
TinyLlama: An Open-Source Small Language Model

Peiyuan Zhang* Guangtao Zeng* Tianduo Wang Wei Lu
StatNLP Research Group
Singapore University of Technology and Design
{peiyuan_zhang, tianduo_wang, luwei}@sutd.edu.sg
guangtao_zeng@mymail.sutd.edu.sg



Abstract

We present TinyLlama, a compact 1.1B language model pretrained on around 1 trillion tokens for up to 3 epochs¹. Building on the architecture and tokenizer of Llama 2 (Touvron et al., 2023b), TinyLlama leverages various advances contributed by the open-source community, e.g., FlashAttention (Dao, 2023) and Lit-GPT (Lightning-AI, 2023), achieving better computational efficiency. Despite its relatively small size, TinyLlama demonstrates remarkable performance in a series of downstream tasks. It significantly outperforms existing open-source language models with comparable sizes. Our model checkpoints and code are publicly available on GitHub at <https://github.com/jzhang38/TinyLlama>.

1 Introduction

Recent progress in natural language processing (NLP) has been largely propelled by scaling up language model sizes (Brown et al., 2020; Chowdhery et al., 2022; Touvron et al., 2023a,b). Large Language Models (LLMs) pre-trained on extensive text corpora have demonstrated their effectiveness on a wide range of tasks (OpenAI, 2023; Touvron et al., 2023b). Some empirical studies demonstrated emergent abilities in LLMs, abilities that may only manifest in models with a sufficiently large number of parameters, such as few-shot prompting (Brown et al., 2020) and chain-of-thought reasoning (Wei et al., 2022). Other studies focus on modeling the scaling behavior of LLMs (Kaplan et al., 2020; Hoffmann et al., 2022). Hoffmann et al. (2022) suggest that, to train a compute-optimal model, the size of the model and the amount of training data should be increased proportionally. This provides a guideline on how to optimally select the model size and allocate the amount of training data when the compute budget is fixed.

*The first two authors contributed equally.

¹Our latest model, TinyLlama v1.1, is only trained for 2 trillion tokens. More details about this latest version will be elaborated in the later section.

Although these works show a clear preference on large models, the potential of training smaller models with larger datasets remains under-explored. Instead of training compute-optimal language models, [Touvron et al. \(2023a\)](#) highlight the importance of the inference budget, instead of focusing solely on training compute-optimal language models. Inference-optimal language models aim for optimal performance within specific inference constraints. This is achieved by training models with more tokens than what is recommended by the scaling law ([Hoffmann et al., 2022](#)). [Touvron et al. \(2023a\)](#) demonstrates that smaller models, when trained with more data, can match or even outperform their larger counterparts. Also, [Thaddée \(2023\)](#) suggest that existing scaling laws ([Hoffmann et al., 2022](#)) may not predict accurately in situations where smaller models are trained for longer periods.

Motivated by these new findings, this work focuses on exploring the behavior of smaller models when trained with a significantly larger number of tokens than what is suggested by the scaling law ([Hoffmann et al., 2022](#)). Specifically, we train a Transformer decoder-only model ([Vaswani et al., 2017](#)) with 1.1B parameters using up to 3 trillion tokens. To our knowledge, this is the first attempt to train a model with 1B parameters using such a large amount of data. Following the same architecture and tokenizer as Llama 2 ([Touvron et al., 2023b](#)), we name our model TinyLlama. TinyLlama shows competitive performance compared to existing open-source language models of similar sizes. Specifically, TinyLlama surpasses both OPT-1.3B ([Zhang et al., 2022](#)) and Pythia-1.4B ([Biderman et al., 2023](#)) in various downstream tasks. Our TinyLlama is open-source, aimed at improving accessibility for researchers in language model research. We believe its excellent performance and compact size make it an attractive platform for researchers and practitioners in language model research.

2 Pre-training

This section outlines the pre-training process of TinyLlama. First, We introduce how we composed the datasets used for pre-training. Next, we describe the model architecture and the hyperparameters implemented during the pre-training phase.

