Rethinking Optimization and Architecture for Tiny Language Models

Yehui Tang [1] **Kai Han** [1] **Fangcheng Liu** [1] **Yunsheng Ni** [1] **Yuchuan Tian** [2] **Zheyuan Bai** [1] **Yi-Qi Hu** [3] **Sichao Liu** [3] **Shangling Jui** [4] **Yunhe Wang** [1]

Abstract

The power of large language models (LLMs) has been demonstrated through numerous data and computing resources. However, the application of language models on mobile devices is facing huge challenge on the computation and memory costs, that is, tiny language models with high performance are urgently required. Limited by the highly complex training process, there are many details for optimizing language models that are seldom studied carefully. In this study, based on a tiny language model with 1B parameters, we carefully design a series of empirical study to analyze the effect of each component. Three perspectives are mainly discussed, *i.e.*, neural architecture, parameter initialization, and optimization strategy. Several design formulas are empirically proved especially effective for tiny language models, including tokenizer compression, architecture tweaking, parameter inheritance and multiple-round training. Then we train PanGu-π-1B Pro and PanGu-π-1.5B Pro on 1.6T multilingual corpora, following the established formulas. Experimental results demonstrate the improved optimization and architecture yield a notable average improvement of 8.87 on benchmark evaluation sets for PanGu-π-1B Pro. Besides, PanGu-π-1.5B Pro surpasses a range of SOTA models with larger model sizes, validating its superior performance. The code is available[1].

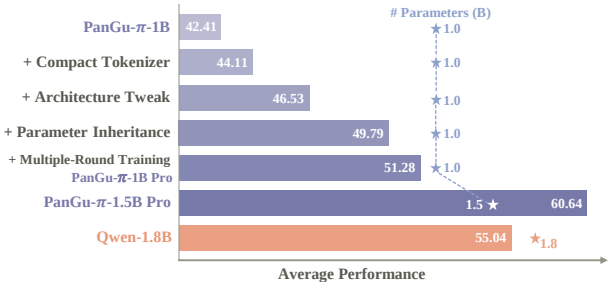


Figure 1. PanGu-π Pro with improved architecture and optimization methods. PanGu-π-1B (Wang et al., 2023) directly use the developing strategies of LLMs while PanGu-π-1B Pro achieves an average performance improvement of 8.87 with our methodology. It is worth mentioning that PanGu-π-1.5B Pro outperforms Qwen-1.8B (Bai et al., 2023) with 16.67% fewer parameters.

1. Introduction

Large language models (LLMs), trained on extensive corpora, have demonstrated impressive performance across diverse natural language tasks. The release of ChatGPT, with its robust generalization capabilities, has captured global attention and holds the potential to revolutionize the interaction between humans and computers.

In addition to the GPT-series models (Radford et al., 2018; Brown et al., 2020; Achiam et al., 2023), various large language models have emerged. PaLM (Chowdhery et al., 2023) trains a model with an impressive 540B parameters across 6144 TPU v4 chips. LLaMA (Touvron et al., 2023) releases a series of foundational language models, ranging from 7B to 70B parameters. Both the model architecture and trained weights are open-source, fostering collaboration within the AI community. Most of the following large models leverage similar architectures and training methodologies. For instance, Baichuan teams (Yang et al., 2023) train 7B and 13B parameter models on a 2.6T token dataset encompassing both Chinese and English corpora. Qwen (Bai et al., 2023), Yi (Yi, 2023), and Skywork (Wei et al., 2023) pursue similar paths, training models with 2.4T, 3T, and 3.2T tokens, respectively. Primarily attributed to the increasing accumulation of cleaned data, the performance of LLMs improves rapidly.

While numerous studies have successfully trained various high-performance language models (Ren et al., 2023; Zeng et al., 2022), the methodologies employed in training such

[1]Huawei Noah's Ark Lab [2]Peking University [3]Consumer Business Group, Huawei [4]Huawei Kirin Solution. Correspondence to: Yunhe Wang <yunhe.wang@huawei.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[1]https://github.com/YuchuanTian/RethinkTinyLM

models remain insufficiently analyzed. On one hand, a substantial body of work concentrates on collecting and cleaning data, with less emphasis on researching effective training strategies. On the other hand, the training of large models demands an exceedingly high computational resource investment, making it impractical to explore a wide range of optimization strategies. As a result, recent works often adopt similar training recipes when constructing LLMs (Touvron et al., 2023; Yi, 2023; Bai et al., 2023; Wei et al., 2023).

Moreover, the implementation of these large models demands prohibitively high memory and computational resources, constraining their practical applicability in various scenarios. For example, the GPT-3 with 175B parameters necessitates approximately 700GB of memory when stored with FP32 datatype. Although the 7B parameter models are relatively more efficient, their resource requirements still render them impractical for deployment on edge devices, such as mobile phones.

In this paper, we systematically rethink the methodology for constructing a tiny language model, including neural architecture, parameter initialization, and optimization strategy:

- Neural architecture: Adopting the tokenizer directly from larger models introduces redundant parameters, resulting in increased computational overhead. Streamlining the tokenizer by removing low-frequency vocabularies enhances the model's representational efficiency. Moreover, we observe that the configuration of the model's architecture (depth, width, and expanding rate in FFN) has a significant impact on the final performance. Depth is the primary factor for tiny language models, and deeper models usually achieve high performance at the expense of lower inference speed.

- Parameter initialization: Inheriting parameters from the large model proves effective in boosting performance and expediting convergence. The identification of crucial parameters is imperative in this context. We have observed that layers situated near the beginning and end of the model often carry more significance than the intermediate layers. Furthermore, within each layer, the adoption of data-driven learnable criteria has demonstrated greater efficacy compared to heuristic methods.

- Model optimization: In comparison to larger models, tiny models face more severe data forgetting issues, and multiple-round training proves beneficial for memory enhancement. We propose a straightforward sample selection strategy to mitigate the training cost associated with multiple-round training. Besides, we also delve into the relationship between batch size and learning rate specifically for tiny models.

Drawing from the aforementioned insights, we develop PanGu-π-1B Pro and PanGu-π-1.5B Pro with enhanced architecture and optimization methods. From the developing strategies of LLMs, we gradually add four core components to improve performance (see Figure 1). The models are evaluated on various benchmarks including examination, knowledge, reasoning, and understanding, where our models achieve SOTA performance when compared with models of similar sizes. For instance, with 16.67% fewer parameters, PanGu-π-1.5B Pro achieves an average score of 60.64, outperforming Qwen-1.8B which achieves a score of 55.04.

2. Neural Architecture

In this section, we investigate the architecture design of tiny language models. The experiments are conducted on 50B tokens randomly sampled from the pre-trained dataset, with equal proportions of Chinese and English corpus. The baseline is a 1B parameter model with LLaMA-like architecture unless specified. The models constructed with different strategies are compared on ARC Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019) and C3 (Sun et al., 2020).

