

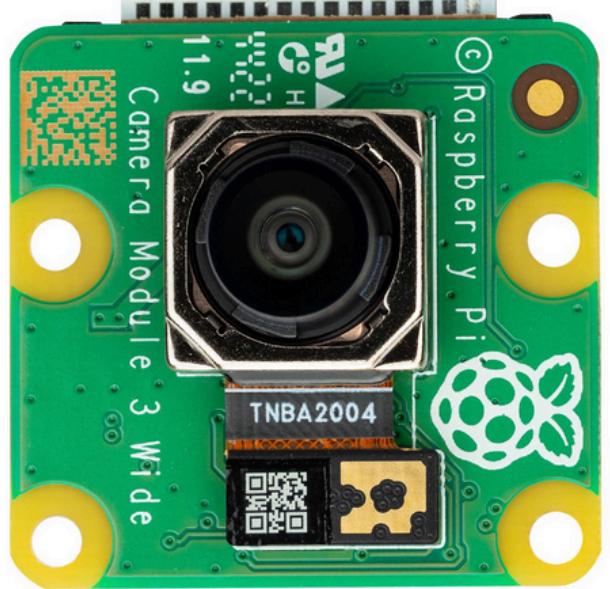
# Mechanistic Interpretability for Vision Models Optimization

Andrea Petruzzi 1982874

Elisabetta Russo 1986112

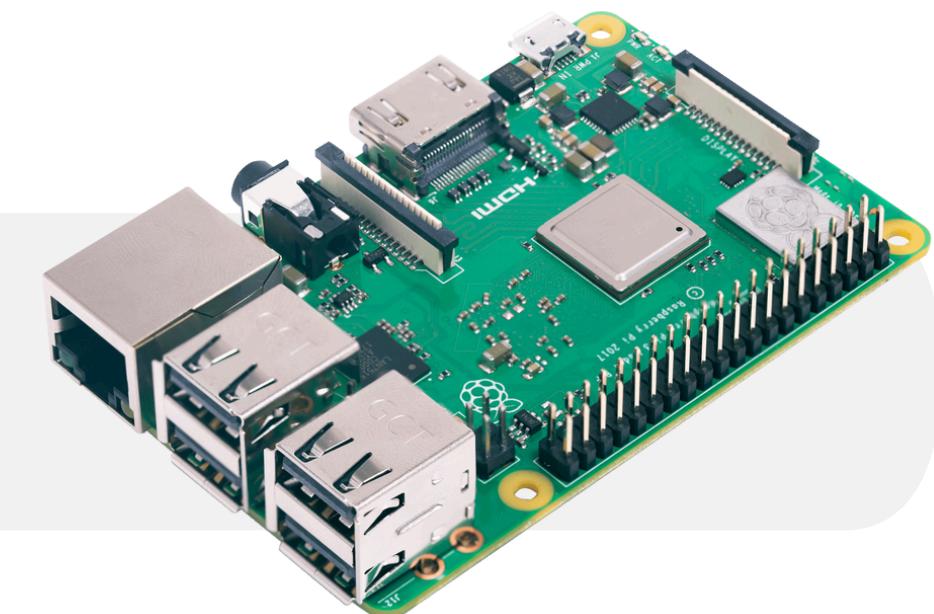
Niccolò Buonfiglio 1983405

Computer Vision  
A.Y. 2024-2025



# Problem Statement

Improve the efficiency of Vision Transformers  
while preserving their accuracy



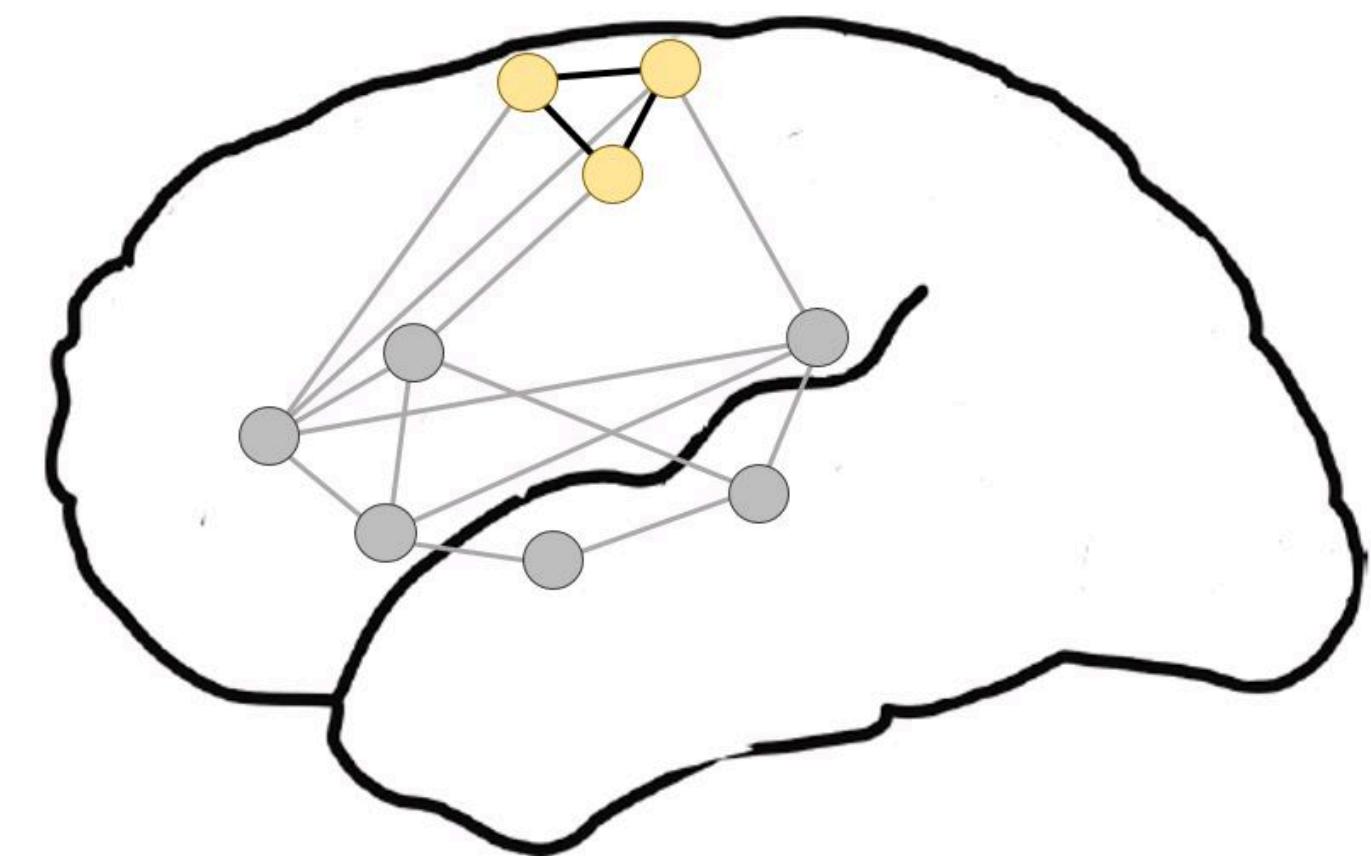
Goal

Approach

# State of the Art

Interpretability research aims to uncover internal neural network circuits that drive specific behaviors.

