Revisiting MoE and Dense Speed-Accuracy Comparisons for LLM Training

Xianzhi Du Tom Gunter Xiang Kong Mark Lee **Zirui Wang**
Aonan Zhang **Nan Du** **Ruoming Pang**
Apple {xianzhi,r_pang}@apple.com

Abstract

Mixture-of-Experts (MoE) enjoys performance gain by increasing model capacity while keeping computation cost constant. When comparing MoE to dense models, prior work typically adopt the following setting: 1) use FLOPs or activated parameters as a measure of model complexity; 2) train all models to the same number of tokens. We argue that this setting favors MoE as FLOPs and activated parameters do not accurately measure the communication overhead in sparse layers, leading to a larger actual training budget for MoE. In this work, we revisit the settings by adopting step time as a more accurate measure of model complexity, and by determining the total compute budget under the Chinchilla compute-optimal settings. To efficiently run MoE on modern accelerators, we adopt a 3D sharding method that keeps the dense-to-MoE step time increase within a healthy range. We evaluate MoE and dense LLMs on a set of nine 0-shot and two 1-shot English tasks, as well as MMLU 5-shot and GSM8K 8-shot across three model scales at 6.4B, 12.6B, and 29.6B. Experimental results show that even under these settings, MoE consistently outperform dense LLMs on the speed-accuracy trade-off curve with meaningful gaps. Our full model implementation and sharding strategy will be released at https://github.com/apple/axlearn.

1 Introduction

Recently, MoE has shown promising results on language (Zoph et al. (2022); Du et al. (2022); Zhou et al. (2022); Fedus et al. (2022); Jiang et al. (2024); Komatsuzaki et al. (2023); Shen et al. (2023); Dai et al. (2024)), multimodal (Mustafa et al. (2022); Lin et al. (2024)) and computer vision (Ruiz et al. (2021); Komatsuzaki et al. (2023); Daxberger et al. (2023); Chen et al. (2023)) tasks. By decoupling computation cost from model scale, MoE scales model capacity without affecting computation cost.

When comparing MoE to dense models, it is common in existing work to use FLOPs or activated model parameters as a measure of a model's computation cost (Ruiz et al. (2021); Jiang et al. (2024); Du et al. (2022); Shen et al. (2023); Zhou et al. (2022); Dai et al. (2024); Lin et al. (2024)). However, as MoE sparsity grows, the communication overhead during routing (e.g. the `all2all` and `allreduce` communication primitives) also increases. Such communication overhead cannot be accurately captured by FLOPs or number of activated parameters, leading to a setting that favors MoE. Furthermore, existing work commonly adopt an identical training recipe for MoE and dense models, i.e. utilizing the same batch size and same number of training steps, resulting in a higher total effective computation cost for MoE.

In this work, we propose an alternative setting for comparing MoE and dense models under the modern LLM training paradigm. Specifically, we propose two main revisions: 1) Use step time to measure model complexity which fully captures the communication overhead in MoE layers. The step time for MoE and dense models are measured and optimized under identical settings. 2) Adopt the Chinchilla (Hoffmann et al. (2022)) compute-optimal setting of a 20:1 token-to-parameter ratio, a setting that is optimized for dense LLM training, to decide the total training budget for MoE vs. dense comparison at multiple model scales.

dense-to-MoE step time increase within 20%.

We compare MoE and dense LLMs across a wide range of scales at 6.4B, 12.6B, 29.6B and evaluate the models on a rich set of LLM benchmarks including nine 0-shot and two 1-shot English tasks covering common sense reasoning, question answering and reading comprehension, as well as MMLU 5-shot and GSM8K 8-shot. Extensive evaluations show that under these challenging settings, MoEs consistently outperform dense LLMs on the speed-accuracy trade-off curve.

