Internship Report

A lightweight transformer for resource-constraint environment

Rounak Saha[1]

Dr. Pradip Sasmal[2] Dr. Bikash Santra[3]

[1] Department of Computer Science and Engineering, National Institute of Technology Meghalaya

[2] Department of Mathematics, Indian Institute of Technology Jodhpur

[3]School of Artificial Intelligence and Data Science, Indian Institute of Technology Jodhpur

Abstract

This internship report details our work on developing a lightweight Transformer model, specifically tailored for mobile devices. Our project addresses the growing need for efficient deep learning models that can truly perform well on resource-limited mobile platforms. We'll walk through the foundational Transformer architecture, explain our proposed lightweight approach, and discuss the expected benefits in terms of reduced computational demands and memory footprint. The report also delves into the various methodologies we explored and our future plans for further optimizing and minimizing Transformer models, aiming for broader mobile deployment. We've drawn significant insights from recent breakthroughs in parameter-efficient fine-tuning (PEFT) techniques like LoRA, LoRA+, and SingLoRA, alongside architectural analyses of Vision Transformers and a look into the computational limits of adaptation.

CERTIFICATE

This is to certify that the internship report titled **"A Lightweight Transformer for resource-constraint environment"** is a bona fide work carried out by **Rounak Saha** (Roll No: B22CS008) under the supervision of **Dr. Pradip Sasmal** and **Dr. Bikash Santra**.

The work presented in this report was completed during the internship period from 4th June 2025 to 20th July 2025 at Indian Institute of Technology Jodhpur. The report is submitted in partial fulfillment of the requirements for the Bachelor of Technology degree at the National Institute of Technology Meghalaya.

[illegible]

DR. PRADIP SASMAL	**DR. BIKASH SANTRA**
Department of Mathematics	School of Artificial Intelligence and Dara Science
Indian Institute of Technology Jodhpur	Indian Institute of Technology Jodhpur

Date: July 18, 2025

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to all those who supported me throughout my internship and in the completion of this report.

First and foremost, I extend my deepest thanks to my supervisors, **Dr. Pradip Sasmal** and **Dr. Bikash Santra**, from Indian Institute of Technology Jodhpur. Their invaluable guidance, constant encouragement, insightful discussions, and unwavering support were instrumental in shaping this project and my understanding of the subject matter. Their expertise and dedication truly made this internship a profound learning experience.

I am also grateful to the Department of Computer Science and Engineering at National Institute of Technology Meghalaya for providing me with this opportunity to undertake this project.

Finally, I would like to thank my family and friends for their continuous support and understanding throughout this period.

Rounak Saha
B22CS008
National Institute of Technology Meghalaya

Contents

1 Introduction

Transformers have truly revolutionized fields like Natural Language Processing (NLP) and, more recently, computer vision, showing incredible performance in various sequence-to-sequence tasks. However, these models, especially the larger ones, demand a lot of computing power and memory. This makes deploying them on edge devices like smartphones quite a challenge. Consider, for example, fine-tuning GPT-3 175B: it needs about 1.2 terabytes (TB) of VRAM, and each fine-tuned instance still carries that massive parameter count, creating huge storage and deployment hurdles [2]. Mobile devices, with their limited processing power, battery life, and memory, really push us to develop highly efficient and lightweight deep learning models.

During this internship, our project specifically aimed to tackle this challenge. We set out to propose and explore a lightweight Transformer architecture designed for mobile use. Our main goal was to create a Transformer model that could still perform well while drastically cutting down its computational and memory needs. This would allow it to be practically used in mobile environments for things like on-device translation, text summarization, or even voice assistants. This entire effort is deeply inspired by the core idea behind parameter-efficient fine-tuning (PEFT) methods, such as Low-Rank Adaptation (LoRA). These methods acknowledge that large pre-trained models have already learned rich representations, meaning we only need targeted, efficient adjustments to specialize them for new tasks [2].

