

Pre-gated MoE: An Algorithm-System Co-Design for Fast and Scalable Mixture-of-Expert Inference

Ranggi Hwang^{*†}

KAIST
ranggi.hwang@kaist.ac.kr

Jianyu Wei^{*†}

USTC / Microsoft Research
noob@mail.ustc.edu.cn

Shijie Cao

Microsoft Research
shijiecao@microsoft.com

Changho Hwang

Microsoft Research
changhohwang@microsoft.com

Xiaohu Tang[†]

Microsoft Research
v-xiaohutang@microsoft.com

Ting Cao

Microsoft Research
ting.cao@microsoft.com

Mao Yang

Microsoft Research
maoyang@microsoft.com

Abstract—Large language models (LLMs) based on transformers have made significant strides in recent years, the success of which is driven by scaling up their model size. Despite their high algorithmic performance, the computational and memory requirements of LLMs present unprecedented challenges. To tackle the high compute requirements of LLMs, the Mixture-of-Experts (MoE) architecture was introduced which is able to scale its model size without proportionally scaling up its computational requirements. Unfortunately, MoE’s high memory demands and dynamic activation of sparse experts restrict its applicability to real-world problems. Previous solutions that offload MoE’s memory-hungry expert parameters to CPU memory fall short because the latency to migrate activated experts from CPU to GPU incurs high performance overhead. Our proposed Pre-gated MoE system effectively tackles the compute and memory challenges of conventional MoE architectures using our algorithm-system co-design. Pre-gated MoE employs our novel pre-gating function which alleviates the dynamic nature of sparse expert activation, allowing our proposed system to address the large memory footprint of MoEs while also achieving high performance. We demonstrate that Pre-gated MoE is able to improve performance, reduce GPU memory consumption, while also maintaining the same level of model quality. These features allow our Pre-gated MoE system to cost-effectively deploy large-scale LLMs using just a single GPU with high performance.

Index Terms—Mixture-of-expert, inference system, machine learning, large language model, memory offloading

I. INTRODUCTION

Machine learning (ML) applications based on large language models (LLMs) have taken the world by storm, widely being deployed in various consumer facing products [24], [26], [33]. The success of LLMs has been driven by scaling up the model capacity (i.e., the model size) and its training dataset, the largest trained model size increasing by around $1,000\times$ within the past 5 years, from a few hundred million parameters to approaching a trillion parameter scale [3], [5], [39]. With larger model size bringing higher model accuracy, it is likely

that future models will also increase in their model capacity. However, a critical challenge in sustainably growing model size is its increasingly demanding computation requirement.

To tackle the high compute requirements of LLMs, the Mixture-of-Experts (MoE) [37] model was suggested as an alternative to the previous *dense* LLMs [3], [5], [29], [39]. The power of MoE comes from its ability to scale up the model capacity by increasing the number of *expert* parameters within an MoE block. Despite the increase in model parameter size, however, MoE utilizes a *gate function* to only partially activate the experts in a *sparse* manner, allowing them to achieve sub-linear compute cost with respect to model capacity. In contrast, prior dense LLMs activate the *entire* model parameters for inference and cause its compute cost to scale quadratically to model size, incurring significant computation overhead. Despite its merits, a **critical challenge of MoE** is its large memory requirement and the dynamically activated sparse experts which cause high deployment cost, rendering MoE’s applicability in real-world problems to be limited.

- 1) **Large memory requirement of experts.** While sparsely activating model parameters (i.e., the experts) helps reduce compute cost to achieve the same model quality as their dense LLM counterparts, MoEs require significantly larger memory to accommodate the large number of experts. For instance, an MoE-based Google Switch-Transformer [8] can have up to $75\times$ more parameters than the FLOPs-equivalent dense T5 model [29]. In other words, MoEs have a much lower *memory-efficiency* vs. dense LLMs, bringing critical system-level challenges in deploying MoE-based LLMs. To accommodate MoE’s large model size under GPU’s limited memory size, multiple GPUs can be utilized for deploying MoE where *expert parallelism* is employed to distribute the expert parameters across the GPUs to store only a portion of the experts for inference [18], [19], [30].
- 2) **Dynamic and sparse activation of experts.** Although multi-GPU solutions can distribute expert parameters across the GPU memory, MoE only partially activates a subset of the experts in a sparse manner [37]. This

^{*} Co-first authors who contributed equally to this research.

