of grayscale images of handwritten digits from 0 to 9, giving us 10 classes total. The dataset has 60,000 training images and 10,000 test images, each resized to 28×28 pixels with a single input channel. Because of its low resolution and simple visual structure, MNIST is often used as a baseline benchmark. For us, it provides a controlled environment where we can study optimization behavior and architectural effects without too many confounding factors.

CIFAR-10: CIFAR-10 contains RGB images of natural objects from 10 categories: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*. It has 50,000 training images and 10,000 test images, each at 32×32 pixel resolution with three color channels. Compared to MNIST, CIFAR-10 is considerably more challenging. It has color variation, diverse object shapes, background clutter, and significant variation within each class. This makes it a better test for the representational capacity of our hybrid attention models.

Both datasets can be solved reasonably well with CNNs or even simpler approaches. Our goal here isn't to achieve state-of-the-art results, but rather to use these as standardized testbeds for analyzing the accuracy-efficiency trade-offs between standard, Performer, and hybrid attention mechanisms.

4.2 Data Augmentation

To improve generalization and reduce overfitting, we apply lightweight data augmentation during training. These augmentations are only applied to the training set, validation and test samples are processed without any randomness. We intentionally keep the augmentations modest so that performance differences between models reflect architectural choices rather than aggressive regularization.

- **Random Rotation (MNIST and CIFAR-10).** We randomly rotate input images by up to $\pm 5°$. This helps the model handle slight orientation variations. For MNIST, this captures natural variability in how people write digits. For CIFAR-10, it adds some geometric diversity without changing what the objects actually are.
- **Random Horizontal Flip (CIFAR-10 only).** Images are horizontally flipped with probability $p = 0.5$. This works well for natural images since most object categories still make sense when mirrored left-to-right. We initially tried this on MNIST too, but it caused problems, flipped digits can look ambiguous or like different numbers entirely, which hurt training. So we only use horizontal flips for CIFAR-10.
- **Tensor Conversion and Normalization (MNIST and CIFAR-10).** After the geometric augmentations, we convert images to tensors with pixel values in $[0, 1]$. For MNIST, we normalize using mean 0.5 and standard deviation 0.5, which roughly maps inputs to $[-1, 1]$ and centers the data. For CIFAR-10, we use channel-wise normalization with dataset-specific statistics:

  $$\boldsymbol{\mu} = (0.4914,\ 0.4822,\ 0.4465), \quad \boldsymbol{\sigma} = (0.2023,\ 0.1994,\ 0.2010).$$

  This normalization is important because it prevents any color channel from dominating the early representations, which matters for transformer models where linear projections and dot-product attention are sensitive to input scale.

4.3 Model Architecture

Here we describe the core components of our Vision Transformer architecture. All our models follow the standard ViT design with modular attention mechanisms, which lets us make fair comparisons across different attention strategies.

4.3.1 Patch Embedding and Tokenization

Vision Transformers work with sequences of tokens, so we need to convert images into a one-dimensional representation. Following the standard ViT approach, we split each input image into fixed-size, non-overlapping patches, which become the input tokens for the Transformer encoder. We use square patches of size $P \times P$ with $P = 4$ in all our experiments.

Patch partitioning: For an input image of size $H \times W$, the total number of patches is

$$N = \left(\frac{H}{P}\right)\left(\frac{W}{P}\right). \tag{5}$$

This gives us $N = 49$ patches for MNIST (28×28 images) and $N = 64$ patches for CIFAR-10 (32×32 images). Each patch acts like a local receptive field, similar to the spatial regions processed by convolutional kernels in CNNs.

We chose this patch size to create an intermediate regime where standard self-attention is still computationally manageable, but where linear attention offers meaningful benefits without completely dominating the comparison. This lets us evaluate hybrid architectures without asymptotic complexity effects overshadowing the architectural differences we care about.

Patch embedding: Each patch is linearly projected into a d-dimensional embedding space (we use $d = 64$ throughout). We implement this projection as a convolutional layer with kernel size P and stride P, which extracts patches and applies the linear transformation in one step. The result is a tensor of shape $[B, N, d]$, where B is the batch size.

Class token and positional embeddings: We prepend a learnable classification token `[CLS]` to the sequence of patch embeddings, this will serve as the global image representation. Since Transformers don't inherently understand order, we also add learnable positional embeddings to all tokens to encode their spatial locations. The input to the Transformer encoder is then

$$\mathbf{Z}_0 = [\mathbf{x}_{\text{cls}};\ \mathbf{X}] + \mathbf{E}_{\text{pos}}, \tag{6}$$

where $\mathbf{X} \in \mathbb{R}^{N \times d}$ is the patch embeddings and $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times d}$ is the positional embeddings.

4.3.2 Transformer Block

Our model is a stack of Transformer encoder blocks using a pre-norm architecture with residual connections. For the ℓ-th block, the updates are

$$\mathbf{Z}'_\ell = \mathbf{Z}_\ell + \mathrm{Attn}(\mathrm{LN}(\mathbf{Z}_\ell))\,, \tag{7}$$

$$\mathbf{Z}_{\ell+1} = \mathbf{Z}'_\ell + \mathrm{MLP}(\mathrm{LN}(\mathbf{Z}'_\ell))\,, \tag{8}$$

where $\mathrm{LN}(\cdot)$ is layer normalization and $\mathrm{Attn}(\cdot)$ is either standard multi-head self-attention or Performer attention, depending on the architecture (described in **Section 4.3.3**).

The feed-forward network is a two-layer MLP with GELU activation [**8**]:

$$\mathrm{MLP}(\mathbf{z}) = \mathbf{W}_2\,\mathrm{GELU}(\mathbf{W}_1\mathbf{z}), \tag{9}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d\times(r\cdot d)}$ and $\mathbf{W}_2 \in \mathbb{R}^{(r\cdot d)\times d}$. The expansion ratio r controls the hidden width, giving us a hidden dimension of $r \cdot d$.

4.3.3 Hybrid ViT Architectures

A key part of our methodology is comparing different attention layouts across the depth of the Transformer. We use the term *hybrid* to refer to any architecture that mixes different attention mechanisms (standard and Performer) across layers. Let L be the total number of Transformer layers. We consider these architectures:

- **All-Standard**: all L layers use standard self-attention (our baseline ViT).
- **All-Performer**: all L layers use Performer attention.
- **Intertwined**: alternating Performer and standard attention layers throughout the depth, starting with Performer in the first layer.
- **Performer-First**: Performer attention in the first $\lfloor L/2 \rfloor$ layers, then standard attention in the remaining layers.
- **Standard-First**: standard attention in the first $\lfloor L/2 \rfloor$ layers, then Performer attention in the remaining layers.

This setup lets us test whether approximate attention works better when applied early (during feature extraction) or later (during global representation learning). We present the evaluation of these strategies in **Section 6**.

4.3.4 Classifier Head

After the final Transformer block, we apply layer normalization to the token sequence. We then extract the embedding corresponding to the `[CLS]` token and pass it through a linear classifier:

$$\hat{\mathbf{y}} = \mathbf{W}\,\mathbf{z}_{\mathrm{cls}} + \mathbf{b}, \tag{10}$$

where $\mathbf{W} \in \mathbb{R}^{C\times d}$ and $\mathbf{b} \in \mathbb{R}^{C}$, with C being the number of classes. This gives us the class logits for classification.

4.4 Training and Evaluation Protocol

4.4.1 Training Procedure

We train all models using cross-entropy loss and the AdamW optimizer. Each training epoch consists of a forward pass, loss computation, backpropagation, and parameter updates. To ensure fair comparisons across architectures, we keep all hyperparameters constant within each dataset: embedding dimension d, depth L, number of attention heads, patch size P, batch size, learning rate, and weight decay are all fixed.

We track training loss and accuracy, along with test loss and accuracy after each epoch. For each configuration, we report the best test accuracy achieved across the entire training run as the final performance metric.

4.4.2 Evaluation and Inference-Time Measurement

We evaluate model performance on the test set using classification accuracy. We also measure inference latency by timing forward passes over a fixed number of mini-batches and reporting the mean and standard deviation of per-batch inference time. For GPU experiments, we perform explicit device synchronization before and after each timed forward pass to get accurate and reproducible measurements. This lets us quantify the practical computational benefits of Performer and hybrid architectures beyond just theoretical complexity analysis.

5 Reproducibility

All experiments, tables, and figures in this report can be reproduced from the provided code and logged outputs.

