IEOR E6617 - Machine Learning and High-Dimensional Analysis in Operations Research

Columbia University - IEOR

Hybrid Vision Transformer Model

Fall 2025

Authors:
Chiara von Gerlach
Edward Lucyszyn
Hannah Tollié

Contents

1 Introduction

Transformers have become a central architecture for sequence modeling since self-attention was introduced as the main mechanism to mix information across tokens [1]. Their success largely comes from the ability of self-attention to form content-dependent interactions between all pairs of tokens, while remaining highly parallelizable. In the standard scaled dot-product attention block, a sequence of length L with token dimension d is first projected into queries, keys, and values, which costs on the order of $O(Ld^2)$ operations. The main computational bottleneck then arises when attention is computed between all token pairs: forming the score matrix for all query–key pairs and applying it to the values scales as $O(L^2d)$. In addition, explicitly materializing attention weights incurs $O(L^2)$ memory. As a result, when L grows—for instance with longer contexts in language or finer patch tokenization in vision—the $O(L^2d)$ time and $O(L^2)$ memory terms become dominant and quickly limit scalability.

These constraints have motivated a broad literature on efficient attention mechanisms designed to reduce the dependence on L. One prominent direction imposes sparsity or locality so that each token attends to only a subset of keys, reducing the attention cost from quadratic to roughly $O(Lsd)$ when each query attends to $s \ll L$ tokens, with corresponding memory savings. Another direction leverages structured approximations. Reformer, for example, uses locality-sensitive hashing to group similar tokens and restrict attention computations within buckets, leading to near-linear behavior in practice under appropriate assumptions [2]. Linformer, in contrast, assumes that the attention matrix is approximately low-rank and projects keys and values to a smaller dimension $k \ll L$, reducing attention computation to approximately $O(Lkd)$ [3]. Across these approaches, the common goal is to preserve the representational benefits of attention while avoiding the explicit $L \times L$ interaction pattern.

Performers provide a particularly general approach by reinterpreting softmax attention through a kernel perspective and then approximating that kernel using randomized feature maps [4]. This idea is closely related to random features methods for large-scale kernel machines [5], where inner products in a higher-dimensional feature space approximate a target kernel. In the Performer framework, the softmax similarity is approximated by a dot product between explicit random features of queries and keys. This enables an associative reordering of computations so that attention can be evaluated without constructing the dense attention matrix. The resulting attention computation scales linearly with the sequence length, with time complexity on the order of $O(Lmd)$ and memory on the order of $O(Lm)$, where m is the number of random features controlling the approximation quality. FAVOR+ (Fast Attention Via positive Orthogonal Random features) is the Performer construction that yields a stable, positive random-feature estimator tailored to the softmax kernel [4]. In practice, alternative feature maps can also be used to define other kernelized (generalized) linear attention variants, trading faithfulness to softmax for simplicity and potential numerical robustness.

In this project, we study Performer-style attention in a vision-transformer setting, where images are tokenized into sequences of patches and the effective sequence length L can grow rapidly with resolution [6].We first present the general principle of kernelized linear attention and contrast two practical modes: FAVOR+, which targets softmax attention via random features, and a ReLU-based feature map that induces an alternative kernelized attention mechanism. We also provide a proof for the induced ReLU kernel form and derive the variance of its random-feature estimator. We then detail the implementation choices in our codebase, including the architecture, configuration-driven experimentation, and training pipeline. Finally, we report empirical results on standard image-classification benchmarks (MNIST [7] and CIFAR-10 [8]), focusing on the trade-offs among accuracy, training stability, and the computational gains obtained by replacing the quadratic attention bottleneck, $O(L^2d)$, with linear-in-L approximations.

2 Theoretical Preliminaries

For the whole section, let $\boldsymbol{X} \in \mathbb{R}^{L\times d}$ denote a sequence of L tokens with model dimension d. This section gathers the main preliminaries needed to interpret our implementation: we first recall the kernel view of self-attention and its linearization, then we compare the two randomized kernels used in practice (softmax via FAVOR+ and ReLU), including closed-form and variance results for the ReLU case (linked to the random-feature analysis of Problem 1 of the list of problems, with proofs deferred to the Appendix), and we finally summarize the role of the MLP hyperparameters and the depth (number of layers) on capacity and complexity.

2.1 Regular Transformer Self-Attention and Performers

A Transformer block typically combines multi-head self-attention with a position-wise MLP, each equipped with residual connections and LayerNorm, as introduced in [1]. We focus here on the self-attention mechanism.

Given an input sequence $\boldsymbol{X} \in \mathbb{R}^{L\times d}$, an attention layer first forms the query, key, and value matrices through linear projections:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V, \qquad \boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{d\times d}. \tag{1}$$

The attention scores are then obtained by applying a kernel to all query–key pairs. The entries $(a_{i,j})$ of the attention matrix for a softmax kernel are given by:

$$\forall (i,j) \in [\![1, L]\!], \quad a_{i,j} = k(\boldsymbol{q}_i, \boldsymbol{k}_j) = \exp\left(\frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{d}}\right), \tag{2}$$

where $\boldsymbol{q}_i^\top$ (resp. $\boldsymbol{k}_j^\top$) denotes the i-th (resp. j-th) row of $\boldsymbol{Q}$ (resp. $\boldsymbol{K}$). In practice, $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ are split into h heads, attention is computed independently within each head, and the results are concatenated and projected with $\boldsymbol{W}_O \in \mathbb{R}^{d\times d}$. This procedure requires $O(L^2 d)$ time and $O(L^2)$ memory due to the explicit construction of an $L \times L$ attention matrix.

A standard approach to obtain linear-time attention is to approximate the pairwise similarity by a kernel that factorizes through a feature map. Let $m \in \mathbb{N}$ and $\phi : \mathbb{R}^d \to \mathbb{R}^m$ be a feature map such that:

$$\forall \boldsymbol{q}, \boldsymbol{k} \in \mathbb{R}^d, \quad k(\boldsymbol{q}, \boldsymbol{k}) = \mathbb{E}\Big[\phi(\boldsymbol{q})^\top \phi(\boldsymbol{k})\Big]. \tag{3}$$

When attention weights are constructed from such a kernel, the attention output can be computed using associativity, without explicitly forming the $L \times L$ matrix:

$$\Big[\phi(\boldsymbol{Q})\phi(\boldsymbol{K})^\top\Big]\boldsymbol{V} \;=\; \phi(\boldsymbol{Q})\Big(\phi(\boldsymbol{K})^\top \boldsymbol{V}\Big). \tag{4}$$

The main computations are then $\phi(\boldsymbol{K})^\top \boldsymbol{V}$ followed by multiplication with $\phi(\boldsymbol{Q})$, which together scale as $O(Lmd)$ in time, with $O(Lm)$ memory to store the feature representations.

2.2 Softmax and ReLU Comparison

We compare two randomized feature maps used to linearize attention: (i) FAVOR+ features that approximate the exponential (softmax) kernel, and (ii) random ReLU features that induce an arc-cosine–type kernel. Throughout, we precise $(\cdot)_+ = \max(0, \cdot)$.

2.2.1 Softmax Kernel and FAVOR+: PRF Feature Map and MSE

The (rescaled) softmax kernel can be written as

$$\mathrm{SM}(\boldsymbol{x}, \boldsymbol{y}) = \exp(\boldsymbol{x}^\top \boldsymbol{y}), \tag{5}$$

and corresponds to $k_{\text{soft}}(\boldsymbol{q},\boldsymbol{k}) = \exp(\boldsymbol{q}^\top \boldsymbol{k}/\sqrt{d})$ after the change $\boldsymbol{x} = d^{-1/4}\boldsymbol{q}$, $\boldsymbol{y} = d^{-1/4}\boldsymbol{k}$.

