OpenELM: An Efficient Language Model Family with Open Training and Inference Framework

Sachin Mehta [1] **Mohammad Hossein Sekhavat** [1] **Qingqing Cao** [1] **Maxwell Horton** [1] **Yanzi Jin** [1] **Chenfan Sun** [1] **Iman Mirzadeh** [1] **Mahyar Najibi** [1] **Dmitry Belenko** [1] **Peter Zatloukal** [1] **Mohammad Rastegari** [1]

Model	Public dataset	Open		Model size	Pre-training tokens	Average acc. (in %)
		Code	Weights			
OPT (Zhang et al., 2022)	✗	✓	✓	1.3 B	0.2 T	41.49
PyThia (Biderman et al., 2023)	✓	✓	✓	1.4 B	0.3 T	41.83
MobiLlama (Thawakar et al., 2024)	✓	✓	✓	1.3 B	1.3 T	43.55
OLMo (Groeneveld et al., 2024)	✓	✓	✓	1.2 B	3.0 T	43.57
OpenELM (Ours)	✓	✓	✓	1.1 B	1.5 T	**45.93**

Table 1. **OpenELM vs. public LLMs.** OpenELM outperforms comparable-sized existing LLMs pretrained on publicly available datasets. Notably, OpenELM outperforms the recent open LLM, OLMo, by 2.36% while requiring 2× fewer pre-training tokens. The average accuracy is calculated across multiple tasks listed in Table 7b, which are also part of the OpenLLM leaderboard (Beeching et al., 2023). Models pretrained with less data are highlighted in gray color.

Abstract

The reproducibility and transparency of large language models are crucial for advancing open research, ensuring the trustworthiness of results, and enabling investigations into data and model biases, as well as potential risks. To this end, we release OpenELM, a state-of-the-art open language model. OpenELM uses a layer-wise scaling strategy to efficiently allocate parameters within each layer of the model, leading to enhanced accuracy. For example, with a budget of around one billion parameters, OpenELM exhibits a 2.36% improvement in accuracy compared to OLMo while requiring 2× fewer pre-training tokens. Our source code along with pre-trained model weights and training recipes is available at https://github.com/apple/corenet. OpenELM HuggingFace models can be found at: https://huggingface.co/apple/OpenELM.

1. Introduction

Transformer-based (Vaswani et al., 2017) large language models (LLMs) are revolutionizing the field of natural language processing (Brown et al., 2020; Touvron et al., 2023). These models are isotropic, meaning that they have the same configuration (*e.g.*, number of heads and feed-forward network dimensions) for each transformer layer. Though such isotropic models are simple, they may not allocate parameters efficiently inside the model.

In this work, we develop and release OpenELM, a family of pre-trained and fine-tuned models on *publicly* available datasets. At the core of OpenELM lies layer-wise scaling (Mehta et al., 2020), enabling *more efficient parameter* allocation across layers. This method utilizes smaller latent dimensions in the attention and feed-forward modules of the transformer layers closer to the input, and gradually widening the layers as they approach the output.

Our release includes the complete framework, encompassing data preparation, training, fine-tuning, and evaluation procedures, alongside multiple pre-trained checkpoints and training logs, to facilitate open research. Importantly, OpenELM outperforms existing open LLMs that are pre-trained using publicly available datasets (Table 1). For example, OpenELM with 1.1 billion parameters outperforms OLMo (Groeneveld et al., 2024) with 1.2 billion parameters, by 2.36% while requiring 2× fewer pre-training tokens.

2. Pre-training

This section describes the framework, including model architecture (§2.1), pre-training data (§2.2), training hyperparameters and evaluation (§2.3).

2.1. OpenELM architecture

We adopt the decoder-only transformer architecture. Following state-of-the-art LLMs, we: (1) do not use learnable bias parameters in any fully-connected (*a.k.a.* linear) lay-

[1]Apple.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

ers, (2) apply pre-normalization using RMSNorm (Zhang & Sennrich, 2019) and also, use rotatory positional embedding (ROPE) (Su et al., 2024) to encode positional information, (3) use grouped query attention (GQA) (Ainslie et al., 2023) instead of multi-head attention (MHA), (4) replace the feed forward network (FFN) with SwiGLU FFN (Shazeer, 2020), (5) use flash attention (Dao et al., 2022) for computing the scaled dot-product attention, and (6) use the same tokenizer as LLama (Touvron et al., 2023).

Existing LLMs use the same configuration for each transformer layer, resulting in a uniform parameter allocation across layers. Unlike these models, each layer in OpenELM has a different configuration (*e.g.*, number of heads and feed forward network dimension), resulting in variable number of parameters in each layer. This lets OpenELM to better utilize the available parameter budget for achieving higher accuracies. We implement this non-uniform allocation of parameters across layers using *layer-wise scaling* (*a.k.a.* block-wise scaling in (Mehta et al., 2020)).

Layer-wise scaling. A standard transformer layer is composed of multi-head attention (MHA) and feed-forward network (FFN). For non-uniform allocation of parameters in the transformer layer, we adjust the number of attention heads and the FFN multiplier in each transformer layer.

Assume that the standard transformer model with uniform parameter allocation has N transformer layers and the dimensionality of the input to each layer is d_{model}. The MHA has n_h heads and dimension of each head is $d_h = \frac{d_{model}}{n_h}$. Also, the hidden dimension for FFN is $d_{\text{FFN}} = m \cdot d_{model}$, where m is a scalar FFN multiplier.