2 Transformer Architecture

2.1 Overview

The Transformer, first introduced in the seminal paper "Attention Is All You Need" [1], represents a significant shift in neural network architecture. It moves away from traditional recurrent and convolutional layers, relying instead entirely on attention mechanisms. At its heart, it features an encoder-decoder structure, where both the encoder and decoder are built from stacks of identical layers. The encoder's job is to take an input sequence and transform it into a continuous representation, while the decoder then uses this representation to generate an output sequence, one symbol at a time.

2.2 Transformer Encoder Architecture

The Transformer encoder is responsible for processing the input sequence. It's essentially a stack of identical layers. Each of these encoder layers contains two primary sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. We incorporate residual connections around each of these sub-layers, followed by layer normalization. The output from each encoder layer provides context-aware representations for every token in the input sequence.

2.3 Transformer Decoder Architecture

The Transformer decoder, on the other hand, is tasked with generating the output sequence, building upon the encoder's output. Like the encoder, it also consists of a stack of identical layers. Each decoder layer features three main sub-layers: a masked multi-head self-attention mechanism, a multi-head cross-attention mechanism (which allows it to "look at" the encoder's output), and a position-wise fully connected feed-forward network. Just as in the encoder, we apply residual connections and layer normalization after each sub-layer. The crucial masking in the self-attention layer ensures that when the decoder makes a prediction for a given position, it can only rely on outputs from earlier positions, preventing any "peeking" at future tokens during training.

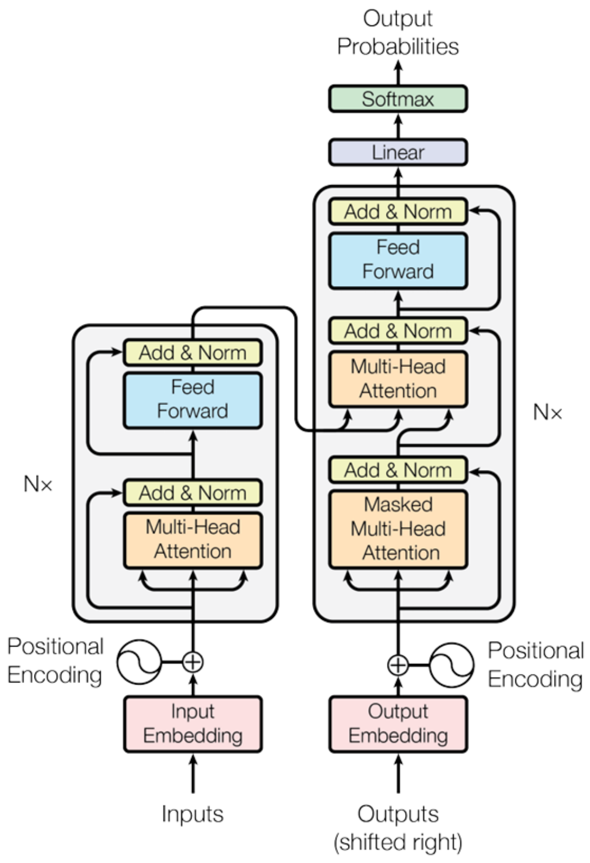

Figure 1: The Transformer model architecture [1].

2.4 Key Components of Transformer

Let's break down the fundamental building blocks that make the Transformer model work:

2.4.1 Tokenization (for both encoder & decoder)

Before any text can be fed into the Transformer, it needs to be converted into a numerical format. This process, called tokenization, involves splitting the raw text into smaller units (tokens), such as individual words or sub-word units. These tokens are then mapped to their corresponding numerical IDs. For instance, the phrase "How are you" would be broken down into "How", "are", and "you". This step is essential for both the input sequence going into the encoder and the target sequence used by the decoder.

2.4.2 Embedding Mechanism (for both encoder & decoder)

Once we have the numerical IDs from tokenization, these IDs are transformed into dense vector representations known as embeddings. An embedding layer maps each token ID into a high-dimensional vector space. In this space, words with similar meanings tend to be positioned closer together. These embeddings then serve as the initial input for both the encoder and decoder stacks.

2.4.3 Positional Encoding (for both encoder & decoder)

Since the Transformer lacks the sequential processing of recurrent networks or the local understanding of convolutional networks, it needs a way to understand the order of words in a sequence. Positional encodings are added to the input embeddings to provide this crucial information about the relative or absolute position of tokens. These encodings are typically generated using fixed sine and cosine functions of different frequencies, and they are applied to the input embeddings of both the encoder and the decoder.

