Efficient Parameter Optimization for Compact Language Models

Youngjun Yu
Pohang University of Science and Technology (POSTECH)
Department of Computer Science and Engineering
colin31472@postech.ac.kr

Abstract

This project aims to develop a high-performance, compact language model optimized for mobile and low-resource devices by reducing the parameters to under a billion without sacrificing effectiveness. Inspired by MobileLLM (Liu et al., 2024), the research explores some techniques beneficial to compact language models, such as Double Up-Projection, SwiGLU activation, GQA (Grouped-Query Attention), Embedding Sharing and Layer Sharing. In addition, techniques such as Enhanced Residual Connection and Deep Thinking strategy will be evaluated to further enhance performance. The ultimate goal is to enable real-time applications on mobile devices by eliminating cloud dependency and latency, thus improving accessibility while minimizing energy consumption and environmental impact.

1 Introduction

Traditional large language models (LLMs) such as GPT-3, with 175 billion parameters, and GPT-4, with over 1 trillion parameters, require substantial computational resources and energy for training and operation. This leads to environmental concerns, such as significant carbon dioxide emissions. Consequently, developing methods to train and deploy LLMs with reduced computational costs has become a critical challenge in recent LLM research.

Several approaches aim to minimize computational resources during model training. Among them, the LoRA (Hu et al., 2021) method stands out for enabling efficient fine-tuning by adjusting only a small subset of parameters while maintaining performance. However, a key limitation of such methods is that they do not reduce the total number of parameters in the model itself, meaning that the initial training phase still demands substantial resources.

Efforts to reduce the number of parameters in language models have been actively explored. Models such as Cerebras-GPT (Dey et al., 2023), OPT (Zhang et al., 2022), GPT-Neo (Black et al., 2022), BLOOM (Scao et al., 2022), Pythia (Biderman et al., 2023), and RWKV (Peng et al., 2023), which contain fewer than a billion parameters, exemplify this trend. However, these models face a significant drawback in that their performance is substantially inferior compared to existing large language models. Therefore, this study aims to develop a lightweight high-performance language model that achieves parameter efficiency optimization while maintaining competitive performance relative to large-scale models.

Ultimately, this study aims to implement a model that efficiently utilizes parameters and can operate on low-resource devices such as mobile devices. Using fewer than one billion parameters, the goal is to design a model that achieves performance close to large-scale models while being capable of real-time operation on mobile devices.

2 Methods

2.1 Baseline Model

The goal of this study is to implement a high performance LLM that efficiently utilizes a small number of parameters through structural improvements. The GPT-2 (Radford et al., 2019) decoder-only model was used as the baseline and the new model was designed following the fundamental architecture of MobileLLM. To achieve model compression, the following techniques were sequentially integrated into the model.

2.2 Double Up-Projection

The FFN (Feed Forward Network) used in the traditional decoder block consists of three stages: up-projection, activation function, and down-projection. To enable the model to learn richer information, a method was applied where two

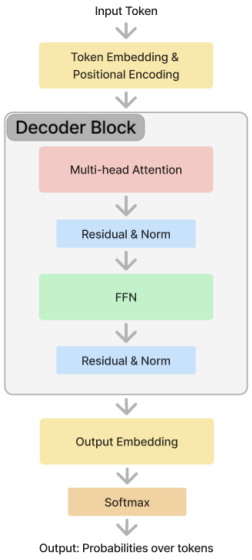

Figure 1: Baseline Model Architecture. The architecture follows the GPT-2 decoder-only design and integrates techniques from MobileLLM for efficient parameter utilization.

