Bridging Theory and Practice: A Comprehensive Analysis of Large Vision Models for Image Classification

Ahmed Abul Hasanaath, Aisha Alansari
SID: 505, 506
College of Information and Computer Science
King Fahd University of Petroleum and Minerals, Dhahran, KSA

Abstract

Large Vision Models (LVMs) have emerged as a cornerstone in modern image classification, offering state-of-the-art performance across diverse datasets. Understanding their efficiency in terms of computational and memory demands remains crucial for real-world applications. This study provides a comprehensive analysis of LVMs, focusing on both theoretical time and memory complexity as well as empirical performance. The analysis encompasses two prominent LVM architectures—Vision Transformer (ViT) and Swin Transformer—evaluated on CIFAR-100; a benchmark dataset. The ResNet50 model is also evaluated to compare small-scale Convolutional Neural Networks (CNNs) and LVMs. The theoretical analysis assesses computational complexity during forward passes. Complementing this, empirical experiments benchmark computational efficiency (latency, training time, FLOPs) and memory utilization during training and inference. The results are visualized through complexity plots (runtime and memory vs. input size) and trade-off graphs (accuracy vs. complexity metrics), highlighting key differences in efficiency and performance among the models. To the best of our knowledge, this work is the first to offer a nuanced cost vs performance analysis of two prominent LVM architectures on image classification. The theoretical and empirical analysis can help researchers and practitioners select models for image classification. This guidance considers both computational constraints and accuracy requirements. This study not only bridges the gap between theoretical and practical evaluations of LVMs but also underscores the trade-offs inherent in deploying these models. It also provides a foundational reference for future research and development in efficient vision model design.

Index Terms

Large Vision Models, Complexity Analysis, Time Complexity, Space Complexity,

I. Introduction

The advent of Large Vision Models (LVMs) represents a significant shift, contesting the supremacy of Large Language Models (LLM). While LLMs like GPT-3 have profoundly transformed Natural Language Processing (NLP), LVMs are pioneering the subsequent era of artificial intelligence by expanding their capabilities to include the visual domain [1]. ViTs exemplify LVMs, which are large-scale deep learning architectures tailored specifically for vision tasks, distinct from Vision-Language Models (VLMs) that handle multi-modal data combining text and images [2].

Leveraging the fundamental concepts of self-attention mechanisms from NLP [3], ViTs have exhibited exceptional performance across several visual tasks, including object recognition and semantic segmentation. In contrast to conventional Convolutional Neural Networks (CNNs), ViTs process images in a manner analogous to how transformers manage sequences of words, effectively extracting relevant characteristics [4]. In ViTs, images are regarded as a succession of patches. Each patch is transformed into a singular vector, akin to applying word embeddings in transformers for textual data. This approach enables ViTs to interpret image structures and predict class labels efficiently.

Despite their success, ViTs' performance strongly correlates with scaling the number of parameters and dataset size. Accordingly, ViTs come with significant computational and memory demands, which pose challenges for scalability and deployment [5]. The quadratic complexity of the self-attention mechanism [3], along with the requirement for extensive pre-training, leads to models that are resource-demanding during both training and inference. Therefore, with ViTs gaining prevalence in real applications, the need for a deep understanding of their resource requirements intensifies.

This work compares the efficiency of two ViT architectures, the original Vision Transformer, and the Swin Transformer, in terms of computational and memory efficiency. First, it performs theoretical analysis to compute the time complexity of the forward pass during pre-training and fine-tuning. Then, it conducts empirical experiments to compute the computational efficiency, including time per forward pass, training time per epoch, and number of FLOPs. This is to compare empirical results with the theoretical analysis. Memory efficiency is also analyzed by monitoring peak memory usage during training and inference. Besides that, the inference latency is measured to provide insight into the model's efficiency and suitability for real-time or resource-constrained applications. Finally, this work provides visualizations illustrating trade-offs between accuracy, runtime, and memory usage and offers intuitive guidance for model selection and optimization. The present work hence tries to help researchers and practitioners in the selection and optimization of ViT architectures for specific applications. This is done by trying to bridge the gap between state-of-the-art performance and practical feasibility by combining theoretical insights with empirical results. Contributions to this study are enlisted as follows:

- Conducting a theoretical analysis of time complexity for the forward pass.
- Evaluate the computational efficiency including time per forward pass, training time per epoch, and Giga Floating Point Operations (GFLOPs).
- Analyze memory efficiency during training and inference by monitoring peak memory usage.
- Measure the inference latency to assess model suitability for real-time applications.
- Provide visualizations that illustrate trade-offs between accuracy, runtime, and memory usage.
- Establish practical guidelines for selecting and optimizing LVM pipelines based on computational, memory, and latency.

The rest of the study is organized as follows: Section 2 reviews the recent literature related to the study, Section 3 describes the theoretical analysis, Section 4 discusses the empirical analysis, and Section 5 concludes the work.

II. Literature Review

LLMs and Their Inference Efficiency. LLMs, like GPT-3, LLaMa, and others, have taken NLP to the next level. Nonetheless, their computational requirements are slowly becoming overwhelming. Yuan et al. [2] discuss various methods for efficient LLM inference. The authors presented a Roofline model that highlights hardware bottlenecks, memory-bound issues, and the substantial computational demands during training and inference. Despite their success, LLMs still face some challenges, including poor representation of data and learning, as pointed out by Lappin [1]. The authors underlined that balanced assessments must be carried out to understand the capability and limitations of such models[1]. Furthermore, FrugalGPT, as Chen et al. [6], offered strategies such as prompt adaptation and model approximation to reduce inference costs while enhancing performance [6]. The carbon footprint of training such large models is also substantial, with the case of BLOOM, a 176B parameter LLM, emitting considerable CO2, as shown by Luccioni et al.[7].

Scaling and Efficiency of LLMs. Another concern for large-scale models is training efficiency. Hoffmann et al. [8] presented a compute-optimal model, Chinchilla, which outperformed larger models by using more training data while reducing compute resources. This implies that probably in the future, the development of LLMs should be directed to strategies that focus on data efficiency rather than just scaling model size.

Vision Transformers. In the vision domain, Transformers, traditionally used in NLP, have made significant strides in tasks such as image recognition, segmentation, and multi-modal applications. Khan et al. [5] provided a comprehensive survey of the application of transformers to computer vision tasks. The authors showed how their parallel processing capabilities and ability to model long dependencies make them attractive for vision tasks. Furthermore, a pioneering work by Dosovitskiy [4] demonstrated that ViTs could compete with CNNs in terms of accuracy while requiring fewer computational resources. Additionally, Touvron et al. [9] discussed techniques for improving data efficiency in ViTs, particularly through distillation methods.

Innovations in ViTs: The Swin Transformer. The Swin Transformer, developed by Liu et al. (2021), further enhances ViTs to include hierarchical structures and shifted windows that significantly improve their performance on a wide variety of vision tasks [10]. This innovation addresses some of the key challenges found in adapting transformer architecture to vision, including handling high computational complexity and improving generalization across tasks.

The Impact of Scaling on Model Performance. The scaling laws for neural language models are discussed by Kaplan et al. [11]; ViTs also follow suit-performance scales up with the increase in model size, dataset size, and available compute. However, how to handle such heavy models is still a challenge for their practical deployment in real-time applications.

Residual Learning in CNNs and ViTs. In contrast to ViTs, the deep residual learning approach described by He et al. [12] remains influential in the development of CNNs for image recognition tasks. It demonstrates that deep architectures can be effectively trained by introducing residual connections, which help mitigate the vanishing gradient problem. While ViTs offer distinct advantages in terms of handling long-range dependencies, residual networks remain highly effective for traditional vision tasks.

A. *Gap Analysis*

Despite advances in LLMs and ViTs, significant gaps remain in balancing performance, efficiency, and resource use. These provide avenues of potential investigation on which the present study can improve. The points below discuss the gaps found in the literature:

- **Lack of Standard Comparisons Across Architectures.** While residual learning in CNNs remains very effective for image recognition tasks [12], it is clear that ViTs bring advantages in modeling long-range dependencies. There is limited research directly comparing the computational trade-offs, such as GFLOPs, memory efficiency, and latency, between CNNs and ViTs across diverse tasks and scenarios.
- **Inefficiencies in LLM Inference and Training.** While methods to improve inference efficiency include the Roofline model [2] and cost-reduction strategies such as FrugalGPT [6], LLMs are still suffering from hardware bottlenecks, memory limitations, and high energy consumption for both training and inference. This includes studies like Luccioni et al. [7] that point out the environmental impact, mostly in terms of the carbon footprint of large-scale models like BLOOM. However, no comprehensive analysis has been performed with respect to the efficiency of inference, latency, and memory consumption of different LLMs during fine-tuning.
- **Time Complexity and Computational Efficiency.** While previous work focuses on scaling models to improve accuracy, few works address the detailed time complexity for the forward pass in fine-tuning. In addition, there is a lack of thorough evaluation of computational efficiency, such as GFLOPs and training time per epoch.
- **Memory Usage Analysis.** Very few related works present a comprehensive analysis of the peak memory usage for training and inference. This gap is crucial for understanding whether it is feasible to consider a large-scale model for resource-constrained systems.

To address the aforementioned gaps, this study aims to establish practical guidelines for selecting LVM architectures based on computational, memory, and latency measures. This is accomplished by conducting a theoretical analysis of the time complexity for the forward pass, training time per epoch, and GFLOPs. This study also analyzes memory efficiency during training and inference and measures the inference latency. Finally, it provides a visualization to illustrate the trade-offs between accuracy, runtime, and memory usage.

III. Theoretical Analysis

A. *Methodology*

For the theoretical analysis, the main computational layers in each model were identified, and their time complexity was calculated. These base layers are the foundation of the basic operations between all three architectures analyzed: ResNet50, ViT, and Swin. Notably, all three models utilize the same set of base layers.

The layers identified and analyzed in this paper are as follows: Linear, Convolutional (Conv2D), Dropout, Layer Normalization, Gaussian Error Linear Unit (GeLU), and Self-Attention. Linear Layers, also known as fully connected layers, are essential for dense computation and transformation of feature representations. They play a critical role in transformers where they map input embeddings to higher-dimensional spaces. The Conv2D layer is a core operation in CNNs, which is a fundamental building block in many hybrid architectures. This layer performs feature extraction by convolving convolutional kernels over the input data to produce feature maps. These feature maps emphasize the spatial patterns of the input. The Dropout layer is a regularization technique applied during training to reduce overfitting. It achieves this by randomly excluding a fraction of input activations. By doing so, it effectively modifies the underlying data flow through the network. Layer Normalization is extensively used in transformer architectures to stabilize training. These layers ensure having similar scales across different features in one layer, preventing exploding or vanishing gradients. The GeLU Activation Function is an advanced non-linear activation function frequently used in transformer architectures. These introduce smooth, non-linear transformations that enable better convergence with higher model expressiveness. The Self-Attention mechanism is the cornerstone of transformer-based models. It enables the network to capture global dependencies by computing pairwise interactions for all pairs of input tokens. This ensures the completeness of feature extraction and interaction modeling.

B. Computational Complexity of Base Layers

Table I summarizes the computational complexity for each base layer, alongside a description of their growth rates. This detailed analysis provides a foundational understanding of the computational requirements for each layer, which is critical for comparing CNNs with LVMs such as ViT and Swin Transformers.

TABLE I
TIME COMPLEXITY AND GROWTH RATE OF BASE LAYERS

Layer	**Time Complexity**	**Growth Rate**
Convolution (Conv2D)	$O(K^2 \cdot C_{\text{in}} \cdot C_{\text{out}} \cdot H_{\text{out}} \cdot W_{\text{out}})$	Quadratic in H, W, Linear in K, C
Linear Layer	$O(m \cdot n)$	Quadratic in m, n
Dropout	$O(n)$	Linear in n
Layer Normalization	$O(n)$	Linear in n
GeLU Activation	$O(n)$	Linear in n
Self-Attention	$O(N^2 \cdot d)$	Quadratic in N, Linear in d

Linear layers map input vectors of size m to output vectors of size n through matrix multiplication, followed by adding bias terms. The number of operations required is proportional to $m \cdot n$, where m denotes the input dimension and n is the output dimension. The result of these operations is a time complexity of $O(m \cdot n)$.

As with Conv2D, the quadratic nature of the computational cost arises from the pairwise interactions between the input and output elements. Accordingly, linear layers become computationally demanding for large input or output dimensions. This layer computes its output by convolving a kernel of size $K \times K$ across the input feature maps. For each element in the output feature map, the kernel performs $K^2 \cdot C_{\text{in}}$ multiplications and additions, where C_{in} represents the number of input channels. The resulting output feature map has dimensions determined by C_{out}, H_{out}, and W_{out}. These notations corresponds to the number of output channels, height, and width, respectively. Consequently, the total computational cost of a Conv2D operation is $O(K^2 \cdot C_{\text{in}} \cdot C_{\text{out}} \cdot H_{\text{out}} \cdot W_{\text{out}})$. The complexity rises quadratically in relation to the dimensions of the output feature map, rendering Conv2D the most computationally demanding operation in CNNs.