Let $\boldsymbol{\omega}_1,\dots,\boldsymbol{\omega}_m \overset{\text{iid}}{\sim} \mathcal{N}(\boldsymbol{0},\boldsymbol{I}_d)$ and define the positive random feature map (PRF)

$$\phi_m^+(\boldsymbol{x}) = \frac{1}{\sqrt{m}}\exp\left(-\frac{\|\boldsymbol{x}\|^2}{2}\right)\big(\exp(\boldsymbol{\omega}_1^\top \boldsymbol{x}),\dots,\exp(\boldsymbol{\omega}_m^\top \boldsymbol{x})\big)^\top. \tag{6}$$

The corresponding kernel estimator is

$$\widehat{\text{SM}}_m^+(\boldsymbol{x},\boldsymbol{y}) = \phi_m^+(\boldsymbol{x})^\top \phi_m^+(\boldsymbol{y}). \tag{7}$$

Result 1 (PRF estimator for softmax and mean-squared error)**.** *For any $\boldsymbol{x},\boldsymbol{y} \in \mathbb{R}^d$, with $\boldsymbol{z} = \boldsymbol{x}+\boldsymbol{y}$, the PRF estimator satisfies*

$$\mathbb{E}\left[\widehat{\text{SM}}_m^+(\boldsymbol{x},\boldsymbol{y})\right] = \text{SM}(\boldsymbol{x},\boldsymbol{y}), \tag{8}$$

and its mean-squared error (equal to its variance) is

$$\text{MSE}\Big(\widehat{\text{SM}}_m^+(\boldsymbol{x},\boldsymbol{y})\Big) = \text{Var}\Big(\widehat{\text{SM}}_m^+(\boldsymbol{x},\boldsymbol{y})\Big) = \frac{1}{m}\exp(\|\boldsymbol{z}\|^2)\,\text{SM}(\boldsymbol{x},\boldsymbol{y})^2\Big(1-\exp(-\|\boldsymbol{z}\|^2)\Big). \tag{9}$$

Proof. All these properties and proofs are from [4]. □

2.2.2 Random ReLU Features: Closed Form and Variance

We now analyze the ReLU feature map used in our implementation. Let $\boldsymbol{g}_1,\dots,\boldsymbol{g}_m \overset{\text{iid}}{\sim} \mathcal{N}(\boldsymbol{0},\boldsymbol{I}_d)$ and define:

$$\phi_m^{\text{relu}}(\boldsymbol{x}) = \frac{1}{\sqrt{m}}\big((\boldsymbol{g}_1^\top \boldsymbol{x})_+,\,\dots,\,(\boldsymbol{g}_m^\top \boldsymbol{x})_+\big)^\top, \qquad \widehat{k}_m^{\text{relu}}(\boldsymbol{q},\boldsymbol{k}) = \phi_m^{\text{relu}}(\boldsymbol{q})^\top \phi_m^{\text{relu}}(\boldsymbol{k}). \tag{10}$$

Let $\theta \in [0,\pi]$ denote the angle between $\boldsymbol{q}$ and $\boldsymbol{k}$:

$$\cos\theta = \frac{\boldsymbol{q}^\top \boldsymbol{k}}{\|\boldsymbol{q}\|\,\|\boldsymbol{k}\|}. \tag{11}$$

Define also the population kernel

$$k_{\text{relu}}(\boldsymbol{q},\boldsymbol{k}) = \mathbb{E}_{\boldsymbol{g}\sim\mathcal{N}(\boldsymbol{0},\boldsymbol{I}_d)}\big[(\boldsymbol{g}^\top \boldsymbol{q})_+\,(\boldsymbol{g}^\top \boldsymbol{k})_+\big]. \tag{12}$$

Result 2 (Closed form for k_{relu})**.** *For any $\boldsymbol{q},\boldsymbol{k} \in \mathbb{R}^d$ with angle $\theta \in [0,\pi]$,*

$$k_{\text{relu}}(\boldsymbol{q},\boldsymbol{k}) = \frac{\|\boldsymbol{q}\|\,\|\boldsymbol{k}\|}{2\pi}\Big(\sin\theta + (\pi-\theta)\cos\theta\Big). \tag{13}$$

Equivalently, $\mathbb{E}\big[\widehat{k}_m^{\text{relu}}(\boldsymbol{q},\boldsymbol{k})\big] = k_{\text{relu}}(\boldsymbol{q},\boldsymbol{k})$.

The proof is deferred to Appendix 1.1.

Result 3 (Variance of the kernel estimator)**.** *Let $\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{0},\boldsymbol{I}_d)$ and set $a = \boldsymbol{g}^\top \boldsymbol{q}$, $b = \boldsymbol{g}^\top \boldsymbol{k}$. Define*

$$E_2(\theta) = \sin\theta + (\pi-\theta)\cos\theta, \qquad E_1(\theta) = 2(\pi-\theta) + (\pi-\theta)\cos(2\theta) + \frac{3}{2}\sin(2\theta). \tag{14}$$

Then

$$\text{Var}\big(a_+b_+\big) = \frac{\|\boldsymbol{q}\|^2\|\boldsymbol{k}\|^2}{2\pi}\,E_1(\theta) - \frac{\|\boldsymbol{q}\|^2\|\boldsymbol{k}\|^2}{4\pi^2}\,E_2(\theta)^2, \tag{15}$$

and, for the Monte Carlo estimator $\widehat{k}_m^{\mathrm{relu}}(\boldsymbol{q},\boldsymbol{k}) = \frac{1}{m}\sum_{i=1}^{m}(a_i)_+(b_i)_+$,

$$\mathrm{Var}\left(\widehat{k}_m^{\mathrm{relu}}(\boldsymbol{q},\boldsymbol{k})\right) = \frac{\|\boldsymbol{q}\|^2\|\boldsymbol{k}\|^2}{m}\left(\frac{1}{2\pi}\,E_1(\theta) - \frac{1}{4\pi^2}\,E_2(\theta)^2\right). \tag{16}$$

Moreover,

$$\min_{\theta\in[0,\pi]} \mathrm{Var}\left(\widehat{k}_m^{\mathrm{relu}}(\boldsymbol{q},\boldsymbol{k})\right) = 0 \quad \textit{(attained at } \theta=\pi\textit{)}, \qquad \max_{\theta\in[0,\pi]} \mathrm{Var}\Big(\widehat{k}_m^{\mathrm{relu}}(\boldsymbol{q},\boldsymbol{k})\Big) = \frac{5}{4}\frac{\|\boldsymbol{q}\|^2\|\boldsymbol{k}\|^2}{m}. \tag{17}$$

In particular, at $\theta = \pi/2$,

$$\mathrm{Var}\Big(\widehat{k}_m^{\mathrm{relu}}(\boldsymbol{q},\boldsymbol{k})\Big) = \frac{\|\boldsymbol{q}\|^2\|\boldsymbol{k}\|^2}{4m}\left(1 - \frac{1}{\pi^2}\right). \tag{18}$$

The proof is deferred to Appendix 1.2.

2.2.3 Practical Takeaway: Optimization and Variance Trade-offs

The above results highlight a trade-off when choosing the kernel.

With FAVOR+ (softmax), the exponential kernel strongly amplifies differences in $\boldsymbol{q}^\top\boldsymbol{k}$, which typically produces more selective attention patterns. This can help optimization when the task benefits from sharp key selection, since gradients concentrate quickly on a few relevant tokens. However, this same amplification makes the random-feature approximation more sensitive to large dot-products, which can increase estimator variance and lead to numerical instabilities unless variance-reduction and stabilization mechanisms are effective.

With random ReLU features, the induced arc-cosine kernel depends smoothly on the angle θ, producing less extreme attention scores. The Monte Carlo variance is explicit and scales as $O(\|\boldsymbol{q}\|^2\|\boldsymbol{k}\|^2/m)$, with worst-case variance for nearly aligned vectors and vanishing variance at $\theta = \pi$. This often translates into more stable attention weights and gradients, but weaker "winner-take-all" behavior, which may slow convergence or reduce peak performance when strong selectivity is required.

2.3 MLP Hyperparameters and Depth

A Transformer encoder block contains, besides attention, a position-wise MLP applied independently to each token. For a token representation $\boldsymbol{x}\in\mathbb{R}^d$, the MLP typically has the form

$$\mathrm{MLP}(\boldsymbol{x}) = \boldsymbol{W}_2\,\sigma(\boldsymbol{W}_1\boldsymbol{x}+\boldsymbol{b}_1)+\boldsymbol{b}_2, \qquad \boldsymbol{W}_1\in\mathbb{R}^{d_{\mathrm{ff}}\times d},\ \boldsymbol{W}_2\in\mathbb{R}^{d\times d_{\mathrm{ff}}}, \tag{19}$$

where σ is an activation (often GELU or ReLU) and d_{ff} is the hidden width of the MLP. In many implementations one sets

$$d_{\mathrm{ff}} = r\,d, \tag{20}$$

with an expansion ratio r. The MLP parameter count per layer is dominated by

$$\#\mathrm{params(MLP)} \approx 2\,d\,d_{\mathrm{ff}} \quad \text{(plus biases)}, \tag{21}$$

so increasing d_{ff} directly increases model capacity and compute, but also tends to make optimization more sensitive (more parameters, stronger risk of overfitting without sufficient regularization).

Depth is controlled by the number of stacked encoder blocks N. To first order, both parameters and compute scale linearly in N. For example, ignoring biases and LayerNorm parameters, one layer has roughly

$$\#\mathrm{params(layer)} \approx 4d^2 \ + \ 2d\,d_{\mathrm{ff}}, \tag{22}$$

where $4d^2$ comes from the attention projections ($\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V, \boldsymbol{W}_O$) and $2d\,d_{\text{ff}}$ from the MLP. Hence

$$\#\text{params(total)} \approx N\,(4d^2 + 2d\,d_{\text{ff}}). \tag{23}$$

Practically, increasing N improves expressivity by composing more attention/MLP transformations, but requires more optimization or stronger regularization settings. In our experiments, N and d_{ff} are therefore the architectural knobs (together with the kernel choice and the feature dimension m) that determine the capacity/compute trade-off observed in training curves.