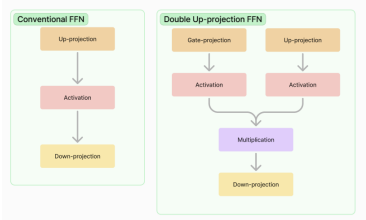

Figure 2: Double Up-Projection Architecture. The FFN is enhanced by adding two up-projection paths: gate-projection and up-projection, followed by an element-wise multiplication. This approach allows the model to capture richer and distinct features.

up-projections, namely gate-projection and up-projection, are used, followed by element-wise multiplication. Using two separate up-projection paths, each path is expected to learn distinct features, thereby improving the model's performance.

2.3 SwiGLU Activation Function

In the traditional FFN approach, the ReLU activation function is used. However, the SwiGLU activation function, known to perform best in smaller language models, was adopted instead. SwiGLU is an activation function that combines the Swish activation function with a Gated Linear Unit (GLU).

$$\text{SwiGLU}(x, W, V, b, c, \beta) = \text{Swish}_\beta(xW + b) \otimes (xV + c)$$

$$\text{Swish}_\beta(x) = x \cdot \sigma(\beta x)$$

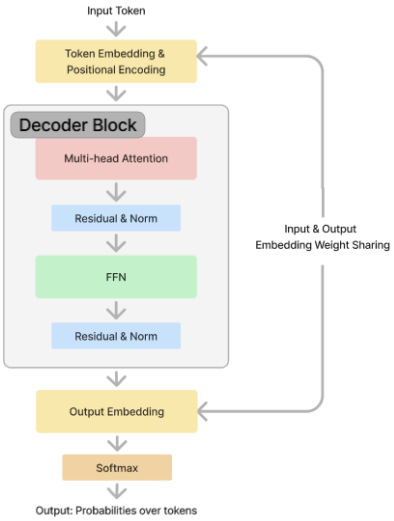

Figure 3: Illustration of Embedding Sharing. The input embedding layer's parameters are reused in the output embedding layer, reducing the overall parameter count and creating a more compact model.

2.4 Embedding Sharing

The parameters of the Embedding layer, which applies token embedding to the input, were reused as the parameters of the output Embedding layer. Using Embedding Sharing, the total number of parameters was reduced from 154.0M to 128.2M, saving approximately 25 million parameters. This technique contributes to building a more efficient and compact model architecture.

2.5 GQA (Grouped-Query Attention)

In the baseline model, Multi-head Attention was used with an equal number of key-value heads and query heads. However, in our model, GQA (Grouped-Query Attention) was applied to use parameters more efficiently. This approach keeps the number of queries the same while sharing the same key-value heads across multiple queries. Specifically, the model uses 3 key-value heads for 9 self-attention heads. To maintain the total number of parameters, we compensated for the parameter reduction from GQA by increasing the number of decoder blocks or expanding the embedding dimension.

2.6 Layer Sharing

The Layer Sharing technique was applied by repeating each decoder block twice with shared parameters. This approach improves performance while using the same number of parameters.

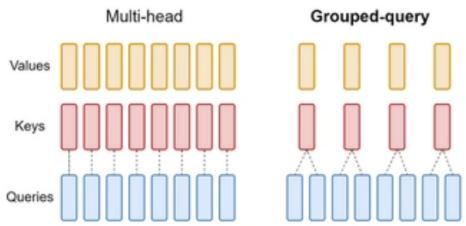

Figure 4: Illustration of Grouped-Query Attention (GQA). The reduction in parameters from GQA is balanced by either increasing the number of decoder blocks or expanding the embedding dimension.

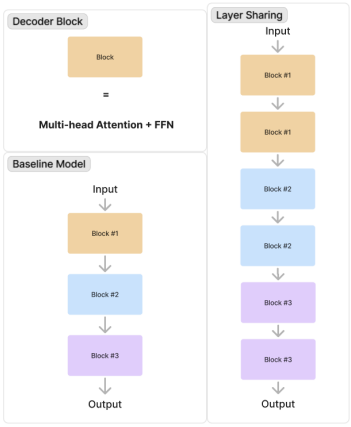

Figure 5: Illustration of Layer Sharing. Each decoder block is repeated twice, sharing the same parameters to enhance performance without increasing the total parameter count. Decoder blocks with the same color represent shared parameters.

