HARMOENY: Efficient Inference of MoE Models

Zachary Doucet*, Rishi Sharma†, Martijn de Vos†, Rafael Pires†, Anne-Marie Kermarrec†, Oana Balmau*
*McGill University, Canada
†EPFL, Switzerland

***Abstract*—Mixture-of-Experts (MoE) models offer computational efficiency during inference by activating only a subset of specialized experts for a given input. This enables efficient model scaling on multi-GPU systems that use expert parallelism without compromising performance. However, load imbalance among experts and GPUs introduces waiting times, which can significantly increase inference latency. To address this challenge, we propose HARMOENY, a novel solution to address MoE load imbalance through two simple techniques: *(i)* dynamic token redistribution to underutilized GPUs and *(ii)* asynchronous prefetching of experts from the system to GPU memory. These techniques achieve a near-perfect load balance among experts and GPUs and mitigate delays caused by overloaded GPUs. We implement HARMOENY and compare its latency and throughput with four MoE baselines using real-world and synthetic datasets. Under heavy load imbalance, HARMOENY increases throughput by 37%–70% and reduces time-to-first-token by 34%–41%, compared to the next-best baseline. Moreover, our ablation study demonstrates that HARMOENY's scheduling policy reduces the GPU idling time by up to 84% compared to the baseline policies.**

***Index Terms*—Mixture-of-Experts Models, Machine Learning Inference, GPU Scheduling, Load Balancing, Expert Parallelism**

I. INTRODUCTION

Scaling machine learning (ML) models to billions of parameters has enabled powerful generative models [1], [2]. One of the main applications of these models is natural language processing, where large language models (LLMs) such as Llama [3] and GPT-4 [4] are widely used for tasks like text generation and question answering. However, these applications come with steep costs and high energy consumption. As model sizes grow, inference becomes increasingly expensive [5], and it already constitutes the majority of ML workloads. NVIDIA and AWS estimate that up to 90% of the ML workloads are serving deep neural network models [6], [7]. To address this critical challenge, this paper focuses on strategies for efficient ML inference.

One promising approach is the use of Mixture-of-Experts (MoE) models [8]. Compared to traditional models, MoE models can provide more than 10× reduction in computation requirements for inference, without sacrificing accuracy [9], [10]. The Switch Transformers [10], Mixtral [11], Qwen [12], and DeepSeek [13] model families are some of the most successful MoEs. Each expert in an MoE model is trained to focus on a specific subset of tasks or data patterns. A gating mechanism, often a smaller neural network called a *router*, decides which experts will process a given input. This approach allows MoE models to scale, using only a fraction of their capacity per input, resulting in computational savings.

‡This work has been partially funded by the Swiss National Science Foundation, under the project "FRIDAY: Frugal, Privacy-Aware and Practical Decentralized Learning", SNSF proposal No. 10.001.796.

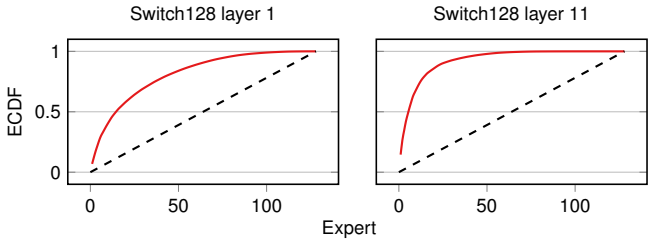


Fig. 1: The ECDF of token placement across all 128 experts on the BOOKCORPUS dataset for the SWITCH128 [10] model. The workload exhibits strong skew where a handful of experts receive over half the total number of tokens. This skew results in load imbalance among the GPUs.

Despite the benefits in terms of computation, MoE models have a significant memory footprint. While only a fraction of experts are activated per input, all the experts need to be available in GPU memory to process a batch of requests. Therefore, serving MoE models requires a combination of multiple GPUs, each holding a subset of the experts, also known as expert parallelism (EP).

Serving MoE models using multiple GPUs and EP has two significant bottlenecks that increase the end-to-end inference latency: *(i)* synchronization, and *(ii)* load imbalance among the GPUs [14]–[17]. First, MoEs models require two synchronization steps (all-to-all communication) between GPUs in every MoE block. Based on the decisions of the router in each MoE block, each GPU first scatters its inputs to the relevant experts on other GPUs and gathers them back after the computation is done. These synchronizations are essential to ensure that all GPUs have the complete results from the current MoE block before moving on to the next one. Recent MoE training and inference frameworks have alleviated this slowdown by optimizing the all-to-all communication [14], [15], [17]. They either resort to replication [15] of popular experts or collocate popular experts across MoE layers [14], [17]. In these cases, the popularity of experts is determined through offline profiling and scheduling. However, expert popularity is dynamic, as it depends on the tokens activated by the input, which in turn are influenced by the domain of the request (*e.g.*, medical prompts activate different tokens than StackOverflow questions) [11].

Indeed, depending on the workload, some experts are significantly more *popular* than others and are assigned more

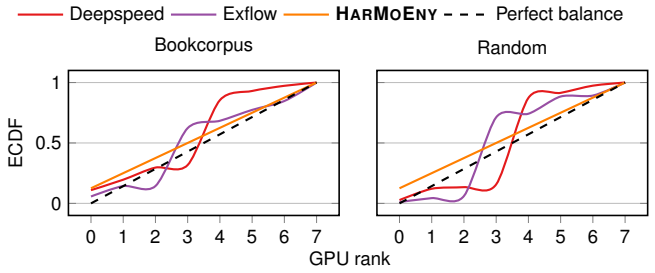


Fig. 2: The ECDF of token placement across GPUs in an 8-GPU NVIDIA DGX machine, when using two datasets, for a SWITCH128 model [10] with 128 experts. HARMOENY achieves near-perfect load balancing.

tokens than other experts. We show this in Figure 1 for the SWITCH128 MoE model during a run with the BOOKCORPUS dataset. Token placements across experts are non-uniform, and tokens are disproportionately routed to a small subset of experts per input. For instance, in the first layer of the SWITCH128 model, three experts receive an average of 17.4% of the tokens, whereas in the final (11th) layer, three experts receive as much as 34.5% of the tokens on average. This skew becomes more pronounced in deeper layers of the model and is dynamic, with the token distribution varying across queries. This skew in expert popularity leads to a load imbalance among the GPUs and to significant GPU under-utilization.

In this paper, we target load imbalance caused by skewed expert popularity to improve the efficiency of multi-GPU MoE inference. Expert popularity imbalance (as shown in Figure 1) leads to significant imbalance in GPU use. Figure 2 shows the load imbalance across GPUs for two popular MoE load balancing approaches, DEEPSPEED and EXFLOW, and our solution, HARMOENY. DEEPSPEED uses a round-robin distribution of tokens to GPUs (also used by others such as FASTMOE [16], and FASTERMOE [15]). Another compelling approach used by EXFLOW [14] is workload profiling and integer programming to determine the optimal expert placement. This technique yields better load balance at times, but is not fast enough to adapt to skew changes across batches. Both approaches yield an imbalanced distribution of tokens to experts, leading to high GPU idle times, as we will show in Section III. We found that this idle time can take up to 86% of the time for GPUs housing unpopular experts. In contrast, Figure 2 shows that HARMOENY, our solution, achieves near-perfect load balance.

HARMOENY uses two simple, yet powerful techniques to achieve load balance among GPUs: *token rebalancing* from overutilized to underutilized GPUs, and *asynchronous prefetching of experts* from system to GPU memory. HARMOENY adapts on the fly to changes in expert popularity with no drops in throughput and does not need any profiling.

To achieve this, HARMOENY modifies the MoE logic. During each batch, the GPUs exchange a summary of the token distribution to get a global vision of the token-to-GPU and token-to-expert assignment. The metadata in this step is small (4kB) and introduces minimal overhead. Then, each GPU can deterministically infer *the same* token-to-expert and token-to-GPU schedule in parallel, with no further synchronization needed. HARMOENY balances the token load such that some of the tokens that were destined to overutilized GPUs (*i.e.*, hosting more popular experts) are rerouted to underutilized GPUs. To ensure that the rerouted tokens can be processed by the popular experts once they are rebalanced to the underutilized GPUs, HARMOENY uses asynchronous expert prefetching. HARMOENY can look ahead and make sure that the right experts are paged into GPU memory from the system memory, and the ones that are not needed anymore are discarded. Swapping out an expert from GPU memory only requires an overwrite.

HARMOENY uses prefetching and load rebalancing, both well-established techniques in datacenter scheduling [18], [19]. However, we are the first to adapt such scheduling techniques to MoE models and show they eliminate GPU idleness almost completely. We compare MoE inference in HARMOENY to four state-of-the-art MoE systems: EXFLOW [14], FASTMOE [16] and FASTERMOE [15], and DEEPSPEED-TUTEL [20]. We show that in workloads with skewed expert popularity, HARMOENY is up to 41.1% faster than the next-best baseline, in terms of time-to-first-token. Our simple and efficient approach reduces the waiting time of GPUs in the all-to-all synchronization step by up to 84.7% compared to baseline policies. Furthermore, HARMOENY maintains stable and low inference latency even as skew and expert popularity changes. Thanks to its lightweight techniques, HARMOENY has the capacity to quickly adapt to various datasets, which is a drawback of profiling-based approaches.