Dropout generates a binary mask of size equal to the input tensor and then applies this mask element-wise. The computational cost of generating the mask and performing the element-wise multiplication is directly proportional to the size of the input tensor. Therefore, the computational complexity of Dropout is $O(n)$, where n denotes the total number of input elements. The linear growth implies that Dropout contributes negligible overhead to the total computation.

Layer Normalization computes the mean and variance along a certain axis of the input features and normalizes the features. Since each input feature requires a constant number of operations to compute its mean, variance, and subsequent normalization, the computational cost is linearly related to the number of input elements. Hence, it has a complexity of $O(n)$. Since it grows linearly, Layer Normalization is computationally efficient and can be used in large-scale models.

The GeLU activation function computes a probabilistic approximation of the Rectified Linear Unit (ReLU) activation by leveraging the cumulative distribution function of the Gaussian distribution. This computation is performed independently for each input element. Accordingly, the total computational cost is proportional to the size of the input tensor. The complexity is therefore $O(n)$, making it computationally efficient for large-scale models.

The computational cost of Self-Attention is dominated by two key operations: the computation of the query (Q), key (K), and value (V) matrices, and the next attention computation involving $Q \cdot K^T$. The computation of Q, K, and V requires $O(N \cdot d^2)$ operations, where N represents the sequence length and d denotes the embedding dimension. Nevertheless, the dominant term in the overall complexity stems from the matrix multiplication $Q \cdot K^T$, which requires $O(N^2 \cdot d)$ operations. This quadratic growth concerning the sequence length N makes Self-Attention a computational bottleneck, specifically for long sequences. However, it also highlights its powerful capacity to capture global relationships in the input data.

C. *Comparative Analysis of Models*

After computing the complexities of the base layers used in Deep Learning (DL) architectures, we extended our analysis to evaluate the computational complexities of selected models: ViT, Swin Transformer, and ResNet50. Table II summarizes our analysis. In this section, we provide a detailed breakdown of the complexities for each model by analyzing their core components. Moreover, we discuss the implications of these complexities on computational performance. For all models, we have ignored layers such as Dropout, GeLU, and Layer Normalization, as their computational cost is negligible compared to the dominant layers.

1) Vision Transformer: ViT represents a pioneering approach to applying transformer-based architectures to image data. ViT processes images by dividing them into patches and feeding these patches into a series of transformer blocks. The first stage of ViT is the *Patch Embedding* layer, which converts the input image into a sequence of patches. This is achieved using a convolutional layer with a kernel size equal to the patch size $P \times P$. The number of patches, N, is given by:

$$N = \left(\frac{H}{P}\right)^2,$$

where H is the height and width of the image, assuming square images. Rearranging, the image height can be expressed as:

$$H = \sqrt{N} \cdot P.$$

The computational complexity of the Patch Embedding layer is dominated by the Conv2D operation, which has a complexity of $O(H^2 \cdot O(1))$. Substituting H in terms of N and P, we find that the complexity simplifies to:

$$O(N \cdot P^2).$$

Following the Patch Embed layer are the transformer blocks. ViT consists of L transformer blocks, where each block contains two main components: *Self-Attention* and a *Multi-Layer Perceptron (MLP)*. The computational cost of Self-Attention for a single block is:

$$O(N^2 \cdot d),$$

where N is the number of patches and d is the embedding dimension.

Each transformer block contains a two-layer MLP. The first layer maps d-dimensional input features to $4d$ dimensions, while the second layer maps it back to d. The complexities of these layers are:

$$O(N \cdot d \cdot 4d) \quad \text{and} \quad O(N \cdot 4d \cdot d),$$

respectively. Combined, the MLP contributes a total complexity of:

$$O(N \cdot d^2).$$

Combining the contributions of Self-Attention and MLP, the total complexity of one transformer block is:

$$O((N^2 \cdot d + N \cdot d^2) \cdot L).$$

where L corresponds to the number of transformer blocks. Between the Patch Embedding layer and the transformer blocks, the latter dominates the computational cost and thus can be considered as the time complexity of the ViT.

2) Swin Transformer: The Swin Transformer is an extension of ViT, designed to address the quadratic scaling issue of Self-Attention in ViT by using a hierarchical approach with windowed attention mechanisms. Similar to ViT, the Patch Embedding layer in Swin Transformer converts the input image into patches. The computational complexity is identical to that of ViT, given by:

$$O(N \cdot P^2).$$

Swin Transformer employs a hierarchical structure with 4 stages, each consisting of 2 transformer blocks. The two primary components of the transformer blocks are *Windowed Self-Attention* and *MLP*. Instead of global Self-Attention, Swin Transformer employs Self-Attention within local windows of size M. The complexity of Window Self-Attention is:

$$O(M \cdot N \cdot d),$$

which is significantly lower than the $O(N^2 \cdot d)$ complexity of global Self-Attention in ViT. Similar to ViT, each transformer block in the Swin Transformer contains a two-layer MLP with a combined complexity of:

$$O(N \cdot d^2).$$

With 2 blocks per stage and 4 stages in total, the overall complexity of the Swin Transformer is:

$$O((M \cdot N \cdot d + N \cdot d^2) \cdot L),$$

where L is the total number of blocks across all stages. As with ViT, the transformer blocks dominate the computational cost of Swin Transformer.

3) ResNet50: ResNet50 is a convolutional neural network (CNN) known for its residual connections, which improve gradient flow during training. The computational complexity of ResNet50 is dominated by its convolutional layers. The Conv2D layers in ResNet50 perform convolutions over the spatial dimensions of the input feature maps. The complexity of a single Conv2D layer is quadratic with respect to the spatial dimensions, given by:

$$O(H^2 \cdot K^2 \cdot C_{\text{in}} \cdot C_{\text{out}}),$$

where H is the height (and width) of the input, K is the kernel size, and C_{in} and C_{out} are the number of input and output channels, respectively. Since ResNet50 consists primarily of stacked Conv2D layers, its total complexity is quadratic in nature.

TABLE II

TIME COMPLEXITY AND GROWTH RATE OF VIT, SWIN TRANSFORMER, AND RESNET50

Model	Time Complexity	Growth Rate
ViT	$O((N^2 \cdot d + N \cdot d^2) \cdot L)$	Quadratic in N, Linear in d, L
Swin Transformer	$O((M \cdot N \cdot d + N \cdot d^2) \cdot L)$	Linear in N, M, d, L
ResNet50	$O(H^2 \cdot K^2 \cdot C_{\text{in}} \cdot C_{\text{out}})$	Quadratic in H, Linear in K, C

D. Implications of the Theoretical Analysis

The computational complexities derived for the ViT, Swin Transformer, and ResNet50 provide insight into their suitability for various applications, trade-offs in performance, and potential bottlenecks. This section explores these implications from a theoretical point of view.