[†] Work done during an internship at Microsoft Research.

This is the author preprint version of the work. The authoritative version will appear in the Proceedings of the 51st IEEE/ACM International Symposium on Computer Architecture (ISCA-51), 2024.

makes the number of experts actually utilized per each GPU to become either very small or non-existent (i.e., none of the experts in a GPU are activated, leaving GPU idle) [21]. Furthermore, the sparse expert activation is dynamically decided at runtime, making it difficult to anticipate how many experts will be activated in each GPU. As such, the effective computation conducted over each GPU becomes low, exhibiting low GPU compute utilization and aggravating the total cost of ownership (TCO) for deployment.

Prior work on deploying MoE seeks to address these dual challenges by *offloading* MoE’s memory hungry expert parameters into CPU memory or SSD [1], [14], [18], [38] (referred to as *MoE-offload* below). The benefit of MoE-offload is that it reduces the number of GPUs required for deploying MoE, which helps increase the GPU’s compute efficiency for inference. Offloading MoE parameters, however, is no silver bullet as it comes with a significant increase in inference latency, deteriorating quality of service (QoS) to end users. This is because CPU offloading can only resolve MoE’s large memory requirement without addressing the *data dependency* issue that arises with dynamic sparse activation, a unique characteristic of the MoE. In an MoE block, there exists a sequential dependency between (1) the “selection” of which experts should be activated (using MoE’s gate function), and (2) the “execution” of activated experts for inference. Because such data dependency is *dynamically* resolved by the input data, the two-stage process of (1) *expert selection* and (2) *expert execution* must be serialized back-to-back. Consequently, the latency to migrate the activated expert parameters from CPU to GPU cannot be hidden under MoE-offload and causes severe performance overheads, failing to address the aforementioned challenges of deploying MoE (Section III-B).

In this work, we propose *Pre-gated MoE*, an algorithm-system co-design that enables MoE inference to incur low GPU memory consumption while still achieving high performance, substantially reducing TCO. We briefly summarize the **key contribution and novelty** of our Pre-gated MoE below.

- **(Algorithm)** In conventional MoE architectures, the gate function in the N -th MoE block selects the experts to activate which will then be executed within the *same* N -th MoE block. In our proposed design, we modify the role of a gate function to *preemptively* select the experts to be activated for the *next* MoE block (hence its new name, the *pre-gate* function). More concretely, the pre-gate function in the N -th MoE block selects the experts to activate for the $(N+1)$ -th MoE block. The novelty of our pre-gate function lies in its ability to completely *eliminate* the sequential dependency between the expert selection and expert execution stage within any given MoE block (i.e., data dependency now exists across the N -th MoE block’s expert selection and the $(N+1)$ -th block’s expert execution), which our proposed system effectively utilizes for performance optimization as detailed below.
- **(System)** Similar to prior MoE-offload systems, our Pre-

gated MoE stores the memory capacity limited expert parameters in CPU memory and reduces the number of GPUs required for inference. Unlike MoE-offload, our Pre-gated MoE utilizes the pre-gate function to overlap the CPU→GPU expert migration latency with the expert execution stage, minimizing the expert migration’s impact on performance. Specifically, Pre-gated MoE utilizes the N -th pre-gate function to identify the set of experts to activate for the $(N+1)$ -th MoE block, in advance, effectively *prefetching* only the activated experts to the GPU in preparation for the $(N+1)$ -th block’s execution while concurrently going through the expert execution for the N -th MoE block.