2 Methods

2.1 Architecture

The dense LLM architecture used in this paper generally follows LLaMA2 (Touvron et al. (2023)). Specifically, we adopt the following transformer architectural modifications: 1) Pre-normalization and RMSNorm (Zhang and Sennrich (2019)); 2) SwiGLU (Shazeer (2020)) as the activation function; 3) RoPE (Su et al. (2023)) as the positional embedding. Grouped-query attention (Ainslie et al. (2023)) is not used as our experiments focus on training.

Our MoE architecture shares the same architecture as the dense LLM, except replacing FFNs with their sparse counterparts in MoE layers. We also make the following design decisions:

Number of sparse layers. More sparse layers leads to higher model sparsity and generally better performance. On the other hand, it increases the model's total parameters and computation cost. Typical choices for this hyper-parameter include Every-K (Jiang et al. (2024); Zoph et al. (2022); Du et al. (2022)) or Last-K (Ruiz et al. (2021); Komatsuzaki et al. (2023)). We adopt the Every-4 setting in our experiments as it provides a better speed-accuracy trade-off. Specifically, we replace the last dense layer out of every 4 layers with a sparse layer.

Number of experts. More experts leads to a higher model capacity while keeping the model's activated parameters constant. Existing work (Du et al. (2022); Fedus et al. (2022); Clark et al. (2022)) shows that using more experts leads to monotonic performance improvement, which gradually diminishes when the number grows beyond 256. In this paper, we experiment with different numbers of experts and show that with our 3D sharding strategy, more experts leads to a better performance without affecting step time.

Routing method and expert capacity. We adopt the Top-K routing (Shazeer et al. (2017)) for autoregressive modeling with $K = 2$ and expert capacity $C = 2$ in all our experiments. Other common routing methods include Top-1 (Fedus et al. (2022)) and expert-choice routing (Zhou et al. (2022)).

Auxiliary loss. To encourage better expert load balancing for Top-K routing, we adopt a load balancing loss with a coefficient of 0.01 (Lepikhin et al. (2021); Du et al. (2022)). We also adopt the router z-loss proposed in ST-MoE (Zoph et al. (2022)) with a coefficient of 0.001, which we found helps with stabilizing large MoE training.

name	type	sharding specification
attention	dense weights	(`Expert`, `Model`)
FFN_1	dense weights	(`Expert`, `Model`)
FFN_2	dense weights	(`Model`, `Expert`)
FFN_1	dense activation	((`Data`, `Expert`), `None`, `Model`)
FFN_2	dense activation	((`Data`, `Expert`), `None`, `Model`)
router, ME	MoE weights	(`None`, `None`)
FFN_1, EMH	MoE weights	(`Expert`, `None`, `Model`)
FFN_2 EHM	MoE weights	(`Expert`, `Model`, `None`)
OGSM	MoE activation	(`Data`, `Expert`, `None`, `Model`)
OGSEC	MoE activation	(`Data`, `Expert`, `None`, `None`, `None`)
OEGCM	MoE activation	(`Data`, `Expert`, `None`, `None`, `Model`)
OGECM	MoE activation	(`Data`, `Expert`, `None`, `None`, `Model`)
OEGCH	MoE activation	(`Data`, `Expert`, `None`, `None`, `Model`)

2.2 3D sharding and MoE Layer Implementation

As we use step time as a measurement for model complexity, optimizing the sparse MoE sharding strategy lies at the very foundation. In this work, we adopt a 3D sharding strategy across 3 axes: (`Data`, `Expert`, `Model`).

`Data`. The `Data` axis is used for data parallelism only. Along this axis, data is evenly partitioned and model weights are fully replicated. This axis is commonly used for inter-slice parallelism, where communication often happens over slower data center networking (DCN).

`Expert`. The `Expert` axis is designed for sharding the experts in sparse FFNs. To optimize the compute-to-memory ratio, we place at most one expert per core. The `Expert` axis provides flexibility and efficiency when scaling the number of experts in one model while keeping step time constant. Note that for dense layers, the `Expert` axis is used as the mesh axis for fully-sharded data parallelism (Zhao et al. (2023)) (FSDP) which shards model parameters across data-parallel cores.