Code availability:
Our full implementation (models, training scripts, and plotting notebooks) is available at:

`https://github.com/SalimBennouna/hybrid-vit-performer`

You can clone and set up the repository like this:

```
git clone https://github.com/SalimBennouna/hybrid-vit-performer.git
cd hybrid-vit-performer
```

Conda-based installation:
If you use Conda, you can create the environment with:

```
conda env create -f environment.yml
conda activate hybrid-vit-performer
```

Pip-based installation:
If you don't use Conda, we also provide a pip-based setup:

```
pip install -r requirements.txt
```

How to reproduce figures:
Each figure caption includes the name of the notebook that generates it (e.g., `figure1.ipynb`). Just run all cells in the corresponding notebook in order to reproduce the figure from our saved experiment logs.

How to reproduce trainings:
You can reproduce individual experiments using our training script with the appropriate configuration. For example, to reproduce a single MNIST experiment with the Performer-first architecture:

```
python scripts/run_one.py  --dataset MNIST --architecture performer_first --kernel_type relu \
  --m_features 128 --seed 42
```

To reproduce our full experimental grid for a dataset (all architectures, kernel types, and feature sizes), use:

```
python scripts/full_grid.py --dataset MNIST
```

This runs all configurations we used: architectures `[all_standard, intertwined, performer_first, standard_first, all_performer]`, kernel types `[relu, softmax]`, and kernel feature sizes `m` $\in \{8, 16, 32, 64, 128, 256\}$. Note that running the full grid takes considerable computation time.

Determinism and numerical variability:
Training time per epoch and inference latency will vary depending on your hardware and software stack, this is expected and doesn't affect the qualitative comparisons between models.

To make predictive performance as reproducible as possible, we seed all sources of randomness: Python's random number generator, NumPy, and PyTorch (both CPU and GPU), and apply per-worker seeding in data loaders. We also enable deterministic CuDNN execution, disable benchmarking, request deterministic PyTorch algorithms when available, and configure deterministic cuBLAS behavior on CUDA systems.

Even with these precautions, we can't guarantee exact bitwise reproducibility across different hardware (e.g., CUDA GPUs, Apple MPS, or different GPU architectures). Low-level floating-point arithmetic and parallel execution order can introduce small numerical variations. So if you re-run experiments on different hardware, you might see slightly different accuracy values than what we report here. These differences are always negligible though, and all the qualitative trends, relative comparisons between architectures, and conclusions remain the same.

6 Results

We begin by reporting the *raw experimental results* for MNIST and CIFAR-10 (Tables 1 and 2), which include final train/test accuracy as well as average training time per epoch and inference latency. We then provide an exhaustive analysis by decomposing the problem into the following subquestions:

1. **Do hybrid Performer–Attention ViTs match or approach standard ViT accuracy?**
2. **Which hybrid layer-placement strategy is most effective?**
3. **How sensitive are hybrid models to the choice of Performer kernel?**
4. **How does accuracy depend on the number of kernel features m?**
5. **Do hybrid architectures exhibit different learning dynamics than ViT?**
6. **Do hybrid architectures provide practical wall-clock speedups on small-scale benchmarks?**
7. **Do Performer-style speed benefits become clear at ImageNet-scale sequence lengths?**

Table 1: MNIST: All Results. Reproducible with notebook `main_table.ipynb`.

Model	Kernel	m	Final Test Acc	Final Train Acc	Time / Epoch (s)	Infer Time (ms / batch)
AllPerformer	ReLU	16	98.5800	99.5233	13.9349	8.3643
		32	98.6200	99.6533	15.9296	9.7538
		64	98.3400	99.6450	19.1523	13.0231
		128	**98.7600**	99.7233	26.5014	19.3046
		256	98.5200	99.5783	39.3978	30.5364
	Softmax	16	98.5500	99.3983	16.2898	10.8524
		32	98.6500	99.3117	19.3035	14.0548
		64	98.2600	99.3650	25.6207	21.1250
		128	98.4600	99.3633	37.7945	34.6716
		256	98.4200	99.2933	60.5609	60.3646
Intertwined	ReLU	16	98.5900	99.6867	**12.7897**	7.4907
		32	98.5900	99.6267	13.7581	8.3375
		64	98.4300	99.6083	15.5546	9.6629
		128	98.7400	99.6000	19.0576	13.3217
		256	98.4500	99.5717	25.6425	18.7727
	Softmax	16	98.7000	99.6267	13.9287	8.7389
		32	98.6600	99.5717	15.4871	10.2379
		64	98.3300	99.5850	18.6798	13.7154
		128	98.4700	99.5317	24.7854	20.7304
		256	98.5600	99.6133	36.3201	34.0365
PerformerFirst	ReLU	16	98.5200	99.6017	12.8602	**7.4769**
		32	98.6800	99.5467	13.7947	8.3128
		64	98.6100	99.6867	15.5137	9.8063
		128	98.7000	99.6817	18.9469	13.5044
		256	98.7200	99.7133	25.6813	18.7585
	Softmax	16	98.5900	99.5750	13.9775	8.6542
		32	98.6900	99.5783	15.6175	10.3779
		64	98.7200	99.6100	18.6489	13.9085
		128	98.7000	99.6033	24.7883	21.5269
		256	98.6700	99.6450	36.1769	33.6348
StandardFirst	ReLU	16	98.5800	99.6400	12.8185	7.6440
		32	98.3800	99.5800	13.7077	8.2627
		64	98.4800	**99.7667**	15.3607	9.4907
		128	98.6000	99.6300	18.8714	12.7920
		256	98.4700	99.7250	25.4726	18.5431
	Softmax	16	98.2300	99.4850	13.8565	8.4290
		32	98.1100	99.5217	15.5801	10.3385
		64	98.2100	99.6767	18.8636	13.9054
		128	98.4500	99.5250	25.0927	20.8986
		256	98.4500	99.4983	47.6635	43.3401
AllStandard	N/A	N/A	98.5400	99.5050	**11.3801**	**6.3688**

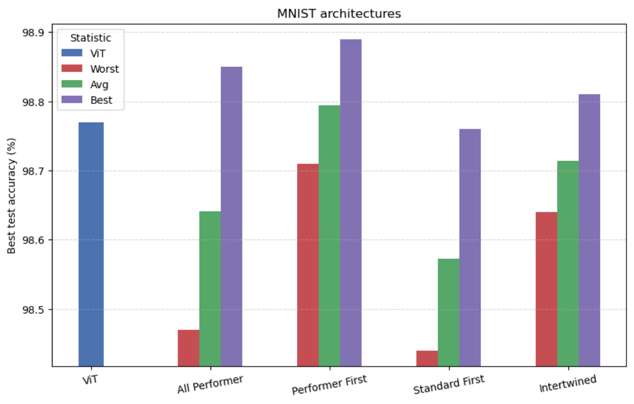
Table 2: CIFAR-10: All Results. Reproducible with notebook `main_table.ipynb`

Model	Kernel	m	Final Test Acc	Final Train Acc	Time / Epoch (s)	Infer Time (ms / batch)
AllPerformer	ReLU	16	72.2900	89.0120	29.7920	12.7822
		32	74.2700	**90.7240**	32.0413	15.1117
		64	72.9300	90.5280	37.6810	20.5412
		128	72.4900	89.6660	48.5818	37.3386
		256	72.4000	89.3560	66.9876	51.2998
		512	73.8100	88.9160	106.9143	98.2485
	Softmax	16	71.7400	83.9420	33.2875	18.3053
		32	72.7300	81.9920	38.5291	23.9698
		64	73.3700	81.6420	47.2564	34.7289
		128	73.0000	82.0760	65.7007	55.0714
		256	73.0900	81.6220	100.7130	97.8083
		512	73.6200	82.3240	172.5789	187.1309
Intertwined	ReLU	16	73.2600	90.6780	**26.0930**	**9.6408**
		32	73.0500	90.2460	27.0740	9.9550
		64	73.6700	90.3260	27.3186	11.0295
		128	73.3300	90.0480	30.6122	13.7734
		256	72.3200	90.5560	37.0278	21.7177
		512	72.2700	90.2360	49.6086	30.9991
	Softmax	16	73.0200	87.8820	26.9947	10.8345
		32	72.5800	88.7940	27.6233	11.9672
		64	73.6300	88.6120	30.4295	15.3398
		128	73.7600	88.5000	35.6770	22.0660
		256	72.8500	89.2840	47.2763	35.6129
		512	73.0300	88.7260	70.6707	62.8751
PerformerFirst	ReLU	16	71.3300	89.6900	30.0093	12.9921
		32	72.6700	89.7680	31.2222	12.7773
		64	72.2800	89.9400	34.7435	15.6508
		128	71.9700	90.0380	38.5590	24.5116
		256	72.5200	90.0220	47.8570	32.0697
		512	73.7200	90.5400	74.7609	52.8222
	Softmax	16	73.1900	88.2740	31.6459	13.8275
		32	72.2100	88.8640	34.6065	19.7200
		64	73.6000	88.3240	38.3809	25.6387
		128	**74.2900**	88.7560	54.8560	38.9071
		256	73.1300	88.4720	64.9799	53.2913
		512	73.5800	89.0540	96.3070	108.5148
StandardFirst	ReLU	16	72.7500	89.4880	28.3190	12.2151
		32	73.3900	90.1240	29.4142	13.4207
		64	73.4400	90.4580	32.4580	15.7429
		128	72.9900	89.9800	42.1755	26.2275
		256	73.8500	89.6840	50.4520	35.1726
		512	73.5200	90.0880	74.4551	54.4031
	Softmax	16	72.2100	86.9460	30.2298	15.6928
		32	71.8800	86.9340	35.8511	18.2394
		64	72.9000	87.3000	37.3495	24.6691
		128	72.1600	86.9600	49.3120	38.9650
		256	71.5400	86.4600	66.3653	57.2466
		512	71.6600	87.4200	100.0499	105.0429
AllStandard	N/A	N/A	72.9600	89.9780	28.3620	12.3959

