TOWARDS SMALLER, FASTER DECODER-ONLY TRANSFORMERS: ARCHITECTURAL VARIANTS AND THEIR IMPLICATIONS

Sathya Krishnan Suresh
Nanyang Technological University
sathyakr001@e.ntu.edu.sg

Shunmugapriya P
Puducherry
pshunmugapriya@gmail.com

ABSTRACT

In recent times, the research on Large Language Models (LLMs) has grown exponentially, predominantly focusing on models underpinned by the transformer architecture, and further developed through the decoder-only models such as Large Language Models (LLM). Contemporary efforts in this field primarily aim to enhance model capabilities by scaling up both the architecture and data volumes utilized during training. However, there has been little exploration into reducing model sizes while maintaining their effectiveness. In this study, we introduce three modifications to the decoder-only transformer architecture—namely ParallelGPT (*pgpt*), LinearGPT (*lgpt*), and ConvGPT (*cgpt*). These variants demonstrate comparable performances to the conventional architecture, with *lgpt* outperforming it in **4 out of 7 benchmarks** with **less than half the parameters**. We open-source the model weights and the complete codebase for these implementations for further research.

1 Introduction

Since the debut of ChatGPT, there has been a notable increase in research on Large Language Models (LLMs) across a broad range of disciplines, made possible by the accessibility of this technology to a diverse user base. This fastly growing field has largely pursued two distinct paths: one aims at either scaling the model size or the training dataset (or both) to enhance performance, while the other concentrates on refining smaller models (ranging from 1B to 7B parameters) with high-quality data. Despite these advances, investigations into the structural modifications of the transformer architecture itself have been relatively overlooked. Recent studies challenge the necessity of perpetually increasing model sizes by demonstrating that the deeper layers of LLMs may have minimal influence on predictive outcomes [1].

In this work, we explore modifications to the decoder-only transformer architecture to address current challenges in the scalability and practical application of Large Language Models (LLMs). Recognizing the significant impact of model size on the computational overhead of training and inference, we introduce three variants—ParallelGPT (*pgpt*), LinearGPT (*lgpt*), and ConvGPT (*cgpt*)—each designed to reduce parameter count while maintaining, or potentially enhancing, model performance.

The models presented in this paper range from 30M to 80 parameters. Our decision to train models of these sizes was influenced by GPT-2 [2], which marked the beginning of the GPT era. We aimed to introduce architectural variations within the vicinity of GPT-2's size to explore similar performance characteristics. Rather than striving for state-of-the-art results, our goal was to develop models with comparable parameters but reduced size, achieving performance similar to the traditional GPT architecture.

We pre-trained each architectural variant on the 10 billion token subset of the fineweb-edu [3] dataset. The dataset was chosen to match the scale of the models we wanted to train. By training on a reasonably extensive dataset, we aimed to ensure that the models could be effectively evaluated on various benchmarks, thereby comparing the performance of different architectures.

The remainder of this paper is organized as follows. The related works are discussed in Section 2. The modifications made to the base architecture and the reasons for the modifications are presented in Section 3. Experimental setup and

GitHub Repository: gpt-variations

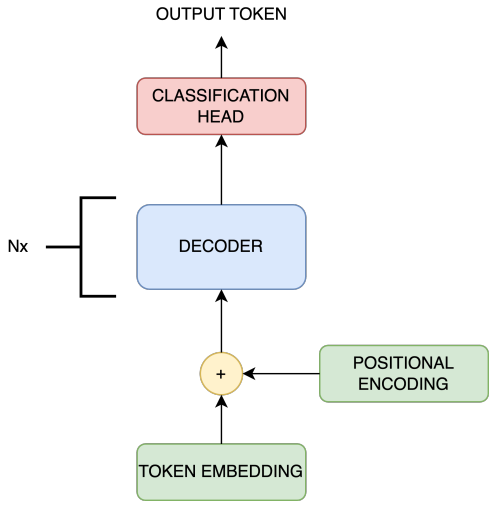

Figure 1: Traditional GPT architecture

the results are presented in Section 4. Section 5 discusses the limitations of the research and finally Section 6 contains the conclusion of our work.