3 Implementation

This section explains the code-level pipeline used to produce our results, from dataset preparation to the artifacts saved during training and post-training analysis. The goal is to make it easy to (i) reproduce the experiments by editing a configuration file, and (ii) understand which component is responsible for which part of the training dynamics and reported metrics.

3.1 Data Pipeline: CIFAR-10 and MNIST Preparation

Our data pipeline is implemented in three files: `data/dataloaders.py`, `data/datasets.py`, and `data/transforms.py`. It provides a unified interface that returns `train`/`val`/`test` dataloaders together with basic metadata, regardless of the dataset.

3.1.1 Datasets and Preprocessing

MNIST consists of 28×28 grayscale digit images (10 classes), while CIFAR-10 consists of 32×32 RGB natural images (10 classes). Although the two datasets differ in content and channel format, we enforce a common model input format (3 channels) so that all architectures can be reused across datasets without changing the first embedding layer.

The preprocessing returns two transformation pipelines: one for training and one for evaluation (validation/test). For CIFAR-10:

- Training transform: random horizontal flip + random crop of size `img_size` with padding 4, then `ToTensor` and per-channel normalization using fixed CIFAR-10 statistics.
- Evaluation transform: `ToTensor` and the same normalization (no stochastic augmentation).

For MNIST:

- Both training and evaluation transforms resize to `img_size`, convert to tensor, then convert to 3 channels via `To3Channels` (channel repetition), followed by normalization using MNIST statistics replicated on all three channels.

The boolean flag `augment` controls whether the training transform includes stochastic augmentations. When `augment` is `false`, the dataset training transform is replaced by the evaluation transform (i.e., no random crop/flip).

3.1.2 Reproducible Splits and Loadings

Both datasets are split into train/validation in a reproducible way using `random_split` with a fixed generator seed: `g = torch.Generator().manual_seed(seed)`. The function `build_dataloaders(cfg)` also returns `train_loader`, `val_loader`, `test_loader`, and a small `meta` dictionary. Reproducibility is reinforced as well via:

- A fixed split seed (`seed`) used in `datasets.py`.

- A worker initialization function `_worker_init_fn` which reseeds Python and NumPy RNGs for each DataLoader worker based on `torch.initial_seed()`. This prevents correlated randomness across workers, which is important when using stochastic data augmentation.

3.1.3 Tokenization and Sequence Length

Images are converted into patch tokens inside the model (patch embedding). The patch size is set by `dataset.patch` in each configuration file (see 3.3 for further details). With patch size p and image size $H \times W$, the number of tokens is:

$$N = \left(\frac{H}{p}\right)\left(\frac{W}{p}\right). \tag{24}$$

In our default experiments, we used `patch`=4. This yields $N = 49$ tokens for MNIST ($28/4 = 7$) and $N = 64$ tokens for CIFAR-10 ($32/4 = 8$). While these values are smaller than typical ViT setups on high-resolution images, they still allow us to compare regular attention to Performer-style attention under controlled conditions. In the additional CIFAR-10 high setting, we decreased `patch` to 2 to increase N and move closer to a regime where quadratic attention becomes more costly. Further details are provided in Section 4.

3.2 Model Implementation and Training Loop Overview

All experiments are driven by YAML configuration files. For a single training entry point (which is the function that trains from a config), the following steps are performed:

1. Load the YAML file (model + dataset + optimization + misc).
2. Build dataloaders by calling `build_dataloaders(...)`.
3. Instantiate the model according to `model.layers` and performer options.
4. Train for `optim.epochs` epochs with a fixed optimization protocol (AdamW, `lr`, `weight_decay`, `batch_size`).
5. At each epoch: compute train metrics, val metrics, and save a row into `metrics.csv`.
6. Save the final model checkpoint and auxiliary artifacts in `runs/<cfg_name>/`.

For each configuration file, the key architectural mechanism is the `model.layers` field: a list encoding the attention type at each depth position. For example:

`["Reg","Reg","Reg"]` or `["Perf","Reg","Perf","Reg"]`.

The model builder iterates through that list and instantiates the corresponding block class:

- `Reg` layers use exact attention.
- `Perf` layers use Performer-style attention with random features.

This design makes it possible to test Full Reg, Full Perf, intertwined hybrids, and staged hybrids with minimal code changes, where the only change needed is to alter the list specified in the config.

3.3 Configuration Files

Each run is controlled by a dedicated `.yaml` file. The main fields are:

- `dataset`: `name`, `root`, `img_size`, `patch`, and `augment`.
- `model`: architectural parameters such as `d_model`, `heads`, `mlp_ratio`, and crucially `layers`.
- `model.performer`: Performer-specific parameters such as `m` (number of random features) and `variant` (`relu` or `softmax`) for Performer layers.
- `optim`: training hyperparameters such as `batch_size`, `lr`, `weight_decay`, and number of epochs.
- `misc`: reproducibility and logging options (e.g. `seed`).

This separation allows us to keep training protocol fixed across most experiments while varying only the architectural pattern (and, when needed, `m` or kernel type). This is very useful to launch a new model, since one model corresponds exactly to one file.

```yaml
experiment: cifar10_intertwined_reg_d=8_m=64_variant=softmax
dataset:
  name: cifar10
  root: ./data/cache
  img_size: 32
  patch: 4
  augment: true
model:
  d_model: 256
  heads: 4
  mlp_ratio: 2.0
  depth: 8
  layers: ["Reg", "Perf", "Reg", "Perf", "Reg", "Perf", "Reg", "Perf"]
  performer:
    variant: "softmax"
    kind: "favor+"
    m: 64
optim:
  batch_size: 128
  lr: 0.0003
  weight_decay: 0.05
  epochs: 80
log:
  out_dir: ./runs
  save_every: 5
misc:
  seed: 17092003
```

3.4 Results Synthesis and Visualization Code

Each training run writes its artifacts under:

`runs/<cfg_name>/`.

The most important file for the analysis scripts is `metrics.csv`, which stores per-epoch values such as `train_loss`, `train_acc`, `val_loss`, `val_acc`, and `epoch_time_sec`. This file is then read by the post-training analysis scripts to build:

- training/validation loss curves,

- training/validation accuracy curves,
- time per epoch curves,
- cumulative training time curves,
- summary tables (best accuracies/losses, best epoch, median epoch time, etc.).

The group analysis script (`models/model_analysis_groups.py`) writes its visualizations under:

`outputs/<group_name>/`,

where each folder contains `train_loss.png`, `val_loss.png`, `train_acc.png`, `val_acc.png`, `epoch_time.png`, `cumulative_time.png`, and a `summary.txt`. The results from these outputs are shown and discussed in further detail in Section 4 below.

4 Results

Unless stated otherwise, all simulations reported in this section were run with the following fixed settings:

- Model: `d_model` sets to 32× (number of layers), `heads`=4, `mlp_ratio`=2.0.
- Optimization: `batch_size`=128, `lr`=0.0003, `weight_decay`=0.05.

For CIFAR-10, we additionally used `img_size`=32 and `patch`=4, and we applied standard train-time augmentations consisting of a random horizontal flip and a random crop of size `img_size` with padding 4, followed by tensor conversion and per-channel normalization using the CIFAR-10 statistics. For MNIST, we used `img_size`=28 and `patch`=4, and we applied resizing to `img_size`, conversion to 3 channels by repeating the single channel, and normalization using the MNIST statistics (no random crop/flip).

Since the models were both numerous and expensive to train (especially for deeper configurations and larger feature maps), we ran each configuration only once due to limited computational resources, rather than repeating runs and averaging the results (which would have been more rigorous). In addition, throughout the analysis we often focus on validation accuracy, because it was logged automatically at every epoch, whereas test accuracy was evaluated less conveniently and, in our experiments, remained extremely close to validation accuracy anyway.

4.1 Overall Results

The results are summarized in Tables 1 and 2.

4.2 Influence of Model Architecture

We now discuss how the architectural choices encoded in the configs (regular attention vs Performer, hybrid placements, and depth) impact both performance and optimization.

4.2.1 Depth

We first isolate the effect of depth (number of attention layers) while keeping the other hyper-parameters fixed. From Table 1, increasing depth improves performance across architectures on MNIST: for instance, `Full Reg` rises from 96.60% (1 layer) to 99.08% (6 layers), and `Full Perf` also benefits from depth (e.g., ReLU kernel: 94.68% → 98.90% from 1 to 6 layers). From Table 2, the depth effect is even more pronounced on CIFAR-10: `Full Reg` improves from 58.58% (1 layer) to 82.28% (8 layers), while `Full Perf` with `m`=64 improves from ≈ 45–50% (1 layer) to

Table 1: MNIST summary (fixed hyperparameters). For each architecture setting and kernel, we report the number of attention layers, the number of random features `m` (for Performer-style attention), the best validation accuracy and its epoch, the best training accuracy, the minimum training/validation losses reached during training, the total number of epochs, and the median epoch time.

