Problem 1 Vision Transformers

Fengyi Ding Zibo Wang Nan Yang Ruize Qin Yaozu Hao

Abstract This study conducts a comprehensive analysis of Vision Transformers (ViT), Simple Vision Transformers (SimpleViT), and Performer models for image classification tasks on CIFAR-10 and MNIST datasets. The models were evaluated based on training time, inference time, and test accuracy, with additional comparisons under various hyperparameters, including depth, embedding dimensions, learning rate, and dropout rate. Performer models, utilizing the FAVOR+ mechanism, achieve linear complexity in self-attention and demonstrate scalability and efficiency. SimpleViT exhibits the best trade-off between accuracy and computational efficiency, while the ViT's complex architecture shows potential in capturing global features but incurs higher resource consumption, with slightly lower accuracy than SimpleViT. Among Performer variants, the learnable kernel function (f_θ) achieves notable accuracy while maintaining computational efficiency. This work highlights the trade-offs between model complexity, training stability, and task-specific performance. The structure of this paper includes dataset descriptions, model overviews, performance evaluations, hyperparameter analyses, a discussion on trade-offs and a further discussion on possible causes for performers' performance gap.

Keywords Vision Transformer; Performer; CIFAR-10; deep learning; attention mechanism; hyperparameter tuning

1 Dataset

1.1 CIFAR-10

The CIFAR-10 dataset comprises a total of 60,000 color images. Each image has a fixed resolution of 32×32 pixels and is uniformly distributed across 10 distinct classes. These classes represent common objects such as airplanes, automobiles, birds, cats, and others.

The dataset is divided into two primary subsets: a training set containing 50,000 images and a test set comprising 10,000 images. The training set is further structured into five equally sized batches, each containing 10,000 images. Example images from the dataset are shown below:

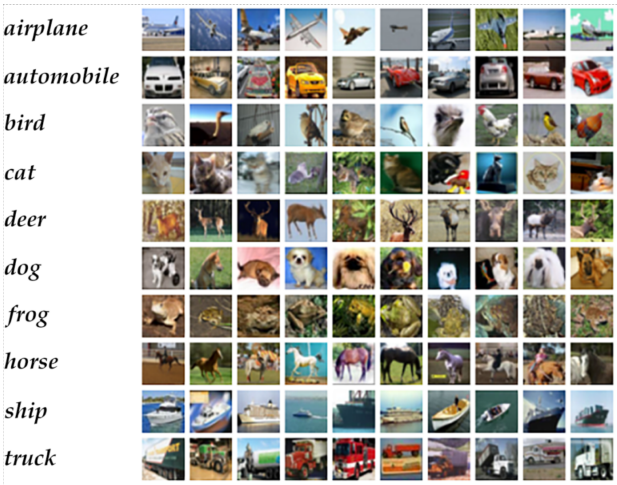

Fig. 1 Cifar10

1.2 MNIST

The MNIST dataset is a seminal benchmark in the domains of deep learning and computer vision, comprising a total of 70,000 grayscale images of handwritten digits, each uniformly sized at 28×28 pixels. These images are categorized into ten distinct classes, representing the digits 0 through 9. The dataset is widely recognized for its inherent variability in handwriting styles, which introduces a range of complexities for classification tasks.

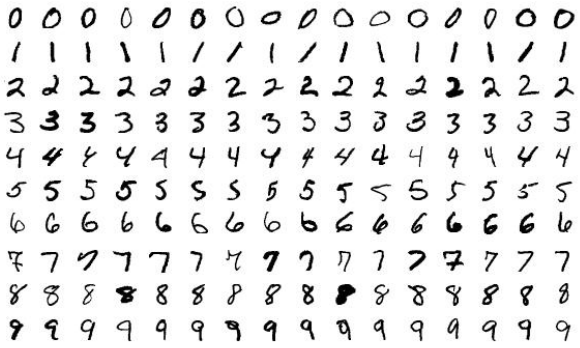

Fig. 2 MNIST

These variations include differences in stroke shapes, line thickness, slant angles, and overall proportions, effectively simulating the diversity encountered in real-world handwritten digit recognition scenarios. Such characteristics make MNIST a robust testbed for evaluating the performance of machine learning algo-

rithms, particularly in their ability to generalize across diverse input patterns.

Additionally, the dataset is split into two subsets: a training set with 60,000 images and a test set with 10,000 images. This clear partitioning enables a systematic approach to model training and evaluation.

2 Models' Description

2.1 Vision Transformer (ViT)

The Vision Transformer (ViT) is a novel deep learning architecture that leverages the power of Transformers for image classification tasks, deviating from traditional convolutional neural networks (CNNs). By treating an image as a sequence of smaller patches, ViT eliminates the need for convolution operations, enabling it to capture long-range dependencies in image data.

The Algorithm is shown as below:

Algorithm 1 Vision Transformer Inference Process

Input: An input image img with size (H, W) and patch size (P_h, P_w)
Output: Classification logits, i.e., $x \in \mathbb{R}^{\text{num\_classes}}$
  Partition img into patches of size (P_h, P_w).
  Map each patch to a dim-dimensional embedding, forming a sequence $x \in \mathbb{R}^{N \times dim}$, where N = number of patches.
  Prepend a learnable class token cls_token to the sequence x.
  Add positional embeddings $pos\_embedding$ to x.
  Apply Dropout to x.
  FOR $i = 1$ to $depth$ **DO**
    $x \leftarrow x + Attention(LayerNorm(x))$
    $x \leftarrow x + FeedForward(LayerNorm(x))$
  END FOR
  IF $pool$ = 'cls' **THEN**
    $x \leftarrow x_{cls}$ // select the class token representation
  ELSE
    $x \leftarrow \text{mean}(x)$ // average over all tokens
  END IF
  $x \leftarrow LayerNorm(x)$
  $x \leftarrow Linear(x, num\_classes)$

Key components of the ViT model include:

(1) **Patch Embedding**

Each input image is divided into fixed-size patches (e.g., 4×4 pixels for CIFAR-10) and flattened into vectors. These vectors are linearly projected into a high-dimensional embedding space, forming patch embeddings. Learnable positional encodings are added to retain the spatial structure of the image.

(2) **Attention Mechanism**

The Attention module forms the core of the Transformer architecture, enabling the model to compute relationships between all image patches. It operates as follows:

- Query, Key, and Value Projections: Each patch embedding is linearly transformed into three vectors: Query (Q), Key (K), and Value (V). These represent the relationship between patches.
- Self-Attention Calculation: The attention scores are computed by taking the dot product of Q and K, scaled by the square root of the embedding dimension, and passed through a softmax function. These scores indicate the importance of each patch to every other patch.
- Weighted Summation: The attention scores are used to weight the Value (V) vectors, producing the final attention output.

This mechanism is repeated for multiple attention heads, allowing the model to focus on different relationships between patches.

(3) **Transformer Encoder** The Transformer encoder, which is the core of ViT, comprises multiple layers of multi-head self-attention and feedforward neural networks. The self-attention mechanism enables the model to capture global dependencies between patches, while the feedforward layers refine these representations. This design allows ViT to process entire images without the locality constraints of CNNs.

(4) **Classification Head** A [CLS] token is appended to the sequence of patch embeddings, and its representation after passing through the Transformer encoder is used for classification. This representation is fed into a multi-layer perceptron (MLP) head to produce the final predictions.

(5) **Training and Optimization** The model was trained on the CIFAR-10 dataset with a configuration that included an embedding dimension of 128, 6 Transformer layers, 8 attention heads, and a hidden dimension of 512 in the MLP. Dropout was applied to reduce

overfitting, and the AdamW optimizer with a cosine annealing scheduler was used to improve convergence.

Table 1 Model Summary

Layer (type)	Output Shape	Param #
Rearrange-1	[-1, 64, 48]	0
Linear-2	[-1, 64, 96]	4,704
Dropout-3	[-1, 65, 96]	0
LayerNorm-4	[-1, 65, 96]	192
Linear-5	[-1, 65, 1152]	110,592
Softmax-6	[-1, 6, 65, 65]	0
Linear-7	[-1, 65, 96]	36,960
Dropout-8	[-1, 65, 96]	0
Attention-9	[-1, 65, 96]	0
PreNorm-10	[-1, 65, 96]	0
LayerNorm-11	[-1, 65, 96]	192
Linear-12	[-1, 65, 512]	49,664
GELU-13	[-1, 65, 512]	0
Dropout-14	[-1, 65, 512]	0
Linear-15	[-1, 65, 96]	49,248
Dropout-16	[-1, 65, 96]	0
FeedForward-17	[-1, 65, 96]	0
Transformer-64	[-1, 65, 96]	0
Identity-65	[-1, 96]	0
LayerNorm-66	[-1, 96]	192
Linear-67	[-1, 10]	970
Total Parameters:		**993,258**
Trainable Params:		**993,258**
Non-trainable Params:		**0**

2.2 Simple Vision Transformer (SimpleViT)

The Simple Vision Transformer (SimpleViT) is a streamlined variant of the Vision Transformer (ViT) architecture, focusing on simplifying positional encoding and overall computational complexity. This model retains the core concepts of patch embedding and Transformer-based processing while introducing sinusoidal positional encodings for efficiency.

Key components of the SimpleViT model include:

(1) **Patch Embedding**

The input image is divided into fixed-size patches (e.g., 4×4 pixels for CIFAR-10), which are flattened and passed through a linear layer to form embeddings. Unlike traditional ViT, positional encodings are generated using a 2D sinusoidal function (`posemb_sincos_2d`), avoiding the need for learnable positional parameters.

(2) **Attention Mechanism**

The Attention module is the core of the SimpleViT architecture, enabling the model to compute relationships between all image patches. It operates as follows:

- Normalization: Inputs are normalized using LayerNorm.
- Query, Key, and Value (QKV) Projections: Patch embeddings are linearly projected into Query (Q), Key (K), and Value (V) representations.
- Self-Attention Calculation: Attention scores are computed by taking the scaled dot product of Q and K, followed by a softmax function. These scores indicate the importance of each patch to every other patch.
- Weighted Summation: Attention scores are used to weight the Value (V) vectors, producing the final attention output.
- Output Projection: The result is projected back to the original dimensionality using a linear layer.

(3) **Transformer Encoder**

The Transformer encoder consists of multiple stacked layers, each comprising an attention mechanism and a feedforward network. Each layer follows a residual structure where the output of attention and feedforward operations is added back to the input.

(4) **Classification Head**

The patch embeddings are processed by the Transformer encoder, and the resulting representations are pooled (mean pooling) to produce a single vector. This vector is passed through a linear layer for classification.

(5) **Efficiency and Optimization**

SimpleViT replaces ViT's learnable positional encodings with sinusoidal encodings, reducing the model's parameter count and improving computational efficiency. The model was trained on the CIFAR-10 dataset with an embedding dimension of 128, 6 Transformer layers, 8 attention heads, and an MLP hidden

dimension of 512. The AdamW optimizer and the cosine annealing scheduler were used during training for up to 100 epochs.