The complexity of ViT, given by $O((N^2 \cdot d + N \cdot d^2) \cdot L)$, highlights its dependence on the number of patches N, the embedding dimension d, and the number of layers L. These factors lead to the following implications:

1) **Scalability with Image Size:** The quadratic scaling of Self-Attention with respect to the number of patches $(N^2 \cdot d)$ makes ViT computationally expensive for high-resolution images. As the resolution increases, the number of patches grows quadratically. Consequently, it leads to a rapid proliferation in computational and memory demands.
2) **Embedding Dimension Impact:** The MLP complexity term, $O(N \cdot d^2)$, suggests that larger embedding dimensions lead to substantial computational costs. This forms a trade-off between utilizing a large d for better feature representation and managing computational efficiency.
3) **Usefulness for Smaller Images:** In low-resolution images, when the number of patches N is small, ViT operates effectively, utilizing its transformer-based architecture to achieve exceptional outcomes in tasks requiring global context, such as image classification or segmentation.
4) **Pre-training Requirements:** The complexity of ViT suggests a high demand for computational resources during training. This is often offset by the usefulness of large-scale pre-training, which can result in exceptional generalization when fine-tuned on downstream tasks.

The Swin Transformer, with a complexity of $O((M \cdot N \cdot d + N \cdot d^2) \cdot L)$, introduces window-based Self-Attention, lessening the quadratic scaling of classical Self-Attention. This has the subsequent implications:

1) **Efficient Attention Mechanism:** By limiting Self-Attention to local windows, the Swin Transformer attains linear complexity in the number of patches per window $(M \cdot N \cdot d)$. As a result, it becomes more efficient for high-resolution images. This allows for scalability to larger input sizes without excessive computational expenses.
2) **Hierarchical Design Advantages:** The hierarchical structure of the Swin Transformer decreases the number of patches progressively through pooling-like operations, allowing the model to focus on global features in later stages. This hierarchical technique balances local feature extraction and global context modeling efficiently.

3) **Versatility in Vision Tasks:** Swin Transformer's capacity to handle high-resolution images efficiently makes it appropriate for tasks demanding fine-grained details, such as object detection, semantic segmentation, and image generation.
4) **Hardware Efficiency:** The localized computation of Window Self-Attention is well-suited for modern computer accelerators, such GPUs and TPUs, which are specialized for batch processing. This improves both training and inference efficiency relative to global Self-Attention in ViT.

ResNet50, characterized by its quadratic computational cost relative to the spatial dimensions of the input, is mostly dominated by convolutional layers. This entails the subsequent implications:

1) **Scalability with Image Size:** The quadratic complexity of convolutional layers poses computing challenges for ResNet50 when processing high-resolution images. Nonetheless, the overhead is less significant in comparison to ViT at equivalent resolutions.
2) **Efficiency in Low-Resolution Tasks:** ResNet50 is highly efficient for tasks involving low-to-moderate-resolution images, such as image classification on datasets like CIFAR or ImageNet. Therefore, it is considered an ideal choice for deployment in resource-constrained environments.
3) **Limited Context Modeling:** ResNet50 struggles with tasks requiring global context understanding (e.g., semantic segmentation or global texture recognition), as convolutions are inherently local in nature. This is where transformer-based architectures have a clear advantage.

The theoretical complexity analysis demonstrates that each model has its strengths and trade-offs. ViT offers strong global context modeling but at a high computational cost. Swin Transformer achieves a balance between efficiency and performance, making it suitable for diverse tasks. ResNet50 remains a reliable, efficient choice for simpler applications.

IV. Empirical Analysis

A. Experimental Setup

The empirical analysis was conducted on a high-performance computational setup featuring an Nvidia A5000 GPU with 24 GB of VRAM, running on an Ubuntu 20.04 operating system. The experiments utilized the PyTorch deep learning framework for implementing and fine-tuning the models. We used the CIFAR-100 dataset, a well-known benchmark dataset for image classification tasks, to fine-tune the selected models. The dataset comprises 100 classes with 50,000 training images and 10,000 test images, each of size 32×32 pixels. We downloaded the pretrained versions of the model and fine-tuned on the CIFAR-100 dataset for an image classification task.

Fine-tuning was carried out using the Adam optimizer, configured with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-8}$. The learning rate was set to 0.0001, and the models were trained for a total of 10 epochs. This setup was selected to ensure stable convergence while maintaining computational efficiency.

For evaluation, three essential performance metrics were used to assess the models' efficiency and scalability. First, inference time was measured, representing the time a model needs to process one batch of input images in the forward pass. Second, memory consumption was measured by checking the peak usage during the inference phase in GPU memory. Finally, the number of floating-point operations (FLOPs) required for a single forward pass was computed to provide a quantitative measure of computational complexity. All of these metrics collectively provide valuable insight into the practical deployment of models in real-world scenarios..

B. Performance vs Cost

In this section, we examine the trade-off between performance and computational cost. We evaluate both the computational expense and the model performance by analyzing several key metrics, including average time per forward pass, training time per epoch, and GFLOPs. As shown in Fig. 1, we observe a clear (and expected) trend: the models with higher time complexity (as explored in III) have higher average forward pass time. Specifically, the ViT model proves to be the most computationally intensive, followed by the Swin model, with ResNet being the least demanding in terms of both forward pass time and training time per epoch.