We introduce parameters α and β to scale the number of attention heads n_h and FFN multiplier m per layer respectively. For the i-th layer, n_h and m are computed as

$$
\begin{aligned}
n_h^i &= \frac{\alpha^i \cdot d_{model}}{d_h}, \quad m^i = \beta^i \\
\text{where } \alpha^i &= \alpha_{min} + \frac{(\alpha_{max} - \alpha_{min}) \cdot i}{N-1}, \\
\text{and } \beta^i &= \beta_{min} + \frac{(\beta_{max} - \beta_{min}) \cdot i}{N-1}, 0 \leq i < N.
\end{aligned} \tag{1}
$$

Here, α_{min} and α_{max} are the hyper-parameters that allow us to scale the attention heads. Similarly, β_{min} and β_{max} let us to vary the width of FFN layers. Therefore, varying the configuration of standard transformer layers using α and β results in non-uniform allocation of parameters in the model. Note, setting $\alpha_{min} = \alpha_{max} = 1.0$ and $m_i = m$ produces the standard uniform transformer model.

2.2. Pre-training data

For pre-training, we use public datasets. Specifically, our pre-training dataset contains RefinedWeb (Penedo et al., 2023), deduplicated PILE (Gao et al., 2020), a subset of RedPajama (Computer, 2023), and a subset of Dolma v1.6 (Soldaini et al., 2024), totaling approximately 1.8 trillion tokens. These details are also summarized in Appendix A.

On-the-fly tokenization and data filtering. Unlike previous approaches that utilize pre-tokenized data (Biderman et al., 2023; Groeneveld et al., 2024), we filter and tokenize text data on-the-fly. This facilitates seamless experimentation with various tokenizers, thereby significantly simplifying prototyping and research endeavors. In our experiments, we use the LLama tokenizer (Touvron et al., 2023).

To filter out low-length sequences, we apply two filtering methods. The first method operates at the *character-level*, checking if the number of characters in the sequence is below a specified threshold. The second method operates at the *token-level*, where it examines whether the sequence contains fewer tokens than a specified threshold. Sequences that are shorter than either of these thresholds are skipped. In our experiments, we use 200 characters and 256 tokens as character and token-level filtering thresholds.

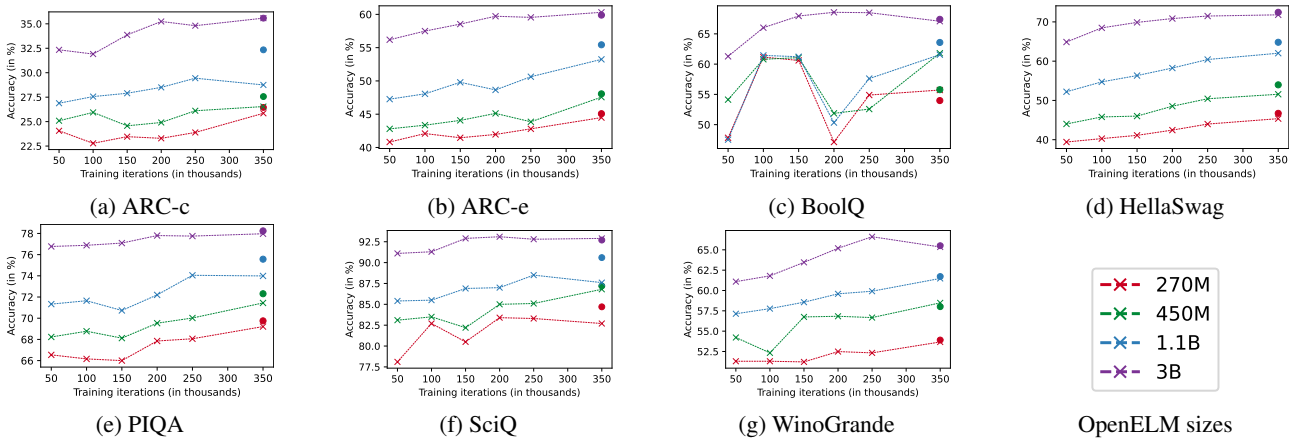
2.3. Training and evaluation details

We train OpenELM variants for 350k iterations (or training steps) using CoreNet (formerly CVNets (Mehta et al., 2022)). We use AdamW (Loshchilov & Hutter, 2017) as an optimizer. We use a cosine learning rate schedule (Loshchilov & Hutter, 2016), with warm up of 5k iterations, and decay the final learning rate down to 10% of maximum learning rate. We use a weight decay of 0.1 and gradient clipping of 1.0. We train four variants of OpenELM (270M, 450M, 1.1B, and 3B), and for some, we use FSDP (Zhao et al., 2023) and activation checkpointing (Chen et al., 2016). We evaluate the accuracy using zero- and few-shot settings across different tasks using LM Evaluation Harness (Gao et al., 2021). Please refer to Appendix A for additional pre-training and evaluation details.

3. Experimental Results

We evaluate the performance of OpenELM on zero-shot and few-shot settings (Table 7). We compare OpenELM with publicly available LLMs, namely PyThia (Biderman et al., 2023), Cerebras-GPT (Dey et al., 2023), TinyLlama (Zhang et al., 2024), OpenLM (Gururangan et al., 2023), MobiLlama (Thawakar et al., 2024), and OLMo (Groeneveld et al., 2024). The works most closely related to ours are MobiLlama and OLMo. These models are trained on comparable dataset mixtures, with similar or larger number of pre-training tokens.