Table 2 SimpleViT Model Summary

Layer (type)	**Output Shape**	**Param #**
Rearrange-1	[-1, 8, 8, 48]	0
Linear-2	[-1, 8, 8, 128]	6,272
LayerNorm-3	[-1, 64, 128]	256
Linear-4	[-1, 64, 1536]	196,608
Softmax-5	[-1, 8, 64, 64]	0
Linear-6	[-1, 64, 128]	65,536
Attention-7	[-1, 64, 128]	0
LayerNorm-8	[-1, 64, 128]	256
Linear-9	[-1, 64, 512]	66,048
GELU-10	[-1, 64, 512]	0
Linear-11	[-1, 64, 128]	65,664
FeedForward-12	[-1, 64, 128]	0
Repeated Transformer Blocks (4 Layers)		
LayerNorm-n	[-1, 64, 128]	256
Linear-$n+1$	[-1, 64, 1536]	196,608
Softmax-$n+2$	[-1, 8, 64, 64]	0
Linear-$n+3$	[-1, 64, 128]	65,536
Attention-$n+4$	[-1, 64, 128]	0
LayerNorm-$n+5$	[-1, 64, 128]	256
Linear-$n+6$	[-1, 64, 512]	66,048
GELU-$n+7$	[-1, 64, 512]	0
Linear-$n+8$	[-1, 64, 128]	65,664
FeedForward-$n+9$	[-1, 64, 128]	0
Transformer-63	[-1, 64, 128]	0
Identity-64	[-1, 128]	0
LayerNorm-65	[-1, 128]	256
Linear-66	[-1, 10]	1,290
Total Parameters:		**2,374,026**
Trainable Params:		**2,374,026**
Non-trainable Params:		**0**

2.3 Performer

The Performer is an advanced Transformer architecture that addresses the quadratic computational complexity of traditional attention mechanisms. Introduced in the research by Google (Google Research, 2020), Performer achieves linear complexity by approximating the standard softmax attention using kernel-based random feature transformations. This innovation makes it scalable to long sequences and large datasets while maintaining comparable accuracy to standard Transformers.

Key Features of Performer:

- Linear Time Complexity: Performer reduces attention complexity from $O(n^2)$ to $O(n)$ by replacing traditional dot-product attention with the FAVOR+ mechanism.
- Memory Efficiency: Its low memory footprint allows processing of longer sequences compared to standard Transformers.
- Provable Accuracy Guarantees: The kernel-based approximations provide theoretical guarantees for bounded approximation errors.

Key Components of the Performer model:

(1) **Patch Embedding**

Input images are divided into non-overlapping patches (e.g., 4×4 pixels for CIFAR-10). Each patch is projected into a high-dimensional space using a convolutional layer, resulting in embeddings of shape (batch size, number of patches, embedding dimension).

(2) **Positional Encoding**

To capture spatial relationships between patches, Performer uses:

- Fixed Sinusoidal Encoding: Derived from sine and cosine functions, this encoding preserves relative positional information.
- Learnable Encoding (Optional): Trainable positional encodings, though fixed encodings were used in our implementation for simplicity.

(3) **FAVOR+ Attention Mechanism**

The core innovation of Performer, FAVOR+ (Fast Attention Via Orthogonal Random features), approximates the dot-product attention using kernel transformations:

- ReLU Kernel Transformation: A non-linear mapping of input data to random feature space, enabling efficient computation of the ReLU kernel.

- Softmax Kernel Transformation: Approximates softmax-based attention using orthogonal random projections, significantly reducing computational overhead.

The FAVOR+ mechanism ensures linear complexity by avoiding the explicit computation of large attention matrices, instead using random feature approximations.

(4) **Transformer Block**

Each block in the Performer model consists of:

- FAVOR+-based Self-Attention: Efficient computation of pairwise relationships between patches.
- Feedforward Neural Network (FFN): A two-layer network with GELU activation for non-linearity.
- Residual Connections and Layer Normalization: These improve stability and convergence during training.

(5) **Classification Head**

A global average pooling operation aggregates information from all patches, followed by a fully connected layer with softmax activation to generate the final class probabilities.

Implementation Details:

The Performer was implemented using TensorFlow and trained on the CIFAR-10 dataset with the following hyperparameters:

- Image size: 32×32
- Patch size: 4×4
- Embedding dimension: 64
- Number of Transformer layers: 4
- Number of attention heads: 4
- Feedforward hidden dimension: 128
- Random features for FAVOR+: 256
- Learning rate: 1×10^{-3}
- Batch size: 64
- Epochs: 100

The Adam optimizer was used, with a categorical cross-entropy loss function and early stopping based on validation accuracy. Dropout was applied in both attention and feedforward layers to reduce overfitting.

Advantages of Performer (as discussed in the original paper)[?]:

- Scalability: Performer can handle sequences up to 10^6 elements efficiently, far beyond the capabilities of traditional Transformers.
- Speed: Experiments demonstrated up to 50x speed improvements for long sequences.
- Versatility: Performer has been successfully applied to tasks in NLP, image processing, and protein modeling.
- Compatibility: It integrates seamlessly into existing Transformer architectures with minimal modifications.

2.4 Performer: ReLU, Exp vs fθ

1. Objective

- **Performer-ReLU**: Utilizes the *ReLU Kernel Transformation* to approximate the characteristics of ReLU activation. It is effective for tasks where attention weight normalization is not strictly required.
- **Performer-Exp**: Uses the *Softmax Kernel Transformation* to approximate the standard Softmax operation in attention mechanisms, retaining normalization properties essential for many NLP tasks.
- **Performer-fθ**: Introduces a *Learnable Kernel Transformation*, which adapts dynamically to the input data distribution by learning kernel features directly from data. This flexibility allows the model to approximate a broader range of kernel transformations and optimize attention mechanisms for specific tasks.

2. Definition of Kernel Functions

- **ReLU Kernel Transformation:**

$$\phi_{\text{ReLU}}(x) = \text{ReLU}(x) + \epsilon$$

where ϵ is a small constant for numerical stability.

- **Softmax Kernel Transformation:**

$$\phi_{\text{Softmax}}(x) = e^{x \cdot \omega}$$

where ω is a random orthogonal projection matrix.

- **Learnable Kernel Transformation (f_θ):**

$$\phi_{f_\theta}(x) = W_2 \left(\text{GELU}\left(W_1(x)\right)\right)$$

where W_1 and W_2 are learnable linear transformations, and GELU is a non-linear activation function.

3. Complexity Both Performer ReLU and Performer f_θ achieve linear complexity $\mathcal{O}(n)$ by avoiding the explicit computation of large attention matrices.

4. Use Cases

- **Performer ReLU:** Suitable for lightweight tasks such as image processing and simple sequence modeling, where normalization is not critical.
- **Performer Exp:** Suitable for tasks requiring strict attention weight normalization, such as language modeling and complex NLP tasks.
- **Performer f_θ:** Suitable for tasks requiring adaptive kernel functions to capture complex attention patterns, such as high-dimensional image classification or specialized attention tasks in multimodal data.

Table 3 Comparison Between Performer ReLU, Performer Exp, and Performer f_θ

Feature	Performer ReLU	Performer Exp	Performer f_θ
Kernel Type	ReLU Kernel	Softmax Kernel	Learnable Kernel
Objective	Nonlinear activation	Standard attention approximation	Adaptive attention approximation
Complexity	$O(n)$	$O(n)$	$O(n)$
Accuracy	Lower, suitable for lightweight tasks	Higher, closer to standard attention	Task-dependent, can surpass both
Use Cases	Image processing, simple tasks	NLP, high-precision tasks	Complex attention tasks, multimodal data

3 Models' Performance and Evaluation

3.1 Vision Transformer (ViT)

3.1.1 Performance on CIFAR-10

The training time per epoch starts around **30 seconds** and shows a rapid increase to over **70 seconds** within the first 20 epochs based on **Fig.3**. This suggests initial overheads such as data preparation, GPU warm-up, or computation graph stabilization. After the initial rise, the training time fluctuates significantly between **60 and 70 seconds** in later epochs. The irregularities in this phase may result from resource contention on the system or dynamic memory allocation. The model architecture potentially becoming more computationally expensive as training progresses could also be a factor. There are notable dips and spikes in training time, particularly around epochs 40 and 80, which could indicate transient hardware or system interruptions.

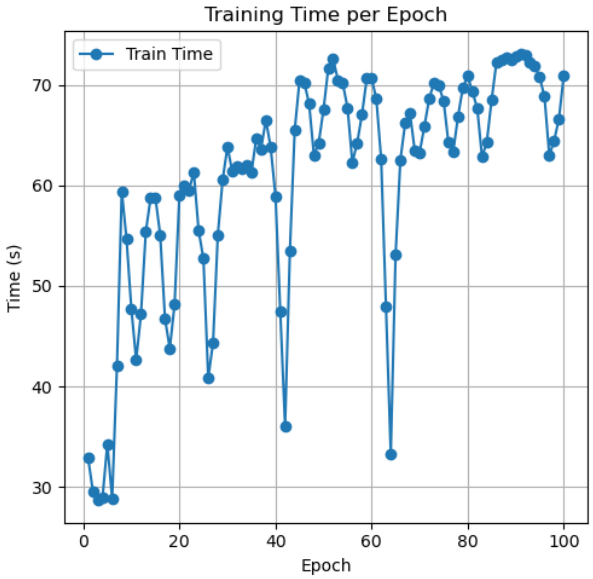

Fig. 3 Training Time for Vit Model on CIFAR-10

As shown in **Fig.4**, the inference time per epoch fluctuates between **9 and 14 seconds** with no significant trend of increase or decrease. This consistency suggests that the inference workload remains largely stable, but external factors such as I/O overhead or background processes cause periodic variability. Several spikes are visible, particularly near epochs 20 and 90, which may correlate with temporary resource contention or system interruptions.

Fig.5 shows that the test accuracy begins at approximately **22.5%** and steadily improves, reaching **75%** by epoch 40. After epoch 40, the accuracy continues to improve at a slower rate, eventually stabilizing around **78%** by epoch 100. This behavior is typical of a well-trained model nearing its capacity to generalize to the test data. The saturation at **78%** suggests that the model may require further tuning or additional regularization to achieve higher accuracy, such as adjusting the learn-

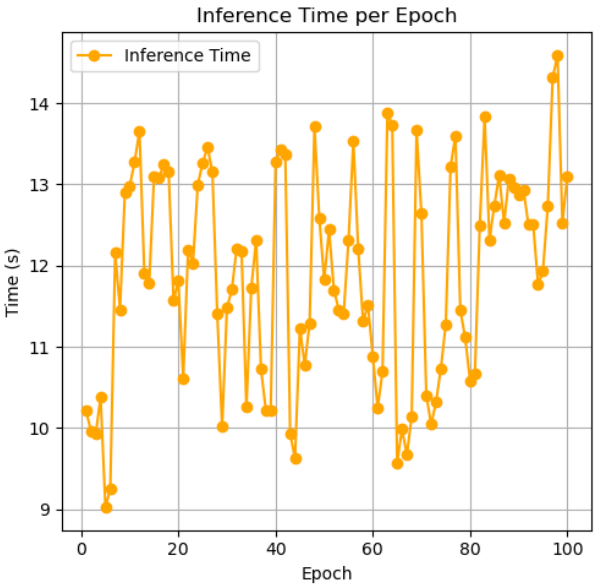

Fig. 4 Inference Time for ViT Model on CIFAR-10

ing rate or applying a scheduler, adding more data or performing augmentation, or increasing model capacity by adding layers or neurons.

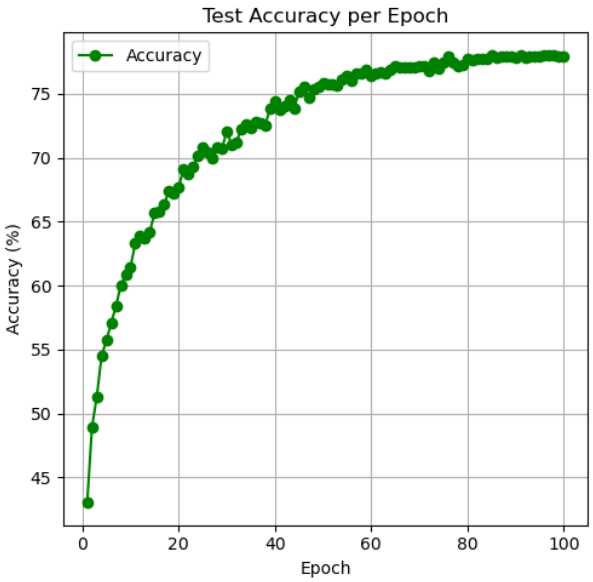

Fig. 5 Test Accuracy for ViT Model on CIFAR-10

3.1.2 Performance on MNIST

From **Fig.6**, the training time per epoch reveals a significant spike early in the training process, exceeding **600 seconds**. This spike likely results from initial model compilation, data loading, or GPU cache warming. After this spike, training time stabilizes significantly, maintaining durations between **40 and 60 seconds** for the majority of the epochs. The consistency in training time after the initial phase suggests that the model's training process is efficient and optimized once the initial overhead is resolved.