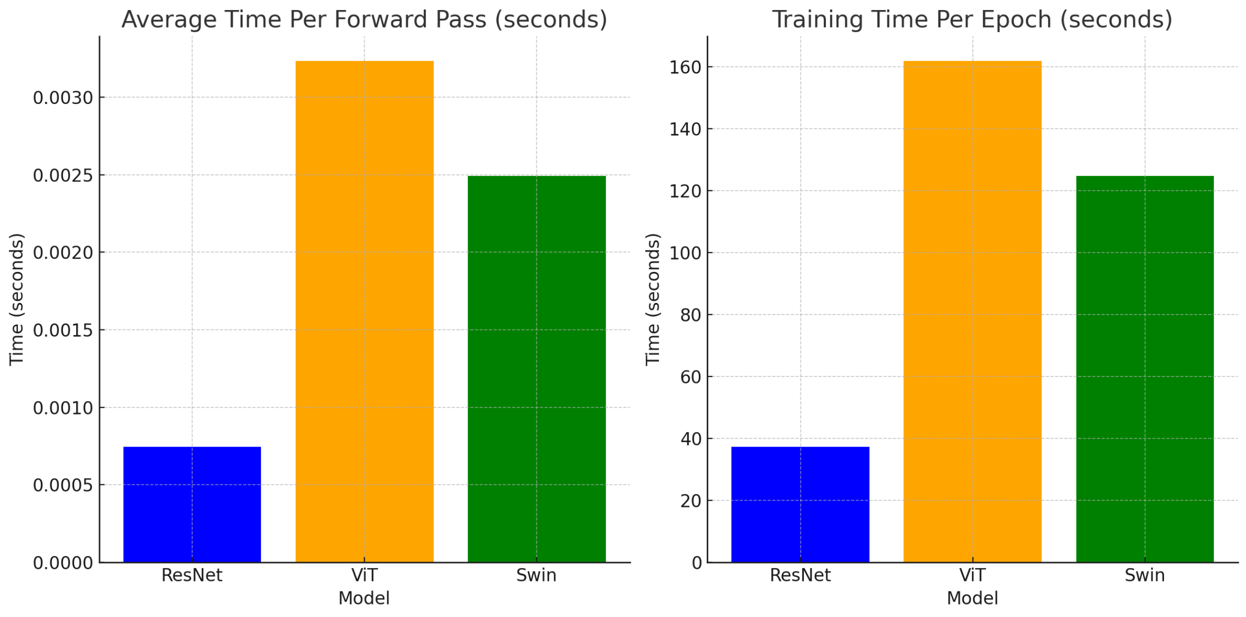

Fig. 1. The first plot illustrates the average training time per forward pass, and the second plot illustrates the average training time per epoch, with the batch size set to 32.

GFLOPs, which represent the computational load during model training, are another crucial measure of performance. As the batch size increases, the GFLOPs double, aligning with theoretical expectations, as summarized in tables III, IV and V. This scaling behavior reflects the increased computational demand of processing larger batches. In addition to examining computational costs, we also analyze the relationship between validation accuracy and computational cost (measured by GFLOPs) for each model, as depicted in Fig. 2. The following trends can be observed. *ResNet* consistently surpasses the other two models, exhibiting the highest validation accuracy from the start and maintaining its lead throughout the training process. *Swin*, while trailing behind ResNet, shows a steady increase in validation accuracy over time. Although its performance is slightly lower than ResNet's, it stays competitive and reveals a promising balance between computational cost and accuracy. *ViT*, on the other hand,

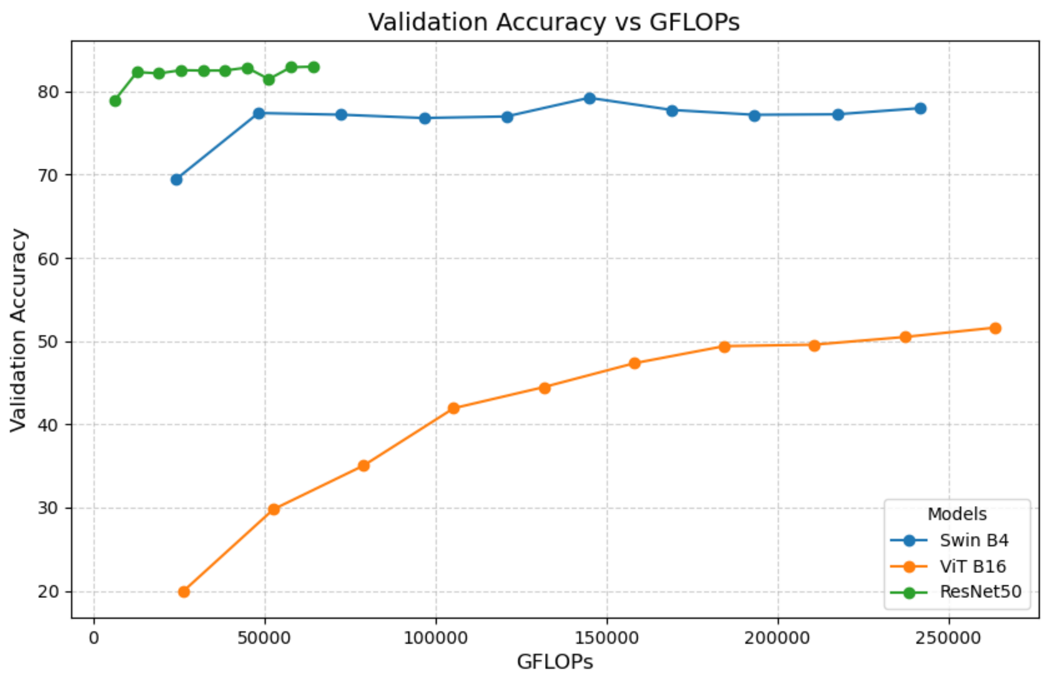

Fig. 2. Validation Accuracy vs GFLOPs. The x-axis represents GFLOPs, corresponding to the computational effort required during training.

suffers from considerable overfitting as training progresses. Despite its higher computational cost, ViT lags behind both ResNet and Swin in terms of validation accuracy. *These findings emphasize the necessity of evaluating both model architecture and training methodologies to enhance performance and computational efficiency.*

C. *Memory Consumption*

The GPU memory consumption for each model was measured across various batch sizes, and several key trends were observed. Fig 3 shows the relationship between GPU memory consumption (in MB) and batch size, while Fig 4 displays the rate at which memory consumption increases as the batch size grows.

One of the primary trends observed is that memory consumption grows at different rates for each model. Specifically, from slowest to fastest growth, the order is **ViT** $\rightarrow$ **ResNet** $\rightarrow$ **Swin**, where ViT consumes the least memory, and Swin consumes the most. Up until a batch size of 64, ViT exhibits faster memory consumption growth compared to ResNet. However, when the batch size exceeds 64, ResNet's memory consumption grows faster than ViT's. This shift suggests that while ViT is more memory-efficient at smaller batch sizes, ResNet becomes more memory-intensive at larger batch sizes.