2 Related Work

The evolution of Large Language Models (LLMs) has been heavily influenced by advances in transformer architecture, first introduced by Vaswani et al. [4] and later adapted into decoder-only models such as GPT [5]. These models typically focus on scaling up, using more complex architectures and larger datasets to improve performance. However, there is growing interest in designing models that maintain strong performance while being smaller and more efficient, addressing the need for models that are simpler to train and deploy.

Our work introduces three new variants of the decoder-only transformer architecture aimed at improving training and inferencing efficiency: ParallelGPT (*pgpt*), LinearGPT (*lgpt*), and ConvGPT (*cgpt*). These models achieve similar performance to traditional transformer architectures in text generation tasks but with fewer parameters and faster training times. We have open-sourced the model weights and codebase to support further research.

Recent research also explores efficiency-focused architectures. For example, the Funnel-Transformer [6] reduces computational costs by compressing the sequence of hidden states through a pooling mechanism, allowing the model to expand in depth or width without increasing the overall cost.

The MobiLlama model [7] is another example, optimized for use in resource-constrained settings. It uses a parameter-sharing strategy within transformer blocks to minimize both pre-training and deployment costs, aligning with the goal of creating smaller, more efficient models.

Innovative attention mechanisms have also contributed to more efficient LLMs. Grouped Query Attention [8] reduces the complexity of attention calculations by grouping inputs, effectively lowering the computation from quadratic to linear, which helps in handling longer sequences. Multi-Query Attention [9] further improves efficiency by allowing multiple queries within a single attention head, enhancing the model's ability to gather and process diverse information.

These advances reflect a clear trend toward making LLMs not just more powerful but also more adaptable to practical constraints like size and computational limits. Our work builds on this trend by exploring novel architectural modifications that offer a balance between performance and efficiency.

3 Architectural modifications

In this section, we introduce three novel architectures derived from the traditional GPT architecture to address various limitations in training and inference. These architectures are designed to enable faster training and inference. The three proposed architectures are ParallelGPT (*pgpt*), LinearGPT (*lgpt*), and ConvGPT (*cgpt*).

3.1 ParallelGPT

In a traditional GPT architecture, the N decoder blocks of the same dimensionality are stacked on top of each other. The drawback of such an architecture is that as *N* increases, the deeper blocks have little information to work with. Also recent research [1] point out the inefficiency of the deeper layers thereby questioning the need for increasing N. To overcome these drawbacks, we propose splitting the N decoder blocks into *P* parallel blocks such that each parallel path P_i consists of N/P decoder blocks. Each P_i has a preceding dense layer PD_i to process the incoming token embeddings. The reason for introducing a dense layer before each parallel block is to make sure that each block learns something different. A vector of learnable weights $W \in \mathbb{R}^P$ is used to combine the outputs of each of the parallel blocks. The following set of equations give the mathematical representation of the architecture.

$$h_i = PD_i(x), \quad \text{for } i = 1, 2, \ldots, P \tag{1}$$

$$y_i = f_i(h_i), \quad \text{for } i = 1, 2, \ldots, P \tag{2}$$

$$\alpha_i = \frac{\exp(w_i)}{\sum_{j=1}^{P} \exp(w_j)}, \quad \text{for } i = 1, 2, \ldots, P \tag{3}$$

$$y = \sum_{i=1}^{P} \alpha_i \cdot y_i \tag{4}$$

We designed this architecture so that we can have the following benefits:

- Faster training, as each parallel block P_i can be assigned to a separate compute node, allowing them to be trained in parallel.
- During inference, we have the flexibility to drop 0 to $P - 1$ number of blocks based on the required inference speed, trading off some output quality for faster performance.

A point to note here is that we have used a simple weighting mechanism for combining the outputs but other methods like mixture of experts, gating functions can also be used, which we leave for future research.

