SUPER TINY LANGUAGE MODELS

Leon Guertler
A*STAR and Nanyang Technological University
leon00@e.ntu.edu.sg

Dylan Hillier
A*STAR and Singapore Management University
das.hillier.2023@phdcs.smu.edu.sg

Palaash Agrawal, Chen Ruirui, Bobby Cheng, Cheston Tan
Centre for Frontier AI Research (CFAR), Institute of High Performance Computing (IHPC), A*STAR
{agrawal_palaash@cfar, chen_ruirui@ihpc, bobby_cheng@i2r, cheston_tan@cfar}.a-star.edu.sg

ABSTRACT

The rapid advancement of large language models (LLMs) has led to significant improvements in natural language processing but also poses challenges due to their high computational and energy demands. This paper introduces a series of research efforts focused on Super Tiny Language Models (STLMs), which aim to deliver high performance with significantly reduced parameter counts. We explore innovative techniques such as byte-level tokenization with a pooling mechanism, weight tying, and efficient training strategies. These methods collectively reduce the parameter count by 90% to 95% compared to traditional models while maintaining competitive performance. This series of papers will explore into various subproblems, including tokenizer-free models, self-play based training, and alternative training objectives, targeting models with 10M, 50M, and 100M parameters. Our ultimate goal is to make high-performance language models more accessible and practical for a wide range of applications.

1 Introduction

The recent rise in popularity of large language models (LLMs) has mostly been fueled by the invention of the attention based autoregressive transformer [1, 2]. These models are trained to predict the next token (a subword unit) on very large corpora of data. There exists an extensive body of research on how scaling these architectures to more parameters and more data, predictably, improves model performance [3, 4, 5]. Thus, in practice, this is the recipe for performance improvement most companies (notably OpenAI, Google DeepMind, Anthropic) follow. However, with ever larger models, a number of problems arise. Firstly, the state-of-the-art models have been scaled to an extent where it is impossible for academic researchers to train competitive models, making safety and capability research harder. Secondly, training models of this magnitude requires so much compute and energy that industry players plan to build nuclear power plants, solely for the purpose of training models [6, 7]. Lastly, when models are large, both inference time and applications that require running models on edge devices suffer.

Small language models on the other hand, on the order of 1 billion parameters, have been trained with increasingly impressive performance. This includes models like TinyLlama [8] (1.1B), Phi-3-mini [9] (3.3B), and MobiLlama [10](0.5B). While these models are able to reasonably compete with large foundation models, they still require thousands of GPU hours to train, which still puts them out of the reach of many researchers, and prevents fast experimentation.

Rather than focusing on recreating foundation models at a smaller scale, in this series of papers, we aim to use the small model size as a test bed for an open exploration of effective methods for improving parameter and sample efficiency. Specifically, we will focus on methods related to tokenizer-free models, weight tying, self-play based training, alternative training objectives, data sampling techniques,

2 Related Works

We provide basic information about transformers, especially as decomposed in our repository in Appendix A In Section 2.1 we cover some of the works on parameter reduction that inform our research

2.1 Parameter Reduction

In the quest to make language models more efficient and accessible, various parameter reduction techniques have been developed. These techniques aim to reduce the number of parameters in the model without significantly compromising performance. Below, we discuss some of the most popular methods, including weight tying, pruning, quantization, and knowledge distillation. We note that many of these techniques are primarily used in a post-hoc fashion and may not be applicable during model training.

Weight Tying Weight tying is a technique where certain weights in the model are shared between different components. This approach not only reduces the total number of parameters but also ensures that certain parts of the model are better aligned. There are different types of weight tying used in various models:

- **Embedding & Head** In GPT-2 and other similar models, the embedding matrix is tied to the weights of the output layer, ensuring that the output probabilites are directly related to the input embeddings
- **FFN sharing** MobiLlama [10] shares weights specifically between the feed-forward network (FFN) layers. By doing so, it achieves parameter efficiency without compromising on the model's ability to learn and generalize.
- **FFN+Attn sharing** ALBERT [11] employs weight tying extensively by sharing parameters across all layers of the transformer. It ties the weights of both the feed-forward network (FFN) and the attention layers, which significantly reduces the model size while maintaining performance.[12].