Contributions. This paper makes the following contributions:

1) We empirically study the compute utilization of a GPU cluster running MoE inference and conclude that expert popularity imbalance has a much higher impact on inference latency than all-to-all synchronization (Section III).
2) We design and implement HARMOENY (Section IV). HARMOENY uses two complementary techniques to achieve almost perfect load balancing: token rebalancing and asynchronous prefetching of experts. HARMOENY is open source[1] and implemented on top of PYTORCH.
3) We evaluate HARMOENY with real datasets and synthetic benchmarks, showing that HARMOENY maintains a low and steady inference latency across different datasets, with different skew levels and batch sizes (Section V).

II. BACKGROUND

Transformer models are nowadays widely used for ML tasks [21]. A typical transformer model consists of multiple transformer blocks, each designed to process tokens through self-attention mechanisms and Feed-Forward Networks (FFNs). A *token* here refers to an intermediate value representing a single element, *e.g.*, a word [22], a subword [23], or a character [24]. A transformer block takes some

[1]See https://doi.org/10.5281/zenodo.18700393.

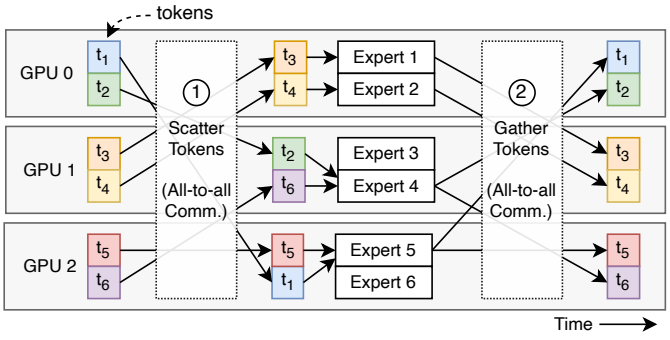


Fig. 3: Token scattering (step 1) and gathering (step 2) when using expert parallelism. Experts are split across GPUs. One batch requires two all-to-all synchronization barriers.

tokens as input and consists of two main components: a self-attention mechanism and an FFN. The self-attention mechanism captures relationships between input elements across the sequence, allowing the model to focus on different parts of the input simultaneously. These outputs are passed into an FFN, which applies two dense layers with an activation function in between to refine the representations. The FFN is the most time-consuming part of the transformer block [25]. The resulting output tokens are then forwarded to the subsequent transformer block.

Mixture-of-Experts (MoEs) is a type of sparse computation that selectively activates only parts of the network, called *experts* [8]. In an MoE model, some or all transformer blocks can have the MoE layer which replaces the FFN. We refer to a transformer block having the MoE layer as an *MoE block*. In contrast to a transformer block with a single FFN, an MoE layer contains multiple experts, implemented by smaller FFNs, with each expert having its own set of weights [26]. MoEs assign each token to only a subgroup of experts, rather than passing it through the entire model.

The assignment of tokens to experts is managed by the *router*, a component responsible for directing each token to a subset of experts. The router is usually implemented as a trainable function [26], optimized through backpropagation to discover productive token-to-expert assignments. To prevent bottlenecks, a loss term during training encourages the router to distribute tokens evenly across the experts. The router assigns each expert a value between 0 and 1. Then, expert assignment is handled with a *top-k* strategy—directing the token to the k experts with the highest values, with each expert's output weighted accordingly. The parameter k is usually set to 1 or 2, as higher values quickly raise costs while offering diminishing returns [10], [11]. After processing the expert FFN, the token's output is combined, normalized, and passed to the next layer in the model.

Expert parallelism (EP). In multi-GPU scenarios, transformers typically use three types of parallelism: *(i)* data parallelism, where input data is sharded across the GPUs [27], *(ii)* model parallelism, where different parts of the model are split across the GPUs at a layer- or component-granularity [28], and *(iii)* tensor parallelism, where large tensors (*i.e.*, matrix operations within a single layer) are split across GPUs [29]. Expert parallelism (EP) [16] combines aspects of data parallelism and model parallelism to support MoEs.

With EP, the self-attention and router layers are replicated across GPUs (data parallelism), but each GPU only loads a subset of experts, distributing the complete set of experts across all GPUs [10] (model parallelism). During the forward pass, each GPU receives a minibatch of the input request comprising a set of input tokens. All the GPUs independently compute self-attention on their minibatches in parallel. Here, the routers in each GPU assign the tokens from the mini-batches to experts. Since the tokens and their assigned experts can potentially be on different GPUs, an all-to-all scatter communication step ensures that each GPU receives the tokens destined for the experts it hosts. This step introduces the first synchronization barrier in MoE blocks. Upon receiving the correct tokens, each GPU performs the expert computation on the received tokens using the experts it holds. After the expert computation, an all-to-all gather communication step returns the computed results for each token back to the GPUs responsible for their corresponding inputs. Finally, the resulting tokens are then used as input for the next layer.

Figure 3 shows an EP assignment with 6 experts split across 3 GPUs. Experts 1 and 2 would be housed in GPU 0, experts 3 and 4 to GPU 1, and experts 5 and 6 to GPU 2. The self-attention mechanism outputs tokens t_1 to t_6. When GPU 0 processes its batch, the router assigns tokens to experts—sending a tokens to expert 1, b to expert 2, c to expert 3, and so on. Since GPU 0 only houses experts 1 and 2, it must send tokens for experts 3 and 4 to GPU 1 and tokens for experts 5 and 6 to GPU 2. Once GPU 0 completes computations for its experts, it returns the processed tokens to their originating GPUs through another all-to-all communication, posing another synchronization barrier.

III. Effect of Load Imbalance in MoE Inference

MoE routers are trained to balance the token load across the experts [10], [11]. However, during inference, the expert selection, and hence the computation load across GPUs is often skewed (see Figure 2). We now show the impact of skewed expert popularity and load imbalance on performance.

We first show that *expert popularity skew causes significant GPU under-utilization* leading to high end-to-end latency, in line with prior work. Figure 4 (a) presents the time breakdown of GPU computations when serving a Switch transformer model with 128 experts. We inject artificial token skew such that 90% of the tokens are assigned to the first 10 experts. We run the experiment with 8x V100 GPUs (see Section V-A for the full experimental setup). In this workload, GPU 0 is assigned the most popular experts. Due to the all-to-all synchronization steps, GPUs 1–7 remain idle for more than 82% of the time, increasing the inference latency. In this situation, balancing the token load across GPUs and consequentially minimizing the waiting times would lead to efficient serving and reduced end-to-end latency of inference requests.

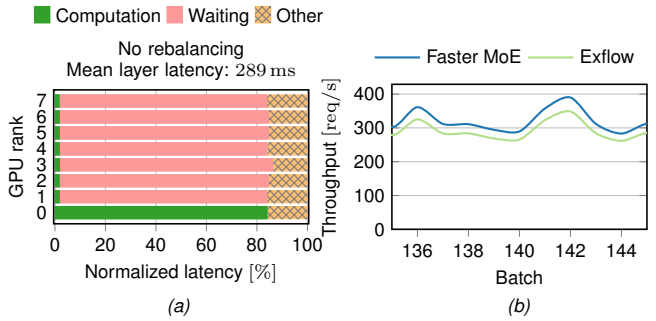


Fig. 4: Static expert placement causes long waiting due to load imbalance *(a)* and fluctuating throughput *(b)*.

Shortcomings of existing solutions. Prior work improved the efficiency of MoE model serving in the case of mostly-static expert popularity [14]–[17]. Simple replication of the popular experts onto the under-utilized GPUs as proposed by FASTERMOE [15] suffices for balancing loads in workloads with static expert popularity. In scenarios where expert popularity skew slowly changes over time, profiling can be done over time and expert placement can be adjusted periodically as done in EXFLOW [14].

Realistically, however, the skew and consequently the load imbalance fluctuate between batches. This variation arises because the skew depends on the token distribution within the input requests, which is in turn influenced by the domain of the requests (*e.g.*, medical, programming, etc.) [11]. The skew fluctuation results in unstable throughput across batches for static solutions like FASTMOE. Furthermore, profiling-based solutions are inefficient for MoE serving in the presence of dynamic skew due to their cost. For instance, in our test bed, profiling and readjusting the schedule with integer programming in EXFLOW takes about 8.5 minutes for the Switch transformer and as much as 45 minutes for experiments with the Qwen MoE model. In contrast, even with low GPU utilization, the mean time to process a single batch of requests through a MoE block is only 289 ms as shown in Figure 4 (a). Therefore, EXFLOW is unable to quickly adapt to the dynamic expert popularity.

Figure 4 (b) shows the impact on the throughput, in terms of completed requests per second, of FASTERMOE [15] and EXFLOW [14] in a longer run of the same setup as Figure 4 (a) with dynamically fluctuating expert-token skew across batches. FASTERMOE starts with a round-robin placement of experts on GPUs and EXFLOW uses 40 samples from the BOOKCORPUS dataset to create a schedule for expert placement on the GPUs using integer programming (see Section V-A). We can see that EXFLOW's and FASTERMOE's throughput fluctuates across batches and drops by up to 37.6% within just two consecutive batches. Both systems have similar performance as neither system has time to adapt to the rapid fluctuations in the skew.