Mechanistic interpretability research aims to automate the process, allowing to find more and larger circuits.



# State of the Art

## First approach

Automated Circuit DisCovery (ACDC) finds these circuits via activation patching. It can be computationally costly, depending on the level of granularity it is applied on.

---

### Algorithm 1: The ACDC algorithm.

---

**Data:** Computational graph  $G$ , dataset  $(x_i)_{i=1}^n$ , corrupted datapoints  $(x'_i)_{i=1}^n$  and threshold  $\tau > 0$ .

**Result:** Subgraph  $H \subseteq G$ .

```
1  $H \leftarrow G$                                 // Initialize H to the full computational graph
2  $H \leftarrow H.\text{reverse\_topological\_sort}()$           // Sort H so output first
3 for  $v \in H$  do
4   for  $w$  parent of  $v$  do
5      $H_{\text{new}} \leftarrow H \setminus \{w \rightarrow v\}$            // Temporarily remove candidate edge
6     if  $D_{KL}(G||H_{\text{new}}) - D_{KL}(G||H) < \tau$  then
7        $H \leftarrow H_{\text{new}}$                            // Edge is unimportant, remove permanently
8     end
9   end
10 end
11 return  $H$ 
```

---

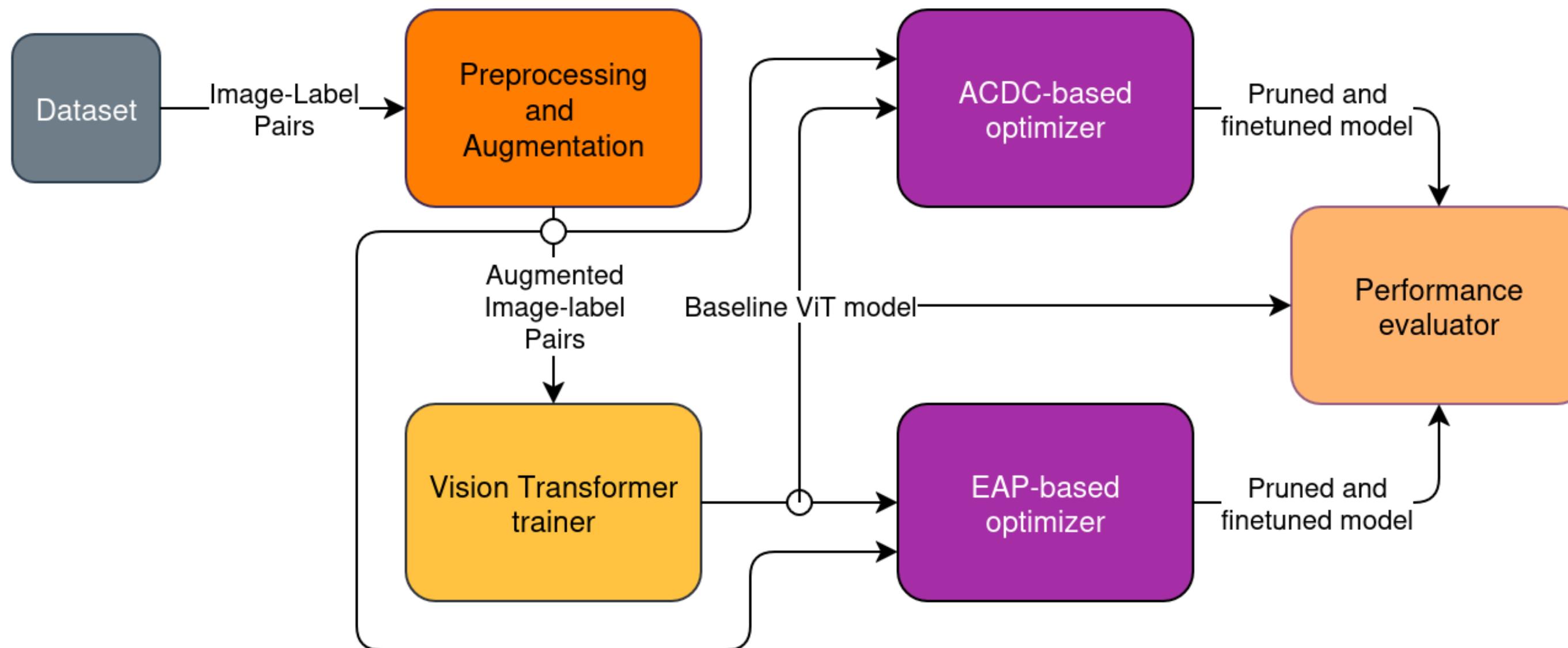
# State of the Art

Newer approach

Edge Attribution Patching (EAP) is a faster, gradient-based approach requiring just two forward passes and one backward pass.

$$L(x_{\text{clean}} \mid \text{do}(E = e_{\text{corr}})) \approx L(x_{\text{clean}}) + \underbrace{(e_{\text{corr}} - e_{\text{clean}})^T \frac{\partial}{\partial e_{\text{clean}}} L(x_{\text{clean}} \mid \text{do}(E = e_{\text{clean}}))}_{\text{Call this } \Delta_e L, \text{ the attribution score.}}$$

# Proposed method



# Dataset

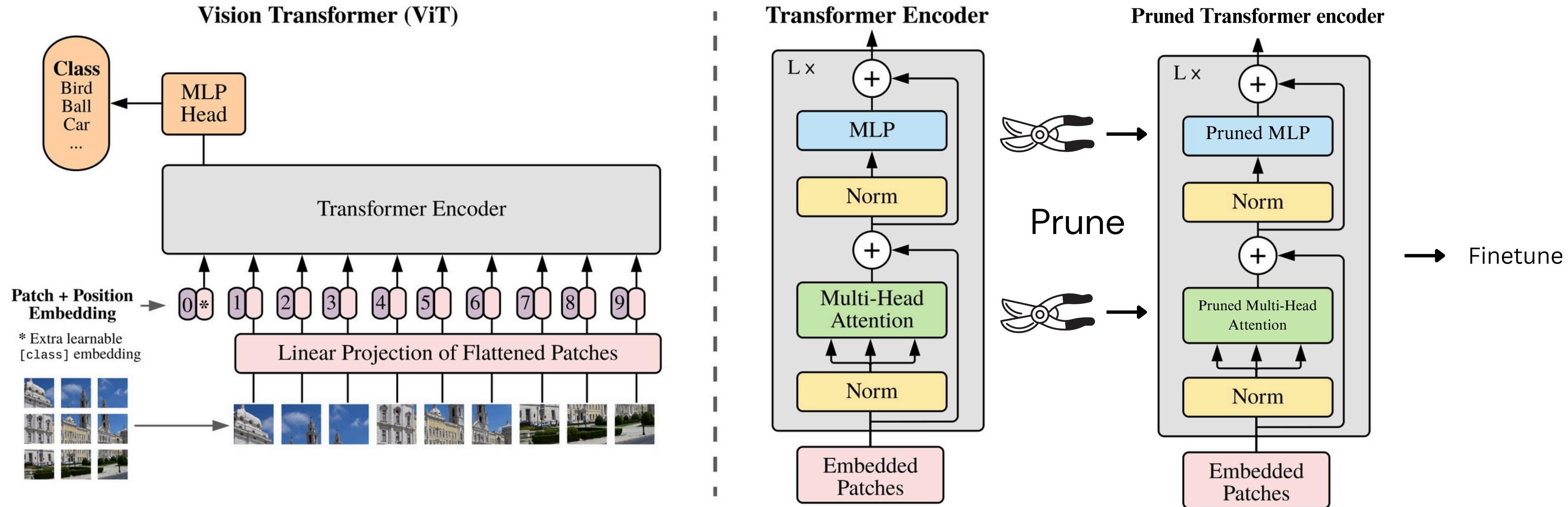
Tiny ImageNet: a compact version of ImageNet