2.7 Enhanced Residual Connection

The traditional Residual Connection (He et al., 2015) can be expressed in the form $x + F(x)$, where the input x is simply added to the output of the function $F(x)$. In contrast, the Enhanced Residual Connection used in our model adopts the form $\alpha \cdot x + (1 - \alpha) \cdot F(x)$. Here, α is a learnable parameter within the range [0, 1], allowing the model to assign weights dynamically to x and $F(x)$ based on their importance. To ensure that the learnable parameter α stays within the range [0, 1], a sigmoid function was applied. Specifically, α is defined as:

$$\alpha = \sigma(w)$$

$$\text{where } \sigma(w) = \frac{1}{1+e^{-w}}$$

$$x_{FFN} = \alpha_1 \cdot x_{input} + (1 - \alpha_1) \cdot F_{attention}(x_{input})$$

$$x_{output} = \alpha_2 \cdot x_{FFN} + (1 - \alpha_2) \cdot F_{FFN}(x_{FFN})$$

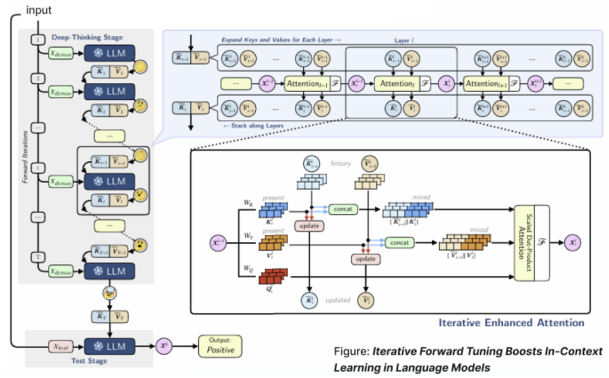

Figure 6: Illustration of the Deep Thinking process. Multiple iterative forward passes enable the model to refine and accumulate information, mimicking human-like iterative reasoning for better input understanding.

Each layer (decoder block) requires two learnable parameters, α_1 and α_2: one for the connection before and after the attention mechanism, and the other for the connection before and after the FFN (Feed Forward Network). With 30 layers used in the model, the total number of additional parameters is calculated as:

$$\begin{aligned}\text{Total } \alpha\text{'s} &= (\#\text{ of } \alpha \text{ per layer}) \cdot (\#\text{ of layers}) \\ &= 2 \cdot 30 \\ &= 60\end{aligned}$$

2.8 Deep Thinking

Deep Thinking (Yang et al., 2024), a method proposed to better handle demonstrations during inference in In-Context Learning (ICL), was employed. This approach mimics the iterative thought processes of humans, allowing the model to think more deeply about the input. Instead of a single forward pass, multiple iterative forward passes are performed, accumulating and refining information to enable the model to better understand and process the input. While previous studies applied Deep Thinking only to the demonstrations in ICL, our model extends this technique to the entire input.

The detailed implementation is as follows. The framework is divided into two stages: the deep thinking stage and the test stage. During the deep thinking stage, the model performs a total of T forward passes, iteratively updating the key-value matrix of the model. Using the output of the previous layer (decoder block) at the t-th forward pass, X_t^{l-1}, the current layer generates the query, key, and value matrices Q_t^l, K_t^l, and V_t^l, respectively. Here, l denotes the layer index, and t represents the forward pass index.

$$K_t^l = W_K X_t^{l-1},$$
$$V_t^l = W_V X_t^{l-1},$$
$$Q_t^l = W_Q X_t^{l-1}.$$

The computed values K_t^l and V_t^l are concatenated with the previously accumulated key and value matrices, $\tilde{K}_{t-1}^l$ and $\tilde{V}_{t-1}^l$, respectively. These concatenated matrices are then passed through the attention mechanism and the Feed Forward Network (FFN, $\mathcal{F}$) to generate the output of the current layer. The blue text indicates that it includes the history accumulated so far.