Algorithm 1: HARMOENY MoE Layer

Require: G: Set of GPUs.
Procedure FORWARD(x)**:**
  // **Step 1: token routing**
  $m_{expert} \leftarrow$ ROUTER(x)
  // **Step 2: metadata exchange**
  SENDMETADATATOGPUS(m_{expert})
  receive m_{all}[i] from each GPU $i \in G$
  // **Step 3: token scheduling**
  $S_{initial} \leftarrow$ INITIALASSIGN(m_{all})
  $S \leftarrow$ REBALANCE($S_{initial}$) ▷ See Section IV-B
  // **Step 4: scatter tokens**
  SENDTOKENSTOGPUS(x, m_{expert}, S)
  receive x' from all other GPUs
  // **Step 5: expert processing and async. loading**
  $x'' \leftarrow$ EXPERTS(S, x') ▷ See Section IV-C
  // **Step 6: gather tokens**
  SENDTOKENSBACKTOGPUS(S, x'')
  receive $y[i]$ from each GPU $i \in G$
  $x \leftarrow$ RECONSTRUCT(S, y, m_{all})
  return x

IV. HARMOENY DESIGN

Based on our findings in Section III, we build HARMOENY as a system to reduce inference latency for MoE models. HARMOENY is composed of two main components: *(i)* a scheduler that load-balances tokens across experts, and *(ii)* an expert pre-fetching protocol that asynchronously prefetches an expert into GPU memory. These two techniques reduce GPU idle time introduced by token load imbalance without any profiling and allow HARMOENY to adapt to rapid workload fluctuations. We first explain the overall workflow of HARMOENY, and then explain each of these two components.

A. HARMOENY *workflow*

Algorithm 1 shows the operations during a forward pass through the MoE logic with HARMOENY. During the forward pass, the FORWARD function is executed by each GPU, taking input tokens x. x is a tensor of shape [batch size, sequence length, hidden dimension]. We assume that input tokens x have already been passed through the self-attention layer and will now be routed to and processed by the relevant experts. This proceeds in the following six steps.

Step 1: token routing. Input tokens x are first assigned to specific experts based on their characteristics using a router. This assignment creates a token-to-expert mapping, m_{expert}, which is a tensor of integers representing the target expert for each token.

Step 2: metadata exchange. Next, all GPUs exchange their computed token-to-expert distribution. Specifically, each GPU broadcasts its local token-to-expert assignments, allowing all GPUs to build a global, shared understanding of the token distribution. This metadata exchange step requires very little communication (typically a few kilobytes) and thus is efficient. This information is stored in an array m_{all} which tracks the token-to-expert assignment for all other GPUs. This metadata exchange step ensures that HARMOENY token scheduling operates with a complete view of the workload. We note that this exchange is unique to HARMOENY, and we show in Section V that its overhead in end-to-end latency is negligible.

Step 3: token scheduling. Based on the token-to-expert assignments in m_{all}, an initial (naive) token schedule $S_{initial}$ is generated. $S_{initial}$ is a three-dimensional tensor of integers where each entry represents the number of tokens assigned from a source GPU to a destination GPU for a particular expert. Since this might result in load imbalance issues, HARMOENY then *rebalances* this schedule through an algorithm that redistributes tokens from overburdened GPUs to underutilized ones. We explain this in Section IV-B.

Step 4: scatter tokens. Based on the rebalanced schedule and the local token-to-expert assignment m_{expert}, each GPU now sends its input tokens in x to the designated GPUs through an all-to-all communication step. Each GPU i then receives all tokens x' from other GPUs that should be processed by the receiving GPU.

Step 5: expert processing and asynchronous loading. The tokens in x' are then processed by the appropriate experts, resulting in x''. It might be that our rebalancing algorithm assigns tokens to experts that are not currently housed by a particular GPU. HARMOENY employs a novel, asynchronous expert pre-fetching protocol to ensure that the required expert weights are loaded into GPU memory without delaying token processing. Weight transfers are overlapped with computation, reducing idle GPU time. We explain this in Section IV-C.

Step 6: gather tokens. After expert processing is completed, each GPU sends the processed tokens x'' back to their original GPUs using a second all-to-all communication step. This ensures that each GPU receives the results for the tokens it initially routed. Once all tokens are collected, the received tokens are restructured to ensure that the processed tokens are aligned correctly with their original order and source. This completes the processing of input tokens x by the MoE logic, yielding the final output tokens.

B. Load-aware token scheduler

One of HARMOENY's innovations is an efficient load-aware token scheduler that assigns each token to one of the available GPUs. The scheduler can rebalance the load by re-assigning tokens that are destined to one GPU to a less crowded GPU. We outline the rebalancing algorithm in Algorithm 2 and visualize this process in Figure 5. The procedure REBALANCE takes as input an initial schedule $S_{initial}$, which is then assigned to the variable S. $S_{initial}$ and S are three-dimensional arrays, containing a mapping between source GPUs, experts, and destination GPUs. Specifically, $S[g_{\text{from}}, e, g_{\text{to}}]$ denotes the number of tokens sent from GPU g_{from} for expert e to GPU g_{to}. Thus, the first dimension corresponds to the source GPUs, the second to the experts, and the third to the destination GPUs where these experts are housed.

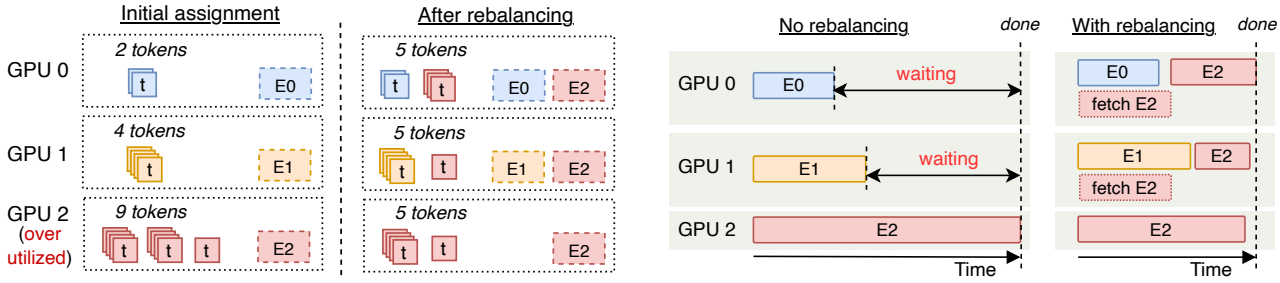
Algorithm 2: HARMOENY Token Rebalancing

Require: G: Set of GPUs, E: Set of experts, q: Token transfer threshold.
Output : Rebalanced schedule S

Procedure REBALANCE($S_{initial}$):
- $S \leftarrow S_{initial}$
- $t_{\text{avg}} \leftarrow \lfloor S.\text{SUM}()/|G| \rfloor$ ▷ Avg. tokens per GPU
- $t_g \leftarrow S.\text{SUM}(\text{dim} = (0, 1))$ ▷ Token count per GPU
- **while** ANY($t_g > t_{avg}$) **do**
  - $g_{\text{max}} \leftarrow \text{ARGMAX}(t_g)$
  - $g_{\text{from}} \leftarrow \text{ARGMAX}(\text{SUM}(S[:, :, g_{\text{max}}], \text{dim} = 1))$
  - $e_{\text{max}} \leftarrow \text{ARGMAX}(S[g_{\text{from}}, :, g_{\text{max}}])$
  - $t_{\text{move}} \leftarrow S[g_{\text{from}}, e_{\text{max}}, g_{\text{max}}]$
  - **if** $t_{move} < q$ **then**
    - **return** S ▷ Insufficient tokens to move
  - $g_{\text{min}} \leftarrow \text{ARGMIN}(t_g)$ ▷ Find least loaded GPU
  - **if** $g_{min} = g_{max}$ ***or*** $t_g[g_{min}] + q > t_{avg}$ **then**
    - **return** S ▷ No feasible transfer
  - $t_s \leftarrow \min(t_{\text{move}}, t_{\text{avg}} - t_g[g_{\text{min}}])$
  - $S[g_{\text{from}}, e_{\text{max}}, g_{\text{max}}] \mathrel{-}= t_s$
  - $S[g_{\text{from}}, e_{\text{max}}, g_{\text{min}}] \mathrel{+}= t_s$
  - $t_g[g_{\text{max}}] \mathrel{-}= t_s$
  - $t_g[g_{\text{min}}] \mathrel{+}= t_s$
- **return** S

Imbalances arise due to uneven distribution across experts (see Section III). We show an example in Figure 5 which illustrates a system with 3 GPUs and 15 input tokens, assigned to three different experts, and using top-1 routing (*e.g.*, in each MoE layer, each token is processed by a single expert). The color of each input token in Figure 5 (a) indicates the expert that the router has assigned to the token. Assume that experts 0, 1, and 2 are located on GPU 0, 1, and 2, respectively. Naively assigning input tokens to experts will result in GPU 2 having to process 9 tokens and GPU 0 just two tokens, resulting in load imbalance. Figure 5 (b, left) shows a timeline with operations per GPU. The disproportionate load on GPU 2 causes the computation time of expert 2 to grow, introducing waiting times for GPU 0 and 1 and prolonging the inference request duration.

HARMOENY rebalances experts across GPUs by analyzing and manipulating the expert-to-GPU assignment in S. If a



(a) During rebalancing, tokens are reassigned from overutilized to underutilized GPUs.

(b) Timeline with operations per GPU, with and without rebalancing. Experts are fetched asynchronously (see Section IV-C).

Fig. 5: An example of the token rebalancing process by HARMOENY. Setup: 15 input tokens, three GPUs and three experts.

GPU receives more tokens than the average allocation, it is considered overutilized. The policy then identifies the least utilized GPU and redirects as many tokens as possible to this GPU without causing it to become overutilized. We visualize this in Figure 5 (a, right) where expert 2 is replicated to GPUs 0 and 1 as well, resulting in a situation where each GPU now processes an equal amount of tokens. This process repeats until either all GPUs have a balanced token load, or there are no further offloading options available.