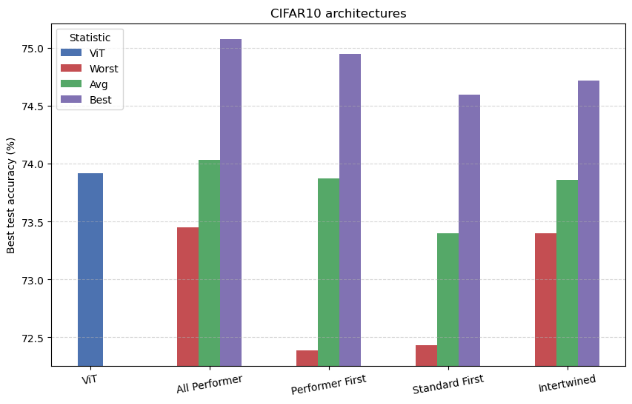
6.1 Do hybrid Performer–Attention ViTs match or approach standard ViT accuracy?

Here we examine whether hybrid Performer–Attention Vision Transformer (ViT) architectures can match or approach the accuracy of a fully standard ViT. We're focused on architectural effects rather than specific hyperparameter tuning. To do this, we aggregate results across all kernel types and random feature dimensions m, and look at the resulting accuracy distributions.

For each of the four Performer-based hybrid architectures (AllPerformer, PerformerFirst, StandardFirst, and Intertwined), we report three statistics computed over all (kernel, m) configurations: the worst, average, and best observed test accuracy. These are shown as histograms in **Figure 1**, alongside the standard ViT baseline.



(a) MNIST



(b) CIFAR-10

Figure 1: Test accuracy distributions across architectures. For hybrid Performer–Attention models, each bar summarizes the worst, average, and best accuracy over all kernel and feature dimension configurations. The standard ViT is shown as a reference baseline. Reproducible with notebook `figure1.ipynb`

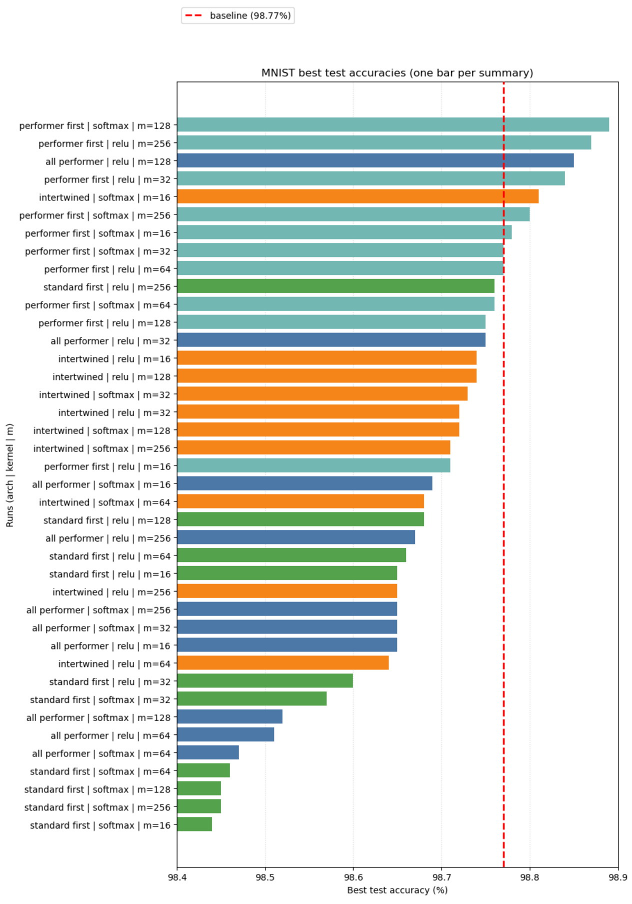
MNIST: On MNIST (**Figure 1a**), all hybrid architectures track the standard ViT baseline pretty closely. While the worst configurations of some hybrids fall slightly below the ViT, their average performance stays consistently high. What's interesting is that the best configurations of PerformerFirst and AllPerformer actually slightly exceed the standard ViT accuracy, showing that the Performer approximation doesn't inherently limit what the model can learn on this dataset. The relatively narrow gap between worst and best cases also suggests that hybrid architectures are fairly robust to kernel choice and feature dimensionality on simpler vision tasks.

CIFAR-10: CIFAR-10 is more challenging (**Figure 1b**), and architectural differences become clearer. Here, hybrid architectures show a wider spread between worst and best configurations, indicating they're more sensitive to hyperparameter choices. Still, the average accuracy of all hybrids remains competitive with standard ViT. Notably, the best configurations of AllPerformer and PerformerFirst clearly beat the baseline, while Intertwined and StandardFirst come close. This shows that hybrid Performer–Attention designs can retain, and sometimes improve upon,ViT-level accuracy even on more demanding datasets.

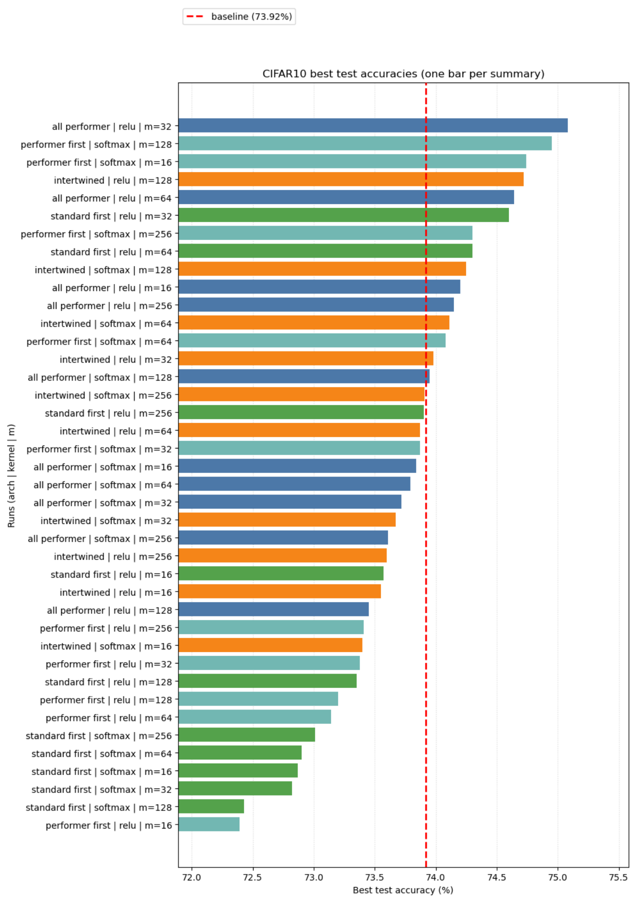
Overall, these results demonstrate that hybrid Performer–Attention ViTs can match or approach the accuracy of fully standard ViTs on both MNIST and CIFAR-10. While hybrid models show greater variance across configurations (especially on CIFAR-10), their best and average performance remains competitive with, and sometimes better than, the baseline. That said, MNIST and CIFAR-10 are relatively simple benchmarks by modern standards and may not fully stress the expressive power of exact self-attention. On more demanding, large-scale, or fine-grained vision tasks where long-range dependencies and precise attention patterns matter more, fully standard ViTs might retain a performance advantage. Still, our results suggest that for small to medium-scale tasks, the Performer approximation can be integrated without a significant accuracy penalty, making hybrid architectures an attractive option for balancing accuracy and efficiency.