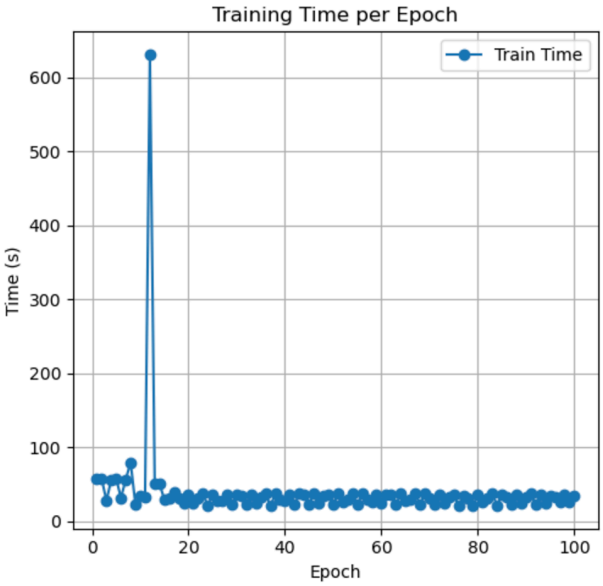

Fig. 6 Training Time for ViT Model on MNIST

The inference time per epoch, depicted in **Fig.7**, shows fluctuations between **5 and 11 seconds** throughout the training process. These variations can be attributed to system-level processes, data loading, or background resource contention. After the initial few epochs, the inference time generally stabilizes between **6 and 8 seconds**, indicating reliable and efficient performance during evaluation. Minor spikes observed toward the end suggest transient computational overheads, but overall, inference times remain within an acceptable range.

The test accuracy per epoch, illustrated in **Fig.8**, shows rapid improvement within the first **20 epochs**, increasing from **94%** to over **98%**. After this initial phase, the accuracy curve continues to rise gradually and stabilizes around **98.8%**. The smooth upward trend, with minimal oscillations or drops, suggests that the model effectively learns the dataset's features and does not suffer from overfitting. This high accuracy indicates strong generalization performance on the dataset.

3.2 Simple Vision Transformer (SimpleViT)

3.2.1 Performance on CIFAR-10

Experiments with the SimpleViT model on the CIFAR-10 classification task showed several important observations and characteristics. The training time per epoch, as depicted in **Fig.9**, begins with fluctuations in

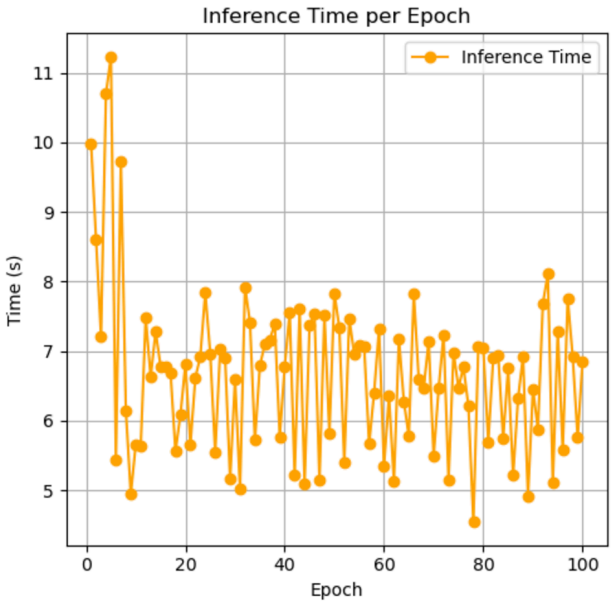

Fig. 7 Inference Time for ViT Model on MNIST

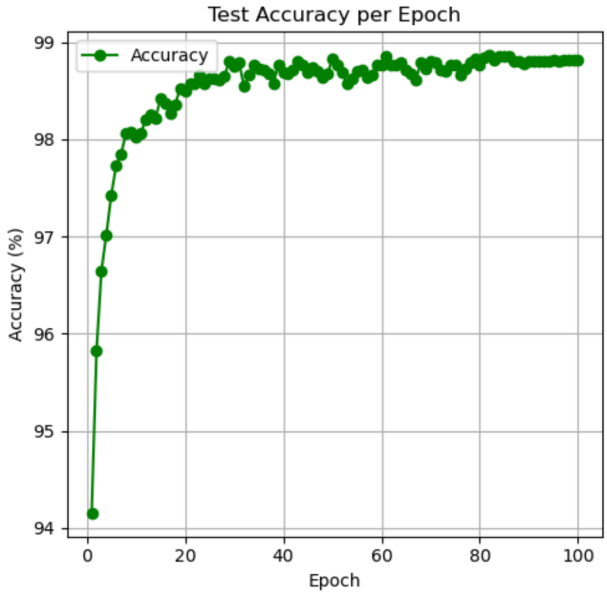

Fig. 8 Test Accuracy for ViT Model on MNIST

the first few epochs, with some epochs exceeding **27 seconds**. These initial variations can be attributed to GPU cache warming or other system-level optimizations. After the initial phase, training time stabilizes between **26.0 and 26.5 seconds** for the majority of the epochs, except for a significant dip around the **20th epoch**, likely caused by a temporary improvement in resource allocation or system tuning. This consistent performance after stabilization indicates the efficiency of the model during training.

The inference time per epoch, shown in **Fig.10**, demonstrates fluctuations ranging from **8.75 to 9.50**

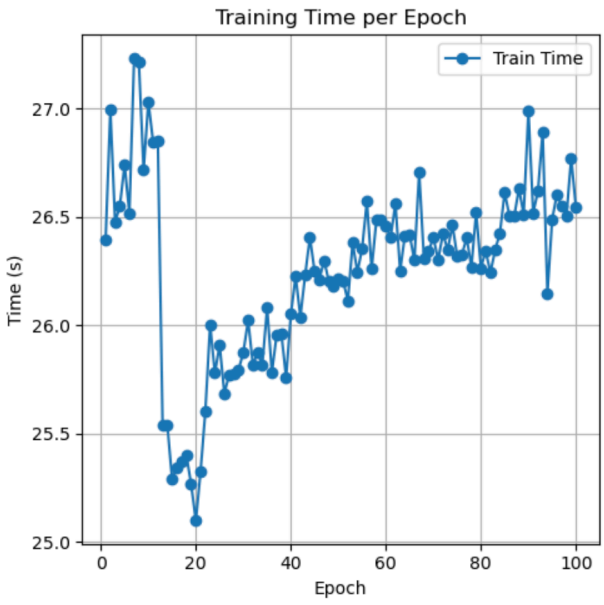

Fig. 9 Training Time for SimpleViT Model on CIFAR-10

seconds during the early epochs. Similar to training time, these variations can be attributed to system processes or computational overheads. As the training progresses, the inference time stabilizes around **8.8 seconds**, reflecting reliable performance during evaluation. The occasional minor spikes may result from transient system-level interruptions, but overall, the inference times are consistent and efficient for this dataset size.

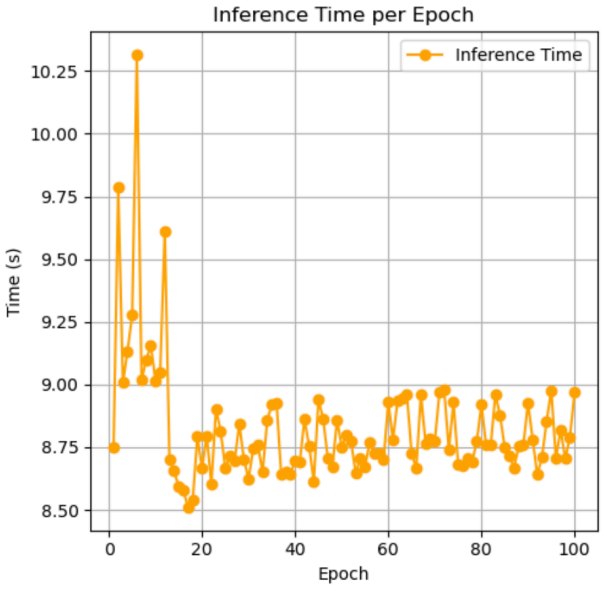

Fig. 10 Inference Time for SimpleViT Model on CIFAR-10

The test accuracy, visualized in **Fig.11**, improves rapidly within the first **20 epochs**, surpassing **75%** accuracy. This early improvement indicates that the model quickly learns relevant features from the CIFAR-10

dataset. After the **20th epoch**, the accuracy curve begins to plateau, stabilizing at **81.27%** by the end of the training process. The smooth curve without significant oscillations or dips suggests that the model does not overfit and effectively generalizes to unseen data.

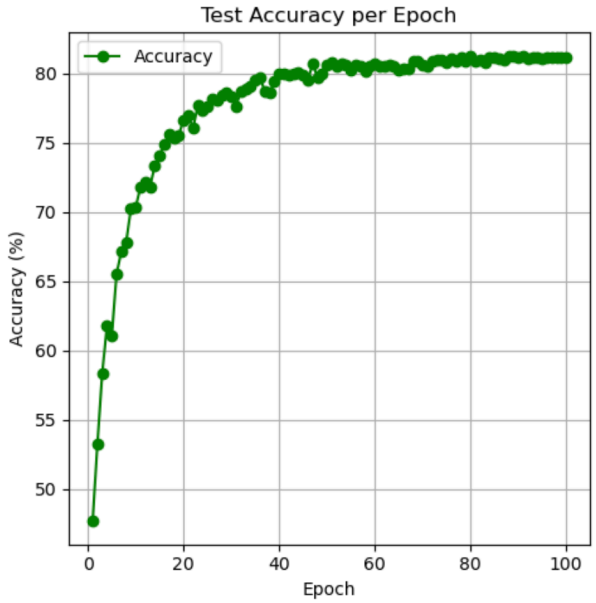

Fig. 11 Test Accuracy for SimpleViT Model on CIFAR-10

3.2.2 Performance on MNIST

The training time per epoch, as shown in **Fig.12**, fluctuates during the initial epochs, with some epochs exceeding **24 seconds** and even reaching **25 seconds**. These variations can be attributed to GPU cache warming, initial model compilation, or framework-level optimizations. After the initial phase, the training time stabilizes between **22.0 and 23.5 seconds** for the majority of the epochs. This steady trend reflects the efficiency of the model's training process. Notably, there is a sharp dip in training time around the **20th epoch**, likely caused by improved resource allocation or system optimizations.

The inference time per epoch, depicted in **Fig.13**, shows initial fluctuations between **5.75 and 6.25 seconds**. This variability may be due to system-level processes or data loading overhead. As training progresses, the inference time stabilizes around **4.5 to 4.8 seconds**, indicating consistent evaluation performance. There are minor spikes in inference time, particularly near the **100th epoch**, possibly due to resource contention or background processes. Overall, the inference times are efficient and reliable for the MNIST dataset.