2.1 Pre-training data

We utilize a blend of natural language and code data to pre-train TinyLlama, drawing from two primary sources: SlimPajama ([Soboleva et al., 2023](#)) and the training data of StarCoder ([Li et al., 2023](#)). A detailed introduction to these two datasets is provided below.

SlimPajama It is a high-quality corpus specifically created for training large language models. This corpus, derived from RedPajama ([Together Computer, 2023](#)), underwent additional cleaning and deduplication processes. The original RedPajama corpus, an open-source research project, was designed to replicate the pretraining data of Llama ([Touvron et al., 2023a](#)) and contains over 1.2 trillion tokens. After extensive filtering to remove low-quality and duplicated content, SlimPajama retains only 50% of the original tokens from RedPajama.

StarCoder Training Dataset This dataset is collected to train StarCoder ([Li et al., 2023](#)), a powerful open-source large code language model. It contains code data in 86 programming languages. In addition to code, it also includes GitHub issues and text-code pairs that involve natural languages. As SlimPajama also contains data from GitHub², we remove the GitHub subset from SlimPajama and only sample code-related data from the StarCoder training dataset to avoid duplication.

After merging these two datasets, we obtained a combined total of approximately 950 billion tokens for pre-training, processed using the Llama tokenizer ([Touvron et al., 2023a,b](#)). TinyLlama is then trained on these tokens across approximately three epochs, cumulatively processing 3 trillion tokens. The training data is derived from the SlimPajama and Starcoder datasets at a sampling ratio of approximately 7:3.

2.2 Architecture

Following the series of the Llama models ([Touvron et al., 2023a,b](#)), our model is based on the decoder-only Transformer ([Vaswani et al., 2017](#)). The details of hyperparameters for our model architecture are provided in Table 1.

Positional embedding We use Rotary Positional Embedding (RoPE) ([Su et al., 2021](#)) to inject positional information into our model. RoPE is a widely adopted method recently used by many

²<https://huggingface.co/datasets/cerebras/SlimPajama-627B>

Table 1: The details of model architecture.

Hidden size	Intermediate Hidden Size	Context Len	Heads	Layers	Vocab size
2,048	5,632	2,048	32	22	32,000

mainstream large language models, such as PaLM (Anil et al., 2023), Llama (Touvron et al., 2023a,b), and Qwen (Bai et al., 2023).

Pre-norm and RMSNorm Following Llama (Touvron et al., 2023a,b), we adopt pre-norm, i.e., normalize the inputs of each sub-layer in Transformer, instead of post-norm to stabilize the training process. Additionally, the RMSNorm (Zhang and Sennrich, 2019) normalization function is applied to improve the training efficiency.

SwiGLU Instead of using the traditional ReLU activation function, we follow Llama (Touvron et al., 2023a,b) and adopt SwiGLU (Shazeer, 2020), i.e., the combination of the Swish activation function and the Gated Linear Units (GLU) (Dauphin et al., 2017).

Grouped-query Attention To reduce memory bandwidth overhead and speed up inference, we use grouped-query attention (Ainslie et al., 2023) in our model. We have 32 heads for query attention and use 4 groups of key-value heads. With this technique, the model can share key and value representations across multiple heads without sacrificing much performance.

2.3 Speed Optimization

Fully Sharded Data Parallel (FSDP) During training, our codebase has integrated FSDP³ to leverage multi-GPU and multi-node setups efficiently. This integration is crucial in scaling the training process across multiple computing nodes, which significantly improves the training speed and efficiency.

FlashAttention Another critical improvement is the integration of FlashAttention-2 (Dao, 2023), an optimized attention mechanism. The repository also provides fused layernorm, fused cross entropy loss, and fused rotary positional embedding, which together play a pivotal role in boosting computational throughput.

xFormers We have replaced the original SwiGLU module with the fused SwiGLU module from the xFormers (Lefaudeux et al., 2022) repository, further enhancing the efficiency of our codebase. With these features, we can reduce the memory footprint, enabling larger batch size during 1.1B model training.