6.2 Which hybrid layer-placement strategy is most effective?

Here we look at how test accuracy varies across different hybrid Performer–Attention architectures. Our goal is to isolate the effect of architectural design, specifically where we place Performer versus standard self-attention layers, while abstracting away from kernel types and random feature dimensions m. We focus on comparing performance trends across architectures rather than individual configurations (**Figure 2**).



(a) MNIST



(b) CIFAR-10

Figure 2: Best test accuracy across architectures. Each bar shows the best-performing configuration for a given architecture, aggregated over kernel type and random feature dimension m. The dashed line indicates the standard ViT baseline. Reproducible with notebook `figure2.ipynb`

MNIST: On MNIST, architecture doesn't matter much for performance (**Figure 2a**). All architectures achieve high test accuracy clustered tightly around the standard ViT baseline. This narrow spread reflects a ceiling effect, the dataset is simple enough that it doesn't really distinguish between different attention mechanisms.
Within this compressed range, *PerformerFirst* consistently gives the best results. Across different kernels and feature dimensions, it most frequently achieves the highest accuracy, often matching or slightly beating the baseline ViT. *Intertwined* sits in the middle, it's stable but rarely the best. *StandardFirst* tends to underperform relative to the other hybrids, especially with less favorable kernel or feature choices. Still, the differences are modest, and all architectures perform competitively due to the task's simplicity.

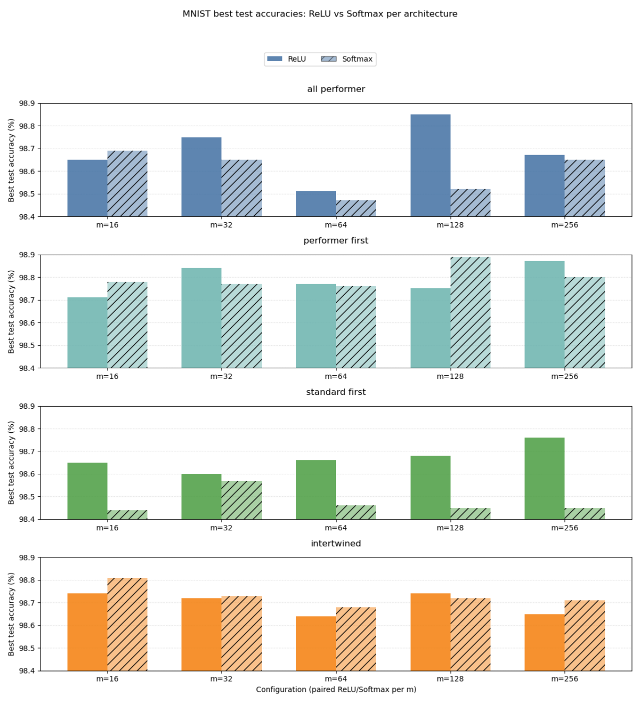
CIFAR-10: On CIFAR-10, architectural effects become much clearer (**Figure 2b**). The dataset's increased complexity leads to better separation between architectures, making performance trends more obvious. Here, *PerformerFirst* frequently outperforms the standard ViT baseline when properly tuned, showing that placing Performer layers early in the network can preserve or even improve classification accuracy.
Intertwined again shows intermediate behavior, it stays close to the baseline but is rarely among the top performers. While it avoids major degradation, it doesn't consistently capture the benefits we see with PerformerFirst. *StandardFirst* more often ends up at the lower end of the accuracy spectrum, suggesting it's more sensitive to how exact self-attention and its linear approximation interact when the approximation is introduced later in the network.

Discussion:

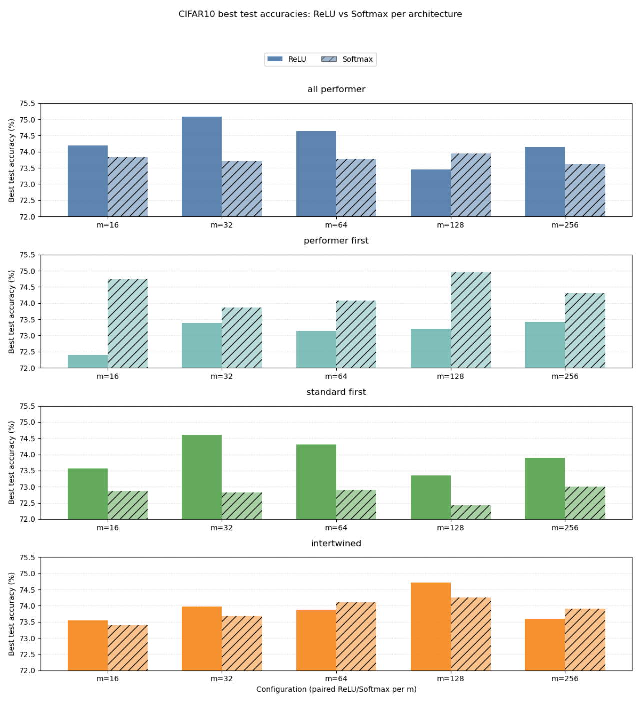
These results suggest a consistent architectural ranking across both datasets (**Figure 2**): *PerformerFirst* performs best on average, followed by *Intertwined*, then *StandardFirst*. This is a statistical trend rather than an absolute rule, individual results still depend on kernel choice and feature dimensionality, but the pattern holds across datasets.
The contrast between MNIST and CIFAR-10 also highlights the role of task complexity. Architectural differences get compressed on simpler benchmarks but become more apparent as the classification problem gets harder. Importantly, even on CIFAR-10, most hybrid architectures stay close to the standard ViT baseline, showing that integrating Performer layers doesn't systematically hurt accuracy. When placed carefully, linear attention approximations can be an effective architectural alternative that maintains competitive accuracy while enabling efficiency gains.

6.3 How sensitive are hybrid models to the choice of Performer kernel?

Now we compare the two Performer kernel variants, ReLU and softmax approximation, across architectures and datasets. Instead of looking at individual configurations, we analyze systematic trends as a function of where the kernels are placed and task complexity (**Figure 3**).



(a) MNIST



(b) CIFAR-10

Figure 3: Comparison of Performer kernels (ReLU vs softmax approximation). Each bar shows the best test accuracy for a given architecture and feature dimension m. Reproducible with notebook `figure3.ipynb`

StandardFirs: For *StandardFirst*, the ReLU kernel consistently outperforms the softmax approximation across all feature dimensions m on both MNIST and CIFAR-10 (**Figures 3a** and **3b**). This pattern is remarkably stable and doesn't depend on the dataset. When Performer layers are introduced after exact self-attention, the representations are already abstract and heterogeneous, which makes the softmax approximation more sensitive to normalization errors. The ReLU kernel provides a more robust approximation in this late-stage regime, leading to consistently better performance.

Intertwined: In the *Intertwined* architecture, there's almost no performance difference between ReLU and softmax kernels on either dataset (**Figures 3a** and **3b**). Across all values of m, the two kernels give nearly identical accuracies with no consistent winner. This suggests that alternating between approximate and exact attention layers effectively balances out kernel-specific biases, making kernel choice basically irrelevant in this setting.

PerformerFirst: We see different behavior for *PerformerFirst*. On CIFAR-10, the softmax approximation consistently beats the ReLU kernel across all values of m (**Figure 3b**). When Performer layers operate on low-level features early in the network, the extra expressiveness of the softmax approximation becomes valuable, especially on more complex datasets. The exact attention layers that follow can then refine these representations, compensating for approximation noise while keeping the benefits of a richer early attention mechanism. On MNIST, this effect is less obvious due to the task's simplicity (**Figure 3a**).

AllPerformer: For *AllPerformer*, neither kernel consistently dominates (**Figure 3**). On MNIST, differences are small across kernels. On CIFAR-10, the ReLU kernel tends to work better at smaller feature dimensions, while the softmax approximation becomes competitive as m increases. This reflects a trade-off between robustness and expressiveness in fully approximate networks, softmax benefits require sufficiently large random feature dimensions to show up.

Summary: Overall, kernel effectiveness strongly depends on where it's placed in the architecture (**Figure 3**). ReLU kernels work best when Performer layers are applied late, softmax approximations are better when applied early on complex datasets, and kernel choice doesn't really matter in intertwined designs. These results show that you can't pick a kernel without considering the architecture and task complexity.

6.4 How does accuracy depend on the number of kernel features m?

Now we look at how model accuracy varies with the number of random kernel features m used in Performer-style attention. **Figure 4** shows test and training accuracy for MNIST and CIFAR-10 across a wide range of m, for all hybrid architectures and both kernel types. We use a logarithmic scale on the horizontal axis to better show trends across orders of magnitude.