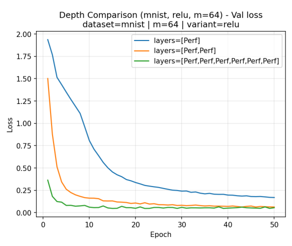
`setting`	`kernel`	#layers	`m`	best val acc (%)	best epoch	best train acc (%)	best train loss	best val loss	epochs	median time/epoch (s)
Full Reg	softmax	1	64	96.600	49	94.233	0.172	0.103	50	3.428
Full Reg	softmax	2	64	98.780	45	98.762	0.037	0.040	50	4.597
Full Reg	softmax	6	64	99.080	39	99.516	0.015	0.033	50	8.409
Full Perf	relu	1	64	94.680	49	91.432	0.265	0.169	50	3.610
Full Perf	softmax	1	64	95.860	50	93.806	0.196	0.129	50	22.728
Full Perf	relu	2	32	98.080	45	97.188	0.086	0.065	50	4.833
Full Perf	softmax	2	32	98.200	49	97.716	0.070	0.063	50	5.593
Full Perf	relu	2	64	98.140	43	97.494	0.078	0.062	50	4.889
Full Perf	softmax	2	64	98.080	50	97.502	0.078	0.066	50	5.602
Full Perf	relu	2	128	98.100	50	97.365	0.082	0.063	50	4.843
Full Perf	softmax	2	128	98.040	46	97.842	0.068	0.063	50	5.606
Full Perf	relu	2	1024	98.020	44	97.192	0.087	0.063	50	7.240
Full Perf	softmax	2	1024	98.360	44	97.736	0.071	0.066	50	8.836
Full Perf	relu	2	4096	97.880	49	97.044	0.092	0.074	50	23.843
Full Perf	softmax	2	4096	98.080	50	97.562	0.078	0.066	50	29.538
Full Perf	relu	6	64	98.900	49	99.716	0.009	0.046	50	9.335
Full Perf	softmax	6	64	98.600	41	99.479	0.016	0.050	50	11.804
Intertwined (start=Perf)	softmax	2	64	98.060	48	97.236	0.086	0.057	50	5.051
Intertwined (start=Perf)	softmax	6	64	98.900	28	99.574	0.012	0.044	50	9.893
Intertwined (start=Reg)	softmax	2	64	98.160	49	97.603	0.077	0.055	50	5.104
Intertwined (start=Reg)	softmax	6	64	98.880	38	99.578	0.012	0.043	50	9.922
Perf→Reg	softmax	6	64	98.940	31	99.552	0.013	0.041	50	10.215
Perf→Reg	softmax	2	64	98.140	50	97.316	0.083	0.057	50	5.159
Reg→Perf	softmax	2	64	98.000	47	97.483	0.079	0.063	50	5.198
Reg→Perf	softmax	6	64	99.020	49	99.647	0.011	0.038	50	10.147

Table 2: CIFAR-10 summary (fixed hyperparameters). For each architecture setting and kernel, we report the number of attention layers, the number of random features `m` (for Performer-style attention), the best validation accuracy and its epoch, the best training accuracy, the minimum training/validation losses reached during training, the total number of epochs, and the median epoch time.

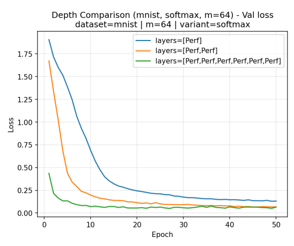
`setting`	`kernel`	#layers	`m`	best val acc (%)	best epoch	best train acc (%)	best train loss	best val loss	epochs	median time/epoch (s)
Full Reg	softmax	1	64	58.580	119	57.975	1.163	1.186	140	3.703
Full Reg	softmax	2	64	75.000	126	79.461	0.573	0.731	140	4.354
Full Reg	softmax	8	64	82.280	106	96.336	0.105	0.614	140	11.860
Full Perf	relu	1	64	49.620	79	46.172	1.465	1.397	80	3.815
Full Perf	softmax	1	64	44.920	78	42.232	1.536	1.471	80	3.902
Full Perf	relu	2	32	62.420	75	60.668	1.096	1.054	80	4.499
Full Perf	softmax	2	32	62.820	80	60.272	1.107	1.035	80	5.227
Full Perf	relu	2	64	62.520	79	60.637	1.091	1.047	80	4.518
Full Perf	softmax	2	64	62.140	79	59.346	1.124	1.068	80	5.108
Full Perf	relu	2	128	62.660	80	61.343	1.073	1.039	80	4.503
Full Perf	softmax	2	128	60.880	80	58.672	1.148	1.123	80	5.205
Full Perf	relu	2	1024	54.880	39	53.134	1.289	1.265	40	6.576
Full Perf	softmax	2	1024	54.500	40	51.487	1.331	1.288	40	8.150
Full Perf	relu	2	4096	54.300	40	52.580	1.302	1.268	40	21.266
Full Perf	softmax	2	4096	54.840	40	52.477	1.298	1.237	40	27.269
Full Perf	relu	8	64	79.080	66	92.831	0.203	0.690	80	12.618
Full Perf	softmax	8	64	78.460	70	90.897	0.250	0.681	80	14.543
Intertwined (start=Perf)	softmax	2	64	60.960	79	57.543	1.168	1.101	80	4.987
Intertwined (start=Reg)	softmax	2	64	63.120	80	60.201	1.105	1.014	80	4.715
Perf→Reg	softmax	2	64	60.980	79	57.692	1.172	1.086	80	4.800
Reg→Perf	softmax	2	64	62.760	76	59.738	1.119	1.042	80	4.899
Intertwined (start=Perf)	softmax	8	64	80.760	64	91.322	0.240	0.634	80	13.191
Intertwined (start=Reg)	softmax	8	64	80.140	80	90.727	0.258	0.648	80	12.923
Perf→Reg	softmax	8	64	80.800	78	91.513	0.235	0.622	80	13.219
Reg→Perf	softmax	8	64	78.940	70	90.678	0.256	0.660	80	13.139

$\approx$ 78–79% (8 layers), confirming that depth is a primary driver of representation power, albeit with higher compute cost per epoch.

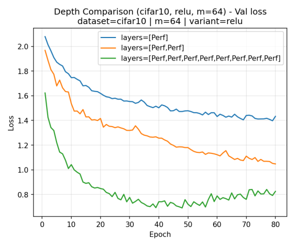
From Figures 1 and 2, we clearly observe that increasing the number of layers leads to faster convergence: deeper models reduce the validation loss more quickly and reach higher training accuracy earlier. Even on CIFAR-10, we see mild overfitting for the deepest setting ($d = 8$), visible through a gap where we see training accuracy improve as validation loss plateaus and



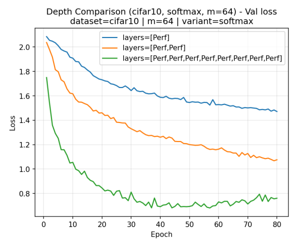
(a) MNIST, ReLU.



(b) MNIST, softmax.

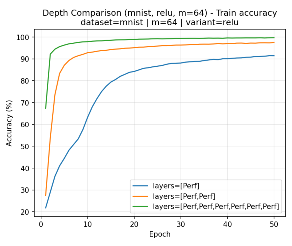


(c) CIFAR-10, ReLU.

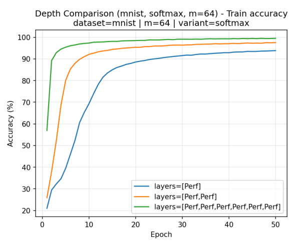


(d) CIFAR-10, softmax.

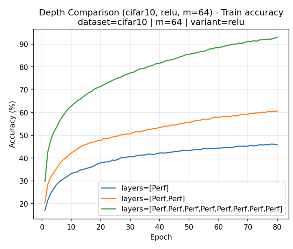
Figure 1: Validation loss curves for depth comparisons (MNIST: 1 vs 2 vs 6 layers; CIFAR-10: 1 vs 2 vs 8 layers), for both kernels with `m`=64.



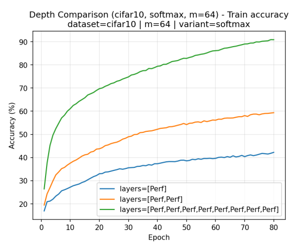
(a) MNIST, ReLU.



(b) MNIST, softmax.

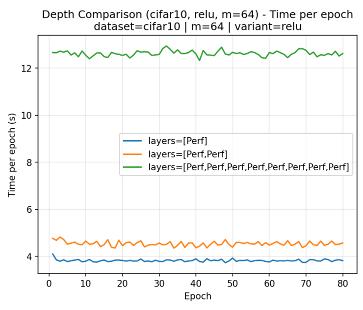


(c) CIFAR-10, ReLU.