Speed Comparison with Existing Models After implementing these speedup modules, we achieved a training throughput of 24,000 tokens per second per A100-40G GPU. We compare our training speed with existing models, Pythia-1.0B (Biderman et al., 2023) and MPT-1.3B⁴, in terms of the GPU hours required to train 300 billion tokens, as detailed in Table 2. The results indicate that our codebase significantly enhances training efficiency compared to these models.

Table 2: Comparison of our training speed with Pythia-1.0B and MPT-1.3B.

	GPU Hours
Pythia-1.0B	4,830
MPT-1.3B	7,920
TinyLlama	3,456

2.4 Training

We build our framework based on lit-gpt (Lightning-AI, 2023). In adhering to Llama 2 (Touvron et al., 2023b), we employ an autoregressive language modeling objective during the pretraining phase. Consistent with Llama 2’s settings, we utilize the AdamW optimizer (Loshchilov and Hutter, 2019), setting β_1 at 0.9 and β_2 at 0.95. Additionally, we use a cosine learning rate schedule with a maximum learning rate of 4.0×10^{-4} and a minimum learning rate of 4.0×10^{-5} . We use 2,000 warmup steps to facilitate optimized learning. We set the batch size as 2M tokens. We assign weight decay as 0.1 and use a gradient clipping threshold of 1.0 to regulate the gradient value. We pretrain TinyLlama with 16 A100-40G GPUs in our project.

³https://huggingface.co/docs/accelerate/usage_guides/fsdp

⁴<https://huggingface.co/mosaicml/mpt-1b-redpajama-200b>

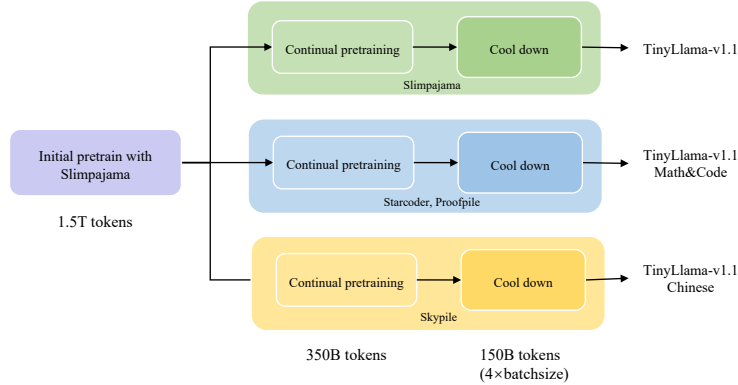


Figure 1: The pre-training stages and specialization pipeline of TinyLlama v1.1.

2.5 Version 1.1

During the open, live phase of pre-training the first version of TinyLlama, a few implementation issues were identified within the training framework. For example, there were a few issues related to the learning rate scheduler⁵ and data loading processes⁶. We therefore re-trained a new model from scratch after fixing those issues. The new model is named **TinyLlama v1.1**. In addition to fixing all those implementation issues, we took the opportunity to incorporate several key modifications in TinyLlama v1.1:

- To reduce communication overhead, we only shard the model parameters within a node in FSDP. We trained with a cluster consisting of 16 nodes, each equipped with four A100-40G GPUs, and set the batch size to approximately 1.8 million tokens.
- We reduced the total number of pre-training tokens from 3 trillion to 2 trillion. Despite the reduction in training data volume, a marginal improvement in performance on downstream tasks was observed, as compared to the original TinyLlama (refer to Section 3).
- Expanding beyond the singular pre-training phase of the original model, we introduced a three-stage pre-training process inspired by recent research (Wei et al., 2023). This includes basic pre-training, continual pre-training targeted at specific domains, and a cooldown phase. An illustrative overview of this approach is provided in Figure 1, with detailed discussions to follow in subsequent sections.

Basic pre-training In the first stage of the pre-training, we trained our model with only SlimPajama (Soboleva et al., 2023) to develop its commonsense reasoning capabilities. We only trained 1.5 trillion tokens during this stage, setting the foundation for more specialized training in subsequent stages.