2.1. Compact Tokenizer

The tokenizer serves to map original natural language into tokens suitable for processing by large language models, with each token representing a word, subword, character, or symbol. A multilingual tokenizer typically has a large vocabulary to cover various corpora. However, in the context of a tiny language model, an overly large vocabulary can significantly occupy a substantial portion of the model's parameters. For instance, Qwen-7B (Bai et al., 2023), Baichuan2-7B (Yang et al., 2023), and PanGu-π-7B (Wang et al., 2023) have vocabulary sizes of 151936, 125696, 100883, respectively. The parameters of their heads and embedding layers account for 16.12% ,13.72%, 10.91% of the overall parameters. While the PanGu-π-1B model with 12 layers and a width of 2048, using the same tokenizer, sees the head and embedding layers' parameters comprising a substantial 36.8% of the total (Figure 3). This distribution leads to a significant allocation of parameters to vocabulary representation rather than the main body, potentially limiting the model's overall representation capacity. Therefore, compressing the tokenizer becomes essential for a tiny language model to reduce its parameter proportion.

Actually, we discover that substantial redundancy exists in the tokenizer. By initializing the tokenizers with the 100k vocabularies inherited from the PanGu-π model, we conducted a frequency analysis across a vast corpus comprising approximately 1.6T tokens. As depicted in Figure 2, it is evident that tokens exhibit a long-tail effect, where the top 48k vocabularies accounting for 97.86% of all the training corpus. We conduct experiments with six vocabulary sizes

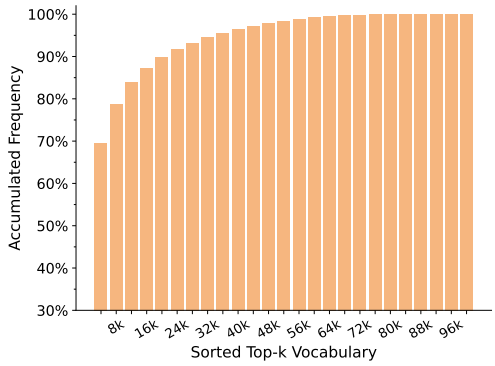


Figure 2. Accumulative frequency of the top-k vocabularies, where 97.86% data can be represented by a small 48k tokenizer.

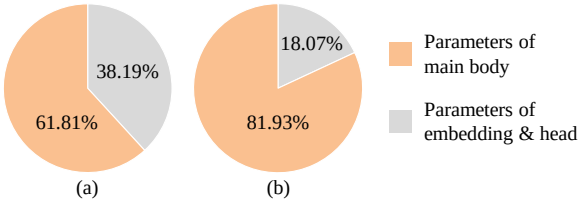


Figure 3. The parameter proportions of model's main body and tokenizer. (a) The large tokenizer inherited from large multilingual models (Wang et al., 2023). (b) Compact tokenizer by removing low-frequency vocabularies.

$\{8k, 16k, 32k, 48k, 72k, 100k\}$, which account for 78.68%, 87.24%, 94.49%, 97.86%, 99.84% and 100% accumulated frequency respectively. Over 50% vocabularies may be redundant as they cater to less than 3% of the corpus.

We advocate for the removal of low-frequency vocabularies to reduce their parameters. Table 1 illustrates the performance variations concerning tokenizer size[2]. The embedding and head layers constitute 18.07% of the 1B model's parameters when using a vocabulary of 48k, showcasing the best average performance followed by the model with a vocabulary of 32k. It is noteworthy that employing an excessively small vocabulary can result in performance degradation. For instance, with an 8k tokenizer covering less than 70% of the corpus, the model exhibits subpar performance on C3 and ARC-E datasets. The tokenizer with a size of 48k also exhibits a similar compression rate to that of the original 100k size tokenizer, which is evaluated across the entire training corpus. Therefore, we recommend using a compact tokenizer covering over 90% of the corpus, while ensuring the parameter proportion of embedding and head layers remains below 20%.

[2]All model's sizes are controlled to 1B by adjusting depth.

Table 1. Performance varies *w.r.t.* tokenizer size. PEHF stands for the proportion of embedding and head layers over the whole model.

Tokenizer	PEHL (%)	ARC-E	HellaSwag	C3	Avg.
8k	2.97	31.39	40.19	42.25	37.94
16k	6.01	30.34	40.10	45.64	38.69
32k	11.79	34.45	40.23	46.77	40.48
48k	18.07	34.39	**41.48**	**47.70**	**41.19**
72k	26.88	34.39	39.21	46.58	40.06
100k	38.19	**34.98**	39.11	47.10	40.40

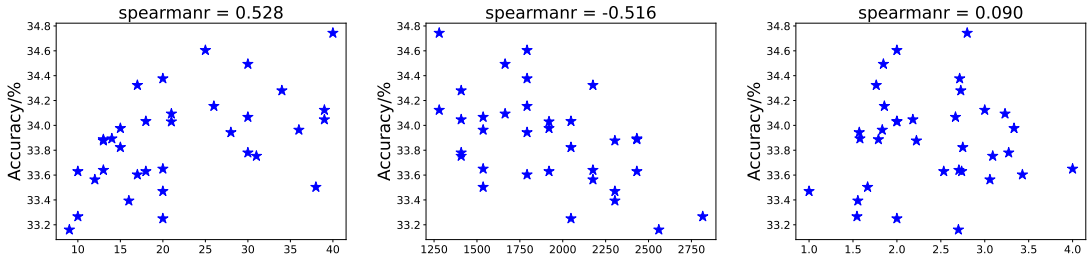
Table 2. Varying the depth and width of a 1B-size model with fixed vocabulary size and expanding rate. The speed is measured by tokens per second.

Depth	Width	Speed	ARC-E	HellaSwag	C3	Avg.
40	1280	12.81	**37.01**	41.00	**48.05**	**42.02**
30	1536	17.71	36.16	40.32	47.84	41.44
20	1792	29.49	34.39	**41.48**	47.70	41.19
15	2048	36.79	32.45	40.22	40.05	37.57
9	2560	**57.53**	32.63	31.06	42.68	35.46

2.2. Architecture Tweak

In this part, we focus on the neural architecture design of LLM for edge devices by exploring the impact of depth, width and the expanding rate of Feed-Forward Networks (FFN) on the performance of a 1B-size language model. Besides accuracy on downstream tasks, decoding speed is another important aspect for tiny language models. We test the end-to-end inference speed (tokens per second) when generating 510 new tokens under two prefix tokens using a randomly initialized model. The speed is tested on a single NVIDIA V100 GPU with batch size 20 using FP16. We fix the vocabulary size to 48k as suggested in Section 2.1. By constraining the model's size to 1B parameters, we explore the effects of varying the model's depth, width, and expansion rate individually. Firstly, we investigate the impact of adjusting two among the three components, while maintaining the third variable at a constant level, on the model's overall performance.

The impact of the depth and width. We thoroughly investigate representative configurations as outlined in Table 2, where we can conclude that *deeper tiny language models exhibit better performance, however, at the cost of inference speed.* As the depth of the model increases, the performance increases for almost all the three benchmarks. Meanwhile, we observed that when the depth is already 20, the performance improvement (41.19 $\rightarrow$ 42.02) by designing deeper architectures is minimal compared to the decrease of the inference speed (29.49 $\rightarrow$ 12.81). Therefore, we recommend setting the number of layers to around 20 for 1B-parameter model with a 48k tokenizer.