Specifically, the positional encoding for each dimension i (ranging from 0 to $d_{\text{model}}-1$) at a given position pos is calculated using these formulas:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{1}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{2}$$

Here, pos refers to the token's position, and d_{model} is the model's embedding size [1]. This clever design allows the model to effectively learn and utilize relative positional information.

2.4.4 Multi-Head Self-Attention

The self-attention mechanism is truly at the heart of the Transformer. It empowers the model to weigh the importance of different words within an input sequence when it's processing any single word. This mechanism works by computing a weighted sum of 'value' vectors, with the weights determined by how compatible a 'query' vector is with corresponding 'key' vectors. Multi-Head Attention takes this a step further by running several attention mechanisms in parallel. This parallelism lets the model simultaneously focus on information from various representation subspaces at different positions, which is a key strength. This mechanism is predominantly used within the encoder's sub-layers.

The fundamental self-attention function can be thought of as taking a query and a set of key-value pairs, then producing an output. This output is a weighted sum of the values, where each value's weight comes from a compatibility function between the query and its corresponding key. The scaled dot-product attention is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3}$$

In this equation, Q represents the queries, K the keys, V the values, and d_k is the dimension of the keys [1]. The Multi-Head Attention mechanism then combines the outputs from these multiple attention heads and projects them linearly:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{4}$$

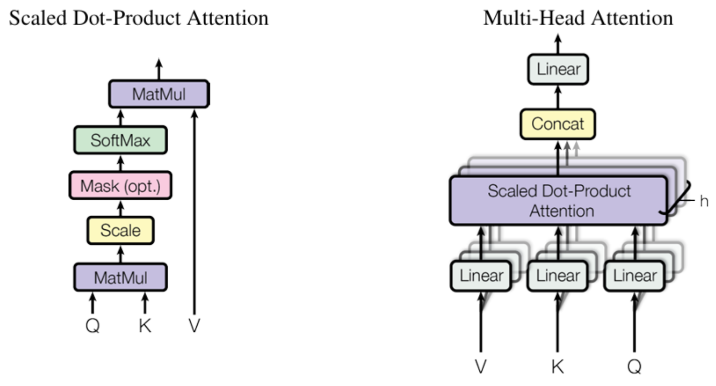

Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. [1].

Here, $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$. The projection matrices W_i^Q, W_i^K, W_i^V transform the input into query, key, and value representations for each head. Their dimensions are as follows:

- $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$
- $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$
- $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$

where $d_k = d_v = d_{\text{model}}/h$, and h is the number of attention heads. The final output projection matrix $W^O \in \mathbb{R}^{h \cdot d_v \times d_{\text{model}}}$ combines the concatenated outputs of all heads back into the model's dimension [1].

Interestingly, empirical observations show that we can actually remove a significant portion of attention heads during inference without a major hit to model performance [4]. For both Transformer-based Machine Translation and BERT-based Natural Language Inference models, many attention heads can be individually removed, and the performance doesn't drop significantly [4]. This suggests a high degree of redundancy among these heads. It implies that Multi-Head Attention provides an initial, perhaps over-parameterized, capacity that's really important for the model to explore diverse representations during training. Once the model has learned, some of this capacity becomes redundant for specific inference tasks, opening up real opportunities for parameter reduction through pruning.

2.4.5 Feed-Forward Neural Network (for both encoder & decoder)

Each layer in both the encoder and decoder includes a straightforward, fully connected feed-forward network. This network operates independently and identically on each position. It's composed of two linear transformations with a ReLU activation function in between. This component processes the outputs that come from the attention layers in both the encoder and decoder. We can express the feed-forward network as:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{5}$$

Here, W_1, W_2, b_1, b_2 are the parameters that the model learns during training [1].