Continual pre-training with specific domain During this phase, we diversified the training by incorporating three distinct types of corpora. The first corpus, identical to the basic pre-training stage, solely consisted of SlimPajama data (Soboleva et al., 2023). The second corpus combined code and mathematical content, leveraging integrations with Starcoder (Li et al., 2023) and Proof Pile 2 (Azerbayev et al., 2023). For Starcoder, we only considered the “Python” and “Jupyter” splits of the original Starcoder dataset. The third corpus focused on Chinese language data, utilizing Skypile (Wei et al., 2023). This strategic corpus diversity facilitated the development of three main TinyLlama v1.1 model variants, each tailored to a different need:

- **TinyLlama v1.1**: A foundational model for general applications.
- **TinyLlama v1.1 - Math&Code**: Enhanced specifically for mathematical and coding tasks.
- **TinyLlama v1.1 - Chinese**: Specialized for processing and understanding Chinese text.

⁵<https://github.com/jzhang38/TinyLlama/issues/27>

⁶<https://github.com/jzhang38/TinyLlama/issues/67>

In this stage, all three variants are trained with 350 billion tokens. For the Math&Code and Chinese variants, we linearly increase the sampling proportion of the domain-specific corpus for the beginning 6 billion tokens. This warmup sampling increasing strategy was designed to gradually adjust the pre-training data distribution, aiming to ensure smoother and more stable training. Following this initial phase of adaptive sampling, we maintained a consistent sampling strategy for the remainder of the training until approximately 1.85 trillion tokens. Detailed of the data sampling ratios are provided in Appendix A.

Cooldown Implementing a cooldown phase is essential for enhancing model convergence towards the end of the pre-training process. Traditionally, this is achieved by modifying the learning rate, as seen in approaches like MiniCPM (Hu et al., 2024) and DeepSeek LLMs (Bi et al., 2024). However, due to the use of a cosine schedule for our model, the learning rate is already low at the later stage of training. Hence, we opted to modify the batch size instead. Specifically, during the cooldown stage, the batch size was increased from 1.8 million tokens to 7.2 million tokens, while maintaining the original cosine learning rate schedule. This adjustment was applied uniformly across all variants, with each undergoing an additional 150 billion tokens of training during this phase. The training curves for all three variants are shown in Figure 2.

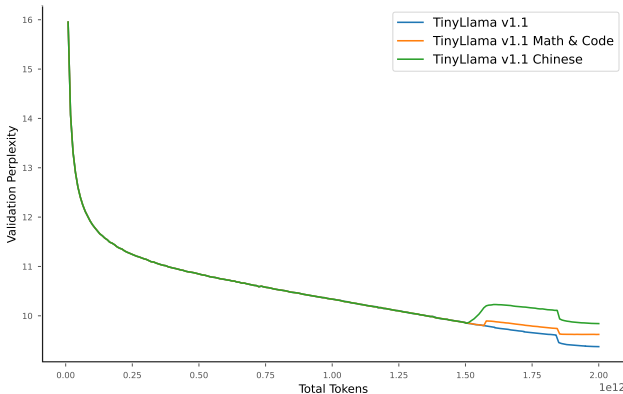


Figure 2: Training Loss for TinyLlama v1.1 models.

3 Results

We evaluate TinyLlama on a wide range of commonsense reasoning and problem-solving tasks and compare it with several existing open-source language models with similar model parameters.

Baseline models We primarily focus on language models with a decoder-only architecture, comprising approximately 1 billion parameters. Specifically, we compare TinyLlama with OPT-1.3B (Zhang et al., 2022), Pythia-1.0B, and Pythia-1.4B (Biderman et al., 2023).

Commonsense reasoning tasks To understand the commonsense reasoning ability of TinyLlama, we consider the following tasks: Hellaswag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), WinoGrande (Sakaguchi et al., 2021), ARC-Easy and ARC-Challenge (Clark et al., 2018), BoolQ (Clark et al., 2019), and PIQA (Bisk et al., 2020). We adopt the Language Model Evaluation Harness framework (Gao et al., 2023) to evaluate the models. Following previous practice (Biderman et al., 2023), the models are evaluated in a zero-shot setting on these tasks. The results are presented in Table 3. We notice that all TinyLlama models outperform baselines on many of the tasks and obtain the highest averaged scores. As for the 3 TinyLlama v1.1 models, we found that the model can achieve the best hellaswag performance while only training on Slimpajama. It is interesting to observe that for the Chinese model, even though the continual pretraining stage involves 50% Chinese data, the model still maintains good performance in the English benchmark.