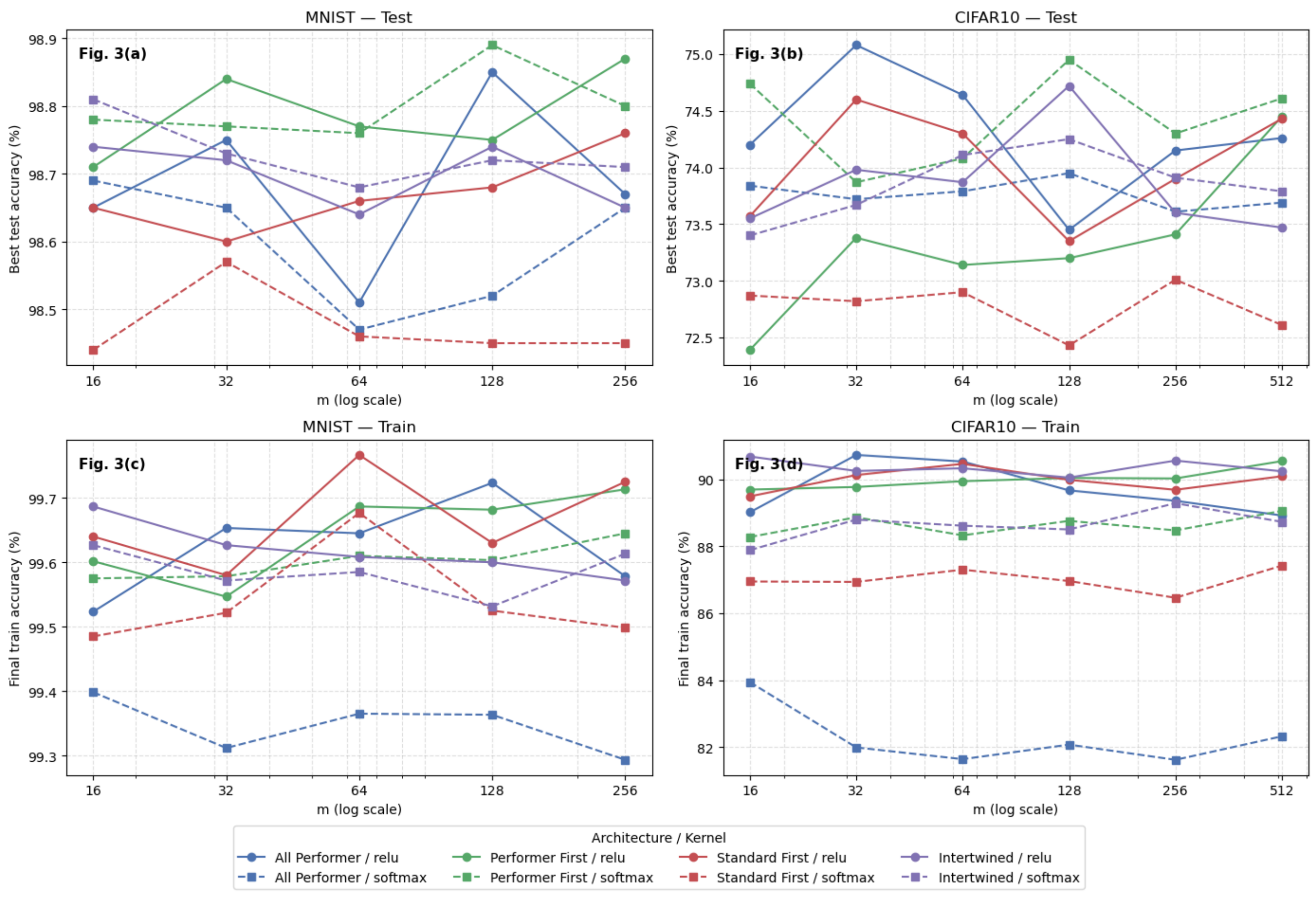


Figure 4: Accuracy as a function of the number of kernel features m. Top row: test accuracy on MNIST (a) and CIFAR-10 (b). Bottom row: corresponding training accuracy (c, d). Results are shown for all hybrid architectures and both kernel types, with m on a logarithmic scale. Reproducible with notebook `figure4.ipynb`

MNIST: On MNIST (**Figures 3(a)** and **3(c)**), accuracy barely depends on m. Test accuracy stays consistently high across all architectures, with variations typically within $\pm 0.2\%$ over the entire range of m. Increasing m beyond moderate values doesn't give systematic improvements, and performance often peaks at intermediate feature dimensions (e.g., $m \in \{32, 128\}$). Training accuracy is close to saturation for all models, indicating that even small values of m provide enough approximation capacity for this dataset.

CIFAR-10: On CIFAR-10 (**Figures 3(b)** and **3(d)**), the dependence on m is more noticeable. Test accuracy generally improves when we increase m from very small values, but tends to plateau around $m \gtrsim 64$, with fluctuations of at most $\approx 0.5\%$ after that. In several cases, increasing m further leads to marginal gains or even slight degradation, especially for softmax-based kernels. Training accuracy increases more smoothly with m, reflecting better expressiveness, but this doesn't always translate to better generalization.

Across both datasets, ReLU-based kernels show smoother and more stable behavior as m increases, while softmax-based kernels are more sensitive to the choice of feature dimension, particularly on CIFAR-10. Importantly, this qualitative dependence on m is consistent across architectures, no hybrid design needs very large kernel feature dimensions to reach competitive accuracy.

Overall, once m reaches moderate values, increasing the number of kernel features gives diminishing returns. Quantitatively, for MNIST, test accuracy varies by less than $\pm 0.2\%$ across all tested values of m, while for CIFAR-10, most architectures stay within $\pm 0.5\%$ of their peak performance for $m \geq 64$. That said, MNIST and CIFAR-10 are relatively simple benchmarks by modern standards and may not fully test the limits of kernel-based attention approximations. On more complex or large-scale vision tasks, higher values of m could matter more. Still, on these datasets, hybrid Performer–Attention ViTs work well with relatively small feature dimensions, enabling good accuracy-efficiency trade-offs without sacrificing performance.

6.5 Do hybrid architectures exhibit different learning dynamics than ViT?

To complement our final accuracy comparisons, we now analyze *learning dynamics*, specifically convergence speed, training stability, and how the train-test gap evolves. **Figure 5** shows accuracy as a function of training epoch for a fixed number of kernel features $m = 128$. We chose this value based on our earlier results showing that accuracy depends only weakly on m once it reaches moderate values, and that extremely small or large feature dimensions don't provide systematic benefits. Fixing m at an intermediate value lets us focus on architectural effects without mixing in capacity-related variations.