We next provide a detailed description of our greedy scheduling algorithm in Algorithm 2. To this end, we first compute the average number of tokens t_{avg} and the total number of tokens t_g that each GPU has to process (S.SUM() returns the total number of tokens across all GPUs and experts). The rebalancing loop operates iteratively to reduce load imbalances across GPUs (line 6).This loop runs as long as there is a GPU that receives more than the average number of tokens t_{avg}. In each batch, HARMOENY identifies the index of the most overloaded GPU, referred to as g_{max}, which has the highest total token count. The scheduler then determines the source GPU g_{from} that contributes the largest share of tokens to g_{max} (line 8). The term SUM(S[:, :, g_{max}], dim=1) calculates the total number of tokens contributed by each source GPU to the overloaded GPU g_{max}, summing over all experts. We then identify within g_{from} the expert e_{max} that is responsible for sending the most tokens to g_{max} (line 9).

Once the relevant source GPU and expert are determined, the algorithm calculates the number of tokens t_{move} to potentially transfer (line 11). If t_{move} is smaller than some token threshold q, the algorithm stops the process, as moving such a small number of tokens would not sufficiently reduce the imbalance. The token threshold q is an important hyperparameter of HARMOENY that decides the lower bound number of tokens necessary to offload tokens to another GPU. The reason for introducing this threshold is to account for the time of loading an expert from memory to overlap the token processing with the expert fetching (see Figure 5 (b, right)). Moving a very small number of tokens might not sufficiently reduce the imbalance to justify the cost of communication overhead, such as reconfiguring schedules and initiating transfers between GPUs. This would result in minimal performance gains or even a net loss in efficiency. q depends on the system specifications and is independent of dynamic aspects of the workload such as the expert popularity in a given batch. We discuss how to determine this hyperparameter in Section IV-C.

If there are sufficient tokens to move, the algorithm then identifies GPU g_{min} that is assigned the least number of tokens (line 15). Tokens from g_{from} and e_{max}, destined for g_{max}, are then redirected to g_{min}, ensuring that g_{min} does not exceed the average load t_{avg}. Specifically, the exact number of tokens transferred, t_s, is the smaller of t_{move} or the remaining capacity of g_{min} (line 19). After transferring tokens, the corresponding entries in S are updated to reflect the new distribution of tokens. The total token counts in t_g are also adjusted accordingly. This rebalancing step is repeated until either all GPUs have token counts close to t_{avg} or no further feasible transfers can be made (*e.g.*, $t_{move} < q$).

The ECDFs in Figure 2 illustrate the distribution of tokens assigned to each GPU across two different datasets. DEEPSPEED, lacking any token load rebalancing mechanism, results in substantial load imbalances, with certain GPUs processing significantly more tokens than others. In contrast, HARMOENY employs an effective rebalancing algorithm that dynamically redistributes tokens, ensuring a near-uniform workload across all GPUs. This improvement is consistent across all evaluated datasets, showcasing HARMOENY's robustness and adaptability to different input distributions. By mitigating load skew, HARMOENY significantly enhances inference efficiency as we will show in Section V.

C. Asynchronous expert fetching

To handle load balancing effectively, the HARMOENY scheduler may assign tokens for certain experts to GPUs that currently do not have these experts loaded in GPU memory. For example, Figure 5 (a) shows that after rebalancing, GPUs 0 and 1 get assigned some tokens designated for expert 2, which is currently not in their memory. Thus, HARMOENY needs a method to efficiently load experts into GPU memory as needed.

Simply loading experts after each expert completes processing introduces delays in the inference pipeline. This approach is inefficient, as the GPU must wait for the current expert to be

offloaded from GPU memory before loading the next expert's weights from the system memory. Given the typically large size of expert weights (*e.g.*, each expert in the SWITCH128 and QWEN models is 18 MB and 33 MB, respectively), this results in frequent stalls where the GPU is idle, waiting for data transfers to complete.

To achieve this efficiently, HARMOENY prefetches experts *asynchronously*, enabling transfers of experts' weights off the critical path. Specifically, once an expert completes processing its allocated tokens, it checks for any remaining experts that need to run but are not currently loaded. If one is found, HARMOENY fetches the weights for this next expert from system memory, directly *overwriting* the memory location of the expert that completed its processing. We show an example of this in Figure 5 (b, right), where GPU 0 and GPU 1 will asynchronously fetch expert 2 while computing with expert 0 and expert 1, respectively. This technique significantly speeds up operations compared to the traditional approach of first writing the current expert to system memory and then loading the new expert into GPU memory, as the offloading to system memory is not needed. Our measurements show that overwriting can speedup expert loading by **5.5x**: from 11 ms to 2 ms for V100 GPUs. We note that HARMOENY benefits from asynchronous expert fetching if at least two experts fit in the GPU memory, a requirement for any system serving MoE models with many experts.

HARMOENY's prefetching protocol relies on the preceding experts to process enough tokens so that the asynchronous weight transfer can be completed before the next expert starts. If the computation of a particular expert is much quicker than the expert transfer time, the gains of this approach diminish. This is influenced by the token threshold q. Thus, if q is chosen appropriately, the transfer time is effectively masked, minimizing idle periods and maintaining efficiency.

Our asynchronous expert fetching protocol resembles DMA-based prefetching techniques and overlap of data transfer with computation, but HARMOENY integrates these mechanisms with MoE-specific scheduling decisions. More broadly, the techniques used by HARMOENY also draw inspiration from classic datacenter scheduling, yet their novelty lies in adapting them to MoE inference. Classic work stealing, *e.g.*, HAWK [30], targets batch or throughput-oriented workloads rather than per-request, low-latency serving. Such approaches are reactive, *i.e.*, workers probe others' queues and fetch work on demand, which introduces additional latency. In the context of MoE inference, this behavior would amplify contention: when a single expert is heavily favored, multiple workers would attempt to steal work from the same GPU. In contrast, HARMOENY leverages router information to proactively compute token and expert placement for each batch and asynchronously prefetches experts ahead of execution.

D. Determining the token threshold q

The token threshold q (see Section IV-B) influences the number of additional experts each GPU has to load. A small value of q causes tokens to be offloaded to GPUs without the corresponding expert and not enough tokens to amortize the cost of fetching the expert from memory. However, a high value of q might not sufficiently address the token load imbalance and not balance the processing times of GPUs.

Ideally, we fix q such that the time to execute an expert exceeds the time to load a new expert. Let $|O|$ be the number of required floating operations to execute a particular expert, ϕ the FLOPS of the GPU being used, $|E|$ the size of an expert in bytes, and β the PCIe bandwidth in bytes per second. Then:

$$\frac{|O|}{\phi} > \frac{|E|}{\beta} \tag{1}$$

Experts are typically two-layer MLPs, with the first one, W^1, being of size $m \times p$, and the second one, W^2, being $p \times m$. Also, let d_{type} be the size of an element in W^1 or W^2 and q the number of tokens being processed by the expert. The expert computation can be expressed as xW^1W^2 where x are the input tokens. Thus, the size $|E|$ of an expert is:

$$|E| = (mp + pm)d_{type} \tag{2}$$

The number of operations required to complete the expert computation with q tokens is given by:

$$|O| = qp(2m - 1) + qm(2p - 1) \tag{3}$$

By plugging Equations (2) and (3) into Equation (1), rearranging terms, and ignoring negligible factors, we get the following inequality:

$$q > \frac{\phi \cdot d_{type}}{2\beta} \tag{4}$$

It is important to note that q only depends on the system specification and the bit-precision of the parameters being served and does not depend on any dynamic properties. Therefore, the lower bound of Equation (4) provides a reliable approximation for q.

V. EVALUATION

We implement HARMOENY and compare its latency and throughput with baseline systems.

A. Experimental setup

We implement HARMOENY in 1115 lines of code in PYTORCH [31]. Our implementation achieves asynchronous expert fetching through a dedicated NVIDIA CUDA stream for expert loading. The MoE layer is written as an `nn.Module` in PYTORCH for expert parallelism. The implementation is modular and can be applied to any PYTORCH model, as we demonstrate by evaluating the performance of HARMOENY with different models.

MoE models. We evaluate HARMOENY using two MoE models: SWITCH128 [10] and QWEN [12]. SWITCH128 is a language model that extends the T5 architecture [32] by replacing its feed-forward layers with MoE logic. It contains 12 total transformer blocks alternating between a MoE block and a regular transformer block. Each MoE block in this model

has 128 experts, where each expert is 18 MB in serialized form. We use the QWEN 1.5 MoE model which features 24 transformer blocks each of which has 60 experts, where each expert is 33 MB in serialized form.

Hardware. All experiments are performed on a DGX1 machine featuring eight NVIDIA V100 GPUs (each with 32 GB GPU memory) interconnected with NVLINK, and 500 GB of system memory. Industry practice shows that older GPUs remain widely deployed for inference while cutting-edge hardware is reserved for training [33], making our evaluation on V100 GPUs practically relevant.

Metrics. In our evaluation, we use two key metrics to quantify system performance: throughput and mean time-to-first-token (TTFT). Throughput is calculated as the total number of tokens generated across the experiment divided by the experiment length. Mean TTFT is a commonly-used metric that captures the average latency between the initiation of a request and the generation of the first token, reflecting the responsiveness of the system during inference. Thus, HARMOENY targets the prefill phase because it directly determines TTFT. Prefill processes entire input sequences simultaneously, creating large token volumes where load imbalance is most pronounced.