Problem-solving tasks We also evaluate TinyLlama’s problem-solving capabilities using the InstructEval benchmark (Chia et al., 2023). This benchmark includes the following tasks:

- Massive Multitask Language Understanding (MMLU) (Hendrycks et al., 2021): This task is used to measure a model’s world knowledge and problem-solving capabilities across various subjects. We evaluate the models in a 5-shot setting.

Table 3: Zero-shot performance on commonsense reasoning tasks.

	HellaSwag	Obqa	WinoGrande	ARC-c	ARC-e	boolq	piqa	Avg
OPT-1.3B	53.65	33.40	59.59	29.44	50.80	60.83	72.36	51.44
Pythia-1.0B	47.16	31.40	53.43	27.05	48.99	57.83	69.21	48.30
Pythia-1.4B	52.01	33.20	57.38	28.50	54.00	63.27	70.95	51.33
TinyLlama v1.0	59.20	36.00	59.12	30.12	55.25	57.83	73.29	52.99
TinyLlama v1.1	61.47	36.80	59.43	32.68	55.47	55.99	73.56	53.63
TinyLlama v1.1 Math&code	60.80	36.40	60.22	33.87	55.20	57.09	72.69	53.75
TinyLlama v1.1 Chinese	58.23	35.20	59.27	31.40	55.35	61.41	73.01	53.41

- BIG-Bench Hard (BBH) (Suzgun et al., 2023): This is a subset of 23 challenging tasks from the BIG-Bench benchmark (Srivastava et al., 2022) designed to measure a language model’s abilities in complex instruction following. The models are evaluated in a 3-shot setting.
- Discrete Reasoning Over Paragraphs (DROP) (Dua et al., 2019): This reading comprehension task measures a model’s math reasoning abilities. We evaluate the models in a 3-shot setting.
- HumanEval (Zheng et al., 2023): This task is used to measure a model’s programming capabilities. The models are evaluated in a zero-shot setting.

The evaluation results are presented in Table 4. We observe that TinyLlama demonstrates better problem-solving skills compared to existing models. Moreover, after pretraining with a corpus containing more code and math data, TinyLlama v1.1 Math&Code can quickly obtain relevant ability and have a significant improvement in HumanEval and Drop compared to the other models, even though this model did not consume lots of code and math corpus at first 1.5T tokens.

Table 4: Performance of problem-solving tasks on the InstructEval Benchmark.

	MMLU	BBH	HumanEval	DROP	Avg.
	5-shot	3-shot	0-shot	3-shot	
OPT-1.3B	24.90	28.57	0.00	14.32	16.95
Pythia-1.0B	25.70	28.19	1.83	4.25	14.99
Pythia-1.4B	25.41	29.01	4.27	12.27	17.72
TinyLlama v1.0	25.34	29.65	9.15	15.34	19.87
TinyLlama v1.1	26.58	29.27	6.71	15.21	19.44
TinyLlama v1.1 Math&Code	24.60	26.35	15.24	18.54	21.18
TinyLlama v1.1 Chinese	26.35	30.04	4.88	15.75	19.26

Evaluation for Chinese tasks To evaluate the Chinese understanding and reasoning ability of TinyLlama v1.1, we test our models with the following tasks: xwinograd (Emelin and Sennrich, 2021), xstorycloze (Lin et al., 2022), xnli (Conneau et al., 2018), and xcopa (Ponti et al., 2020). They are all multilingual multiple choice question answering tasks but we only consider Chinese in this evaluation exercise. We summarize the overall performance across a suite of benchmarks in Table 5. After continual pretraining on the Chinese corpus, we observe that TinyLlama v1.1 Chinese outperforms other models in most tasks. Surprisingly, we also find that TinyLlama v1.0 and TinyLlama v1.1 Math&Code can achieve a much better performance than the original TinyLlama v1.1. We initially hypothesize that involving more code data helps improving the multilingual abilities. Upon checking the starcoder and Proof Pile datasets, we discovered that approximately 3.5% of the Python corpus and 4% of the Jupyter corpus consist of Chinese text. This may explain the improvement in the Chinese benchmark performance for TinyLlama v1.0 and TinyLlama v1.1 Math&Code.