`Model`. The `Model` axis is used to shard attention heads and FFN hidden dimensions. This axis incurs heaviest communication among all of the axes and is typically used for intra-slice model parallelism, where communication happens over fast inter-core interconnect (ICI).

GShard (Lepikhin et al. (2021)) provides an efficient MoE layer implementation that extensively uses Einsum notations to express gating, dispatching and combining operations. In this work, we adopt the GShard implementation provided in the Praxis library[1]

[1] https://github.com/google/praxis

, as well as the outer batch trick used in Mesh-Tensorflow [2]

[2] https://github.com/tensorflow/mesh/blob/master/mesh_tensorflow/transformer/moe.py

. Specifically, the input tokens to the MoE layer are evenly divided into O outer batches, resulting in an additional leading dimension of shape O in all activations. We then shard the outer batch dimension along the `Data` axis for best efficiency. Table 1 provides an overview of our MoE transformer sharding specifications.

tokens for the corresponding dense model. The total compute budget is determined by multiplying the training step time of the dense model and the total number of training steps. Given this budget, we design MoEs and determine the step time and the training steps, fixing the batch size and hardware accelerators. Unlike previous work that constructs MoE from the same scale dense backbone [3]

[3]MoE's dense backbone is the dense model from which the MoE is constructed. For example, 1.2B dense is the backbone of 1.2B/64E.

, we trade model size for training tokens and design MoEs from smaller backbones. Ablations in Sec. 3.3 shows that MoEs designed with this method gives a better speed-accuracy trade-off curve. Table 8 provides the final train budget for all models used in our experiments.

Table 2: Dense and MoE LLM comparisons at 6.4B, 12.6B and 29.6B scales. Note that the step time is only comparable for models within each scale due to being measured on different devices. Note that the 6.4B scale comparison uses CoreEN (0S) while other scales use the CoreEN (all).

model	$\text{params}_{\text{act}}$	**step time**	CoreEN	MMLU	GSM8K
6.4B	**6.4B**	**1.69s**	63.25 (0S)	-	-
1.6B/256E	**2.1B**	**0.82s (-51.5%)**	65.00 (0S)	-	-
4.8B/256E	**6.2B**	**1.41s (-16.7%)**	**65.40** (0S)	-	-
12.6B	**12.6B**	**2.94s**	57.25 (all)	24.84	5.08
4.5B/256E	**5.3B**	**1.50s (-49.0%)**	60.62 (all)	29.52	**12.13**
8.1B/256E	**9.4B**	**2.60s (-11.2%)**	**61.38** (all)	**30.18**	11.97
29.6B	**29.6B**	**6.56s**	62.73 (all)	47.03	**18.88**
6.4B/64E	**7.5B**	**1.85s (-71.8%)**	**63.20** (all)	**48.37**	16.91

3 Experiments

3.1 Experimental Setup

We use a training corpus with a similar data mixture as LLaMA2 (Touvron et al. (2023)). All models are trained with the AdamW optimizer (Loshchilov and Hutter (2019)) with $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\epsilon = 10^{-5}$. We adopt a cosine learning rate schedule with 2000 warmup steps and decay to 10% of the peak learning rate. We use a weight decay of 0.1 and gradient clipping of 1.0. We use the SentencePiece tokenizer (Kudo and Richardson (2018)) with the Byte-Pair Encoding algorithm (Sennrich et al. (2016)). We train our small scale models on TPU-v5e devices and large scale models on TPU-v4 devices.