3.2 LinearGPT

As discussed in [1], the earlier layers contribute more to the generation capabilities of an LLM than the deeper layers. This questions the need for each decoder block having the same dimension as this results in the model having much more parameters than that is actually needed for the desired performance. The extra parameters of the model might also result in the model overfitting to the training set which LLMs are known to do.

To address these issues, we propose an architecture, *lgpt*, in which the dimension of the decoder blocks keeps reducing by half as the embedding of a token passes through the architecture. This concept is inspired from the continuous downsampling of images in CNNs for a task like image classification. To make sure that the dimension of the embedding of a token does not get too small by the time it reaches the classification head, we only reduce the dimension after every n number of decoder blocks. This is done by introducing a dense or a linear layer after every n decoder blocks. The following equations 5, 6 and 7 give a mathematical representation for the architecture discussed. Here D_i represents the i^{th} decoder block in an N number of stacked decoder blocks and Eq. 7 gives us the dimension d_m of a vector before it goes through a decoder block.

$$\mathbf{h}_i = \mathrm{D}_i(\mathbf{h}_{i-1}), \quad i = 1, 2, \ldots, N \tag{5}$$

$$\mathbf{h}_i = \begin{cases} \mathrm{L}_j(\mathbf{h}_{i-1}), & \text{if } i = nj \\ \mathbf{h}_{i-1}, & \text{otherwise} \end{cases} \tag{6}$$