In Figure 1, the accuracy of OpenELM is plotted against training iterations for 7 standard zero-shot tasks. We ob-



(a) ARC-c (b) ARC-e (c) BoolQ (d) HellaSwag

(e) PIQA (f) SciQ (g) WinoGrande

Figure 1. **OpenELM's performance across training iterations on standard zero-shot tasks**. In the majority of tasks, the performance of OpenELM shows improvement with increasing training duration. Furthermore, the model checkpoint obtained by averaging the last five checkpoints, collected at intervals of 5k iterations, demonstrates comparable or slightly better performance (indicated by ● markers) as compared to the last checkpoint obtained after 350k iterations.

Model	Model size	Pretraining tokens	ARC-c	ARC-e	BoolQ	HellaSwag	PIQA	SciQ	WinoGrande	Average	Average w/o SciQ
OpenELM (Ours)	0.27 B	1.5 T	26.45	45.08	53.98	46.71	69.75	84.70	53.91	54.37	49.31
MobiLlama (Thawakar et al., 2024)	0.50 B	1.3 T	26.62	46.04	55.72	51.06	71.11	83.60	53.20	55.34	50.63
OpenELM (Ours)	0.45 B	1.5 T	**27.56**	**48.06**	**55.78**	**53.97**	**72.31**	**87.20**	**58.01**	**57.56**	**52.62**
TinyLlama (Zhang et al., 2024)	1.10 B	3.0 T	30.12	55.25	57.83	59.20	73.29	-	59.12	-	55.80
OpenLM (Gururangan et al., 2023)	1.00 B	1.6 T	31.00	56.00	**65.00**	61.00	74.00	-	60.00	-	57.83
MobiLlama (Thawakar et al., 2024)	0.80 B	1.3 T	28.84	49.62	60.03	52.45	73.18	85.90	55.96	58.00	53.35
MobiLlama (Thawakar et al., 2024)	1.26 B	1.3 T	31.91	56.65	60.34	62.18	74.81	89.10	59.27	62.04	57.53
OLMo (Groeneveld et al., 2024)	1.18 B	3.0 T	31.06	**57.28**	61.74	62.92	75.14	87.00	59.98	62.16	58.02
OpenELM (Ours)	1.08 B	1.5 T	**32.34**	55.43	63.58	**64.81**	**75.57**	**90.60**	**61.72**	**63.44**	**58.91**
OpenELM (Ours)	3.04 B	1.5 T	35.58	59.89	67.40	72.44	78.24	92.70	65.51	67.39	63.18

(a) Results on zero-shot tasks with respect to the standard metrics defined in Table 7a.

Model	Model size	Pretraining tokens	ARC-c	HellaSwag	MMLU	TruthfulQA-mc2	WinoGrande	Average
Cerebras-GPT (Dey et al., 2023)	0.26 B	5.1 B	22.01	28.99	**26.83**	**45.98**	52.49	35.26
OPT (Zhang et al., 2022)	0.35 B	0.2 T	23.55	36.73	26.02	40.83	52.64	35.95
OpenELM (Ours)	0.27 B	1.5 T	**27.65**	**47.15**	25.72	39.24	**53.83**	**38.72**
Pythia (Biderman et al., 2023)	0.41 B	0.3 T	24.83	41.29	25.99	**40.95**	54.38	37.49
MobiLlama (Thawakar et al., 2024)	0.50 B	1.3 T	29.52	52.75	**26.09**	37.55	56.27	40.44
OpenELM (Ours)	0.45 B	1.5 T	**30.20**	**53.86**	26.01	40.18	**57.22**	**41.50**
MobiLlama (Thawakar et al., 2024)	0.80 B	1.3 T	30.63	54.17	25.2	38.41	56.35	40.95
Pythia (Biderman et al., 2023)	1.40 B	0.3 T	32.68	54.96	25.56	**38.66**	57.30	41.83
MobiLlama (Thawakar et al., 2024)	1.26 B	1.3 T	34.64	63.27	23.87	35.19	60.77	43.55
OLMo (Groeneveld et al., 2024)	1.18 B	3.0 T	34.47	63.81	26.16	32.94	60.46	43.57
OpenELM (Ours)	1.08 B	1.5 T	**36.69**	**65.71**	**27.05**	36.98	**63.22**	**45.93**
OpenELM (Ours)	3.04 B	1.5 T	42.24	73.28	26.76	34.98	67.25	48.90

(b) Results on OpenLLM Leaderboard tasks with respect to the metrics defined in Table 7b.

Model	Model size	Pretraining tokens	ARC-c	CrowS-Pairs	HellaSwag	MMLU	PIQA	RACE	TruthfulQA	WinoGrande	Average
OpenELM (Ours)	0.27 B	1.5 T	27.65	66.79	47.15	25.72	69.75	30.91	39.24	53.83	45.13
MobiLlama (Thawakar et al., 2024)	0.50 B	1.3 T	29.52	65.47	52.75	**26.09**	71.11	32.15	37.55	56.27	46.37
OpenELM (Ours)	0.45 B	1.5 T	**30.20**	**68.63**	**53.86**	26.01	**72.31**	**33.11**	**40.18**	**57.22**	**47.69**
MobiLlama (Thawakar et al., 2024)	0.80 B	1.3 T	30.63	66.25	54.17	25.2	73.18	33.68	**38.41**	56.35	47.23
MobiLlama (Thawakar et al., 2024)	1.26 B	1.3 T	34.64	70.24	63.27	23.87	74.81	35.02	35.19	60.77	49.73
OLMo (Groeneveld et al., 2024)	1.18 B	3.0 T	34.47	69.95	63.81	26.16	75.14	**36.75**	32.94	60.46	49.96
OpenELM (Ours)	1.08 B	1.5T	**36.69**	**71.74**	**65.71**	**27.05**	**75.57**	36.46	36.98	**63.22**	**51.68**
OpenELM (Ours)	3.04 B	1.5 T	42.24	73.29	73.28	26.76	78.24	38.76	34.98	67.25	54.35

(c) Results on LLM360 tasks with respect to the metrics defined in Table 7c.