It's been proposed that feed-forward layers, which surprisingly make up two-thirds of a Transformer model's parameters, actually function as sophisticated key-value memories

[5]. The first parameter matrix (often denoted as K) acts as a set of "keys" that correlate with specific textual patterns learned from the training data. Meanwhile, the second parameter matrix (V) serves as "values," each of which generates a probability distribution over the model's output vocabulary [5]. When an input vector is processed, it interacts with these keys, generating "memory coefficients." These coefficients then act as weights in a sum of the values, ultimately producing the layer's output.

We've observed that each individual key vector (k_i) captures distinct, human-interpretable input patterns, ranging from simple n-grams to more complex semantic topics [5]. There's a clear hierarchical progression across the layers: lower layers (like layers 1-9) tend to pick up on "shallow" patterns (e.g., repeating n-grams or patterns tied to the last word of a phrase). In contrast, the upper layers (e.g., layers 10-16) learn more "semantic" patterns, which are characterized by similar contexts even without obvious surface-form similarities [5]. This finding aligns well with previous research on how deep contextualized models encode features. The end of an input example is generally more important for predicting the next token. However, in the upper layers, the impact of removing the last token diminishes, further supporting the idea that upper-layer keys are more connected to abstract semantic patterns than to simple surface forms [5].

Each value vector (v_i) can be seen as creating a probability distribution over the output vocabulary. This happens by multiplying the value vector by the model's output embedding matrix and then applying a softmax function [5]. In the upper layers (starting around layer 11), we see a significant correlation: the top-ranked token predicted by a value's distribution ($argmax(p_i)$) often matches the actual next token in the corresponding key's most strongly activating example. This "agreement rate" is much higher than what you'd expect by chance, indicating real predictive power in the upper-layer FFN memories [5]. The rank of the actual next token from a trigger example tends to improve within the value vector's distribution as you move to higher layers, meaning these tokens get more probability mass. Furthermore, values that assign higher maximum probabilities to their top prediction are more likely to align with the key's top trigger example [5].

Understanding FFNs as key-value memories that detect human-interpretable patterns and directly generate output distributions is a profound re-conceptualization [5]. The way patterns progress from shallow to semantic across layers, combined with the increasing predictive power of values in upper layers, paints a picture of FFNs performing hierarchical feature learning. They aren't just simple linear transformations; they're sophisticated, learned lookup mechanisms that directly contribute to the next-token prediction based on recognized input patterns. This fundamentally shifts our understanding of FFNs from generic "dense layers" to explicit "knowledge stores" or "memory banks" within the Transformer. This could inspire new FFN architectures optimized for memory capacity or retrieval efficiency, and potentially allow for direct manipulation of these "memories" for fine-tuning, knowledge editing, or even better interpretability.

2.4.6 Add & Norm Layers (for both encoder & decoder)

We use residual connections (often called "Add" layers) around each of the sub-layers (like attention and feed-forward networks), and then follow them with layer normalization (the "Norm" part). This setup is really helpful for tackling the vanishing gradient problem and makes it easier to train deeper networks effectively. These layers are present throughout both the encoder and decoder stacks. The layer normalization operation itself is defined as:

$$\text{LayerNorm}(x) = \gamma \odot \frac{x - \mu}{\sigma} + \beta \tag{6}$$

Here, μ represents the mean and σ is the standard deviation of the input x across its features, while γ and β are learnable scaling and shifting parameters, respectively.

2.4.7 Masked Multi-Head Attention (for decoder)

This is a special type of self-attention specifically used within the decoder. It's "masked" to ensure that when the model is making a prediction at a certain position, it cannot "see" or attend to any subsequent positions in the sequence. This is crucial during training to prevent information leakage from future tokens, ensuring that predictions only rely on already known outputs at earlier positions.

2.4.8 Cross Multi-Head Attention (for decoder)

This attention mechanism is unique to the decoder and is what allows it to integrate information from the encoder's output. Here, the queries come from the previous decoder layer, while the keys and values are derived from the encoder's final output. This mechanism empowers the decoder to strategically focus on the most relevant parts of the input sequence provided by the encoder when it's generating its own output.