Pruning Pruning involves removing weights that contribute least to the model's performance. This can be done during or after training. Pruning results in a sparser model with fewer parameters and reduced computational requirements. This is inspired by the *lottery ticket hypothesis* which states that there exists a smaller subnetwork (a "winning ticket") that, when trained in isolation, can achieve performance comparable to the original model [13]. Pruning methods inspired by this hypothesis identify and retain only the most critical parameters.

Quantization Quantization reduces the precision of the model's weights and activations from 32-bit floating-point numbers to lower-bit representations such as 8-bit integers. This technique significantly reduces model size (if not parameter count) and often speeds up training/inference with minimal impact on performance.[14]

Low-Rank Factorization This technique decomposes large weight matrices into products of smaller matrices, which reduces the number of parameters and the computational cost. This has been used in e.g. Ma et al.[15] for compressing a pretrained BERT model

2.2 Data Quality and Training Efficiency

In place of directly reducing the number of parameters, other approaches focus on improving the quality of the training signal, and thereby enabling the use of fewer parameters. Below, we cover data selection and knowledge distillation.

Data Selection A key argument of the Phi series of language models[9, 16, 17] is that by improving the quality of data, the performance of small language models can be increased, far in excess of the predictions of e.g. scaling laws. In contrast to training on crawls of the internet, these models are trained on textbooks and heavily filtered web data, and are able to match performance with larger models trained on larger training sets. Some data augmentation and filtering techniques go further to use pretrained large language models as part of the process. For example Zhang et al. [18] utilise a pretrained language model to verify the quality of training data as a mathematic source during finetuning.

Knowledge Distillation Knowledge distillation transfers the knowledge from a larger "teacher" model to a smaller "student" model. In particular the idea is that rather than the model being trained on a single hard label (i.e. a particular token), the student model instead has the entire probability distribution of the teacher model to guide its learning. For example, DistilBERT [19] is a smaller version of BERT created using knowledge distillation which retains 97% of BERT's language understanding capabilities while being 60% faster and 40% smaller .

3 Motivated Statement of the Goals/ Measurements of Success

Whilst the end-goal is to create highly performant super tiny language models, we measure our progress towards that goal along multiple dimensions, including:

Model Size We propose no hard parameter count cut-off, but aim to build a family of models with 10M, 50M and 100M parameters. Most experiments will focus on models with close to 50M parameters, and once we have narrowed down an architecture, we will scale it up and down to 100M and 10M parameters respectively.

Training Time As mentioned in the introduction, one of the key motivators for this series of papers is to make research more accessible. For this to work, training time on consumer architecture needs to be short enough to allow for experimentation. Thus, for the 50M model, the overall training time should be less than 48H on a single consumer GPU.

Model Performance To ensure that our STLMs achieve competitive results despite their reduced size, we will evaluate their performance across a comprehensive set of benchmarks and metrics commonly used for evaluating much larger models. This has the added benefit of making them easier to benchmark against their larger counterparts. Specifically, we aim to achieve competitive performance to models in the size range 3B-7B parameters on GSM8K [20], MMLU [21], and LMSYS Chatbot Arena [22].

4 Proposed Approach

4.1 Technical

Our technical contribution is a research repository in which we aim to enable other researchers to easily run experiments on small models by surfacing clean, accessible interfaces, with interpretable and understandable model code. This is implemented in PyTorch [23], with Hydra for config management, and logging in Weights and Biases. The repository is initially based off of Karpathy's minGPT repository [24], but we also implement the so-called "modern transformer" [25] used in e.g. Llama 2 [26]. As such the core components exposed in our library consist of:

- GPT2/BPE Tokenizers
- SWIGlu/ Standard FFN layers
- Attention supporting RoPE embeddings, Causal Masks, Grouped Query Attention

4.1.1 Training Data

To simplify the learning task/computational load, our primary training data set for pretraining is currently simple-english-Wikipedia[1]. This is a roughly 370Mb dataset consisting in articles written with plainer language than the full Wikipedia text.[2] Interestingly this consists in around 34 million words. The small size of the training data allows it to be easily run on consumer hardware with limited memory, nevertheless it also means that the models may not be sample efficient enough to reach a fluent level within 100 million words in contrast to humans who are estimated to acquire fluency at this level [27]. Altering this training data mix is accordingly of early interest to us.