The MoE and dense LLMs are evaluated on a wide range of benchmarks:

Core English tasks (CoreEN). We evaluate the performance on common sense reasoning, reading comprehension and question answering benchmarks including ARC Easy and Challenge (0-shot) (Clark et al. (2018)), HellaSwag (0-shot) (Zellers et al. (2019)), WinoGrande (0-shot) (Sakaguchi et al. (2019)), PIQA (0-shot) (Bisk et al. (2019)), SciQ (0-shot) (Sap et al. (2019)), LAMBADA (0-shot) (Paperno et al. (2016)), TriviaQA (1-shot) (Joshi et al. (2017)) and WebQS (1-shot) (Berant et al. (2013)). In our experiments, we report both the average performance on the seven 0-shot tasks (denoted as CoreEN (0S)) and the average performance on the whole nine tasks (denoted as CoreEN (all)).

MMLU. We report the 5-shot performance on MMLU (Hendrycks et al. (2021)).

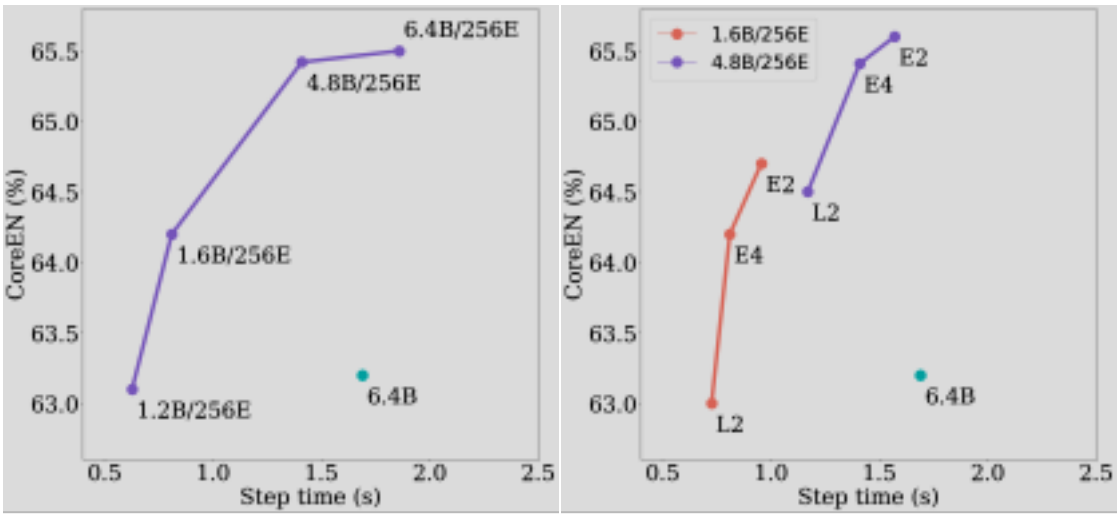
Mathematical reasoning. We report the 8-shot performance on GSM8K (Cobbe et al. (2021)).

while being faster in training step time.

6.4B scale comparison. The 6.4B dense baseline achieves 63.25% accuracy on CoreEN (0S) average and runs at 1.69 second per step on 512 TPU-v4 cores. The two MoE models we tested, 1.6B/256E and 4.8B/256E, both outperform the 6.4B dense while being faster in step time. In particular, the 1.6B/256E model achieves +1.75% on CoreEN (0S) while being 2.06 × as fast as the dense 6.4B. Due to insufficient model capacity and train tokens at this scale, all the models fail to show meaningful results on MMLU 5-shot or GSM8K 8-shot.

12.6B scale comparison. The 12.6B dense baseline achieves 57.25% on CoreEN (all) and runs at 2.94 second per step on 1024 TPU-v4 cores. It fails to show meaningful results on MMLU or GSM8K. The two MoEs we evaluated, 4.5B/256E and 8.1B/256E, both achieve better results on CoreEN (all) while being much faster in step time. At this scale, we further observe that the MoEs begin to achieve meaningful results on MMLU and GSM8K. In particular, the 4.5B/256E model achieves +3.37% on CoreEN average, +4.68% on MMLU and +7.05% on GSM8K while being 1.96 × as fast as the dense 12.6B.

