OpenMoE: An Early Effort on Open Mixture-of-Experts Language Models

Fuzhao Xue [1] **Zian Zheng** [1] **Yao Fu** [2] **Jinjie Ni** [1] **Zangwei Zheng** [1] **Wangchunshu Zhou** [3] **Yang You** [1]

Abstract

To help the open-source community have a better understanding of Mixture-of-Experts (MoE) based large language models (LLMs), we train and release OpenMoE, a series of fully open-sourced and reproducible decoder-only MoE LLMs, ranging from 650M to 34B parameters and trained on up to over 1T tokens. Our investigation confirms that MoE-based LLMs can offer a more favorable cost-effectiveness trade-off than dense LLMs, highlighting the potential effectiveness for future LLM development.

One more important contribution of this study is an in-depth analysis of the routing mechanisms within our OpenMoE models, leading to three significant findings: Context-Independent Specialization, Early Routing Learning, and Drop-towards-the-End. We discovered that routing decisions in MoE models are predominantly based on token IDs, with minimal context relevance. The token-to-expert assignments are determined early in the pre-training phase and remain largely unchanged. This imperfect routing can result in performance degradation, particularly in sequential tasks like multi-turn conversations, where tokens appearing later in a sequence are more likely to be dropped. Finally, we rethink our design based on the above-mentioned observations and analysis. To facilitate future MoE LLM development, we propose potential strategies for mitigating the issues we found and further improving off-the-shelf MoE LLM designs.

1. Introduction

Large Language Model (LLM) has exhibited remarkable performance on various NLP tasks (Raffel et al., 2020; Li et al., 2022), and has even become a part of our daily lives through chatbot applications such as ChatGPT, Bard, and Copilot. However, LLMs are computationally expensive in both training and inference. As LLMs become increasingly prevalent, enhancing their performance without proportionally increasing computational resources is a critical challenge. In response to this challenge, Riquelme et al. (2021); Fedus et al. (2021) proposed the Mixture-of-Experts (MoE) to scale up the trainable parameters of the transformer with little additional computation overhead. Recent advancements in MoE-based language models, such as GLaM (Du et al., 2022) and ST-MoE (Zoph et al., 2022) have demonstrated superior performance in various tasks. However, before the release of OpenMoE, there were few open-sourced MoE language models trained with trillion-level diverse datasets.

In this work, we set forth three primary goals: (1) To offer a first-attempt solution in detail for training a decoder-only MoE model within the existing framework of training LLMs. (2) To perform an in-depth analysis of the MoE routing mechanisms, thereby providing the research community with deeper insights into the behaviors and potential limitations of MoE-based LLMs. (3) To pave the way for future MoE LLM development. Through this early endeavor, we aim to stimulate and accelerate the growth of the open-source MoE community.

Releasing OpenMoE. First, we release OpenMoE, a series of open-sourced MoE-based LLMs, including: (1) OpenMoE-Base/16E: a small model with 0.65B parameters for debugging purposes. 16E means 16 experts per MoE layer; (2) OpenMoE-8B/32E: this variant features 8B parameters in total, activating around 2B parameters per token in Transformer blocks, and is pre-trained on over 1 trillion tokens; (3) OpenMoE-8B/32E-Chat, a chat version of OpenMoE-8B/32E, fine-tuned with a 100K subset of the WildChat (Anonymous, 2024) dataset; (4) OpenMoE-34B/32E: a larger scale model, activating 6B parameters per token in Transformer blocks and trained with 200B tokens, serving as a testament to the scalability of our approach. Detailed configuration can be found in Appendix F Our OpenMoE-8B/32E models achieved comparable performance with OpenLLaMA-3B (Geng & Liu, 2023) and TinyLLaMA-1.1B (Zhang et al., 2024), two dense open LLMs used higher training cost. Notably, On the MT-Bench (Zheng et al., 2023), OpenMoE-8B/32E-

[1]National University of Singapore [2]University of Edinburgh [3]ETH Zurich. Correspondence to: Fuzhao Xue <f.xue@u.nus.edu>.

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

Chat outperformed the two dense LLMs significantly on the single-turn conversation. In addition, we release 5 intermediate checkpoints of OpenMoE-8B/32E, each trained with 200B more tokens than the previous one, to support and encourage future research. Section 2 and 3 will discuss the design, training details, and evaluation results of OpenMoE.

Exploring Advanced Training Strategies. As part of our research endeavor, we are committed to exploring more advanced Techniques in LLM training: (1) Different from the common practice of training models on in-house or text-dominated open-sourced data, we train OpenMoE with a substantial proportion of code, constituting up to 52.25% during the early stages of pre-training; (2) Moving beyond the conventional next-token prediction training objective, we investigate UL2 training objective (Tay et al., 2022a), motivated by its proven effectiveness in previous work (Anil et al., 2023) and its good alignment with coding data (Bavarian et al., 2022). We acknowledge that the performance of our model, while acceptable, does not significantly exceed our expectations, which may be attributed to some sub-optimal design choices. Nevertheless, we believe that this exploratory work offers substantial value to the open-source community, particularly in assessing the potential and effectiveness of these under-explored techniques.

Studying MoE Routing In-depth. While MoE is effective, there remains a lack of study on why MoE performs well. From a high level, MoE introduces more trainable parameters than its dense counterpart. To keep the FLOPs fixed when scaling the number of parameters, MoE applies a routing layer that sparsely and adaptively assigns each token to a few experts. This process of sparse expert selection is crucial to MoE's functionality. Unfortunately, despite existing pieces of literature briefly visualizing the routing decison (Shazeer et al., 2017; Lewis et al., 2021; Zoph et al., 2022; Mustafa et al., 2022; Riquelme et al., 2021), we still don't have a clear understanding of how the router works and how the routing decision impacts the results in MoE models, especially for the post-ChatGPT LLMs trained on a mixture of datasets from diverse domains. In this work, we study this problem based on various taxonomies, including domain, language, task, and token. Our key findings are as follows: (1) **Context-independent Specialization**: MoE tends to simply cluster tokens based on similar token-level semantics, implying that, regardless of context, a certain token is more likely to be routed to a certain expert; (2) **Early Routing Learning**: Token ID routing specialization is established early in pre-training and remains largely fixed, resulting in tokens being consistently processed by the same experts throughout the training; (3) **Drop-towards-the-End**: Since each expert has a fixed max capacity, tokens appearing later in the sequence face a higher risk of being dropped if the expert is already at capacity. This issue is more severe in instruction-tuning datasets. These datasets often exhibit a domain gap compared to the pre-training data, meaning that the balanced token assignment strategies established and solidified during early pre-training may not be samely effective in instruction-tuning scenarios. This is concerning as instruction data plays an important role in deploying LLMs to real-world applications. Section 4 discusses the above phenomenons in detail.

Rethinking Our Mistakes and Proposing Potential Solutions. In retrospect, our project encountered several mistakes and made sub-optimal decisions (*e.g.,* aggressive data mixture), as detailed in Section 5. As an early open-source effort, we believe that sharing these experiences and insights is crucial, perhaps even more important than solely focusing on successful strategies. Based on our empirical findings during training and subsequent visualization analysis (Section 4), we have developed a set of potential solutions. We sincerely hope these insights can help the community develop better models in the future.

The structure of this paper mirrors the lifecycle of the OpenMoE project, encompassing all its phases. This includes the initial design (Section 2), training and evaluation (Section 3, in-depth analysis (Section 4), and a rethinking of the OpenMoE project (Section 5).

2. Designing OpenMoE

2.1. Pre-training Dataset: More Code than Usual

First, we introduce our initialized design of OpenMoE models regarding the pre-training data, model architecture, training objective, and supervised fine-tuning data.

Modern LLMs are usually trained by a combination of datasets from various domains, *i.e.,* data mixture (Brown et al., 2020; Rae et al., 2021; Hoffmann et al., 2022; Chowdhery et al., 2022; Touvron et al., 2023). Except for the LLMs customized towards coding (*e.g.,* StarCoder (Li et al., 2023), CodeLLaMA (Roziere et al., 2023)), most existing models' pre-training data is dominated by text data. For instance, the sampling rate of the GitHub dataset is only 4.5% for LLaMA (Touvron et al., 2023). However, we argue that the code data is highly important for two reasons. First, the code data is likely to improve the ability of complex reasoning with chain-of-thought (Fu & Khot, 2022). More importantly, different from natural language, which is sometimes blurry and easy to misunderstand, code is always precise. This enables code to be a more efficient language for machines to convey information concisely without misunderstanding between different (embodied) AI agents, and as a result, code has great potential to dominate LLM communications in real-life applications. Therefore, we design a more code-dominated pre-training data mixture with over 50% code. Specifically, we extracted 50% of data from the RedPajama (Computer, 2023) and 50% of data from the

duplication version of The Stack (Kocetkov et al., 2022). Our experimental results show that the version I data mixture might be a bit aggressive in its code proportion. We fix these issues at the later stage of pre-training, please see the following Section 3.1 for details. More detailed data mixture can be found in Appendix C.

2.2. Model Architecture: Decoder-only ST-MoE

Tokenizer. We applied umT5 (Chung et al., 2023) tokenizer with 256K vocab size for two reasons: (1) umT5 tokenizer with a large multi-lingual vocab supports low-resource language better than the tokenizers using a small vocab (*e.g.,* LLaMA tokenizer with 32K vocab); (2) comparing to some old tokenizers, such as BERT (Kenton & Toutanova, 2019) and T5 (Raffel et al., 2020) tokenizer, umT5 tokenizer has byte fallback feature to support out-of-vocab tokens better.

Model Architecture. We generally follow ST-MoE (Zoph et al., 2022) for our model architecture and routing design to ensure training stability, which is extremely important when training larger models. Specifically, we use token-choice routing for better compatibility with decoder-only LLM. Top-2 selection is used to improve the performance with a small computational overhead. Residual MoE design is also adopted. Note that, inspired by the findings in ViT-MoE (Riquelme et al., 2021), to achieve a better cost-effective trade-off, we use MoE-based Transformer blocks in an interleaved manner instead of placing MoE in every Transformer block. Last, we follow Zoph et al. (2022) and Shazeer et al. (2017) to use MoE load balance loss to ensure a balanced number of tokens assigned to different experts. Router z-Loss (Zoph et al., 2022) is also adopted to improve the training stability. More detailed model architecture description can be found in Appendix D.

2.3. Training Objective: UL2 and CasualLM

Instead of adopting vanilla casual language modeling (CasualLM) directly, we explore UL2 (Tay et al., 2022b), a more diverse language model pre-training objective combining span corruption (SpanCorrupt) and prefix language modeling (PrefixLM) (Raffel et al., 2020). It is noteworthy that the SpanCorrupt in UL2 is more diverse than the vanilla SpanCorrupt because it mixes various span lengths and corruption rates. We have two reasons to explore UL2 in OpenMoE. First, UL2 has shown promising results in PaLM-2 (Anil et al., 2023). More importantly, the aggressive token masking is very similar to the code completion task in the real world, such as Copilot. Bavarian et al. (2022) also found that the similar filling-in-the-middle (FiM) objective can model the code better than the vanilla training objective. Since we used more code in our pre-training data mixture, adapting UL2 that covers FiM is a more reasonable choice intuitively. Our detailed UL2 training objective

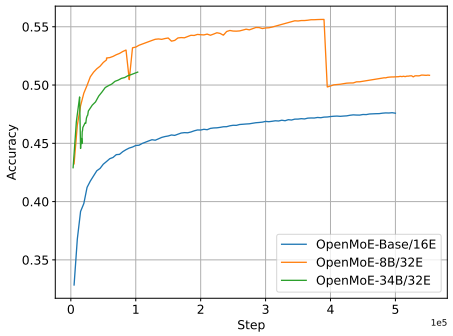


Figure 1: Token prediction accuracy of OpenMoE models. OpenMoE-8B/32E uses UL2 before 390K steps and falls back to CasualLM after 380K steps. OpenMoE-34B/32E uses UL2 until 50B tokens.

configuration can be found in Appendix E. We faced some difficulties when training with UL2 in OpenMoE, which will be discussed in Section 3.1.

2.4. Supervised Fine-tuning

Although alignment is not the focus of this project, we conduct supervised fine-tuning (SFT) with a subset of the open-sourced WildChat dataset (Anonymous, 2024) to study the behavior of the MoE model before and after SFT. We only pick the data from GPT-4 in WildChat because of the lack of computation resources at the late stage of OpenMoE development. The subset includes 58K conversations and each conversation includes 1.8 turns on average.

2.5. Other Designs

Following recent LLMs, we adopt RoPE (Su et al., 2024) for position embedding and SwiGLU (Shazeer, 2020) for activation function for FFNs in both dense and MoE Transformer blocks. More detailed model configuration and training hyperparameters for OpenMoE models can be found in Appendix F. We applied data parallelism, tensor parallelism (Xu et al., 2021; Shoeybi et al., 2019), and expert parallelism (Lepikhin et al., 2020) for training models at scale. We train OpenMoE models on Google Cloud TPU with 64 to 512 v3 chips depending on the availability.

3. Training OpenMoE

Before scaling up, we conducted an initial evaluation of our design decisions using the OpenMoE-Base/16E model in Appendix G. Although the experiments at this scale may not generalize well to larger scales due to computational resource constraints, we observe reasonable results that provide early insights and verify the correctness of our design.

3.1. Training Progress

UL2 Saturation During training, we found that, although UL2 can help the model to learn faster at the early stage of

Table 1: Results on HumanEval (Pass@1). We also report the number of training tokens from the code domain.

Model	Act. Params	Total Tokens	Code Tokens	HumanEval
TinyLLaMA-1.1B	0.9B	3.0T	900B	9.1
OpenLLaMA-3B	2.9B	1.0T	59B	0
OpenMoE-8B/32E	2.1B	1.1T	456B	9.8
OpenMoE-34B/32E	6.4B	0.2T	70B	10.3

Table 2: Results on TriviaQA (Exact Match). We also report the number of training tokens from Wikipedia because the commonsense questions in TriviaQA have a relatively close relation with Wikipedia data.

Model	Act. Params	Total Tokens	Text Tokens	Wiki Tokens	TriviaQA
TinyLLaMA-1.1B	0.9B	3.0T	2.1T	75B	11.2
OpenLLaMA-3B	2.9B	1.0T	991B	24B	29.7
OpenMoE-8B/32E	2.1B	1.1T	644B	58B	32.7
OpenMoE-34B/32E	6.4B	0.2T	130B	14B	31.3

training, it is easier to saturate at the later training stage of OpenMoE-8B/32E. As shown in Figure 1, if we zoom in, we can find that OpenMoE-8B/32E improves very slowly from 35K to 39K steps. We suggest that this may be because, although UL2 is more diverse, the SpanCorrupt is still relatively easy compared to CasualLM. Therefore, we fall back to CasualLM after 390K steps (780B) tokens. In addition, since code data aligns better with UL2 and our initial code data mixture is relatively aggressive, we also decreased our code data sampling ratio to 15%. The Second version data mixture is reported in Table 7.

Obviously, in Figure 1, after 780B tokens, there is a significant drop in the token prediction accuracy after 390K steps for OpenMoE-8B/32E. This is caused by the more difficult CasualLM objective and less easy code data. Note that, although we encountered a saturation issue at the later stage of OpenMoE-8B/32E training, we think such an easy-to-hard curriculum may be helpful for LLM training. Therefore, we still adapted UL2 for 25K steps (50B tokens) in OpenMoE-34B/32E. We used a relatively moderate code-heavy data mixture in OpenMoE-34B/32E. In Table 7, we utilize 35% of code data in total. Due to the computation resource limitation, we train OpenMoE-34B/32E with only 200B tokens to verify its scalability. We leave training a large-scale OpenMoE with more tokens as future work if possible.

3.2. Evaluation on Benchmarks

3.2.1. Raw Model Evaluation

Before all, we highlight that we did not hack the benchmarks at all and the pre-training is purely on the open-sourced datasets mentioned above. Since our model is relatively small in terms of training budget, we mainly evaluate the raw model on established but not that hard benchmarks, *i.e.,* TriviaQA (Joshi et al., 2017), HumanEval (Chen et al., 2021), WMT16-En-Ro (Bojar et al., 2016), BigBench-Lite (24 tasks) (bench authors, 2023), and a subset of the lm-evaluation-harness collection (Gao et al., 2023) with 13 tasks. For popular but relatively challenging benchmarks like 5-shot MMLU (Hendrycks et al., 2020), our OpenMoE-8B/32E achieves around 26.2% accuracy, which means the model is almost randomly guessing from the four options. We mainly compare with the open-sourced models with more training cost, *i.e.,* TinyLLaMA-1.1B (Zhang et al., 2024) and OpenLLaMA-3B (Geng & Liu, 2023). On BigBench-Lite, we also compare with GPT-3 (Brown et al., 2020), Big-G (bench authors, 2023) and Big-G-Sparse (bench authors, 2023). Big-G and Big-G-Sparse are two sets of Google in-house models evaluated on BigBench-Lite. Big-G-Sparse models are MoE-based Transformers.

We first report our results on Commonsense QA (TriviaQA), Coding (HumanEval), and Low-resource Machine Translation (WMT16 En-Ro). We think these three benchmarks are meaningful for us because (1) Commonsense is to check whether OpenMoE can memorize more given its efficient parameter scaling advantage; (2) Coding is important because of its prevalent use cases in solving coding-related user prompts, LLM as agents, and embodied AI; (3) Low-resource Machine Translation is important because we want to share the benefits of foundation models to everyone on earth. In Table 2, OpenMoE-8B/32E outperforms baselines clearly with less training cost (Activated parameter$\times$Total Training tokens). Also, note that TinyLLaMA-1.1B performs significantly worse than other models on TriviaQA although it has a comparable training cost with OpenLLaMA-3B. Therefore, this highlights the importance of the number of parameters to keeping knowledge in LLMs, which also indicates the significance of using MoE.

In Table 1, OpenMoE models achieve better performance than baselines. OpenMoE-34B/32E only used 70B code data (The Stack and GitHub data), while it still performs relatively well on HumanEval, which shows the scalability of OpenMoE, although we don't have enough resources to train it until the end. OpenLLaMA-3B struggles on HumanEval because consecutive whitespaces are treated as one, contradicting the Python syntax (Nijkamp et al., 2023).

Table 3 shows our results on WMT16 En-Ro translation task.

Table 3: Results on WMT16 En-Ro (BLEU score). We also report the number of explicit multi-lingual tokens in the pre-training dataset, *i.e.,* the multi-lingual version of Wikipedia from the RedPajama dataset.

Model	Act. Params	Total Tokens	Multi-lingual Tokens	WMT16 En-Ro
TinyLLaMA-1.1B	0.9B	3.0T	75B	2.6
OpenLLaMA-3B	2.9B	1.0T	24B	1.9
OpenMoE-8B/32E	2.1B	1.1T	38B	3.1
OpenMoE-34B/32E	6.4B	0.2T	9B	3.4

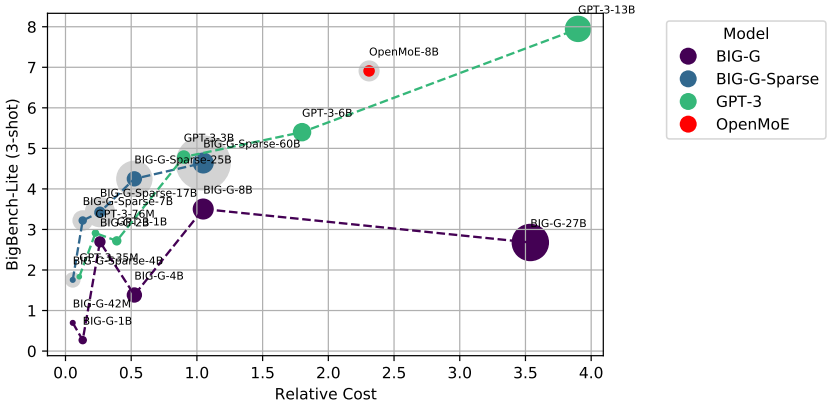


Figure 2: Results on BigBench-Lite. The relative cost is computed based on multiplying activated parameters in the Transformer and the number of training tokens. The size of the color dots denotes the number of activated parameters, and the size of the shadow denotes the number of total parameters for MoE models.

Table 4: Average scores on MT-Bench.

Model	MT-Bench		
	1st Turn	2nd Turn	Avg
GPT-J-6B	2.51	2.35	2.43
TinyLLaMA-1.1B	4.08	2.54	3.31
OpenLLaMA-3B	4.36	**3.62**	**3.99**
OpenMoE-8B/32E	**4.69**	3.26	**3.98**

Note that our model did not include much multi-lingual data intentionally. Most multi-lingual data is from the multi-lingual Wikipedia in the RedPajama, which is also used in TinyLLaMA-1.1B and OpenLLaMA-3B. However, OpenMoE models still show better results than baselines, which potentially highlights the importance of umT5 tokenizer.

In Figure 2, the relative cost is computed based on multiplying activated parameters (Act. Params) in Transformer blocks and the number of training tokens. The size of the color dots denotes the number of activated parameters, and the size of the shadow denotes the number of total parameters for MoE models. We can observe that OpenMoE achieved a better cost-effectiveness trade-off on BigBench-Lite, in terms of both training and inference cost. In addition to BigBench-Lite, we also evaluate OpenMoE on the 13 tasks from the LM-Evaluation-Harness collection. Detailed results can be found in Appendix I.

3.2.2. Chat Model Evaluation

We further evaluate our model on MTBench, an established ChatBot benchmark that can examine models comprehensively. We report both single-turn and multi-turn results in Table 4 and Appendix J. We can observe that OpenMoE outperforms baselines by a large margin on the single-turn results, especially on coding tasks. However, OpenMoE's performance drops more on the second turn, which results in worse multi-turn results in Figure 9b. We found that this probably be caused by the token drop of a long sequence. Please see the following Section 4 for a detailed analysis.

4. Analyzing OpenMoE

We generally think MoE is an effective way to scale parameters up with a fixed computation budget. However, we have little idea about what the experts in MoE specialize in. In this section, we conduct an in-depth analysis of OpenMoE in multiple aspects to study the routing behavior.

4.1. What are the Experts Specializing in?

Does MoE specialize in domain level? We first visualize the routing decision of the tokens from different subsets in the RedPajama dataset. Note that all visualization results are from the third MoE layer by default because we did not observe significant differences across layers. We can observe that the tokens from different subsets (*i.e.,* domains) are uniformed distributed on the plot. That is, although E_{21} slightly prefers code tokens and E_{10} like books a little, most experts in MoE are not specialized based on the domains.

Does MoE specialize in language level? We move forward toward finer-grain data to check whether MoE specializes in

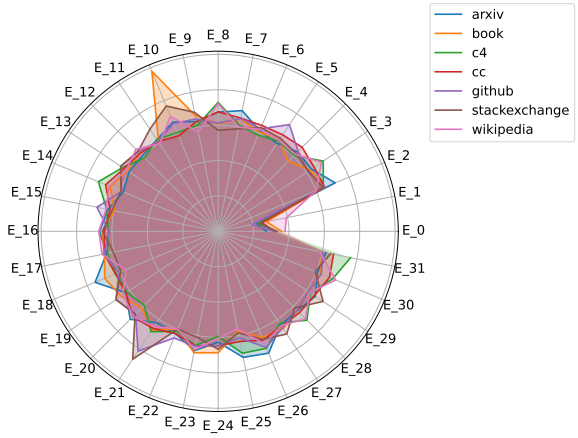


Figure 3: Visualization of the routing decision on the RedPajama dataset. E_i denotes the ratio of tokens routed to i_{th} expert.

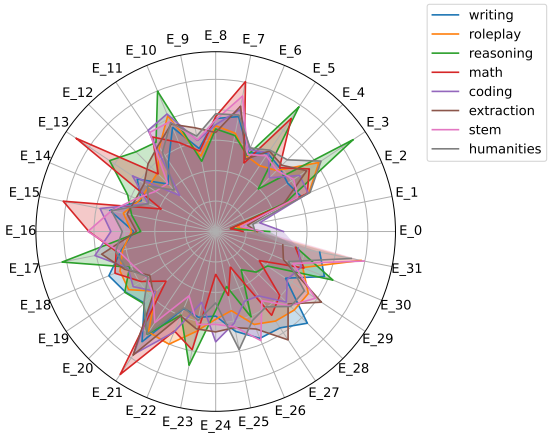


Figure 4: Visualization of the routing decision on MT-Bench. We adopt the conversation history when evaluating OpenMoE MT-Bench as the visualization data source. E_i denotes the ratio of tokens routed to the i_{th} expert.

4 different coding languages (*i.e.,* Assembly, Blitzmax, Java, and Python) and 12 different natural languages. We found that there is a relatively clear specialization among different experts, especially in natural languages. For instance, zh-cn (Chinese, Simplified) and zh-tw (Chinese, Traditional) both have a strong preference for E_5 and E_{16}; ja (Japanese), and ko (Korean) both prefer E_{14}. More detailed visualization can be found in Appendix K.

Does MoE specialize in task level? Based on the findings above, finer-grained data has clearer expert specialization observation. We then visualize the routing decision on MT-Bench conversation data in Figure 4. We can see a similar specialization as above, especially for the math data. We suggest that the main reason is that the math tasks include more special tokens than other tasks.

Does MoE specialize in Position ID? Routers in MoE make decisions based on the token representations. The token representations are from token embeddings and position embeddings. We thus visualize the routing decisions on different positions in Appendix L and observe:(1) there are indeed some specializations in different Position IDs; (2) consecutive positions prefer similar experts, such as the E_{10} and E_{19} in Figure 12b.

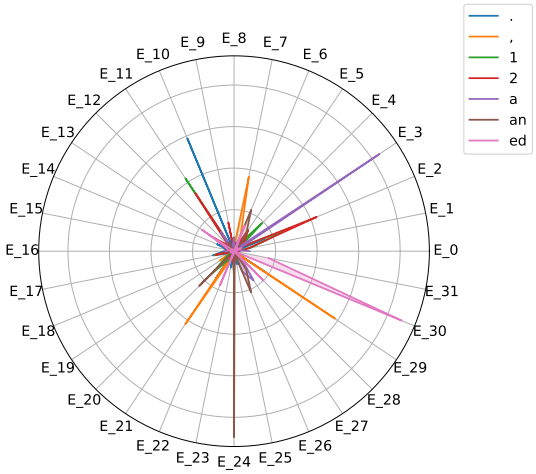


Figure 5: Visualization of the routing decision at different Token IDs. E_i denotes the ratio of tokens routed to the i_{th} expert.

Does MoE specialize in Token ID? Since we are using the umT5 tokenizer, tokens from different languages usually have different token IDs. Therefore, we further study whether the router in MoE mainly makes its decisions based on the Token ID. We visualize the routing decisions of a few representative tokens in Figure 5. All these tokens show a very strong specialization on only a few experts. This is a very interesting finding because the tokens with the same Token ID have very diverse contexts in different sentences. For instance, the token "ed" can be the suffix of many different words, *e.g.,* "preferred", and "led". The token "an" can also be part of "an apple" or "another". However, all these tokens have very strong specialization on only a few fixed experts. That means, MoE simply routes based on the Token ID instead of high-level semantics. We name this observation as **Context-independent Specialization** in the following sections. To verify that the Context-independent Specialization also exists for other Token IDs, we plot the routing decision standard deviation in Appendix N.

4.2. Token Specialization Study

Are experts clustering similar tokens? As we discussed above, the tokens with the same Token ID are always routed to the same expert no matter what the context is, *i.e.,* Context-independent Specialization. We thus investigate whether the experts prefer the Token IDs corresponding to the tokens with similar low-level semantics. We list the top 10 favorite tokens for each expert in Table 5. We can observe that similar tokens are clustered in experts. For

Table 5: Top Tokens selected by each expert. The ID denotes the Expert ID.

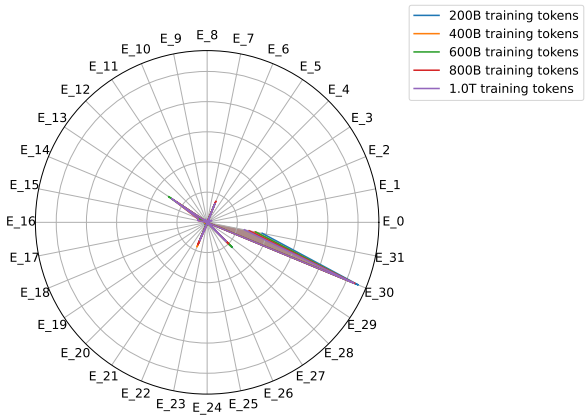
ID	Top Tokens
0	\n , ‘ , ’ , s , - , $, y , _ , , 2
1	\n , 1 , , 2 , \\ , S , . , - , C , {
21	, , and , , . , \n , = , \t , the , , n
30	} , ed , d , have , ing , , , has , s ,” , had
31	to , can , s , of , ing , will , not , e , ed , would

instance, “can”. “will”, and “would” are all in expert 31. “have”. “has”, and “had” are all included in expert 30. This visualization can also explain many observations above. An example is that, in most figures above, we can find most coding and math data prefer expert 21. Here it reveals the real reason. Expert 21 has a strong preference for “=”, “and”, and “\n”, which appear more frequently in math and code.

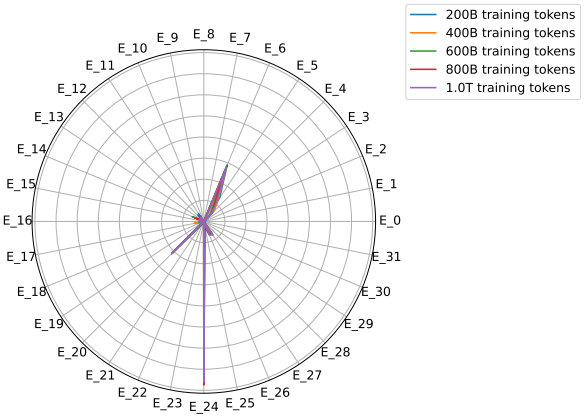
When did the model learn the specialization? According to the Context-independent Specialization observed above, the model is not learning how to route based on high-level semantics. Therefore, we raise another question, when did the model learn and fix the routing decision for the tokens? We compare the routing decisions of different OpenMoE intermediate checkpoints in Figure 6a and Figure 6b. We can see that the expert preferences are almost totally overlapped for different checkpoints, which means that the model has started to fix its routing at the very early stage of training. Even if we change the training data mixture (from 52.25% code to 20% code) and training objective (from UL2 to CasualLM), the routing decision is still fixed. We infer that the reason is that, when the token is usually assigned to one specific expert, the loss would increase a lot if the token is sent to another unseen expert, which pushes the model to assign the token back to the original expert. Therefore, the routing probably has been learned at the warmup stage or so, and kept throughout the whole following training stage.

4.3. Token Drop During Routing

Drop-towards-the-End In MoE models, we usually set a pre-defined max capacity C for every expert to ensure a balanced workload, which means each expert cannot process more than C tokens. This can ensure the throughput when training and deploying the MoE model with expert parallelism, *i.e.,* distributing different experts to different GPUs. However, this will also introduce an issue, the later tokens would be dropped if the previous tokens have filled the expert. In decoder-only MoE architecture, due to the auto-regressive nature, the later tokens in a sequence may be dropped more. For instance, if one expert prefers “\n” token, and a sequence starts with many “\n”s and also has a lot of “\n’s in the following output generated, the expert would be filled with “\n” tokens quickly and all other tokens appeared later, which should be assigned to this expert, would



(a) Token “ed” routing decision of different intermediate checkpoints.

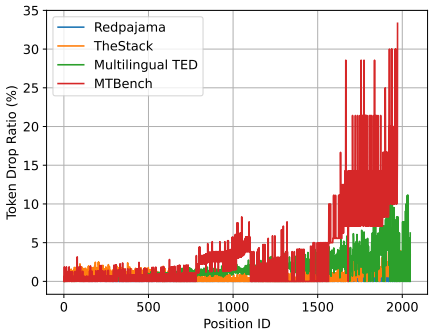


(b) Token “an” routing decision of different intermediate checkpoints.

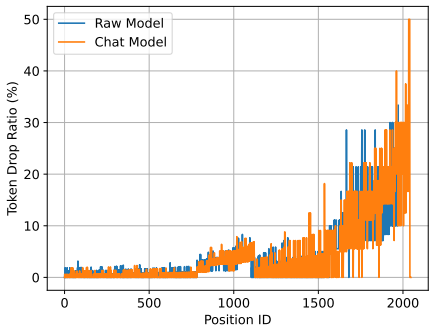
Figure 6: Visualization of token IDs’ routing decision of different intermediate checkpoints.

be dropped. To verify this, we visualize the ratio of tokens dropped at different position IDs. As shown in Figure 7a, the general pre-training datasets, *e.g.,* RedPajama and TheStack achieved balanced token assignment, only having a small proportion of tokens dropped, even for the Position ID after 1500. However, for multi-lingual and instruction-following datasets, a large ratio of tokens is dropped. We suggest the reason is, as we discussed above, the routing decision is fixed at the early stage of training and does not change anymore, so the load balance is also achieved based on the pre-training dataset. The instruction following data can be seen as a type of out-of-domain (OOD) data of the MoE router, which would induce an unbalanced token assignment so that many tokens appearing later would be dropped.

Can supervised fine-tuning with instruction-following data alleviate this Drop-towards-the-End issue? Since the Drop-towards-the-End issue is mainly caused by the OOD data, it is natural to think and study whether it is possible to convert the instruction-following data to in-domain data by tuning MoE with instruction datasets. Therefore, we compare the models before and after SFT in Figure 7b. We can see the models do not have a significant difference in the



(a) Different datasets.



(b) Before and after SFT.

Figure 7: Comparing the ratio of tokens dropped at different position IDs.

Drop-towards-the-End issue. This matches well with our insight above, *i.e.,* the routing behavior learned and fixed at the very early stage of LLM pre-training.

5. Rethinking OpenMoE

Working on this project is a long journey for authors. We indeed made some mistakes during design and development, but we also achieved some new insights in the analysis. We thus write down everything we found without any reservation in this paper to help future practitioners. In this section, we discuss how to train a better model in the future.

How much code shall we use? To be honest, we do not have a very precise answer. Conducting an ablation study is extremely expensive because of the cost of pre-training LLM at scale. The conclusion may also strongly depend on the model size and data quality. However, according to our observation, over 50% code looks too aggressive which may harm the abilities on text tasks, but considering the importance of writing code, we suggest using around 30% code as we used in OpenMoE-34B/32E.

Tokenizer Selection Our large tokenizer vocabulary introduces computation overhead at the last output layer after Transformer blocks. Although this overhead would become relatively small after scaling the Transformer model up, it is still valuable to make the tokenizer selection smarter. We conduct a quantitative analysis of the tokenizer. The detailed results can be found in Appendix Q. In general, umT5 tokenizer is indeed much better than LLaMA tokenizer on the multi-lingual dataset, especially on the low-resource language. It is also slightly better than LLaMA on the instruction-following data. However, it did not match well with our expectation that it could save more tokens for the code data. In addition, we observe that the token usage in both tokenizers is extremely long-tail distributed, which indicates that there is a large room to improve (Zhang et al., 2023). Since we only have a little multi-lingual data in pre-training, the computation cost of predicting the logits of those low-resource tokens is wasted. Based on our sub-optimal choice, we also need a solid tokenizer benchmark to help people evaluate tokenizers systematically. And we can then pick the best tokenizer before training the model.

More Efficient MoE Architecture According to our observation, MoE routing is almost context-independent (*i.e.,* Context-independent Specialization), we suggest that we can (1) remove the trainable router after warmup stage; (2) adopt parallel Transformer layer (Chowdhery et al., 2022; Wang & Komatsuzaki, 2021) computing FFN layer based on the input directly instead of the output of attention layer; (3) overlapping the attention layer computation and MoE layer all-to-all communication. (1) and (3) will improve the hardware utilization and (2) can enable (3) without performance drop when scaling up (Chowdhery et al., 2022).

Mix instruction-following data during pre-training warm-up to control load balance and alleviate Drop-towards-the-End. According to our results on multi-turn MT-Bench, it is important to alleviate the Drop-towards-the-End issue. To this end, the key is to achieve load balance on instruction-following data. Again, since the MoE learns and fixes the routing behavior at the early stage of pre-training, a straightforward solution is mixing the instruction-tuning data into the pre-training corpus during warm-up. This data mixing is not to align the model to learn how to follow instructions. Instead, we hope the model achieves the balanced token routing on instruction-tuning data, which paves the way to our final usage case of LLMs.

6. Conclusion

In this work, we explore how to train MoE for open-sourced communities. We achieved positive results that verified the effectiveness of MoE-based LLM in the post-ChatGPT stage. We disclosed all details, and our model is fully reproducible with the open-sourced code and data. More importantly, we conducted an in-depth analysis on our MoE-based LLM and found important "Context-independent Specialization" "Early Routing Learning" and "Drop-towards-the-End". We also rethink the mistakes we made and propose possible solutions for future developers. We sincerely hope this work can help the open-source community have a better understanding of MoE models. All the best!

Acknowledge

Yang You's research group is being sponsored by NUS startup grant (Presidential Young Professorship), Singapore MOE Tier-1 grant, ByteDance grant, ARCTIC grant, SMI grant, Alibaba grant, and Google grant for TPU usage.

Impact Statement

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

References

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Anonymous. (inthe)wildchat: 570k chatGPT interaction logs in the wild. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bl8u7ZRlbM.

Artetxe, M., Bhosale, S., Goyal, N., Mihaylov, T., Ott, M., Shleifer, S., Lin, X. V., Du, J., Iyer, S., Pasunuru, R., et al. Efficient large scale language modeling with mixtures of experts. *arXiv preprint arXiv:2112.10684*, 2021.

Bavarian, M., Jun, H., Tezak, N., Schulman, J., McLeavey, C., Tworek, J., and Chen, M. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022.

bench authors, B. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=uyTL5Bvosj.

Bojar, O. r., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huck, M., Jimeno Yepes, A., Koehn, P., Logacheva, V., Monz, C., Negri, M., Neveol, A., Neves, M., Popel, M., Post, M., Rubino, R., Scarton, C., Specia, L., Turchi, M., Verspoor, K., and Zampieri, M. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation*, pp. 131–198, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W16/W16-2301.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. 2021.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Chung, H. W., Constant, N., Garcia, X., Roberts, A., Tay, Y., Narang, S., and Firat, O. Unimax: Fairer and more effective language sampling for large-scale multilingual pretraining. *arXiv preprint arXiv:2304.09151*, 2023.

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

Dai, D., Deng, C., Zhao, C., Xu, R., Gao, H., Chen, D., Li, J., Zeng, W., Yu, X., Wu, Y., et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pp. 5547–5569. PMLR, 2022.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res*, 23:1–40, 2021.

Fu, Yao; Peng, H. and Khot, T. How does gpt obtain its ability? tracing emergent abilities of language models to their sources. *Yao Fu's Notion*, Dec 2022.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

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

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.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., Casas, D. d. l., Hanna, E. B., Bressand, F., et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Joshi, M., Choi, E., Weld, D., and Zettlemoyer, L. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Barzilay, R. and Kan, M.-Y. (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147.

Kenton, J. D. M.-W. C. and Toutanova, L. K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, pp. 2, 2019.

Kocetkov, D., Li, R., Ben Allal, L., Li, J., Mou, C., Muñoz Ferrandis, C., Jernite, Y., Mitchell, M., Hughes, S., Wolf, T., Bahdanau, D., von Werra, L., and de Vries, H. The stack: 3 tb of permissively licensed source code. *Preprint*, 2022.

Komatsuzaki, A., Puigcerver, J., Lee-Thorp, J., Ruiz, C. R., Mustafa, B., Ainslie, J., Tay, Y., Dehghani, M., and Houlsby, N. 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.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Lewis, M., Bhosale, S., Dettmers, T., Goyal, N., and Zettlemoyer, L. Base layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, pp. 6265–6274. PMLR, 2021.

Li, J., Zhang, Z., and Zhao, H. Self-prompting large language models for open-domain qa. *arXiv preprint arXiv:2212.08635*, 2022.

Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Lou, Y., Xue, F., Zheng, Z., and You, Y. Cross-token modeling with conditional computation. *arXiv preprint arXiv:2109.02008*, 2021.

Mustafa, B., Riquelme, C., Puigcerver, J., Jenatton, R., and Houlsby, N. Multimodal contrastive learning with limoe: the language-image mixture of experts. *Advances in Neural Information Processing Systems*, 35:9564–9576, 2022.

Nijkamp, E., Xie, T., Hayashi, H., Pang, B., Xia, C., Xing, C., Vig, J., Yavuz, S., Laban, P., Krause, B., et al. Xgen-7b technical report. *arXiv preprint arXiv:2309.03450*, 2023.

Puigcerver, J., Riquelme, C., Mustafa, B., and Houlsby, N. From sparse to soft mixtures of experts. *arXiv preprint arXiv:2308.00951*, 2023.

Rae, J. W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Riquelme, C., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Susano Pinto, A., Keysers, D., and Houlsby,

N. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34: 8583–8595, 2021.

Roller, S., Sukhbaatar, S., Weston, J., et al. Hash layers for large sparse models. *Advances in Neural Information Processing Systems*, 34:17555–17566, 2021.

Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

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

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Soldaini, L., Kinney, R., Bhagia, A., Schwenk, D., Atkinson, D., Authur, R., Bogin, B., Chandu, K., Dumas, J., Elazar, Y., Hofmann, V., Jha, A. H., Kumar, S., Lucy, L., Lyu, X., Magnusson, I., Morrison, J., Muennighoff, N., Naik, A., Nam, C., Peters, M. E., Ravichander, A., Richardson, K., Shen, Z., Strubell, E., Subramani, N., Tafjord, O., Walsh, E. P., Hajishirzi, H., Smith, N. A., Zettlemoyer, L., Beltagy, I., Groeneveld, D., Dodge, J., and Lo, K. Dolma: An Open Corpus of Three Trillion Tokens for Language Model Pretraining Research. *arXiv preprint*, 2023.

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

Tay, Y., Dehghani, M., Tran, V. Q., Garcia, X., Bahri, D., Schuster, T., Zheng, H. S., Houlsby, N., and Metzler, D. Unifying language learning paradigms. *arXiv preprint arXiv:2205.05131*, 2022a.

Tay, Y., Dehghani, M., Tran, V. Q., Garcia, X., Wei, J., Wang, X., Chung, H. W., Bahri, D., Schuster, T., Zheng, S., et al. Ul2: Unifying language learning paradigms. In *The Eleventh International Conference on Learning Representations*, 2022b.

Team, L.-M. Llama-moe: Building mixture-of-experts from llama with continual pre-training, Dec 2023. URL https://github.com/pjlab-sys4nlp/llama-moe.

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.

Wang, B. and Komatsuzaki, A. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

Wenzek, G., Lachaux, M.-A., Conneau, A., Chaudhary, V., Guzmán, F., Joulin, A., and Grave, E. CCNet: Extracting high quality monolingual datasets from web crawl data. In Calzolari, N., Béchet, F., Blache, P., Choukri, K., Cieri, C., Declerck, T., Goggi, S., Isahara, H., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J., and Piperidis, S. (eds.), *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pp. 4003–4012, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.494.

Xu, Y., Lee, H., Chen, D., Hechtman, B., Huang, Y., Joshi, R., Krikun, M., Lepikhin, D., Ly, A., Maggioni, M., et al. Gspmd: general and scalable parallelization for ml computation graphs. *arXiv preprint arXiv:2105.04663*, 2021.

Xue, F., He, X., Ren, X., Lou, Y., and You, Y. One student knows all experts know: From sparse to dense. *arXiv preprint arXiv:2201.10890*, 2022a.

Xue, F., Shi, Z., Wei, F., Lou, Y., Liu, Y., and You, Y. Go wider instead of deeper. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8779–8787, 2022b.

Yu, P., Artetxe, M., Ott, M., Shleifer, S., Gong, H., Stoyanov, V., and Li, X. Efficient language modeling with sparse all-mlp. *arXiv preprint arXiv:2203.06850*, 2022.

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

Zhang, Y., Kang, B., Hooi, B., Yan, S., and Feng, J. Deep long-tailed learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V., Dai, A. M., Le, Q. V., Laudon, J., et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

Zhou, Y., Du, N., Huang, Y., Peng, D., Lan, C., Huang, D., Shakeri, S., So, D., Dai, A. M., Lu, Y., et al. Brainformers: Trading simplicity for efficiency. In *International Conference on Machine Learning*, pp. 42531–42542. PMLR, 2023.

Zoph, B., Bello, I., Kumar, S., Du, N., Huang, Y., Dean, J., Shazeer, N., and Fedus, W. St-moe: Designing stable and transferable sparse expert models. *URL https://arxiv.org/abs/2202.08906*, 2022.

A. Frequent Asked Questions

We list the potentially frequently asked questions and the point-to-point5 answers as follows:

A.1. Why not show the token specialization of the checkpoints at the warmup stage?

We did not expect that the routing would be learned and fixed so early. During training, due to limited storage quota, we only keep the checkpoints every 200B tokens.

A.2. Why not compare with advanced open MoE models like Mixtral and DeepSeek-MoE?

First, our model was announced and released over 4 months earlier than Mistral and even more than DeepSeek-MoE. Second, different from models used in-house training data, our model is fully transparent. We also disclose all details and code to ensure everyone can train a comparable OpenMoE model from scratch.

A.3. Why not use MoE upcycling (Komatsuzaki et al., 2023)?

MoE is more efficient in training instead of inference, because of better parallelism induced by large batch size. Building MoE on top of dense LLMs is a smart and faster way to get an MoE model, but not a more efficient way from a long-term view. Instead, maybe distilling MoE into a dense model (Xue et al., 2022a) would be helpful if there is little performance drop.

A.4. Why not use AdamW optimizer and Cosine Learning Rate Schedule?

We applied Adafactor optimizer and Inverse Square Root learning rate schedule following ST-MoE (Zoph et al., 2022). We tried AdamW Optimizer but found that would introduce unstable issues (*i.e.,* NAN loss) frequently, which may introduce a significant amount of hyper-parameter sweep. Considering the limited computational resources we have, we decide to simply follow the well-studied learning rate schedule from ST-MoE (Zoph et al., 2022).

A.5. Why not use better and larger datasets?

When launching this project in 2023 May, there were only a few available open-source pre-training datasets. However, the scale and quality of open-sourced pre-training datasets are getting better. For instance, Soldaini et al. (2023) released 3T tokens with careful cleaning. Computer (2023) also released a huge dataset with 30T tokens in total. We believe training on the future better data will improve the LLM performance generally by a large margin.

B. Related Work

B.1. Before OpenMoE

MoE is not new. One representative early effort is, Shazeer et al. (2017) embed the MoE layer into a recurrent language model. Due to the scalability of Transformer architecture, GShard (Lepikhin et al., 2020) integrates MoE into Transformer layer and uses expert parallelism to train MoE-based Transformer at scale. Switch Transformer (Fedus et al., 2021) is the earliest open-source MoE-based LM to our best knowledge, which used encoder-decoder architecture and trained with C4 (Raffel et al., 2020) dataset. Due to the success of Switch Transformer on large-scale pre-training, MoE got more attention, and more advanced routing algorithms were invented. For instance, BASE Layers (Lewis et al., 2021) formulates token-to-expert allocation as a linear assignment problem, allowing an optimal assignment in which each expert receives an equal number of tokens. Roller et al. (2021) simply modifies the feedforward layer to hash to different sets of weights depending on the current token and achieves promising results compared to learning-based routing. Different Token-based routing above, Zhou et al. (2022) propose to let experts select their favorite tokens, *i.e.,* Expert-Choice Routing. Expert-choice Routing achieves more balanced token assignment and better cost-effectiveness trade-off.

Beyond the routing algorithm, there is also some work focusing on scaling MoE efficiently. Artetxe et al. (2021) trained their MoE models mainly on the datasets used in RoBERTa (Liu et al., 2019) and CC100 (Wenzek et al., 2020) (112B tokens in total). GaLM (Du et al., 2022) further scale decoder-only MoE model with an in-house high-quality dataset with 1.6T tokens. Brainformer (Zhou et al., 2023) proposes an evolutionary search to discover MoE attributes, *e.g.,* the best way to interleave layers and layer capacities, when to fuse layers, and when to specialize layers with MoE modules and show its effectiveness at different scales.

In addition to language modeling, Vision Transformer (ViT) (Dosovitskiy et al., 2020) can also be enhanced by MoE architecture. ViT-MoE (Riquelme et al., 2021) verifies the scalability of MoE on ViT models. WideNet (Xue et al., 2022b) shares MoE-based Transformer blocks with individual layer normalization to achieve better parameter efficiency. SoftMoE (Puigcerver et al., 2023) further improves the routing algorithm by applying soft token selection, which not only keeps the efficiency but also stabilizes the routing gradient. There are also some efforts devoted to include MoE into non-Transformer architecture, *e.g.,* Sparse-MLP (Lou et al., 2021) for computer vision and s-MoE for language modeling (Yu et al., 2022).

B.2. After OpenMoE

Table 6: Open-sourced MoE LLMs timeline. We use the model release date as the key to sort the open-souced MoE LLMs. Dataset Size is the number of tokens in the pre-training dataset, *i.e.,* the number of tokens for one epoch. LLaMA-MoE is continued pre-trained on off-the-shelf LLaMA family models. We account its continue training dataset only.

Model Name	Dataset Size	Reproducible	Release Date
Switch Transformer (Fedus et al., 2021)	156B	Yes	Feb 2021
Meta-MoE (Artetxe et al., 2021)	112B	Yes	Dec 2021
OpenMoE (Ours)	1.1T	Yes	Aug 2023
Mixtral of Experts (Jiang et al., 2024)	Unknown	No	Dec 2023
LLaMA-MoE (Team, 2023)	200B	Yes	Dec 2023
DeepSeek-MoE (Dai et al., 2024)	2T	No	Jan 2024

We released our model and implementation much earlier than writing this report. As shown in Table 6, after our release, there are some partially open-sourced models released, *e.g.,* Mixtral (Jiang et al., 2024) and Deepseek-MoE (Dai et al., 2024). As we known, these models are significantly better in terms of final results. However, since these models are trained with in-house data, we have no idea about how things happened. We believe, although our results are not that amazing, the fully open-sourced nature and the in-depth analysis are both meaningful for the community.

C. Data Mixture

Table 7: Three versions of OpenMoE pre-training data mixture.

	Version I	**Version II**	**Version III**
Model	OpenMoE-Base, OpenMoE-8B/32E		OpenMoE-34B/32E
Period	before 780B tokens → after 780B tokens		from start to end
Dataset		**Sampling Ratio**	
RedPajama	50.0%	83.5%	67.5%
C4	7.50%	15.0%	15.0%
Wikipedia	2.25%	6.50%	4.50%
Stackexchange	1.00%	2.50%	1.00%
ArXiv	1.25%	4.50%	4.50%
Books	2.25%	6.50%	4.50%
GitHub	2.25%	5.00%	5.00%
Commoncrawl	33.5%	43.5%	33.0%
Wikipedia-en	0.00%	6.50%	2.50%
The Stack Dedup	50.0%	10.0%	30.0%

As shown in Table 7, we extracted 50% of data from the RedPajama (Computer, 2023) and 50% of data from the duplication version of The Stack (Kocetkov et al., 2022). Our experimental results show that the version I data mixture might be a bit aggressive in its code proportion. We fix these issues at the later stage of pre-training

D. Model Architecture

Token-choice Routing. We generally follow ST-MoE (Zoph et al., 2022) for our model architecture and routing design to ensure training stability, which is extremely important when training larger models. Given E trainable experts and input representation $x \in \mathbb{R}^D$, the output of MoE model can be formulated as:

$$\mathrm{MoE}(x) = \sum_{i=1}^{E} g(x)_i e_i(x), \tag{1}$$

where $e_i(\cdot)$ is a non-linear transformation $\mathbb{R}^D \to \mathbb{R}^D$ of the i^{th} expert, and $g(\cdot)_i$ is the i^{th} element of the output of the trainable router $g(\cdot)$, a non-linear mapping $\mathbb{R}^D \to \mathbb{R}^E$. Usually, both $e(\cdot)$ and $g(\cdot)$ are parameterized by neural networks. Please note each expert is an FFN layer instead of a complete Transformer model in most MoE-based Transformer models, including ours.

Top-2 Selection. According to the formulation above, when $g(\cdot)$ is a sparse vector, only part of the experts would be activated and updated by back-propagation during training. We set the gating layer as a top-K selection as:

$$\mathrm{g}(x) = \mathrm{TopK}(\mathrm{softmax}(f(x))), \tag{2}$$

where $f(\cdot)$ is routing linear transformation $\mathbb{R}^D \to \mathbb{R}^E$. When $K \ll E$, most elements of $\mathrm{g}(x)$ would be zero so that sparse conditional computation is achieved. We set $K = 2$ following Zoph et al. (2022).

Residual MoE. Each vanilla Transformer block can be written as:

$$\begin{aligned} x' &= \mathrm{LayerNorm}_i^{\mathrm{att}}(x), \\ x &= \mathrm{MHA}(x') + x, \\ x'' &= \mathrm{LayerNorm}_i^{\mathrm{ffn}}(x), \\ x &= \mathrm{FFN}(x'') + x, \end{aligned} \tag{3}$$

In OpenMoE, for each MoE-based Transformer block, we use one residual MoE layer to ensure that one fixed FFN layer is always activated for every token. That is:

$$\begin{aligned} x' &= \mathrm{LayerNorm}_i^{\mathrm{att}}(x), \\ x &= \mathrm{MHA}(x') + x, \\ x'' &= \mathrm{LayerNorm}_i^{\mathrm{ffn}}(x), \\ x &= \mathrm{MoE}(x'') + \mathrm{FFN}(x'') + x, \end{aligned} \tag{4}$$

Note we use MoE-based Transformer blocks in an interleaved manner instead of placing MoE in every Transformer block. In our setting, we use MoE every 4 layers in OpenMoE-Base/16E and OpenMoE 34B/32E and use MoE every 6 layers for OpenMoE-8B/32E. This setting is inspired by the findings in ViT-MoE (Riquelme et al., 2021), *i.e.,* using MoE every layer introduces more computational overhead during routing, and then induces a worse cost-effective trade-off than interleaved MoE usage.

Load Balance Loss and Router Z-loss. ST-MoE (Zoph et al., 2022) follows Shazeer et al. (2017), using MoE load balance loss to ensure a balanced number of tokens assigned to different experts so that MoE models can achieve better parallelism. For each routing operation, given E experts and N batches with $B = NL$ tokens, the following auxiliary loss is added to the total model loss during training:

$$L_b = E \cdot \sum_{i=1}^{E} m_i \cdot P_i, \tag{5}$$

where m is a vector, P_i is $\mathrm{softmax}(f(x))$. i denotes the expert ID. The i^{th} element is the fraction of tokens dispatched to expert i:

$$m_i = \frac{1}{B} \sum_{j=1}^{B} \mathrm{h}(x_j)_i, \tag{6}$$

where $\mathrm{h}(\cdot)$ is an index vector selected by TopK in Eq. 2. $\mathrm{h}(x_j)_i$ is the i^{th} element of $\mathrm{h}(x_j)$. It is noticeable that, different from $g(x)_i$ in Eq. 2, m_i and $\mathrm{h}(x_j)_i$ are non-differentiable. However, a differentiable loss function is required to optimize MoE in an end-to-end fashion, so we use the routing score $\mathrm{softmax}(f(x))$ in Eq. 2 (*i.e.,* P_i in Eq. 5)to make the routing decision differentiable and then learnable.

In addition to the load balance loss, Zoph et al. (2022) proposed router z-loss for more stable MoE training:

$$L_z(x) = \frac{1}{B}\sum_{i=1}^{B}\left(\log\sum_{j=1}^{E} e^{x_j^{(i)}}\right)^2 \tag{7}$$

This router z-loss can penalize large logits input to the gating network and encourage the absolute magnitude of numbers to be small so that it can reduce the round-off errors in MoE layers. Please refer to ST-MoE paper (Zoph et al., 2022) for a detailed explanation.

Taken together, our final training loss can be written as:

$$L = L_{CE} + L_b + L_z \tag{8}$$

where L_{CE} is the cross-entropy loss in language model pre-training.

E. UL2 Training Objective

Table 8: UL2's mixture-of-denoisers configuration, μ is average span length and r is the mask ratio.

Training Objective	Percentage
PrefixLM, r=0.5	50%
SpanCorrupt	
μ=3, r=0.15	10%
μ=8, r=0.15	10%
μ=3, r=0.5	10%
μ=8, r=0.5	10%
μ=64, r=0.5	10%

Our detailed UL2 training objective configuration is shown in Table 8. We use only 20% low mask ratio (r=0.15) because there are fewer output tokens during training, which may slow down the learning. We also use more PrefixLM than the default UL2 setting because we think the zero-shot and in-context learning ability enhanced by PrefixLM training is important.

F. Hyper-parameters

Table 9: Model Configurations. H is the hidden size. "Layout" means the way of using the MoE layer. For instance, "Every 4" means we use one MoE layer for every 4 transformer blocks. H_{FFN} is the FFN intermediate size. N_{Head} and H_{Head} are the number of attention heads and attention head dimensions. L is the number of layers. #Param is the total parameters. #ActParam is the number of parameters we used to process each token in Transformer blocks. #ActParam w/ E is the sum of the #ActParam and the number of parameters in the token embedding layer.

Model	Layout	H	H_{FFN}	N_{Head}	H_{Head}	L	#Param	#ActParam w/ E	#ActParam
OpenMoE-Base/16E	Every 4	768	3072	12	64	12	650M	339M	142M
OpenMoE-8B/32E	Every 6	2048	8192	24	128	24	8.7B	2.6B	2.1B
OpenMoE-34B/32E	Every 4	3072	12288	24	128	32	34B	6.8B	6.0B
TinyLLaMA	-	2048	5632	32	64	22	1.0B	1.0B	0.9B
OpenLLaMA-3B	-	3200	8640	32	64	26	3.0B	3.0B	2.9B
LLaMA-7B	-	4096	11008	32	128	32	6.6B	6.4B	6.4B

For OpenMoE-8B/32E, we set the head dimension as 128 instead of 64, which may be too large for a model using 2B activated Transformer parameters. We suggest that using 64 may induce a better cost-effectiveness trade-off than ours. For the number of parameters in the table above, since most parameters in Transformer blocks are from attention layer and FFN layer, we only account the trainable parameters from these two for simplicity.

Table 10: OpenMoE training hyper-parameters.

	Base/16E	**8B/32E**	**34B/32E**
Optimizer	Adafactor		
Batch Size	128	2048	2048
Training Steps	500K	500K	100K
Peak Learning Rate	0.01		
Learning Rate Schedule	Inverse Square Root Decay		
Warmup Steps	10K		
Sequence Length	2048		
Load Balance Loss Weight	0.01		
Z-Loss Weight	0.001		
Router Z-Loss Weight	0.0001		

Different from existing LLMs trained with AdamW, we used Adafactor, a more memory-efficient optimizer. Although it performs slightly worse than AdamW with the same training steps, the memory efficiency enables us to use less model parallelism and more data parallelism. In this case, using Adafactor makes our training cheaper than using AdamW to train the same model on the same data. However, we highlight that the margin of this gap is unclear because it highly depends on the hardware and model size. For our infrastructure, *i.e.,* TPUv3, this gap should be relatively larger due to the limited on-chip memory (16 GB per core).

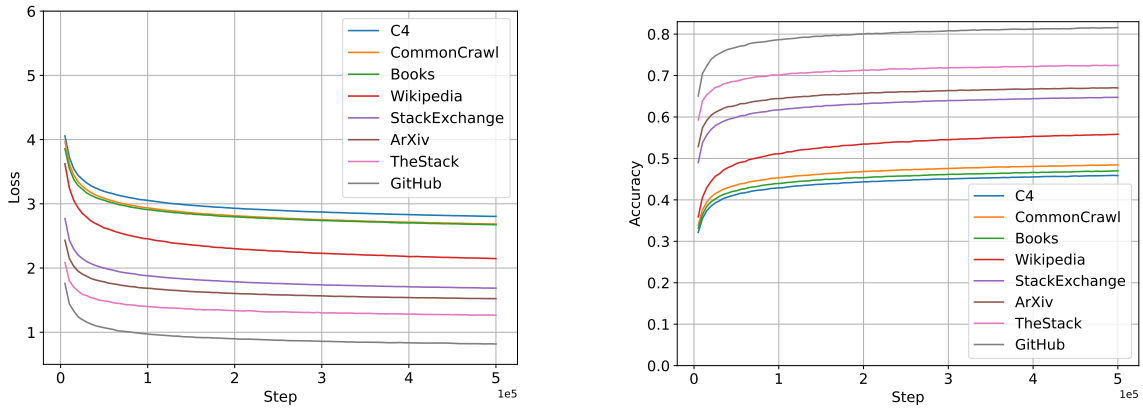
G. Ablation Study

Table 11: Ablation study with OpenMoE-Base/16E on zero-shot TriviaQA (Joshi et al., 2017).

Method	EM	F1
OpenMoE	1.4	4.5
w/o MoE	0.1	0.3
w/o UL2 (PrefixLM only)	0.0	0.0
w/o Code data	0.7	1.1
w/ LLaMA tokenizer	2.2	5.7

As an initial evaluation of our design decisions, we conducted an ablation study using the OpenMoE-Base/16E model. It's important to note that while these results provide early insights, we cannot be certain of their generalizability to larger models, primarily due to computational resource constraints that preclude larger-scale ablations.

Our findings indicate that several elements — the MoE approach, the UL2 training objective, and the increased emphasis on code data — all contribute positively to the base version's performance in zero-shot TriviaQA tasks. The model using LLaMA tokenizer (Touvron et al., 2023) outperforms the one with umT5 tokenizer. This outcome is considered acceptable, even though a larger vocabulary size might slightly impair performance. We believe that supporting low-resource languages is crucial, as foundational models should be accessible and beneficial to a diverse global audience. After this sanctity check, we proceed to scale OpenMoE up to OpenMoE-8B/32E.



(a) Comparison of the validation loss on different pre-training datasets.

(b) Comparison of validation accuracy on different pre-training datasets.

Figure 8: Comparison of the validation loss and accuracy on different pre-training datasets. We can observe that models are easier to achieve higher accuracy and lower loss on code data.

We also conduct an ablation study to compare the progress of learning the data from different domains. As shown in Figure 8, we can observe that models are easier to achieve higher accuracy and lower loss on code data. On Github, although our model is small, it can still achieve over 80% token prediction accuracy. We infer that this is because of the long-tail token distribution in code data. For instance, a large number of tokens in code are "\n" and "\t", which are relatively easier to predict.

H. BigBench-Lite Results

Table 12: Detailed BigBench-Lite results. Note that BIG-G-sparse 8B is an MoE model with 60B parameters in total.

Model	BIG-G 8B	BIG-G-sparse 8B	GPT-3 6B	**OpenMoE-8B**
auto_debugging	0.0	0.0	0.0	17.65
bbq_lite_json	58.63	46.13	49.85	42.67
code_line_description	4.66	2.44	20.18	2.44
conceptual_combinations	−2.16	1.07	−3.36	0.81
conlang_translation	31.38	33.25	37.92	36.93
emoji_movie	3.75	7.5	−5.0	3.75
formal_fallacies_syllogisms_negation	0.78	−0.39	−0.8	−0.56
hindu_knowledge	12.44	8.63	19.29	16.24
known_unknowns	−34.78	−4.35	−8.7	−13.04
language_identification	1.39	−0.33	1.66	1.77
linguistics_puzzles	0.0	0.0	0.0	0.05
logic_grid_puzzle	−2.45	0.01	−0.28	0.89
logical_deduction	1.38	4.2	1.05	0.09
misconceptions_russian	−34.69	−38.78	−34.69	−38.78
novel_concepts	10.16	14.06	17.97	6.25
operators	10.48	16.67	20.0	20.48
parsinlu_reading_comprehension	0.0	0.0	0.0	11.97
play_dialog_same_or_different	12.5	4.69	−3.8	1.1
repeat_copy_logic	0.0	6.25	0.0	3.12
strange_stories	−7.15	−4.77	9.54	14.52
strategyqa	7.23	8.4	−3.8	3.36
symbol_interpretation	6.06	0.13	4.17	2.65
vitaminc_fact_verification	6.25	1.27	−3.2	21.34
winowhy	4.69	5.27	11.6	10.14
Average	3.77	4.63	5.40	**6.93**

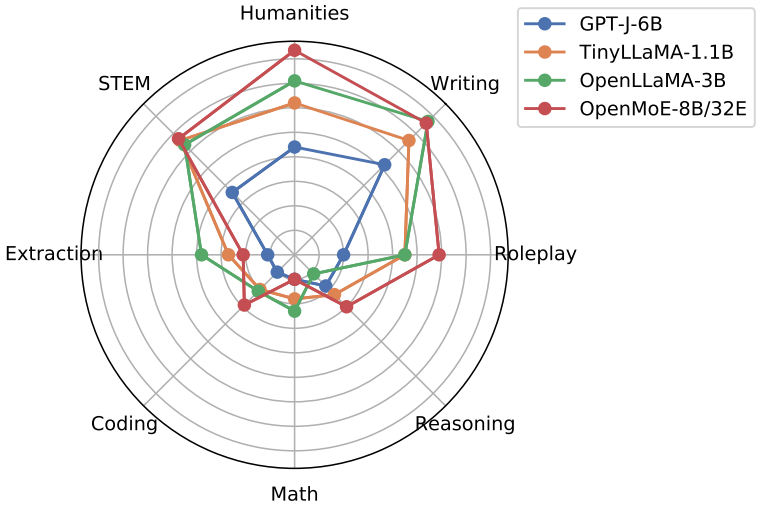
Table 13: Evaluate OpenMoE-8B/32E on lm-evaluation-harness. The results of OpenLLaMA are from its homepage, which only provides two effective digits.

Dataset	TinyLLaMA-1.1B	OpenLLaMA-3B	OpenMoE-8B/32E
ANLI-R1	34.2	33.0	32.7
ANLI-R2	32.4	36.0	33.2
ANLI-R3	35.1	38.0	33.9
HellaSwag	59.2	52.0	45.5
WinoGrande	59.1	63.0	60.3
PIQA	73.3	77.0	74.2
ARC-Easy	55.2	68.0	64.1
ARC-Challenge	30.1	34.0	30.3
Boolq	57.8	66.0	61.2
TruthfulQA	37.6	35.0	36.0
OpenbookQA	21.8	26.0	24.6
RTE	51.9	55.0	53.4
WiC	50.1	50.0	49.8
Average	45.9	**48.7**	46.1

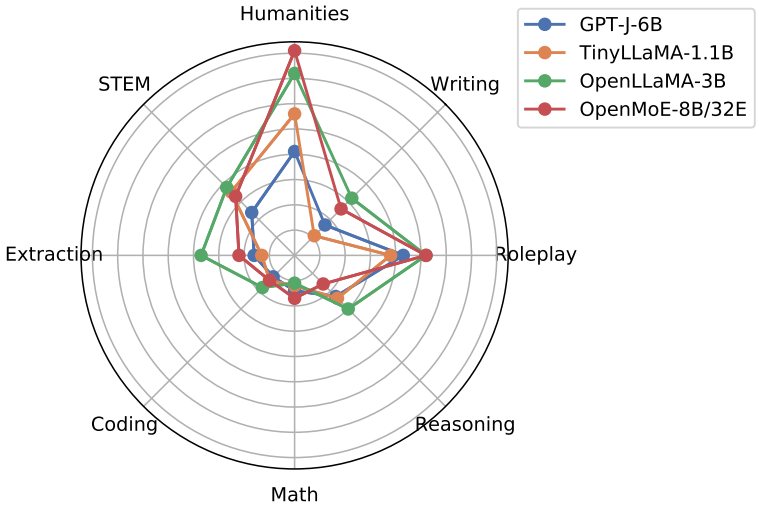
I. LM-Evaluation-Harness Results

We also evaluate OpenMoE on the 13 tasks from the LM-Evaluation-Harness collection. As shown in Table 13, both OpenMoE and TinyLLaMA performed worse than OpenLLaMA. However, the scores achieved by OpenMOE are acceptable. We suggest that the initial high sampling rate on the code data may harm the results on these text-dominated benchmarks, which is one of the issues we will discuss in Section 5.

J. MT-Bench Results



(a) Single-turn results.



(b) Multi-turn results.

Figure 9: Evaluate OpenMoE on MTBench.

K. Language-level Specialization

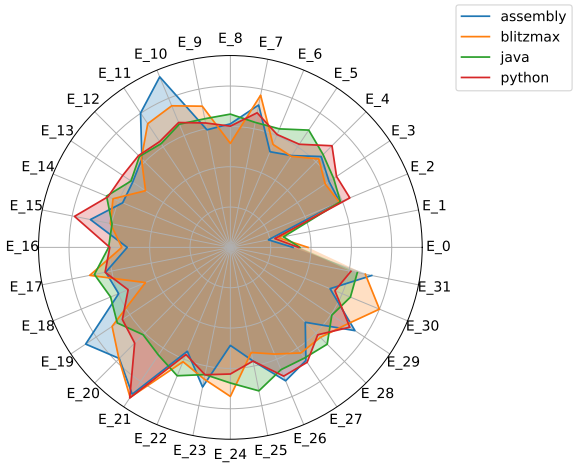


Figure 10: Visualization of the routing decision on TheStack dataset. E_i denotes the ratio of tokens routed to i_{th} expert.

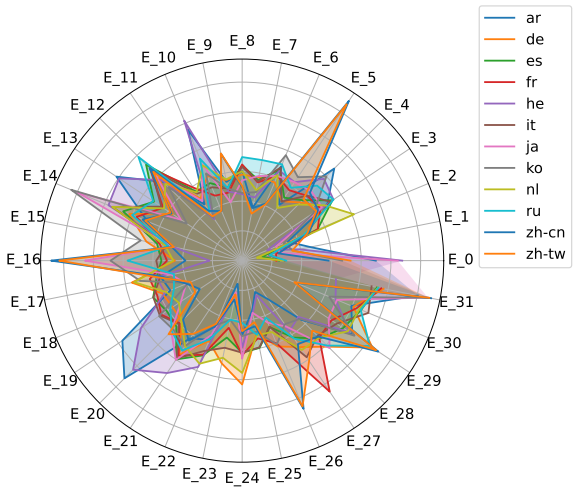


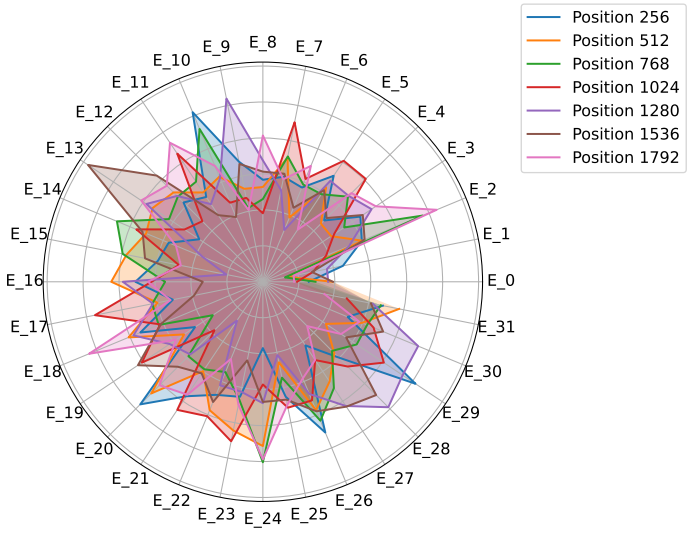
Figure 11: Visualization of the routing decision on TED-Parallel-Corpus including 12 languages, *i.e.,* ar (Arabic), de (German), es (Spanish), fr (French), he (Hebrew), it (Italian), ja (Japanese), ko (Korean), nl (Dutch), ru (Russian), zh-cn (Chinese Simplified), zh-tw (Chinese, Traditional), E_i denotes the ratio of tokens routed to the i_{th} expert.

In Figure 10, we compare 4 different coding languages, *i.e.,* Assembly, Blitzmax, Java, and Python. Similar to the domain level, even for Assembly and Blitzmax, *i.e.,* two low-resource languages compared with Java and Python, they still did not exhibit significant expert specialization.

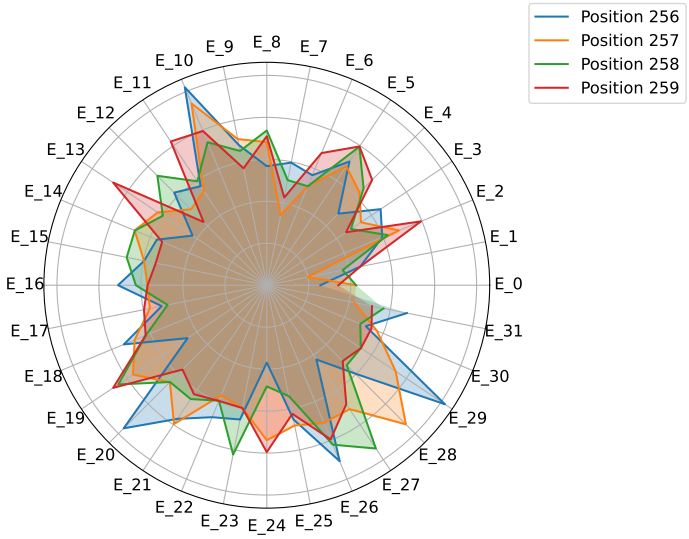
We further study the expert specialization on different natural languages. We adopted a multi-lingual parallel corpus, *i.e.,* TED-Parallel-Corpus [1] as the platform. In Figure 11, we found that there is a relatively clear specialization among different experts. For instance, zh-cn (Chinese, Simplified) and zh-tw (Chinese, Traditional) both have a strong preference for E_5 and E_{16}; ja (Japanese), and ko (Korean) both prefer E_{14}.

[1] https://github.com/ajinkyakulkarni14/TED-Multilingual-Parallel-Corpus

L. Position ID Specialization



(a) Uniform sampled token IDs.

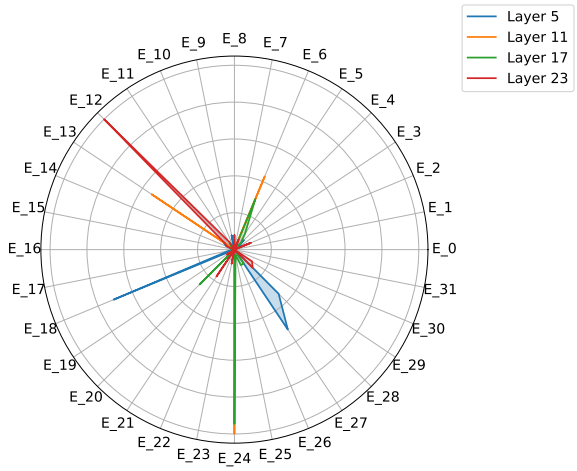


(b) Consecutive token IDs.

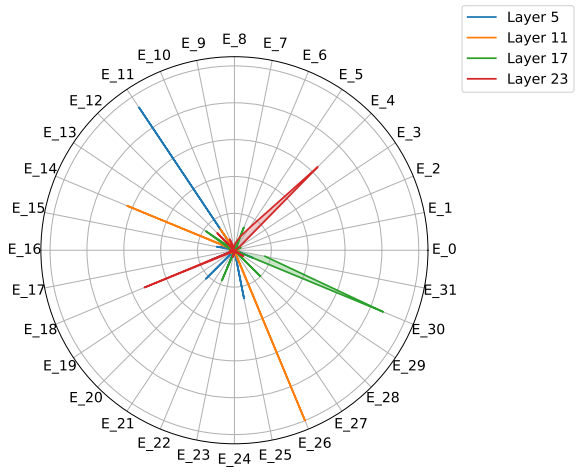
Figure 12: Visualization of the routing decision at different Position IDs. E_i denotes the ratio of tokens routed to the i_{th} expert.

We thus visualize the routing decisions on different positions in Figure 12a and Figure 12b. We can observe:(1) there are indeed some specializations in different Position IDs; (2) consecutive positions prefer similar experts, such as the E_{10} and E_{19} in Figure 12b.

M. Layer ID Specialization



(a) Token “an” routing decision at different layers.



(b) Token “ed” routing decision at different layers.

Figure 13: Visualization of the routing decision at different layers. E_i denotes the ratio of tokens routed to the i_{th} expert.

We can see both token “an” and “ed” do not have any special patterns at different layers.

N. Routing Decision Standard Deviation

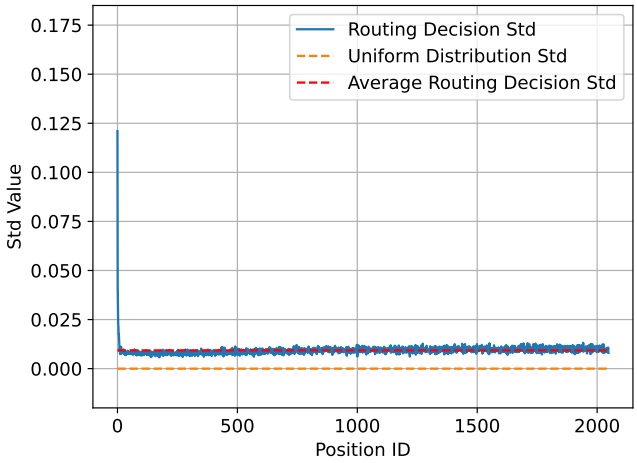


Figure 14: The routing decision standard deviation at different position IDs.

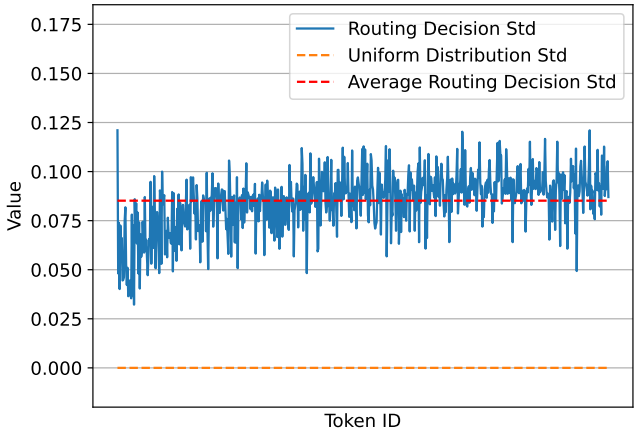


Figure 15: The routing decision standard deviation at different token IDs. We only take the token IDs with over 128 tokens, because the extremely low-resourced tokens always have large routing decision standard deviation. The token IDs never appeared also have variance at all.

In Figure 14 and 15, we can clearly see that the token IDs have a larger standard deviation on routing decisions than position IDs. Also, most token IDs have a relatively large standard deviation, which means most of the token IDs have Context-independent Routing.

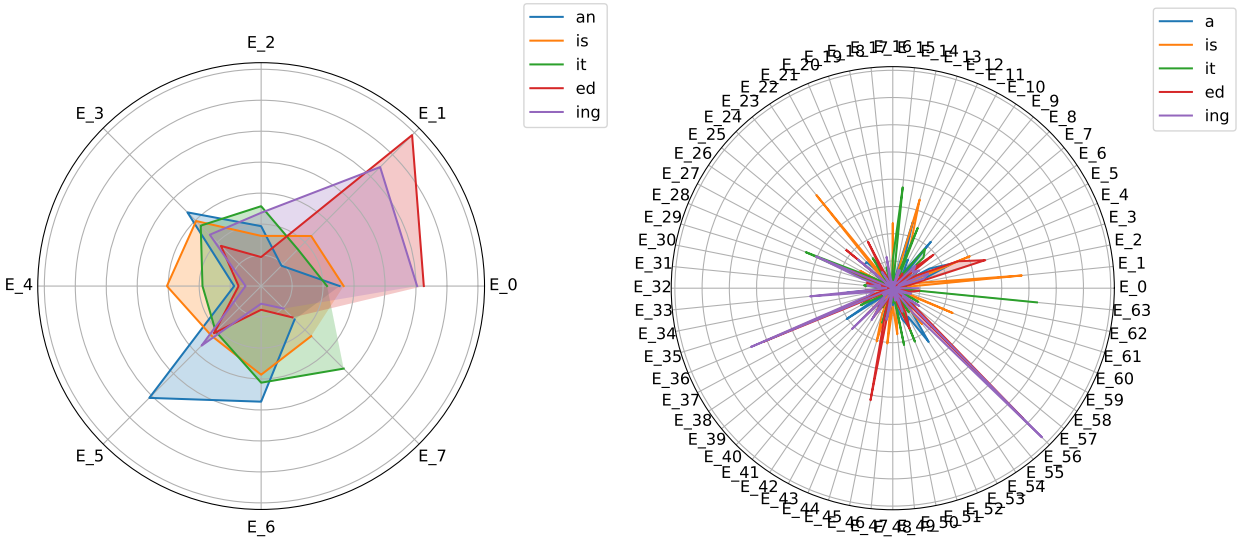
O. Top Token Selection by Experts

Table 14: Top Tokens selected by each expert.

Expert ID	Top Tokens
0	“\n’, “ ‘ ”, “ ’ ”, “s”, “-”, “$”, “y”, “_”, “ ”, “2”
1	“\n’, “1”, “ ”, “2”, “’\\”, “S”, “.”, “-”, “C”, “{”
2	“in”, “.”, “2”, “1”, “0”, “\n’, “ ”, “3”, “_”, “4”
3	“s”, “)”, “a”, “\n’, “which”, “es”, “);”, “}”, “\\”, “e”
4	“\n’, “.”, “0”, “the”, “,“, “_”, “that”, “1”, “as”, “ˆ’
5	“ ”, “\n’, “s”, “2”, “a”, “on”, “ter”, “*”, “\\”, “all”
6	“the”, “,”, “.”, “a”, “to”, “of”, “ ”, “s”, “de”, “\n’
7	“,“, “and”, “\n’, “:”, “_”, “ ”, “0”, “on”, “at”, “{”
8	“(”, “.”, “that”, “s”, “ ”, “,“, “C”, “which”, “of”, “G”
9	“(”, “this”, “2”, “\n’, “\\”, “ ”, “3”, “also”, “I”, “1”, “,“
10	“\n’, “.”, “and”, “\r”, “).”, “;”, “\t”, “:”, “?”, “The”
11	“to”, “1”, “the”, “2”, “0”, “s”, “for”, “t”, “3”, “\n’
12	“the”, “,“, “$’, “to”, “in”, “?”, “as”, “that”, “In”, “who”
13	“in”, “/”, “0”, “\n’, “with”, “-”, “ ”, “{”, “of”, “2”
14	“is”, “.”, “are”, “be”, “was”, “s”, “\n’, “,“, “has”, “not”
15	“of”, “\n’, “_”, “s”, “ ”, “.”, “S”, “the”, “for”, “\\”
16	“cite”, “,“, “\n’, “.”, “{”, “s”, ””””, “ing”, “data”, “\\$”, “\t”
17	“the”, “.”, “\n’, “The”, “0”, “1”, “as”, “of”, “5”, “2”
18	“-”, “{”, “for”, “(”, “_”, “ ”, “$”, “(”, “\n’, “}”
19	“ ”, “and”, “in”, “to”, “,“, “of”, “or”, “\n’, “by”, “$”
20	“\n’, “the”, “$”, “a”, “0”, “}”, “this”, “1”, “s”, “9”, “ ”
21	“,“, “and”, “ ”, “.”, “\n’, “=”, “\t”, “the”, “ ”, “n”
22	“the”, “\n’, “)”, “,“, “his”, “their”, “s”, ””””, “,“, “i”
23	“.”, “\n’, “,“, “*”, “¡pad¿”, “Cha”, “i”, “!”, “our”, “/”
24	“a”, “with”, ’}”, “in”, “)””, “:”, “an”, “1”, “\n’, “at”
25	“\\”, “the”, “.”, “of”, “er”, “, ”, “s”, “ter”, “book”, “model”
26	“\n’, “, ”, “.”, “a”, “¡pad¿”, “s”, “de”, “al”, “-”
27	“the”, ””””, “I”, “The”, “, ”, “it”, “we”, “he”, “a”, “x”
28	“, ”, “ly”, “{”, “_{”, “new”, “-”, “ed”, “more”, “\n’, “d”
29	“, ”, “.”, “of”, “;”, “by”, “,:”, “\n’, “to”, “from”, “(”,
30	“}”, “ed”, “d”, “have”, “ing”, “, ”, “has”, “s”, ””””, “had”
31	“to”, “can”, “s”, “of”, “ing”, “will”, “not”, “e”, “ed”, “would”

P. Study Other MoE Models

Background In this section, we investigate whether the issues we found above exist in other MoE-based LLMs, *i.e.,* Mixtral and DeepSeek-MoE. Both Mixtral and Deepseek-MoE are trained by dropless token routing, which means that these models would not drop the token even if the workload of different is unbalanced. This design is fine if our model is not that large after applying some implementation tricks like Megablock (**?**), which can handle the imbalanced workload better if the experts are on the same GPU. However, implementation tricks like Megablock cannot work efficiently when there is only one expert on the single GPU, and unfortunately, this happens for very large MoE LLM (e.g. one GPT-style MoE with over 2T parameters). Considering that the GPU memory size is not growing as fast as before, having a balanced workload for each expert is still extremely important for efficient large MoE model training.



(a) Mixtral-8×7B

(b) Deepseek-MoE-16B

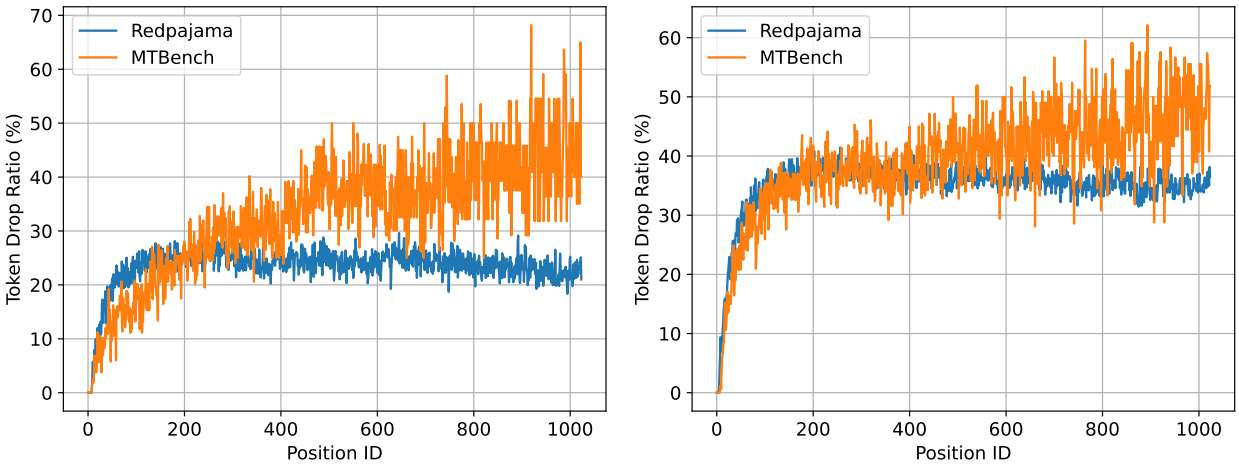
Figure 16: Visualization of the routing decision at different Token IDs.

Context-Independent Specialization We visualize the token ID specialization of Mixtral and Deepseek-MoE in Figure 16a and 16b. We found, similar to OpenMoE, Deepseek-MoE has a clear Context-Independent Specialization, but Mixtral doesn't have that. We suggest that the reason is, according to this blog[2], Mixtral is probably finetuned based on the Mistral-7B dense checkpoint, i.e. MoE upcycling (Komatsuzaki et al., 2023), instead of training from scratch like deepseek-MoE and OpenMoE. Since the experts in Mixtral are very similar, it makes sense that there is a relatively weak specialization in their MoE model, and at the same time, since the model has learned high-level semantics when converting dense LLM to MoE LLM, it is less likely to develop Context-Independent Specialization. Therefore, we suggest that Context-Independent Specialization is an issue only for training MoE from scratch. However, as we discussed in our paper, MoE is more efficient during training than inference, it is still highly desirable to study how to avoid Context-Independent Specialization when training MoE from scratch or converting dense LLM to MoE at the early stage of training. One feasible solution can be that, first train a dense half-cooked LLM (maybe using 20% pretraining tokens or so), and then convert the dense LLM to MoE via MoE upcycling. We can then train the MoE with 80% of the training tokens left to ensure a better cost-effectiveness trade-off.

Early Routing Learning Since Mixtral and Deepseek-MoE have no open-sourced intermediate checkpoints, we cannot study this issue on these models.

Drop-towards-the-End As mentioned before, Mixtral and Deepseek-MoE have no token drop mechanism. However, this is not friendly to expert parallelism, especially for very large MoE-LLM with trillion-level parameters, although it is okay if the model is relatively small (¡100B). Therefore, we still study whether there is a Drop-towards-the-End issue in Mixtral and Deepseek-MoE by manually adding a token drop mechanism when there are too many tokens routed to an expert. As

[2] https://x.com/tianle_cai/status/1734188749117153684?s=20



(a) Mixtral-8×7B

(b) DeepSeek-MoE-16B

Figure 17: Comparing the ratio of tokens dropped at different position IDs.

shown in Figure 17a and Figure 17b, there is a clear token drop at the later tokens in the input sequences, which means the Drop-towards-the-End is an issue for all these MoE LLMs.

Q. Tokenizer Analysis

Table 15: Compare umT5 tokenizer and LLaMA tokenizer on the subsets extracted from different datasets. Vocab used denotes the number of token IDs activated when tokenizing the whole subset. The umT5/LLaMA means, when tokenizing the same subset, the ratio of the number of tokens generated by umT5 and LLaMA.

Dataset	Subset	LLaMA Tokenizer		umT5 Tokenizer		umT5/LLaMA
		#Tokens	Vocab Used	#Tokens	Vocab Used	
RedPajama	arxiv	125,339	8,327	131,059	8,762	1.046
	book	137,972	11,603	131,072	15,202	0.950
	c4	28,592	5,439	26,428	5,554	0.924
	cc	78,450	8,738	73,403	9,927	0.936
	github	54,707	4,769	59,732	4,539	1.092
	stackexchange	40,659	4,714	43,195	4,317	1.062
	wikipedia	37,406	7,179	30,555	8,748	0.817
TheStack	assembly	49,143	3,066	50,738	3,130	1.032
	blitzmax	78,259	4,200	80,658	4,209	1.031
	java	64,236	4,229	69,902	3,905	1.088
	python	66,243	5,095	70,795	4,799	1.069
MTBench	writing	6,062	1,700	5,786	1,535	0.954
	roleplay	4,309	1,291	4,076	1,172	0.946
	reasoning	2,369	478	2,309	429	0.975
	math	5,163	290	5,154	282	0.998
	coding	4,955	651	5,256	631	1.061
	extraction	7,058	1,376	6,817	1,234	0.966
	stem	4,783	1,151	4,527	1,039	0.946
	humanities	6,398	1,451	5,946	1,320	0.929
Multi-lingual TED	ar	256,952	187	88,406	8,037	0.344
	de	103,270	4,880	80,593	8,470	0.780
	es	101,212	4,745	78,713	8,519	0.778
	fr	115,057	5,156	95,978	8,164	0.834
	he	242,446	239	86,891	4,074	0.358
	it	109,591	4,593	84,201	8,833	0.768
	ja	144,825	931	63,491	6,860	0.438
	ko	257,107	596	106,770	2,736	0.415
	nl	102,703	4,234	75,084	7,540	0.731
	ru	107,144	2,502	74,445	9,658	0.695
	zh-cn	149,581	1,058	88,107	3,611	0.589
	zh-tw	173,415	1,107	93,693	3,619	0.540