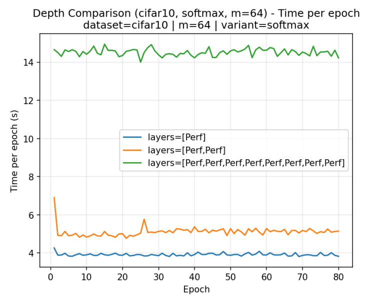


(d) CIFAR-10, softmax.

Figure 2: Training accuracy curves for the same depth comparisons as in Figure 1.



(a) CIFAR-10, ReLU kernel (`m`=64).



(b) CIFAR-10, softmax kernel (`m`=64).

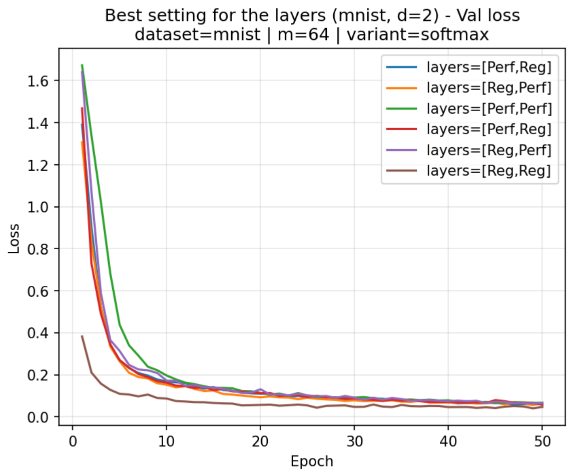
Figure 3: Median time per epoch across depth comparisons for CIFAR-10 dataset (same runs as Figures 1–2).

slightly degrades.

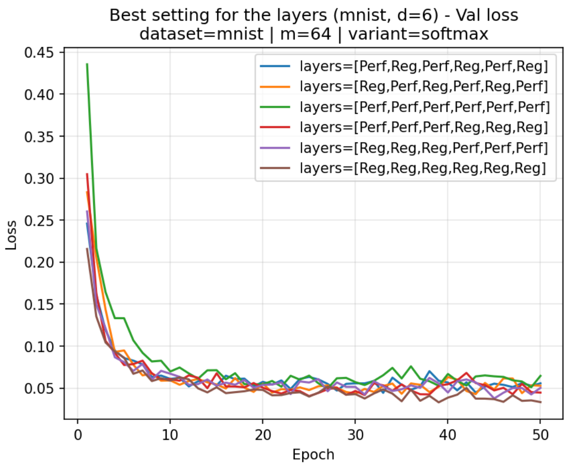
Regarding training time, Figure 3 shows that the $d = 8$ model is more than three times slower per epoch than $d = 1$, and more than two times slower than $d = 2$. Therefore, $d = 2$ offers a much faster training regime than very deep networks, but it cannot reach the maximum accuracy achieved with more depth. Importantly, the choice of kernel (ReLU vs softmax) does not change these global depth-driven trends. So far, we do not see anything unexpected in our results. We now move on to the analysis of layer-type settings.

4.2.2 Implications of the Layer Type

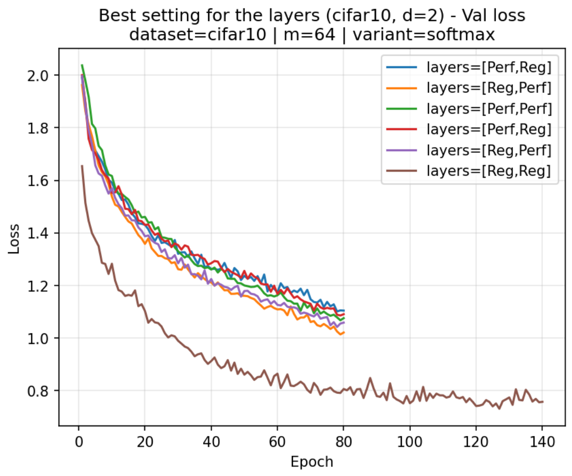
At first glance, the global summary tables indicate that the best validation accuracy is primarily driven by depth: deeper models (larger number of layers) achieve higher peaks on both MNIST and CIFAR-10, while shallower models train faster per epoch. In contrast, changing the `setting` (Full Reg, Full Perf, Intertwined, Reg→Perf, Perf→Reg) yields comparatively smaller differences once depth is fixed, especially at larger depths.



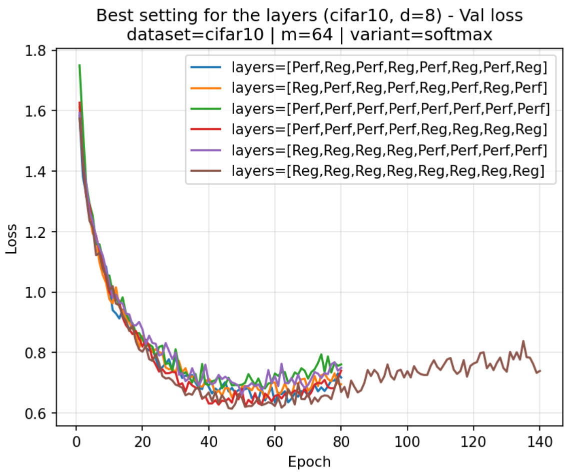
(a) MNIST, $d = 2$ (softmax).



(b) MNIST, $d = 6$ (softmax).

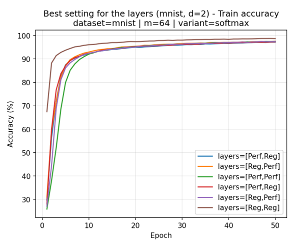


(c) CIFAR-10, $d = 2$ (softmax).

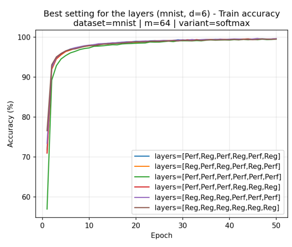


(d) CIFAR-10, $d = 8$ (softmax).

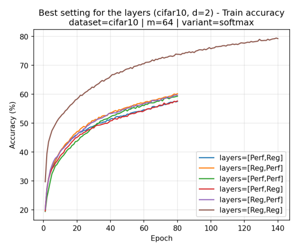
Figure 4: Validation loss for the "best setting for the layers" comparisons, for each dataset and depth.



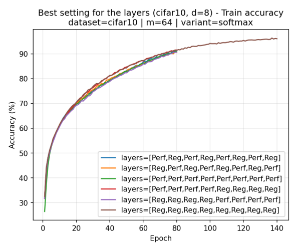
(a) MNIST, $d = 2$.



(b) MNIST, $d = 6$.



(c) CIFAR-10, $d = 2$.

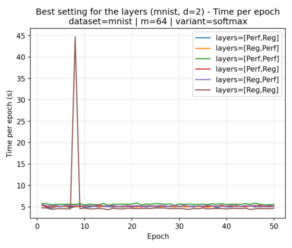


(d) CIFAR-10, $d = 8$.

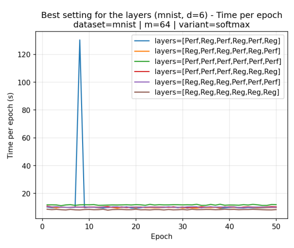
Figure 5: Training accuracy for the "best setting for the layers" comparisons.

More precisely, by inspecting Figures 4 and 5, we see that the different hybrid configurations do not create strong distinctions in either validation loss or training accuracy trajectories. The main visible exception occurs for CIFAR-10 with $d = 2$, where `Full Reg` achieves noticeably better performance than all Performer-based variants. However, when depth increases (e.g. CIFAR-10 with $d = 8$, or MNIST with $d = 6$), this advantage largely disappears and the curves become much closer across settings.

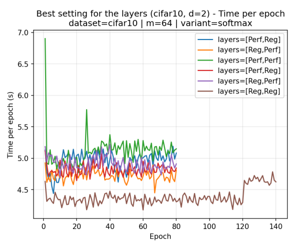
A somewhat surprising phenomenon also appears in Figure 6: `Full Reg` can take less time per



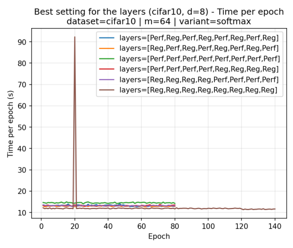
(a) MNIST, $d = 2$.



(b) MNIST, $d = 6$.



(c) CIFAR-10, $d = 2$.



(d) CIFAR-10, $d = 8$.