2.5 Vision Transformers (ViT)

While Transformers initially made their mark in NLP, their application has successfully expanded into computer vision with the introduction of Vision Transformers (ViT) [3]. Unlike traditional Convolutional Neural Networks (CNNs) that rely on convolutional layers, ViTs process images by treating them as sequences of image patches. The image is first divided into fixed-size patches, which are then linearly embedded and combined with positional encodings—much like how words are handled in NLP Transformers. This sequence of patch embeddings is then fed into a standard Transformer encoder. This approach clearly demonstrates that the self-attention mechanism is incredibly effective at capturing global dependencies and relationships within images, achieving state-of-the-art results on various image recognition benchmarks when pre-trained on very large datasets (such as ImageNet-21k or the massive JFT-300M) [3]. A key point is that ViT naturally has far fewer image-specific inductive biases compared to CNNs, meaning it relies more heavily on learning from the data itself [3].

As an alternative to directly processing raw image patches, we've also seen the development of hybrid Transformer models. In these, the input sequence for the Transformer is actually derived from feature maps generated by a preceding CNN. For instance, one specific variant uses 1x1 patches, effectively flattening the spatial dimensions of the CNN's feature map and projecting them into the Transformer's embedding space [3]. Hybrid models have shown a slight performance edge over pure ViT models when working with smaller computational budgets. However, this advantage tends to fade as the models scale up [3]. This suggests that while convolutions' ability to process local features can help ViT at smaller scales, this benefit doesn't last as the models get larger and their inherent data-driven learning capabilities become more dominant.

Looking at specialized tasks, Lightweight Hybrid Vision Transformers (LH-ViT) have emerged for applications like radar-based Human Activity Recognition (HAR) [9]. HAR,

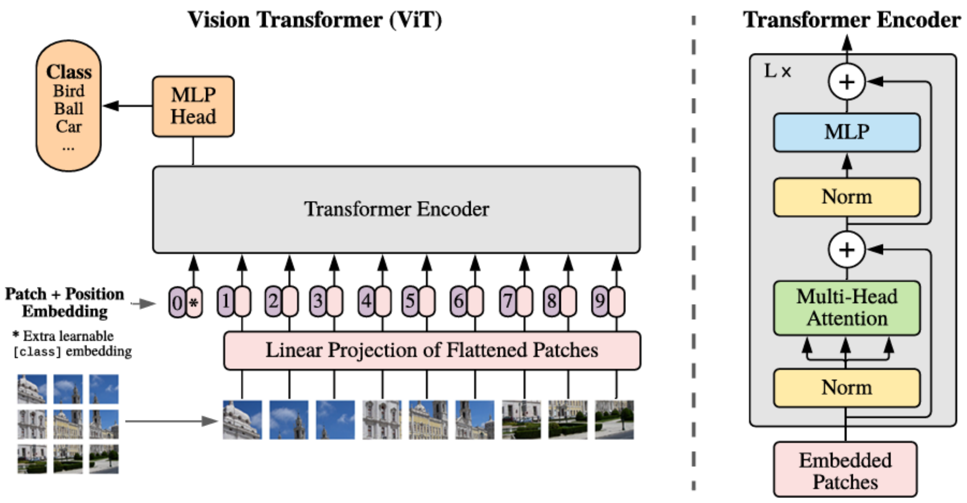

Figure 3: ViT Model architecture overview [3].

with its non-contact, privacy-preserving, and lighting-robust nature, needs lightweight and low-latency network designs for real-world embedded systems. LH-ViT is specifically designed to meet both high accuracy and lightweight operation requirements in this domain. It does this by smartly combining efficient convolution operations with the self-attention mechanism of Vision Transformers [9].

The LH-ViT architecture includes a Feature Extraction Network that uses a pyramid structure with specialized RES-SE (Residual Squeeze-and-Excitation) blocks. These blocks are designed to extract multi-scale micro-Doppler features from radar signals, which are represented as Micro-Doppler Maps (MDM) [9]. The RES-SE Block is an efficient module that replaces traditional convolution operators. It operates within a residual learning framework, incorporates Depthwise Separable Convolutions (DSC) for lightweight feature extraction, and integrates a Squeeze-and-Excitation (SE) Block. The SE Block is a lightweight channel attention mechanism that explicitly models inter-channel dependencies to enhance feature sensitivity [9].