29.6B scale comparison. The 29.6B dense baseline achieves 62.73% on CoreEN (all), 47.03% on MMLU and 18.88% on GSM8K. The 6.4B/64E MoE attains very close performance on the three benchmarks: +0.47% on CoreEN (all), +1.34% on MMLU, and −1.97% on GSM8K, while being 3.55 × as fast as the baseline.



(a) (b)

Figure 1: Comparing MoEs designed from a wider range of choices to the 6.4B dense. We vary the dense backbone scale and the number of sparse layers. On the right figure, L2, E4, E2 represent last-2, every-4, every-2 number of layers, respectively.

3.3 Explore a Wider Range of MoE Architecture Designs

In this section, we show that MoE architectures constructed from a wide range of design choices outperform the dense baseline. Aside from the architectures we used in the main results, we design MoEs by using a wider range of dense backbone scales and different number of MoE layers. We fix the train budget for all ablations to the 6.4B dense

256 then vary the dense backbone scale with 1.2B, 1.6B, 4.8B and 6.4B. Figure 1 (left) shows that the MoEs form a speed-accuracy trade-off curve that outperform the 6.4B dense with a meaningful gap. The trade-off curve also suggests that more MoE architectures interpolated onto this curve should outperform the 6.4B dense. The ablations also suggest that designing MoEs using smaller scale backbones gives better results than using the same dense scale backbone. In particular, the 4.8B/256E MoE underperforms the 6.4B/256E MoE by only 0.08% on CoreEN (0S) while reducing the step time by 24.2%. One hypothesis is that the equivalent dense scale of a MoE lies at somewhere between its activated parameters and its total parameters. Under the current dense Chinchilla setting, designing MoEs from the same scale dense backbone surely makes its equivalent dense scale being away from the Chinchilla compute-optimal frontier. We will leave the full study on the compute-optimal frontier for MoE as a future work.

Vary number of MoE layers. Under this setting, we use 1.6B and 4.8B as the dense backbones and vary the number of MoE layers with last-2, every-4 and every-2. Figure 1 (right) shows that MoEs at the 1.6B and 4.8B backbone scales form speed-accuracy trade-off curves that outpform the 6.4B dense with a meaningful gap. The curves further suggest that MoEs constructed from more design choices interpolated onto the curves should outperform the dense baseline.

Table 3: The effectiveness of scaling number of experts on the 1.6B MoEs. All models are trained with the 6.4B dense Chinchilla budget on 256 TPU-v5e cores. Mesh shape is ordered as (`Data`, `Expert`, `Model`).

model	mesh shape	step time	CoreEN (0S)
1.6B/16E	(16, 16, 1)	1.07s	62.48
1.6B/64E	(4, 64, 1)	1.05s	63.68
1.6B/256E	(1, 256, 1)	1.07s	65.0

Table 4: The effectiveness of scaling number of experts on the 4.5B and 8.1B MoEs. All models are trained with the 12.6B dense Chinchilla budget on 1024 TPU-v4 cores. Mesh shape is ordered as (`Data`, `Expert`, `Model`).

model	mesh shape	step time	CoreEN (all)	MMLU	GSM8K
4.5B/64E	(8, 64, 1)	1.50s	58.60	26.68	6.75
4.5B/256E	(2, 256, 1)	1.50s	60.62	29.52	12.13
8.1B/64E	(8, 64, 1)	2.57s	59.60	28.39	8.64
8.1B/256E	(2, 256, 1)	2.60s	61.38	30.18	11.97

3.4 Effectiveness of the 3D sharding method

Our 3D sharding method brings two major benefits. First, it allows us to scale the number of experts without affecting train step time much. Second, it controls the communication overhead from routing in MoE layers within a healthy range.