Table 2. **Comparison of OpenELM with publicly available LLMs across various evaluation frameworks.**. We chose MobiLlama and OLMo as our baselines because they are pre-trained on public datasets using a similar or larger number of tokens. We evaluate OpenELM, MobiLlama, and OLMo using the same LM evaluation harness version. Results for other models in Table 2a and Table 2b are taken from their official GitHub repositories and the OpenLLM leaderboard (Beeching et al., 2023), respectively. Best task accuracy for each model category is highlighted in bold. Models pre-trained with less data are highlighted in gray color.

serve an overall increase in accuracy with longer training durations across most tasks. Additionally, the checkpoint obtained by averaging the last five checkpoints, collected at intervals of 5000 iterations, demonstrates comparable or slightly better accuracy compared to the final checkpoint obtained after 350k iterations. This improvement is likely due to noise reduction through weight averaging. Consequently, we use the averaged checkpoint for our main evaluations in Table 2, instruction tuning experiments in Table 9, and parameter-efficient tuning experiments in Table 10.

The results in Table 2 span across various evaluation frameworks, and highlights OpenELM's effectiveness over existing methods. For instance, an OpenELM variant with 1.1 billion parameters achieves 1.28% (Table 2a), 2.36% (Table 2b), and 1.72% (Table 2c) higher accuracy compared to OLMo with 1.2 billion parameters. Remarkably, OpenELM achieves this level of accuracy while using 2× less pre-training data.

Please refer to Appendix B for additional results on instruction and parameter-efficient fine-tuning.

4. Benchmarking

We benchmark OpenELM on two consumer-grade devices: (1) NVIDIA RTX 4090 GPU and and (2) Apple Macbook Pro (see Appendix C for additional details about hardware and evaluation set-up).

Tables 3a and 3b shows the benchmarking results on GPU and MacBook Pro respectively. Despite OpenELM's higher accuracy for a similar parameter count, we observe that it is slower than OLMo. While the primary focus of this study is reproducibility rather than inference performance, we did comprehensive profiling to understand the bottlenecks. Our analysis reveals that a significant portion of OpenELM's processing time can be attributed to our naive implementation of RMSNorm (Table 4). Specifically, naive RMSNorm implementation results in many individual kernel launches each of which processes a small input, rather than a launch of a single, fused kernel, as would be the case with *e.g.* LayerNorm. By replacing the naive RMSNorm with Apex's RMSNorm (NVIDIA Corporation, 2024), we observe a notable increase in OpenELM's throughput. However, a substantial performance gap persists compared to the models that use optimized LayerNorm, in part because (1) OpenELM has 113 RMSNorm layers as compared to 33 LayerNorm layers in OLMo and (2) Apex's RMSNorm is not optimized for small inputs. To further illustrate the performance degradation attributable to RMSNorm, we replaced the LayerNorm in OLMo with RMSNorm, and observed a significant drop in generation throughput. In future work, we plan to explore optimization strategies to further improve the inference efficiency of OpenELM.

Model	Model size	Throughput (Tokens per second)		
		Prompt	Generation	Total
OPT (Zhang et al., 2022)	0.35 B	6524.17	214.11	220.21
OpenELM (Ours)	0.27 B	6427.27	159.67	165.85
MobiLlama (Thawakar et al., 2024)	0.50 B	3423.25	136.35	146.86
OpenELM (Ours)	0.45 B	5211.35	128.46	133.42
MobiLlama (Thawakar et al., 2024)	0.80 B	4151.75	126.01	130.08
Pythia (Biderman et al., 2023)	1.40 B	4501.85	139.65	143.83
MobiLlama (Thawakar et al., 2024)	1.26 B	4938.29	142.96	147.67
OLMo (Groeneveld et al., 2024)	1.18 B	7151.65	203.40	209.26
OpenELM (Ours)	1.08 B	3681.73	92.15	95.72
OpenELM (Ours)	3.04 B	2712.56	70.11	72.82

(a) Results on NVIDIA CUDA / Linux.

Model	Throughput (Tokens per second)		
	Prompt	Generation	Total
OpenELM-0.27B	1151.41	212.40	218.45
OpenELM-0.27B-4bit	803.99	256.35	262.70
OpenELM-0.45B	910.61	147.26	151.57
OpenELM-0.45B-4bit	883.19	197.81	203.16
OpenELM-1.08B	508.56	78.72	81.04
OpenELM-1.08B-4bit	554.17	117.90	121.14
OpenELM-3.04B-bf16	234.96	33.96	34.97
OpenELM-3.04B-bf16-4bit	211.32	60.33	61.83

(b) Results for the MLX port on Apple macOS.