4.1.2 Evaluation

The usual measure of language modelling performance for a model with probability distribution P, sequence $\mathbf{x}$ is given by the perplexity:

$$\log_2(\text{Perplexity}) = -\frac{1}{N}\sum_{i=1}^{N}\log_2(P(x_i|\mathbf{x}_{:i}))$$

One intuitive way of thinking of the perplexity is to measure how many guesses are required on average per prediction step, the issue is that for different tokenizers, there will be different sized prediction steps. In this project, since we aim to enable experiments across different tokenizers, we need to equalize this measure by using the original string length, rather than the length of the input string. Perplexity is evaluated on a held out Wikipedia test set.

In addition we initially use the following as question answering datasets for evaluating performance:

[1] Downloaded from `https://huggingface.co/datasets/wikipedia`

[2] `https://simple.wikipedia.org/wiki/Simple_English_Wikipedia`

- ARC [28] - This tests the reasoning of LLMs
- Winogrande [29] - The famous commonsense benchmark
- Hellaswag [30] - This tests the ability of the model to distinguish the plausibility of continuations of a story.
- MMLU [21]

These are all multiple choice QA datasets. We use the path probabilities over the different options to measure which option is "preferred" by the model, since simple Wikipedia does not include much question answering this is intended to better measure performance disparities of these models.

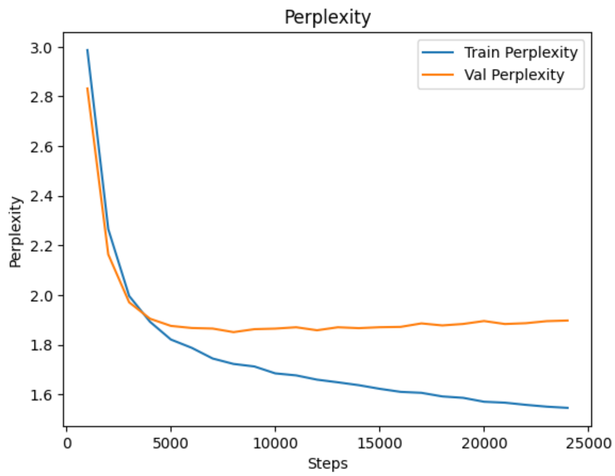

Figure 1: Perplexity during training for the baseline model

Model	
number of layers	10
ffn type	SWIGlu
ffn dimension	1536
number of attention heads	16
tokenizer type	gpt2
hidden dim	512
max context window	512
vocab_size	50257
Training	
batch_size	24
gradient accumulation_steps	20
total iterations	25000
warmup iterations	5000
dropout	0.1
Optimizer	
lr	0.0006
weight_decay	0.1
Miscellaneous	
GPU	H100
GPU Hours	10

Table 1: The Paramaters used for our baseline

Model	Model Size	arc_easy	winogrande	hellaswag	mmlu
TinyLlama [8]	1B	55.3%	59.1%	59.2%	25.7%[3]
MobiLlama [10][4]	500M	–	57.5%	52.5%	26.5%
Phi 3[9][5]	3B	94.6%	70.8%	76.7%	68.8%
Baseline	50M	21 %	50.7%	25.6%	25.3%
Metric	MB	Accuracy	Accuracy	Accuracy	Accuracy
Choices	–	4	2	4	4
Chance	0	25%	50%	25%	25%

Table 2: Initial Results on Chosen Benchmarks

4.2 Benchmarking

We run a benchmark using a 10-layer Llama-style "Modern Transformer" as a baseline benchmark. For full details see Table 1. We note that we do not use checkpointing here, but will in the future. The training curves in Figure 1 demonstrate that the model is clearly overfitting on this small dataset. This is somewhat unsurprising given that during this training process the model sees on the order of 3 billion words - in other words the model sees each word in the training dataset 10 times. Note that to avoid significant over-fitting, a model of this size would require roughly a dataset of roughly 1 billion words according to the Kaplan Scaling laws [3]. Additionally compare this to the roughly 100 million that a 12 year old is estimated to have seen [27]. In Table 2 we show the initial results on the benchmarks described in Section 4.1.2 which are around the level of chance. Clearly there is significant room for improvement.