Table 5: Zero-shot performance on Chinese understanding tasks.

	xwinograd_zh	xstorycloze_zh	xnli_zh	xcopa_zh	Avg
OPT-1.3B	54.96	48.38	33.45	53.00	47.45
Pythia-1.0B	60.31	48.97	33.77	55.20	49.56
Pythia-1.4B	60.51	50.56	35.38	53.00	49.86
TinyLlama v1.0	68.45	54.53	33.69	56.80	53.37
TinyLlama v1.1	55.75	47.98	34.70	51.40	47.46
TinyLlama v1.1 Math&Code	63.69	56.32	33.57	57.00	52.64
TinyLlama v1.1 Chinese	74.20	60.29	33.77	65.20	58.37

4 Conclusion

In this paper, we introduce TinyLlama, an open-source, small-scale language model. With its compact architecture and promising performance, TinyLlama can enable end-user applications on mobile devices, and serve as a lightweight platform for testing a wide range of innovative ideas related to language models. We also additionally verify the effectiveness of multi-stage training and data schedule through our v1.1 series. To promote transparency in the open-source language model pre-training community, we have released all relevant information, including our pre-training code, intermediate model checkpoints, and the details of our data processing steps. We hope that TinyLlama will make a valuable contribution to the community as we aim to democratize language model research and provide useful insights for future research in this field.

Acknowledgements

We express our gratitude to the open-source community for their strong support during the open, live phase of our research. Special thanks go to Qian Liu, Longxu Dou, Hai Leong Chieu, and Larry Law for their help to our project. We would like to thank the National Supercomputing Centre (NSCC) Singapore for their support in training our model v1.1. This research/project is supported by Ministry of Education, Singapore, under its Academic Research Fund (AcRF) Tier 2 Programme (MOE AcRF Tier 2 Award No.: MOE-T2EP20122-0011), Ministry of Education, Singapore, under its Tier 3 Programme (The Award No.: MOET320200004), the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Program (AISG Award No: AISG2-RP-2020-016), an AI Singapore PhD Scholarship (AISG Award No: AISG2-PhD-2021-08-007), an SUTD Kick-Starter Project (SKI 2021_03_11), and the grant RS-INSUR-00027-E0901-S00. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the funding agencies.

References

- Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebron, F., and Sanghai, S. (2023). GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of EMNLP*.
- Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., Chu, E., Clark, J. H., Shafey, L. E., Huang, Y., Meier-Hellstern, K., Mishra, G., Moreira, E., Omernick, M., Robinson, K., Ruder, S., Tay, Y., Xiao, K., Xu, Y., Zhang, Y., Abrego, G. H., Ahn, J., Austin, J., Barham, P., Botha, J., Bradbury, J., Brahma, S., Brooks, K., Catasta, M., Cheng, Y., Cherry, C., Choquette-Choo, C. A., Chowdhery, A., Crepy, C., Dave, S., Dehghani, M., Dev, S., Devlin, J., Díaz, M., Du, N., Dyer, E., Feinberg, V., Feng, F., Fienber, V., Freitag, M., Garcia, X., Gehrman, S., Gonzalez, L., Gur-Ari, G., Hand, S., Hashemi, H., Hou, L., Howland, J., Hu, A., Hui, J., Hurwitz, J., Isard, M., Ittycheriah, A., Jagielski, M., Jia, W., Kenealy, K., Krikun, M., Kudugunta, S., Lan, C., Lee, K., Lee, B., Li, E., Li, M., Li, W., Li, Y., Li, J., Lim, H., Lin, H., Liu, Z., Liu, F., Maggioni, M., Mahendru, A., Maynez, J., Misra, V., Moussalem, M., Nado, Z., Nham, J., Ni, E., Nystrom, A., Parrish, A., Pellat, M., Polacek, M., Polozov, A., Pope, R., Qiao, S., Reif, E., Richter, B., Riley, P., Ros, A. C., Roy, A., Saeta, B., Samuel, R., Shelby, R., Slone, A., Smilkov, D., So, D. R., Sohn, D., Tokumine, S., Valter, D., Vasudevan, V., Vodrahalli, K., Wang, X., Wang, P., Wang, Z., Wang, T., Wieting, J., Wu, Y., Xu, K., Xu, Y., Xue, L., Yin, P., Yu, J., Zhang, Q., Zheng, S., Zheng, C., Zhou, W., Zhou, D., Petrov, S., and Wu, Y. (2023). Palm 2 technical report.
- Azerbayev, Z., Schoelkopf, H., Paster, K., Santos, M. D., McAleer, S., Jiang, A. Q., Deng, J., Biderman, S., and Welleck, S. (2023). Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*.
- 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. (2023). Qwen technical report.