Size & Content:

- 100,000 images across 200 classes
- 64×64 resolution, colored (RGB)



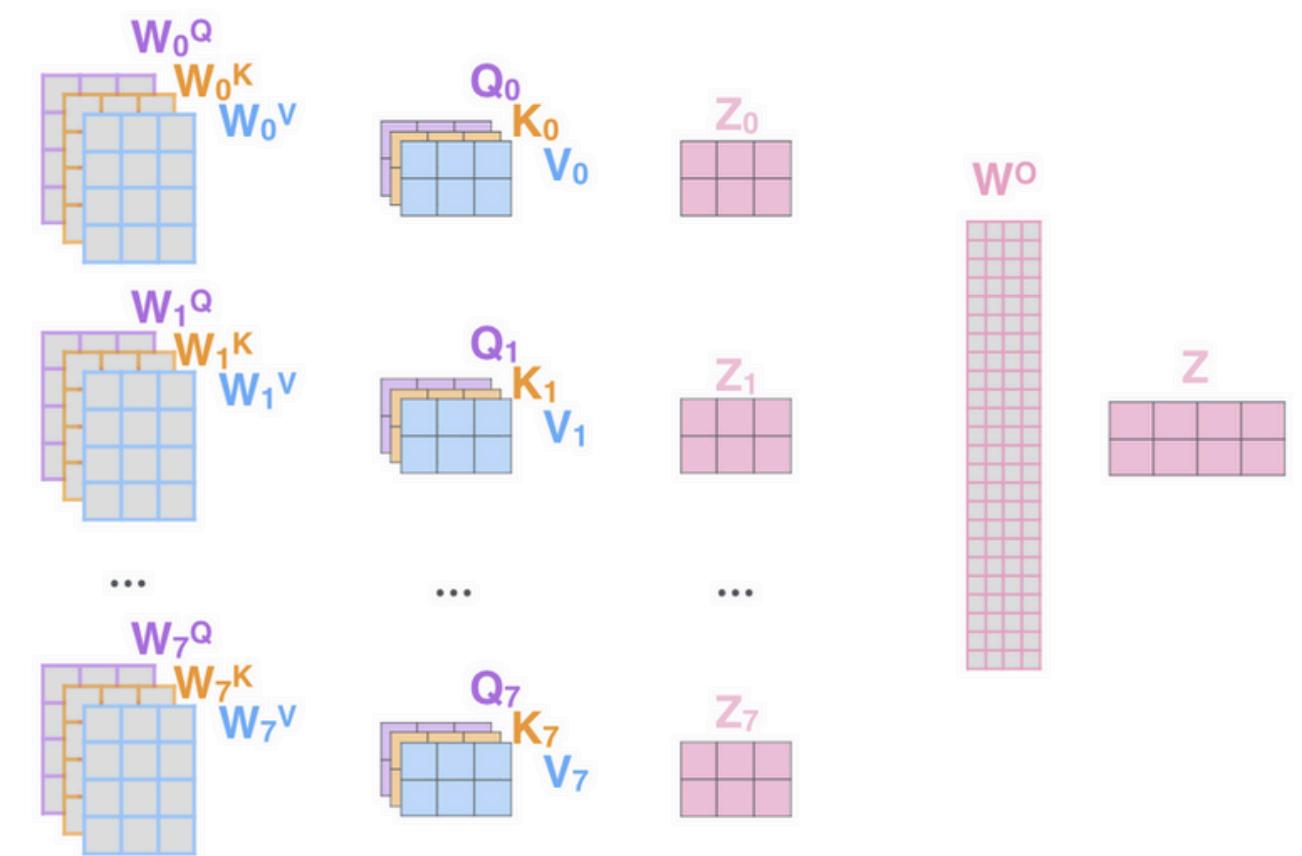
# Experimental Setup



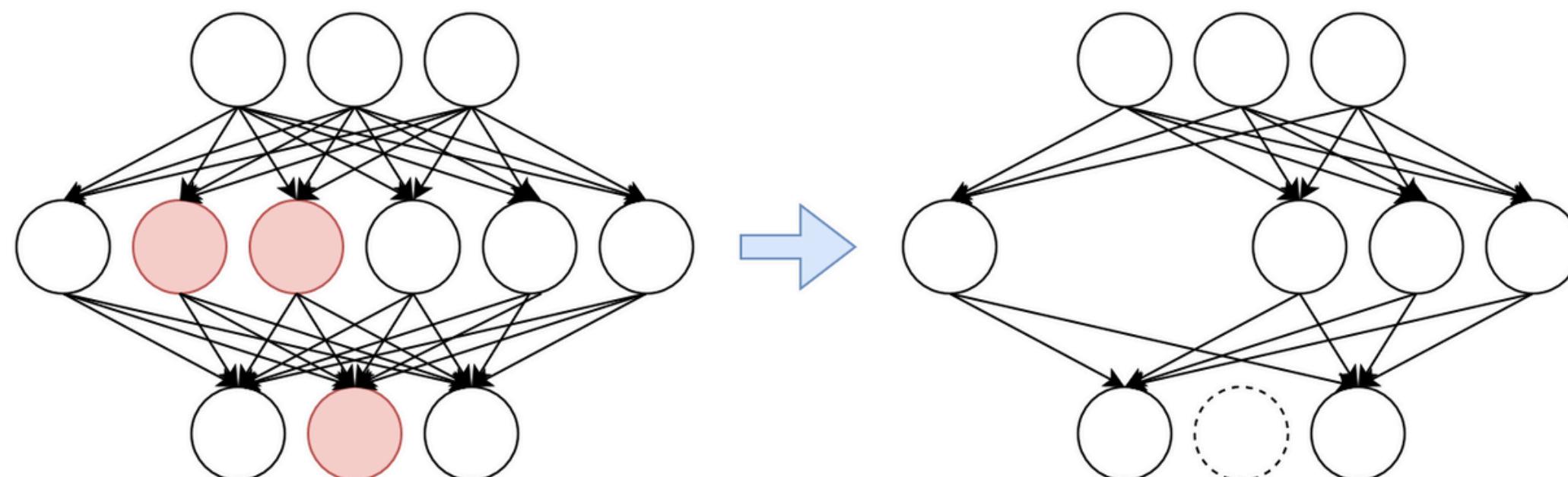
- **ACDC pruning:** corrupt attention heads and neuron chunks, keep only those that negatively affect the KL divergence above a certain threshold
- **EAP pruning:** corrupt attention heads and neuron chunks, evaluate their score, then keep only the top performers

# Experimental Setup

The pruned multi-head attention is formed by pruning entire attention heads, and computing smaller QKV matrices.