4.3 Research

In order to increase the use of our research we plan on committing to releasing reports regardless of the outcome, and release negative results alongside positive ones.

5 Proposed Research Projects

In line with the research goals described above we outline a number of research projects that we aim to explore in the coming months.

5.1 Weight Tying

As introduced in section 5.1, weight tying is a commonly used method of parameter reduction. Recent methods for creating tiny language models including MobiLlama[10] have gone further to share the weights between the feed forward layers. Since the majority of transformer parameters are stored in these layers this makes sense as a target for parameter reduction, additionally recent research has shown that many of these layers are redundant[31]. We currently propose to explore different styles of feedforward weight sharing, including using LoRA adaptors[32] to allow for parameter efficient adaptation of the computation performed by these layers.

5.2 Byte-level / tokenizer free

One of the most parameter expensive components of the modern day transformer architecture are the embedding layer and the next token head. The reason for the high parameter count in these two layers is the number of unique tokens produced by the tokenizer. The size of the this embedding layer is vocabulary size * embedding dimension, where the vocabulary size is typically on the order of 50000 tokens (e.g. 50,257 for GPT2 [33]). This is then replicated in the LM head, which in some cases may not share the weights of the embedding layer e.g. in Press et al. [12]. As such for a GPT-2 sized model, the embedding and lm head layers can account for 45.20% to 62.25% of total parameter count depending on whether weight tying is used.

One solution to this is finding some sort of architecture that requires a significantly smaller vocabulary size (in the extreme this could be 256 for byte level models), however, as the vocabulary size shrinks, it becomes harder for the

[3]This is 5 shot
[4]All 5-shot plus
[5]These results are all 5 shot plus, pending replacement with zero shot versions

model to fit the data, and efficiency degrades (as the sequence lengths (in terms of token count) increase for the same input string). We propose to explore a byte-level tokenizer with a pooling mechanism. Specifically, the proposed method works as follows:

- **Byte-level Embedding** The input string is first embedded using a byte-level embedder. Each character/byte in the input string is converted into a byte representation, resulting in a sequence of byte embeddings. Since the vocabulary size is only 256, a very small embedding dimension can be used (64 for example), this step only requires a negligible number of trainable parameters compared to the total model.
- **Tokenization and Chunking** The byte-embedded sequence is then split into chunks using the bounding boxes of a standard Byte Pair Encoding (BPE) tokenizer. Importantly, this step requires no additional parameters as the tokenizer is only used to find the bounding boxes of tokens and not to embed the input.
- **Pooling Mechanism** Based on the bounding boxes obtained from the BPE tokenizer, the byte embeddings are chunked into tokens. Each chunk is then individually passed through a smaller two-layer transformer. The output of this transformer is pooled into a single token representation. At this point the dimensions of the hidden state are the exact same it would be if a standard BPE tokenizer were used, however, this method requires only a fraction (10%) or the parameters.
- **Core Model Processing** The pooled token representations are then passed through the core model. The dimension of the data going through the core model is the same as it would be if just a BPE tokenizer was used.
- **Decoding and Final Prediction** Instead of using a language model head, each BPE-level token is decoded back into bytes. The final loss is calculated as the next BPE token prediction but for each byte in the next BPE token. This is done via a similar mechanism as the byte-level encoder, and thus, also requires substantially fewer parameters.

Overall, compared to using a BPE tokenizer, the proposed Byte-level model uses between 90% and 95% fewer parameters, depending on whether weight-tying is used or not.

5.3 Early Exit and Conditional Computation

The success of techniques like Mixture of Experts e.g.[34] is thought to lie in the fact that different tokens may require different computation pathways. More generally we can expect that different tokens require different amounts of computation. As argued in the previous section, deeper layers of transformers may be redundant for a majority of inputs – we aim to exploit this by exploring methods that skip this computation for easy to predict tokens. In particular we will explore mixture of depths[35] and the recently proposed layerskip[36] methods.