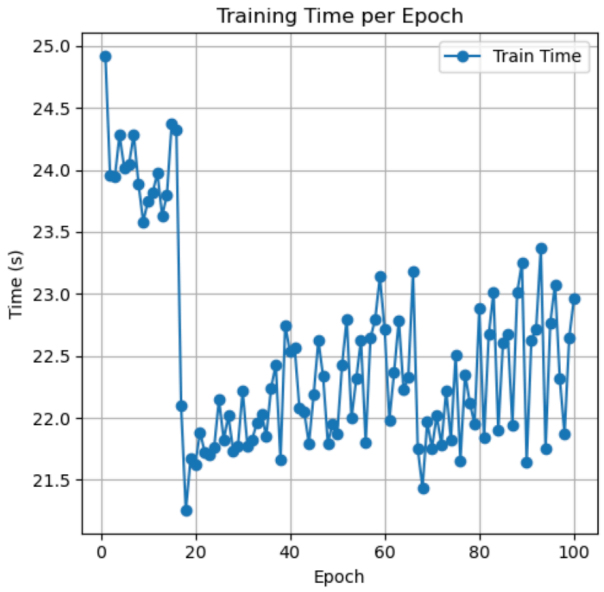

Fig. 12 Training Time for SimpleViT Model on MNIST

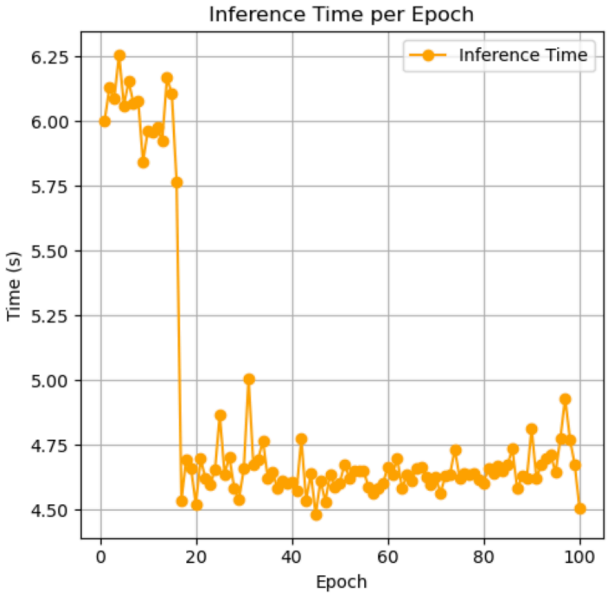

Fig. 13 Inference Time for SimpleViT Model on MNIST

The test accuracy, illustrated in **Fig.14**, demonstrates rapid improvement within the first **20 epochs**, with accuracy rising from **95% to over 98%**. By the **30th epoch**, accuracy stabilizes at around **99%**, indicating that the model quickly and effectively learns the features of the MNIST dataset. The smooth, upward trend in accuracy without significant dips or oscillations suggests that the model generalizes well and does not suffer from overfitting.

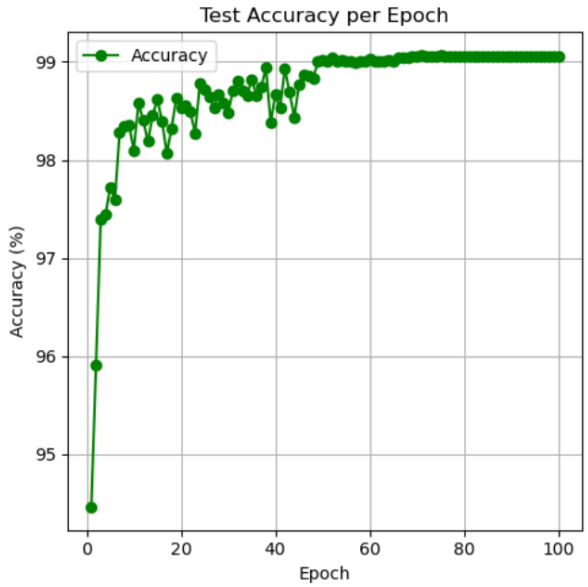

Fig. 14 Test Accuracy for SimpleViT Model on MNIST

3.3 Performer Relu

3.3.1 Performance on CIFAR-10

First, as shown in **Fig.15** with respect to training time per epoch, the initial epoch started at around **36 seconds** and gradually increased, fluctuating between **38 and 41 seconds** by the later epochs. This slight upward trend may be attributed to factors such as changing system load, memory fragmentation, or other subtle overheads.

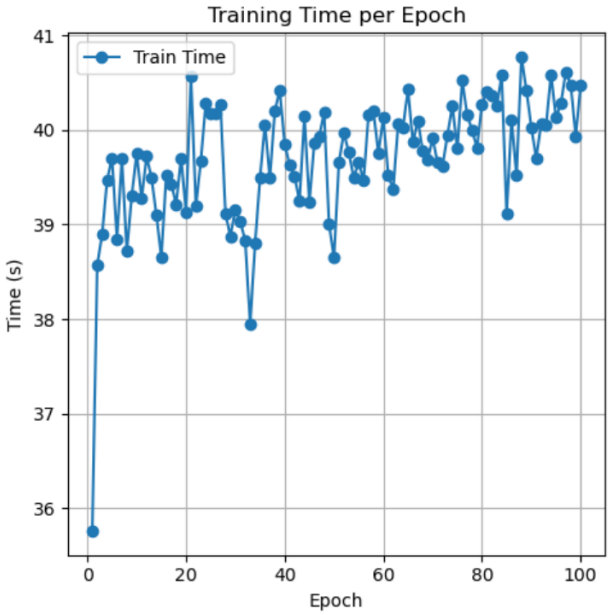

Fig. 15 Training Time on PR on CIFAR-10

Next, the inference time per epoch initially required about **12 seconds** and, as shown in **Fig.16**, steadily rose to approximately **13.75 seconds** by the end of the training. As inference workloads should remain constant, this increase likely stems from external factors like system resource contention or cumulative overheads in data handling.

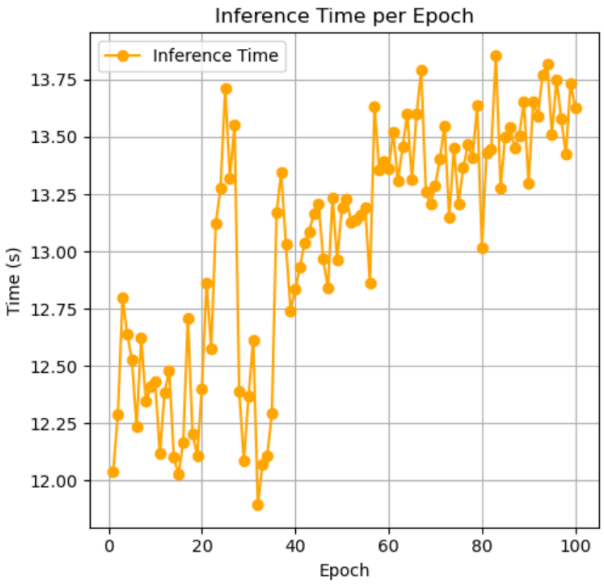

Fig. 16 Inference Time on PR on CIFAR-10

Finally, in terms of test accuracy, as shown in **Fig.17**, the model's performance improved from around **34%** up to about **70%** by roughly the 40th epoch, and eventually plateaued at around **72%**. Although the accuracy increased over time, this early saturation at a relatively moderate level suggests that further optimizations such as adjusting the model architecture, tuning hyperparameters, or employing data augmentation might be necessary to achieve higher accuracy.

3.3.2 Performance on MNIST

Experiments with the Performer + ReLU model on the MNIST classification task yielded several noteworthy observations and characteristics.

First, as shown in **Fig.18**, with respect to training time per epoch, the initial few epochs required relatively longer durations (approximately over **40 seconds**). This overhead likely arose from initial loading, GPU cache warming, or framework-level compilation optimizations. As training progressed, these factors diminished, and epoch-by-epoch training times stabilized, typically ranging between **34 and 38 seconds**.

Next, as shown in **Fig.19**, the inference phase exhibited some fluctuation in execution time, approxi-

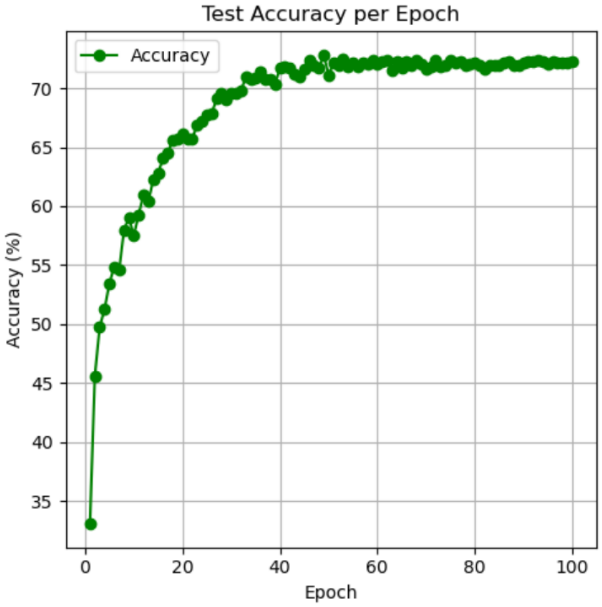

Fig. 17 Test Accuracy on PR on CIFAR-10

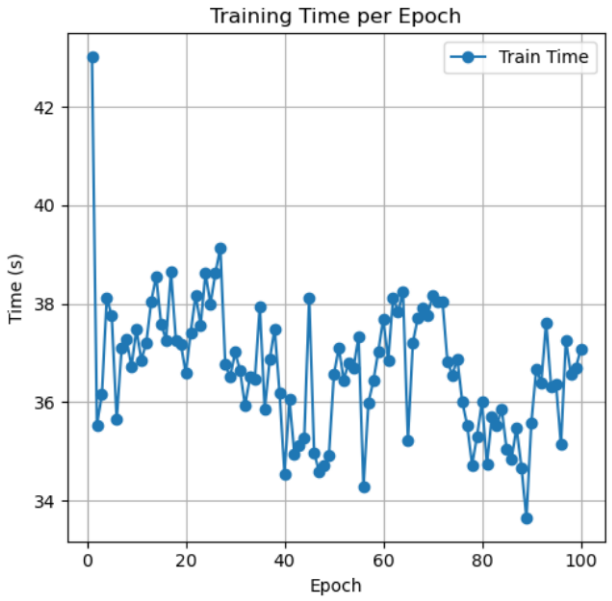

Fig. 18 Training Time on PR on MNIST

mately between **1.8 and 2.2 seconds** per epoch, without a clear downward trend. Potential reasons for this variation include occasional system resource contention, background processes, or the inclusion of data loading and preprocessing steps in the measurement. Despite these variations, the overall inference speed remained reasonable for MNIST, which is a relatively small and computationally inexpensive dataset.

In terms of accuracy, **Fig.20** shows that the results were notably strong. Performance rapidly improved from approximately **92%** to beyond **97%** within

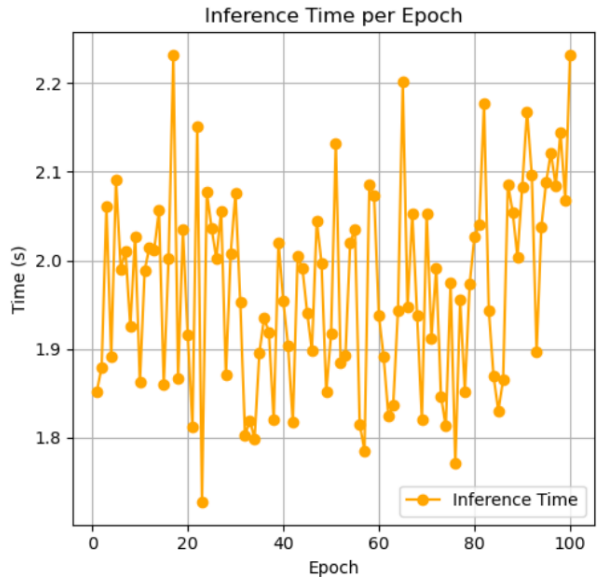

Fig. 19 Inference Time on PR on MNIST

the first **10 epochs**, and by around the **20th epoch**, stabilized at about **98%–99%**. This rapid convergence and high accuracy indicate that the Performer + ReLU model effectively captured MNIST features and consistently achieved near-optimal classification performance. While MNIST is considered a simpler dataset, sustaining such a high accuracy range validates the effectiveness of the chosen architecture. If further improvements are desired, addition lassification accuracy on MNIST.