$$d_m = \frac{d_0}{2^m}, m = \left\lfloor \frac{N}{n} \right\rfloor \tag{7}$$

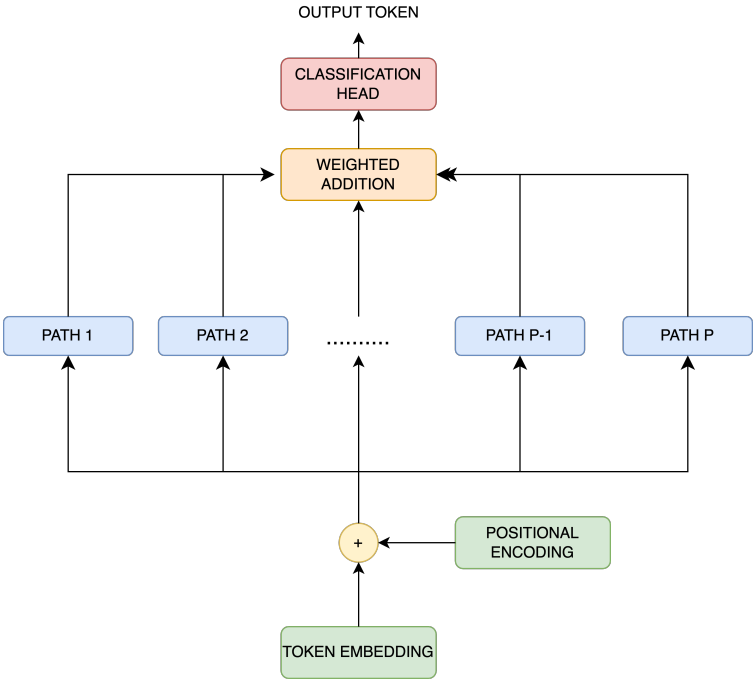

Figure 2: ParallelGPT

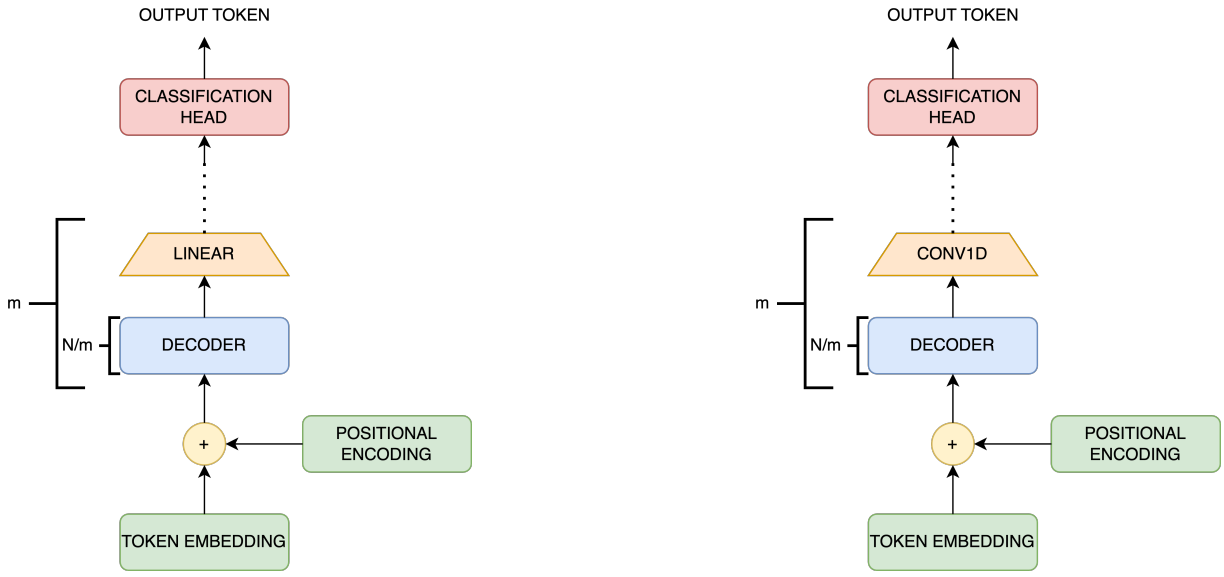

Figure 3: LinearGPT and ConvGPT

The *lgpt* architecture can reduce the number of parameters in the model in terms of millions or billions. The reduction potential of this architecture is shown in the Appendix A. In the experiments we did, the number of parameters in the traditional *gpt* architecture was **77.2M** parameters while the *lgpt* architecture had only **36.4M** parameters and it still performed competitively with *gpt*, while outperforming it COPA and ARC Easy [10] as shown in Table 4.

3.3 ConvGPT

A one-dimensional convolution with kernel size and stride as 1 is a very similar operation when compared with the linear operation as both of them have the same number of parameters to train and both of them carry out a matrix

multiplication of similar dimensions but at a fundamental level both are different because of the positional context, spatial invariance and weight sharing properties possessed by the convolutional operations. In order to find whether using one dimensional convolution for compressing the dimensions, has a performance difference compared to the traditional *gpt* and the previous architectures introduced, we introduce a third new architecture *cgpt* which replaces the linear compression layers in *lgpt* with a one dimensional convolution layer. *cgpt* offers the same reduction in number of parameters as *pgpt* and as can be seen from Table 4 it is competitive with the traditional *gpt* architecture.

4 Experimental Setup and Results

This section discusses the specific parameters we used for the experiments, the training parameters used and the results of each architecture on various benchmarks.

4.1 Model Configuration

For each of the architecture the common hyperparameters are set as illustrated in Table 1. For the *pgpt* architecture, we set the number of parallel blocks P in this experiment to 2 with each parallel path P_i having 4 decoder blocks. For the *lgpt* and *cgpt* architectures we set n (number of blocks after which a linear or convolutional layer is introduced) as 2 resulting in the final embedding dimension to be 32. The number of parameters and the memory requirements for each of the models are shown in Table 2, where *pgpt-1*, refers to a *pgpt* model in which the one parallel block is dropped. Note that during training, none of the parallel blocks are dropped. During inferencing and saving the model, 0 to $P-1$ blocks can be dropped to save time and memory. We have algorithmically illustrated the reduction potential of the *pgpt* architecture in Appendix B

Parameter	Value
`context_size`	1024
`vocab_size`	50304
`n_layer`	8
`n_head`	8
`embedding_dimension`	512
`dropout`	0.0

Table 1: Common Configuration Parameters

Model	# Params	Memory
`gpt`	77.2M	294.53MB
`pgpt`	77.7M	296.53MB
`lgpt`	36.4M	138.95MB
`cgpt`	36.4M	138.95MB
`pgpt-1`	64.8M	247.52MB

Table 2: Model Hardware Requirements

4.2 Training

We train all the models in the same training setup which we have listed in Table 3. We use a cosine decay for the learning rate. We train each model for 5000 steps which is about 25% percent of the entire training set and the training was stopped here as due to compute constraints. We train each model on `4xA5000` GPUs, where the *gpt* and *pgpt* architectures took about 2 hours to train while *lgpt* and *cgpt* architectures took about 1 hour to train.