Scale number of experts without affecting step time. Table 3 and Table 4 show the effectiveness of scaling number of experts with the 3D sharding method. We evaluate the 1.6B MoEs using the 6.4B scale train budget on 256 TPU-v5e cores and the 4.5B and 8.1B MoEs using the 12.6B scale budget on 1024 TPU-v4 cores. We see that scaling number of experts monotonically improves model performance on all three CoreEN, MMLU and GSM8K benchmarks while not affecting train step time much. To optimize model performance for training, the results suggest it would be beneficial to scale the number of experts to the limit of device efficiency[4]

[5]Existing work (Fedus et al. (2022); Du et al. (2022); Clark et al. (2022)) shows that scaling number of experts beyond 256 gives diminished return.

.

Controlled MoE communication overhead. We evaluate the dense-to-MoE step time increase across a wide range of model scales with 85M, 1.2B and 6.4B. Table 5 summaries the results. We can see that the 3D sharding method controls the dense-to-MoE step time increase within a healthy range, typically below 20%. It is worth noting that for one extreme setup where the dense model just fits into the device memory without sharding along the `Model` axis but MoEs would run out of memory and have to shard along the `Model` axis. In this case the step time increase would become much larger and we suggest either reducing batch size or using more devices to control the step time increase.

Our sharding method vs. other solutions. We compare our sharding method to two other solutions under the GShard MoE implementation. 1) The conventional 2D sharding method. When the number of experts is smaller than the number of devices, we have to shard the expert dimension of the MoE weights and activations along both `Data` and `Model` axes, leading to suboptimal efficiency; 2) Another 2D method is to pad the expert dimension to the number of devices so it can be sharded only along `Data`. This is equivalent to adding more experts that would never process any tokens. We evaluate the three methods using a 1.6B/64E model trained with the 6.4B budget on 256 TPU-v5e cores. Table 6 shows the step time comparisons.

Table 5: Step time increase from dense to MoEs across a wide range of model scales. Mesh shape is ordered as (`Data`, `Expert`, `Model`).

model	devices	batch size	mesh shape	step time	delta
85M	64 TPU-v5e	25k	(64, 1, 1)	0.50s	-
85M/32E	64 TPU-v5e	25k	(2, 32, 1)	0.53s	+6.0%
85M/64E	64 TPU-v5e	25k	(1, 64, 1)	0.54s	+8.0%
1.2B	256 TPU-v5e	1M	(256, 1, 1)	0.69s	-
1.2B/256E	256 TPU-v5e	1M	(1, 256, 1)	0.81s	+17.4%
6.4B	512 TPU-v4	1M	(1, 256, 1)	1.69s	-
6.4B/64E	512 TPU-v4	1M	(4, 64, 1)	1.92s	+13.6%
6.4B/256E	512 TPU-v4	1M	(1, 256, 1)	1.86s	+10.1%

Table 6: Effectiveness of our final sharding specifications compared to naive 2D sharding or 2D sharding with padding. All numbers are measured on 256 TPU-v5e cores using a 1.6B/64E MoE.

	$\text{params}_{\text{total}}$	step time
2D sharding (naive)	23.8B	1.98s
2D sharding (padding)	46.6B	1.07s
3D sharding	23.8B	1.05s

44.93% 14.87% 40.19%

3.5 Supervised Fine-Tuning Results

We also study MoE and dense models in the context of instruction-finetuing. Specifically, we finetune the 8.1B/256E MoE and the 12.6B dense pretrain models on an internal dataset to ensure the model is capable of instruction following. Following Dettmers et al. (2024), we compare the quality of these two models by directly comparing outputs given the same prompt set. To realize this, we prompt GPT-4 to decide the better response or to announce a tie. Subsequently, the win/loss ratio is calculated and presented in Table 7. The 8.1B/256E MoE shows clear advantage over the 12.6B dense, demonstrating the performance gain on pretrain transfers to SFT.

3.6 Other Ablations

Router z-loss stabilizes training. When training large MoEs, e.g. 6.4B/64E, we sometimes hit training instability at the first a few thousand steps. The router z-loss (Zoph et al. (2022)) stabilizes training and does not show any negative impact on model performance and we adopt it in all the model training.