A key observation across all models is that GPU memory consumption does not increase linearly with the batch size, especially at smaller batch sizes. Initially, when the batch size is doubled, the memory consumption increases by a smaller factor than expected, showing diminishing returns in memory growth. However, after a certain batch size threshold, the rate of increase starts to stabilize and approaches a factor of 2. This suggests that, beyond a

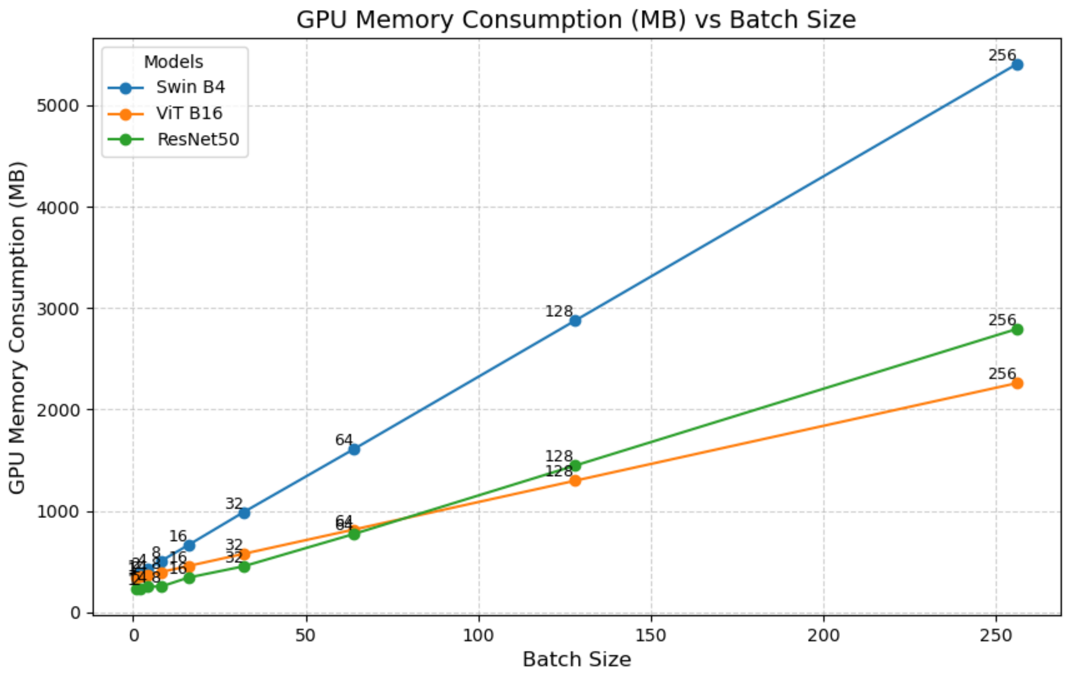

Fig. 3. GPU Memory Consumption (MB) vs Batch Size

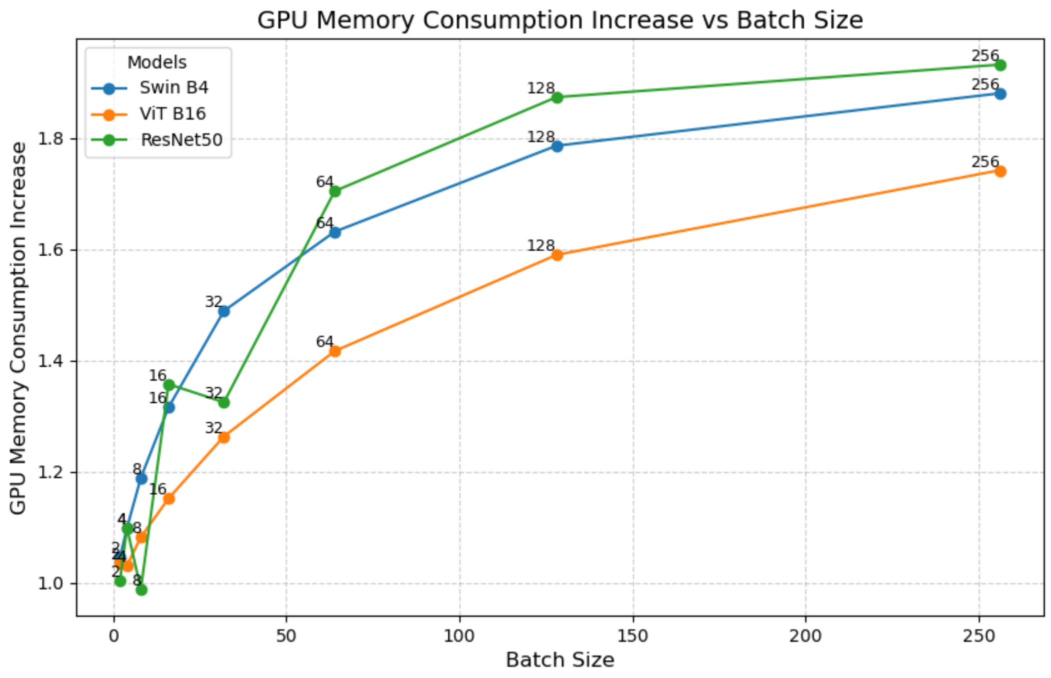

Fig. 4. GPU Memory Consumption Increase vs Batch Size

certain point, the memory consumption becomes more predictable and scales closer to what would be expected from doubling the batch size.

Another observation across all models is that a significant portion of the GPU memory is consumed by the model's parameters. As summarized in Table III, Table IV and Table V, doubling the batch size results in only a fractional increase in the ratio. The model's memory requirements dominate, and changes in batch size contribute incrementally to the total memory usage. This trend also emphasizes the importance of optimizing model architectures to reduce memory usage, as the model itself is the dominant factor driving GPU memory consumption. For example, a reduction in the number of model parameters could lead to a more memory-efficient design, enabling better scalability without relying solely on increasing batch size.

D. Inference Latency

Inference latency was measured for each model across different batch sizes, and several important trends emerged. The results show that the relationship between batch size and inference latency is non-monotonic, with smaller batch sizes often resulting in higher latency, followed by a decrease and subsequent increase as the batch size increases. Fig 5 shows the relationship between inference latency and batch size, while Fig 6 displays the rate at which inference latency increases as the batch size grows.

For the ViT model, the latency for a batch size of 1 was measured at 70.57 ms, which is relatively high. However, when the batch size increased to 2, the latency dropped significantly to 8.68 ms, and then started increasing as the batch size grew larger. Similarly, for Swin, the latency for a batch size of 1 was 112.14 ms, which decreased to 29.5 ms for a batch size of 2, and further dropped to 19.3 ms at batch size 4 before increasing again at larger batch sizes. In the case of ResNet, the latency for a batch size of 1 was 151.01 ms, which reduced to 47.67 ms for batch size 2, and 42.97 ms for batch size 4, before rising again for larger batch sizes. These trends point to a common observation: smaller batch sizes appear to cause higher latency initially, with a significant reduction in latency at a batch size of 2 or 4, after which the latency begins to increase again. This pattern is observed across all models, with Swin showing the fastest increase in inference latency, followed by ViT and then ResNet. This order aligns with the expected model complexity, as Swin, being the most complex model, exhibits the highest latency, while ViT, with its relatively simpler architecture, shows a lower latency. ResNet, being more efficient in terms of architecture, has the lowest initial latency but still experiences an increase with larger batch sizes. Similar to memory consumption, past a certain batch size, the factor by which inference latency increases seems to stabilize at a factor of 2.