4.3 Results

In this subsection, we present the loss curves and the evaluation results of our proposed model variants, comparing their performance on the following benchmarks

- **HellaSwag:** [11] Tests the model's ability to understand commonsense reasoning and continuation of textual descriptions, evaluating how well the model predicts the next plausible sentence in narrative-like sequences.
- **WinoGrande:** [12] Measures the model's performance on pronoun resolution tasks, which challenges the model's understanding of context and disambiguation of pronouns in complex sentences.

Hyperparameter	Value
Batch size	16
Gradient accum. steps	8
Min learning rate	6e-5
Max learning rate	6e-4
Max steps	19000
Warmup steps	700
Weight decay	1e-1

Table 3: Training parameters

- **CommonSenseQA: [13]** Focuses on evaluating the model's general commonsense knowledge and reasoning abilities, particularly in answering multiple-choice questions based on everyday scenarios.
- **ANLI (Adversarial Natural Language Inference): [14]** Tests the model's robustness and understanding of natural language inference under adversarial conditions, requiring the model to distinguish between entailment, contradiction, and neutral statements.
- **PIQA (Physical Interaction Question Answering): [15]** Evaluates the model's grasp of physical commonsense reasoning, specifically how well it can predict plausible actions or outcomes related to everyday physical interactions.
- **COPA (Choice of Plausible Alternatives): [16]** Assesses the model's ability to infer causality and select the more plausible of two alternative outcomes given a premise, testing causal reasoning skills.
- **ARC Easy: [10]** Tests scientific knowledge and reasoning in a simplified setting, focusing on how well the model can answer multiple-choice questions related to elementary and middle school science topics.

Benchmark	*gpt*	*pgpt*	*lgpt*	*cgpt*	*pgpt-1*
HellaSwag	**0.2625**	0.2604	0.2517	0.2492	0.2527
WinoGrande	**0.5036**	0.4925	0.4870	0.4751	0.4933
CommonSenseQA	0.1376	**0.1605**	0.1458	0.1540	0.1572
ANLI	0.3300	**0.3350**	0.3290	0.3240	0.3340
PIQA	0.5071	0.5131	0.5185	**0.5256**	0.5125
COPA	0.5300	0.5200	0.5500	0.5000	**0.5600**
ARC Easy	0.2333	0.2439	**0.2491**	0.2298	0.2509

Table 4: Benchmark Results for Different Models

Table 4 shows the performance of the various proposed architectures on a set of standard benchmarks. Note that the aim of this work is not to achieve state-of-the-art language models since the scale of the models and data used in this work is very small compared to the state-of-the-art models. The goal of this research is to produce architectural variants that are smaller in size with similar model parameters and have a performance comparable to the traditional *gpt* architecture in the same training and evaluation environments.

Comparison with the traditional architecture: The results from the table indicate that the *lgpt*, *cgpt* and *pgpt-1* architectures perform competitively with the traditional *gpt* architecture. The *pgpt-1* outperforms *gpt* on **5 out of the 7 benchmarks** while *lgpt* and *cgpt* outperform *gpt* on **4 out of the 7 benchmarks** and **2 out of the 7 benchmarks** respectively. Even on benchmarks that *gpt* performs the best, the other models perform competitively with *gpt*. We hypothesize that the possible reason for the competitive performance of the architectural variants is that the reduced number of parameters acts as a regularization and reduces the likelihood of the model overfitting to the training data which is usually the case in the traditional decoder-only models which can be prompted to output its training data. The results further indicate that even with reduced number of parameters the architectural variants are competitive thereby asking the following questions which we believe future research will answer,

- Which components of the decoder are most critical for generative capabilities?
- Is it necessary to scale uniformly across all decoder blocks, or can varied dimensional scaling offer similar or improved results?
- Should we test different architecture designs before scaling the models?