We evaluate our Pre-gated MoE system using state-of-the-art MoE models, achieving comparable or even higher model accuracy compared to the original MoE model across a wide range of natural language processing (NLP) tasks (e.g., summarization, and question answering). At the same time, decoupling the expert selection vs. expert execution stage provides our Pre-gated MoE to significantly reduce end-to-end inference latency, only adding 23% performance overhead than the oracular, performance-optimal GPU-only solution that can store the entire MoE parameters in GPU memory. Pre-gated MoE also reduces peak GPU memory consumption by $4.2\times$ vs. GPU-only, allowing the deployment of larger LLMs within a single GPU. Overall, our Pre-gated MoE system presents a fast, scalable, and cost-effective solution for serving MoE-based LLMs.

II. BACKGROUND

A. Dense LLMs using Transformers

Transformer model architecture. Transformer models [42] have become the dominant approach in designing ML applications for natural language processing (NLP), due to their ability to capture long-range dependencies and complex patterns in data [6], [39]. There are two primary ways in which a transformer model is structured: an encoder-decoder architecture [29] and a decoder-only architecture [3]. An encoder-decoder architecture consists of an encoder module that processes the input data (sequence of input tokens) and a decoder module that generates the output (an output token per decoder). The decoder-only architecture on the other hand implicitly incorporates the encoding process within the transformer blocks, eliminating the need for a separate encoder module. Regardless of which architecture is employed, both encoder-decoder and decoder-only architectures employ the following key components of transformers: the self-attention layer, the position-wise feed-forward networks (FFN) layer, normalizations, and residual connections, as shown in Figure 1(a). Self-attention helps determine the inter-word relationships and their dependencies within a sequence, whereas the FFN layer applies non-linear transformations to capture complex patterns in the input data. Both self-attention and FFN account for a significant portion of computation as well as memory requirements of transformer models.

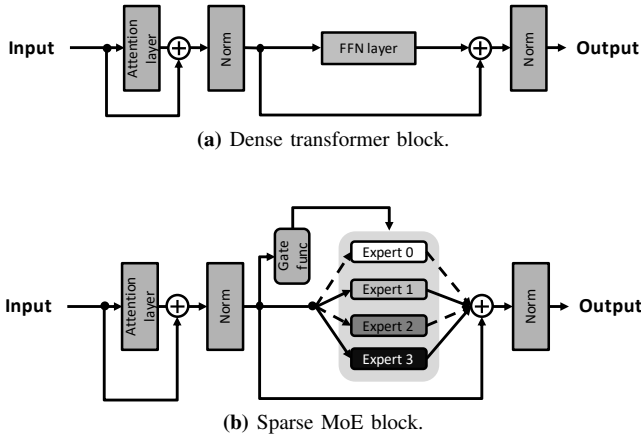


Fig. 1: (a) A dense transformer block that consists of the self-attention layer, feed-forward networks (FFN) layer, normalizations, and residual connections. (b) An MoE block that replaces a conventional transformer block’s FFN layer to induce sparsity. The example assumes the MoE block has four expert layers. Each expert has the same dimension as the FFN layer of the corresponding dense transformer block.

Challenges in scaling dense LLMs. The success of transformer based dense LLMs has primarily been driven by scaling up the model’s capacity (i.e., model size) by stacking a series of transformer blocks [17], [28], providing higher model accuracy. However, a key challenge in sustainably growing model capacity is its increasingly demanding compute and memory cost, for both training and inference. In particular, as the size of the LLM increases, the demands on compute and memory grow *quadratically*, making it challenging to fit the model within the memory constraints of modern GPUs while also maintaining high compute efficiency [40]. Furthermore, the energy costs associated with training these LLMs are increasing significantly, raising serious concerns on the environmental impact of training and serving LLMs [27], [44].

B. Sparse LLMs using Mixture-of-Experts (MoE)

MoE model architecture. To address the high computational requirements of dense LLMs, the Mixture-of-Experts (MoE) [7], [8], [11], [37], [41] model was introduced which exploits *sparsity* in the model architecture to reduce LLM’s high computation cost. MoE is designed to mimic the behavior of the human brain, which consists of specialized regions that are tailored for specific tasks. By sparsely activating only a subset of the parameters, MoE is able to scale up the model size without a corresponding increase in its computation cost (FLOPs).