Table 3. **Benchmark measurements of OpenELM compared to other similar LLMs in its class.**. On CUDA, we evaluate OpenELM, MobiLlama, and OLMo using the CoreNet version of OpenELM and HuggingFace for the other two. On macOS, we only provide results for the MLX version of OpenELM.

Model	Normalization layer (# Invocations per token)	Throughput (Tokens per second)		
		Prompt	Generation	Total
OLMo	LayerNorm (33)	7151.65	203.40	209.26
	RMSNorm-Naive (33)	5360.56	171.41	176.92
OpenELM (Ours)	LayerNorm (113)	4697.50	130.34	135.38
	RMSNorm-Naive (113)	3681.73	92.15	95.72
	RMSNorm-Apex (113)	4280.66	113.42	117.81

Table 4. **Normalization layers are a bottleneck.** The throughput of both OLMo-1.18B and OpenELM-1.08B significantly decreases with the naive RMSNorm PyTorch implementation compared to highly optimized LayerNorm (Ba et al., 2016). Although Apex's (NVIDIA Corporation, 2024) RMSNorm implementation leads to notable throughput improvements compared to the naive implementation, a considerable performance gap persists compared to LayerNorm. This highlights the substantial optimization potential for future endeavors. The number of invocations per token for each normalization layer is indicated next to the layer name in brackets.

5. Conclusion

This work introduces OpenELM, a decoder-only transformer based open language model. The OpenELM uses a layer-wise scaling method for efficient parameter allocation within the transformer model, resulting in improved accuracy compared to existing models. Additionally, we release the entire framework open, including training logs, multiple checkpoints, pre-training configurations, and MLX inference code. This extensive release aims to empower the open research community, facilitating future research effors.

References

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.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Bartolome, A., Martin, G., and Vila, D. Notus. https://github.com/argilla-io/notus, 2023.

Beeching, E., Fourrier, C., Habib, N., Han, S., Lambert, N., Rajani, N., Sanseviero, O., Tunstall, L., and Wolf, T. Open llm leaderboard. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard, 2023.

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. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 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.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 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. *arXiv preprint arXiv:1803.05457*, 2018.

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

Cui, G., Yuan, L., Ding, N., Yao, G., Zhu, W., Ni, Y., Xie, G., Liu, Z., and Sun, M. Ultrafeedback: Boosting language models with high-quality feedback, 2023.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Dey, N., Gosal, G., Khachane, H., Marshall, W., Pathria, R., Tom, M., Hestness, J., et al. Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster. *arXiv preprint arXiv:2304.03208*, 2023.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, September 2021. URL https://doi.org/10.5281/zenodo.5371628.

Groeneveld, D., Beltagy, I., Walsh, P., Bhagia, A., Kinney, R., Tafjord, O., Jha, A. H., Ivison, H., Magnusson, I., Wang, Y., et al. Olmo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*, 2024.

Gururangan, S., Wortsman, M., Gadre, S. Y., Dave, A., Kilian, M., Shi, W., Mercat, J., Smyrnis, G., Ilharco, G., Jordan, M., Heckel, R., Dimakis, A., Farhadi, A., Shankar, V., and Schmidt, L. OpenLM: A minimal but performative language modeling (lm) repository, 2023. URL https://github.com/mlfoundations/open_lm/. GitHub repository.

Hannun, A., Digani, J., Katharopoulos, A., and Collobert, R. MLX: Efficient and flexible machine learning on apple silicon, 2024. URL https://github.com/ml-explore.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Hu, J. E., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., and Chen, W. Lora: Low-rank adaptation of large language models. *ArXiv*, abs/2106.09685, 2021. URL https://api.semanticscholar.org/CorpusID:235458009.

Hu, Z., Lan, Y., Wang, L., Xu, W., Lim, E.-P., Lee, R. K.-W., Bing, L., and Poria, S. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *ArXiv*, abs/2304.01933, 2023. URL https://api.semanticscholar.org/CorpusID:257921386.

Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

Lin, S., Hilton, J., and Evans, O. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*, 2021.

Liu, T., Zhao, Y., Joshi, R., Khalman, M., Saleh, M., Liu, P. J., and Liu, J. Statistical Rejection Sampling Improves Preference Optimization, January 2024. URL http://arxiv.org/abs/2309.06657. arXiv:2309.06657 [cs].

Liu, Z., Qiao, A., Neiswanger, W., Wang, H., Tan, B., Tao, T., Li, J., Wang, Y., Sun, S., Pangarkar, O., et al. Llm360: Towards fully transparent open-source llms. *arXiv preprint arXiv:2312.06550*, 2023.

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.

Mehta, S., Ghazvininejad, M., Iyer, S., Zettlemoyer, L., and Hajishirzi, H. Delight: Deep and light-weight transformer. *arXiv preprint arXiv:2008.00623*, 2020.

Mehta, S., Abdolhosseini, F., and Rastegari, M. Cvnets: High performance library for computer vision. In *Proceedings of the 30th ACM International Conference on Multimedia*, pp. 7327–7330, 2022.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Nangia, N., Vania, C., Bhalerao, R., and Bowman, S. R. Crows-pairs: A challenge dataset for measuring social biases in masked language models. *arXiv preprint arXiv:2010.00133*, 2020.

NVIDIA Corporation. Apex: A pytorch extension with tools for mixed precision training and more. GitHub, 2024. URL https://github.com/NVIDIA/apex.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, volume 32, 2019.