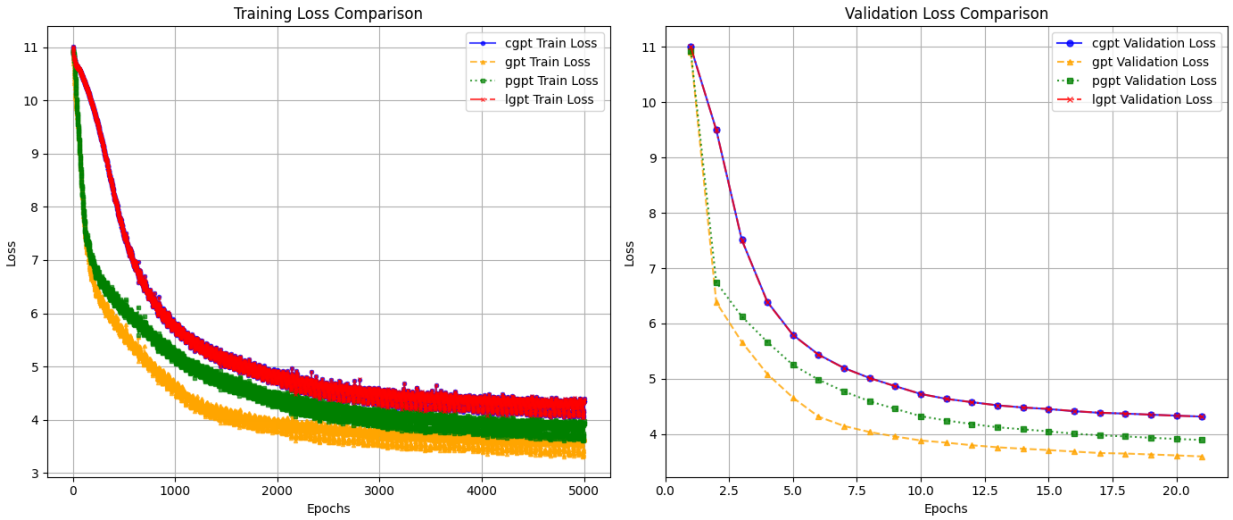

Figure 4: Loss comparison b/w the 4 models

Comparison b/w *pgpt* and *pgpt-1*: From the results it can be seen that *pgpt-1* with just four decoder blocks is able to perform as well as the *pgpt* model with eight decoder blocks. To further investigate if the network had learned to ignore the signal for one of the parallel paths and just used the signal from the other parallel path, we checked the final trained weight vector $W \in \mathbb{R}^2$. The weights of the parallel paths P_1 and P_2 were `0.5964` and `0.4036` respectively, which shows that the model used the signal from both the paths during training and the performance of the architecture remained robust during inference even when P_2 was dropped. The results raise the following questions for future research,

- Does each parallel path model different part of the knowledge and if so can an efficient mechanism be designed process different sequences only with the required parallel paths?
- Can the *pgpt* architecture be leveraged to introduce domain knowledge more effectively compared to post-pretraining methods like LoRA [17]? Specifically, can pgpt's unique parallel paths enable the direct integration of domain-specific knowledge during the pretraining stage by assigning dedicated blocks to learn varied domain aspects, or through the addition of specialized blocks post-pretraining, thus simplifying the process and enhancing model adaptability?

5 Conclusion

In this work, we presented three architectural variants of the traditional GPT architecture, aiming to investigate whether models with fewer parameters can achieve performance levels comparable to the original GPT. Our results across various benchmarks indicate that this is a promising and underexplored area of research. Through our findings, we raise critical questions and emphasize the need for further exploration. We believe that advancing this line of research is crucial, as AI can only reach its full potential to benefit humanity when it becomes widely accessible—a goal hindered by the high training and inference costs of large models today.