A lower expert capacity for training. We evaluated the modeling trick of using 1.25 train capacity and 2.0 eval capacity in ST-MoE (Zoph et al. (2022)) and we found it doesn't help much on the speed-accuracy trade-off curve. For instance, under the 1.6B/256E setting, the trick improves model speed by <5%, at the expense of hurting CoreEN by 0.2%. We use a train capacity of 2.0 in our experiments.

Naive second expert routing works. When selecting the second expert, we found the naive routing that always picks the second best expert works as good as the random routing method used in GShard (Lepikhin et al. (2021)). We adopt the naive method in all the experiments.

Table 8: Train compute budget and settings for the main models. Mesh shape is ordered as (`Data`, `Expert`, `Model`).

model	train tokens	batch size	TPU devices	mesh shape	step time
6.4B	128B	1M	512 v4	(1, 256, 1)	1.69s
1.6B/256E	264B	1M	512 v4	(1, 256, 1)	0.82s
4.8B/256E	153B	1M	512 v4	(1, 256, 1)	1.41s
12.6B	252B	2M	1024 v4	(1, 512, 1)	2.94s
4.5B/256E	494B	2M	1024 v4	(2, 256, 1)	1.50s
8.1B/256E	285B	2M	1024 v4	(2, 256, 1)	2.60s
29.6B	592B	2M	1024 v4	(1, 512, 1)	6.56s
6.4B/64E	2128B	2M	1024 v4	(8, 64, 1)	1.85s

4 Conclusion

In this work, we revisited the step-accuracy trade-off comparisons between MoE and dense LLMs under a challenging setting that favors dense models to MoEs. We first use train step time as a measure for model's computation cost, taking communication overhead in MoE implementations into consideration. Then we adopt the Chinchilla compute-optimal setting, which is optimized for dense model training, as our train compute budget to design comparisons at various scales. We also adopt a 3D sharding method that effectively reduces the communication overhead of MoE implementa-

References

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

Berant et al. [2013]
Jonathan Berant, Andrew K. Chou, Roy Frostig, and Percy Liang.
Semantic parsing on freebase from question-answer pairs.
In *Conference on Empirical Methods in Natural Language Processing*, 2013.
URL https://api.semanticscholar.org/CorpusID:6401679.

Bisk et al. [2019]
Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi.
Piqa: Reasoning about physical commonsense in natural language, 2019.

Chen et al. [2023]
Tianlong Chen, Xuxi Chen, Xianzhi Du, Abdullah Rashwan, Fan Yang, Huizhong Chen, Zhangyang Wang, and Yeqing Li.
Adamv-moe: Adaptive multi-task vision mixture-of-experts.
In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 17346–17357, October 2023.

Clark et al. [2022]
Aidan Clark, Diego de las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, George van den Driessche, Eliza Rutherford, Tom Hennigan, Matthew Johnson, Katie Millican, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Jack Rae, Erich Elsen, Koray Kavukcuoglu, and Karen Simonyan. Unified scaling laws for routed language models, 2022.

Clark et al. [2018]
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, 2018.

Cobbe et al. [2021]
Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman.
Training verifiers to solve math word problems, 2021.

Dai et al. [2024]
Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models, 2024.

Mobile v-moes: Scaling down vision transformers via sparse mixture-of-experts, 2023.

Dettmers et al. [2024] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer.
Qlora: Efficient finetuning of quantized llms.
Advances in Neural Information Processing Systems, 36, 2024.

Du et al. [2022] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P Bosma, Zongwei Zhou, Tao Wang, Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc Le, Yonghui Wu, Zhifeng Chen, and Claire Cui.
GLaM: Efficient scaling of language models with mixture-of-experts.
In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5547–5569. PMLR, 17–23 Jul 2022.
URL https://proceedings.mlr.press/v162/du22c.html.

Fedus et al. [2022] William Fedus, Barret Zoph, and Noam Shazeer.
Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022.