(a) Depth (b) Width (c) Expansion rate

Figure 4. Performance varies *w.r.t.* model's width, depth and expansion rate. The experiments are conducted on a streamlined dataset comprising 5B tokens. The accuracy is averaged among ARC Easy, HellaSwag and C3. Spearman coefficient is used to measure the correlation between performance and model's configure.

Table 3. Varying the expanding rate and width of a 1B-size model with fixed vocabulary size and depth.

EP Rate	Width	Speed	ARC-E	HellaSwag	C3	Avg.
1.00	2304	28.39	31.75	38.71	42.68	37.71
2.00	2048	28.68	33.33	41.34	**48.55**	41.07
2.77	1792	28.40	34.39	**41.48**	47.70	**41.19**
4.00	1536	28.53	**35.27**	39.36	47.18	40.60

Table 4. Performance under different random initialization strategies, where the constant standard deviation method performs best.

Initialization Method	ARC-E	HellaSwag	C3	Avg.
Constant	37.57	41.16	**49.04**	**42.59**
GPT2 (Radford et al., 2019)	**38.62**	39.34	48.44	42.13
InternLM (Team, 2023)	34.39	**41.48**	47.70	41.19

As shown in Table 3, we observe close inference speed for different expanding rates when the depth is fixed. It's obviously that the 1:1 setting gets significantly worse performance. To further investigate the interplay among depth, width and expansion rate, we sample about 30 different parameter configurations while maintaining the model size at 1B parameters and conduct training on a further streamlined dataset comprising 5B tokens. As illustrated in Figure 4, the correlation between the depth (width) and the downstream task's average performance is notably higher, with a Spearmanr correlation coefficient reaching up to 0.528. In contrast, there is no apparent linear relationship between the expansion rate and the model's ultimate performance.

Discussion. A compact tokenizer holds particular significance for a tiny language model, as it strikes a crucial balance between representation ability and implementation cost. The removal of low-frequency vocabularies enables the efficient elimination of substantial redundancy without significantly compromising representation capacity. Additionally, the architecture's configurations, such as width, depth, and expanding rate, exert a considerable influence on the final performance of a tiny model. Among them, depth is the primary factor for tiny language models, and deeper models usually achieve high performance at the expense of lower speed. Following the above observations, we design the architecture of PanGu-π Pro as detailed in Table 9.

3. Parameter Initialization

In this section, we investigate how to initialize model's parameters with a given neural architecture, including random initialization and inheriting parameters from a large model.

3.1. Random Initialization

When training model from scratch, the parameters are usually initialized with random numbers obeying normal distribution $N(0, \sigma^2)$ with zero mean and standard deviation σ. A series of well-known large language models carefully design the value of σ, especially changing it *w.r.t.*layers. For example, GPT2 (Radford et al., 2019) applies a scale of $1/\sqrt{N}$ to all linear layer parameters, where N is the number of residual layers. InternLM (Team, 2023) only applies the same scale to some special linear layer parameters, namely the out projection in MHA layers and the gate projection in MLP layers. We investigate these different initialization strategies for training tiny language model, whose results are shown in Table 4. We note that different strategies result in similar results. For simplicity and generalization, we recommend using a constant value for all layers when training tiny language models. More analyses are presented in Appendix B.

3.2. Parameter Inheritance

Besides random initialization, the initial parameter of this tiny language model can also inherit from a large language model. The strong generalization ability of large model is expected to transfer to the tiny model. Compared the tiny model, the large model usually has more layers with

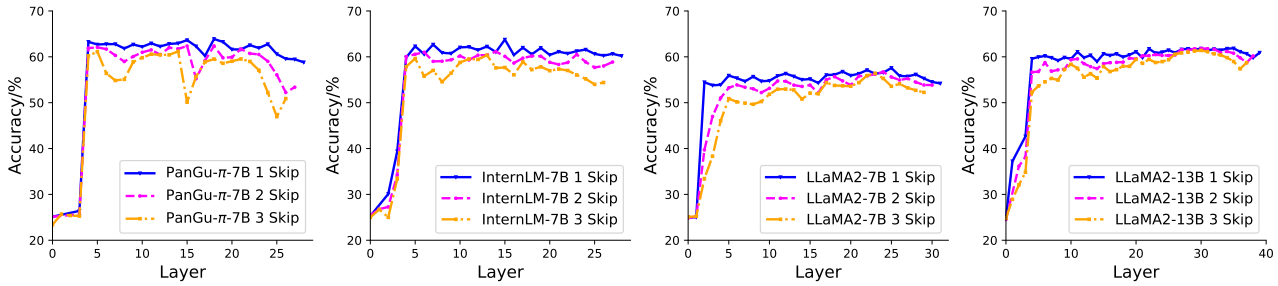


Figure 5. Performance of large language models when skipping a few layers. "x Skip" denotes adjacent x layers are discarded. Redundancies are observed within intermediate layers while the layers situated near the beginning and end are crucial for maintaining performance.

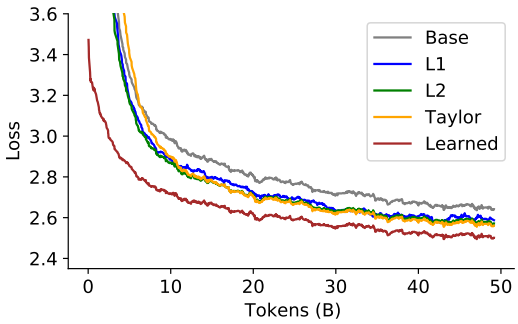


Figure 6. Training loss with different pruning strategies. "Base" denotes training from scratch without inheritance. Inheriting the model parameters with pruning yields a lower loss.

more neurons. We firstly select important layers and then recognize critical neurons in the selected layers.

Important layers selection. Considering that the tiny model usually has fewer layers than the large language model, the most important layers that contribute to the final performance are required to recognize. Consequently, we conduct ablation experiments to assess the impact of individual layers on the overall performance.

To uncover general principles regarding layer importance, we conduct a variety of experiments on multiple widely-used large language models, including LLaMA2-7B, LLaMA2-13B, InternLM-7B and PanGu-π-7B. During the inference phase, we skip specific layers and assess the resulting performance drop. Three types of layer skipping experiments are conducted for each model, involving skipping one layer, two neighboring layers, and three contiguous layers. The outcomes are depicted in Figure 5, where we analyze the average performance of large language models on three downstream tasks, *i.e.*, ARC-E, HellaSwag, and C3. The x-axis represents the skipped layer index, while the y-axis signifies performance accuracy.