$$X_t^l = \mathcal{F}(\text{Attention}_l(\{\tilde{K}_{t-1}^l \| K_t^l\}, \{\tilde{V}_{t-1}^l \| V_t^l\}, Q_t^l))$$

The previously accumulated $\tilde{K}_{t-1}^l$ and $\tilde{V}_{t-1}^l$ are combined with the newly computed K_t^l and V_t^l in the current layer to form the updated accumulated matrices $\tilde{K}_t^l$ and $\tilde{V}_t^l$, which now include the information from the current layer. The update equations are as follows:

$$\tilde{K}_t^l = \text{update}(\tilde{K}_{t-1}^l, K_t^l) = \eta K_t^l + (1-\eta)\tilde{K}_{t-1}^l$$

$$\tilde{V}_t^l = \text{update}(\tilde{V}_{t-1}^l, V_t^l) = \eta V_t^l + (1-\eta)\tilde{V}_{t-1}^l$$

Here, η is a hyperparameter that determines the ratio between the contributions of the previously accumulated information and the current information. After T iterations of the deep thinking stage, the accumulated matrices $\tilde{K}_T^l$ and $\tilde{V}_T^l$ are used in the test stage to generate the model's output.

3 Experiments

3.1 Experimental Settings

The training and evaluation were conducted on four models: the **Baseline** model, the **Layer Sharing** model, which includes techniques such as Double Up-Projection, SwiGLU, Embedding Sharing, GQA, and Layer Sharing, the **Enhanced Residual** model, which adds Enhanced Residual Connection to the Layer Sharing model, and the **Deep Thinking** model, which further incorporates Deep Thinking into the Enhanced Residual model.

The models were trained on the Wikitext-103 dataset, and their performance was evaluated using perplexity on this dataset. For training, the initial learning rate was set to 2×10^{-3}, and weight decay was set to 0.1, using the Adam optimizer. The batch size was set to 4, and the experiments were conducted on a TitanXP GPU.

3.2 Model Details

The details of each model are as follows. The vertical axis lists the models, where LS, ER, and DT refer to the Layer Sharing, Enhanced Residual, and Deep Thinking models, respectively. These three models share the same model details.

Model	D_{model}	D_{hidden}	N_{head}	$N_{\text{kv\_head}}$	N_{layer}	$N_{\text{parameter}}$
Baseline	768	1536	12	-	12	134.0M
LS, ER, DT	576	1536	9	3	30	135.4M

Table 1: Details of the models. D_{model} is the embedding dimension, D_{hidden} is the hidden dimension, N_{head} is the number of attention heads, $N_{\text{kv\_head}}$ is the number of KV heads in GQA, N_{layer} is the number of layers, and $N_{\text{parameter}}$ is the total number of parameters. LS, ER, and DT refer to the Layer Sharing, Enhanced Residual, and Deep Thinking models, respectively.

In the Deep Thinking model, the number of deep thinking steps (T) was set to 3, and the value of η used for KV-memory updates was set to 0.01.

3.3 Main Results

3.3.1 Perplexity

The Perplexity of the four models on the Wikitext dataset is as follows. Perplexity represents the average uncertainty of the probability distribution predicted by the model, with lower values indicating better performance.

	Baseline	Layer Sharing	Enhanced Residual	Deep Thinking
Perplexity	31.02	28.95	28.85	**18.67**

Table 2: Perplexity of different models on the Wikitext dataset. Lower perplexity indicates better performance.

The performance of the Baseline model improved gradually as techniques such as Double Up-Projection, SwiGLU, Embedding Sharing, GQA, Layer Sharing, and Enhanced Residual were added. A significant performance improvement was observed when Deep Thinking was applied.