Pruned MLP have entire nodes removed. If output nodes are removed, their missing response gets “padded” with zeroes to keep the original embedding dimensions for subsequent layers.



# Experimental Setup

[Insert lots of training and testing with different parameters]

```
# Device configuration
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Dataset parameters
DATA_PATH = './tiny-imagenet-200' # Tiny-ImageNet has 200 classes
# Tiny-ImageNet images are 64x64
NUM_CLASSES = 200
IMAGE_SIZE = 64

# Model architecture parameters
PATCH_SIZE = 8 # (IMAGE_SIZE // PATCH_SIZE) ** 2
NUM_PATCHES = 96
HEAD_DIM = 6
NUM_HEADS = 6
EMBED_DIM = HEAD_DIM * NUM_HEADS
MLP_DIM = 1536
DROPOUT = 0.1

# Training parameters
BATCH_SIZE = 64
LEARNING_RATE = 1e-4
NUM_EPOCHS = 50
WEIGHT_DECAY = 0.01
NUM_WORKERS = 4
ABEL_SMOOTHING = 0.1 # Label Smoothing
AD_CLIP = 1.0 # Label clipping

# Apply Attribution Patching
print(f"\nApplying Attribution Patching with keep_ratio={args.keep_ratio}...") if val_acc > best_val_acc:
    patcher = AttributionPatcher(model, train_loader, device=DEVICE, use_clean_label_smoothing=True, keep_ratio=args.keep_ratio)
    attribution_scores = patcher.compute_attribution_scores(num_batches=args.analysis_batches)
    select_components_to_remove(select_components_to_remove(attribution_scores))

# DC optimization parameters
THRESHOLD = 0.01 # Threshold for DC optimization
ANALYSIS_BATCHES = 30 # Number of batches for analysis
# Fraction of components to remove
IS_BATCHES = 20 # Number of batches for DC optimization
# Number of components to remove
IS_COMPONENTS = 5 # Number of components to remove

# Select components to remove
select_components_to_remove(select_components_to_remove(attribution_scores))

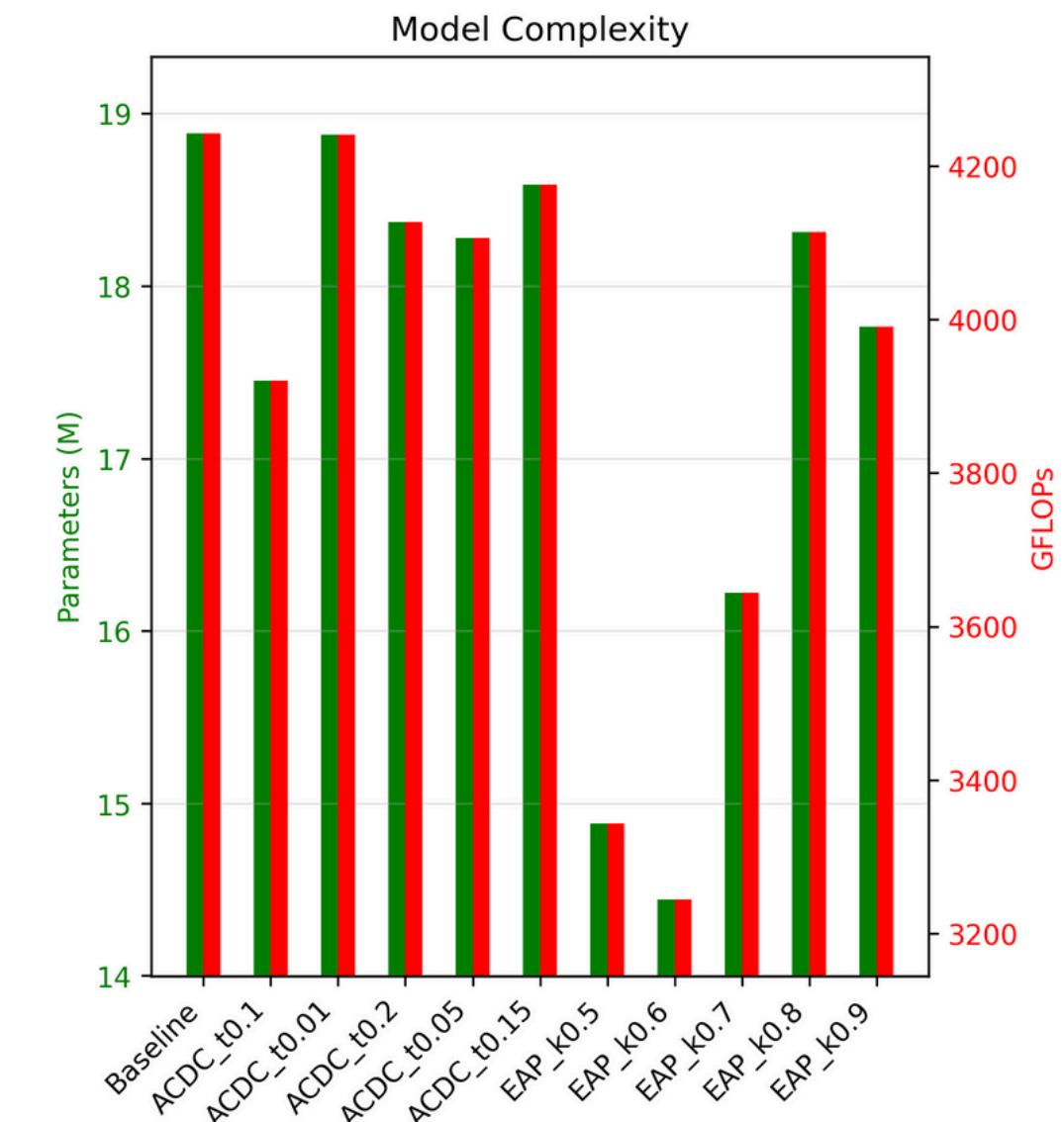
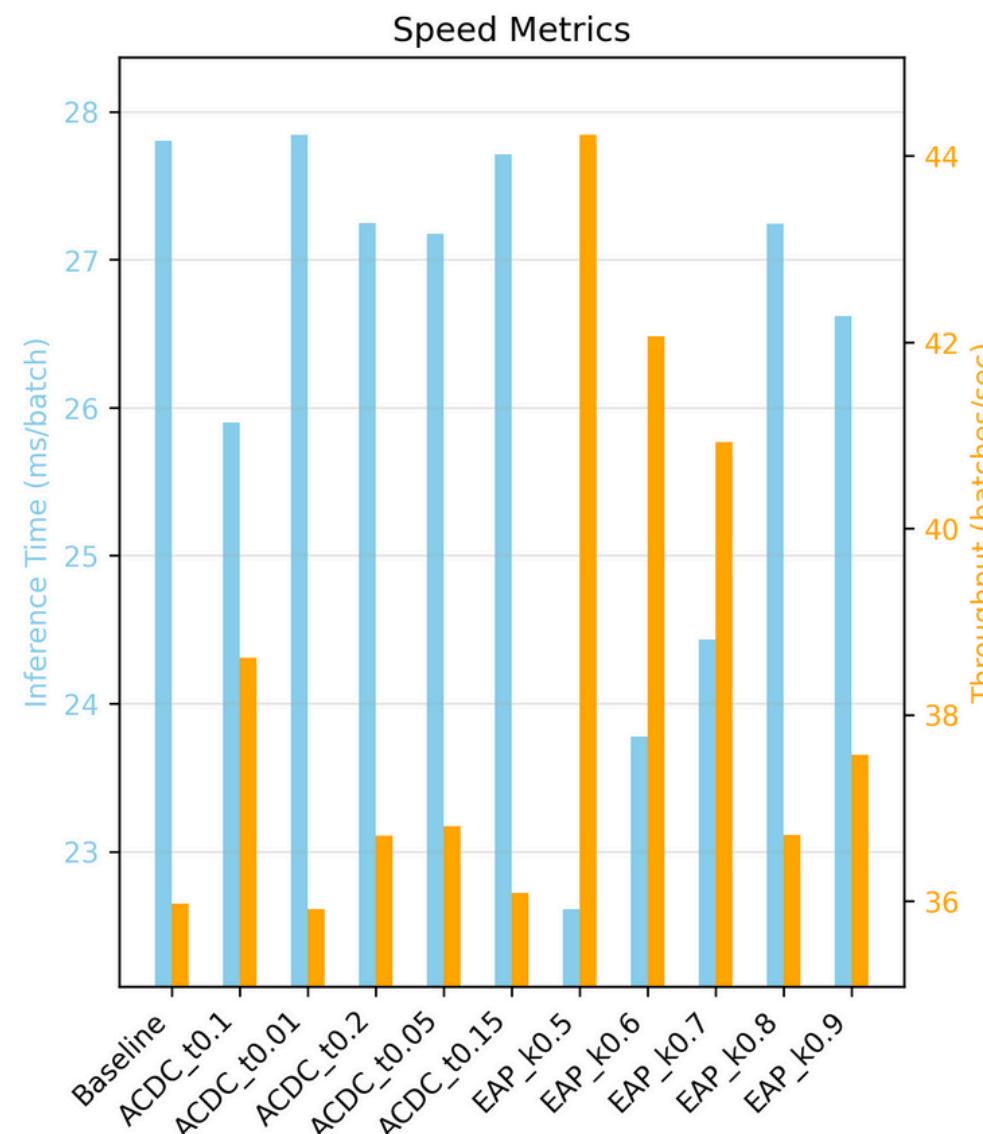
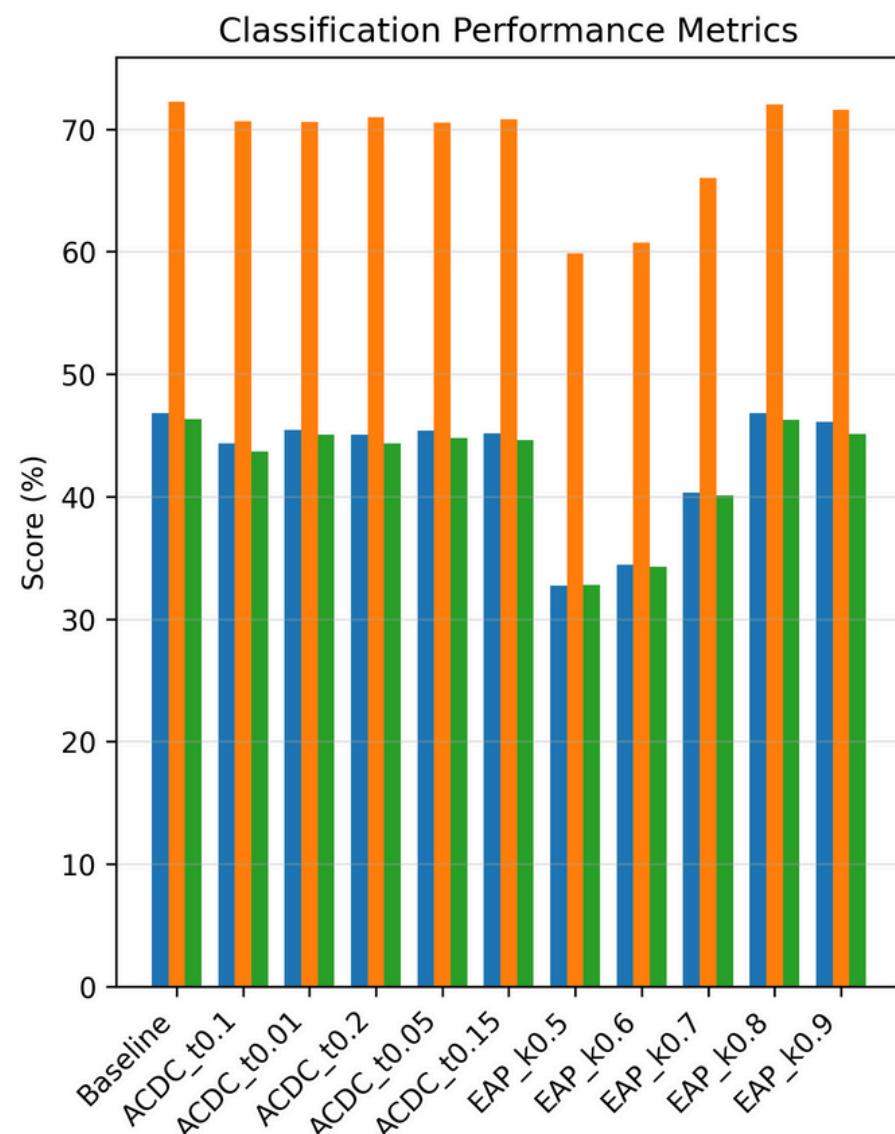