Penedo, G., Malartic, Q., Hesslow, D., Cojocaru, R., Cappelli, A., Alobeidli, H., Pannier, B., Almazrouei, E., and Launay, J. The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023.

Press, O. and Wolf, L. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., and Finn, C. Direct Preference Optimization: Your Language Model is Secretly a Reward Model, December 2023. URL http://arxiv.org/abs/2305.18290. arXiv:2305.18290 [cs].

Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3505–3506, 2020.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Sap, M., Rashkin, H., Chen, D., LeBras, R., and Choi, Y. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

Shazeer, N. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Soldaini, L., Kinney, R., Bhagia, A., Schwenk, D., Atkinson, D., Authur, R., Bogin, B., Chandu, K., Dumas, J., Elazar, Y., et al. Dolma: An open corpus of three trillion tokens for language model pretraining research. *arXiv preprint arXiv:2402.00159*, 2024.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Thawakar, O., Vayani, A., Khan, S., Cholakal, H., Anwer, R. M., Felsberg, M., Baldwin, T., Xing, E. P., and Khan, F. S. Mobillama: Towards accurate and lightweight fully transparent gpt. *arXiv preprint arXiv:2402.16840*, 2024.

Ting, H. W. Accuracy not matched for llama1-7b. GitHub issue, 2024. https://github.com/EleutherAI/lm-evaluation-harness/issues/1294.

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.

Tunstall, L., Beeching, E., Lambert, N., Rajani, N., Huang, S., Rasul, K., Rush, A. M., and Wolf, T. The alignment handbook. https://github.com/huggingface/alignment-handbook, 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.

Welbl, J., Liu, N. F., and Gardner, M. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*, 2017.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

yang Liu, S., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. Dora: Weight-decomposed low-rank adaptation. *ArXiv*, abs/2402.09353, 2024. URL https://api.semanticscholar.org/CorpusID:267657886.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zhang, B. and Sennrich, R. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Zhang, P., Zeng, G., Wang, T., and Lu, W. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhao, Y., Gu, A., Varma, R., Luo, L., Huang, C.-C., Xu, M., Wright, L., Shojanazeri, H., Ott, M., Shleifer, S., et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

A. Additional pre-training details

A.1. Pre-training datasets

Table 5 represents the dataset mix used for pre-training OpenELM together with the number of tokens in each data source.

Source	Subset	Tokens
RefinedWeb		665 B
RedPajama	Github	59 B
	Books	26 B
	ArXiv	28 B
	Wikipedia	24 B
	StackExchange	20 B
	C4	175 B
PILE		207 B
Dolma	The Stack	411 B
	Reddit	89 B
	PeS2o	70 B
	Project Gutenberg	6 B
	Wikipedia + Wikibooks	4.3 B

Table 5. Dataset used for pre-training OpenELM.

A.2. Pre-training hyper-parameters

The pre-training hyper-parameters for different OpenELM configurations are given in Table 6.

	270M	450M	1.1B	3B
Dimension d_{model}	1280	1536	2048	3072
Num. of layers N	16	20	28	36
Head dimension d_h	64	64	64	128
$\alpha_{min}, \alpha_{max}$ (Equation (1))	0.5, 1.0			
β_{min}, β_{max} (Equation (1))	0.5, 4.0			
Normalization layer	RMSNorm			
Positional embeddings	RoPE			
Attention variant	Grouped query attention			
Activation	SwiGLU			
Context length	2048			
Batch size (tokens)	approx. 4M			
Weight tying (Press & Wolf, 2016)	yes			
Warm-up iterations	5,000			
Training steps	350,000			
Warm-up init. LR	0.000001			
Max. LR	0.0053	0.0039	0024	0.0012
Min. LR	10% of the max. LR			
Loss function	Cross-entropy			
Optimizer	AdamW (β_1=0.9, β_2=0.95, $\epsilon = 1.e-8$)			
Weight decay	0.1			
Activation checkpointing	✗	✓	✓	✓
FSDP	✗	✗	✗	✓
GPUs	128	128	128	128
GPU Type	A100	H100	A100	H100
GPU Memory	80 GB	80 GB	80 GB	80 GB
Training time (in days)	3	3	11	13

Table 6. Pre-training details for different variants of OpenELM.

A.3. Evaluation details

Following previous works, we evaluate the performance across different tasks using LM Evaluation Harness (Gao et al., 2021)[1]:

- **Standard zero-shot tasks.** We consider 7 standard common-sense reasoning tasks: ARC easy and challenge (Clark et al., 2018), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), SciQ (Welbl et al., 2017), and WinoGrande (Sakaguchi et al., 2021).
- **OpenLLM leaderboard tasks.** We use 5 tasks from OpenLLM leaderboard (Beeching et al., 2023): ARC challenge, HellaSwag, MMLU (Hendrycks et al., 2020), TruthfulQA (Lin et al., 2021), and WinoGrande.
- **LLM360 leaderboard tasks.** We use 7 tasks from LLM360 leaderboard (Liu et al., 2023) for evaluation: ARC challenge, CrowS-Pairs (English version) (Nangia et al., 2020), HellaSwag, WinoGrande, MMLU, PIQA, and RACE (Lai et al., 2017).