1) MoE System Baselines: We compare HARMOENY against four baselines:

DEEPSPEED [20], with TUTEL enabled to support efficient inference. We set the capacity factor to prevent DEEPSPEED from dropping tokens, for a fair comparison. DEEPSPEED uses a round-robin placement of experts on the GPUs.[2]

FASTMOE is a system for distributed training of MoE models [16] that relies on optimized CUDA kernels for rapid data movement and expert selection. FASTMOE also uses expert parallelism.

FASTERMOE addresses token load imbalance in FASTMOE through expert replication and fine-grained scheduling [15]. FASTERMOE also introduces a topology-aware gating function that directs inputs to the experts with lower latency. This function reduces communication overhead by prioritizing local expert assignments.

EXFLOW optimizes expert placement based on inter-layer expert affinity [14]. The key innovation lies in leveraging the observed tendency for tokens to follow predictable routing patterns across consecutive MoE layers.

2) Datasets: We use three real datasets and two synthetic datasets. (1) BOOKCORPUS is a dataset comprising 7185 unique books originally collected from smashwords.com [34]. (2) WIKITEXT is a collection of 100 million tokens scraped from *Good* and *Featured* articles on Wikipedia [35]; (3) WMT19 includes translation pairs between two languages [36]. We work with the German-to-English set with 34.8 million translation pairs. (4) RANDOM is a dataset constructed by stringing random tokens in a sequence of a set length, starting from a seed. (5) CONSTANT is a dataset where a single token is repeated to a set length for all batches.

[2]DEEPSPEED-MII, separate to DEEPSPEED, is not evaluated as the framework cannot execute on pre-Ampere GPUs, such as the V100s.

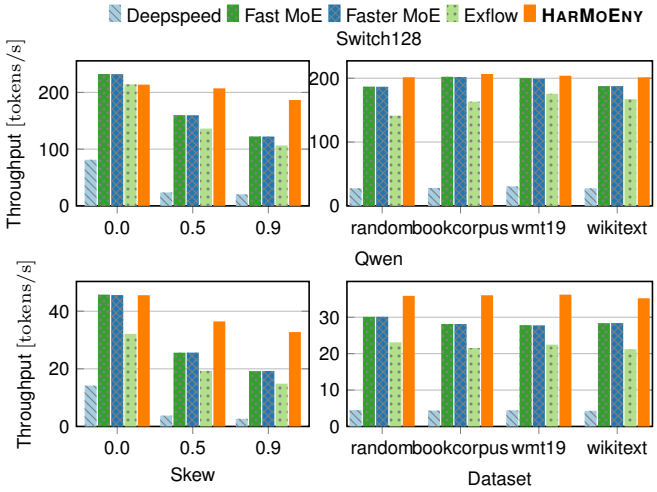


Fig. 6: Throughput (↑ is better) for different systems and different skews (left) and datasets (right) when using the SWITCH128 (top) and QWEN models (bottom).

Expert popularity skew. In addition to the skew naturally present in real datasets, we experiment with controlled popularity skews. The experiments with artificial skew aim to isolate and analyze the behavior of HARMOENY under controlled conditions. To introduce skew, we modify the router in each MoE block. We implement a configurable router skew mechanism according to the desired skew α, where $0 \leq \alpha \leq 1$, and the number of experts E. The selected skewed experts are assigned a probability proportional to α and the remaining experts share the remaining probability evenly, ensuring that the sum of the distribution equals 1. During routing, the router uses the multinomial distribution to sample tokens based on these probabilities, ensuring that the token distribution aligns with the desired skew. This allows us to have fine-grained control over the load imbalance and study the performance of HARMOENY in more detail.

B. Comparison to state-of-the-art systems

Skewed datasets. HARMOENY performs especially well when workloads exhibit high expert popularity skew. Figure 6 (left) and Figure 7 (left) show the throughput and mean TTFT for the CONSTANT dataset with different levels of router skew. In the scenario with 90% skew (*i.e.*, 90% of the tokens are routed to one popular expert), HARMOENY outperforms FASTMOE and FASTERMOE by 1.5x and 1.7x for both the SWITCH128 and QWEN models, respectively. HARMOENY beats DEEPSPEED by 9.1x for the 90% skew and 8.8x for 50% skew scenario for SWITCH128. The trend remains the same for QWEN. Compared to the MoE inference solution EXFLOW, HARMOENY performs 1.7x and 1.5x better on the 90% and 50% skew, respectively, for SWITCH128. The performance boost is even higher for QWEN that has larger experts with HARMOENY, outperforming EXFLOW by 2.2x and 1.8x in the 90% and 50% skew scenarios, respectively. In the 50%

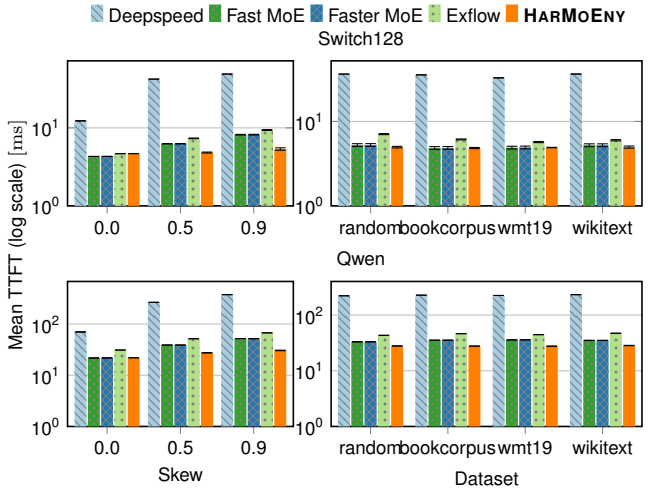


Fig. 7: Mean TTFT (↓ is better) for different systems and different skews (left) and datasets (right) when using the SWITCH128 (top) and QWEN (bottom) models.

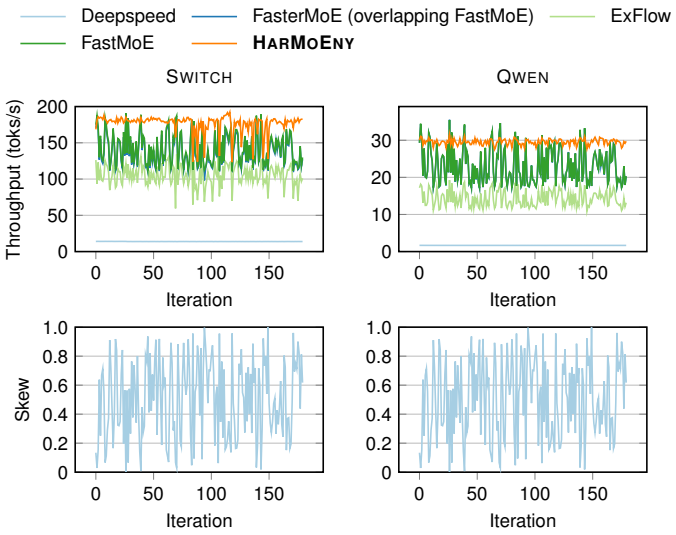


Fig. 8: The throughput of HARMOENY and baselines across iterations (top) and skew in expert popularity (bottom). HARMOENY maintains consistent throughput while skew varies from 0% to 95% per batch.

skewed workload, HARMOENY is 1.3x and 1.4x faster than FASTERMOE for SWITCH128 and QWEN, respectively.

Similar to the baselines, there is a deterioration in the throughput of HARMOENY as the skew increases. This happens because in extremely skewed cases, all the experts except for the popular one have very few tokens to process. Therefore, HARMOENY is not able to efficiently mask the expert fetching with computation leading to slightly lower throughputs. However, it is important to note that HARMOENY still outperforms the state-of-the-art baselines in the skewed scenarios.

Uniform datasets. As expected, there is little difference between most systems when all experts are equally popular (0% skew). FASTMOE and FASTERMOE are 8% faster than HARMOENY in the SWITCH128 workload with no skew due to HARMOENY's increased overhead when scheduling experts for each batch. Note that this is a synthetic scenario. Real-world datasets present a level of skew, as we discuss next.

Real-world datasets. Figure 6 (right) and Figure 7 (right) show the throughput and mean TTFT of HARMOENY, DEEPSPEED, FASTMOE, FASTERMOE, and EXFLOW running the SWITCH128 (top) and QWEN (bottom) models. The figures show the three real-world datasets and the RANDOM dataset described in Section V-A2. Notably, HARMOENY maintains steady high throughput (201 tokens/s and 36 tokens/s for SWITCH128 and QWEN, respectively) and low mean TTFT (5 ms and 27 ms for SWITCH128 and QWEN, respectively) for all real-world datasets. Compared to EXFLOW and FASTERMOE, this results in a speedup of 20% and 7% on the RANDOM and WIKITEXT datasets, respectively. This is because HARMOENY's lightweight load rebalancing mechanism and asynchronous pre-fetching keeps all GPUs operating at close to 100% (more details in Section V-C2). The throughput difference is accentuated for larger models. For QWEN (33 MB per expert), the gap widens compared to SWITCH128. HARMOENY is 15% to 28% faster than FASTMOE and FASTERMOE because the extent of the expert shadowing is constrained by the GPU memory. EXFLOW, while having consistent performance, falls behind HARMOENY for both models because it does not run the expert placement optimization often enough to keep up with expert popularity fluctuations.