- Bi, X., Chen, D., Chen, G., Chen, S., Dai, D., Deng, C., Ding, H., Dong, K., Du, Q., Fu, Z., et al. (2024). Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*.
- Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O’Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., et al. (2023). Pythia: A suite for analyzing large language models across training and scaling. In *Proceedings of ICML*.
- Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. (2020). Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of AAAI*.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *Proceedings of NeurIPS*.
- Chia, Y. K., Hong, P., Bing, L., and Poria, S. (2023). INSTRUCTEVAL: towards holistic evaluation of instruction-tuned large language models. *CoRR*, abs/2306.04757.
- Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2022). Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.
- Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. (2019). BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of NAACL*.
- Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. (2018). Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.
- Conneau, A., Rinott, R., Lample, G., Williams, A., Bowman, S., Schwenk, H., and Stoyanov, V. (2018). XNLI: Evaluating cross-lingual sentence representations. In Riloff, E., Chiang, D., Hockenmaier, J., and Tsujii, J., editors, *Proceedings of EMNLP*, pages 2475–2485, Brussels, Belgium. Association for Computational Linguistics.
- Dao, T. (2023). Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.
- Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *Proceedings of ICML*.
- Dua, D., Wang, Y., Dasigi, P., Stanovsky, G., Singh, S., and Gardner, M. (2019). DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of NAACL*.
- Emelin, D. and Sennrich, R. (2021). Wino-X: Multilingual Winograd schemas for commonsense reasoning and coreference resolution. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t., editors, *Proceedings of EMNLP*, pages 8517–8532, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac’h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. (2023). A framework for few-shot language model evaluation.
- Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. (2021). Measuring massive multitask language understanding. In *Proceedings of ICLR*.
- Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Vinyals, O., Rae, J. W., and Sifre, L. (2022). Training compute-optimal large language models. In *Proceedings of NeurIPS*.