These evaluation frameworks, built on top of LM Evaluation Harness, allows us to comprehensively evaluate OpenELM in terms of reasoning (*e.g.*, ARC-c, HellaSwag, and PIQA), knowledge understanding (*e.g.*, MMLU and RACE), and misinformation & bias (*e.g.*, TruthfulQA and CrowS-Pairs). While there may be some overlap in tasks among these frameworks, they primarily differ in the few-shot settings, as outlined in Table 7.

B. Additional results

This section provides additional results on instruction and parameter-efficient tuning tasks.

B.1. Instruction Tuning

We use the cleaned variant of UltraFeedback (Cui et al., 2023; Bartolome et al., 2023) dataset that consists of 60k prompts for instruction tuning. We do instruction tuning using Alignment Handbook library (Tunstall et al., 2023). For optimization, we use either the statistical rejection sampling method (Liu et al., 2024) or the direct preference optimization method (Rafailov et al., 2023).

Hyper-parameters. We conducted a grid search to determine optimal values for the learning rate and training epochs. For the learning rate, we explored values in the range of [2e-5, 3e-5, 5e-5, 8e-5, 1e-4], while for training epochs, we investigated the range of [3, 5, 8, 10]. The final

[1]We use commit dc90fec of https://github.com/EleutherAI/lm-evaluation-harness

Task	Metric
ARC-c	Normalized accuracy
ARC-e	Normalized accuracy
BoolQ	Accuracy
HellaSwag	Normalized accuracy
PIQA	Normalized accuracy
SciQ	Accuracy
WinoGrande	Accuracy

(a) Standard zero-shot metrics

Task	Metric	Num. few shot examples
ARC-c	Normalized accuracy	25
HellaSwag	Normalized accuracy	10
MMLU	Accuracy	5
TruthfulQA-mc2	Accuracy	0
WinoGrande	Accuracy	5

(b) OpenLLM leaderboard

Task	Metric	Num. few shot examples
ARC-c	Normalized accuracy	25
CrowsPairs-En	PCT stereotype	25
HellaSwag	Normalized accuracy	10
WinoGrande	Accuracy	5
MMLU	Accuracy	5
PIQA	Normalized accuracy	0
RACE	Accuracy	0

(c) LLM360

Table 7. Tasks and metrics used for evaluating OpenELM.

	270M	450M	1.1B	3B
Batch size			8	
Training epochs	5	8	5	10
Learning rate	2e-5	3e-5	5e-5	1e-4
Loss function	hinge	hinge	sigmoid	hinge
DeepSpeed Zero3 (Rasley et al., 2020)	✗	✓	✓	✓
GPUs			8	
GPU Type	A100	A100	A100	A100
GPU Memory	40 GB	40 GB	40 GB	80 GB
Training time (in hours)	2.5	4.3	6.6	14.2

Table 8. Instruction tuning details for different variants of OpenELM.

recipe selected is the one that yielded the highest average accuracy across various tasks as presented in Table 7a and Table 7c. We finetune all the models with BFloat16 as a data type. We use activation checkpointing along with gradient accumulation with a step size of two. We use the AdamW optimizer with default beta values. We use the cosine learning rate scheduler with a warm-up ratio of 0.1, and we set the weight decay to 0 and loss temperature beta to 0.01. We set the maximum context length to 1024 and maximum prompt length to 512. Other hyper-parameters are included in Table 8.

Results. Table 9 shows that instruction tuning consistently improves OpenELM's average accuracy by 1-2% across different evaluation frameworks.

B.2. Parameter-efficient fine-tuning (PEFT)

We use the CommonSense reasoning training and evaluation setup (Hu et al., 2023). This setup provides 170k training samples across 8 multiple-choice datasets for PEFT studies with different methods, including LoRA (Hu et al., 2021) and DoRA (yang Liu et al., 2024). We integrate OpenELM with these methods, and finetune the resulting model for three epochs using 8 NVIDIA H100 GPUs. Table 10 shows that PEFT methods can be applied to OpenELM. LoRA and DoRA deliver similar accuracy on average across the given CommonSense reasoning datasets.

C. Additional benchmarking details

Hardware. We benchmark on modern, consumer-grade hardware with BFloat16 as the data type. Specifically, CUDA benchmarks were performed on a workstation with an Intel i9-13900KF CPU, equipped with 64 GB of DDR5-4000 DRAM, and an NVIDIA RTX 4090 GPU with 24 GB of VRAM, running Ubuntu 22.04. PyTorch v2.2.2 (Paszke et al., 2019) was used, with the most recent versions of models and the associated libraries. HuggingFace Transformers v4.39.3 (Wolf et al., 2020) was used to benchmark HuggingFace models. We did not use Torch Inductor for model compilation.

To benchmark OpenELM models on the Apple silicon, we used an Apple MacBook Pro with an M2 Max system-on-chip and 64GiB of RAM, running macOS 14.4.1. We ported the code and model weights to Apple MLX v0.10.0 (Hannun et al., 2024). To maximize the throughput, lazy evaluation was used in MLX with 8 tokens evaluated at a time.

Evaluation. We provide two separate measurements for token throughput (measured in terms of tokens processed per second): (1) prompt processing (pre-fill), and (2) token generation. Additionally, we also report the total combined throughput. We benchmark all models sequentially, and execute one full "dry run" generating 1024 tokens for the first model, since we found that this significantly increases the throughput of generation for subsequent models. Before measurement for each individual model, we warm up the model by executing a single forward pass to allow the frameworks to perform further auto-tuning, if any. In all experiments, we use key-value caching and generate 1024 tokens in addition to the prompt tokens in all tests. Static key-value cache was used whenever supported. The same prompt was used for all runs, resulting in prompt lengths of 35-36 tokens (depending on the tokenizer).