Fluctuating expert popularity over time. Figure 8 (top) shows the throughput of HARMOENY and baseline systems where each batch has a different randomly chosen expert popularity skew (between 0% to 95%). HARMOENY maintains high and steady throughput while the other baselines have drops in throughput by up to 51%. Figure 8 (bottom) shows the associated skew for every batch of input request. As expected, we can see that the expert pre-fetching mechanism running out of the critical path maintains stable high throughput. HARMOENY not only achieves the highest overall throughput but also demonstrates significantly lower variance across batches of the same size but differing skew levels. Specifically, HARMOENY shows a variance of 152 $toks^2/s^2$ compared to EXFLOW's 206 $toks^2/s^2$, FASTERMOE's 447 $toks^2/s^2$, and FASTMOE's 477 $toks^2/s^2$. In summary, when the expert popularity changes across batches, HARMOENY maintains a high throughput by adapting to the imbalance.

C. Ablation study of HARMOENY

1) Time breakdown of HARMOENY components: We now conduct a time breakdown of the different operations when serving an MoE model with HARMOENY. We adopt the CONSTANT workload and assign 90% of all tokens to the first 10 experts ($\alpha = 0.9$). We use NVIDIA CUDA Events to obtain a fine-grained time breakdown of the operations in the first MoE layer. Figure 9 shows the duration of different operations in HARMOENY without any rebalancing (top), with rebalancing but without asynchronous expert loading (middle),

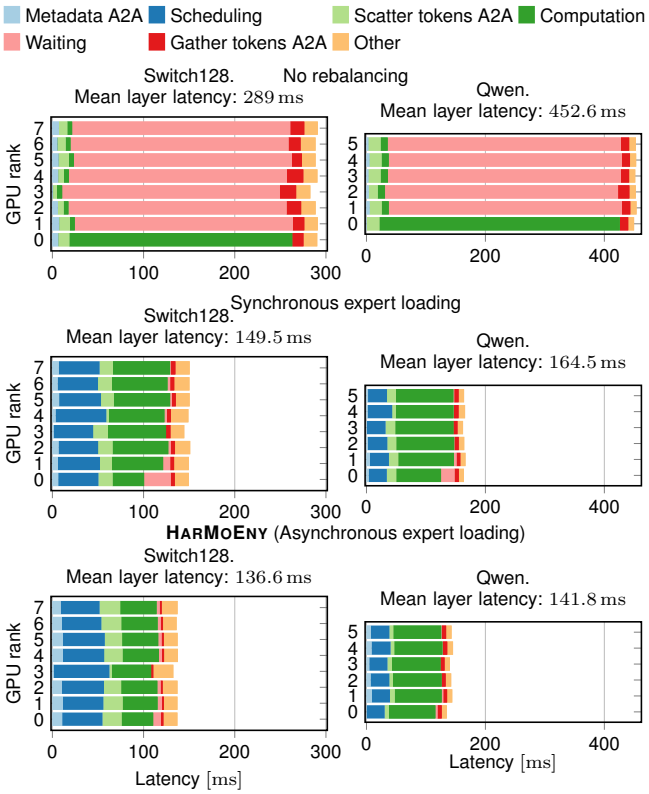


Fig. 9: Time breakdown of HarMoEny and baselines for the Switch128 (left) and Qwen (right) models.

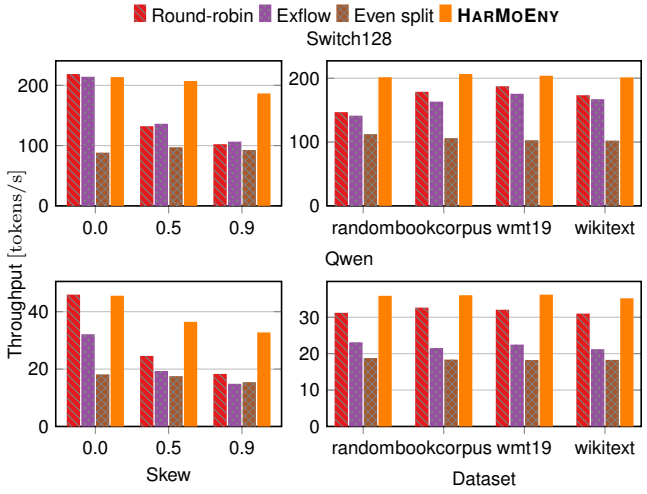


Fig. 10: Throughput (↑ is better) for different token load policies and different skews (left) and datasets (right) when using the Switch128 (top) and Qwen models (bottom).

and our original HarMoEny with all its components (bottom). Since the first 10 experts are loaded on GPU 0, we observe significant GPU waiting time for all other GPUs when we do not rebalance the token load using Algorithm 2, which is in line with the discussion in Section III. Specifically for Qwen, GPUs 1–7 spend on average 85.7% of the time waiting for GPU 0 to finish processing experts.

Our token rebalancing algorithm reduces the waiting time on all GPUs. The mean waiting time goes from 82.6% and 85.7% of the total GPU time down to a mere 2.6% and 1% for Switch128 and Qwen, respectively, across all GPUs, as seen in Figure 9 (middle). For the Switch128 model, token rebalancing reduces the mean layer latency from 289 ms to 149.5 ms, a total reduction of 48.3%. This reduction is even more pronounced with the Qwen model: 63.7% compared to when not rebalancing tokens. On average, our scheduling algorithm takes, 30.8% and 20.3% of the mean latency for the Switch128 and Qwen model, respectively. While scheduling and rebalancing in HarMoEny takes time, it brings a significant decrease in total inference latency.

Figure 9 (bottom) shows the timeline of operations of HarMoEny with all components enabled. Asynchronous expert loading further reduces the latency to 136.6 ms (-8.63% over synchronous loading) for Switch128 and to 141.8 ms (-13.8% over synchronous loading) for Qwen. Thus, we conclude that the combination of token rescheduling and asynchronous fetching in HarMoEny effectively minimizes the idling of GPUs and reduces the latency of MoE inference.

2) Load balancing policies: We next experiment with HarMoEny and when using different token rebalancing policies and measure the throughput in tokens/s. We implement the token rebalancing policies on top of HarMoEny to account for differences in system baselines implementations.

We evaluate HarMoEny against three other policies for token routing: (1) With the Round-robin policy, tokens are sent to the GPU housing the specified expert, regardless of imbalance, with experts distributed to GPUs in a round-robin manner. This policy is employed by DeepSpeed, FastMoE, and FasterMoE; (2) The ExFlow policy utilizes an integer programming-based approach to optimize token scheduling and routing by modeling the inter-layer affinity between tokens and experts, formulating a placement optimization problem to minimize the communication cost of routing tokens between GPUs; (3) Even Split evenly distributes the tokens for each expert to each GPU and replicates all experts on all GPUs. For example, if there are a tokens for expert 0 and four GPUs then $\frac{a}{4}$ tokens will be sent to each of the four GPUs. This achieves a perfect load balance across all experts at the cost of replication of all experts across all the GPUs.

Skewed datasets. Figure 10 (left) and Figure 11 (left) show the throughput and mean TTFT under different token load policies and skew levels α, for the two MoE models. Here, we skew the load of a single expert. HarMoEny, when using Switch128 with $\alpha = 0$ (no skew), reaches a throughput of 213 tokens/s which is comparable to that of the Round-robin and ExFlow policies. However, as α increases and consequentially, the load imbalance, HarMoEny reaches a significantly higher throughput than the other policies. For $\alpha = 0.9$, HarMoEny reaches a throughput of 186 tokens/s compared to 106 tokens/s for ExFlow, the best-performing baseline. We observe similar trends when using Qwen (Fig-

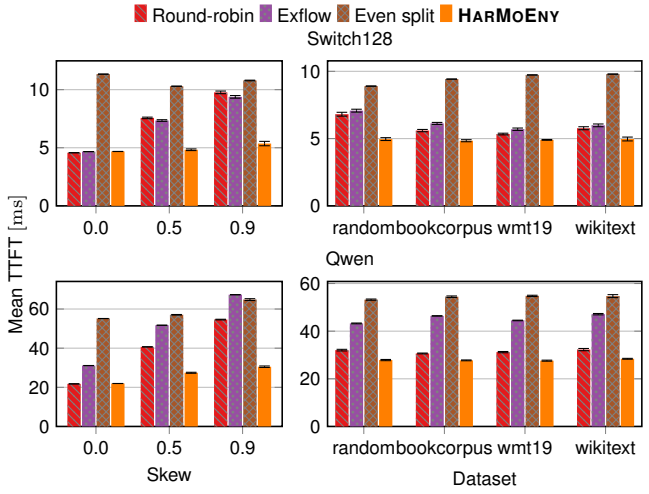


Fig. 11: Mean TTFT (↓ is better) for different token load policies and different skews (left) and datasets (right) when using the SWITCH128 (top) and QWEN models (bottom).

ure 10, bottom left). Even though the even split policy achieves perfect load balance, its performance is relatively low since each GPU has to load and execute every expert, thus increasing the time taken to process a request batch.

Figure 11 (left) shows that for $\alpha = 0$ and with the SWITCH128 model, HARMOENY has a mean TTFT of $4.68\,\text{ms}$, which is comparable to that of the Round-robin and EXFLOW policies. However, when α increases, so does the mean TTFT of other policies. With $\alpha = 0.9$ and with the SWITCH128 model, HARMOENY has a mean TTFT of $5.36\,\text{ms}$, compared to $9.77\,\text{ms}$ and $9.38\,\text{ms}$ for the Round-robin and EXFLOW policies, respectively. The mean TTFT of HARMOENY is also competitive when using the QWEN model. Thus, HARMOENY exhibits excellent mean TTFT, even under heavy token imbalances.

