# Vision Transformer End Evaluation Report

## 1. Introduction

The Vision Transformer (ViT) represents a shift in computer vision from local spatial hierarchies (CNNs) to global dependency modeling.[1] By treating an image as a sequence of discrete patches, the architecture applies self-attention mechanisms originally developed for natural language processing.[1] This implementation evaluates a ViT model built from scratch and optimized for the CIFAR-10 dataset.[4]

## 2. Data Engineering and Regularization

### 2.1 Dataset Configuration

The CIFAR-10 benchmark consists of 60,000 $32 \times 32$ color images across 10 balanced classes.[6] Due to the low resolution, the model requires high-granularity patching to capture sufficient structural detail.[4]

### 2.2 Preprocessing and Augmentation

To improve generalization, a multi-stage transformation pipeline was implemented:

1. **Random Horizontal Flip**: Probability of 0.5 to ensure orientation invariance.[9]
2. **Random Cropping**: 4-pixel padding followed by $32 \times 32$ cropping to simulate spatial shifts.[4]
3. **Standardization**: Normalization using CIFAR-10 specific constants ( $\mu = [0.4914, 0.4822, 0.4465]$ , $\sigma = [0.247, 0.243, 0.261]$ ) to stabilize gradient descent.[4]

Python

```
# Data Transformation Pipeline
train_transform = transforms.Compose(, std=[0.247, 0.243, 0.261])
])
```

### 2.3 Regularization Protocols

- **Dropout**: A rate of 0.1 was applied to attention weights and MLP activations to prevent co-adaptation.[4]
- **Weight Decay**: AdamW was utilized with a coefficient of 0.05 to penalize large weights.[11]
- **Gradient Clipping**: Max norm of 1.0 was enforced to mitigate the exploding gradient problem typical in deep transformers.[4]

## 3. Hyperparameter Optimization

Through empirical testing, the following configuration was established as the optimal balance between capacity and convergence stability:

| Hyperparameter | Tested Values | Final Selection |
|---|---|---|
| Patch Size ($P$) | 2, 4, 8 | 4 |
| Latent Dimension ($D$) | 256, 384, 768 | 384 |
| Attention Heads | 4, 8 | 8 |
| Encoder Layers | 4, 6, 8 | 6 |
| Learning Rate | 1e-3, 3e-4, 1e-4 | 3e-4 |
| Batch Size | 64, 128 | 128 |

## 4. Architectural Synthesis

The core of the ViT is the "tokenization" of visual data, transforming spatial grids into sequence vectors.[5]

### 4.1 Tokenization and Embedding

The image is divided into $N$ non-overlapping patches $x_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$, where $N = (HW)/P^2$.[15] For $32 \times 32$ images with $P = 4$, $N = 64$. These are projected

into a latent space $D$ using a linear layer.[4]

## 4.2 Positional Encoding and CLS Token

- **CLS Token**: A learnable parameter $z_0 = x_{class}$ is prepended to aggregate global information.[1]
- **Position Embeddings**: Since attention is permutation-invariant, learnable spatial identifiers are added to preserve the image grid structure.[18]

Python

```
# Patching and Embedding Initialization
self.patch_size = config.patch_size
self.num_patches = (self.image_size // self.patch_size) ** 2
self.position_embeddings = nn.Parameter(torch.zeros(1, self.num_patches + 1,
config.hidden_size))
self.cls_token = nn.Parameter(torch.zeros(1, 1, config.hidden_size))
```

## 4.3 Multi-Head Self-Attention (MHSA)

The MHSA module computes dynamic relationships between every patch pair.[18] It utilizes Queries ($Q$), Keys ($K$), and Values ($V$) to calculate attention scores:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The scaling factor $\sqrt{d_k}$ ensures numerical stability during backpropagation.[16]

## 4.4 Transformer Encoder and Classification

Each of the 6 encoder blocks consists of a pre-normalization MHSA layer and a Feed-Forward Network (MLP) with GELU activation.[4] The final classification head maps the extracted CLS token to the 10 class logits.[17]

Python

```
# MLP Block Implementation
def forward(self, x):
    x = self.fc1(x)
    x = F.gelu(x)
    x = self.dropout(x)
    x = self.fc2(x)
    return x
```

# 5. Performance Evaluation

The model was trained for 50 epochs, achieving significant fit on the training data but highlighting the data-hungry nature of transformers:

- **Training Accuracy**: 95.03%
- **Peak Test Accuracy**: 79.30%

The generalization gap is attributed to the low inductive bias of the ViT compared to CNNs.[18] While CNNs assume local connectivity, transformers must learn these relationships from scratch, which typically requires larger datasets or more extensive pre-training to outperform traditional architectures on low-resolution benchmarks.[1]