Model Size	Instruction Tuned?	ARC-c	ARC-e	BoolQ	HellaSwag	PIQA	SciQ	WinoGrande	Average
0.27 B	✗	26.45	45.08	**53.98**	46.71	69.75	**84.70**	**53.91**	54.37
	✓	**30.55**	**46.68**	48.56	**52.07**	**70.78**	84.40	52.72	**55.11**
0.45 B	✗	27.56	48.06	55.78	53.97	72.31	87.20	58.01	57.56
	✓	**30.38**	**50.00**	**60.37**	**59.34**	**72.63**	**88.00**	**58.96**	**59.95**
1.08 B	✗	32.34	**55.43**	63.58	64.81	**75.57**	**90.60**	61.72	63.44
	✓	**37.97**	52.23	**70.00**	**71.20**	75.03	89.30	**62.75**	**65.50**
3.04 B	✗	35.58	59.89	67.40	72.44	78.24	**92.70**	65.51	67.39
	✓	**39.42**	**61.74**	**68.17**	**76.36**	**79.00**	92.50	**66.85**	**69.15**

(a) Results on zero-shot tasks with respect to the metrics defined in Table 7a.

Model Size	Instruction Tuned?	ARC-c	HellaSwag	MMLU	TruthfulQA	WinoGrande	Average
0.27 B	✗	27.65	47.15	25.72	**39.24**	**53.83**	38.72
	✓	**32.51**	**51.58**	**26.70**	38.72	53.20	**40.54**
0.45 B	✗	30.20	53.86	**26.01**	40.18	57.22	41.50
	✓	**33.53**	**59.31**	25.41	**40.48**	**58.33**	**43.41**
1.08 B	✗	36.69	65.71	**27.05**	36.98	63.22	45.93
	✓	**41.55**	**71.83**	25.65	**45.95**	**64.72**	**49.94**
3.04 B	✗	42.24	73.28	**26.76**	34.98	67.25	48.90
	✓	**47.70**	**76.87**	24.80	**38.76**	**67.96**	**51.22**

(b) Results on OpenLLM Leaderboard tasks with respect to the metrics defined in Table 7b.

Model Size	Instruction Tuned?	ARC-c	CrowS-Pairs	HellaSwag	MMLU	PIQA	RACE	TruthfulQA	WinoGrande	Average
0.27 B	✗	27.65	**66.79**	47.15	25.72	69.75	30.91	**39.24**	**53.83**	45.13
	✓	**32.51**	66.01	**51.58**	**26.70**	**70.78**	33.78	38.72	53.20	**46.66**
0.45 B	✗	30.20	**68.63**	53.86	**26.01**	72.31	33.11	40.18	57.22	47.69
	✓	**33.53**	67.44	**59.31**	25.41	**72.63**	**36.84**	**40.48**	**58.33**	**49.25**
1.08 B	✗	36.69	**71.74**	65.71	**27.05**	**75.57**	36.46	36.98	63.22	51.68
	✓	**41.55**	71.02	**71.83**	25.65	75.03	**39.43**	**45.95**	**64.72**	**54.40**
3.04 B	✗	42.24	**73.29**	73.28	**26.76**	78.24	**38.76**	34.98	67.25	54.35
	✓	**47.70**	72.33	**76.87**	24.80	**79.00**	38.47	**38.76**	**67.96**	**55.73**

(c) Results on LLM360 tasks with respect to the metrics defined in Table 7c.

Table 9. **Instruction tuning improves OpenELM's accuracy across different model sizes.**

Model Size	PEFT	ARC-c	ARC-e	BoolQ	HellaSwag	PIQA	SIQA	WinoGrande	OBQA	Average
0.27 B	LoRA	24.57	26.60	62.14	24.84	50.05	42.02	49.88	28.00	38.51
	DoRA	26.19	28.07	62.20	25.22	50.11	44.42	50.12	31.20	39.69
0.45 B	LoRA	28.67	29.88	62.29	25.85	52.39	49.59	50.91	33.20	41.60
	DoRA	28.33	30.39	62.26	25.12	52.29	49.28	50.83	32.00	41.31
1.08 B	LoRA	45.14	61.11	61.77	77.95	72.31	69.70	61.64	59.20	63.60
	DoRA	44.11	61.49	61.68	78.92	71.38	69.04	64.01	58.80	63.68
3.04 B	LoRA	46.93	66.25	62.48	81.22	75.19	70.62	65.51	58.20	65.80
	DoRA	46.50	66.46	62.35	80.84	75.73	70.83	63.77	58.20	65.59

Table 10. **OpenELM with PEFT.** Both LoRA and DoRA demonstrate comparable performance when OpenELM is finetuned on CommonSense reasoning benchmark. It's important to note that these *fine-tuning results*, obtained using the evaluation setup of LLM-Adapters (Hu et al., 2023), differ from the results in Tables 2 and 9. This is because the results in Tables 2 and 9 are obtained under zero- and few-shot settings using LM Evaluation Harness. Note that we did not use social interactions QA (SIQA; (Sap et al., 2019)) and OpenBookQA (OBQA; (Mihaylov et al., 2018)) in Tables 2 and 9 because of evaluation issues with LLama tokenizer in LM Evaluation Harness (see (Ting, 2024)).