Real-world datasets. Figure 10 (right) and Figure 11 (right) show the throughput and mean TTFT of HARMOENY and other policies for real-world datasets. For all datasets and models used, HARMOENY reaches the highest throughput. This is the most pronounced when using the RANDOM dataset and SWITCH128 model. Figure 11 (right) shows that HARMOENY exhibits the lowest mean TTFT, thus justifying the token distribution policy in HARMOENY.

VI. RELATED WORK

Efficient MoE inference. Recent works improve MoE inference efficiency by optimizing communication, token imbalance, and using specialized GPU kernels. DeepSpeed-MoE Inference [20] provides flexible parallelization schemes, highly optimized MoE-related kernels, and an efficient communication subsystem. Tutel extends this by introducing adaptive parallelism at runtime [37]. However, both systems rely on static expert placement, limiting their ability to handle severely skewed workloads. ExFlow reduces the communication overhead by exploiting inter-layer expert affinity [14], though it assumes stable expert routing patterns that may not adapt to fluctuating inputs. Lina [17] addresses skewed workloads through a 2-phase scheme by profiling experts and predicting expert selection. Though effective in scenarios where inference requests are from similar domains, Lina needs to reallocate resources when the expert popularity changes. FloE optimizes MoE inference on memory-constrained devices by using model compression at the cost of inference utility [38]. In contrast, HARMOENY directly targets load imbalance in skewed workloads through token redistribution and dynamic expert placement without sacrificing utility. In addition, frameworks like DeepSpeed-MII [39] and vLLM [40] are under active development and utilize highly optimized CUDA kernels targeted to specific GPU architectures. Our approach is orthogonal. HARMOENY works at the application layer and can be further optimized with specialized GPU kernels, as in DeepSpeed-MII or vLLM. We also remark that HARMOENY is complementary to prediction-based approaches and can be combined by using prediction-based placement as an initial configuration and HARMOENY for further runtime adaptation.

Efficient MoE training. FastMoE [16] introduces an MoE training system with hierarchical interfaces and optimized CUDA kernels, enabling scalability but relying on static expert placement. FasterMoE [15] builds on this by addressing load imbalance through dynamic shadowing and fine-grained scheduling, while introducing congestion-avoiding expert selection during training. Since the experts are sparsely activated, Megablocks [41] achieves hardware efficiency by combining expert computations into block-sparse operations. Similar to DeepSpeed-MoE, SmartMoE [42] supports dynamic and hybrid parallelization strategies for MoE training. Several systems exploit workload patterns: Prophet [43] leverages token distribution similarity across training iterations, NetMoE [44] exploits routing locality within batches, and PopFetcher [45] prefetches popular experts based on training data statistics. Contrary to these MoE training systems, HARMOENY specifically targets dynamic load imbalance during inference.

VII. CONCLUSION

We presented HARMOENY, a novel system that addresses load imbalance in multi-GPU inference of MoE models. Through the combination of dynamic token rebalancing and asynchronous expert fetching, HARMOENY achieves near-perfect load balancing, significantly reducing inference latency. HARMOENY increases throughput by up to 70.1% and reduces time-to-first-token by up to 41.1%, compared to the next-best competitor, while maintaining stable throughput.

Our work focusses on a single-node setup with multiple GPUs attached to it. In multi-node settings, if all experts fit within one node, one can simply replicate HARMOENY independently across the nodes without any inter-node rebalancing. If experts must be distributed across nodes, *i.e.*, they do not fit within a single node, the all-to-all communication cost increases for HARMOENY. In such cases, the benefit of mitigating severe expert popularity skew outweighs the additional metadata exchange overhead introduced by HARMOENY.

REFERENCES

[1] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv:2001.08361*, 2020. [Online]. Available: https://arxiv.org/abs/2001.08361

[2] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, S. Zhong, B. Yin, and X. Hu, "Harnessing the power of LLMs in practice: A survey on chatgpt and beyond," *ACM Transactions on Knowledge Discovery from Data*, vol. 18, no. 6, 2024. [Online]. Available: https://doi.org/10.1145/3649506

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[4] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "GPT-4 technical report," *arXiv:2303.08774*, 2023. [Online]. Available: https://arxiv.org/abs/2303.08774

[5] R. Bianchini, C. Belady, and A. Sivasubramaniam, "Datacenter power and energy management: past, present, and future," *IEEE Micro*, 2024. [Online]. Available: https://doi.org/10.1109/MM.2024.3426478

[6] G. Leopold, "AWS to offer nvidia's t4 GPUs for AI inferencing," 2019, accessed: January 2025. [Online]. Available: https://www.hpcwire.com/2019/03/19/aws-upgrades-its-gpu-backed-ai-inference-platform/

[7] J. Barr, "Amazon EC2 update – inf1 instances with AWS inferentia chips for high performance cost-effective inferencing," 2019, accessed: January 2025. [Online]. Available: https://aws.amazon.com/blogs/aws/amazon-ec2-update-inf1-instances-with-aws-inferentia-chips-for-high-performance-cost-effective-inferencing

[8] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, 1991. [Online]. Available: https://doi.org/10.1162/neco.1991.3.1.79

[9] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," in *ICLR*, 2021. [Online]. Available: https://iclr.cc/virtual/2021/poster/3196

[10] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-0998.html

[11] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mixtral of experts," *arXiv:2401.04088*, 2024. [Online]. Available: https://arxiv.org/abs/2401.04088

[12] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei *et al.*, "Qwen2. 5 technical report," *arXiv:2412.15115*, 2024. [Online]. Available: https://arxiv.org/abs/2412.15115

[13] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "Deepseek-v3 technical report," *arXiv:2412.19437*, 2024. [Online]. Available: https://arxiv.org/abs/2412.19437

[14] J. Yao, Q. Anthony, A. Shafi, H. Subramoni, and D. K. D. Panda, "Exploiting inter-layer expert affinity for accelerating mixture-of-experts model inference," in *IEEE IPDPS*, 2024. [Online]. Available: https://doi.org/10.1109/IPDPS57955.2024.00086

[15] J. He, J. Zhai, T. Antunes, H. Wang, F. Luo, S. Shi, and Q. Li, "Fastermoe: Modeling and optimizing training of large-scale dynamic pre-trained models," in *PPoPP*, 2022. [Online]. Available: https://doi.org/10.1145/3503221.3508418

[16] J. He, J. Qiu, A. Zeng, Z. Yang, J. Zhai, and J. Tang, "Fastmoe: A fast mixture-of-expert training system," *arXiv:2103.13262*, 2021. [Online]. Available: https://arxiv.org/abs/2103.13262

[17] J. Li, Y. Jiang, Y. Zhu, C. Wang, and H. Xu, "Accelerating distributed MoE training and inference with lina," in *USENIX ATC*, 2023. [Online]. Available: https://www.usenix.org/conference/atc23/presentation/li-jiamin

[18] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," *ACM Sigplan Notices*, vol. 49, no. 4, 2014. [Online]. Available: https://doi.org/10.1145/2541940.2541941

[19] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, "Borg: the next generation," in *EuroSys*, 2020. [Online]. Available: https://doi.org/10.1145/3342195.3387517

[20] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation ai scale," in *ICML*, 2022. [Online]. Available: https://proceedings.mlr.press/v162/rajbhandari22a/rajbhandari22a.pdf

[21] A. Vaswani *et al.*, "Attention is all you need," *NeurIPS*, 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[22] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *NeurIPS*, 2000. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf

[23] T. Kudo, "Subword regularization: Improving neural network translation models with multiple subword candidates," in *ACL*, 2018. [Online]. Available: https://aclanthology.org/P18-1007/

[24] R. Gupta, L. Besacier, M. Dymetman, and M. Gallé, "Character-based NMT with transformer," *arXiv:1911.04997*, 2019. [Online]. Available: https://arxiv.org/abs/1911.04997

[25] M. Xu, D. Cai, W. Yin, S. Wang, X. Jin, and X. Liu, "Resource-efficient algorithms and systems of foundation models: A survey," *ACM Computing Surveys*, 2024. [Online]. Available: https://doi.org/10.1145/3706418

[26] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *ICLR*, 2017. [Online]. Available: https://openreview.net/forum?id=B1ckMDqlg

[27] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. a. Ranzato, A. Senior, P. Tucker, K. Yang, Q. Le, and A. Ng, "Large scale distributed deep networks," in *NeurIPS*, vol. 25, 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf

[28] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Technical Report, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[29] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2020. [Online]. Available: https://arxiv.org/abs/1909.08053

[30] P. Delgado, F. Dinu, A.-M. Kermarrec, and W. Zwaenepoel, "Hawk: Hybrid datacenter scheduling," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, 2015, pp. 499–510.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[32] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, 2020. [Online]. Available: https://jmlr.org/papers/volume21/20-074/20-074.pdf

[33] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024.