Figure 6: Time per epoch for the "best setting for the layers" comparisons.

epoch than `Full Perf`, even though Performer attention is designed to reduce the asymptotic cost of attention. In our implementation and for the sequence lengths considered here, the overhead of random feature construction (and related operations) can dominate, so linear-time attention does not necessarily translate into lower wall-clock time.

Because these comparisons were not strongly conclusive in favor of Performer-based settings, we launched additional CIFAR-10 simulations with modified hyperparameters chosen to be more favorable to Performers. In these new runs, we used:

$$\texttt{mlp\_ratio} = 1.0, \qquad \texttt{d\_model} : 64 \to 128, \qquad \texttt{patch} : 4 \to 2,$$

while keeping the remaining training hyperparameters unchanged.

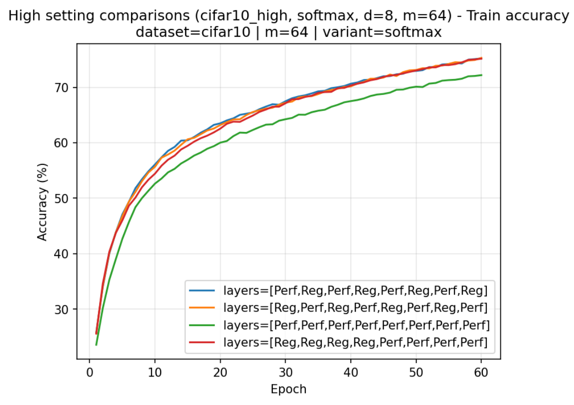
These changes are expected to benefit Performer variants for two reasons. First, decreasing `mlp_ratio` reduces the relative computational dominance of the MLP block, making attention (and its approximation quality) more central to performance and runtime comparisons. Second, using smaller patches increases the number of tokens L (since more patches are extracted per image), which makes the quadratic cost of regular attention more burdensome and pushes the model into a regime where linear-time attention should become comparatively more attractive. Increasing `d_model` also increases the expressiveness of the representation learned per token, which can partially compensate for approximation errors induced by random features.

Table 3: CIFAR-10 (high-token / higher-capacity setting). Summary of the additional runs with `mlp_ratio`=1.0, `d_model`=128, and `patch`=2. We report the number of attention layers, the number of random features `m`, and the best validation accuracy (and epoch), together with the best training accuracy, the minimum training/validation losses reached during training, the total number of epochs, and the median epoch time.

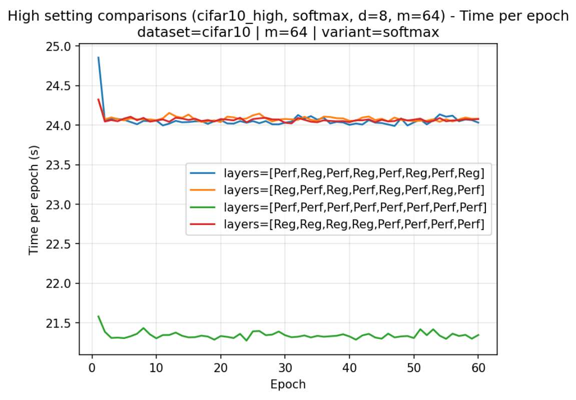
`setting`	`kernel`	#layers	`m`	best val acc (%)	best epoch	best train acc (%)	best train loss	best val loss	epochs	median time/epoch (s)
Full Perf	softmax	8	64	70.940	58	72.187	0.776	0.826	60	21.335
Intertwined (start=Perf)	softmax	8	64	73.140	58	75.209	0.694	0.765	60	24.048
Intertwined (start=Reg)	softmax	8	64	73.400	59	75.303	0.696	0.755	60	24.074
Reg→Perf	softmax	8	64	73.920	58	75.147	0.697	0.764	60	24.063

Overall, these new CIFAR-10 runs show a consistent improvement over the pure Performer baseline: introducing some regular attention layers (either intertwined or in a Reg→Perf schedule) increases the best validation accuracy by roughly 2–3 points compared to `Full Perf`. Among the tested variants, `Reg→Perf` achieves the best validation accuracy (73.92%). However, the time-per-epoch also increases slightly for hybrid variants compared to `Full Perf`, indicating that the accuracy gain comes at the cost of reintroducing some regular attention computation. This reinforces the idea that hybrid attention can be a practical compromise: it mitigates approximation-induced degradation while preserving part of the scalability benefits targeted by Performer-style layers.

Figure 7 confirms the trends highlighted by Table 3: hybrid settings (mixing regular and Performer layers) improves the training accuracy compared to `Full Perf`, while also incurring a slightly larger time per epoch.



(a) Training accuracy (CIFAR-10 high setting).



(b) Time per epoch (CIFAR-10 high setting).

Figure 7: CIFAR-10 high setting: comparison of `Full Perf`, intertwined variants, and `Reg`$\rightarrow$`Perf`.

4.3 Implications of the Kernel Type

We now study the impact of the attention kernel by comparing ReLU and softmax Performer variants at fixed depth and number of random features. We rely on the global summary tables for MNIST (Table 1) and CIFAR-10 (Table 2). Overall, on MNIST the softmax kernel tends to match or slightly outperform ReLU in terms of best validation accuracy once training converges. For CIFAR-10, the gap is less stable and appears more sensitive to the training horizon and model depth. In particular, for shallow settings (e.g. $d = 1$) the softmax Performer often underperforms, whereas for deeper settings the two kernels are more similar, consistent with the notion that the softmax kernel tends to require longer optimization time to exploit its stronger selectivity.

A notable phenomenon emerges in Figure 8: ReLU-based attention often achieves higher training accuracy early in training, but as training continues, the softmax kernel catches up and sometimes even surpasses the ReLU-based attention implementation. This effect is particularly clear on MNIST, where the softmax model has enough epochs to truly converge. On CIFAR-10, the dataset is harder to optimize and convergence is slower; hence, in some shallow settings (e.g. CIFAR-10 with a single layer), the difference is striking because softmax does not have sufficient time to catch up within the chosen training horizon.

This behavior is consistent with the optimization and variance trade-offs discussed in Section 2.2.3. In brief, the ReLU kernel yields smoother, less selective attention and typically provides more stable gradients early on, which can accelerate initial progress. On the other hand, the softmax kernel can yield sharper, more selective attention patterns that improve final performance, but its random-feature approximation is more sensitive and may require more optimization time (and stabilization) to fully exploit this selectivity.

Overall, this pattern makes us notice a similar two-regime effect to the one reported for generative vs. discriminative classifiers, where one method can dominate early (small sample / few updates) while the other catches up and wins asymptotically [9]. From our results, we see how this is the case with the softmax kernel.

4.4 Implications of the Number of Features

We now investigate the influence of the number of random features `m` used by Performer-style attention, using the global summary tables for MNIST (Table 1) and CIFAR-10 (Table 2). Overall, the best accuracies do not improve monotonically with `m`: in several configurations, smaller feature maps achieve comparable (or slightly better) validation accuracy than larger ones. However, when `m` becomes very large (e.g. 1024 or 4096), the computational cost increases