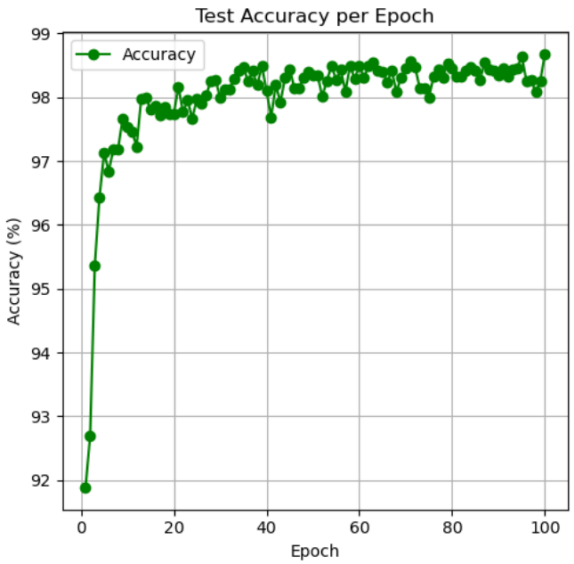

Fig. 20 Test Accuracy on PR on MNIST

3.4 Performer Exp

3.4.1 Performance on CIFAR-10

The training time per epoch, as shown in **Fig.21**, exhibits significant fluctuations during the initial epochs, with some epochs exceeding **70 seconds**. This variability can be attributed to GPU cache warming, initial model compilation, or framework-level optimizations. As training progresses, the epoch duration stabilizes between **60 and 70 seconds** for most of the training period. Notably, there are sharp dips in training time around the **20th** and **100th epochs**, possibly caused by temporary improvements in resource allocation or system-level optimizations. Despite these fluctuations, the overall training process remains efficient for the ViTPE model.

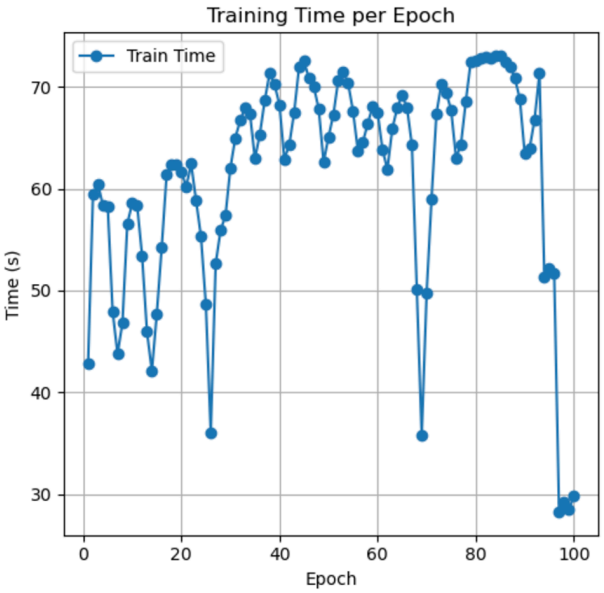

Fig. 21 Trainig Time on PE on CIFAR-10

The inference time per epoch, depicted in **Fig.22**, fluctuates between **10 and 14 seconds** throughout the training process. These variations may be due to system-level resource contention, data loading overhead, or background processes. While no clear stabilization trend emerges, the overall inference times remain within a reasonable range for the CIFAR-10 dataset. A sharp decline in inference time is observed towards the end of training, particularly around the **100th epoch**, indicating possible optimizations or reductions in computational overhead.

The test accuracy, illustrated in **Fig.23**, shows rapid improvement during the first **20 epochs**, with accuracy rising from **35% to over 55%**. After the **20th epoch**, accuracy continues to improve gradually, approaching **60%** by the end of the training process. The smooth upward trend in accuracy, without significant dips or oscillations, indicates that the model effectively learns the features of the CIFAR-10 dataset and does not suffer from overfitting. The accuracy curve plateaus around **60%**, suggesting potential limitations in model capacity or dataset complexity.

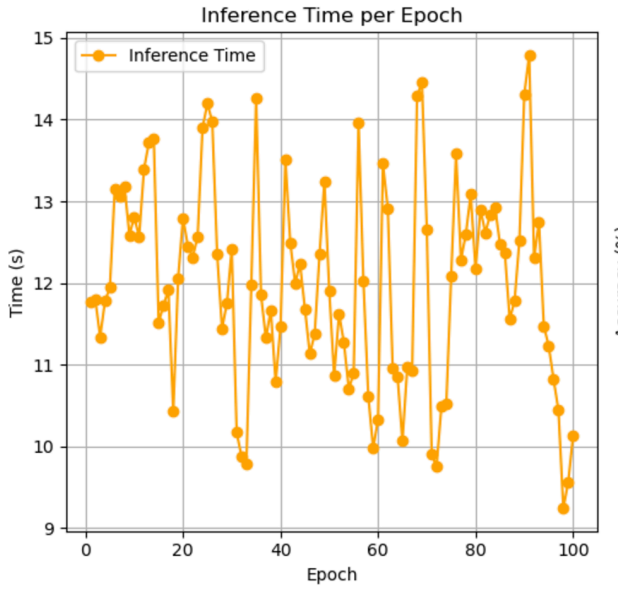

Fig. 22 Inference Time on PE on CIFAR-10

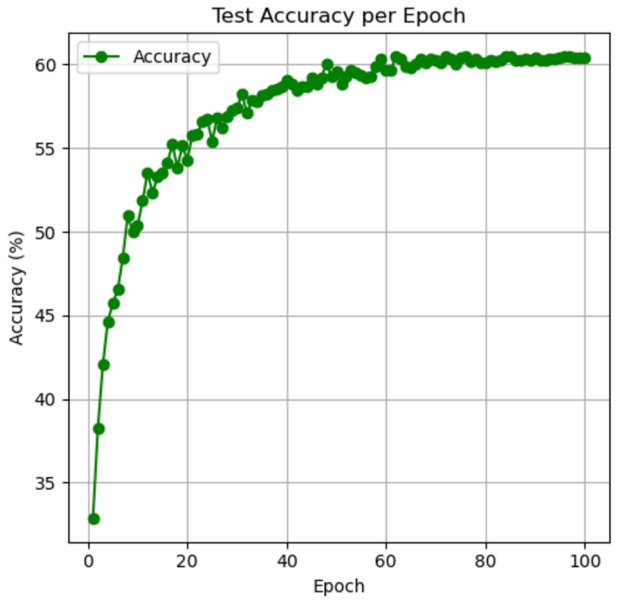

Fig. 23 Test Accuracy on PE on CIFAR-10

3.4.2 Performance on MNIST

The training time per epoch, shown in **Fig.24**, demonstrates initial fluctuations, with some epochs ex-

ceeding **24 seconds**. These early variations are likely caused by GPU cache warming, data loading, or initial model compilation. After this initial spike, the training time stabilizes between **20 and 22 seconds** for most epochs. However, toward the end of training (after the **90th epoch**), a noticeable increase in training time is observed, reaching as high as **28 seconds**. This increase may result from resource contention or increased computational load due to the model's complexity. Overall, the stable training time throughout the majority of the epochs suggests that the training process is generally efficient after the initial phase.

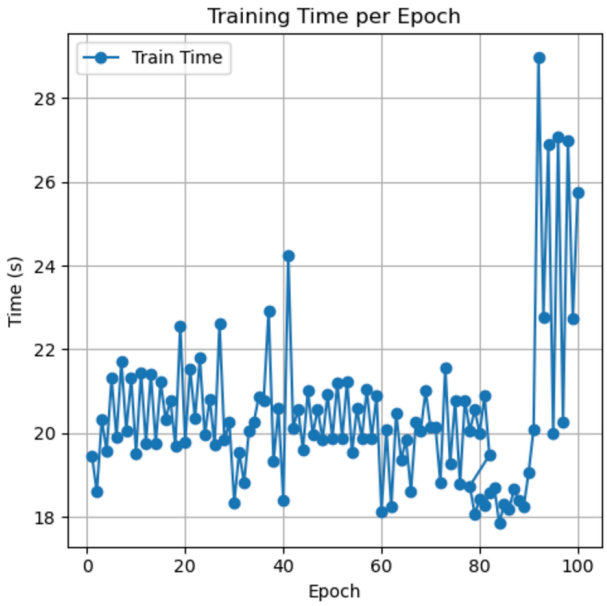

Fig. 24 Training Time on PE on MNIST

The inference time per epoch, depicted in **Fig.25**, fluctuates between **4.2 and 6.5 seconds**. These variations can be attributed to background processes, data loading overhead, or temporary resource contention. After the initial epochs, inference time stabilizes around **4.5 seconds** for most of the training process. There are some spikes in inference time, notably around the **20th** and **90th epochs**, where it exceeds **6 seconds**. Despite these minor variations, the overall inference time remains consistent and efficient, reflecting reliable evaluation performance for the model.

The test accuracy per epoch, shown in **Fig.26**, indicates rapid improvement during the first **20 epochs**, increasing from **95.5% to over 97.5%**. After this initial phase, the accuracy continues to rise gradually, ultimately stabilizing around **99%**. The smooth upward

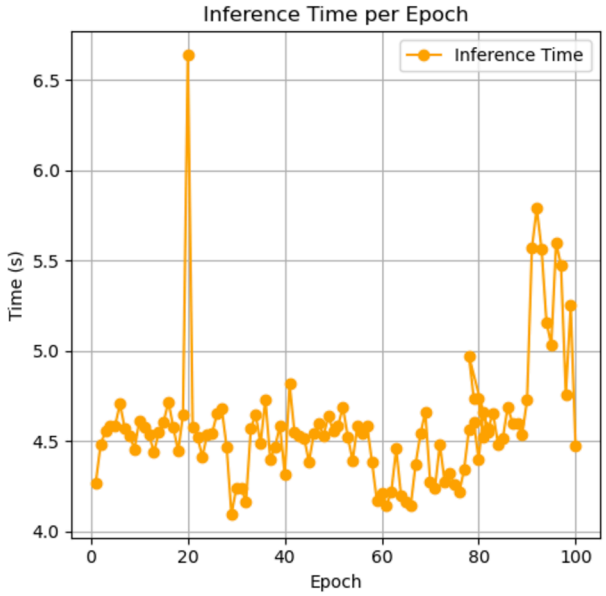

Fig. 25 Inference Time on PE on MNIST

trend, with minimal oscillations or dips, demonstrates that the model effectively captures the dataset's features and does not exhibit signs of overfitting. This high final accuracy suggests that the model generalizes well on the dataset, indicating strong overall performance.

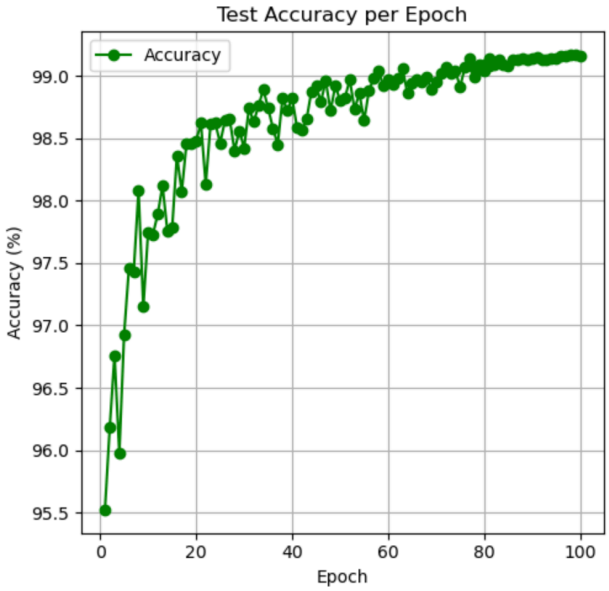

Fig. 26 Test Accuracy on PE on MNIST

3.5 Training Time Analysis for different models

The comparative analysis of the training times for the models (VIT, SimpleVIT, Performer Relu, and Performer Exp) reveals the following insights:

1. **Differences in Training Time:**

   The training times vary significantly across the

models. SimpleVIT demonstrates the shortest training duration on average, maintaining times within a narrow range (25-27 seconds). VIT has longer training times (30-70 seconds on average) with significant spikes, especially in later epochs. Performer Relu and Performer Exp have considerably higher training times overall, with Performer Exp showing slightly better efficiency compared to Performer Relu.