6 Limitations

While our proposed architectures demonstrate promising results, several limitations should be noted:

- **Lack of Comparative Analysis with SOTA Models:** The architectural variants have not been directly compared to state-of-the-art (SOTA) models due to resource constraints. While our research focuses on architectures of GPT-2 scale, direct comparisons with SOTA models could provide critical insights into the performance and efficiency of the proposed variants.

- **Computational Resource Constraints:** Due to computational and scale limitations, the training of our models was stopped after what we determined to be a reasonable number of steps. Further experiments with extended training durations on larger compute setups are necessary to explore the limits of these architectures.
- **Scalability and Applicability:** The scalability of the proposed architectures to larger model sizes and datasets remains untested. While effective at smaller scales, the performance and efficiency of *pgpt*, *lgpt*, and *cgpt* in large-scale, real-world applications need further investigation. Future work should explore how these models adapt when scaled up or deployed in complex scenarios.
- **Impact of Simplified Weighting Mechanisms:** The use of simple weighting for combining outputs from parallel paths in *pgpt* may not fully exploit the architecture's potential. Future research could explore more sophisticated methods, such as gating mechanisms or mixture of experts, to enhance the model's performance and adaptability.

Despite these limitations, our work provides valuable insights into architectural innovations in transformer models and opens new avenues for future research.

References

[1] Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*, 2024.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):1–9, 2019.

[3] HuggingFace. Fineweb-edu dataset. https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu, 2024. A dataset of educational web pages curated from the FineWeb corpus, designed for educational and academic applications.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[5] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[6] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc Le. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. In *Advances in Neural Information Processing Systems*, 2020.

[7] Omkar Thawakar, Ashmal Vayani, Salman Khan, Hisham Cholakkal, Rao Muhammad Anwer, Michael Felsberg, Timothy Baldwin, Eric P. Xing, and Fahad Shahbaz Khan. Mobillama: Towards accurate and lightweight fully transparent gpt, 2024.

[8] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023.

[9] Noam Shazeer. Fast transformer decoding: One write-head is all you need. 2019. cite arxiv:1911.02150.

[10] Peter Clark, James Cowhey, Oren Etzioni, Tushar Khot, Bhavana Dalvi Mishra, Clare Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[11] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800. Association for Computational Linguistics, 2019.

[12] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[13] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4149–4158. Association for Computational Linguistics, 2019.

[14] Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. Adversarial nli: A new benchmark for natural language understanding. *arXiv preprint arXiv:1910.14599*, 2019.

[15] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[16] Andrew Gordon, Zornitsa Kozareva, and Melissa Roemmele. Semeval-2012 task 7: Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics*, pages 394–398, Montréal, Canada, 7-8 jun 2012. Association for Computational Linguistics.

[17] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

[18] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio Cesar Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Conti Kauffmann, Gustavo Henrique de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sebastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. Textbooks are all you need, 2023.

[19] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *arXiv preprint arXiv:2106.04554*, 2021.

[20] Wenfeng Zheng, Gu Gong, Jiawei Tian, Siyu Lu, Ruiyang Wang, Zhengtong Yin, Xiaolu Li, and Lirong Yin. Design of a modified transformer architecture based on relative position coding. *International Journal of Computational Intelligence Systems*, 16(1):1–17, 2023.

[21] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11106–11115. AAAI, 2021.

[22] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

[23] Li Shen and Yangzhu Wang. Tcct: Tightly-coupled convolutional transformer on time series forecasting. *Neurocomputing*, 480:131–145, 2022.

[24] Andrej Karpathy. nanogpt: Minimal gpt implementation with pytorch. https://github.com/karpathy/nanoGPT, 2023.