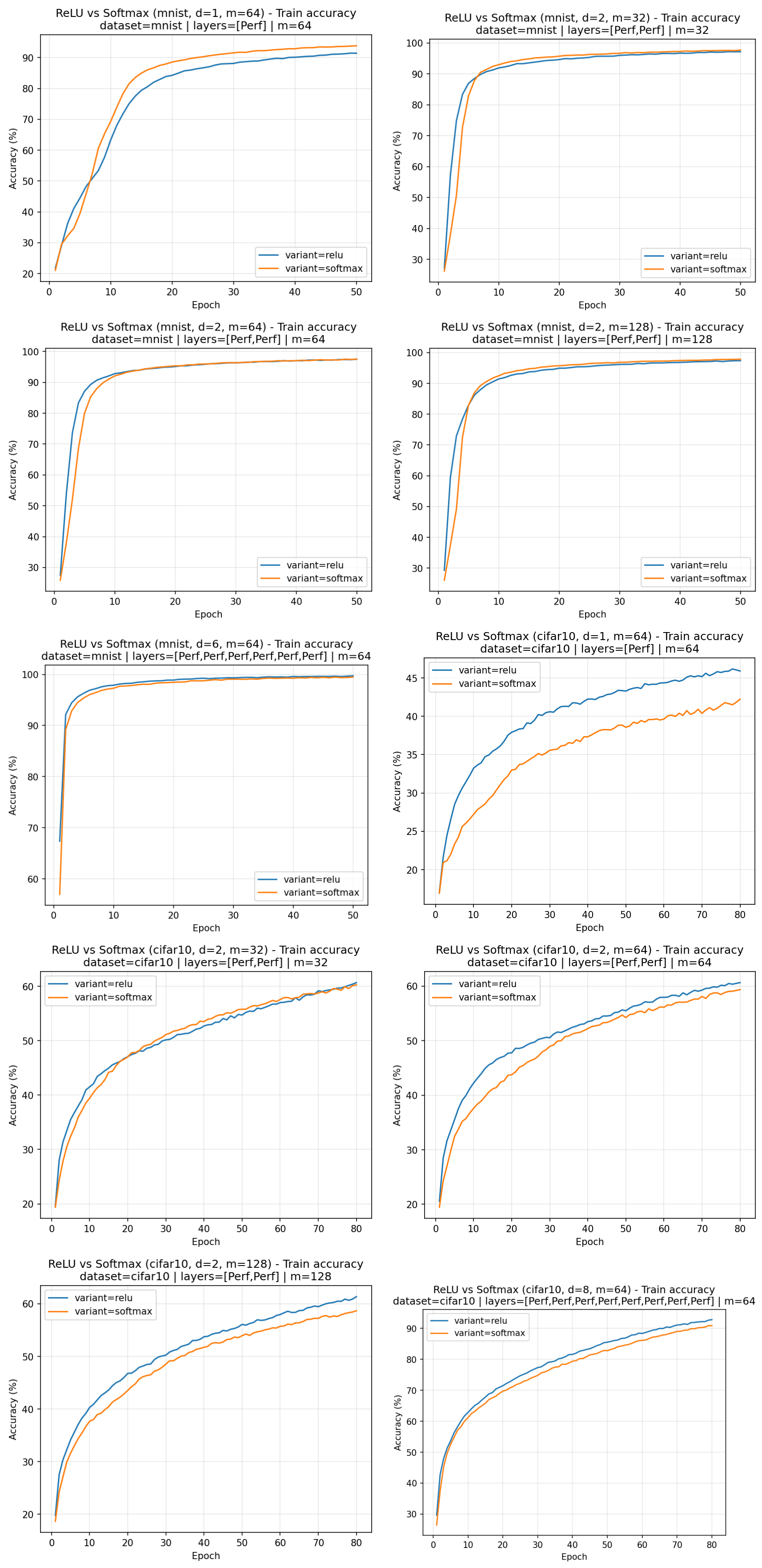
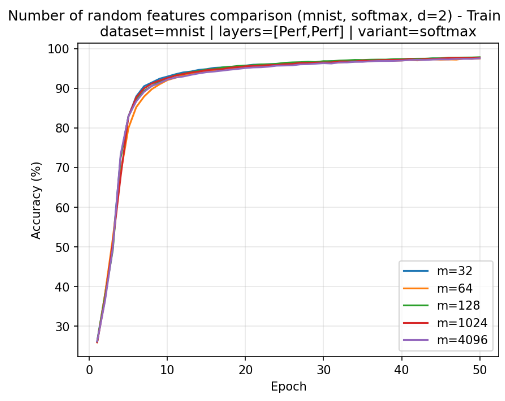
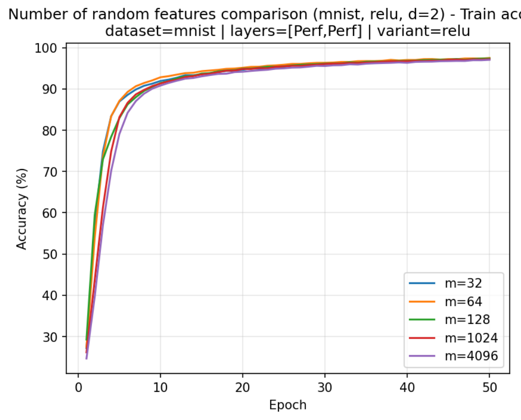


Figure 8: Training accuracy curves comparing ReLU and softmax kernels across MNIST and CIFAR-10 configurations (two plots per row, five rows).

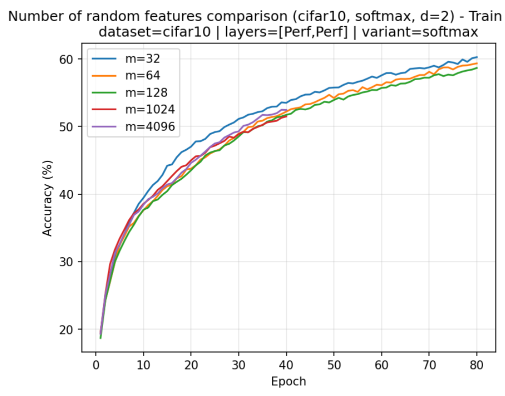
substantially, leading to much longer training times per epoch.



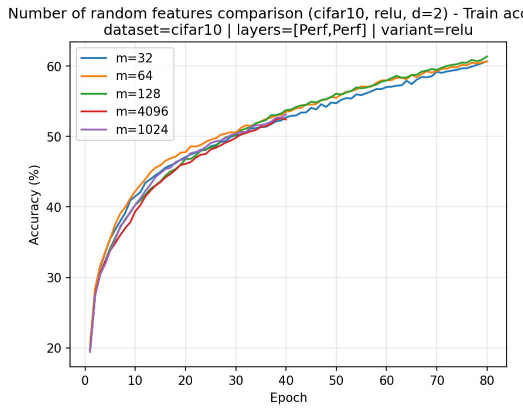
(a) MNIST, softmax, $d = 2$.



(b) MNIST, ReLU, $d = 2$.

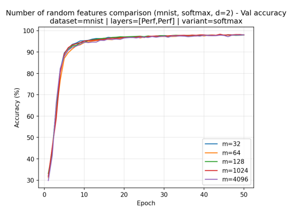


(c) CIFAR-10, softmax, $d = 2$.

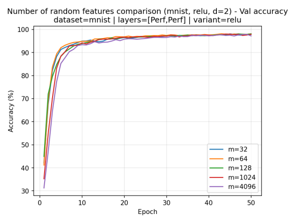


(d) CIFAR-10, ReLU, $d = 2$.

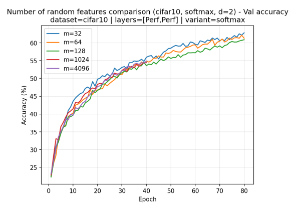
Figure 9: Training accuracy as a function of the number of random features m.



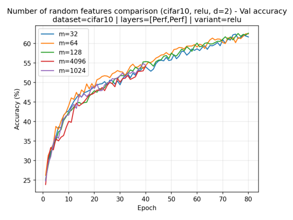
(a) MNIST, softmax.



(b) MNIST, ReLU.

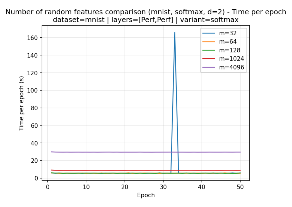


(c) CIFAR-10, softmax.

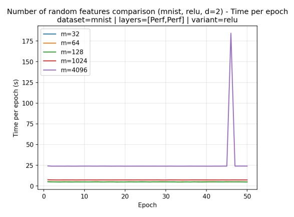


(d) CIFAR-10, ReLU.

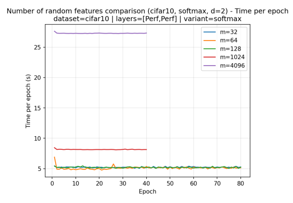
Figure 10: Validation accuracy as a function of the number of random features m.



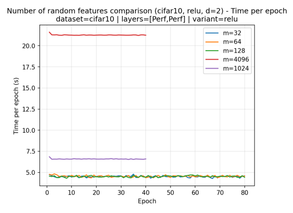
(a) MNIST, softmax.



(b) MNIST, ReLU.



(c) CIFAR-10, softmax.



(d) CIFAR-10, ReLU.

Figure 11: Median time per epoch as a function of the number of random features m.

Figures 9 and 10 suggest that, in our runs, using fewer random features (`m` smaller) often yields better accuracy. This trend should be interpreted with caution as it would require multiple runs per configuration to confirm whether it persists on average, and to quantify variance across random seeds.

As expected, Figure 11 shows that increasing `m` quickly increases the time per epoch, especially for the largest feature maps. Moreover, we see that the time gap between small and large `m` becomes larger when depth increases, since random-feature construction is repeated at every Performer layer.

4.5 Conclusion and Future Directions

This paper highlights that the strongest performance driver is the architectural capacity of the model. In particular, increasing the number of attention layers consistently improves the best validation accuracy on both MNIST and CIFAR-10, at the price of longer training times per epoch (and, in some cases, mild overfitting at the largest depths). In contrast, changing the layer-type setting (Full Reg, Full Perf, intertwined, or staged hybrids) typically leads to smaller differences once depth is fixed.

For Performer-style attention, the number of random features `m` mainly impacts computational cost: very large `m` values substantially increase the epoch time, while the accuracy gains are not systematic in our single-run results. Similarly, the kernel choice (ReLU vs softmax) affects the optimization dynamics more than the final peak: ReLU often gives a faster initial rise in accuracy, while softmax catches up as training time increases.