# Apply ACDC optimization
print(f"\nApplying ACDC optimization")
acdc = ACDCOptimizer(model, train_loader, threshold=args.threshold)

# Run ACDC algorithm (replaces analyze_edges, create_edge_masks, and apply_masks)
optimized_model, removed_components = acdc.apply_acdc_optimization(
    num_batches=args.analysis_batches)

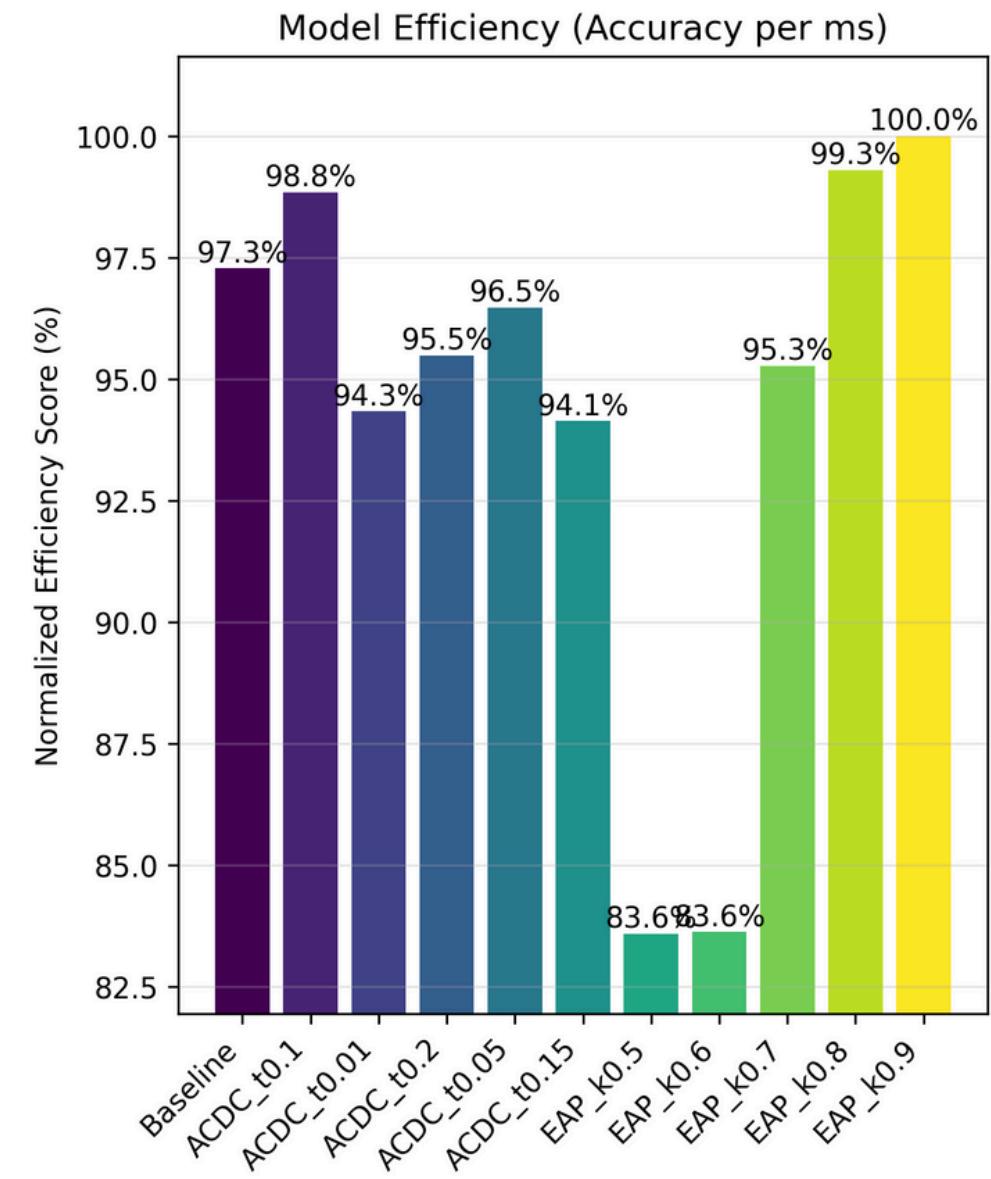
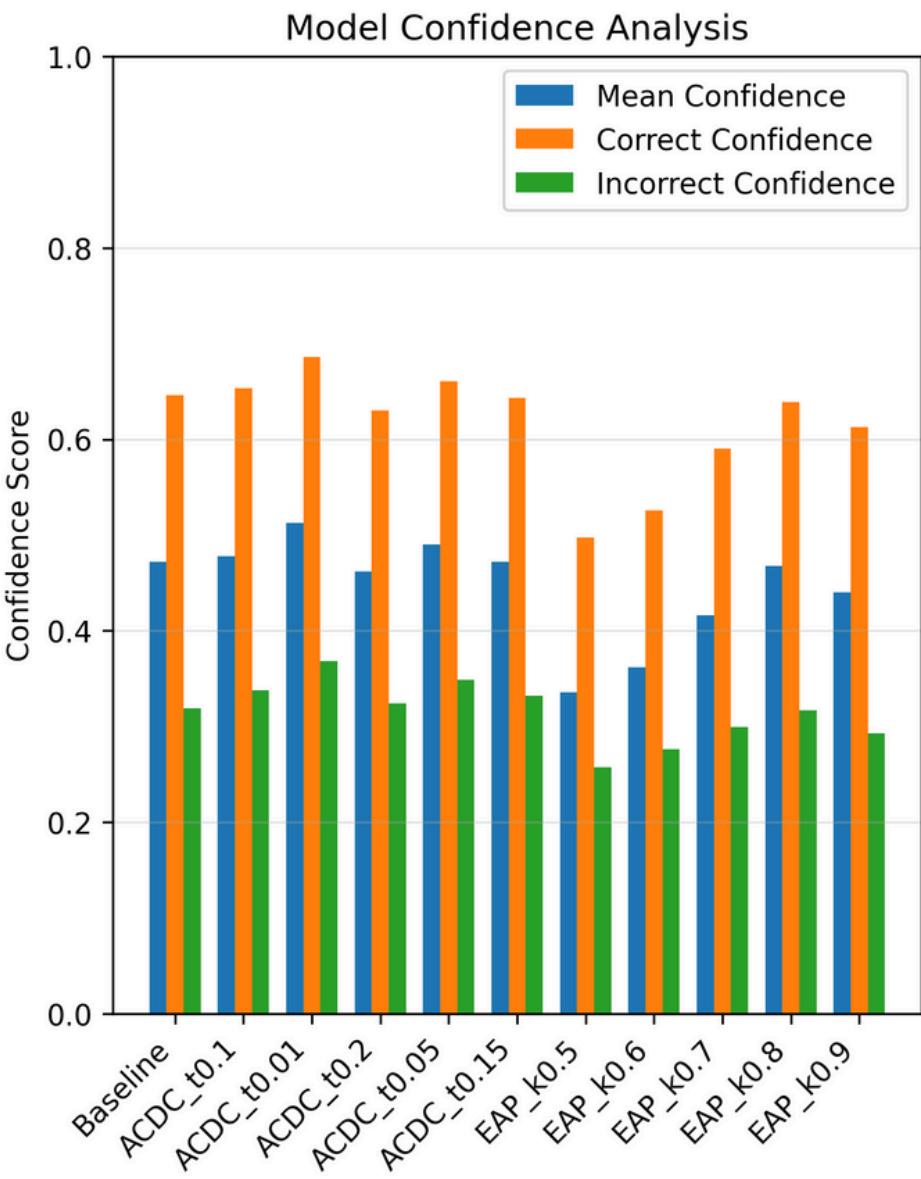
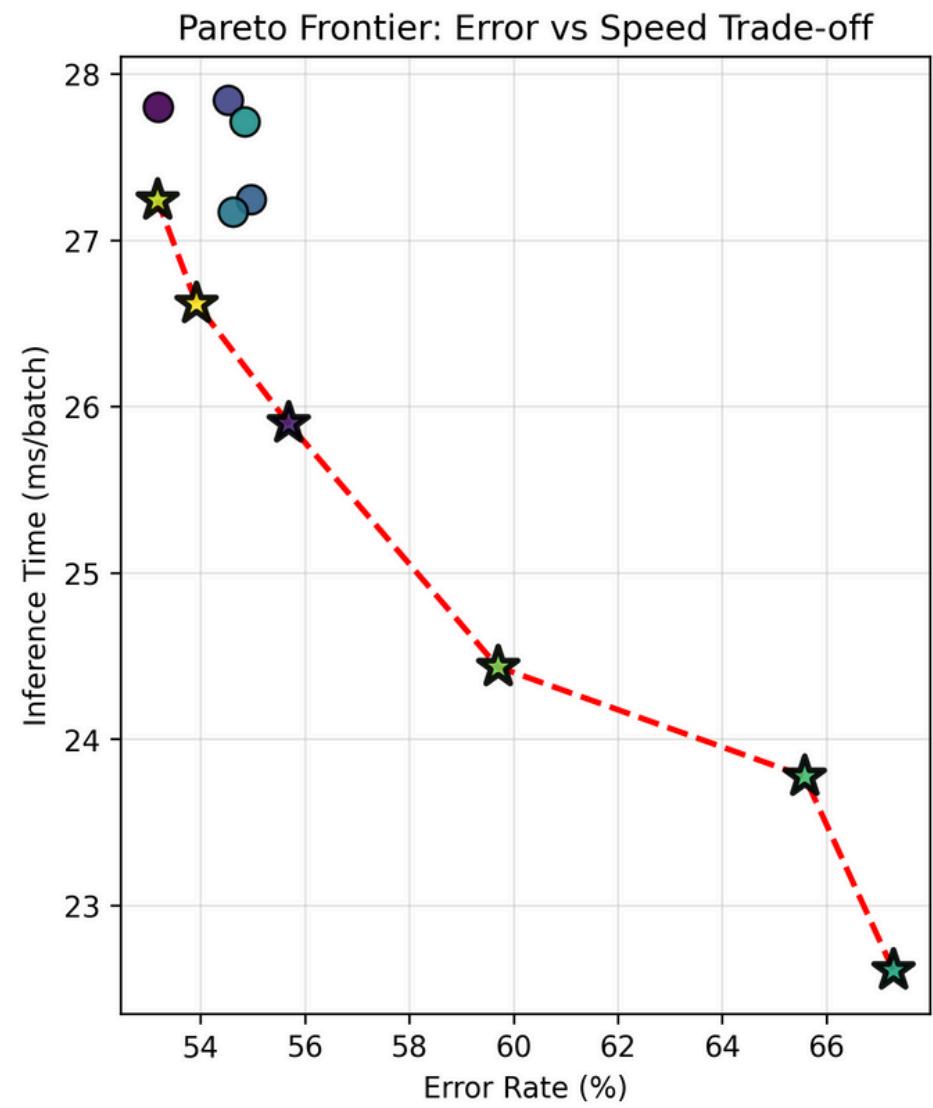
# Save best model
best_val_acc = val_acc
torch.save({
    'epoch': epoch,
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'val_acc': val_acc,
    'train_history': train_history,
    'config': {
        'embed_dim': EMBED_DIM,
        'num_heads': NUM_HEADS,
        'num_layers': NUM_LAYERS,
        'mlp_dim': MLP_DIM,
        'patch_size': PATCH_SIZE,
        'image_size': IMAGE_SIZE,
        'is_batches': IS_BATCHES,
        'is_components': IS_COMPONENTS
    }
}, f'model_{epoch}.pt')
```