One possible explanation for the higher latency observed at small batch sizes, especially for batch size 1, is that models are often optimized for batch processing rather than single-inference scenarios. Deep learning frameworks and hardware accelerators (such as GPUs) are highly optimized for parallel computations. When processing a batch of data, operations can be parallelized across the batch, leading to more efficient execution. However, when the batch size is small, especially batch size 1, the computational workload is smaller, and the model may not fully leverage the parallelism capabilities of the hardware. This could lead to higher overhead per sample, causing increased inference latency. Additionally, for very small batch sizes, the normalization layers and other operations designed

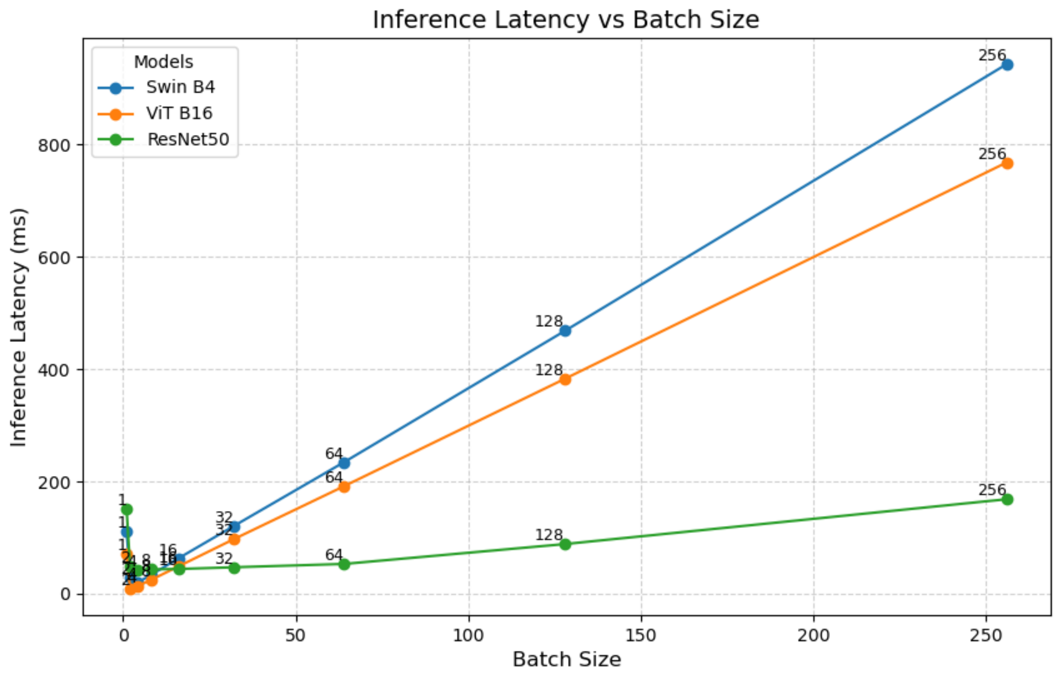

Fig. 5. Inference Latency vs Batch Size

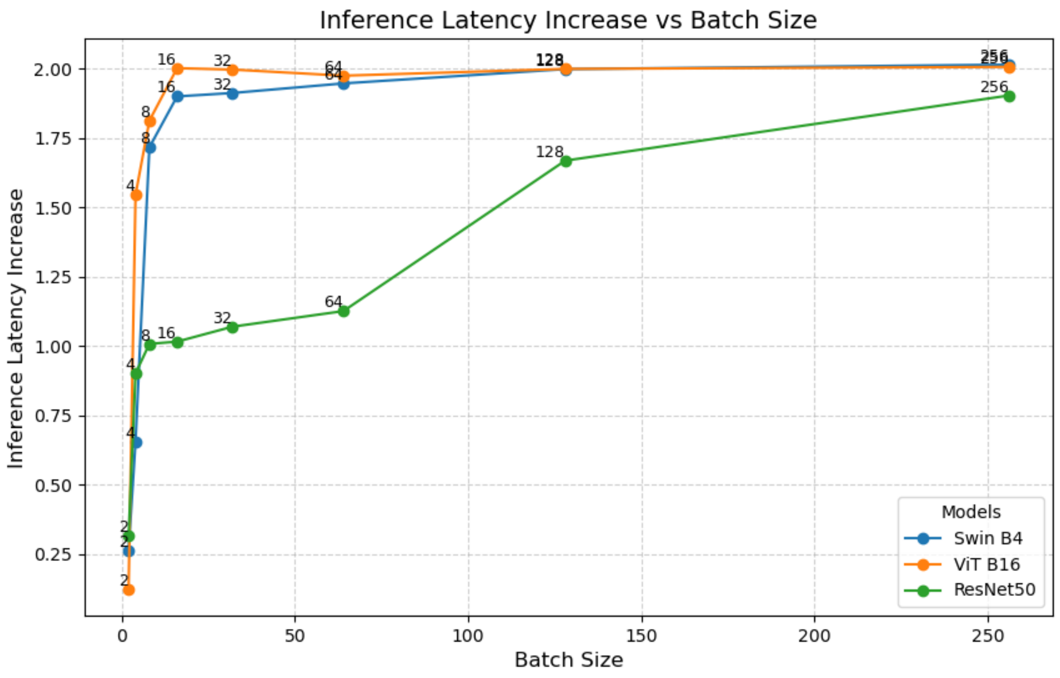

Fig. 6. Inference Latency Increase vs Batch Size

to optimize processing over multiple inputs might be less efficient, contributing further to the latency observed in these cases. As the batch size increases, the efficiency of these operations improves, leading to reduced latency until the benefits of further increasing batch size are outweighed by other factors, such as memory bandwidth limitations or hardware bottlenecks, causing the latency to increase at very large batch sizes.

TABLE III
PERFORMANCE METRICS FOR RESNET50 ACROSS DIFFERENT BATCH SIZES.