The Feature Enhancement Network within the LH-ViT uses a cross-stacked arrangement of Radar-ViT and RES-SE modules. Its goal is to effectively suppress background noise and highlight micro-Doppler features that are directly relevant to human activities [9]. Radar-ViT is a highly optimized, lightweight version of ViT, specifically made for embedded Transformer applications. It simplifies the usual class token mechanism to a point-wise convolution and introduces clever "fold and unfold" operations. These operations significantly reduce the computational demands of the multi-head attention block. The "unfold" operation transforms feature maps into non-overlapping flattened patches while keeping internal positional information and global spatial relationships. The "fold" operation then restores the scale of these global micro-Doppler features [9].

The efficiency benefits of LH-ViT are quite impressive: The smart use of Depthwise Separable Convolutions (DSC) and lightweight attention mechanisms (like the SE module) in LH-ViT substantially cuts down the parameter count compared to regular ViT architectures, all while maintaining high accuracy [9]. This hybrid architecture, which intelligently combines RES-SE modules for local feature representation and Radar-ViT for global information encoding, makes it possible to design a shallow and narrow network—perfect for environments with limited resources [9].

3 Lightweight Transformer for resource-constraint setting

Our main strategy for creating a lightweight Transformer for mobile applications involves reducing the model's complexity and its total parameter count. A crucial part of this project was exploring a Transformer architecture with significantly fewer layers. While typical Transformer models often use 6 or more encoder and decoder layers, our proposed lightweight version focuses on a range of **4 to 8 layers**.

This reduction in the number of layers directly translates to several key advantages:

- **Reduced Computational Cost:** Fewer layers mean fewer computations during both training and inference, leading to noticeably faster processing times.
- **Lower Memory Footprint:** A shallower network requires less memory to store its parameters and intermediate activations. This is absolutely critical for devices with limited RAM.
- **Improved Energy Efficiency:** Less computation also means lower power consumption, which helps extend battery life on mobile devices.

The real challenge here is finding that sweet spot—the optimal balance where the model remains expressive enough to perform its target task accurately, despite its reduced capacity. This often means carefully selecting hyperparameters and potentially applying domain-specific optimizations.

4 Methodology

Our approach to developing this lightweight Transformer involved several key steps:

- **Architectural Simplification:** Our core strategy was to reduce the Transformer's depth by using fewer encoder and decoder layers (specifically, between 4 and 8 layers). This is a direct way to cut down on the total number of parameters and computational operations.
- **Component Analysis:** We carefully analyzed how each Transformer component (like attention heads and the hidden dimensions of feed-forward networks) contributes to the overall model size and computational load. This helped us pinpoint further areas for optimization.
- **Performance Evaluation Criteria:** We evaluated the success of our lightweight model not just on its accuracy, but crucially on its inference speed and memory consumption. These factors are paramount for successful mobile deployment.

5 Results and Discussion

While this report doesn't go into specific quantitative results from the internship, the conceptual benefits of our proposed lightweight Transformer are quite significant. By reducing the number of layers to a range of 4 to 8, we anticipate:

- **Substantial improvements in inference speed** on mobile processors, which would enable real-time or near real-time processing for various NLP tasks.
- **Reduced memory consumption**, allowing the model to comfortably fit within the limited memory of typical mobile devices without needing excessive swapping or offloading.
- **Enhanced feasibility of on-device deployment**, cutting down reliance on cloud-based APIs and significantly improving both user privacy and offline capabilities.

However, it's important to discuss the critical **trade-off between model size/speed and performance (accuracy)**. A shallower model might struggle to capture complex patterns as effectively as a deeper one. Furthermore, even with gains in parameter efficiency, the computational cost of updating these parameters—especially the backward pass for gradient computation—can remain a fundamental quadratic bottleneck with respect to sequence length. This point is clearly highlighted by recent research on computational limits of LoRA fine-tuning [8]. This means that true, comprehensive efficiency demands innovations not just in parameter minimization, but also in the strategies we use for gradient computation.