Hendrycks et al. [2021] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt.
Measuring massive multitask language understanding.
Proceedings of the International Conference on Learning Representations (ICLR), 2021.

Hoffmann et al. [2022] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre.
Training compute-optimal large language models, 2022.

Jiang et al. [2024] 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, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed.
Mixtral of experts, 2024.

Komatsuzaki et al. [2023]

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=T5nUQDrM4u.

Kudo and Richardson [2018]

Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, 2018.

Lepikhin et al. [2021]

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. {GS}hard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

Lin et al. [2024]

Bin Lin, Zhenyu Tang, Yang Ye, Jiaxi Cui, Bin Zhu, Peng Jin, Jinfa Huang, Junwu Zhang, Munan Ning, and Li Yuan. Moe-llava: Mixture of experts for large vision-language models, 2024.

Loshchilov and Hutter [2019]

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

Mustafa et al. [2022]

Basil Mustafa, Carlos Riquelme Ruiz, Joan Puigcerver, Rodolphe Jenatton, and Neil Houlsby. Multimodal contrastive learning with LIMoe: the language-image mixture of experts. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=Qy1D9JyMBg0.

Paperno et al. [2016]

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context, 2016.

In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.
URL https://openreview.net/forum?id=NGPmH3vbAA_.

Sakaguchi et al. [2019]
Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi.
Winogrande: An adversarial winograd schema challenge at scale, 2019.

Sap et al. [2019]
Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi.
Socialiqa: Commonsense reasoning about social interactions, 2019.

Sennrich et al. [2016]
Rico Sennrich, Barry Haddow, and Alexandra Birch.
Neural machine translation of rare words with subword units, 2016.

Shazeer [2020]
Noam Shazeer.
Glu variants improve transformer, 2020.

Shazeer et al. [2017]
Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean.
Outrageously large neural networks: The sparsely-gated mixture-of-experts layer.
In *International Conference on Learning Representations*, 2017.
URL https://openreview.net/forum?id=B1ckMDqlg.

Shen et al. [2023]
Sheng Shen, Le Hou, Yanqi Zhou, Nan Du, Shayne Longpre, Jason Wei, Hyung Won Chung, Barret Zoph, William Fedus, Xinyun Chen, Tu Vu, Yuexin Wu, Wuyang Chen, Albert Webson, Yunxuan Li, Vincent Zhao, Hongkun Yu, Kurt Keutzer, Trevor Darrell, and Denny Zhou.
Mixture-of-experts meets instruction tuning:a winning combination for large language models, 2023.

Su et al. [2023]
Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu.
Roformer: Enhanced transformer with rotary position embedding, 2023.

Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom.
Llama 2: Open foundation and fine-tuned chat models, 2023.

Xu et al. [2021]

Yuanzhong Xu, HyoukJoong Lee, Dehao Chen, Blake Hechtman, Yanping Huang, Rahul Joshi, Maxim Krikun, Dmitry Lepikhin, Andy Ly, Marcello Maggioni, Ruoming Pang, Noam Shazeer, Shibo Wang, Tao Wang, Yonghui Wu, and Zhifeng Chen.
Gspmd: General and scalable parallelization for ml computation graphs, 2021.

Zellers et al. [2019]

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi.
Hellaswag: Can a machine really finish your sentence?, 2019.

Zhang and Sennrich [2019]

Biao Zhang and Rico Sennrich.
Root Mean Square Layer Normalization.
In *Advances in Neural Information Processing Systems 32*, Vancouver, Canada, 2019.
URL https://openreview.net/references/pdf?id=S1qBAf6rr.

Zhao et al. [2023]

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li.
Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023.

Zhou et al. [2022]

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, zhifeng Chen, Quoc V Le, and James Laudon.
Mixture-of-experts with expert choice routing.
In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 7103–7114. Curran Associates, Inc., 2022.

Zoph et al. [2022]

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus.
St-moe: Designing stable and transferable sparse expert models, 2022.