Batch Size	**Inference Latency (ms)**	**Inference Latency Increase**	**GPU Mem. (MB)**	**GPU Mem. Increase**	**GFLOPs**
1	151.015808	-	231.0	-	4.109669
2	47.665791	×0.315634 ↓	232.0	×1.004329 ↑	8.219339
4	42.961727	×0.901312 ↓	255.0	×1.099138 ↑	16.438678
8	43.294369	×1.007743 ↑	252.0	×0.988235 ↓	32.877355
16	43.982975	×1.015905 ↑	342.0	×1.357143 ↑	65.754710
32	47.054367	×1.069831 ↑	453.0	×1.324561 ↑	131.509420
64	52.987358	×1.126088 ↑	772.0	×1.704194 ↑	263.018840
128	88.408066	×1.668475 ↑	1446.0	×1.873057 ↑	526.037680
256	168.276199	×1.903403 ↑	2793.0	×1.931535 ↑	1052.075360

TABLE IV
PERFORMANCE METRICS FOR VIT B16 ACROSS DIFFERENT BATCH SIZES.

Batch Size	**Inference Latency (ms)**	**Inference Latency Increase**	**GPU Mem. (MB)**	**GPU Mem. Increase**	**GFLOPs**
1	70.571777	-	343.0	-	16.866722
2	8.658816	×0.122695 ↓	355.0	×1.034985 ↑	33.733443
8	24.253216	×1.812799 ↑	396.0	×1.081967 ↑	134.933772
16	48.554913	×2.001999 ↑	456.0	×1.151515 ↑	269.867545
32	96.959648	×1.996907 ↑	576.0	×1.263158 ↑	539.735089
64	191.492828	×1.974974 ↑	816.0	×1.416667 ↑	1079.470178
128	382.852844	×1.999306 ↑	1297.0	×1.589461 ↑	2158.940357
256	768.274963	×2.006711 ↑	2259.0	×1.741712 ↑	4317.880713

E. *Implications of Empirical Analysis*

The empirical results uncover noteworthy insights into the trade-offs between model performance, computational cost, and inference latency, with significant implications for model selection. First, the relationship between model complexity and computational cost is evident. As expected, more complex models like ViT demand higher com-

TABLE V

Performance metrics for Swin B4 across different batch sizes.

Batch Size	Inference Latency (ms)	Inference Latency Increase	GPU Mem. (MB)	GPU Mem. Increase	GFLOPs
1	112.144958	-	369.0	-	15.466905
2	29.392000	×0.262089 ↓	386.0	×1.046070 ↑	30.933809
4	19.286816	×0.656193 ↓	424.0	×1.098446 ↑	61.867618
8	33.108864	×1.716658 ↑	504.0	×1.188679 ↑	123.735237
16	62.912254	×1.900163 ↑	663.0	×1.315476 ↑	247.470473
32	120.322243	×1.912541 ↑	987.0	×1.488688 ↑	494.940946
128	468.238892	×1.998597 ↑	2875.0	×1.785714 ↑	1979.763786
256	943.388184	×2.014758 ↑	5405.0	×1.880000 ↑	3959.527571

putational resources, especially in forward pass time. The ViT model, in particular, is the most computationally intensive, followed by Swin and then ResNet, which proves to be the least demanding.

This trend is in line with theoretical expectations on computational cost: the more complex the architecture, the longer it takes to train and perform inference. The scaling behavior of GFLOPs with batch size also shows that increasing the batch size increases the computational burden: doubling the batch size roughly doubles the GFLOPs, as theory would suggest. However, it is important to relate this to the actual cost in terms of validation accuracy. ResNet is consistently superior in validation accuracy, while its variance is small; therefore, ResNet offers a better balance between performance and computational efficiency. Swin has improved gradually with more computational overheads, whereas the ViT model is computationally more expensive yet suffers from overfitting and results in poor performance in terms of validation accuracy. These results stress the need to further balance model architecture and training techniques to improve performance with affordable computational costs.

Besides, memory consumption plays a crucial role in determining model efficiency, especially when scaling batch sizes. The relationship between batch size and GPU memory consumption is nonlinear because a large portion of the usage is due to the parameters of the model rather than the batch size. As a result, memory consumption would increase with the batch size, but increasing further will not have such strong effects. This suggests that above a certain threshold, reducing batch size alone is not an effective strategy to control the GPU memory. There could be a better approach towards managing GPU memory, in particular for large-scale models and datasets, which could involve model optimization. Such optimizations might involve pruning parameters, quantization, and memory-efficient architecture, where the model requires less memory without any loss in performance. These findings suggest that careful memory management is essential to achieve an optimal balance between batch size and available GPU memory, especially when utilizing large models like ViT and Swin.

These results also demarcate how the batch size and inference latency relate to each other: Typically, smaller batch sizes have very high latencies that decrease considerably until a certain batch size is reached. For example, for the ViT model, latency decreased significantly from 70.57 ms for a batch size of 1 to 8.68 ms for a batch size of 2, after which latency gradually increased with larger batch sizes. A similar trend was seen in Swin and ResNet, except that Swin had the highest initial latency and also the fastest increase in latency as the batch size grew. This can be explained by the capability of the hardware to optimize operations at higher batches, thus giving higher parallelism and efficiency. Interestingly, beyond a certain batch size, the factor by which latency increases stabilizes, showing diminishing returns in performance with larger batches. Inference latency behavior across these models is reflective of their architectural complexities: the Swin model presents the fastest latency increase due to its higher complexity, while the opposite can be seen from ResNet, which maintains the lowest latency due to more efficient design. These observations call for careful choice of the batch size with respect to the application at hand. Low-latency applications may want to use smaller batches, while computationally demanding tasks may be more efficient with larger batches.

V. Conclusion

The continued scaling of LLMs and ViTs has revolutionized AI applications, offering state-of-the-art performance in a wide variety of tasks. Regardless, as these models grow in size and further complexity, the challenges to their computational efficiency and deployment consideration become increasingly noteworthy. For the vision domain, there are transformers like ViT and Swin Transformers that come with substantial potential and represent a feasible alternative to the classical convolutional networks. This is achieved by taking care of the long-range dependencies and also providing better generalization. With all those advancements, residual learning in CNN has still remained a crucial thing in image recognition tasks.

Future research and development must focus on constructing more efficient models, reducing their carbon footprint, and exploring data-efficient training approaches to make large-scale AI systems more accessible and sustainable. Balancing the soundnesses of these advanced architectures with practical deployment constraints will be fundamental in ensuring that the usefulness of AI can be fully acknowledged across various applications.

References

[1] Shalom Lappin. Assessing the strengths and weaknesses of large language models. *Journal of Logic, Language and Information*, 33(1):9–20, 2024.

[2] Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, et al. Llm inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*, 2024.

[3] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[4] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[5] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

[6] Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

[7] Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the carbon footprint of bloom, a 176b parameter language model. *Journal of Machine Learning Research*, 24(253):1–15, 2023.

[8] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[9] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

[10] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[11] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.