[34] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *arXiv:1506.06724*, 2015. [Online]. Available: https://arxiv.org/abs/1506.06724

[35] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016. [Online]. Available: https://arxiv.org/abs/1609.07843

[36] W. Foundation. Acl 2019 fourth conference on machine translation (wmt19), shared task: Machine translation of news. [Online]. Available: http://www.statmt.org/wmt19/translation-task.html

[37] C. Hwang, W. Cui, Y. Xiong, Z. Yang, Z. Liu, H. Hu, Z. Wang, R. Salas, J. Jose, P. Ram *et al.*, "Tutel: Adaptive mixture-of-experts at scale," *MLSys*, 2023. [Online]. Available: https:

//proceedings.mlsys.org/paper_files/paper/2023/hash/5616d34cf8ff73942cfd5aa922842556-Abstract-mlsys2023.html

[38] Y. Zhou, zheng li, J. Zhang, J. WANG, Y. Wang, Z. Xie, K. Chen, and L. Shou, "Floe: On-the-fly moe inference on memory-constrained GPU," in *Forty-second International Conference on Machine Learning*, 2025. [Online]. Available: https://openreview.net/forum?id=i5aHAkkhJH

[39] Microsoft, "Deepspeed-mii: Mii makes low-latency and high-throughput inference possible, powered by deepspeed," https://github.com/microsoft/DeepSpeed-MII, 2022, accessed: 2025-01-13.

[40] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with PagedAttention," in *SOSP*, 2023. [Online]. Available: https://doi.org/10.1145/3600006.3613165

[41] T. Gale, D. Narayanan, C. Young, and M. Zaharia, "Megablocks: Efficient sparse training with mixture-of-experts," in *MLSys*, 2023. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2023/hash/5a54f79333768effe7e8927bcccffe40-Abstract-mlsys2023.html

[42] M. Zhai, J. He, Z. Ma, Z. Zong, R. Zhang, and J. Zhai, "SmartMoE: Efficiently training Sparsely-Activated models through combining offline and online parallelization," in *USENIX ATC*, 2023. [Online]. Available: https://www.usenix.org/conference/atc23/presentation/zhai

[43] W. Wang, Z. Lai, S. Li, W. Liu, K. Ge, Y. Liu, A. Shen, and D. Li, "Prophet: Fine-grained load balancing for parallel training of large-scale moe models," in *IEEE International Conference on Cluster Computing (CLUSTER)*, 2023. [Online]. Available: https://doi.org/10.1109/CLUSTER52292.2023.00015

[44] X. Liu, Y. Wang, F. Fu, X. Miao, S. Zhu, X. Nie, and B. CUI, "Netmoe: Accelerating moe training through dynamic sample placement," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=1qP3lsatCR

[45] J. Zhang, C. Ma, X. Wang, Y. Nie, Y. Li, Y. Xu, X. Liao, B. Li, and H. Jin, "{PopFetcher}: Towards accelerated {Mixture-of-Experts} training via popularity based {Expert-Wise} prefetch," in *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, 2025, pp. 1053–1069.

Appendix: Artifact Description/Artifact Evaluation

TABLE I: Summary of artifacts.

Artifact ID	Related Paper Elements
A_1	Figures 6, 7, 10, and 11
A_2	N/A
A_3	Figure 8
A_4	Figure 9
A_5	Figures 1, and 2

Artifact Description (AD)

VIII. Overview of Contributions and Artifacts

A. Paper's Main Contributions

C_1 We empirically study the compute utilization of a GPU cluster running MoE inference.
C_2 We design and implement HarMoEny.
C_3 We evaluate HarMoEny with real datasets and synethetic benchmarks.

B. Computational Artifacts

$A_1 - A_5$ 10.5281/zenodo.18700393

Table I provides a mapping of the artifacts to the figures in the paper.

IX. Artifact Identification

A. Building Artifacts

a) Hardware: At least 2 NVIDIA GPUs with NVLINK interconnect, and 128 GB of DRAM are required.

b) Software: Artifacts A_1–A_5 are provided as code with a Dockerfile for reproducibility.

c) Datasets: The datasets BOOKCORPUS, WMT19, and WIKITEXT are publicly available on HuggingFace and downloaded automatically at runtime.

d) Installation and Deployment: Build the container with `./build_image.sh` and instantiate it with `./start_image.sh` from the repository root.

e) Setup Time: Building the Docker image takes approximately one hour.

B. Computational Artifact A_1

Relation To Contributions

A_1 gives the throughput and the mean TTFT of every system and token loading policy across the two models. This artifact empirically shows that HarMoEny outperforms other baselines on real and synthetic benchmarks.

Expected Results

HarMoEny should achieve a higher throughput and a lower mean TTFT against the baselines. When the dataset features very little skew, HarMoEny should be comparable to the baseline systems. Furthermore, HarMoEny should achieve better performance against all other token loading policies.

Expected Reproduction Time (in Minutes)

Running all experiments within this artifact will take about 32 hours.

Artifact Execution

This workflow comprises two tasks. Task T_1 executes the experiments and collects raw data. The corresponding scripts are located in the `experiments` directory. Before execution, ensure the Docker container is running and modify the configuration variables at the top of each script to match your setup.

Task T_2 processes the collected data into CSV files for plotting:

```
python create_csv.py \
  --paths <path_to_run/*> \
  --metric <throughput | mttft> \
  --variables <dataset | router_skew> \
              <scheduling_policy | system_name>
```

Use `mttft` for experiments with a `_ttft` suffix; otherwise, use `throughput`. Note that the non-TTFT system experiments reuse the ExFlow and HarMoEny runs from their corresponding policy experiments (e.g., `run_qwen_systems_datasets` incorporates the ExFlow and HarMoEny results from `run_qwen_policies_datasets`). Processed outputs are saved to the `data_processed` directory.

C. Computational Artifact A_2

Relation To Contributions

A_2 produces the ExFlow-optimized expert-to-GPU placement needed to evaluate ExFlow as a baseline against HarMoEny in A_1.

Expected Results

Given the same model, dataset, and GPU count, the linear optimizer should produce identical placements.

Expected Reproduction Time (in Minutes)

Approximately 2 hours total (1 hour per model). Per model, the four tasks take roughly 3 min, 10 sec, 45 min, and 5 sec, respectively. Increasing the trace length significantly increases the optimizer runtime.

Artifact Execution

A Gurobi license (`gurobi.lic`) is required in the `licenses` directory. The artifact output is generated through four sequential tasks ($T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$) in the `ExFlow` directory. We used 680 samples from BOOKCORPUS with `scheduling_policy=deepspeed` to capture the natural expert balance; a single GPU suffices.

```
# T1: Trace Collection -> raw_trace/
./get_trace_switch  # or ./get_trace_qwen

# T2: Trace Processing -> trace/
./process_raw_trace.sh \
<path_to_trace> <num_moe_layers>

# T3: Optimization -> solutions/
python solve_affinity.py \
  <path_to_npy> <num_experts> <num_gpus>

# T4: Format Conversion
python convert_exflow_into_harmony.py \
  <path_to_csv> <num_experts> <num_gpus>
```

Pass the resulting JSON via `--expert_placement` with `--scheduling_policy=exflow` to run EXFLOW.

D. Computational Artifact A_3

Relation To Contributions

A_3 reports per-iteration throughput rather than averages, showing that HARMOENY maintains consistent throughput with low variance across batches compared to other systems.

Expected Results

HARMOENY should display stable throughput with little per-batch variation, while baseline systems exhibit greater fluctuation.

Expected Reproduction Time (in Minutes)

Approximately 350 minutes for data collection and processing.

Artifact Execution

This workflow comprises two tasks ($T_1 \rightarrow T_2$). T_1 collects data and T_2 processes it into CSVs saved to `data_processed`.

```
# T1: Data Collection (experiments/)
./run_qwen_policies_timeline
./run_qwen_systems_timeline
./run_switch128_policies_timeline
./run_switch128_systems_timeline

# T2: Data Processing
python create_csv.py \
  --paths <path_to_data> \
  --metric timeline \
  --variables <scheduling_policy | system_name>
```

E. Computational Artifact A_4

Relation To Contributions

A_4 provides a time breakdown of the forward pass under three configurations: no rebalancing, synchronous expert loading, and asynchronous expert loading. This demonstrates the latency benefit of HARMOENY's asynchronous rebalancing mechanism.

Expected Results

HARMOENY with asynchronous expert loading should exhibit lower latency compared to both synchronous loading and no rebalancing for both models.

Expected Reproduction Time (in Minutes)

Approximately 30 min for data collection and 5 min for processing. An artificially skewed dataset is generated at runtime.

Artifact Execution

```
./run_time_time_qwen
./run_time_time_switch
python create_csv.py \
  --paths <path_to_run> \
  --metric timebreakdown \
  --num_moe_layers <#> \
  --world_size <num_gpus>
```

F. Computational Artifact A_5

Relation To Contributions

A_5 reports expert activation frequencies and per-GPU token balance, illustrating the skew in expert popularity and the effectiveness of HARMOENY's rebalancing.

Expected Results

A high activation skew toward a small subset of experts is expected. HARMOENY should achieve the most balanced per-GPU token distribution among all evaluated systems.

Expected Reproduction Time (in Minutes)

Approximately 5 minutes for processing. This artifact reuses the data collected by A_1.

Artifact Execution

This workflow processes data already collected in A_1.

Expert Frequencies.

```
python create_csv.py \
  --paths <path_to_a1_datasets> \
  --num_moe_layers <#> \
  --world_size <num_gpus> \
  --metric expert-freq
```

GPU Token Balance.

```
python create_csv.py \
  --paths <path_to_a1_datasets> \
  --num_moe_layers <#> \
  --world_size <num_gpus> \
  --metric gpu-balance \
  --variables scheduling_policy dataset
```