- Hu, S., Tu, Y., Han, X., He, C., Cui, G., Long, X., Zheng, Z., Fang, Y., Huang, Y., Zhao, W., et al. (2024). Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*.
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.
- Lefaudeux, B., Massa, F., Liskovich, D., Xiong, W., Caggiano, V., Naren, S., Xu, M., Hu, J., Tintore, M., Zhang, S., Labatut, P., and Haziza, D. (2022). xformers: A modular and hackable transformer modelling library. <https://github.com/facebookresearch/xformers>.
- Li, R., allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., LI, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Lamy-Poirier, J., Monteiro, J., Gontier, N., Yee, M.-H., Umapathi, L. K., Zhu, J., Lipkin, B., Oblokulov, M., Wang, Z., Murthy, R., Stillerman, J. T., Patel, S. S., Abulkhanov, D., Zocca, M., Dey, M., Zhang, Z., Bhattacharyya, U., Yu, W., Luccioni, S., Villegas, P., Zhdanov, F., Lee, T., Timor, N., Ding, J., Schlesinger, C. S., Schoelkopf, H., Ebert, J., Dao, T., Mishra, M., Gu, A., Anderson, C. J., Dolan-Gavitt, B., Contractor, D., Reddy, S., Fried, D., Bahdanau, D., Jernite, Y., Ferrandis, C. M., Hughes, S., Wolf, T., Guha, A., Werra, L. V., and de Vries, H. (2023). Starcoder: may the source be with you! *Transactions on Machine Learning Research*.
- Lightning-AI (2023). Lit-gpt.
- Lin, X. V., Mihaylov, T., Artetxe, M., Wang, T., Chen, S., Simig, D., Ott, M., Goyal, N., Bhosale, S., Du, J., Pasunuru, R., Shleifer, S., Koura, P. S., Chaudhary, V., O’Horo, B., Wang, J., Zettlemoyer, L., Kozareva, Z., Diab, M., Stoyanov, V., and Li, X. (2022). Few-shot learning with multilingual generative language models. In Goldberg, Y., Kozareva, Z., and Zhang, Y., editors, *Proceedings of EMNLP*, pages 9019–9052, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.
- Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *Proceedings of ICLR*.
- Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. (2018). Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of EMNLP*.
- OpenAI (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Ponti, E. M., Glavaš, G., Majewska, O., Liu, Q., Vulić, I., and Korhonen, A. (2020). XCOPA: A multilingual dataset for causal commonsense reasoning. In Webber, B., Cohn, T., He, Y., and Liu, Y., editors, *Proceedings of EMNLP*, pages 2362–2376, Online. Association for Computational Linguistics.
- Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. (2021). Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.
- Shazeer, N. (2020). GLU variants improve transformer. *CoRR*, abs/2002.05202.
- Soboleva, D., Al-Khateeb, F., Myers, R., Steeves, J. R., Hestness, J., and Dey, N. (2023). SlimPajama: A 627B token cleaned and deduplicated version of RedPajama.
- Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., Brown, A. R., Santoro, A., Gupta, A., Garriga-Alonso, A., et al. (2022). Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.
- Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. (2021). Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*.
- Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q., Chi, E., Zhou, D., and Wei, J. (2023). Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of ACL*.
- Thaddée, Y. T. (2023). Chinchilla’s death. <https://espadrine.github.io/blog/posts/chinchilla-s-death.html>.

- Together Computer (2023). Redpajama: an open dataset for training large language models.
- Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. (2023a). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. (2023b). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Proceedings of NeurIPS*.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., and Zhou, D. (2022). Chain of thought prompting elicits reasoning in large language models. In *Proceedings of NeurIPS*.
- Wei, T., Zhao, L., Zhang, L., Zhu, B., Wang, L., Yang, H., Li, B., Cheng, C., Lü, W., Hu, R., et al. (2023). Skywork: A more open bilingual foundation model. *arXiv preprint arXiv:2310.19341*.
- Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. (2019). HellaSwag: Can a machine really finish your sentence? In *Proceedings of the ACL*.
- Zhang, B. and Sennrich, R. (2019). Root mean square layer normalization. In *Proceedings of NeurIPS*.
- Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. (2022). Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.
- Zheng, Q., Xia, X., Zou, X., Dong, Y., Wang, S., Xue, Y., Shen, L., Wang, Z., Wang, A., Li, Y., Su, T., Yang, Z., and Tang, J. (2023). Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, pages 5673–5684. ACM.

Based on this observation, we reduced the number of tokens to 2T in total in the second run. Instead of training a single base model, we first pretrained the model for 1.5T tokens and then continually pretrained it with different types of corpora. This resulted in three variants, TinyLlama v1.1, TinyLlama v1.1 Math&Code and TinyLlama v1.1 Chinese.

A Data sampling ratio for TinyLlama v1.1

The data sampling ratio for TinyLlama v1.1 Math&Code and Chinese model are shown in Table 6 and 7.

Table 6: Data sampling ratio for TinyLlama-Math&Code

	Basic pretraining	Continual pretraining with specific domain	Cooldown
Slimpajama	100.0%	75.0%	75.0%
Starcode	-	15.0%	15.0%
Proof pile	-	10.0%	10.0%

Table 7: Data sampling ratio for TinyLlama-v1.1-Chinese

	Basic pretraining	Continual pretraining with specific domain	Cooldown
Slimpajama	100.0%	50.0%	50.0%
Skypile	-	50.0%	50.0%