A Reduction Potential of the *lgpt* Architecture

In this section, we illustrate the reduction potential possessed by the *lgpt* architecture. The python code given below illustrates the algorithm we used for calculating the number of parameters in the traditional *gpt* architecture and the *lgpt* architecture.

```python
def calculate_params_and_reduction(config, layer_red=2):
    # GPT Parameters Calculation
    embedding_params = config.vocab_size * config.n_embd
    block_params = config.n_layer * (12 * config.n_embd**2 + (config.n_embd if
        config.bias else 0))
    classification_head_params = config.n_embd * config.vocab_size
    gpt_params = embedding_params + block_params + classification_head_params

    # LGPT Parameters Calculation
    total_lgpt_params = embedding_params
    current_embd = config.n_embd

    for i in range(config.n_layer):
        linear_layer_params = 0
        if i % layer_red == 0 and i != 0:
            current_embd = max(1, current_embd // 2)
            linear_layer_params = current_embd * 2 * current_embd
        block_params = 12 * current_embd**2 + (current_embd if config.bias else 0)
        total_lgpt_params += block_params + linear_layer_params

    classification_head_params = current_embd * config.vocab_size
    lgpt_params = total_lgpt_params + classification_head_params

    # Reduction Calculation
    reduction_frac = (gpt_params - lgpt_params) / gpt_params

    return gpt_params, lgpt_params, reduction_frac
```

Listing 1: Calculation of Parameters and Reduction for GPT and LGPT Models

We count the parameters in both the *gpt* and the *lgpt* model the parameters as following: the vocab size is 50304, context length is 1024, embedding dimension is 1600, number of layers is 48 and the number of heads is 16. In the *lgpt* architecture we reduce the dimension after every 12 blocks so that the embedding dimension does not become too small. The following results demonstrate the effectiveness of *lgpt* in reducing the overall model size:

- **gpt Parameters:** 1,635,532,800
- **lgpt Parameters:** 581,827,200
- **Reduction in Size:** 64.43%

The results show that *lgpt* architecture can reduce the number of parameters in the model by **more than a billion parameters** for the same configuration parameters. We leave the comparison of models of this scale for future research due to compute constraints.

B Reduction Potential of the *pgpt* Architecture

In this section, we illustrate the reduction potential possessed by the *pgpt* architecture. The python code given below illustrates the algorithm we used for calculating the number of parameters in the traditional *gpt* architecture and the *pgpt* architecture.

```

def calculate_params_and_reduction_pgpt(config, m_parallel_paths=8, m_drop=4):
    # GPT Parameters Calculation
    embedding_params = config.vocab_size * config.n_embd
    block_params = config.n_layer * (12 * config.n_embd**2 + (config.n_embd if
        config.bias else 0))
    classification_head_params = config.n_embd * config.vocab_size
    gpt_params = embedding_params + block_params + classification_head_params

    # PGPT Parameters Calculation
    total_pgpt_params = embedding_params
    params_per_layer = 12*config.n_embd*2 + (config.n_embd if config.bias else 0)
    block_params = config.n_layer * params_per_layer
    drop_block_params = m_drop*(config.n_layer//m_parallel_paths) *
        params_per_layer
    block_params -= drop_block_params
    projection_params = (m_parallel_paths-m_drop)*(config.n_embd*config.n_embd)
    classification_head_params = config.n_embd*config.vocab_size
    total_pgpt_params +=  block_params + projection_params +
        classification_head_params

    # Reduction Calculation
    reduction_frac = (gpt_params - total_pgpt_params) / gpt_params

    return gpt_params, total_pgpt_params, reduction_frac
```

Listing 2: Calculation of Parameters and Reduction for GPT and PGPT Models

We count the parameters in both the *gpt* and the *pgpt* model the parameters as following: the vocab size is 50304, context length is 1024, embedding dimension is 1600, number of layers is 48 and the number of heads is 16. In the *pgpt* architecture we introduce 8 parallel paths and calculate the parameters during inference assuming four parallel paths are dropped.The following results demonstrate the effectiveness of *pgpt* in reducing the overall model size:

- **gpt Parameters:** 1,635,532,800
- **pgpt Parameters:** 172,134,404
- **Reduction in Size:** 89.48%

The results show that *pgpt* architecture can reduce the number of parameters in the model by **almost a billion and a half parameters** for the same configuration parameters. We leave the comparison of models of this scale for future research due to compute constraints.