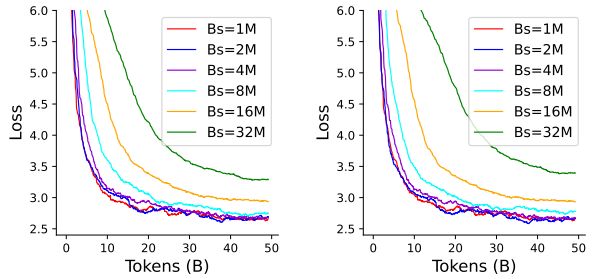
Some interesting common phenomenons are identified in these models. The shallow layers, especially the initial two to three layers, play a pivotal role in extracting features from input sequences. Removing these layers will incur significant performance drop on downstream tasks. Similarly, deep layers is also crucial, and removing them results in a deterioration of performance. Conversely, when the intermediate layers are removed, the performance is less affected, indicating that redundancy occurs within these layers. These layers are tend to be removed when inheriting parameters.

Intra-layer parameters selection. Within a layer, important parameters can be recognized by various metrics. How to recognizing essential parameters has well been discovered in the model pruning area (Frantar & Alistarh, 2023; Ma et al., 2023). The importance of neurons can be measured by various criteria and the most significant neurons are used as the initialization of tiny models. Weight norms, such as ℓ_1 and ℓ_2-norm, are commonly employed to measure importance, indicating that larger weights encapsulate more crucial information (Han et al., 2015; Guo et al., 2016; Lee et al., 2021; Tang et al., 2024). The first-order Taylor expansion (Lee et al., 2019; Tanaka et al., 2020), which incorporates both weight values and gradients, is regarded as a more accurate estimation of the output. In addition to empirical criteria, essential weights can also be identified through binary masks, which are automatically learned during the training process (Xia et al., 2023; Tang et al., 2020). In the subsequent sections, we adopt these methodologies to select vital parameters from the PanGu-π-7B (Wang et al., 2023) model as initial values for a 1B model with smaller weight dimensions.

Training loss curves and evaluation results are presented in Figure 6 and Table 5. In comparison to the baseline model initialized randomly, each of the small models initialized with the pruning strategy converges to a lower loss. Among the empirical criteria, the Taylor expansion yields superior

Table 5. Comparison between different parameter inheritance strategies. "Base" denotes training without inheritance.

Inheritance Strategy	ARC-E	HellaSwag	C3	Avg.
Base	36.68	40.34	49.15	42.06
L1 (Ma et al., 2023)	39.51	47.70	50.96	46.06
L2 (Ma et al., 2023)	41.98	48.33	50.68	47.00
Taylor (Ma et al., 2023)	**43.21**	48.43	**52.05**	47.90
Learned (Xia et al., 2023; Tang et al., 2020)	40.74	**51.77**	51.73	**48.08**



(a) $r = 0.5$ (b) $r = 1.0$

Figure 7. Training losses under different batchsize & learning rate.

results, primarily attributed to its accurate estimation of neuron importance. The model pruned using learnable masks starts with a significantly lower initial loss than the other models and ultimately converging to the lowest loss. Evaluation results across the three datasets validate the effectiveness of parameter inheritance. We recommend inheriting model parameters with the learnable strategies.

Discussion. The aforementioned observation confirms that the initial parameters of a model exert a substantial influence on both the convergence rate and ultimate performance of a tiny language model. Opting to inherit parameters from a larger model is generally a more favorable choice, as it allows the smaller model to assimilate the robust representation abilities of its larger models. The process of selecting significant parameters is a crucial step in this regard. Through thorough empirical investigation, we have observed that intermediate layers tend to exhibit more redundancy, and data-driven learnable masks proves effective in excavating these redundant parameters.

4. Model Optimization

In this section, we investigate the optimization strategies with given neural architecture and initial parameters. Firstly, the scaling rule between learning rate and batch size is analyzed. Besides, we observe substantial performance improvement from continuing multiple rounds of training.

4.1. Batchsize & Learning Rate

In practical language model training, the choice of batch size is frequently tailored to the computational resources at hand. When dealing with a limited number of GPUs, opting for a smaller batch size becomes necessary. Conversely, in scenarios where a substantial number of GPUs is at our disposal, enlarging the batch size can effectively diminish the number of iterations, thereby expediting the overall training process.

However, adjusting the batch size typically has a notable impact on the final performance of the model. When increasing the batch size, it is common for the learning rate to be adjusted proportionally. We explore their combined effects in Figure 7 and Figure 8, using the formula $lr = (bs/bs_0)^r \times lr_0$, where the default batchsize bs_0 and learning rate lr_0 are set to 1M and 1×10^{-4}, respectively. r denotes the increment rate, which is usually set as 0.5 or 1.0 (Krizhevsky, 2014; Goyal et al., 2017). When the batchsize is smaller than 4M, the convergence speeds with different learning rates remain consistent. When the batchsize further increases, a moderate increment rate ($r = 0.5$) is preferable. With the same training tokens, employing an excessively large batchsize ($\geq$16M) adversely affects the convergence speed. In the majority of cases, a batch size smaller than 4M is considered the safe range for optimizing model performance. Otherwise, optimization strategies need to be specifically tailored for large batch sizes (Keskar et al., 2016; You et al., 2017; 2019).

4.2. Multiple-Round Training

The existing methods usually train the language model with only one round, *i.e.*, all the data are only used for one time to update the model, leaving the model's parameters unconverged. Besides, learning on large corpora may suffer from the catastrophic forgetting (Toneva et al., 2018; Winata et al., 2023) issue, *i.e.*, the model performance drops for data seen before. For tiny models, the limited model capacity makes the forgetting problem more serious. Continuing training the model can further reduce the training loss.

We conduct a simple experiment to validate the forgetting problem. As the training loss is calculated by the model parameters at the corresponding timestamp and the model parameters are updated as the training continues, the later data tend to have low loss values. Therefore, We recompute the batch-wise loss on the previous data using a PanGu-π-1B model trained on 1.6T tokens. The training data is evenly and randomly divided into eight parts before training. Figure 9 shows how loss value varies *w.r.t.* data on each part. The high loss indicate previous knowledge have been seriously forgot. Therefore, it is necessary to train the model for multiple rounds to fit the forgotten data.

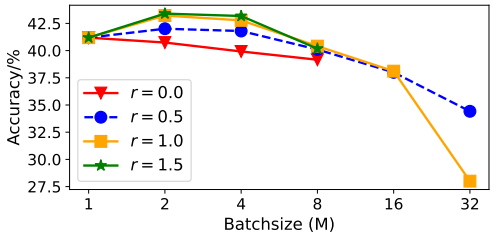


Figure 8. Performance under different batchsize & learning rate.

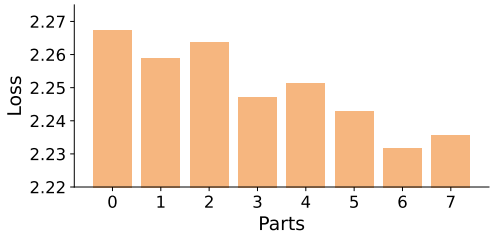


Figure 9. Loss value varies *w.r.t.* data on different iterations using a pretrained PanGu-π-1B model. The loss is averaged among batches in each part.