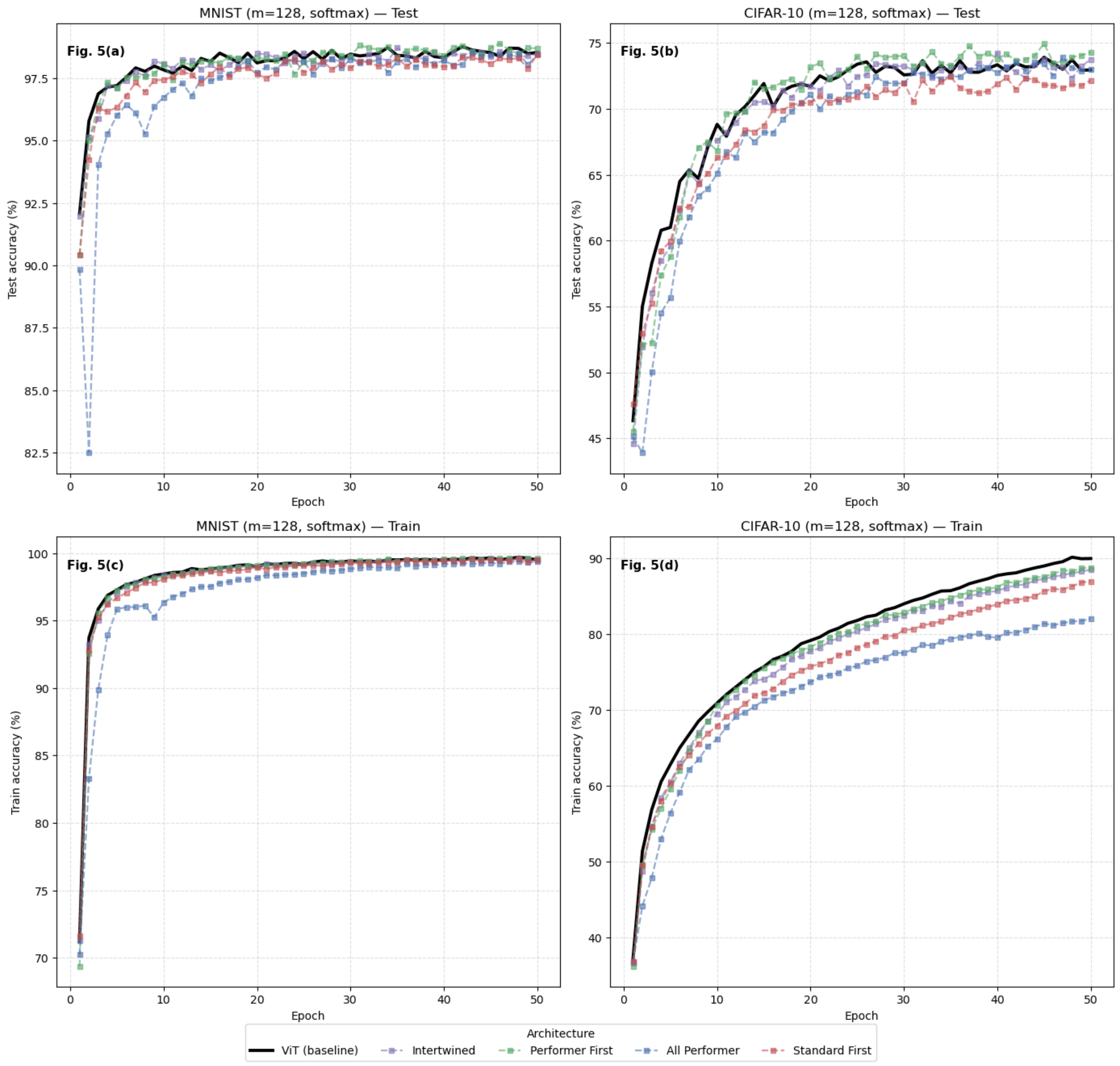


Figure 5: Learning curves at $m = 128$. (a) MNIST test accuracy, (b) CIFAR-10 test accuracy, (c) MNIST train accuracy, and (d) CIFAR-10 train accuracy, for the ViT baseline and hybrid Performer–Attention architectures. Reproducible with notebook `figure5.ipynb`

Similarly, we fix the Performer kernel to the softmax approximation, which is typically more sensitive to approximation quality and can lead to less stable optimization compared to ReLU-based kernels. By choosing the less stable kernel, we deliberately test a more challenging regime, which gives us a clearer view of training stability and convergence behavior across hybrid architectures. **Figures 5(a)** and **5(b)** show test accuracy on MNIST and CIFAR-10, while **Figures 5(c)** and **5(d)** show the corresponding training accuracy, comparing a standard ViT baseline to several hybrid Performer–Attention designs.

Convergence speed (early vs late training):

A consistent pattern emerges across datasets: hybrid Performer–Attention models learn slightly slower than standard ViT during the *early* phase of training, but progressively close the gap.

On MNIST (**Figure 5(a)**), all models reach high test accuracy within just a few epochs. Hybrids may lag a bit in the first iterations, but differences become small by around 10 epochs and stay negligible after that.

On CIFAR-10 (**Figure 5(b)**), the effect is clearer: during roughly the first 10-15 epochs, hybrid models trail the ViT baseline, then catch up steadily, and by later epochs reach comparable performance (with some hybrids approaching or matching the baseline).

Training stability:
Despite the approximation introduced by Performer-style attention, training remains *stable* across all hybrid variants. Both test and train curves evolve smoothly without abrupt drops, oscillations, or divergence (**Figures 5(a-d)**). This is an important practical observation, replacing exact attention in parts of the network doesn't introduce pathological optimization behavior in these settings.

Architecture-dependent dynamics:
The relative ordering of curves provides some architectural insight. Hybrids that approximate attention more aggressively tend to pay a larger early-training cost. Architectures closer to standard ViT (e.g., approximating only a subset of attention blocks) generally track the baseline more closely, while the fully approximated variant (AllPerformer) shows the slowest early progress on both MNIST and CIFAR-10. This supports the intuition that *where* the approximation is inserted matters for optimization dynamics, approximating fewer blocks preserves early learning speed.

Optimization vs generalization (role of training curves):
The training curves (**Figures 5(c)** and **5(d)**) are essential for interpreting the test behavior. The early lag we see in CIFAR-10 test accuracy is mirrored by a corresponding lag in training accuracy, indicating this is primarily an *optimization* phenomenon rather than a generalization issue. In other words, hybrids don't just generalize worse early on, they also fit the training data more slowly, consistent with the idea that approximation noise or reduced gradient quality can slow initial representation formation.

Capacity and underfitting signals:
On MNIST (**Figure 5(c)**), all models approach near-saturated training accuracy, suggesting that capacity is sufficient even with approximate attention and that the dataset doesn't really stress the approximation. On CIFAR-10 (**Figure 5(d)**), the baseline ViT achieves the highest training accuracy, while hybrid models plateau at slightly lower values. This gap indicates either a modest reduction in effective capacity or increased optimization difficulty for hybrids on a harder dataset, which is consistent with the larger spread we saw in earlier results. Importantly, the corresponding test curves (**Figure 5(b)**) remain competitive, suggesting this doesn't translate into a severe generalization penalty in our experiments.

Takeaway:
Overall, **Figure 5** shows that Performer-based approximations primarily affect *early optimization speed*, not training stability. Hybrid models converge more slowly at the beginning, especially on CIFAR-10, but the gap shrinks with training, and final test performance stays close to standard ViT. These dynamics reinforce a central point of this work: hybrid Performer–Attention architectures can provide an attractive accuracy-efficiency trade-off, where the main cost is a temporary slowdown in early learning rather than unstable training or a fundamental loss in achievable accuracy.

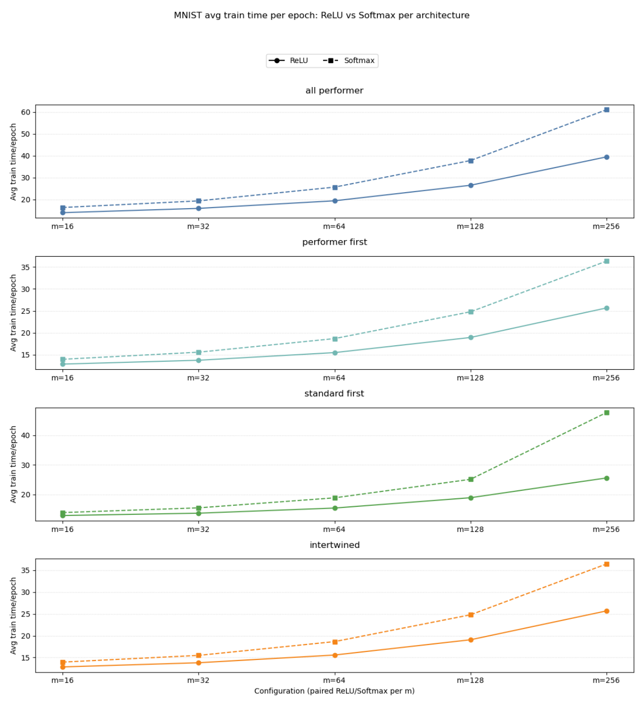
6.6 Do hybrid architectures provide practical wall-clock speedups on small-scale benchmarks?

Before presenting the efficiency results, we need to clarify the experimental setup for our timing measurements. We initially ran experiments on NVIDIA T4 GPUs using Google Colab, but the observed training and inference times showed inconsistent and non-monotonic behavior, in several cases, more complex configurations appeared faster than simpler ones. These effects weren't reproducible across runs and didn't align with the expected computational scaling of the models.

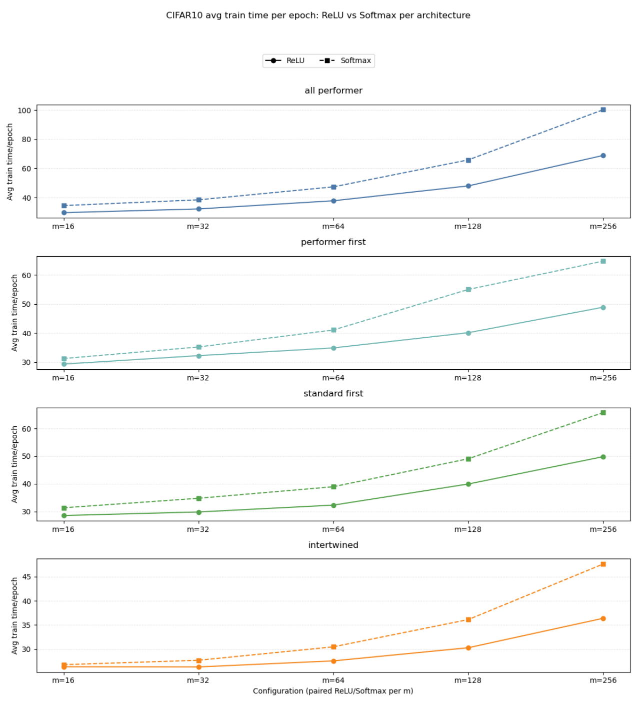
We think this happened because of the strong hardware- and kernel-level optimizations on the T4 GPU, which can mask architectural and algorithmic differences at the scale we're working with, leading to timing variability dominated by system-level effects rather than actual model complexity. These measurements weren't suitable for a meaningful efficiency comparison.