5.4 Next thought prediction

Again building on the idea of conditional computation, we are interested in exploring methods that perform computation over sequences of thoughts rather than sequences of test - more generally decoupling the language modelling capabilities from the reasoning capabilities. Language models are of course primarily designed to model language, unfortunately this means that while they have impressive emergent downstream capabilities[33] and (arguably) various interesting properties[37] these firstly may not emerge at smaller model scales and furthermore may occur in spite of rather than because of the language modelling training paradigm. In particular per the arguments of LeCun et al.[38], it may be more efficient to perform predictions in a latent space that is not directly tied to the output sequence space. Indeed this is similar to the way that Chain-of-thought[39] augments a model's downstream capabilities by leveraging the models knowledge of reasoning rather than directly answering questions. Thus for example Goyal et al.[40] utilise pause tokens that do not correspond to tokens in the source text, and enable the model to perform additional computation before outputting an answer.

5.5 Dropout and Learning Rate Scheduling

We would like to explore the effect of using dropout. While most large models are not trained with dropout due to using sufficiently large datasets as to avoid over-fitting, Liu et al [41] proposes that scheduling dropout in the early phases of training can help reduce underfitting (which may be relevant when training on small amounts of high quality data) and if scheduled for the late phases is more effective at combating overfitting. Additionally most approaches, such as Llama [26] use Cosine Learning Rate Schedulers. Given the relatively fast training times it would be useful to verify the efficacy of this scheduler.

5.6 Curriculums, Data mixes, and Multimodality

As discussed in section 4.1.1, the quality of the training data can greatly impact the performance of the trained model. Additionally having training data that coverages a sufficient portion of use cases (i.e. not just factual wikipedia articles) is necessary for drawing reasonable inferences from the downstream performances of tiny llms. Measuring the quality and coverage of training data is difficult however, especially for foundation models where the downstream tasks may be unclear. One potential option is to use high quality datasets like the British National Corpus [42] with a sufficiently broad set of data sources, or to use a refined/ heavily filtered version of webcrawls in the manner of Penedo et al. [43]

6 Conclusion

In this introductory paper, we have outlined the vision and approach for developing Super Tiny Language Models (STLMs) aimed at achieving high performance with significantly reduced parameter counts. The growing computational and energy demands of large language models underscore the necessity of this research. Our proposed methods, including byte-level tokenization with pooling, weight tying, and efficient training strategies, have the potential to reduce the parameter count by 90% to 95% compared to traditional models while maintaining competitive performance.

We have detailed various techniques for parameter reduction, such as weight tying, pruning, quantization, and knowledge distillation, and discussed their relevance to our goals. Our technical approach involves creating a research repository that facilitates experimentation with small models, making cutting-edge NLP research more accessible.

The performance of our STLMs will be rigorously evaluated using standard benchmarks, aiming to match or exceed the capabilities of much larger models. This series of papers will delve into specific subproblems, including tokenizer-free models, self-play based training, and alternative training objectives, targeting models with 10M, 50M, and 100M parameters.

Ultimately, our goal is to democratize access to high-performance language models, enabling more researchers and practitioners to contribute to and benefit from advancements in NLP. We believe that the development of STLMs will pave the way for more sustainable and efficient AI, broadening the scope of applications and fostering innovation across various domains.

We invite the research community to engage with our work, explore the proposed techniques, and contribute to the ongoing effort to make language models more efficient and widely usable.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[3] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[4] Joel Hestness, Sharan Narang, Niki Ardalani, Greg Diamos, Heewoo Jun, Hessam Kianinejad, Mostofa Patwary, Mohammad Shoeybi Ali, Erich Yong, and Stephen Zhu. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

[5] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Aurelia Guy, Sarah Henderson, Katie Millican, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[6] Mark Zuckerberg Dwarkesh Patel. Mark zuckerbart - llama 3, open sourcing 10b models, & caesar augustus. Podcast, 2024. Accessed: 2024-05-19, URL: https://open.spotify.com/episode/6Lbsk4HtQZfkJ4dZjh7E7k?si=c31a37a19f994175.