To reduce the training cost, we propose a simple data refining strategy for the multiple-round training. Considering some examples are hard to fit, they should be used for further training with a high probability. Denoting the loss values in certain part as $L = \{l_1, l_2, \cdots, l_N\}$, where N is the total batches in this part. Note that data are randomly shuffled in the training process, and thus each batch contains various type data. In each part, the loss values are normalized, denoting the sampling probability, *i.e.*, $p_i = \frac{\exp(l_i)}{\sum_{j=1}^{N}\exp(l_j)}$. In the next round training we sample N_0 batches out of N according to the sampling probability p. The impact of sampling rate ($r = \frac{N_0}{N}$) is shown in Table 6. It shows that a higher sampling rate tends to achieve high performance. The performance improvement is marginal when the sampling rate r exceeds 50%. We plot how the evaluation metric on HellaSwag evolves during training in Figure 10. As the second-round training goes ahead, the accuracy on HellaSwag keeps rising but get converged in the later phase. In Table 7, we also try to train the models with more rounds. However, the performance also saturate gradually. To achieve a balance between performance and training efficiency, we recommend to train the model with two rounds and set sampling rate to 50%.

Discussion. In contrast to larger models, tiny language models face a significant challenge of data forgetting due to their limited capacity. As a result, adopting a multi-round training approach becomes crucial to enhance performance. Employing data sampling proves effective in improving learning on challenging examples while simultaneously reducing training costs. Additionally, the choice of batch size and learning rate plays a pivotal role in model performance. For optimal results, it is advisable to use a batch size smaller than 4M, unless a specialized large batch optimizer is employed for a tailored approach.

Table 6. Sampling rate for the next round training. The model is training with two rounds. $r = 0$ denotes training with one round.

Sampling Rate r	ARC-E	HellaSwag	C3	Average
0%	42.68	57.95	54.19	51.61
25%	43.95	59.55	56.01	53.17
50%	45.33	60.67	57.37	54.46
75%	45.52	60.34	58.16	54.67
100%	44.98	60.88	58.74	54.87

Table 7. The impact of number of training rounds. The sampling rate r is set to 50%.

Training round	ARC-E	HellaSwag	C3	Average
Single round	42.68	57.95	54.19	51.61
Two round	45.33	60.67	57.37	54.46
Three round	45.11	61.32	56.88	54.44

5. PanGu-π Pro

Based on the above extensive and rigorous set of experiments, we make a significant improvement on our previous PanGu-π-1B and meanwhile construct a larger and more powerful PanGu-π-1.5B Pro. In this section, we make a comparison with the existing open-source tiny language models, where our result establishes a new SOTA. Specifically, PanGu-π-1.5B Pro outperforms the recent proposed Qwen-1.8B (Bai et al., 2023) and Phi2-2.7B (Li et al., 2023b), which is 1.8x larger, in average performance.

Implementation details. The pre-training data, which consists of 1.6T tokens, is gathered from diverse sources from the Internet, covering English and Chinese corpus with around $1:1$ scale. The used 48k tokenizer is built by byte-pair encoding (BPE, Shibata et al. (1999)) from SentencePiece (Kudo & Richardson, 2018) upon our data. Our models are trained using the AdamW optimizer (Loshchilov & Hutter, 2017) with $\beta_1 = 0.9, \beta_2 = 0.95$ utilizing the cosine learning rate decay (Loshchilov & Hutter, 2016) with an initial learning rate 2×10^{-4}. The total batch size for the training process is 2M. We follow the PanGu-π (Wang et al., 2023) architecture while making PanGu-π-1B much deeper. The detailed configuration can be found in Table 9. We set the expansion rate to 2.77 as suggested in Table 3. For

Table 8. Comparison with SOTA open-source tiny language models. The best model is listed in bold and second-best is listed in underlined.

	Examination				Knowledge	Reasoning		Understanding			
Models	C-Eval	CMMLU	MMLU	AGI-Eval	BoolQ	AX-b	PIQA	EPRSTMT	XSum	C3	**Average**
MobileLLaMA-1.4B	23.93	25.10	25.05	18.53	58.75	45.20	71.27	46.25	18.19	37.42	36.97
Sheared-LLaMA-1.3B	24.28	25.10	25.77	18.01	62.39	43.57	72.91	46.25	16.44	35.45	37.02
TinyLLaMA-1.1B	27.85	24.64	25.75	18.54	56.06	45.47	70.62	46.25	20.15	36.71	37.20
MobileLLaMA-2.7B	23.53	25.55	26.63	18.43	54.74	55.80	72.85	46.25	16.96	36.11	37.69
Chinese-LLaMA2-1.3B	28.70	24.78	24.55	19.40	56.79	47.46	56.91	72.50	8.90	43.12	38.31
RWKV-5-1.5B	25.92	25.14	25.66	19.01	62.29	54.05	71.22	46.25	20.67	49.15	39.94
Phi-1.3B	27.78	25.85	44.32	23.42	73.52	44.20	76.99	50.00	14.90	38.96	41.99
PanGu-π-1B	36.85	35.90	35.96	30.77	58.44	43.48	61.92	55.62	15.92	49.21	42.41
Open-LLaMA-3B	27.50	25.42	27.09	20.68	60.58	52.72	77.09	82.50	19.75	43.23	43.66
Phi2-2.7B	31.86	32.18	58.49	28.51	**77.40**	43.57	78.89	46.25	13.66	40.11	45.09
PanGu-π-1B Pro (**Ours**)	46.50	46.56	50.38	41.58	63.43	53.99	64.96	74.38	18.40	52.66	51.28
Qwen-1.8B	**53.60**	**52.12**	46.43	35.83	64.31	57.79	73.83	**88.12**	20.03	58.30	55.04
PanGu-π-1.5B Pro (**Ours**)	52.39	48.51	**62.54**	**44.89**	70.61	**67.93**	**79.55**	86.12	**24.61**	**69.24**	**60.64**

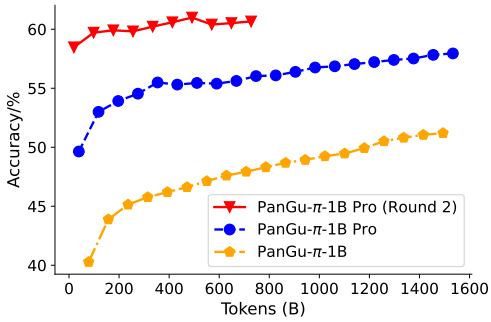


Figure 10. Accuracies of PanGu-π-1B and PanGu-π-1B Pro on HellaSwag during training.

the parameters initialization method, we inherit parameters from PanGu-π-7B via learnable binary mask after removing intermediate redundant layers. We use the Huawei Ascend 910 card to train and evaluate the proposed PanGu-π Pro.

Table 9. Model configuration.