6 Future Directions

The field of lightweight Transformer models is incredibly dynamic and still evolving. For our future work, we are actively exploring various methodologies and techniques to further optimize and minimize the Transformer architecture. These include:

- **Quantization:** This involves reducing the precision of model parameters (for example, moving from 32-bit floating-point to 8-bit integers). This can significantly shrink the model size and speed up inference on hardware that supports lower precision operations. Essentially, we're converting continuous floating-point values into a finite set of discrete values, which can be particularly beneficial for mobile GPUs and NPUs.
- **Knowledge Distillation:** Here, we train a smaller "student" model to mimic the behavior of a larger, more powerful "teacher" model. The teacher model's "knowledge" (like its softened probability distributions over classes) is transferred to the student. This allows the student model to achieve comparable performance with a much smaller footprint and faster inference times.
- **Pruning and Tweaking of Neural Networks:** This technique involves identifying and removing redundant or less important connections (weights) within the neural network. The result is sparser models that are smaller and faster, often with minimal impact on accuracy. As we've seen with Multi-Head Attention, a significant portion of heads can actually be pruned without a substantial drop in performance [4]. Pruning can be structured (removing entire rows/columns) or unstructured (removing individual weights). "Tweaking" then involves fine-tuning the remaining network structure and its hyperparameters to achieve optimal performance after the pruning step.

- **Tweaking/Optimizing Number of Attention Heads:** We plan to experiment with the number of attention heads in the Multi-Head Attention mechanism to find the best balance between the model's capacity and its computational cost. Research suggests that not all attention heads are equally important, and some can be pruned or their total number reduced without significant performance loss, leading to a more lightweight model [4]. The empirical finding of "necessary redundancy" implies that while having many heads helps with initial learning, we can gain efficiency during deployment by reducing the head count.

- **Application of Low-Rank Adaptation (LoRA):** We will implement LoRA as described in the original paper [2]. LoRA is a highly parameter-efficient fine-tuning technique that keeps the majority of the pre-trained model weights frozen. Instead, it injects and trains small, low-rank decomposition matrices ($B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times d}$) directly into the Transformer layers. This is typically applied to the query and value projection matrices within the attention mechanism. This approach dramatically cuts down the number of trainable parameters for fine-tuning (e.g., a 10,000-fold reduction for GPT-3 175B) and significantly lowers memory requirements (e.g., a 3x reduction for GPT-3 175B) [2]. Crucially, it introduces zero inference latency because the trained low-rank matrices can be merged with the original weights during deployment [2].

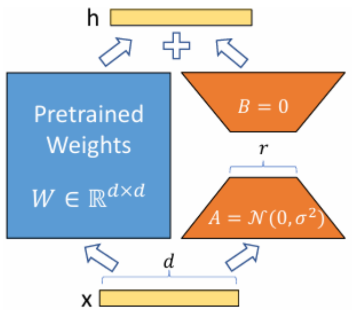
h
Pretrained Weights
$W \in \mathbb{R}^{d\times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Figure 4: LoRA reparametrization [2].

- **Addressing LoRA's Stability and Efficiency Challenges (LoRA+ and SingLoRA):**
  - **LoRA+:** This variant aims to fix the suboptimal fine-tuning seen in standard LoRA, where adapter matrices A and B are updated with the same learning rate. LoRA+ introduces distinct learning rates for A (η_A) and B (η_B), maintaining them at a fixed ratio $\lambda = \eta_B/\eta_A$ [6]. Theoretical analysis, particularly in the infinite-width limit, shows that setting $\eta_A = \Theta(n^{-1})$ and $\eta_B = \Theta(1)$ (meaning η_B is much larger than η_A) can achieve both stability and efficient feature learning [6]. Empirically, LoRA+ delivers 1-2% performance improvements and up to a 2X speed-up without any extra computational cost [6].
  - **SingLoRA:** This is a new and innovative way to perform low-rank adaptation. It learns the weight update as a decomposition of a single low-rank matrix multiplied by its transpose, $W = W_0 + \frac{\alpha}{r}AA^\top$ [7]. This design inherently eliminates the inter-matrix scale conflicts that plagued earlier LoRA versions, ensuring stable feature learning and transformation-invariance [7]. SingLoRA

simplifies optimization by not requiring specialized tuning for learning rates. It achieves roughly a 50% reduction in parameters compared to standard LoRA and has shown to outperform it (e.g., +2% for Llama on MNLI) with much greater robustness to learning rate variations [7].