[7] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Morgan Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

[8] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

[9] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[10] Omkar Thawakar, Ashmal Vayani, Salman Khan, Hisham Cholakal, Rao M Anwer, Michael Felsberg, Tim Baldwin, Eric P Xing, and Fahad Shahbaz Khan. Mobillama: Towards accurate and lightweight fully transparent gpt. *arXiv preprint arXiv:2402.16840*, 2024.

[11] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[12] Ofir Press and Lior Wolf. Using the output embedding to improve language models, 2017.

[13] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[14] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[15] X Ma, J Zhang, R Wang, Q Xu, and D Lin. Tensorized embedding layers for efficient model compression. *Advances in Neural Information Processing Systems*, 32, 2019.

[16] Mojan Javaheripi, Sébastien Bubeck, Marah Abdin, Jyoti Aneja, Sebastien Bubeck, Caio César Teodoro Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, Sivakanth Gopi, et al. Phi-2: The surprising power of small language models. *Microsoft Research Blog*, 2023.

[17] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio Cesar Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Conti Kauffmann, Gustavo Henrique de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Behl, Xin Wang, Sebastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. Textbooks are all you need, 2024.

[18] Yifan Zhang, Yifan Luo, Yang Yuan, and Andrew C Yao. Autonomous data selection with language models for mathematical texts. In *ICLR 2024 Workshop on Navigating and Addressing Data Problems for Foundation Models*, 2024.

[19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[20] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Alhussein Fawzi, Felix Chen, Mateusz Malinowski, Sanja Fidler, Oleg Klimov, Tim Vogels, Jacob Hilton, Jessica Han, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2106.07311*, 2021.

[21] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

[22] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating llms by human preference, 2024.

[23] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[24] Andrej Karpathy. mingpt. `https://github.com/karpathy/minGPT`, 2022. GitHub repository.

[25] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[26] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[27] Alex Warstadt, Aaron Mueller, Leshem Choshen, Ethan Wilcox, Chengxu Zhuang, Juan Ciro, Rafael Mosquera, Bhargavi Paranjabe, Adina Williams, Tal Linzen, et al. Findings of the babylm challenge: Sample-efficient pretraining on developmentally plausible corpora. In *Proceedings of the BabyLM Challenge at the 27th Conference on Computational Natural Language Learning*, 2023.

[28] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[29] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[30] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[31] Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*, 2024.

[32] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[33] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[34] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[35] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

[36] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

[37] Michal Kosinski. Theory of mind may have spontaneously emerged in large language models.

[38] Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62(1), 2022.

[39] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[40] Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*, 2023.

[41] Zhuang Liu, Zhiqiu Xu, Joseph Jin, Zhiqiang Shen, and Trevor Darrell. Dropout reduces underfitting. In *International Conference on Machine Learning*, pages 22233–22248. PMLR, 2023.

[42] Geoffrey Neil Leech. 100 million words of english: The british national corpus (bnc).

[43] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Hamza Alobeidli, Alessandro Cappelli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data only. *Advances in Neural Information Processing Systems*, 36, 2024.

[44] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[45] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152, 2012.

[46] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[47] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

A Overview of LLM Architectures

Large language models (LLMs) have become a cornerstone of modern natural language processing (NLP) due to their ability to understand and generate human-like text. At the heart of these models lies the transformer architecture, introduced by [1], which has since become the standard for NLP tasks. The transformer architecture is composed of several key components:

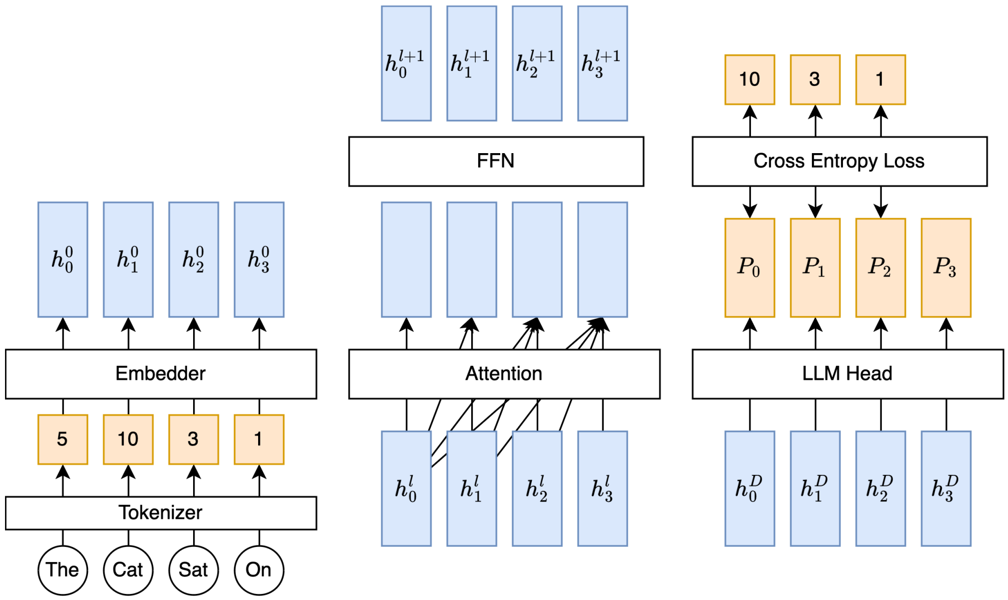

Figure 2: Diagram demonstrating flow of information through transformer components during training

A.1 Tokenization

The process begins with tokenization, where input text is split into smaller units called tokens. These tokens can be words, subwords, or even characters, depending on the chosen tokenizer. Common tokenization methods include Byte Pair Encoding (BPE) [44] and WordPiece [45], which help in handling the vast vocabulary of natural languages by breaking down rare words into more frequent subwords.

A.2 Embeddings

Embedding Lookup Tables After tokenization, the token indices are typically use to index an embedding matrix W_e of shape $N \times E$ where N is the vocab size and E is the embedding dimension.

Positional Embeddings Unless explicitly given, by default a transformer would have no way of distinguishing adjacent tokens from distant tokens. As such this information has to be encoded into the model's embeddings. In the original transformer paper[1], this was done in terms of Sine/Cosine embeddings, but other methods include learnt positional embeddings and more recently RoPE Embeddings[46] which modify the hidden states directly in the attention layers.

A.3 Transformer Blocks

The core of an LLM consists of multiple transformer blocks, each containing two main sub-layers: multi-head self-attention mechanisms and feed-forward neural networks (FFNs).

Self-Attention Mechanism This mechanism allows the model to weigh the importance of different tokens in the input sequence when making predictions. By considering the entire sequence, the model can capture long-range dependencies and contextual information effectively. The self-attention mechanism is computed as follows:

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

where Q, K, and V are the query, key, and value matrices repsectively and d_k is the dimension of the key vectors.

Feed-Forward Neural Networks (FFNs) Following the self-attention mechanism, each token's representation is passed through a feed-forward neural network. At least in the original transformer design, this sub-layer consists of two linear transformations with a non-linear activation function in between, further refining the token representations:

$$\text{FFN}(x) = \sigma(xW_1 + b_1)W_2 + b_2$$

where W_1 and W_2 are weight matrices, b_1 and b_2 are bias vectors, and σ is a non-linear activation function. More recent transformers like [26] forgo the bias and use the SwiGLU feed forward network[47] given by:

$$\text{SwiGLU}(x) = (\text{Swish}(xW) \otimes xV)W_2$$

Where $\otimes$ is the hadamard or elementwise product, W, V are are additional weight matrices, and $\text{Swish}(x) = \sigma(x) \cdot x$ where σ is the sigmoid function.

Next Token Prediction Head At the final stage of the transformer model, the refined token representations are used to predict the next token in the sequence. This is achieved by applying a linear transformation followed by a softmax function, which generates a probability distribution over the vocabulary, indicating the likelihood of each possible next token:

$$P(x_i|x_{<i}) = \text{softmax}(h_i W_e^T)$$

where h_i is the hidden state for the i-th token and W_e is the embedding matrix. The model is typically trained using the cross-entropy loss function, defined as:

$$\mathcal{L} = -\sum_i \log P(x_i|x_{<i})$$

These components work together to enable LLMs to perform a wide range of NLP tasks with high accuracy and fluency. However, the immense size of these models, often containing billions of parameters, poses significant challenges in terms of computational requirements and energy consumption.