Model	Width	Depth	Vocabulary	Initialization
PanGu-π-1B	2048	12	100883	Random
PanGu-π-1B Pro	1792	21	48000	PanGu-π-7B
PanGu-π-1.5B Pro	2048	22	48000	PanGu-π-7B

Benchmarks. We use OpenCompass (Contributors, 2023) to evaluate on an extensive suite of downstream tasks, covering examination, knowledg, reasoning, and understanding abilities for a comprehensive comparison. C-Eval (Huang et al., 2023) is a Chinese benchmark to evaluate the knowledge and reasoning abilities. CMMLU (Li et al., 2023a) covers 67 topics including science, engineering, and humanities. MMLU (Hendrycks et al., 2021) proposes an English benchmark for measuring LLM's multitask accuracy by covering 57 tasks including mathematics, history, computer science, and law. AGI-Eval (Zhong et al., 2023) is a benchmark specifically designed to evaluate the general abilities in tasks pertinent to human cognition and problem-solving. BoolQ (Clark et al., 2019) is a reading comprehension dataset to evaluate the difficult entailment-like inference ability of LLMs. AX-b (Wang et al., 2020) is a broad-coverage diagnostic task and PIQA (Bisk et al., 2020) is a physical interaction question-answering task. EPRSTM (Xu et al., 2021) is a binary sentiment analysis dataset based on product reviews. XSum (Narayan et al., 2018) is a summarization task collected from the British Broadcasting Corporation and C3 (Sun et al., 2020) contains 13,369 documents and their associated 19,577 multiple-choice questions.

Comparison with tiny language models. We collect multiple tiny language models with different sizes, ranging from 1B to 3B. These include TinyLLaMA-1.1B (Peiyuan Zhang & Lu, 2023), Chinese-LLaMA2-1.3B (Cui et al., 2023), Sheared-LLaMA-1.3B (Xia et al., 2023), and Open-LLaMA-3B (Geng & Liu, 2023). Meituan (Chu et al., 2023) released MobileLLaMA-1.4B and MobileLLaMA-2.7B that were trained from scratch on the RedPajama dataset (Computer, 2023). Microsoft developed the series of Phi (Li et al., 2023b) that focusing on using "textbook-quality" data with small language models. RWKV-5-1.5B (Peng et al., 2023) is a parallelizable RNN with Transformer-level LLM Performance. Qwen-1.8B (Bai et al., 2023) is pretrained on 2.2 trillion tokens including web texts, books, codes, *etc*.

Extensive experiments in Table 8 show that PanGu-π-1.5B Pro significantly outperforms existing LLMs of similar or even larger sizes, *e.g.*, Phi2-2.7B and Open-LLaMA-3B. We observe a notable improvement of 8.77 on average performance from PanGu-π-1B to PanGu-π-1B Pro. With 16.67% fewer parameters, PanGu-π-1.5B Pro outperforms Qwen-1.8B (Bai et al., 2023) and exhibits the best or second-best performance in the vast majority of the benchmarks. Overall, our model exhibits consistently better average performance

compared to the current state-of-the-art models.

From PanGu-π-1B, we gradually add the core components to validate the effectiveness of our methodology. As shown in Figure 1, the removal of low-frequency vocabularies leads to an improvement of average performance from 42.41 to 44.11 while the architecture tweak contributes another 2.42 improvement. Parameter inheritance, the most effective approach, pushes the average performance to 49.79. Multiple-round training further enhances the average performance of PanGu-π-1B Pro.

6. Conclusion and Discussion

In this paper, we systematically discuss how to construct a tiny language model from three perspectives, *i.e.*, neural architecture, parameter initialization, and optimization strategies. By carefully designed empirical study, we recognized several effective design formulas to improve performance with given parameter restriction and data size, including compact tokenizer, architecture tweak, parameter inheritance, multiple-round training *etc*. Then we train PanGu-π Pro models with 1B and 1.5B parameters, which significantly improve performance than the baseline models.

Based on the observations, we also note several intriguing directions for further exploration. In terms of neural architecture, how to directly learn a compact tokenizer that seamlessly integrates both representation ability and parameter efficiency. Additionally, exploring hardware-friendly architectures holds promise for mitigating computational and storage costs. For example, GQA (Ainslie et al., 2023) is an effective strategy to reduce RAM require of edge devices (refer to Appendix A). Concerning model optimization, the importance of effective parameter initialization cannot be overstated, setting the model on a path toward high performance from the outset. Nevertheless, the challenge remains in identifying effective parameters, which is an open question in the field. Besides, the training characteristics of tiny models differ significantly from their larger counterparts. For instance, within the framework of multiple-round training, there is an urgent demand for the development of new parameter optimization techniques and data refining methods. Numerous questions warrant in-depth exploration, and we hope the findings presented in this paper can spark inspiration for further research.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., Hui, B., Ji, L., Li, M., Lin, J., Lin, R., Liu, D., Liu, G., Lu, C., Lu, K., Ma, J., Men, R., Ren, X., Ren, X., Tan, C., Tan, S., Tu, J., Wang, P., Wang, S., Wang, W., Wu, S., Xu, B., Xu, J., Yang, A., Yang, H., Yang, J., Yang, S., Yao, Y., Yu, B., Yuan, H., Yuan, Z., Zhang, J., Zhang, X., Zhang, Y., Zhang, Z., Zhou, C., Zhou, J., Zhou, X., and Zhu, T. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Chu, X., Qiao, L., Lin, X., Xu, S., Yang, Y., Hu, Y., Wei, F., Zhang, X., Zhang, B., Wei, X., and Shen, C. Mobilevlm: A fast, strong and open vision language assistant for mobile devices, 2023.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018.

Computer, T. Redpajama: An open source recipe to reproduce llama training dataset, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Contributors, O. Opencompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass, 2023.

Cui, Y., Yang, Z., and Yao, X. Efficient and effective text encoding for chinese llama and alpaca. *arXiv preprint arXiv:2304.08177*, 2023. URL https://arxiv.org/abs/2304.08177.

Frantar, E. and Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Geng, X. and Liu, H. Openllama: An open reproduction of llama, May 2023. URL https://github.com/openlm-research/open_llama.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient dnns. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 1379–1387, 2016.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015. URL http://arxiv.org/abs/1506.02626.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Huang, Y., Bai, Y., Zhu, Z., Zhang, J., Zhang, J., Su, T., Liu, J., Lv, C., Zhang, Y., Lei, J., Fu, Y., Sun, M., and He, J. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *arXiv preprint arXiv:2305.08322*, 2023.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Krizhevsky, A. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

Kudo, T. and Richardson, J. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

Lee, J., Park, S., Mo, S., Ahn, S., and Shin, J. Layer-adaptive sparsity for the magnitude-based pruning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=H6ATjJ0TKdf.

Lee, N., Ajanthan, T., and Torr, P. H. S. Snip: single-shot network pruning based on connection sensitivity. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=B1VZqjAcYX.

Li, H., Zhang, Y., Koto, F., Yang, Y., Zhao, H., Gong, Y., Duan, N., and Baldwin, T. Cmmlu: Measuring massive multitask language understanding in chinese, 2023a.