# Model Evaluation

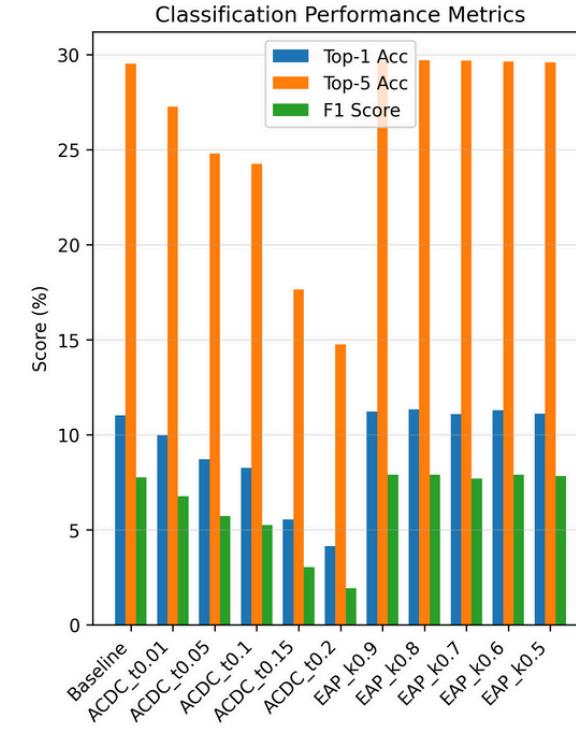
Typical results for a sample evaluation:



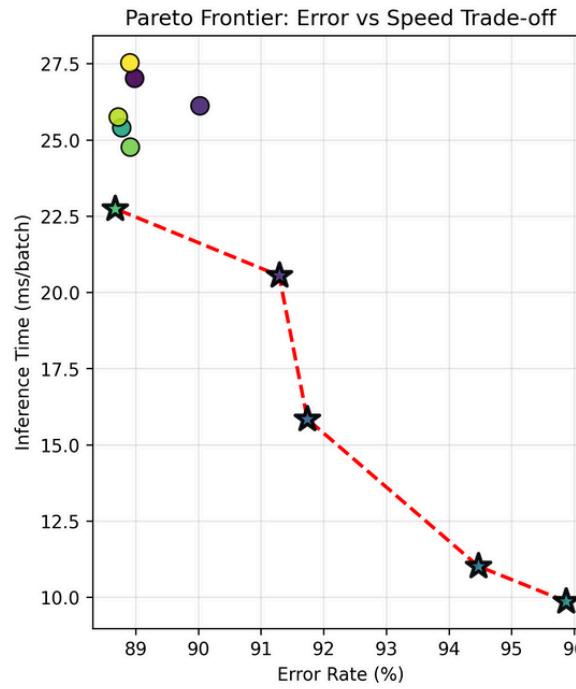
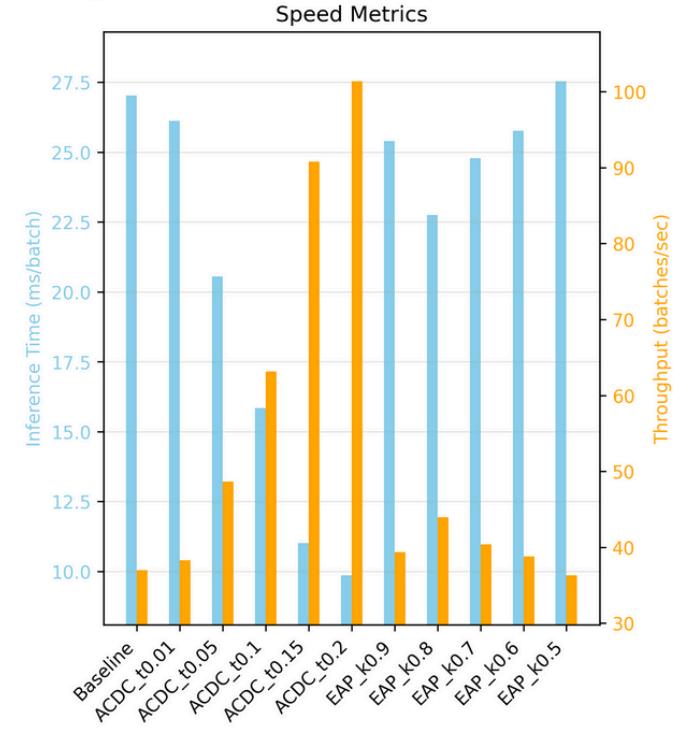
# Model Evaluation