3.3.2 Comparing α values in Enhanced Residual

When Enhanced Residual is applied, the plot of alpha values for each layer is as follows.

α_1 represents the weight applied when connecting the input and output before and after the attention, while α_2 represents the weight applied when connecting the input and output before and after the FFN.

A large α value indicates a higher weight on the input x before applying the function $F(x)$, while

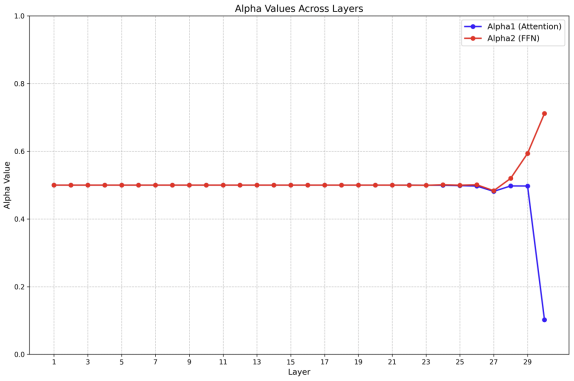

Figure 7: Alpha values for each layer.

a smaller α value indicates a higher weight on the output of the function $F(x)$.

According to the graph above, both α_1 and α_2 assign equal weights to x and $F(x)$ in layers 1 through 26. However, in layers 27 to 30, a divergence occurs. α_1 decreases, assigning more weight to the output after attention, while α_2 increases, assigning more weight to the input before the FFN.

4 Discussion

4.1 Interpretation for α_1

In the Transformer architecture, attention integrates and understands contextual information, while the Feed Forward Network (FFN) transforms the contextualized representation into a non-linear and feature-extractive form

In the later layers (27–30), just before generating the output, the model focuses more on the newly integrated contextual information from attention. This can be interpreted as a decrease in α_1, assigning greater weight to the output of attention mechanism.

4.2 Interpretation for α_2

In the later layers (27–30), the focus should be on refinement rather than drastically modifying representations. Since the early and middle layers (1–26) have already constructed complex syntactic meanings from token embeddings, the non-linear transformations of the FFN may become unnecessary in the later layers (27–30). Consequently, less weight is assigned to the values after the FFN, which can be interpreted as an increase in α_2.

5 Future Work

The Deep Thinking method used in the model significantly improves performance but increases inference time due to multiple Deep Thinking stages during inference. This presents a trade-off, considering the model is designed for use on mobile devices. Therefore, future research will aim to optimize inference time by dynamically adjusting the number of thinking stages based on the complexity of the input, rather than applying the same number of thinking stages to all inputs.

References

Stella Biderman, Hailey Schoelkopf, Joseph Levine, Eric Hallahan, Quentin Anthony, Mohammad Aflah Khan, Kyle McDonell, Edward Raff, Albert Villanova del Moral, and et al. 2023. Pythia: A suite for analyzing the performance of large language models across training. *arXiv*.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. Gpt-neox-20b: An open-source autoregressive language model. *arXiv*.

Nolan Dey, Gurpreet Gosal, Zhiming (Charles)Chen, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, and Joel Hestness. 2023. Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster. *arXiv*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *arXiv*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv*.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. 2024. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. *arXiv*. Accepted at ICML 2024.

Bo Peng, Zhenghua Xu, Chuhui Zhang, Jinghui Hu, Zhaoyi Zhan, Xiubo Geng, Jian Wu, and Furu Wei. 2023. Rwkv: Reinventing rnns for the transformer era. *arXiv*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, and et al. 2022. Bloom: A 176b parameter open-access multilingual language model. *arXiv*.

Jiaxi Yang, Zihan Wang, Hengrui Zhang, Dianwen Zhang, Xiangyu Zhang, and Xuanjing Huang. 2024. Iterative forward tuning boosts in-context learning in language models. *arXiv*. Accepted at ACL 2024.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open pre-trained transformer language models. *arXiv*.