Li, Y., Bubeck, S., Eldan, R., Del Giorno, A., Gunasekar, S., and Lee, Y. T. Textbooks are all you need ii: **phi-1.5** technical report. *arXiv preprint arXiv:2309.05463*, 2023b.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Ma, X., Fang, G., and Wang, X. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023.

Narayan, S., Cohen, S. B., and Lapata, M. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization, 2018.

Peiyuan Zhang, Guangtao Zeng, T. W. and Lu, W. Tinyllama, Sep 2023. URL https://github.com/jzhang38/TinyLlama.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Cao, H., Cheng, X., Chung, M., Grella, M., GV, K. K., et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Ren, X., Zhou, P., Meng, X., Huang, X., Wang, Y., Wang, W., Li, P., Zhang, X., Podolskiy, A., Arshinov, G., et al. Pangu-Σ: Towards trillion parameter language model with sparse heterogeneous computing. *arXiv preprint arXiv:2303.10845*, 2023.

Shazeer, N. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., Shinohara, T., and Arikawa, S. Byte pair encoding: A text compression scheme that accelerates pattern matching. 1999.

Sun, K., Yu, D., Yu, D., and Cardie, C. Investigating prior knowledge for challenging chinese machine reading comprehension. *Transactions of the Association for Computational Linguistics*, 2020. URL https://arxiv.org/abs/1904.09679v3.

Tanaka, H., Kunin, D., Yamins, D. L. K., and Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Tang, Y., Wang, Y., Xu, Y., Tao, D., Xu, C., Xu, C., and Xu, C. Scop: Scientific control for reliable neural network pruning. *Advances in Neural Information Processing Systems*, 33:10936–10947, 2020.

Tang, Y., Wang, Y., Guo, J., Tu, Z., Han, K., Hu, H., and Tao, D. A survey on transformer compression. *arXiv preprint arXiv:2402.05964*, 2024.

Team, I. Internlm: A multilingual language model with progressively enhanced capabilities, 2023.

Toneva, M., Sordoni, A., Combes, R. T. d., Trischler, A., Bengio, Y., and Gordon, G. J. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*, 2018.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Superglue: A stickier benchmark for general-purpose language understanding systems, 2020.

Wang, Y., Chen, H., Tang, Y., Guo, T., Han, K., Nie, Y., Wang, X., Hu, H., Bai, Z., Wang, Y., et al. Pangu-π: Enhancing language model architectures via nonlinearity compensation. *arXiv preprint arXiv:2312.17276*, 2023.

Wei, T., Zhao, L., Zhang, L., Zhu, B., Wang, L., Yang, H., Li, B., Cheng, C., Lü, W., Hu, R., Li, C., Yang, L., Luo, X., Wu, X., Liu, L., Cheng, W., Cheng, P., Zhang, J., Zhang, X., Lin, L., Wang, X., Ma, Y., Dong, C., Sun, Y., Chen, Y., Peng, Y., Liang, X., Yan, S., Fang, H., and Zhou, Y. Skywork: A more open bilingual foundation model, 2023.

Winata, G. I., Xie, L., Radhakrishnan, K., Wu, S., Jin, X., Cheng, P., Kulkarni, M., and Preotiuc-Pietro, D. Overcoming catastrophic forgetting in massively multilingual continual learning. *arXiv preprint arXiv:2305.16252*, 2023.

Xia, M., Gao, T., Zeng, Z., and Chen, D. Sheared llama: Accelerating language model pre-training via structured pruning. 2023.

Xu, L., Lu, X., Yuan, C., Zhang, X., Xu, H., Yuan, H., Wei, G., Pan, X., Tian, X., Qin, L., et al. Fewclue: A chinese few-shot learning evaluation benchmark. *arXiv preprint arXiv:2107.07498*, 2021.

Yang, A., Xiao, B., Wang, B., Zhang, B., Bian, C., Yin, C., Lv, C., Pan, D., Wang, D., Yan, D., et al. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*, 2023.

Yi. A series of large language models trained from scratch by developers at 01-ai. https://github.com/01-ai/Yi, 2023.

You, Y., Gitman, I., and Ginsburg, B. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence?, 2019.

Zeng, A., Liu, X., Du, Z., Wang, Z., Lai, H., Ding, M., Yang, Z., Xu, Y., Zheng, W., Xia, X., et al. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*, 2022.

Zhong, W., Cui, R., Guo, Y., Liang, Y., Lu, S., Wang, Y., Saied, A., Chen, W., and Duan, N. Agieval: A human-centric benchmark for evaluating foundation models, 2023.

A. Multi-head Attention v.s. Group-query Attention

Current mainstream LLaMA architectures, such as Qwen (Bai et al., 2023) and InternLM (Team, 2023), commonly employ multi-head attention (MHA) mechanisms (Vaswani et al., 2017). However, storing a large amount of KV (key and value) cache in memory-constrained edge devices poses a significant challenge, especially in long text input scenarios. Group query attention (Ainslie et al., 2023) is recently proposed to seek a balance between multi-head attention and multi-query attention (Shazeer, 2019). By sharing the same key-value heads for all queries in each group, the RAM requirement for the KV cache can be largely reduced in edge devices. We convert our pretrained PanGu-π-1.5B Pro to its GQA version by mean-pooling (Ainslie et al., 2023) the KV heads in each group and then continually training on only 5% of the origin dataset. The number of groups is set to 8. As shown in Table 10, the converted GQA version model exhibits comparable performance to the MHA counterpart with fewer parameters.

Table 10. Comparison between MHA and GQA.

Attn.	Size	C-Eval	CMMLU	MMLU	AGI-Eval
MHA	1.5B	52.91	49.51	53.76	44.42
GQA	1.4B	48.75	46.94	51.97	43.59

B. Improved Random Initialization

When training the model from scratch, we typically initialise the weight of linear layers using a normal distribution $N(0, \sigma^2)$ with zero mean and standard deviation σ. Some methods (Radford et al., 2019; Team, 2023) use different standard deviations for different layers. Figure 11 shows the standard deviations of different layers after pretraining, indicating that the parameter distributions will be similar from different initialisation values. In particular, the variance of the four linear layers within the MHA layer varies with the depth, while the variance of the linear layers within the MLP layers remains almost constant.

We introduce an improved initialization method that adapts the standard deviation of different linear weights to the depth of the model. In the MHA, the initial standard deviation of the query and key projection parameters decreases from $\sqrt{2}\sigma$ to σ, while that of the value and out projection parameters increases from $\sigma/\sqrt{2}$ to σ, as the layer index increases. In the MLP, we keep the initial standard deviation of the linear weights constant at σ for all layers. As demonstrated in Table 11, our approach exhibits a marginal superiority over the constant initialization approach.

C. The Impact of Attention Head

Table 12 shows how performance varies *w.r.t.* the number of attention heads. The results show that it does not affect the inference speed or the model performance significantly, as long as the hidden size is fixed. The head dimension is set to 128 in PanGu-π-1B Pro and PanGu-π-1.5B Pro.