We note that these conclusions must be taken with caution, as most configurations were only run once due to limited resources and long training times. As a consequence, we do not report means or standard deviations over multiple seeds, and some observed differences (especially when small) may be influenced by run-to-run variability.

For future research directions, a more rigorous study would (i) repeat each configuration with several random seeds and report averaged results, (ii) further explore hyperparameter regimes that are more favorable to Performer-based attention (as suggested by the CIFAR-10 high setting), and (iii) broaden the sweep to additional configurations, including larger depths, alternative hybrid schedules, and other model sizes, to better understand when Performer variants provide a consistent benefit.

References

[1] A. Vaswani *et al.*, "Attention Is All You Need," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 30, 2017. [Online]. Available: `https://papers.neurips.cc/paper/7181-attention-is-all-you-need.pdf`

[2] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The Efficient Transformer," *arXiv preprint arXiv:2001.04451*, 2020. [Online]. Available: `https://arxiv.org/pdf/2001.04451`

[3] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-Attention with Linear Complexity," *arXiv preprint arXiv:2006.04768*, 2020. [Online]. Available: `https://arxiv.org/pdf/2006.04768`

[4] K. Choromanski *et al.*, "Rethinking attention with performers," *arXiv preprint arXiv:2009.14794*, 2020. [Online]. Available: `https://arxiv.org/abs/2009.14794`

[5] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 20, 2007, pp. 1177–1184. [Online]. Available: `https://papers.neurips.cc/paper/3182-random-features-for-large-scale-kernel-machines.pdf`

[6] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2021. [Online]. Available: `https://arxiv.org/pdf/2010.11929`

[7] Y. LeCun, C. Cortes, and C. J. C. Burges. The MNIST Database of Handwritten Digits. 1998. `https://yann.lecun.org/exdb/mnist/`. Accessed: 2025-12-15.

[8] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical Report, University of Toronto, 2009. `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

[9] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, 2001, pp. 841–848.

1 Appendix

1.1 Proof of Result 2

Proof. Let $\boldsymbol{g} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_d)$ and set $a = \boldsymbol{g}^\top \boldsymbol{q}$ and $b = \boldsymbol{g}^\top \boldsymbol{k}$. By rotational invariance of the Gaussian, we may assume

$$\boldsymbol{q} = \|\boldsymbol{q}\|\boldsymbol{e}_1, \qquad \boldsymbol{k} = \|\boldsymbol{k}\|(\cos\theta\, \boldsymbol{e}_1 + \sin\theta\, \boldsymbol{e}_2). \tag{25}$$

Writing $\boldsymbol{z} = (z_1, z_2) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_2)$, we obtain $a = \|\boldsymbol{q}\| z_1$ and $b = \|\boldsymbol{k}\|(\cos\theta\, z_1 + \sin\theta\, z_2)$, hence

$$\mathbb{E}[a_+ b_+] = \|\boldsymbol{q}\|\,\|\boldsymbol{k}\|\,\mathbb{E}\big[z_{1,+}\,(\cos\theta\, z_1 + \sin\theta\, z_2)_+\big]. \tag{26}$$

Using polar coordinates $(z_1, z_2) = (r\cos\varphi, r\sin\varphi)$ with density $\frac{1}{2\pi} e^{-r^2/2} r\, dr\, d\varphi$, the positivity constraints become $\cos\varphi \geq 0$ and $\cos(\varphi - \theta) \geq 0$, i.e., $\varphi \in [\theta - \pi/2, \pi/2]$. Therefore,

$$\mathbb{E}[a_+ b_+] = \frac{\|\boldsymbol{q}\|\,\|\boldsymbol{k}\|}{2\pi} \int_0^\infty r^3 e^{-r^2/2}\, dr \int_{\theta-\pi/2}^{\pi/2} \cos\varphi\, \cos(\varphi - \theta)\, d\varphi. \tag{27}$$

For the radial part, with $t = r^2/2$ (so $dt = r\, dr$ and $r^2 = 2t$),

$$\int_0^\infty r^3 e^{-r^2/2}\, dr = \int_0^\infty (2t) e^{-t}\, dt = 2. \tag{28}$$

For the angular part, use

$$\cos\varphi \cos(\varphi - \theta) = \frac{1}{2}\Big(\cos(2\varphi - \theta) + \cos\theta\Big), \tag{29}$$

so

$$\begin{aligned}
\int_{\theta-\pi/2}^{\pi/2} \cos\varphi \cos(\varphi - \theta)\, d\varphi &= \frac{1}{2}\int_{\theta-\pi/2}^{\pi/2} \cos(2\varphi - \theta)\, d\varphi + \frac{1}{2}\cos\theta \int_{\theta-\pi/2}^{\pi/2} 1\, d\varphi \\
&= \frac{1}{4}\Big[\sin(2\varphi - \theta)\Big]_{\theta-\pi/2}^{\pi/2} + \frac{1}{2}(\pi - \theta)\cos\theta \\
&= \frac{1}{4}\big(\sin(\pi - \theta) - \sin(\theta - \pi)\big) + \frac{1}{2}(\pi - \theta)\cos\theta \\
&= \frac{1}{2}\sin\theta + \frac{1}{2}(\pi - \theta)\cos\theta.
\end{aligned} \tag{30}$$

This yields the claimed formula. □

1.2 Proof of Result 3

Proof. Keeping the same notations as in the proof of Result 2, define $Z = a_+ b_+$. By independence,

$$\mathrm{Var}\big(\widehat{k}_m^{\mathrm{relu}}(\boldsymbol{q}, \boldsymbol{k})\big) = \frac{1}{m}\mathrm{Var}(Z) = \frac{1}{m}\Big(\mathbb{E}[Z^2] - \mathbb{E}[Z]^2\Big). \tag{31}$$

Computation of $\mathbb{E}[Z^2]$. Similarly,

$$\mathbb{E}[Z^2] = \|\boldsymbol{q}\|^2 \|\boldsymbol{k}\|^2 \frac{1}{2\pi} \int_0^\infty \int_{\theta-\pi/2}^{\pi/2} (r\cos\varphi)^2\, (r\cos(\varphi - \theta))^2\, e^{-r^2/2}\, r\, d\varphi\, dr, \tag{32}$$

hence

$$\mathbb{E}[Z^2] = \|\boldsymbol{q}\|^2 \|\boldsymbol{k}\|^2 \frac{1}{2\pi} \Big(\int_0^\infty r^5 e^{-r^2/2}\, dr\Big)\Big(\int_{\theta-\pi/2}^{\pi/2} \cos^2\varphi\, \cos^2(\varphi - \theta)\, d\varphi\Big). \tag{33}$$

For the radial term, with $t = r^2/2$,

$$\int_0^\infty r^5 e^{-r^2/2}\,dr = \int_0^\infty (2t)^2 e^{-t}\,dt = 8. \tag{34}$$

For the angular term, expand $\cos^2 u = \frac{1}{2}(1+\cos 2u)$:

$$\cos^2\varphi\,\cos^2(\varphi-\theta) = \frac{1}{4}\Big(1+\cos 2\varphi + \cos(2\varphi-2\theta) + \cos 2\varphi\,\cos(2\varphi-2\theta)\Big), \tag{35}$$

and use

$$\cos 2\varphi\,\cos(2\varphi-2\theta) = \frac{1}{2}\Big(\cos(4\varphi-2\theta)+\cos 2\theta\Big). \tag{36}$$

Integrating term-by-term over $\varphi \in [\theta-\pi/2, \pi/2]$ yields

$$\int_{\theta-\pi/2}^{\pi/2} \cos^2\varphi\,\cos^2(\varphi-\theta)\,d\varphi = \frac{1}{8}\Big(2(\pi-\theta) + (\pi-\theta)\cos 2\theta + \frac{3}{2}\sin 2\theta\Big) = \frac{1}{8}E_1(\theta), \tag{37}$$

so

$$\mathbb{E}[Z^2] = \frac{\|\boldsymbol{q}\|^2\|\boldsymbol{k}\|^2}{2\pi}\,E_1(\theta). \tag{38}$$

Conclusion and extrema. Combining with Result 2 gives

$$\mathrm{Var}(Z) = \frac{\|\boldsymbol{q}\|^2\|\boldsymbol{k}\|^2}{2\pi}E_1(\theta) - \frac{\|\boldsymbol{q}\|^2\|\boldsymbol{k}\|^2}{4\pi^2}E_2(\theta)^2, \tag{39}$$

and thus

$$\mathrm{Var}\big(\widehat{k}_m^{\mathrm{relu}}(\boldsymbol{q},\boldsymbol{k})\big) = \frac{\|\boldsymbol{q}\|^2\|\boldsymbol{k}\|^2}{m}\left(\frac{1}{2\pi}E_1(\theta) - \frac{1}{4\pi^2}E_2(\theta)^2\right). \tag{40}$$

The endpoint evaluations yield the stated minimum and maximum, and plugging $\theta = \pi/2$ gives the explicit value. □