# Model Evaluation

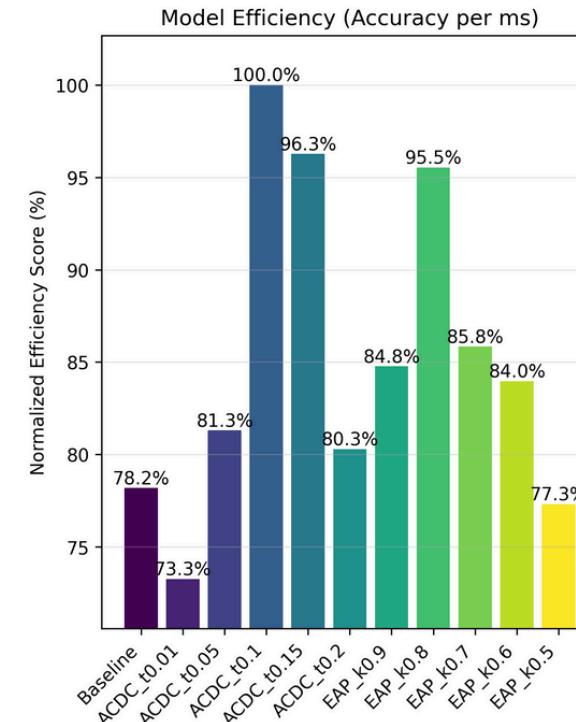
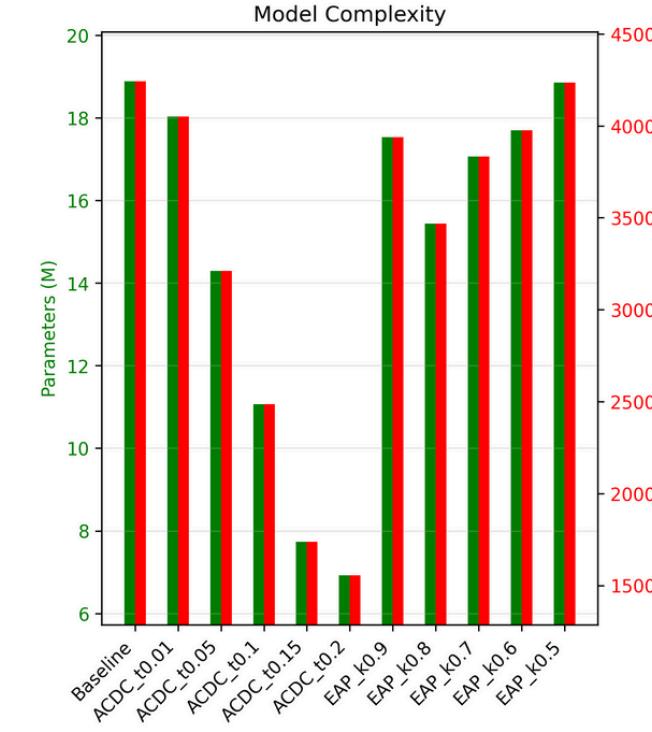
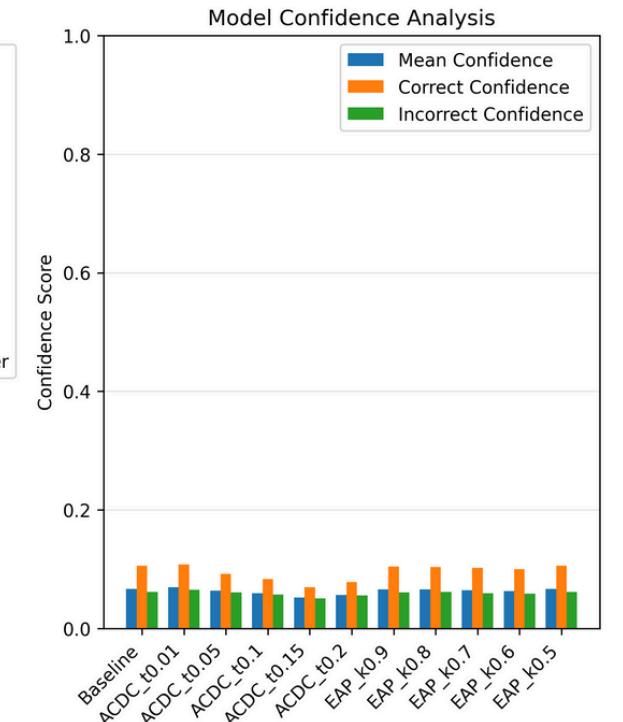


**Comprehensive Model Evaluation Results**



Legend:

- Baseline
- ACDC\_t\_0.01
- ACDC\_t\_0.05
- ACDC\_t\_0.1
- ACDC\_t\_0.15
- ACDC\_t\_0.2
- EAP\_k\_0.9
- EAP\_k\_0.8
- EAP\_k\_0.7
- EAP\_k\_0.6
- EAP\_k\_0.5
- Pareto Frontier



Among the different ViT sizes, hyperparameters and optimization parameters tested, the **only** scenarios where ACDC managed to beat EAP, was with undertrained (badly performing) starting baseline models

# Conclusions

Proposed approach benefits:

- Enables smaller, faster networks while maintaining accuracy
- Consistently pushes the Pareto frontier (inference time vs. error rate)
- Reduces parameters, making models better for edge devices

EAP consistently outperforms ACDC in optimization time and optimized model results.

ACDC greatly benefits from the fine-tuning steps, regaining a lot of final model performance, allowing it to be competitive with EAP in some occasions.

# Future work

Future work could explore:

- Testing on larger base models for broader applicability
- Exploring more granular patching
- Dynamic pruning: automatically find the minimal set of circuits needed to preserve model performance, optimize pruning by targeting specific output thresholds

# References:

1

A. Conmy et al. (2023). *Towards Automated Circuit Discovery for Mechanistic Interpretability*. In: Advances in Neural Information Processing Systems 36 (NeurIPS 2023)

2

A. Syed, C. Rager and A. Conmy, (2024). *Attribution Patching Outperforms Automated Circuit Discovery*, BlackboxNLP 2024.

3

A. Amangeldi, A. Taigonyrov, M. H. Jawad, C. E. Mbonu (2025) CNN and ViT Efficiency Study on Tiny ImageNet and DermaMNIST Datasets Available at: <https://arxiv.org/pdf/2505.08259>

# References:

4

C. Schiavella, L. Cirillo, L. Papa, P. Russo, and I. Amerini, (2023). Optimize vision transformer architecture via efficient attention modules: a study on the monocular depth estimation task.

5

PyTorch Documentation. (2024). *PyTorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration*. Available at: <https://pytorch.org/docs/>.