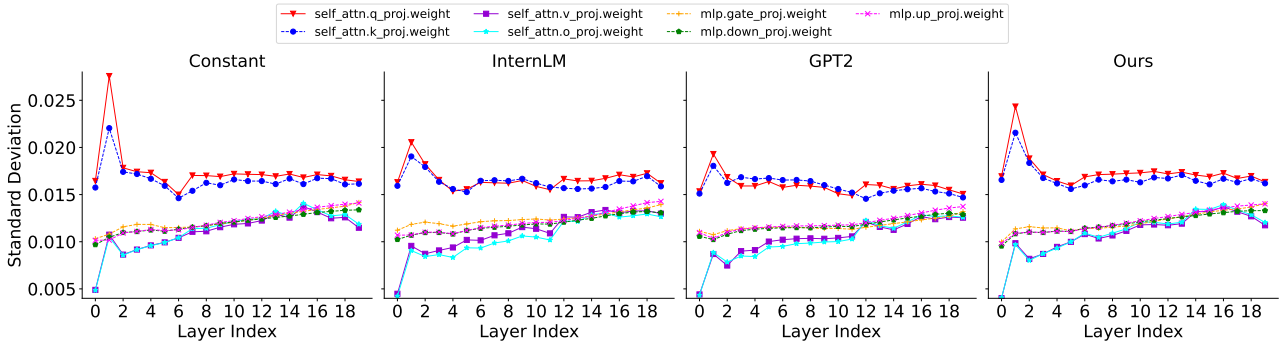


Figure 11. The standard deviations of different layers after pretraining.

Table 11. Performance under different initialization strategies. Our method exhibits a slight edge over the constant standard deviation approach.

Initialization Method	ARC-E	HellaSwag	C3	Avg.
Constant	37.57	41.16	49.04	42.59
GPT2 (Radford et al., 2019)	**38.62**	39.34	48.44	42.13
InternLM (Team, 2023)	34.39	41.48	47.70	41.19
Ours	37.57	**42.00**	**49.26**	**42.94**

Table 12. Varying the number of attention heads.

Heads	Head Dimension	Speed	ARC-E	HellaSwag	C3	Avg.
14	128	29.49	34.39	41.48	47.70	41.19
28	64	30.11	35.39	41.63	48.09	41.70
56	32	30.49	33.16	41.36	48.17	40.90

D. Weight Decay

Weight decay (Loshchilov & Hutter, 2017) is a commonly employed regularization method aimed at mitigating overfitting on the training set. We delve into its impact in Table 13. Elevating the weight decay imparts more robust regularization, albeit at the expense of constraining the model's representation capacity. Through empirical experiments, we observe that the model attains optimal performance when the weight decay is set at 0.1.

Table 13. Performance under different weight decay. The model achieved the best performance with a weight decay of 0.1.

Weight Decay	ARC-E	HellaSwag	C3	Average
0.2	34.68	36.15	45.31	38.71
0.1	34.39	**41.48**	**47.70**	**41.19**
0.01	**34.74**	36.76	45.26	38.92
0.001	33.59	37.07	44.93	38.53
0.0001	31.22	37.76	44.11	37.70

E. Additional Results of Layer Selection

The layer skipping results of single downstream tasks are released in Figure 12. The performance trend of single downstream tasks are consistent to the average results shown Figure 5 of the main paper. Layers situated near the beginning and end of the model often carry more significance than the intermediate layers.

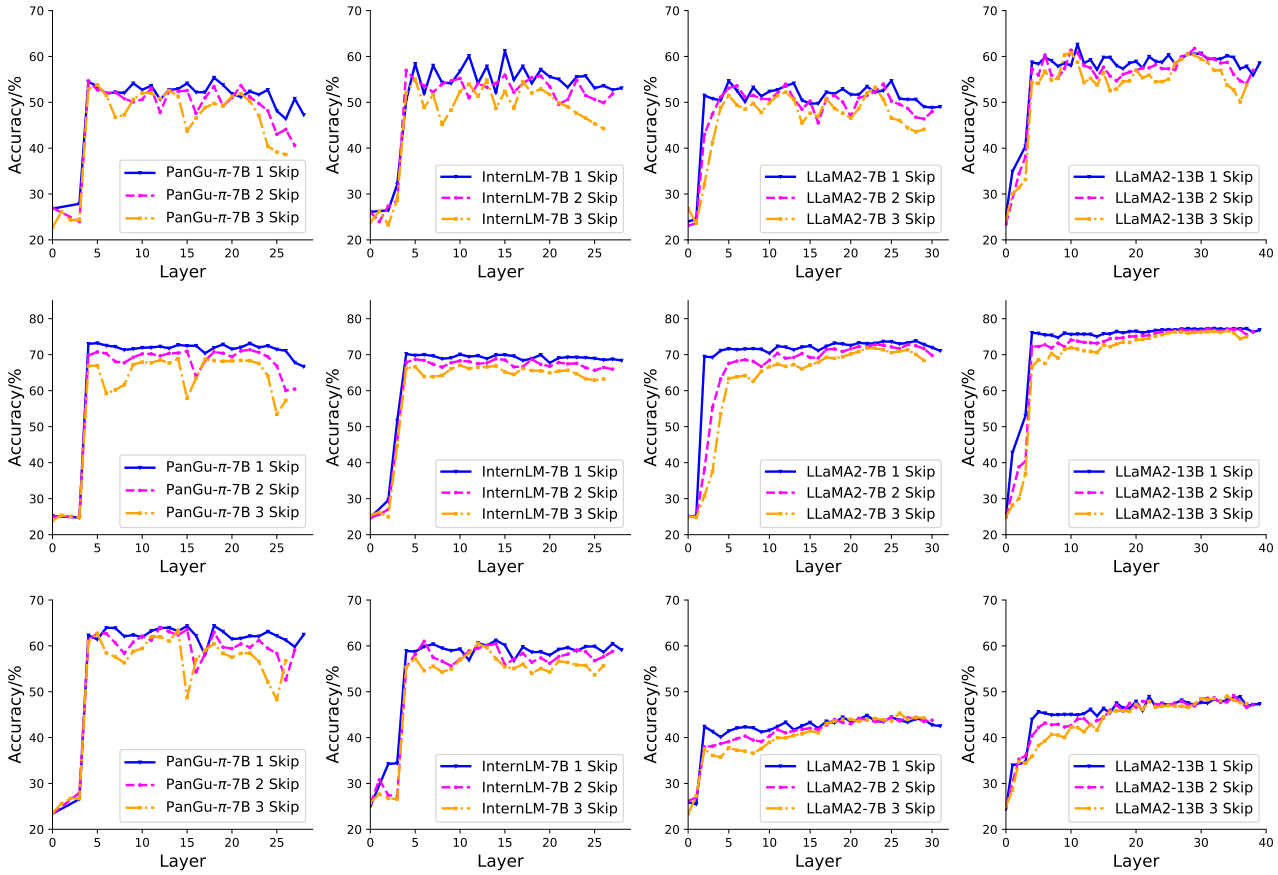


Figure 12. Layer-skipped performance of large language models on single downstream tasks. From top to bottom, the tasks are ARC-E, HellaSwag, and C3, respectively.