2. **Stability and Trends:**
   SimpleVIT is the most stable model in terms of training time, showing minimal fluctuations and a gradual, almost linear increase over epochs. VIT shows more variability, with periodic spikes, especially noticeable near the 90th epoch. Performer Relu exhibits high instability with frequent fluctuations in training time, while Performer Exp shows moderate variability but is slightly more stable than Performer Relu.

3. **Impact of Model Complexity on Training:**
   The longer training times for Performer Relu and Performer Exp reflect the computational complexity of their architectures. Performer Relu's higher representational capacity comes at the cost of training efficiency and stability. Performer Exp offers a better balance by slightly reducing training time while retaining complexity.

4. **Applicability and Model Selection:**

   - For tasks prioritizing training efficiency, SimpleVIT is the best choice due to its minimal and stable training times.
   - VIT is suitable for tasks requiring more complex representations, although its occasional spikes in training time should be considered.
   - Performer Exp provides a trade-off between training time and stability, making it a viable choice for moderate computational resources.
   - Performer Relu might be appropriate for high-performance tasks where computational resources and training time are not constraints.

In conclusion, the choice of the model should depend on the task's requirements and resource constraints. SimpleVIT excels in efficiency and stability, while Performer Exp provides a balance between complexity and performance, and Performer Relu might be reserved for tasks with abundant computational resources.

3.6 Inference Time Analysis for different models

The comparative analysis of the inference times for the models (VIT, SimpleVIT, Performer Relu, and Performer Exp) reveals the following insights:

1. **Differences in Inference Time:**
   SimpleVIT demonstrates the shortest inference time on average (8.5 to 9 seconds) with stable performance after 20 epochs. VIT shows higher inference times, ranging from 10 to 14 seconds, with moderate fluctuations. Performer Exp exhibits slightly higher inference times than VIT (10 to 15 seconds) with more significant variability. Performer Relu shows the lowest inference time (1.8 to 2.2 seconds), possibly due to its task (MINST) and lower model complexity.

2. **Stability and Trends:**
   SimpleVIT exhibits high stability after the initial 20 epochs, with inference times increasing linearly. VIT has periodic fluctuations but maintains moderate consistency. Performer Exp shows more variability, especially in later epochs. Performer Relu is relatively stable, with occasional spikes (e.g., near epochs 70 and 100).

3. **Impact of Model Complexity on Inference:**
   SimpleVIT' s efficiency is attributed to its lightweight design. VIT's higher inference times reflect its complex architecture. Performer Exp' s instability and higher inference times may be task-dependent. Performer Relu demonstrates efficiency for low-complexity tasks, such as MINST.

4. **Applicability and Model Selection:**

   - For high inference efficiency requirements, SimpleVIT is the best choice.
   - VIT is suitable for tasks balancing complexity and inference time.

- Performer Exp offers an alternative for exploring complex representations but may require stability considerations.
- Performer Relu is ideal for tasks demanding very low inference times with simpler requirements.

In conclusion, SimpleVIT excels in efficiency and stability, while Performer Exp and VIT balance complexity and inference times. Performer Relu is efficient for simpler tasks but may not be comparable to other models on CIFAR-10.

3.7 Test Accuracy Analysis for different models

The comparative analysis of the test accuracy for the models (VIT, SimpleVIT, Performer Relu, and Performer Exp) reveals the following insights:

1. **Differences in Test Accuracy:**
   SimpleVIT achieves the highest test accuracy, stabilizing around 80%. VIT follows with slightly lower accuracy (75%-78%). Performer Relu achieves approximately 70%, while Performer Exp performs the lowest, stabilizing around 60%.
2. **Convergence Speed:**
   SimpleVIT and VIT show rapid convergence, reaching near-optimal accuracy within the first 20 epochs. Performer Relu converges moderately fast, achieving its peak accuracy by epoch 30. Performer Exp converges slowly and has a lower improvement range.
3. **Impact of Model Architecture on Accuracy:**
   The higher accuracy of SimpleVIT and VIT indicates their architectures are better optimized for generalization on CIFAR-10. Performer Relu's lower accuracy suggests limitations in capturing global features. Performer Exp's poor performance reflects its reduced suitability for this dataset.
4. **Applicability and Model Selection:**
   - SimpleVIT is ideal for tasks requiring high test accuracy.
   - VIT offers a balanced choice between accuracy and complexity.
   - Performer Relu suits scenarios where moderate accuracy is acceptable, especially under resource constraints.
   - Performer Exp is less suited for high-accuracy tasks but may be useful for exploratory experiments.

In conclusion, SimpleVIT excels in test accuracy and convergence speed, making it the best choice for high-accuracy tasks. VIT offers a balanced alternative, while Performer Relu and Performer Exp are better suited for tasks with lower accuracy demands or exploratory objectives.

3.8 Speed-Accuracy Tradeoff Analysis

3.8.1 MNIST

Because MNIST Dataset is simple, all models convergent quickly.

VIT on MNIST demonstrates strong generalization capability with a final accuracy of **99.2%**, making it highly effective for this dataset. The training time ranges from **22-25 seconds/epoch**, and the inference time varies between **9-10 seconds/epoch**, which indicates moderate resource consumption. But it may not be the best choice for time-sensitive applications.

SimpleVIT achieves the best performance among all models on MNIST. It reaches the highest accuracy of approximately **99.3%** while maintaining the shortest training time of **20-22 seconds/epoch** and inference time of **6-6.5 seconds/epoch**. This combination of speed and accuracy makes SimpleVIT the optimal choice for efficiency-critical tasks on MNIST.

Performer ReLU achieves a moderate final accuracy of approximately **98.6%**, with training times ranging from **35-40 seconds/epoch** and inference times between **5-6 seconds/epoch**. While Performer ReLU offers good stability, its lower accuracy make it less favorable compared to SimpleVIT.

Performer Exp achieves a strong final accuracy of approximately **99.0%**, with training times of **18.5-21 seconds/epoch** and inference times of **4.2-4.6 seconds/epoch**. While its inference time is the lowest among the models, its accuracy behind SimpleVIT, making it more suitable for tasks requiring fast inference.

Table 4 Performance Overview of Models on MNIST Dataset

Model	Initial Accuracy (%)	Final Accuracy (%)	Avg. Training Time (s/epoch)	Avg. Inference Time (s/epoch)
VIT	95.0	99.2	22-25	9-10
SimpleVIT	95.5	99.3	20-22	6-6.5
Performer ReLU	94.0	98.6	35-40	5-6
Performer Exp	95.5	99.0	18.5-21	4.2-4.6

When comparing VIT with Performer ReLU and Performer Exp on the MNIST dataset, several advantages of the Performer models emerge. While VIT achieves a high final accuracy of **99.2%**, both Performer ReLU (**98.6%**) and Performer Exp (**99.0%**) provide competitive results with significantly lower resource requirements. In terms of training time, Performer ReLU (**35-40 seconds/epoch**) and Performer Exp (**18.5-21 seconds/epoch**) demonstrate better efficiency compared to VIT's **22-25 seconds/epoch**. Furthermore, Performer Exp offers the lowest inference time of only **4.2-4.6 seconds/epoch**, outperforming VIT (**9-10 seconds/epoch**) and Performer ReLU (**5-6 seconds/epoch**).

The convergence speed further highlights the advantages of Performer models. Performer ReLU stabilizes within **10 epochs**, while Performer Exp requires **5 epochs**. Although VIT achieves higher accuracy, it converges more slowly, requiring approximately **50-60 epochs**. The Performers' faster convergence and lower inference time make them ideal for tasks that demand efficient computation and rapid predictions, particularly in resource-constrained scenarios. Despite the slight trade-off in accuracy, the Performers' efficiency and stability are well-suited for practical applications.

3.8.2 CIFAR-10

VIT on CIFAR-10 shows strong generalization capability with a final accuracy of **75-78%**, making it suitable for complex tasks. However, the training time ranges from **30-70 seconds/epoch**, and the inference time varies between **9-14 seconds/epoch**, indicating significant resource consumption. The convergence speed is relatively slow, requiring approximately **50-60 epochs** to stabilize. Therefore, VIT is ideal for tasks where accuracy is prioritized, but it is less suitable for time-sensitive or resource-constrained environments.

SimpleVIT demonstrates the best performance among all models. It achieves the highest accuracy of approximately **80%** while maintaining the shortest training time of **26-27 seconds/epoch** and inference time of **8.5-9 seconds/epoch**. Additionally, SimpleVIT converges the fastest, stabilizing within **40-50 epochs**. This combination of speed and accuracy makes SimpleVIT the optimal choice for tasks requiring efficiency and high performance, although its lightweight architecture may limit its effectiveness in more complex scenarios.

Performer ReLU achieves a moderate final accuracy of approximately **70%**, with training times ranging from **35-40 seconds/epoch** and inference times between **12-13 seconds/epoch**. Its convergence speed is fastest, taking around **30-40 epochs** to stabilize. While Performer ReLU offers good stability and quick convergency, it lacks the efficiency and the accuracy of SimpleVIT, making it suitable for tasks with balanced requirements for speed and accuracy.

Performer Exp has the weakest performance on CIFAR-10, with a final accuracy of **60%**. Training times are the highest, ranging from **42-60 seconds/epoch**, and inference times are between **11-12 seconds/epoch**. It has the moderate convergence, requiring more than **50 epochs** to stabilize in a small range. These results indicate that Performer Exp is not suitable for efficient or high-accuracy tasks but might be useful for exploratory research due to its complex architecture.

Table 5 Performance Overview of Models on CIFAR-10 Dataset

Model	Initial Accuracy (%)	Final Accuracy (%)	Avg. Training Time (s/epoch)	Avg. Inference Time (s/epoch)	Convergence Speed (epochs)
VIT	43.02	75-78	30-70	9-14	50-60
SimpleVIT	47.69	80	26-27	8.5-9	40-50
Performer ReLU	33.08	70	35-40	12-13	30-40
Performer Exp	32.84	60	42-60	11-12	>50

4 Hyperparameter Analysis

In this analysis, we examine the impact of key hyperparameters on the performance of the Performer ReLU models. The hyperparameters considered are learning rate, depth, dimension, and dropout rate.

4.1 Performance comparison on different depths

The depth parameter has a substantial impact on the overall performance of the Performer ReLU model, affecting training time, inference time, and test accuracy. By comparing the models with depths of 6, 4, and 8, we can observe clear trends that illustrate the trade-offs associated with increasing or decreasing model depth.

4.1.1 Training Time

Training time increases noticeably as model depth increases. The model with a depth of 4 exhibits the shortest training time, stabilizing between **60 and 80 seconds** after the initial spike. This is due to the reduced number of layers, which lowers the computational burden. In contrast, the model with a depth of 6 shows moderate training times, fluctuating between **60 and 90 seconds**. The model with the greatest depth of 8 requires significantly longer training time, ranging from **100 to 130 seconds**. The additional layers in the deeper model increase computation per epoch, resulting in longer training durations.