For this reason, we reran all timing experiments locally on a MacBook Pro with an Apple M3 chip, using the Metal Performance Shaders (MPS) backend. This setup gave us more stable and interpretable timing behavior, where architectural, kernel, and feature-dimension effects were clearly visible. All the results below are based exclusively on these local measurements.

Now we compare the computational efficiency of the different hybrid Performer–Attention architectures, focusing on both training time per epoch and inference time per batch (**Figures 6** and **7**). Unlike accuracy, efficiency metrics show highly consistent and interpretable trends across datasets, kernels, and feature dimensions.



(a) MNIST



(b) CIFAR-10

Figure 6: Average training time per epoch as a function of the feature dimension m, comparing ReLU and softmax Performer kernels across hybrid architectures. Reproducible with notebook `figure6.ipynb`

Training time:

Figure 6 shows the average training time per epoch on MNIST and CIFAR-10. Two systematic patterns emerge across all hybrid architectures. First, models using the softmax Performer kernel are consistently slower to train than their ReLU counterparts (**Figures 6a** and **6b**). This gap appears for all values of m and reflects the higher computational overhead of the softmax approximation, which requires additional feature transformations and normalization steps.

Second, training time increases monotonically with m for all architectures and kernels. Larger feature dimensions directly increase the size of intermediate projected representations, leading to higher computational and memory costs during both forward and backward passes. While this scaling is predictable, its impact on accuracy is not monotonic and depends on the dataset, architecture, and kernel choice, as we discussed earlier.

Finally, for identical configurations, training is uniformly faster on MNIST than on CIFAR-10 (**Figures 6a** and **6b**). This difference comes from dataset complexity, CIFAR-10 involves higher-dimensional inputs and more expensive feature extraction, resulting in higher per-epoch computational cost across all architectures.

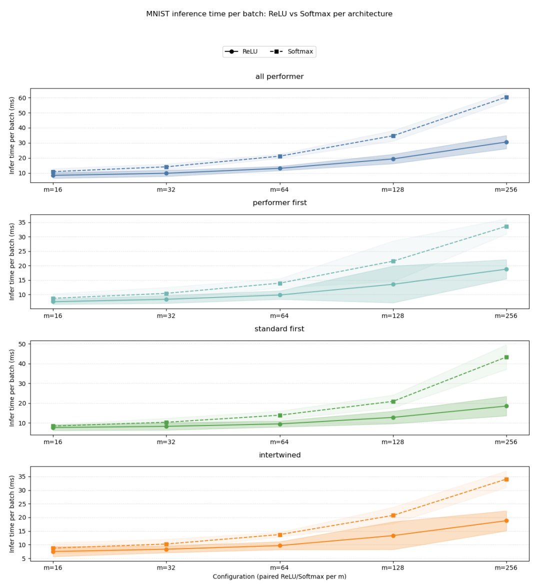
Inference time:

Inference-time trends, shown in **Figure 7**, closely mirror what we saw during training. ReLU-based Performer kernels consistently give lower inference latency than softmax approximations (**Figures 7a** and **7b**), and inference time increases steadily with m. These trends reflect the same underlying computational structure as training, with inference being just a forward-only pass.

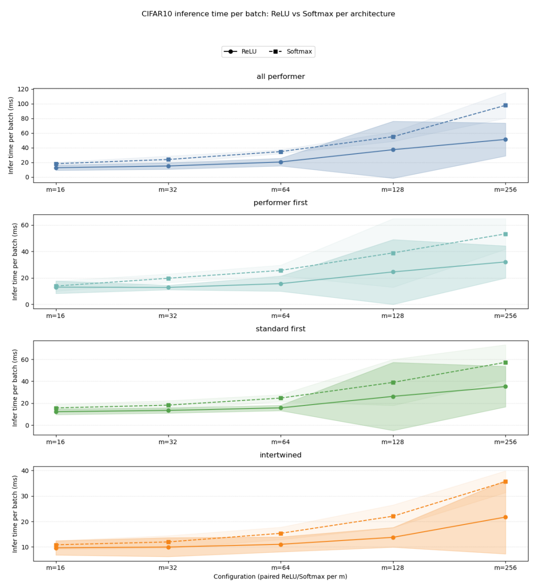
One notable difference is the variability of inference time. Compared to MNIST, CIFAR-10 shows substantially larger variance for high feature dimensions ($m \geq 128$), regardless of architecture (**Figure 7b**). This increased variability is consistent with higher memory pressure and hardware-level effects from larger intermediate tensors on a more demanding dataset. MNIST configurations stay smoother and more stable across the full range of m (**Figure 7a**).

Summary:

Overall, efficiency trade-offs among hybrid architectures are clear and robust (**Figures 6** and **7**). ReLU kernels are consistently faster than softmax approximations, larger feature dimensions increase both training and inference cost, and dataset complexity strongly affects absolute runtime and variability. Unlike efficiency, accuracy doesn't admit universal conclusions and must be evaluated together with architecture, kernel choice, and feature dimension. So while we can reliably predict computational cost, performance trade-offs need to be assessed case-by-case.



(a) MNIST



(b) CIFAR-10

Figure 7: Inference time per batch as a function of the feature dimension m, comparing ReLU and softmax Performer kernels across hybrid architectures. Reproducible with notebook `figure7.ipynb`

6.7 Do Performer-style speed benefits become clear at ImageNet-scale sequence lengths?

Having established that hybrid Performer–Attention ViTs can match or closely approach the accuracy of fully standard ViTs on both MNIST and CIFAR-10, we now turn to computational efficiency, the primary motivation behind Performer-style attention. Specifically, we compare average training and inference time per epoch between standard ViTs and hybrid architectures to see whether the theoretical reduction in attention complexity translates into practical wall-clock speedups.

On MNIST, the standard ViT consistently achieves the lowest training and inference time. This makes sense, the sequence lengths are small and exact attention is cheap, while Performer-style approximations introduce extra overhead from kernel feature projections that can't be amortized at this scale.

On CIFAR-10, we observe that hybrid models can sometimes achieve slightly lower training or inference time than standard ViT, but only for small feature dimensions m and when using the ReLU kernel. Even then, the gains are modest. For larger m or when using the softmax kernel, hybrid models are generally comparable to or slower than the baseline. These results show that in our experimental regime, the theoretical computational advantages of linearized attention are largely offset by practical implementation costs and the presence of exact-attention blocks in hybrid architectures.

Overall, these findings suggest that the wall-clock speed benefits of Performer-style attention are limited on small to medium-scale vision benchmarks, and are more likely to show up in settings with substantially longer sequences or more attention-dominated workloads.

7 Going further: Synthetic ImageNet-scale efficiency

The limited wall-clock speedups we observed on MNIST and CIFAR-10 naturally raise the question of whether Performer-style attention can deliver on its promised efficiency gains in a more demanding setting. The computational advantages of linearized attention should show up when attention operations dominate the overall cost, specifically, with substantially longer token sequences and higher-resolution inputs. To explore this regime, we turn to ImageNet-scale inputs.

Why synthetic ImageNet?

Initially, we wanted to benchmark training and inference efficiency on the full ImageNet dataset [**4**]. However, running large-scale ImageNet experiments requires significant computational resources and storage. Practical constraints around dataset access, download size, and runtime availability made these experiments infeasible for us. More importantly, for measuring computational efficiency specifically, the semantic content of the images doesn't actually matter.

Instead, we use *synthetic ImageNet-style inputs*, a common approach in efficiency studies. Since the computational cost of Vision Transformers is determined primarily by input resolution, patch size, and the resulting token sequence length, randomly generated inputs with the same spatial dimensions as ImageNet images let us faithfully reproduce the attention and projection workloads encountered in real ImageNet training. This eliminates confounding factors like disk I/O, image decoding, and data augmentation while preserving what we care about, the computational characteristics.

