1. Summary Document on VILA[1]

1.1 Key Points

Focus: VILA explores pre-training strategies for VLMs, enhancing the alignment between visual and textual modalities.

Objective: To integrate vision capabilities into LLMs while preserving text-only functionalities.

Architecture: Utilizes an auto-regressive design where visual tokens are processed like textual tokens, making it a flexible and unified framework for multi-modal inputs.

Applications: Excels in vision-language tasks such as Visual Question Answering (VQA), caption generation, and multi-image reasoning.

1.2 Technical Contributions

Pre-training Strategies:

- Demonstrates that fine-tuning LLMs during visual language pre-training is critical for deep embedding alignment and in-context learning.
- Highlights the benefits of using interleaved image-text datasets, which maintain better alignment and minimize text-only capability degradation compared to plain image-text pairs.

Data Blending:

- Reintroduces text-only instruction data during supervised fine-tuning to recover degraded text-only capabilities and boost VLM task performance.
- Proposes blending interleaved datasets with image-text pairs to enhance diversity and downstream task accuracy.

Performance:

- Consistently outperforms state-of-the-art models like LLaVA-1.5 across multiple benchmarks.
- Demonstrates robust multi-image reasoning and improved world knowledge retention.

Efficiency:

- Employs scalable techniques like resolution adjustments and lightweight projection layers for better performance-cost trade-offs.

1.3 Areas for Improvement

Scaling:

- Limited pre-training data (~50M images) compared to billion-scale datasets used in other works.
- Expanding the training dataset could further improve results.

Instruction Dataset Quality:

- While effective, the instruction-tuning dataset could benefit from greater diversity and higher quality prompts.

Edge Deployment:

- Although deployable on devices like Jetson Orin, the current model size may still pose challenges for resource-constrained environments.

Generalization:

- While VILA retains competitive text-only capabilities, smaller models show more degradation, indicating room for improvement in preserving text-only skills during pre-training.

---

2. Efficiency Optimization Report: Dynamic Token Pruning in Transformer Models

2.1. Introduction

Transformer models are computationally expensive due to their high reliance on token-level computations in self-attention and feedforward layers. Reducing these costs while maintaining accuracy is critical for deploying efficient models in real-world scenarios. This report documents the process of identifying, optimizing, and evaluating efficiency bottlenecks in a transformer model using **Dynamic Token Pruning**.

2.2 Efficiency Bottleneck Identification

2.2.1 Profiling Results

We performed profiling on a baseline transformer model to identify the primary efficiency bottlenecks. The results are summarized below:

Major Bottlenecks:

1. **Self-Attention Layers**: Dominant contributor to FLOPs and computational time.
2. **Feedforward Layers (Linear Layers)**: Responsible for secondary computational costs.

Baseline Model FLOPs and Time

FLOPs: **12.908G**

CUDA Time: **12.490ms**

2.2.2 Conclusion

The inefficiency arises from redundant computations on tokens that contribute little to the final model output. To address this, we focus on optimizing token usage through **Dynamic Token Pruning**.

2.3 Optimization Method: Dynamic Token Pruning

2.3.1 Overview

Dynamic Token Pruning is a method that adaptively removes tokens with low relevance (as measured by saliency scores) from computation. This reduces the sequence length dynamically during inference, leading to significant FLOPs and time savings.

2.3.2 Algorithm

1. **Compute Saliency Scores**: Each token's importance is estimated using its L2 norm .
2. **Generate Token Mask**: Tokens with saliency scores below a predefined threshold are pruned.
3. **Layer-Wise Pruning**: Pass the remaining tokens to the next transformer block.
4. **Final Classification**: Aggregate the output from all active tokens.

2.3.3Pseudo Code

```
class PrunedTransformerEncoder(nn.Module):
    def forward(self, src):
        keep_tokens = torch.ones(src.shape[:2], device=src.device).bool()
        for i, layer in enumerate(self.layers):
            saliency = self.token_pruning.compute_saliency(src)
            keep_tokens = keep_tokens & (saliency > self.token_pruning.saliency_threshold)
            src = layer(src, keep_tokens=keep_tokens)
        return src
```

2.3.4 Implementation Details

Saliency Score: Computed as the L2 norm of each token vector.

Threshold: Adjustable parameter (e.g., `13.0` in this implementation).

Token Mask: Dynamically updated across layers, preserving only the most relevant tokens.

2.4 Results and Evaluation

2.4.1 Accuracy

The optimized model maintains the same accuracy as the baseline:

Baseline Model Accuracy: **52.5%**

Pruned Model Accuracy: **52.5%**

2.4.2 FLOPs and Time Comparison

Model	FLOPs	CUDA Time	FLOPs Reduction	Time Reduction
Baseline	12.908G	12.490ms	-	-
Pruned	8.874G	9.968ms	**31.2%**	**20.2%**

The pruning method significantly reduces FLOPs and computational time while preserving accuracy.

2.4.3 Profiling Results

Baseline Model

- FLOPs: **12.908G**
- CUDA Memory Usage: **72.02MB**

- CUDA Time: **12.490ms**

Pruned Model

- FLOPs: **8.874G**
- CUDA Memory Usage: **72.02MB**
- CUDA Time: **9.968ms**

2.4.4 Efficiency Analysis

Dynamic Token Pruning successfully reduces computation while maintaining the accuracy of the model. Profiling indicates that FLOPs and execution time reductions are most prominent in the self-attention layers.

2.5 Discussion and Considerations

2.5.1 Strengths

Efficiency Gains:

- Reduced FLOPs and execution time, achieving over **31.2%** FLOPs reduction.
- Preserves accuracy on binary classification tasks.

Scalability:

- The pruning method is layer-wise and dynamic, allowing integration into large-scale models.

2.5.2 Limitations

Static Threshold: The saliency threshold is fixed, which may not generalize well across diverse datasets.

Evaluation on Toy Data: While accuracy is maintained on simulated tasks, real-world datasets may require further validation.

2.6. Conclusion

This report demonstrates the feasibility of **Dynamic Token Pruning** in optimizing transformer efficiency. By dynamically reducing sequence length through saliency-based pruning, we achieved:

1. Significant reductions in FLOPs and execution time.
2. Maintenance of accuracy on a binary classification task.

Future work will involve validating this approach on larger datasets and exploring adaptive thresholds for saliency computation.

Reference:

[1] Ji Lin, Hongxu Yin, Wei Ping, Yao Lu, Pavlo Molchanov, Andrew Tao, Huizi Mao, Jan Kautz, Mohammad Shoeybi, and Song Han. Vila: On pre-training for visual language models, 2023.

Appendix:

Github links:

codes: