Demons in the Detail: On Implementing Load Balancing Loss for Training Specialized Mixture-of-Expert Models

Zihan Qiu[*1], **Zeyu Huang**[*2], **Bo Zheng**[*1], **Kaiyue Wen**[3], **Zekun Wang**[1], **Rui Men**[1]
Ivan Titov[2], **Dayiheng Liu**[✉1], **Jingren Zhou**[1], **Junyang Lin**[✉1]
[1] Qwen Team, Alibaba Group [2] University of Edinburgh [3] Stanford University

Abstract

This paper revisits the implementation of **L**oad-**b**alancing **L**oss (LBL) when training Mixture-of-Experts (MoEs) models. Specifically, LBL for MoEs is defined as $N_E \sum_{i=1}^{N_E} f_i P_i$, where N_E is the total number of experts, f_i represents the frequency of expert i being selected, and p_i denotes the average gating score of the expert i. Existing MoE training frameworks usually employ the parallel training strategy so that f_i and the LBL are calculated within a **micro-batch** and averaged across parallel groups. However, a micro-batch for training billion-scale LLMs typically contains very few sequences, leading to the micro-batch LBL being almost at the sequence level, and the router is pushed to distribute the token evenly within each sequence. Under this strict constraint, even tokens from a domain-specific sequence (*e.g.*, code) are uniformly routed to all experts, thereby inhibiting expert specialization. In this work, we propose calculating LBL using a **global-batch** to loose this constraint. Because a global-batch contains much more diverse sequences than a micro-batch, which will encourage load balance at the corpus level. Specifically, we introduce an extra communication step to synchronize f_i across micro-batches and then use it to calculate the LBL. Through experiments on training MoEs-based LLMs (up to **42.8B** parameters and **400B** tokens), we surprisingly find that the global-batch LBL strategy yields excellent performance gains in both pre-training perplexity and downstream tasks. Our analysis reveals that the global-batch LBL greatly improves the domain specialization of experts. **Global-batch LBL is also used in Qwen3-MoEs.**

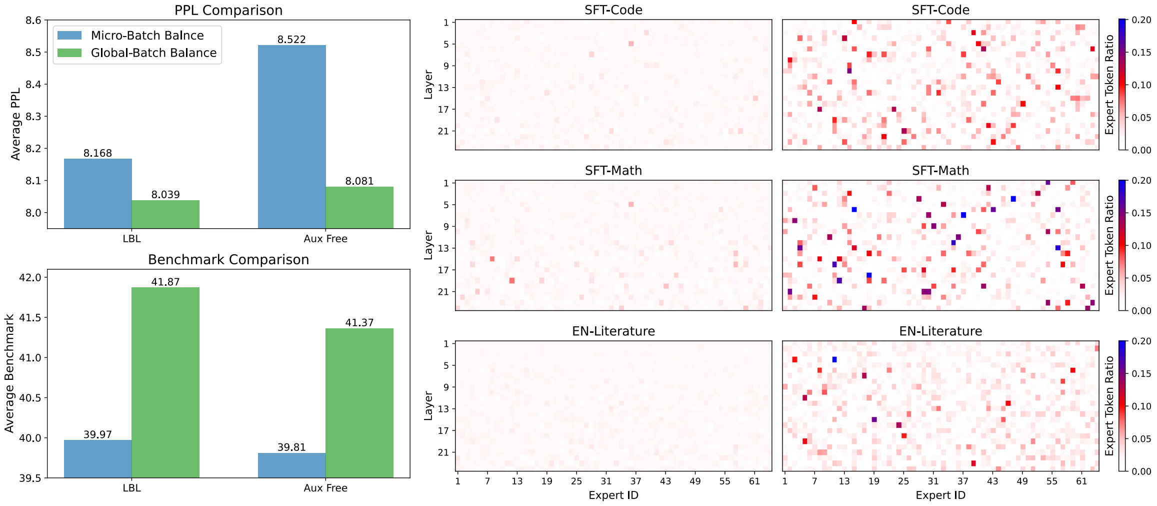
1 Introduction

In recent years, the Mixture-of-Experts (MoE) framework (Szymanski and Lemmon, 1993; Shazeer et al., 2017) has become a popular technique to scale the model parameters up (Jiang et al., 2024; Dai et al., 2024; Liu et al., 2024a; Yang et al., 2024). For instance, Mixtral (Jiang et al., 2024) (141B), Deepseek-v3 (Liu et al., 2024a) (671B), MiniMax-01 (Li et al., 2025) (456B), Qwen3 (Yang et al., 2025) (235B) reach a scale of hundreds of billion parameters while maintaining affordable training and inference efficiency. Typically, MoE comprises a *router* network and a group of *expert* modules. Given a set of inputs, the *router* distributes each input to its corresponding experts conditionally and sparsely. Then, the outputs from experts are aggregated based on the score that the router assigned to the expert.

One critical factor for training MoE-based models is encouraging the router to assign input to experts in a balanced manner (Fedus et al., 2022; Zoph et al., 2022; Qiu et al., 2024a). The reasons are twofold: (1) *effectiveness*: if the router continually prioritizes some experts during training, these experts will get more updates than others and will soon dominate that MoE layer, finally resulting in parameter redundancy issue (Shazeer et al., 2017; Wang et al., 2024); (2) *efficiency*: training and deploying large-scale MoE-based models often requires the *Expert Parallel*, where different experts will be in different parallel groups to process their inputs. Then, their outputs will be gathered and aggregated. In this case, the imbalanced expert utilization would heavily slow the forward process. In light of these two points, previous works training MoEs generally employ an auxiliary loss, called Load-balancing Loss (LBL), to encourage the balanced routing decision (Shazeer et al., 2017).

Nevertheless, in most open-source MoE training frameworks like Deepspeed-MoE (Liu et al., 2024a), Tutel (Hwang et al., 2023), Megablocks (Gale et al., 2023) and Megatron-Core (Shoeybi et al., 2019), the LBL is calculated at the *micro-batch level*, which, as we will soon empirically demonstrate, negatively affects the performance and expert specialization of MoE-based LLMs. Specifically, during large-scale MoE train-

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 5005–5018
July 27 - August 1, 2025



(a) Ablation of Balance Scope

(b) Expert Frequency for Micro-Batch Balance (Left) and Global-Batch Balance (Right)

Figure 1: The impact of the balance batch on different methods **(a)** and expert specialization **(b)**. **(a)** When only micro-batch level load balance is used, both methods based on LBL and auxiliary loss free approaches perform significantly worse than global-batch balance. **(b)** When only micro-batch balance is used, there is no significant difference in the selection frequency of different domain data, and the selection frequency of different experts within the same domain is essentially the same. With global-batch balance, there is a noticeable difference in the selection frequency of experts on different domain data, and within the same domain, there are experts with high selection frequency (marked in blue).

ing, each micro-batch usually contains only up to thousands of tokens and, thus, only a handful of sequences. Therefore, the micro-batch LBL is almost calculated at the *sequence level*. Suppose a micro-batch contains some domain-specific sequences (*i.e.* code and math), the micro-batch LBL still pushes routers to distribute these domain-specific tokens to all experts evenly, introducing an overly strict constraint and may hurt the model performance.

In this work, we propose calculating the LBL at the *global-batch* level by synchronizing the expert selection frequency across all parallel groups and then computing the LBL. According to the Fig. 1 (a), the global-batch LBL significantly enhances model performance (**approximately 0.1 in pre-training PPL and 2 in benchmark scores**). Fig. 1 (b) showcases that the domain specialization only clearly emerges when trained with the global-batch LBL. Despite the improved performance and enhanced specialization, we also demonstrate that the model performance effectively increases with the global batch size (Section 4.2). Our further ablation studies verify that introducing more diverse training tokens instead of more training tokens is the main contributor to performance gains (Section 5). Besides, because the expert selection frequency is just an expert-number-dimensional vector, our method introduces less than 3% latency under appropriate configurations and achieves more performant and interpretable models.

In summary, we investigate the challenges associated with the LBL in training MoEs. By introducing global-batch LBL, we achieve improved performance and foster expert specialization. This advancement addresses an essential limitation in existing MoE training, offering a novel perspective for MoE optimization. Notably, this method has been applied in *training the Qwen3-MoE models, enabling the released MoE models to exhibit clear domain specialization characteristics.*

2 Preliminary

2.1 Mixture-of-Experts

MoEs consist of several parallel modules (the ‘experts’) and a router that assigns weights to each expert for a given input. (Szymanski and Lemmon, 1993; Shazeer et al., 2017). Combined with the transformer layer (Vaswani, 2017), the most common approach is to introduce a set of parallel feed-forward networks (FFN). Suppose there are N_E experts, denoted as $E_i, i \in [1, N_E]$. The router g followed by a $\mathrm{softmax}$ function maps the input $\mathbf{x}$ to a score distribution over the experts, $\mathrm{softmax}(g(\mathbf{x})) \in \mathbb{R}^{N_E}$. Typically, for each input, only topK experts with the highest scores are activated and used. Given $\mathbf{x} \in \mathbb{R}^h$, the output $\mathbf{y} \in \mathbb{R}^h$ is the weighted sum of the outputs from all experts:

$$\mathbf{y} = \sum_{i \in N_E,\, g_i \in \mathrm{topK}(g(\mathbf{x}))} g_i(\mathbf{x}) E_i(\mathbf{x}) \quad (1)$$

2.2 Load-balancing Loss

The Load-balancing Loss (LBL) in training MoE models is a regularization technique that encourages balanced expert utilization (Fedus et al., 2022). Without the LBL, the model tends to concentrate its updates on a limited subset of experts, leading to a severe imbalance in expert utilization. To address this issue, LBL penalizes the router if it routes excessive tokens to a few particular experts. To compute LBL for a batch of tokens, we consider the fraction of tokens f_i routed to each expert E_i and the total routing probability P_i allocated to the expert E_i. The LBL is calculated as the sum of the product of f_i and P_i across all experts N_E, normalized by the number of experts:

$$\text{LBL} = N_E \sum_{i=1}^{N_E} f_i \cdot P_i. \tag{2}$$

By minimizing the load-balancing loss, the model is encouraged to distribute the considered tokens more evenly among the experts, ensuring that each expert receives a fair share of updates during training. This helps maintain a balanced utilization of experts and prevents the entire model from collapsing into only activating just a few experts.

However, when employing data parallelism and model parallelism strategies, each parallel group (*e.g.*, one GPU) only contains data from very limited domains. Existing MoE frameworks (Shoeybi et al., 2019; Gale et al., 2023) only utilize the information of P_i and F_i within every single parallel group to calculate LBLs and then average them:

$$\text{LBL}_{\text{micro}} = \frac{1}{N_P} \sum_{j=1}^{N_p} (N_E \sum_{i=1}^{N_E} f_i^j \cdot P_i^j), \tag{3}$$

where N_P is the number of parallel groups and f_i^j, P_i^j are the frequency and probability in parallel state j. This loss requires the model to *achieve load balance within each parallel group*, thus we call it $\text{LBL}_{\text{micro}}$. However, supposing one parallel group (one micro-batch) contains data from specific domains, the router is still pushed to distribute inputs uniformly to all experts, thereby preventing specialization. This situation is even more common regarding LLMs pretraining. Because to control the training data distribution, one micro-batch is usually packed with sequences from one specific domain, and a global-batch consists of micro-batches sampled from different domains according to particular data recipes (Ding et al., 2024; Yang et al., 2024). *So the micro-batch balancing will hinder the MoE model from allocating data from specific domains to specific experts*, which also partially explains why most MoE models only observe token-level expert routing patterns rather than expert-level selections. (Jiang et al., 2024; Xue et al., 2024).

3 Method

This section introduces how to turn the micro-batch LBL into global-batch LBL by allowing different parallel groups to synchronize their expert select frequencies. We then discuss the scenario in which the number of compute nodes is limited and the sum of micro-batches is smaller than the global batch size. In such cases, we propose using a buffer to store the synchronized expert select counts at each gradient accumulation (GA) step to approximate the global batch LBL.

Synchronizing expert selection frequency across parallel groups. Thanks to the format of the LBL in Eq.3, we can synchronize f_i across parallel groups to get $\bar{f}_i$ for the global batch. This allows the global averaged LBL to be equivalent to the LBL computed from statistics in the global-batch:

$$\begin{aligned}
\text{LBL}_{\text{global}} &= N_E \sum_{i=1}^{N_E} \bar{f}_i \cdot \bar{P}_i && (4)\\
&= N_E \sum_{i=1}^{N_E} \bar{f}_i \cdot (\frac{1}{N_P} \sum_{j=1}^{N_p} P_i^j) && (5)\\
&= \frac{1}{N_P} \sum_{j=1}^{N_p} (N_E \sum_{i=1}^{N_E} \bar{f}_i \cdot P_i^j) && (6)
\end{aligned}$$

Communicating $f_i \in \mathbb{R}^{N_E}$ avoids directly transmitting the token-expert selection matrix and the expert selection scores (with a shape of tokens numbers $\times$ experts numbers).

Using a buffer to approximate the Global-Batch LBL. When training LLMs, the global-batch size is often up to 10^3. When each micro-batch size is less than 10^1, due to the limited number of compute nodes, the sum of all micro-batch sizes is smaller than the global-batch size, thus gradient accumulation (GA) is often used. Therefore, we introduce a buffer to store synchronized c_i, the expert i's selection count across micro-batches in one GA step. Then, the information in the buffer is used to calculate the current f_i at each GA step. After completing the GA, the buffer is reset. The complete algorithm is shown in the Alg. 1 in the App. A.2 . Through this accumulation process, f_i approaches $\bar{f}_i$ with gradient accumulation steps, approximating $\text{LBL}_{\text{global}}$ with limited compute nodes.

Table 1: Performance of different balance methods and Balance BSZ. 'LBL' refers to using LBL, and Aux Free refers to the auxiliary loss free method (Wang et al., 2024). 'LBL+sync' means synchronizing expert selection frequency across parallel groups in 3. 'LBL+sync+buffer' means further using a buffer to expand the Balance BSZ in 3.

Balance Method	Balance BSZ	Hellaswag	MMLU	GSM8k	C-eval	Avg PPL
MoE-3.4A0.6B (Train 120B Tokens, Global Batch Size 512)						
LBL	4	62.81	41.63	13.57	41.87	8.167
LBL+sync	32	63.58	42.08	15.01	41.58	8.062
LBL+sync	512	**63.75**	**43.48**	**15.31**	44.95	**8.038**
Aux Free	4	61.99	41.30	12.43	43.53	8.521
Aux Free	512	63.51	42.74	14.18	**45.03**	8.080
MoE-3.4A0.6B (Train 400B Tokens, Global Batch Size 1024)						
LBL	4	67.21	48.97	21.30	49.02	7.347
LBL+sync	128	68.08	49.02	**28.81**	49.12	7.214
LBL+sync	512	**68.32**	**49.84**	25.40	**51.59**	**7.198**
LBL+sync+buffer	128	68.18	49.59	24.94	50.37	7.199
MoE-15A2.54B (Train 400B Tokens, Global Batch Size 1024)						
LBL	16	75.69	59.99	48.07	64.38	5.778
LBL+sync	512	**76.96**	**60.78**	**54.28**	**64.31**	**5.603**
MoE-43A6.6B (Train 120B Tokens, Global Batch Size 512)						
LBL	8	75.2	54.98	42.08	57.06	5.862
LBL+sync+buffer	128	**75.94**	**57.30**	**46.32**	**57.98**	**5.779**

4 Experiments

4.1 Experimental Setups

Model Architecture and Training Settings We conduct experiments on three sizes of MoE models: **(1)** 3.4B total parameters with 0.6B activated (**3.4A0.6B**); **(2)** 15B parameters with 2.54B activated (**15A2.54B**), and **(3)** 43B parameters with 6.6B activated (**43A6.6B**). Each model utilizes the fine-grained expert (Dai et al., 2024) and shared experts (Rajbhandari et al., 2022; Dai et al., 2024) methods. Specifically, the 3.4A0.6B model employs 64 experts with top4 activated and 4 shared experts, while the 15A2.54B and 43A6.6B models use a setting of 160 experts with top4 activated and 4 shared experts. All models default to using softmax gating and z-loss. The auxiliary loss weights follow previous works (Zoph et al., 2022). *To avoid the impact of token drop for different methods, we use the dropless routing strategy* (Gale et al., 2023). In the 3.4A0.6B setting, we also implement the auxiliary loss free (with sigmoid gating) method (Wang et al., 2024). We train the models on 120B and 400B high-quality tokens, encompassing multilingual, math, and general knowledge content. A sequence length of 4096 is used, with global-batch sizes of 512 and 1024 for the 120B and 400B training settings, respectively, comprising 60k and 100k training steps. We use the term **Balance BSZ** to indicate the number of tokens considered when calculating the expert selection frequency.

Evaluation We mainly test the zero-shot capabilities on four popular benchmarks, including English, Hellaswag (Zellers et al., 2019), general knowledge MMLU (Hendrycks et al., 2020), math GSM8k (Cobbe et al., 2021), and Chinese proficiency C-eval (Huang et al., 2024). Given that benchmarks that are evaluated with accuracy have certain random factors, for more detailed analysis, we mainly refer to the PPL on held-out test sets, which include SFT-EN, EN-Literature, SFT-Code, SFT-Math, SFT-ZH, ZH-Law, ZH-Literature, and SFT-Other from different domains.

4.2 Main Results

Global load balance boosts model performance. In this section, we compare the performance of using micro-batch and global-batch loss. The 3.4A0.6B models are trained only with data parallelism and a micro-batch size 4. If f_i is synchronized among the 8 GPUs on the same node, the Balance BSZ is 32. When training with 16 nodes and synchronizing across data parallel groups, the Balance BSZ can reach 512. From the first part of Tab. 1, it can be seen that *as the Balance BSZ increases, all metrics consistently improve*. For the aux-free method, we also compare the results under micro-batch and global-batch conditions and find the latter is much more better. For the 3.4A0.6B model trained on 400B tokens, we compare the results when the Balance BSZ could only reach 128 due to the limited compute nodes with the results of using a buffer to approximate the global-batch. The latter's performance is closer to the results with a Balance BSZ of 512 and significantly better than 128, proving that introducing a buffer can approximate the global-batch when nodes are limited. As training the 15A2.54B and 43A6.6B models requires using model parallelism strategies, we employ expert parallelism for both models, allowing a micro-batch size of 2 and 1 per GPU, respectively. We compared the results of synchronizing f_i within the same machine and across all data parallel groups, as shown in the last two parts of Tab. 1. It is evident that increasing the Balance BSZ also significantly improves larger models.

Global load balance encourages expert specialization. We further analyse the selection frequency of each layer's experts across different domains using held-out PPL test sets. Specifically, we record the selection frequency for each expert for each domain. In Fig. 1, we compare the expert selection distributions under SFT-Code, SFT-Math, and EN-Literature for models trained with micro-batch balance and global-batch balance. It can be observed that (1) with micro-batch balance, most of the selection frequency is the same under EN-Literature, and only a few experts have slightly higher frequencies under SFT-Code and SFT-Math, yet none exceed 0.15. This aligns with existing analysis about MoE specialization: models using default LBL hardly exhibit domain-level specialization and only show token-level specialization (Jiang et al., 2024; Xue et al., 2024). (2) In contrast, with global-batch balance, more pronounced high-frequency experts emerge, with many experts in SFT-Math having frequencies exceeding 0.2. This confirms that global-batch balance is more conducive to domain specialization.

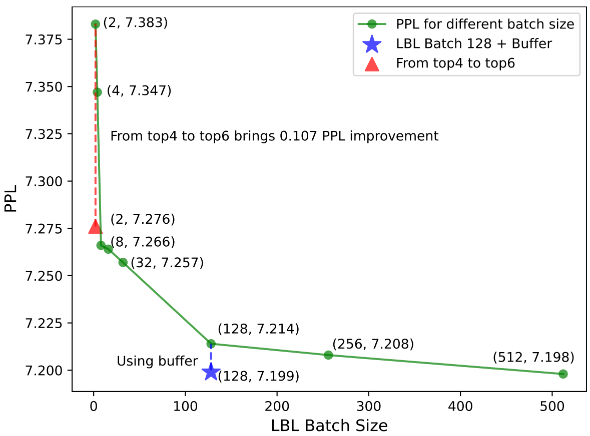


Figure 2: The performance of MoE-3.4A0.6B trained on 400B tokens with different Balance BSZ.

Model performance increases with Balance BSZ. To further illustrate the impact of Balance BSZ, we control the micro-batch size, synchronization scope, and number of devices in training the 3.4A0.6B model on 400B tokens, and plot the test PPL from a Balance BSZ of 2 (micro-batch size 2, without any synchronization for expert selection frequency) to 512 as shown in Fig. 2. As the Balance BSZ increases, the test PPL consistently decreases, **with an overall decrease of 0.185 from 2 to 512**. It is also noticeable that the improvement rate slows down after increasing to 128, and the result of adding the buffer is very close to that of 512. *This indicates that synchronization and buffer mechanisms can bring significant improvements compared to micro-batch in MoE training across various computing node scales.* Additionally, we supplement experiments by increasing the activation from top4 experts to top6 experts under the micro-batch condition and found that the improvement brought by a 50% increase in activated expert FLOPs is even less than the improvement from increasing the Balance BSZ from 2 to 8. Furthermore, expanding the Balance BSZ is efficient since the additional overhead from synchronization and buffer is much less than that from increasing the number of activated experts and FLOPs.

5 Analysis

Table 2: Ablation of the number of tokens and distributional bias for computing LBL on MoE-3.4A0.6B .

LBL type	Hellaswag	MMLU	Avg PPL
120B Tokens, Global Batch Size 512, Micro Batch Size 4			
Micro	62.81	41.63	8.167
Global	**63.75**	**43.48**	**8.038**
Shuffle	63.57	43.37	8.041
400B Tokens, Global Batch Size 1024, Micro Batch Size 2			
Micro	67.22	48.77	7.383
Global	68.32	**49.84**	**7.198**
Shuffle	**68.43**	49.68	7.214

Ablation Study on Token Numbers and Token Distributional Bias As aforementioned, one possible factor for global-batch LBL to outperform micro-batch LBL is that the latter pushes the router to achieve sequence-level balanced expert utilization, which may be overly stringent. However, another naive assumption is that the $\text{LBL}_{\text{global}}$ involves more tokens to estimate the expert selection frequency, thus reducing the variance and ameliorating the MoE training. To verify, we introduce an ablation setting: *Shuffle* $\text{LBL}_{\text{micro}}$. Specifically, when calculating LBL, we first synchronize the token-expert score matrix G (with a shape of number of tokens $\times$ number of experts), where $G_{ij} = 1$ if the token i selects the expert j, other-

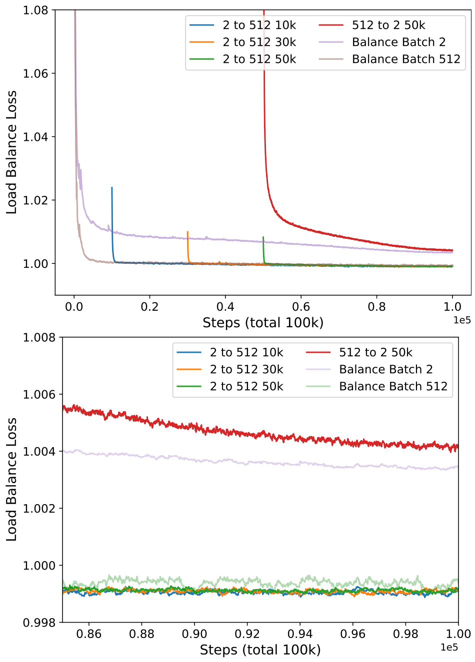


Figure 3: The LBL curve for MoE-3.4A0.6B trained on 400B tokens under different Balance BSZ, with a zoom-in of the last 15k steps shown below.

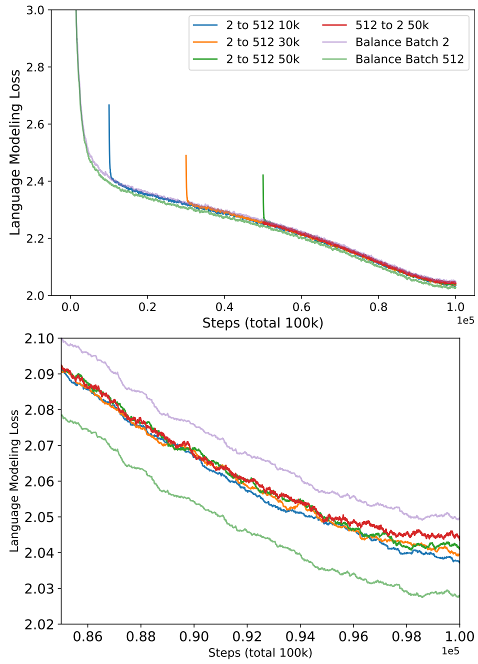


Figure 4: The language modeling loss curve for MoE-3.4A0.6B corresponding to Fig. 3

wise $G_{ij} = 0$. Then, we randomly select a batch of tokens (without replacing) to calculate the expert selection frequency, where the batch size is equal to the micro-batch size. In this setting, the random batch has the same token numbers as the micro-batch and identical token distribution as the global-batch, enabling us to tell the difference between these two confounders. The results are shown in the Tab. 2. We observe that the *Shuffle* $\text{LBL}_{\text{micro}}$ achieves similar results as $\text{LBL}_{\text{global}}$, and outperforms the $\text{LBL}_{\text{micro}}$, verifying the motivation of our paper and the assumption about the improvement.

$\text{LBL}_{\text{global}}$ is a looser constraint than $\text{LBL}_{\text{micro}}$. Intuitively, global-batch balance is a looser constraint than micro-batch balance: the former only requires that tokens be evenly distributed globally, while the latter demands uniform distribution within each micro-batch. In Fig. 3, we show the loss curves of the two methods using the same load balance weight for MoE-3.4A0.6B trained on 400B. Additionally, we add the results of switching from micro-batch balance to global-batch balance at 10k, 30k, and 50k training steps. It can be observed that (1) after switching to global-batch balance, the LBL rapidly decreases to a range close to that when the global-batch balance is used from scratch, and the final convergence trend is also similar. This is because transitioning from a tighter constraint (balance within a micro-batch) to a looser one (balance within a global-batch) is relatively easy. (2) Moreover, if global batch balance is switched to micro-batch balance at the 50k step, the originally converged load balance first rises to a much higher range, then slowly decreases, and the final convergence loss is still higher than that of micro-batch balance used from scratch. This indicates that transitioning from a looser constraint to a tighter one can significantly alter the convergence state.

Table 3: The impact of changing the Balance BSZ during training on the final results. Step indicates the step at which the Balance BSZ is switched.

Balance BSZ	Step (/100k)	PPL
2	-	7.383
2→512	50k	7.322
2→512	30k	7.297
2→512	10k	7.283
512	-	**7.199**
512→2	50k	7.373

In Fig. 4, we present the language modeling loss curves. The corresponding test PPL is in Tab. 3. It can be observed that (1) the loss of global-batch balance is over 0.02 lower than that of micro-batch balance, corresponding to the large performance gap between the two as shown in Tab. 3. (2) Switching from micro-batch to global-batch balance results in performance improvements, with earlier switches yielding better outcomes. However, even the switch at the 10k step is inferior to training with global-batch balance from scratch. This aligns with existing findings that router choices tend to become fixed early in training (Xue et al., 2024; Muennighoff et al., 2024b): although increasing the Balance BSZ at any training stage can bring benefits, the router trained with micro-batch balance has already saturated very early, thus the gains from switching during training are limited. (3) Switching from global-batch to micro-batch balance degrades performance.

Table 4: Results for different load balance weight.

Balance BSZ	LBL weight	Hellaswag	MMLU	Avg PPL
4	0.008	62.81	41.63	8.167
4	0.004	62.95	42.13	8.154
4	0.001	62.97	41.71	8.159
512	0.008	**63.75**	**43.48**	**8.038**

Since micro-batch balance is a tighter constraint than global-batch balance, we further test reducing the load balance weight of micro-batch balance in Tab 4. It can be observed that appropriately reducing the LBL weight can slightly improve the model's performance, but too small LBL weight leads to worse results. This may be due to the overly imbalanced distribution affecting expert utilization. Moreover, the performance of micro-batch balance under various LBL weights is inferior to that of global-batch balance, further highlighting the differences between the two balancing methods.

Table 5: Performance and speed (seconds per iteration) in 43A6.6B setting. '128+buffer & 8' means adding micro-batch balancing loss with Balance BSZ 8.

Balance BSZ	Hellaswag	MMLU	Avg PPL	Speed/s
8	75.20	54.98	5.862	**1.55**
128+buffer	**75.94**	**57.30**	**5.779**	1.64
128+buffer & 8	75.87	57.00	5.795	1.59

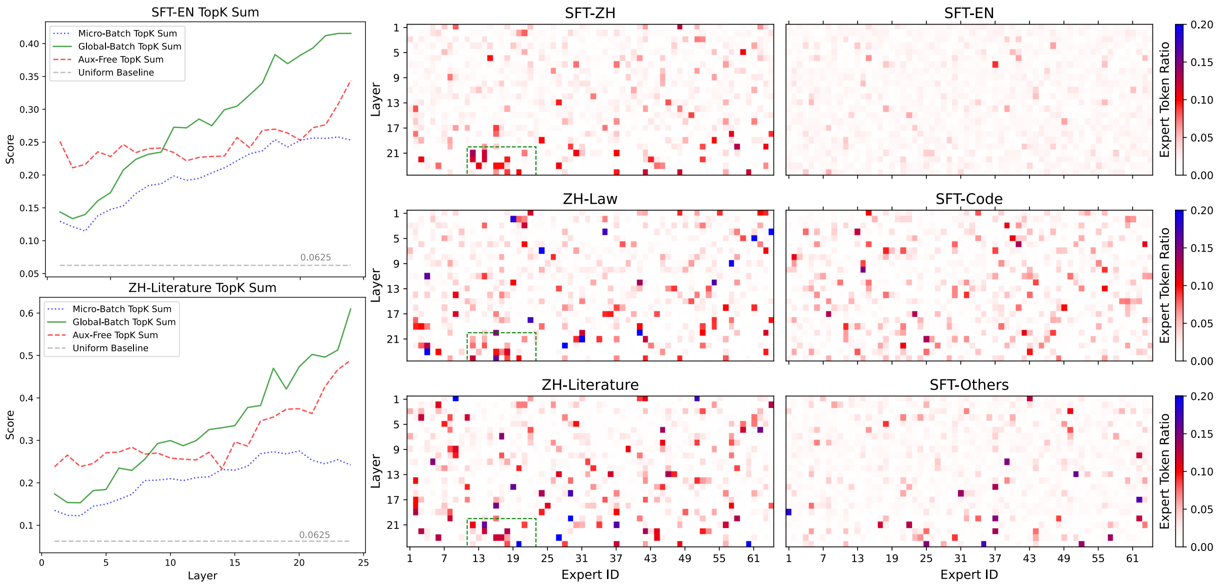
The training efficiency of global-batch balance. Because a dropless strategy is employed, the FLOPs calculation is identical across different methods. However, due to differences in local balance conditions, methods using global-batch balance may experience local computational imbalance. To address this, we recorded the speed and results of micro-batch balance and global-batch balance during the training of the 43A6.6B model in Tab. 5. (1) It can be seen that the speed using global-batch balance (1.64 s/iteration) is 5.8% slower than micro-batch balance (1.55 s/iteration). Further analysis revealed that about 1% of this slowdown is due to communication overhead within all data parallel groups, the remain is due to local expert load imbalance under the dropless strategy. Drawing inspiration from sequence-level LBL, we introduced a very low-weighted (1% of the global-batch weight) micro-batch balancing loss into the global-batch balance at the 20k step and continued training the model. We found that (2) adding a small amount of micro-batch balancing loss increased the speed to 1.59 s/iteration (2.6% slower than the baseline) with only a minimal decrease in performance. *It should be noted that since the computation of LBL is independent from other parts of the network and takes very little time, it can be overlapped to further reduce the efficiency gap to within 2%.*

Global batch balance brings interpretable specialization. We further analyze the specialization of models using global-batch balance. In Fig. 5 (a), we record the scores assigned to each expert by tokens across different domains and calculate the average of the topK score sums. When all experts are assigned identity scores, the topK sum is illustrated by the uniform baseline (gray dashed line). We can observe: (1) Models using global-batch balance have a higher topK sum. Since the LBL and z-loss in MoE encourage routing scores to be uniform, while only the language modeling loss encourages an increase in routing scores, this suggests that *under the global-batch balance, routing is more aligned with the language modeling task*. (2) Models using global-batch balance have a larger topK sum in domains where expert selection is more concentrated. For example, in Fig. 5 (b), the high-frequency experts in ZH-Literature are more than those of SFT-EN, especially in layers 17 to 24. Correspondingly, in Fig. 5 (a), the topK sum of ZH-Literature in layers 17 to 24 is higher than that of SFT-EN. (3) Models using micro-batch balance have lower topK sums, with little difference across domains, which corresponds to the existing work that current MoE routing is uncertain (Wu et al., 2024). (4) Under global-batch balance, the topK sum of using aux loss free is smaller than that of LBL, but higher than micro-batch balance. *This also illustrates that expert specialization promotes the concentration of expert scores.*

In Fig. 5 (b), we compare the distribution of high-frequency experts across domains. We observe that Chinese domains (SFT-ZH, ZH-Law, ZH-Literature) have many similar high-frequency experts (indicated by the dashed box). Moreover, although both Chinese-related domains and SFT-Code have high-frequency activated experts, these experts hardly overlap. For domains with more general content (such as SFT-EN), there are fewer experts being highly activated.

6 Related works

Load Balancing Shazeer et al. (2017) introduce the topK sparse activation in MoE (Szymanski and Lemmon, 1993), which tends to elect only a few experts for updates during training without constraints. Although LBL can alleviate this issue, strict constraints affect model performance. Ex-



(a) TopK score sum across layers

(b) Expert selection frequency across domains

Figure 5: The topK score sums across layers (a), and the distribution of high-frequency experts on different domains for models using global-batch balance (b). The topK sum of global-batch balance is higher than other methods and shows a similar distribution of high-frequency experts on closer domains.

pert Choice Routing (Zhou et al., 2022) achieves load balance naturally by allowing each expert to select tokens based on its load capacity. However, it uses the information of the entire sequence when allocating tokens, making it non-causal and impractical for decoder-only models. Although subsequent work adds extra routers and training phases to address this, it has only been valuated when only using 2 experts (Raposo et al., 2024). Wang et al. (2024) proposes the Aux Loss Free method, which adds a bias term updated based on expert selection frequency to balance expert selection. However, they don't emphasize whether the expert selection frequency is calculated based on micro-batch or global-batch. The subsequent work deepseek-v3 (Liu et al., 2024a), concurrent with ours, highlights that the expert selection frequency in Aux Loss Free is based on 'the whole batch of each training step' and discusses the results of using batch-wise LBL and Aux Loss Free method, also finding that the two methods yield similar results. GRIN (Liu et al., 2024b) proposes Global Load Balance Loss Adaptations. However, the it mainly introduces this as an advantage of the training framework without employing expert parallelism. It doesn't show the effects of using global load balance independently and emphasizes the importance and properties of global load balance. More discussions can be found in App. A.1.

Expert Specialization Initially, MoE is designed to *devide and conquer*, allowing different experts to specialize strongly for efficient parameter utilization (Szymanski and Lemmon, 1993; Qiu et al., 2024b). With the tight micro-batch balance, most MoE models (Jiang et al., 2024; Lo et al., 2024; Zhao et al., 2024; Du et al., 2024), even multimodal MoEs (Lin et al., 2024; Team, 2024), haven't exhibited domain-level specialization. Lory (Zhong et al., 2024) calculates expert merge scores for each sequence and merges all experts into a single expert before computing the corresponding sequence. This changes the sparse activation mechanism of MoE and avoids the imbalance issue. Although Lory shows improvements and specialization, its complex mechanism poses challenges for large-scale training. OLMoE (Muennighoff et al., 2024a) observes more pronounced specialization compared to Mixtral-8×7B. However, it does not provide a detailed discussion of the factors influencing specialization.

7 Conclusion

In this work, we identify that the LBL in mainstream MoE frameworks has degraded into micro-batch balance, which imposes an overly tight constraint. This constraint limits expert specialization and negatively impacts performance. To address this issue, we propose methods based on synchronization and buffering to relax micro-batch balance to global-batch balance. We validate these methods across models of various sizes. Through analysis of expert selection under global-batch balance, we observe that it enables domain-level and interpretable specialization. We hope that adopting the global-batch balance will facilitate developing more performant and interpretable MoE-based LLMs.

Limitations

This paper primarily focuses on analyzing the impact of micro-batch LBL on LLMs during the pre-training stage. It does not further investigate its effects during fine-tuning or in the vision and multimodal domains. Our analysis of specialization is mainly centered on the selection frequency across different domains without conducting more rigorous validation. Relaxing micro-batch LBL can introduce some latency. Future work could consider including more diverse sequences within each micro-batch to mitigate this local imbalance.

References

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y Wu, et al. 2024. Deepseek-moe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*.

Hantian Ding, Zijian Wang, Giovanni Paolini, Varun Kumar, Anoop Deoras, Dan Roth, and Stefano Soatto. 2024. Fewer truncations improve language modeling. *arXiv preprint arXiv:2404.10830*.

Wenyu Du, Shuang Cheng, Tongxu Luo, Zihan Qiu, Zeyu Huang, Ka Chun Cheung, Reynold Cheng, and Jie Fu. 2024. Unlocking continual learning abilities in language models. *arXiv preprint arXiv:2406.17245*.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23:120:1–120:39.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. 2023. Megablocks: Efficient sparse training with mixture-of-experts. *Proceedings of Machine Learning and Systems*, 5:288–304.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Yao Fu, et al. 2024. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *Advances in Neural Information Processing Systems*, 36.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, et al. 2023. Tutel: Adaptive mixture-of-experts at scale. *Proceedings of Machine Learning and Systems*, 5:269–287.

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

Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, Cheng Zhu, Chunhao Zhang, Congchao Guo, Da Chen, Dong Li, et al. 2025. Minimax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv:2501.08313*.

Bin Lin, Zhenyu Tang, Yang Ye, Jinfa Huang, Junwu Zhang, Yatian Pang, Peng Jin, Munan Ning, Jiebo Luo, and Li Yuan. 2024. Moe-llava: Mixture of experts for large vision-language models. *Preprint*, arXiv:2401.15947.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Liyuan Liu, Young Jin Kim, Shuohang Wang, Chen Liang, Yelong Shen, Hao Cheng, Xiaodong Liu, Masahiro Tanaka, Xiaoxia Wu, Wenxiang Hu, Vishrav Chaudhary, Zeqi Lin, Chenruidong Zhang, Jilong Xue, Hany Awadalla, Jianfeng Gao, and Weizhu Chen. 2024b. Grin: Gradient-informed moe. *Preprint*, arXiv:2409.12136.

Ka Man Lo, Zeyu Huang, Zihan Qiu, Zili Wang, and Jie Fu. 2024. A closer look into mixture-of-experts in large language models. *arXiv preprint arXiv:2406.18219*.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. 2024a. Olmoe: Open mixture-of-experts language models. *Preprint*, arXiv:2409.02060.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. 2024b. Olmoe: Open mixture-of-experts language models. *arXiv preprint arXiv:2409.02060*.

Zihan Qiu, Zeyu Huang, Shuang Cheng, Yizhi Zhou, Zili Wang, Ivan Titov, and Jie Fu. 2024a. Layerwise recurrent router for mixture-of-experts. *Preprint*, arXiv:2408.06793.

Zihan Qiu, Zeyu Huang, and Jie Fu. 2024b. Unlocking emergent modularity in large language models. *Preprint*, arXiv:2310.10908.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 2022. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pages 18332–18346. PMLR.

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

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

Peter T. Szymanski and Michael D. Lemmon. 1993. Adaptive mixtures of local experts are source coding solutions. In *Proceedings of International Conference on Neural Networks (ICNN'88)*, pages 1391–1396. IEEE.

Chameleon Team. 2024. Chameleon: Mixed-modal early-fusion foundation models. *Preprint*, arXiv:2405.09818.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. 2024. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*.

Haoze Wu, Zihan Qiu, Zili Wang, Hang Zhao, and Jie Fu. 2024. Gw-moe: Resolving uncertainty in moe router with global workspace theory. *arXiv preprint arXiv:2406.12375*.

Fuzhao Xue, Zian Zheng, Yao Fu, Jinjie Ni, Zangwei Zheng, Wangchunshu Zhou, and Yang You. 2024. Openmoe: An early effort on open mixture-of-experts language models. *arXiv preprint arXiv:2402.01739*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

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

Chenggang Zhao, Shangyan Zhou, Liyue Zhang, Chengqi Deng, Zhean Xu, Yuxuan Liu, Kuai Yu, Jiashi Li, and Liang Zhao. 2025. Deepep: an efficient expert-parallel communication library. `https://github.com/deepseek-ai/DeepEP`.

Hao Zhao, Zihan Qiu, Huijia Wu, Zili Wang, Zhaofeng He, and Jie Fu. 2024. Hypermoe: Towards better mixture of experts via transferring among experts. *arXiv preprint arXiv:2402.12656*.

Zexuan Zhong, Mengzhou Xia, Danqi Chen, and Mike Lewis. 2024. Lory: Fully differentiable mixture-of-experts for autoregressive language model pre-training. *Preprint*, arXiv:2405.03133.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. 2022. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. 2022. St-moe: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*.

A Example Appendix

A.1 More Related Works

Load Balancing Wang et al. (2024) argue that the load balance loss, which is not entirely consistent with the language modelling loss, can impact model performance. Therefore, they propose adding a bias term updated based on expert selection frequency to balance expert selection without changing routing scores. However, they don't emphasize whether the expert selection frequency is calculated based on micro-batch or global-batch. The subsequent work deepseek-v3 (Liu et al., 2024a), concurrent with ours, highlights that the expert selection frequency in Aux Loss Free is based on 'the whole batch of each training step' and discusses the results of using batch-wise load balance loss and auxiliary free method, also finding that the two methods yield similar results. In this work, we propose synchronizing expert selection and buffering methods that can be easily integrated into existing MoE frameworks, leading to improvements under various computational configurations. Our work also provides a detailed analysis of Balance BSZ's impact on performance and demonstrates that global-batch significantly improves performance by incorporating more diverse domain information. Additionally, we show that adding a small amount of micro-batch load balance while using global-batch balance can maintain model performance while reducing latency from local imbalance. Another concurrent work, Minimax-01 (Li et al., 2025), synchronizes expert select frequency within expert parallel groups, primarily aiming to reduce the drop rate of experts when using drop strategies (Fedus et al., 2022), without focusing on the impact of different Balance BSZ.

GRIN (Liu et al., 2024b) proposes Global Load Balance Loss Adaptations. However, the it mainly introduces this balance method as an advantage of the training framework without employing expert parallelism. GRIN does not present more motivation for using global load balance. Additionally, it does not show the effects of using global load balance independently and emphasizes the importance and properties of global load balance.

A.2 Using a buffer to approximate the Global-Batch LBL.

We introduce a buffer to store synchronized c_i, the expert i's selection count across micro-batches in one GA step. Then, the information in the buffer is used to calculate the current f_i at each GA step. After completing the GA, the buffer is reset. The complete algorithm is shown in the Alg. 1. Through this accumulation process, f_i approaches $\bar{f}_i$ with gradient accumulation steps, approximating $\text{LBL}_{\text{global}}$ with limited compute nodes. We also provide the PyTorch implementation in the Listing 1.

A.3 More Discussion on Computation Overhead

The latency introduced by global-batch balancing primarily arises from two factors: synchronization and micro-batch imbalance. Both issues can be effectively mitigated through deployment-level optimizations during training and inference.

Algorithm 1 Approximate Global-Batch LBL

```
Initialize an empty buffer for each expert, c_i = 0
Initialize accumulated gradients to zero
while training continues do
    for each gradient accumulation step g = 1 to G do
        Forward pass: compute loss using current micro-batch
        Backward pass: accumulate gradients (without applying optimizer step)
        Update buffer: add new synchronized selection counts to c_i for each expert i
        Calculate the current f_i using buffered c_i, i ∈ N_E (to approximate a larger bsz)
    end for
    Apply optimizer step using accumulated gradients
    Clear accumulated gradients
    Reset buffer: set c_i = 0 for all experts
end while
```

Synchronization: To reduce synchronization overhead, we explore methods that decrease the frequency of synchronization operations. By leveraging frequency information stored in a buffer, we reduce synchronization to occur only once per optimizer step, significantly lowering communication costs (Appendix A.7). Additionally, system-level optimizations allow for overlapping synchronization with forward computation, thereby eliminating communication overhead entirely during this phase, as communication resources are underutilized while computation is ongoing. During inference, synchronization is not required at all since there is no need to compute the LBL loss.

Micro-Batch Imbalance: Micro-batch imbalance can be alleviated during inference by increasing the micro-batch size through adjustments in parallelization strategies. When the token distribution within a micro-batch closely resembles that of the global batch, the computational load across micro-batches becomes more balanced. Furthermore, infrastructure-level optimizations also play a crucial role in addressing expert imbalance. For instance, the DeepEP (Zhao et al., 2025) framework supports dynamic allocation of additional computing resources to frequently activated experts when deploying MoE models using expert parallelism. This ensures that high-load experts do not become bottlenecks for overall system performance.

```
# init buffer for tokens per expert; do not buffer across iteratio
self.tokens_per_expert_buffer = 0

# compute the number of tokens per expert
probs = torch.softmax(logits, dim=-1)
probs, top_indices = torch.topk(probs, k=self.topk, dim=-1)
tokens_per_expert = torch.histc(top_indices, bins=self.num_experts,
                                min=0, max=self.num_experts)

# sync the number of tokens per expert across data parallel group
if self.config.moe_router_sync_tokens_per_expert_across_dp:
    with torch.no_grad():
        torch.distributed.all_reduce(tokens_per_expert,
                    group=get_data_parallel_group())
        tokens_per_expert = tokens_per_expert / torch.distributed.get_world_size(
                                group=get_data_parallel_group()))

# update the number of tokens per expert buffer
if self.config.moe_router_buffer_tokens_per_expert:
    self.tokens_per_expert_buffer = self.tokens_per_expert_buffer +
    tokens_per_expert
    tokens_per_expert = self.tokens_per_expert_buffer

# compute LBL
.....
# reset the buffer if optimizer step is called
# therefore, the buffer doesn't expand the balance batch beyond global BSZ
optimizer.step()
if self.config.moe_router_reset_tokens_per_expert_buffer:
    self.tokens_per_expert_buffer.zero_()
```

Listing 1: Pytorch style code for synchronizing and buffering tokens per expert

It is worth emphasizing that our primary objective is to investigate the impact of balance bsz on MoE specialization and model performance. As such, detailed discussions on deployment-level optimizations are beyond the scope of this paper.

A.4 More Experiments in Hard Tasks

To further evaluate the effectiveness of expert specialization in hard tasks, we conduct additional experiments using the MoE-43A6.6B model under two different training configurations: (1) balance bsz = 8, and (2) balance bsz = 128 with the buffer mechanism. These settings align with those reported in Table 1. We assess performance across several code-related and reasoning-intensive benchmarks, including HumanEval, MBPP, BBH, and MMLU-pro. The results are summarized below:

Table 6: Performance on hard tasks for the MoE-43A6.6B model under different balancing strategies.

Balance BSZ	MBPP	HumanEval	BBH	MMLU-pro
8	40.6	29.27	41.41	22.79
128 + buffer	**42.6**	**33.53**	**42.74**	**24.01**

As shown in Table 6, increasing the balance bsz while incorporating the buffer mechanism leads to improved performance across all evaluated benchmarks. This suggests that larger balance bsz promote better expert specialization, particularly in complex domains such as code generation and multi-step reasoning.

A.5 Global-Batch Balance with Token Dropping

We also test global-batch balance with token dropping under a capacity factor of 1. We observe that the drop ratio is significantly higher than using only micro-batch balance. For example, in the scenario of selecting 4 out of 160 experts, when using the default LBL weight and micro-batch balance, approximately 10% of the tokens are dropped. However, if global-batch balance is used from the beginning, the drop ratio would be around 30%. A large number of tokens being dropped leads to a significant reduction in FLOPs, which in turn makes the result of global-batch balance similar to that of micro-batch balance. We recommend that if token dropping is to be introduced when using global-batch balance, it is best to follow the approach described in Sec 5: start with dropless training, then add micro-batch balance, and finally introduce a certain capacity factor constraint.

```python
# init buffer for tokens per expert; enable buffering across iteratio
_TOKENS_PER_EXPERT = [0]

# functions
def update_tokens_per_expert(tokens_per_expert):
    global _TOKENS_PER_EXPERT
    _TOKENS_PER_EXPERT[-1] = _TOKENS_PER_EXPERT[-1] + tokens_per_expert
    return torch.stack(_TOKENS_PER_EXPERT, dim=0).sum(dim=0)

def reset_tokens_per_expert():
    args = get_args()
    global _TOKENS_PER_EXPERT
    if len(_TOKENS_PER_EXPERT) < args.moe_router_buffer_capacity:
        _TOKENS_PER_EXPERT.append(0)
    elif len(_TOKENS_PER_EXPERT) == args.moe_router_buffer_capacity:
        _TOKENS_PER_EXPERT = _TOKENS_PER_EXPERT[1:] + [0]
    else:
        raise ValueError

def sync_tokens_per_expert():
    global _TOKENS_PER_EXPERT
    temp_tpe = _TOKENS_PER_EXPERT[-1]
    torch.distributed.all_reduce(temp_tpe,
                                 group=get_data_parallel_group())
    _TOKENS_PER_EXPERT[-1] = temp_tpe / torch.distributed.get_world_size(
    group=get_data_parallel_group())

...
# compute the number of tokens per expert
probs = torch.softmax(logits, dim=-1)
probs, top_indices = torch.topk(probs, k=self.topk, dim=-1)
tokens_per_expert = torch.histc(top_indices, bins=self.num_experts,
                                min=0, max=self.num_experts)

# locally update the number of tokens per expert buffer
if self.config.moe_router_buffer_tokens_per_expert:
    tokens_per_expert = update_tokens_per_expert(tokens_per_expert)

# compute LBL
.....
# reset part of the buffer if optimizer step is called
# therefore, the buffer properly expands the balance batch beyond global BSZ
optimizer.step()
if self.config.moe_router_reset_tokens_per_expert_buffer:
    # sync only when one iteration is finished
    # get and buffer tokens per expert for current iteration
    sync_tokens_per_expert()
    # clear old tokens per expert
    reset_tokens_per_expert()
```

Listing 2: Pytorch style code for buffering tokens per expert and only synchronizing at each iteration

A.6 Expand Buffer Capacity

A natural question arises: if model performance improves with the growth of the balance batch, could expanding the balance batch beyond the global batch size through the buffer mechanism further enhance the benefits? Our experiments find:

(1) When training from scratch, if the buffer retains the tokens per expert statistics from the past three iterations to compute the current LBL, *the convergence speed of LBL will significantly slow down and ultimately fail to converge near 1*. We think this is because the router changes rapidly in the early stages of training, causing the previously recorded expert balance statistics to deviate significantly from the actual situation, which in turn introduces bias into the calculated LBL.

(2) In the middle stages of training, if the buffer retains statistics from the past two or three iterations to compute the LBL, the model performance is similar to that when using only one iteration's statistics. *This observation allows us to approximate the results obtained through global communication in the current iteration using the statistics* *from the previous iteration*, see next part A.7. Consequently, this approach can reduce the frequency of synchronization across data parallel groups.

(3) Even in the middle stages of training, when the buffer capacity is expanded to eight iterations, the LBL gradually increases during training, which negatively impacts model performance. This indicates that although the model's balance situation is relatively stable in the middle stages of training, using an incorrect LBL can still cause the model to gradually deviate from the desired balance.

A.7 Decrease Synchronization Frequency

In our large-scale experiments, we observe that when the data parallel group is very large (*e.g.*, 2048 GPUs), synchronizing tokens per expert at every update step is highly susceptible to cluster performance fluctuations. Specifically, if one node computes more slowly, the entire cluster is delayed while waiting for the synchronization of tokens per expert. Building on our previous experiments with small buffer sizes, we further optimized the synchronization method as follows:

Early training phase (approximately 10k iterations, within 5% of total training steps): When the LBL has not yet converged, we maintain synchronization at every step, and the buffer only records information from the current iteration.

Stabilized phase: Once the LBL converges and training becomes relatively stable, we decrease the synchronization frequency. Specifically, we use the expanded buffer in App. A.6 to store the information from the past 2 to 3 iterations (global batches). During each step of the current iteration, we calculate the LBL using locally computed tokens per expert plus the information stored in the buffer. The local tokens per expert are then updated to the buffer. After the current iteration ends (optimizer steps), we synchronize the locally calculated tokens per expert of this iteration in the buffer across the data parallel group to obtain accurate statistics for the iteration.

Iteration Transition: The oldest iteration's information in the buffer is discarded, and the process begins for the next iteration. For the specific implementation, please refer to Listing 2. By reducing the frequency of cross-data parallel group synchronization, we can mitigate latency even when training with a large number of nodes.