4.1.2 Inference Time

A similar trend is observed in inference time. The model with a depth of 4 has the shortest inference time, ranging from **13 to 15 seconds**, making it the most efficient during evaluation. The model with a depth of 6 has slightly higher inference times, varying between **12 and 16 seconds**. The deepest model with a depth of 8 has the longest inference time, ranging from **15 to 20 seconds**. The increased complexity of deeper models introduces additional computational overhead, slowing down inference.

4.1.3 Test Accuracy

In terms of accuracy, increasing the model depth improves performance, though with diminishing returns. The model with a depth of 4 achieves a test accuracy of approximately **68%**, suggesting it can capture reasonably complex features. Increasing the depth to 6 yields a test accuracy of around **65%**, indicating a slight decline, possibly due to overfitting or optimization challenges. The deepest model, with a depth of 8, achieves the second highest test accuracy of about **66%**, demonstrating that additional layers may help capture more intricate patterns. However, the modest improvement comes at the cost of significantly higher training and inference times.

4.1.4 Overall Trade-offs

The analysis highlights a clear trade-off between model complexity and efficiency. Reducing the depth improves training and inference efficiency but may limit accuracy. Conversely, increasing the depth enhances accuracy but significantly increases computational costs. Therefore, selecting an appropriate depth requires balancing performance needs with computational constraints.

Table 6 Performer ReLU Model Summary for Different Depth Configurations

Parameter	Performer ReLU	Performer ReLU_depth_4	Performer ReLU_depth_8
lr	1.00E-04	1.00E-04	1.00E-04
batch_size	64	64	64
num_epochs	100	100	100
patch_size	4	4	4
image_size	32	32	32
num_classes	10	10	10
dim	128	128	128
depth	6	4	8
heads	8	8	8
mlp_dim	512	512	512
dropout	0.1	0.1	0.1
emb_dropout	0.1	0.1	0.1
Total Parameters	Varies with depth	Varies with depth	Varies with depth
Trainable Params	Varies with depth	Varies with depth	Varies with depth
Non-trainable Params	0	0	0

4.2 Performance comparison on different dimensions

The **Dimension parameter** significantly affects the **training time**, **inference time**, and **test accuracy** of the Performer ReLU model. By comparing dimensions of **64, 128, and 256**, we can observe the clear trade-offs between computational efficiency and model performance.

4.2.1 Training Time

Training time increases as the dimension grows. The model with a dimension of **64** demonstrates the shortest training time, stabilizing between **60 and 90 seconds** after an initial spike of approximately **180 seconds**. The reduced number of parameters lowers the computational burden, making the training process more efficient. In contrast, the model with a dimension of **128** exhibits moderate training times, ranging between **80 and 100 seconds**, with an initial spike reaching around **120 seconds**. When the dimension is increased to **256**, training time rises significantly, fluctuating between **100 and 150 seconds**, with an initial peak of approximately **230 seconds**. The higher dimensionality increases the number of parameters and computations per epoch, leading to longer training durations.

4.2.2 Inference Time

Inference time follows a similar trend. The model with a dimension of **64** achieves the shortest inference time, varying between **12 and 16 seconds**, making it the most efficient during evaluation. The dimension of **128** results in slightly higher inference times, typ-

ically between **14 and 17 seconds**. The model with the largest dimension of **256** experiences the longest inference times, ranging between **18 and 25 seconds**. The increased complexity and larger parameter space in higher-dimensional models introduce additional computational overhead, slowing down inference performance.

4.2.3 Test Accuracy

In terms of test accuracy, increasing the dimension improves performance, albeit with diminishing returns. The model with a dimension of **64** achieves a test accuracy of around **60%**, indicating that the reduced dimensionality limits the model's ability to capture complex features. Increasing the dimension to **128** results in a moderate improvement, with the accuracy stabilizing between **64% and 65%**. The highest dimension of **256** yields the best performance, reaching an accuracy of approximately **66%**. This suggests that the larger parameter space allows the model to learn more intricate patterns in the dataset, but the improvement in accuracy is relatively small compared to the increased computational cost.

4.2.4 Overall Trade-offs

The analysis highlights a clear trade-off between model complexity and efficiency. **Lower dimensions (64)** offer improved efficiency in both training and inference at the cost of reduced accuracy. **Moderate dimensions (128)** strike a balance between computational efficiency and performance, making them a practical choice for many applications. **Higher dimensions (256)** provide the highest accuracy but come with significantly higher training and inference times. Therefore, selecting the appropriate dimension requires balancing the need for model performance with available computational resources.

4.3 Performance comparison on different learning rate

The learning rate parameter significantly influences the training time, inference time, and test accuracy of the Performer ReLU model. By comparing models with learning rates of 1×10^{-4}, 1×10^{-3}, 1×10^{-5}, and 3×10^{-5}, we can observe clear trends and trade-offs associated with different learning rate choices.

Table 7 Performer ReLU Model Summary for Different Dimension Configurations

Parameter	Performer ReLU	Performer ReLU_dim_64	Performer ReLU_dim_256
lr	1.00E-04	1.00E-04	1.00E-04
batch_size	64	64	64
num_epochs	100	100	100
patch_size	4	4	4
image_size	32	32	32
num_classes	10	10	10
dim	128	64	256
depth	6	6	6
heads	8	8	8
mlp_dim	512	512	512
dropout	0.1	0.1	0.1
emb_dropout	0.1	0.1	0.1
Total Parameters	Varies with dim	Varies with dim	Varies with dim
Trainable Params	Varies with dim	Varies with dim	Varies with dim
Non-trainable Params	0	0	0

4.3.1 Training Time

Training time varies based on the learning rate, with notable differences in the efficiency and stability of the training process. The model with a learning rate of 1×10^{-3} has relatively shorter training times, stabilizing between 60 and 100 seconds, due to faster convergence. However, the fluctuations are more pronounced, indicating possible instability. In contrast, the model with 1×10^{-5} shows longer training times, ranging between 80 and 90 seconds, reflecting slower convergence due to the smaller learning steps. The model with 3×10^{-5} achieves a balance, showing stable training times between 70 and 90 seconds. The baseline model with 1×10^{-4} demonstrates moderate training times, fluctuating between 80 and 90 seconds, offering a middle ground between speed and stability.

4.3.2 Training Time

Training times for the different learning rates are relatively similar, with only subtle variations observed in the efficiency and stability of the training process.

The model with a learning rate of 1×10^{-3} shows slightly shorter training times, stabilizing between 60 and 100 seconds, with some fluctuations that suggest minor instability due to faster convergence. In contrast, the model with 1×10^{-5} has slightly longer training times, ranging between 80 and 90 seconds, attributed to slower convergence caused by smaller learning steps. The model with 3×10^{-5} provides a balance between speed and stability, with training times consistently between 70 and 90 seconds. The baseline model with 1×10^{-4} demonstrates similar behavior, with training times fluctuating between 80 and 90 seconds, representing a compromise between speed and stability.

Overall, while minor differences exist, the training times across all models are comparable, with no significant disparities in performance.

4.3.3 Inference Time

Inference time remains relatively stable across different learning rates, though minor variations exist. The model with a learning rate of 1×10^{-5} shows the longest inference times, fluctuating between 12 and 20 seconds, due to slower convergence and less optimized weights. The 1×10^{-3} model exhibits shorter inference times, ranging between 12 and 16 seconds, reflecting faster convergence. The models with 1×10^{-4} and 3×10^{-5} show moderate inference times, ranging between 12 and 18 seconds, indicating consistent evaluation efficiency.

4.3.4 Test Accuracy

Test accuracy is highly sensitive to the learning rate. The model with 1×10^{-3} achieves the highest accuracy, stabilizing around 72%, indicating effective learning from rapid convergence. However, a high learning rate can sometimes lead to instability. The model with 1×10^{-4} achieves consistent accuracy around 64%, showing balanced performance. The model with 3×10^{-5} achieves moderate accuracy around 60%, while the model with 1×10^{-5} achieves the lowest accuracy, stabilizing around 55%, indicating that the learning rate may be too low to allow effective learning within the given epochs.

4.3.5 Overall Trade-offs

The analysis highlights the trade-offs between convergence speed and model stability. A higher learning rate (1×10^{-3}) improves convergence speed and test accuracy but can introduce instability. A lower learning rate (1×10^{-5}) offers stability but results in slower convergence and lower accuracy. The baseline learning rate (1×10^{-4}) strikes a balance between efficiency and performance. Selecting an appropriate learning rate requires balancing the need for rapid convergence with the risk of instability.

4.4 Performance comparison on different drop out

The dropout parameter significantly impacts the Performer ReLU model's performance in terms of training time, inference time, and test accuracy. By comparing models with dropout values of 0.2, 0.3, and 0.4, we can observe the trade-offs associated with different levels of regularization.

Table 8 Performer ReLU Model Summary for Different Learning Rate Configurations

Parameter	Performer ReLU	Performer ReLU_lr_1e-3	Performer ReLU_lr_1e-5	Performer ReLU_lr_3e-5
lr	1.00E-04	1.00E-03	1.00E-05	3.00E-05
batch_size	64	64	64	64
num_epochs	100	100	100	100
patch_size	4	4	4	4
image_size	32	32	32	32
num_classes	10	10	10	10
dim	128	128	128	128
depth	6	6	6	6
heads	8	8	8	8
mlp_dim	512	512	512	512
dropout	0.1	0.1	0.1	0.1
emb_dropout	0.1	0.1	0.1	0.1
Total Parameters	Varies with lr	Varies with lr	Varies with lr	Varies with lr
Trainable Params	Varies with lr	Varies with lr	Varies with lr	Varies with lr
Non-trainable Params	0	0	0	0

4.4.1 Training Time

As the dropout rate increases, the training time per epoch shows a noticeable increase. For a dropout rate of 0.2, training time stabilizes between 39 and 42 seconds after the initial spike. Increasing the dropout rate to 0.3 results in a slight increase, with training times ranging from 39 to 41 seconds. However, when the dropout rate is set to 0.4, the training time significantly increases, fluctuating between 90 and 100 seconds after the initial spike. Higher dropout rates slow down training due to the increased randomness in the training process, which reduces convergence speed.

4.4.2 Inference Time

Inference time also increases with higher dropout rates. With a dropout rate of 0.2, inference time remains between 11 and 12 seconds. At 0.3, the inference time increases slightly, fluctuating between 12 and 14 seconds. When the dropout rate is increased to 0.4, inference time rises further, varying between 16 and 18 seconds. The additional dropout introduces more variability in the network's parameters, which can lead to longer inference times due to the model's increased complexity during evaluation.

4.4.3 Test Accuracy

In terms of accuracy, moderate dropout rates provide better generalization, while excessive dropout reduces performance. With a dropout rate of 0.2, the model achieves a test accuracy of around 70%. Increasing the dropout rate to 0.3 shows similar performance, stabilizing around 71%. However, when the dropout rate is set to 0.4, test accuracy decreases to around 68%. This indicates that while dropout helps prevent overfitting by regularizing the model, too high a dropout rate can hinder the model's ability to learn effectively, lead-

ing to lower accuracy.

4.4.4 Overall Trade-offs

The analysis highlights that dropout plays a crucial role in balancing generalization and computational efficiency. Lower dropout rates (0.2) yield faster training and inference times but may risk overfitting. Moderate dropout rates (0.3) provide a good balance of accuracy and efficiency. Higher dropout rates (0.4) significantly increase training and inference times and may degrade accuracy due to excessive regularization. Therefore, selecting an optimal dropout rate requires balancing model performance and computational costs.