Figure 1(b) illustrates the model architecture of an MoE block, which is converted from the dense transformer block in Figure 1(a) by replacing the original FFN layer in the transformer block with an MoE block. The MoE block consists of two key components: the gate function and the expert layer. The gate function is responsible for determining the relevance of each expert for a given input token, thereby assigning probabilities to each expert based on their importance to the

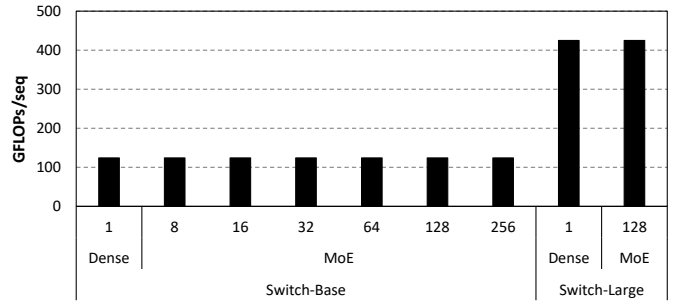


Fig. 2: Required number of FLOPs per sequence in deploying Switch-Transformer (MoE) and T5 (dense). In this figure, we show both the “Base” and “Large” model versions of SwitchTransformer and its FLOPs-equivalent T5 (Section V details the model configurations studied in this work). The numbers represent how many experts are available within the MoE block (i.e., dense T5 is equivalent to having just a *single* expert).

specific input. The expert layer, on the other hand, is a dense FFN layer that focuses on processing distinct patterns in the input data. During model inference, the gate function *selects* which experts should be activated for each input token based on their assigned probabilities. Subsequently, the activated experts process the input tokens by *executing* the assigned input tokens and generate the output tokens. As such, the evaluation of an MoE block involves a two-stage process, (1) *expert selection* and (2) *expert execution*, an input data-dependent procedure that must be executed sequentially.

In state-of-the-art MoE models, the number of experts that are activated is generally very small (e.g., Google’s SwitchTransformer [8] and Meta’s NLLB-MoE [41] only activates the top-1 and top-2 experts, respectively), rendering MoE’s inference to exhibit high sparsity.

Computation cost of sparse MoE vs. dense LLMs. MoE’s compute efficiency is achieved by selectively activating a small subset of the experts for each input token, instead of densely connecting all layers and neurons in the model. Figure 2 compares the required number of FLOPs per sequence between a representative sparse MoE model (Google’s SwitchTransformer [8]) and its dense model counterpart with an iso-FLOPs count (Google’s T5 [29]). As shown, the computation cost of MoE remains constant, regardless of the number of experts (i.e., the model size), highlighting the fact that MoE can scale up the model’s capacity with minimal computation overheads.

III. MOTIVATION

A. Key Challenges of MoE inference

While MoE offers advantages in scaling the LLM model size without significantly increasing its computation cost, it introduces several key challenges as summarized below.

- 1) **Large memory footprint.** The biggest advantage of MoE is its high compute efficiency, which comes from its ability to cost-effectively scale the model capacity by employing a large number of experts. This, however, comes at the cost of high memory consumption, leading MoE’s overall memory footprint to become an order

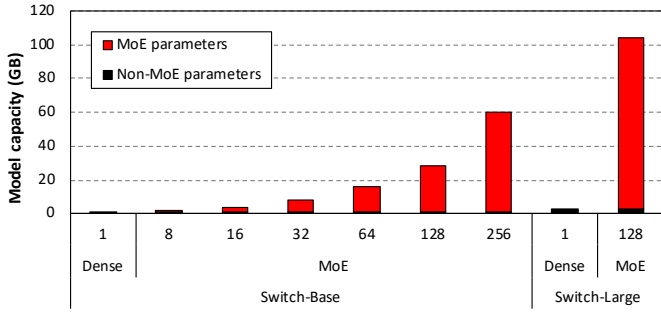


Fig. 3: Memory capacity requirement of deploying SwitchTransformer (MoE) and T5 (dense). MoE parameters include both the expert layer and the gate function, while the rest of the layers are marked as Non-MoE parameters. As depicted, MoE expert parameters account for the majority of the model’s memory consumption.