Experimental setup:
We generate synthetic batches of images $x \in \mathbb{R}^{B\times3\times224\times224}$, matching the resolution and channel dimensions of ImageNet inputs, with i.i.d. entries drawn from a standard normal distribution, along with randomly sampled labels $y \in \{1,\ldots,1000\}^B$. We tokenize images using a patch size of $p = 8$, resulting in $N = (224/8)^2 = 784$ tokens per image. This setup reproduces the patch embedding, sequence length, and attention workload encountered in ImageNet-scale ViT training while abstracting away image semantics. All models use identical hyperparameters, batch sizes, and optimization settings, differing only in their attention architecture.

We run all experiments on an **NVIDIA T4 GPU** using Google Colab. To ensure fair timing comparisons, we report average training time per epoch and average inference time per batch after a short warm-up period, focusing exclusively on wall-clock efficiency rather than convergence or accuracy. All results in this section are reproducible using the notebook `imagenet_synthetic_efficiency.ipynb`.

Models compared:
To isolate the effect of architectural design on computational efficiency, we fix the number of kernel features to $m = 128$ across all Performer-based models. This reflects a moderate regime that's neither too small nor excessively large. Fixing m lets us make controlled comparisons where differences in runtime come primarily from the attention layout and kernel type, rather than from variations in approximation dimension.

Table 3: Training and inference efficiency on synthetic ImageNet-style inputs (224×224). All timings are measured on a single NVIDIA T4 GPU. Reproducible with notebook `imagenet_synthetic_efficiency.ipynb`

Model	Kernel	Train Time / Epoch (s)	Infer Time (ms/batch)
Baseline ViT	N/A	136.3	302.16
PerformerFirst	ReLU	116.5	252.2
	Softmax	124.0	272.4
StandardFirst	ReLU	118.7	254.7
	Softmax	125.2	274.3
Intertwined	ReLU	117.4	252.4
	Softmax	124.1	273.4
AllPerformer	ReLU	**89.4**	**196.4**
	Softmax	105.3	242.5

Results and discussion:
Table 3 shows training and inference efficiency for standard and hybrid ViT architectures on synthetic ImageNet-scale inputs. Unlike our small-scale benchmarks, clear speed differences emerge in this more attention-dominated regime. The fully standard ViT baseline has the highest training and inference cost, reflecting the quadratic complexity of exact self-attention at a sequence length of N=784.

All hybrid architectures achieve consistent speedups over the baseline, confirming that replacing some exact-attention blocks with Performer-style linearized attention gives tangible wall-clock benefits at ImageNet-scale resolutions.

The biggest gains come from the *AllPerformer* architecture, which replaces all attention layers with linearized attention. Here, training time per epoch drops by approximately 35% compared to standard ViT, with a similar reduction in inference latency. This aligns with theoretical expectations—eliminating all quadratic attention operations lets the linear complexity of Performer-style attention dominate.

Across all architectures, ReLU-based kernels consistently outperform softmax-based kernels in terms of efficiency, both during training and inference. This gap reflects the extra computational overhead of the softmax approximation, reinforcing our earlier observation that ReLU kernels offer a better efficiency–stability trade-off when approximation quality is sufficient.

Overall, these results show that while Performer-style attention offers limited practical speedups on small-scale vision benchmarks, its advantages become pronounced at ImageNet-scale sequence lengths, particularly when a large fraction—or all—of the attention layers are linearized. This supports the view that hybrid and fully linearized attention mechanisms are most beneficial in regimes where attention computation dominates the overall model cost.

Scope and limitations:
We want to emphasize that these synthetic ImageNet experiments are designed solely to characterize computational efficiency and scaling behavior. They're not intended to assess classification performance or generalization on ImageNet. Still, this setup provides a controlled and informative benchmark to evaluate whether the theoretical advantages of Performer-style attention translate into practical speedups at realistic vision resolutions, and our results clearly show that they do.

8 Conclusion

In this report, we studied whether *hybrid* Vision Transformers that mix exact self-attention and Performer-style linearized attention can offer a favorable accuracy–efficiency trade-off compared to a fully standard ViT. Concretely, we implemented and evaluated five architectural strategies, *AllStandard* (baseline), *AllPerformer*, *Intertwined*, *PerformerFirst*, and *StandardFirst*, and compared two Performer kernel variants (ReLU-based and softmax approximation) across a range of random feature dimensions m. We conducted controlled experiments on MNIST and CIFAR-10, analyzed accuracy distributions, kernel effects, dependence on m, learning dynamics, and measured training and inference runtime. Finally, to probe an attention-dominated regime, we extended the efficiency analysis to synthetic ImageNet-scale inputs with longer sequences.

Across both MNIST and CIFAR-10, hybrid Performer–Attention ViTs can *match or closely approach* the accuracy of a fully standard ViT, and in several configurations even slightly exceed it. However, the variance across configurations is larger for hybrid models, especially on CIFAR-10, indicating that hybrid designs can be more sensitive to kernel choice and hyperparameters than a fully standard ViT. Among hybrid architectures, *PerformerFirst* emerged as the most consistently strong design: placing Performer layers early and retaining exact attention in later layers tended to preserve accuracy while remaining robust across kernels and feature dimensions. In contrast, *StandardFirst* more frequently underperformed, suggesting that introducing approximation only in later layers can be more detrimental.

We also found that the impact of the Performer kernel depends strongly on where it is used. ReLU kernels are generally more robust (and later shown to be more efficient), while softmax-based kernels can offer accuracy benefits in certain regimes (notably early placement on CIFAR-10), at the cost of higher runtime and occasionally slower early optimization. Importantly, accuracy depends only weakly on the number of kernel features once m reaches moderate values: performance quickly saturates and exhibits diminishing returns for larger m. This indicates that competitive accuracy does not require very large random feature expansions on these datasets.

Beyond final accuracy, learning-curve analysis revealed a consistent optimization pattern: hybrid models tend to learn more slowly during early training, particularly on CIFAR-10, but remain stable and largely close the gap by later epochs. Training curves mirror this behavior, suggesting the early lag is primarily an *optimization* effect (slower fitting) rather than an immediate generalization issue. Fully linearized models (*AllPerformer*) show the strongest early slowdown, while hybrids that retain more exact-attention blocks track the baseline more closely.

On MNIST and CIFAR-10, wall-clock speedups from Performer-style attention were limited. The standard ViT was often fastest on MNIST, and on CIFAR-10 hybrids only occasionally achieved modest gains for small m with ReLU kernels. This supports the practical conclusion that at short sequence lengths, overheads from random feature projections (and the presence of remaining exact-attention blocks in hybrids) can offset the theoretical asymptotic advantage of linear attention.

However, when moving to an attention-dominated regime using synthetic ImageNet-scale inputs (224×224, patch size $p = 8$, $N = 784$ tokens), the expected efficiency benefits became clear. All hybrid architectures outperformed the fully standard ViT in both training time per epoch and inference latency, and *AllPerformer* achieved the largest speedups (roughly 35% faster than the baseline in our setting). Across all architectures, ReLU-based kernels were consistently more efficient than softmax approximations, highlighting a clear runtime advantage of the simpler kernel in long-sequence regimes.

Overall, our results show that hybrid Performer–Attention ViTs provide a practical path to reducing attention cost without sacrificing much accuracy on small vision benchmarks, and that the benefits of linearized attention become substantially more pronounced as sequence length grows. Crucially, *where* approximate attention is placed matters as much as *how* it is approximated: strategically mixing Performer and exact attention can preserve accuracy while improving efficiency in regimes where attention dominates.

This study opens several directions for extending and strengthening the conclusions. First, evaluating on more challenging and larger-scale vision tasks (e.g., full ImageNet, fine-grained datasets, detection/segmentation) would better test whether hybrid designs retain accuracy as the task demands richer global interactions. Second, a broader sweep over sequence length (e.g., varying patch size p and image resolution) would more directly characterize the crossover point where Performers transition from being overhead-dominated to advantage-dominated. Third, systematic tuning of m and kernel choice per layer (rather than globally) could yield stronger trade-offs, potentially using adaptive or learned feature dimensions. Finally, exploring implementation-level optimizations (fused kernels, FlashAttention-style baselines, and more optimized linear-attention kernels) would provide a more realistic picture of practical speedups on modern hardware, and help identify when Performer-style attention is most competitive in real training pipelines.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. *International Conference on Learning Representations (ICLR)*, 2022.

[3] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers. In *International Conference on Learning Representations (ICLR)*, 2020.

[4] 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 (CVPR)*, 2009.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

[6] 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 (ICLR)*, 2021.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[8] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[9] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning (ICML)*, 2020.

[10] Alex Krizhevsky and Geoffrey E. Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[12] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[13] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[14] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2007.

[15] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 2020.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[17] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

[18] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. In *International Conference on Machine Learning (ICML)*, 2020.

[19] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.