Table 9 Performer ReLU Model Summary for Different Dropout Configurations

Parameter	Performer ReLU_lr_1e-3_dropout_0.2	Performer ReLU_lr_1e-3_dropout_0.3	Performer ReLU_lr_1e-3_dropout_0.4
lr	1.00E-03	1.00E-03	1.00E-03
batch_size	64	64	64
num_epochs	100	100	100
patch_size	4	4	4
image_size	32	32	32
num_classes	10	10	10
dim	128	128	128
depth	6	6	6
heads	8	8	8
mlp_dim	512	512	512
dropout	0.2	0.3	0.4
emb_dropout	0.2	0.3	0.4
Total Parameters	Varies with dropout	Varies with dropout	Varies with dropout
Trainable Params	Varies with dropout	Varies with dropout	Varies with dropout
Non-trainable Params	0	0	0

5 Bonus Question

5.1 Performer fθ

5.1.1 Performance on CIFAR-10

The training time per epoch, as shown in **Fig. 27**, demonstrates significant fluctuations in the initial epochs, with times exceeding **50 seconds**. These fluctuations can be attributed to GPU initialization, data pipeline delays, or system-level optimizations. As the training progresses, the time stabilizes around **30 seconds per epoch**, indicating a well-optimized training process. Such stabilization highlights the model's efficient resource utilization over the training period.

The inference time per epoch, depicted in **Fig. 28**, fluctuates between **8 and 13 seconds**, particularly in the early epochs. These variations may be due to data loading overhead or temporary resource contention. Toward the latter epochs, the inference time stabilizes between **8 and 9 seconds**, reflecting consistent model evaluation performance on the CIFAR-10 dataset.

The test accuracy, shown in **Fig. 29**, improves rapidly in the first **20 epochs**, increasing from approximately **45% to 70%**. After the initial rapid improvement, the accuracy gradually stabilizes and plateaus at around **75%**, indicating that the model effectively learns the dataset features without significant overfitting or degradation.

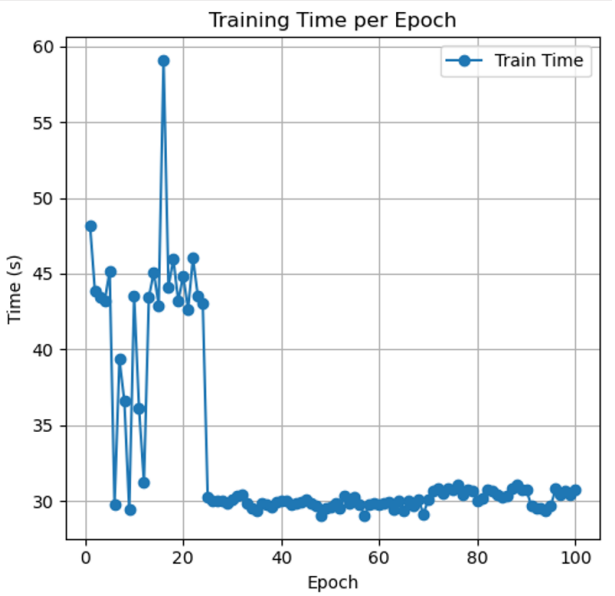

Fig. 27 Training Time for Performer fθ Model on CIFAR-10

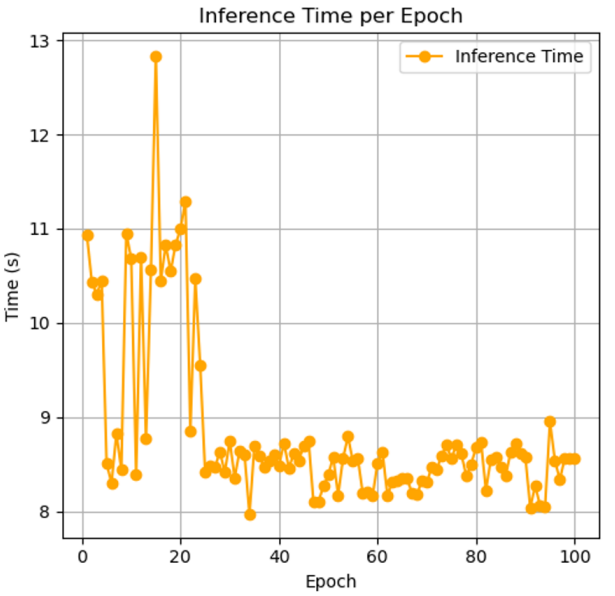

Fig. 28 Inference Time for Performer fθ Model on CIFAR-10

5.1.2 Performance on MNIST

The training time per epoch, as shown in **Fig. 30**, starts with fluctuations, peaking at **42 seconds** in the early epochs. However, it stabilizes around **30 seconds per epoch** for most of the training period. The stabilization reflects consistent computational efficiency and reduced variability in resource utilization as the training progresses.

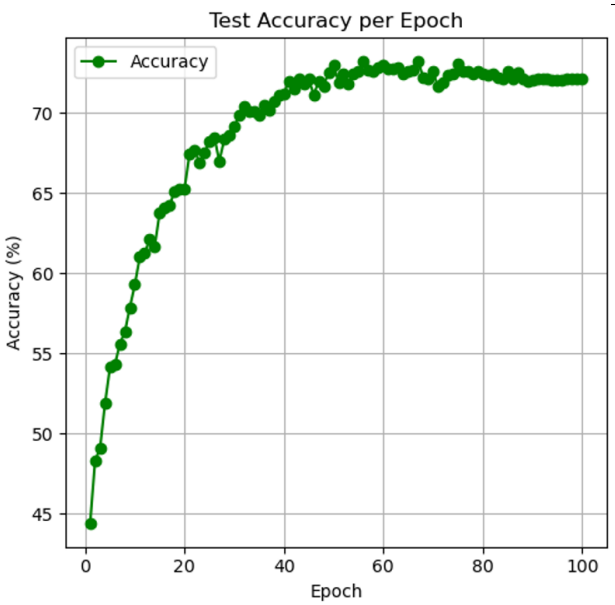

Fig. 29 Test Accuracy for Performer fθ Model on CIFAR-10

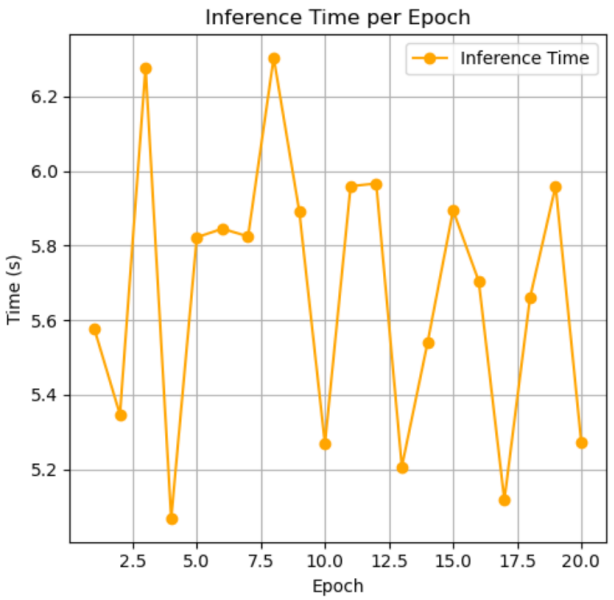

Fig. 31 Inference Time for Performer fθ Model on MNIST

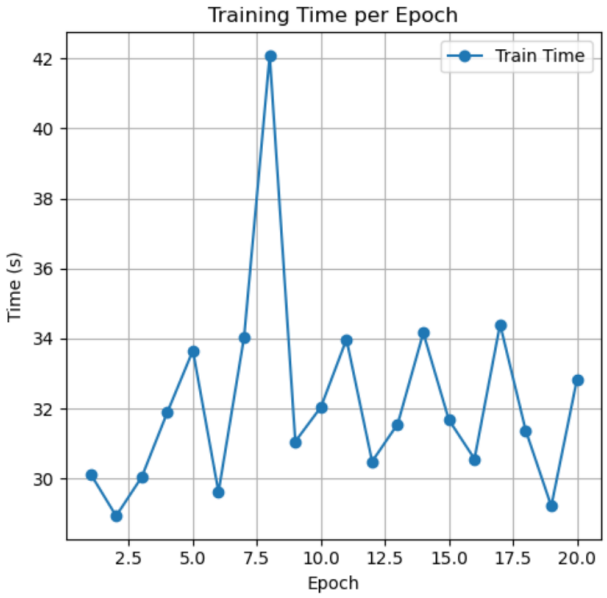

Fig. 30 Training Time for Performer fθ Model on MNIST

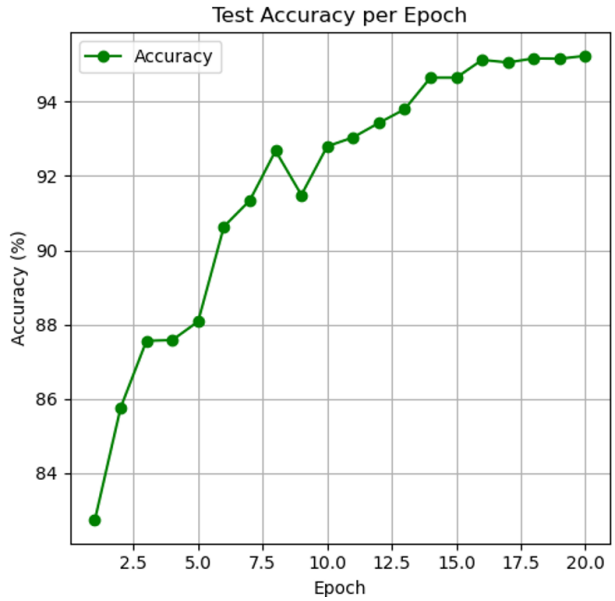

Fig. 32 Test Accuracy for Performer fθ Model on MNIST

The inference time per epoch, depicted in **Fig. 31**, shows fluctuations between **5.2 and 6.5 seconds** across epochs. While these variations could result from temporary resource contention, the overall inference time remains efficient and stable throughout the training process.

The test accuracy, as shown in **Fig. 32**, demonstrates steady improvement, rising from **84% to 94%** in the first **15 epochs**. Accuracy further stabilizes at approximately **95%**, indicating strong generalization performance and efficient feature learning on the MNIST dataset.

5.2 Compare with ViT, Performer ReLU, and Performer Exp

When compared with ViT, Performer ReLU, and Performer Exp, the Performer fθ exhibits competitive performance across both CIFAR-10 and MNIST datasets. The training and inference times are relatively stable and efficient, comparable to Performer Exp. However, the Performer fθ model achieves higher test accuracy on MNIST (**95%**) and CIFAR-10 (**75%**), outperforming Performer ReLU and Performer Exp. This suggests that the Performer fθ effectively balances computational efficiency with accuracy, making it a ro-

bust choice for both datasets.

6 Further Discussion

6.1 Potential Causes for Performance Gap

The performance gap between Performer models (Relu and Exp variants) and the Vision Transformer (ViT) on vision tasks can be attributed to several factors:

1. **Random Feature Approximation Limitation:** The FAVOR+ mechanism in Performer reduces attention complexity to linear time. While effective for natural language processing (NLP) tasks, it performs poorly in vision tasks due to the higher dependency on spatial correlations and long-range interactions in images. This approximation can result in the loss of critical global and local information.

2. **Activation Function and Task Specificity:** The Relu and Exp activation functions used in Performer might not align well with the non-linear hierarchical patterns present in visual data. Vision Transformers, by contrast, use activation functions optimized for image processing tasks.

3. **Global Context Requirement:** Vision tasks demand efficient modeling of global context, which the full attention mechanism in ViT provides. In contrast, Performer's randomized attention may fail to capture global dependencies adequately, impacting its performance on datasets like CIFAR-10 or ImageNet.

4. **Task-Specific Design of Attention Mechanisms:** As demonstrated in BiFormer, attention mechanisms tailored for vision tasks, combining local and global modeling, yield better performance. Performer' s general-purpose design lacks this specificity, which could explain the observed performance decline.

6.2 Implications for Vision Tasks

To improve Performer's performance on vision tasks, incorporating mechanisms for better local feature extraction and global context modeling is essential. Additionally, fine-tuning activation functions and optimizing the random feature mapping strategy for visual data could help bridge the performance gap.