- **Decomposition of Learnable Parameters into Smaller Scalable Matrices:** Beyond LoRA, we'll explore other matrix factorization or decomposition techniques (like Singular Value Decomposition - SVD) to approximate large weight matrices with products of smaller, more manageable, and scalable matrices. This approach further reduces the total number of learnable parameters and computational complexity, making models even more suitable for mobile deployment.
- **Computational Limits of Low-Rank Adaptation:** We recognize that despite the parameter efficiency, the backward pass for gradient computation in Transformers can still be a quadratic bottleneck ($O(L^2)$) [8]. Our future work will investigate methods for Approximate LoRA Gradient Computation (ALoRAGC) to achieve sub-quadratic or even almost linear time complexity [8]. This requires strict normalization of inputs, pre-trained weights, and LoRA matrices, as "outliers" can significantly hinder both efficiency and performance [8]. We believe that effective normalization techniques, such as pre-activation layer normalization or outlier-removing attention activations, are crucial enablers for these speedups [8].
- **Efficient Attention Mechanisms:** We'll continue to research alternative attention mechanisms that are computationally less expensive than the standard self-attention (which has quadratic complexity). This includes approaches like linear attention, sparse attention (which restricts attention to a subset of tokens), or attention mechanisms with fixed patterns. All these aim to reduce the computational burden while maintaining sufficient representational power.
- **Hardware-Aware Design:** Our goal is to design Transformer architectures that are specifically optimized for the underlying mobile hardware. This means leveraging particular chip capabilities (e.g., specialized AI accelerators, memory hierarchies) for faster computation and lower power consumption. Often, this involves co-designing the model and the hardware for truly optimal performance.

These avenues offer exciting directions for pushing the boundaries of what's possible with deep learning on mobile devices, making advanced AI capabilities more accessible and widespread.

7 Conclusion

This internship project provided us with valuable insights into designing and optimizing Transformer models for mobile-use cases. By focusing on a lightweight architecture with a reduced number of layers, we aimed to directly address the critical challenges of computational and memory constraints on edge devices. Our work underscored the importance of balancing model performance with efficiency for practical mobile deployment. The detailed exploration of parameter-efficient fine-tuning techniques (like LoRA, LoRA+, and SingLoRA), alongside architectural insights into attention and feed-forward networks, and the integration of Vision Transformers, collectively highlight our ongoing commitment to making powerful AI models more accessible and efficient for a wide range of mobile applications. The future directions we've outlined leverage cutting-edge research to continue minimizing Transformer models while maximizing their impact on mobile platforms.

8 References

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems*, 30.

2. Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2021). LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685.*

3. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2021). An Image Is Worth 16x16 Words: Transformers for Image Recognition At Scale. *International Conference on Learning Representations (ICLR).*

4. Michel, P., Levy, O., & Neubig, G. (2019). Are Sixteen Heads Really Better than One?. *Advances in Neural Information Processing Systems*, 32.

5. Geva, M., Schuster, R., Berant, J., & Levy, O. (2020). Transformer Feed-Forward Layers Are Key-Value Memories. *arXiv preprint arXiv:2012.14913.*

6. Hayou, S., Ghosh, N., & Yu, B. (2024). LoRA+: Efficient Low Rank Adaptation of Large Models. *arXiv preprint arXiv:2402.12354.*

7. Bensaïd, D., Rotstein, N., Velich, R., Bensaïd, D., & Kimmel, R. (2025). SingLoRA: Low Rank Adaptation Using a Single Matrix. *arXiv preprint arXiv:2507.05566.*

8. Hu, J. Y., Su, M., Kuo, E. J., Song, Z., & Liu, H. (2024). Computational Limits of Low-Rank Adaptation (LoRA) Fine-Tuning for Transformer Models. *Published as a conference paper at ICLR 2025.*

9. Huan, S., Wang, Z., Wang, X., Wu, L., Yang, X., Huang, H., & Dai, G. E. (2023). A lightweight hybrid vision transformer network for radar-based human activity recognition. *Scientific Reports*, 13(1), 1-13.