of magnitude larger than its dense counterpart, e.g., SwitchTransformer can consume as much as $75\times$ higher memory consumption than the dense T5 (Figure 3). Such large memory usage poses several obstacles for MoE, one critical challenge being its inability to fit the model within a single GPU’s local memory which is only several tens of GBs in size.

- 2) **Dynamic and sparse expert activation.** Because it is challenging to store the entire MoE model parameters within a single GPU, multi-GPU solutions that split the expert parameters across multiple GPU’s memory can be a viable solution for high-performance MoE inference. Unfortunately, because MoE only partially activates the experts in a sparse manner, the number of experts actually executed by each GPU for inference becomes very low. In effect, multi-GPU solutions for deploying MoE suffer from low GPU compute utilization and deteriorate the TCO. Furthermore, because MoE experts are sparsely activated, a significant fraction of expert parameters allocated inside the expensive GPU memory is most likely not going to be utilized when servicing any given inference request (e.g., SwitchTransformer activating top-1 among the 128 experts executes only 0.8% of its experts per inference). Finally, because the sparsely activated experts are determined dynamically in an input data dependent manner, it becomes challenging to predict which experts will be activated at runtime, preventing any load-balancing solutions to better distribute the number of activated experts across the GPUs in an even manner. Since GPU’s limited memory capacity was the very reason why a multi-GPU system was necessary for deploying MoE, such sub-optimal utilization of GPU memory is a significant waste.

B. Prior Solution: CPU Offloading of Expert Parameters

The challenge of efficiently managing LLM’s large model size within the constraints of limited GPU memory has led to several prior work advocating to offload the memory hungry LLM parameters to CPU DRAM or even SSD [31], [35]. These *CPU offload* based approaches have also been

explored under the context of MoE in order to address its aforementioned challenges, i.e., its large memory footprint and high deployment cost [1], [14], [15], [38]. Although CPU offload based solutions can help reduce the number of GPUs required for servicing MoE, the latency to transfer the CPU offloaded model parameters to the GPU memory can deteriorate end-to-end performance. This is because none of the prior work fundamentally addresses the dynamic and sparse expert activation challenge and its sequential dependency issue. Below we classify prior CPU offload based solutions into two categories, (1) *fetch-on-demand* and (2) *prefetch-all*, discussing its benefits as well as its limitations.

Fetch-on-demand. This design point [15] employs the fetch-on-demand based CPU offloading for MoE serving. Under this system design, *all* the expert parameters are offloaded to the capacity-optimized CPU memory. At runtime, once the expert selection stage identifies which experts are activated, those activated experts are migrated to the GPU memory *on-demand*. Because GPU memory is only used to store the activated experts (and not the entire experts as done in a baseline multi-GPU system), it helps improve GPU memory utilization significantly. However, the process of migrating activated experts on-demand serializes the expert selection stage with the expert execution stage, causing noticeable performance overhead. In the rest of this paper, we refer to this design point as *MoE-OnDemand*.

Prefetch-all. To better hide the CPU→GPU expert transfer latency, prior work on SE-MoE [38] proposes a *prefetching* based CPU offloading for MoE, where the expert parameters are proactively migrated to the GPU memory before its actual usage (henceforth referred to as *MoE-Prefetch*). Similar to MoE-OnDemand, MoE-Prefetch offloads all expert parameters in CPU memory. MoE-Prefetch then migrates the *entire* expert parameters to be used by the *next* MoE block while the *current* MoE block’s expert execution is taking place. While MoE-Prefetch can help overlap compute (current MoE block’s expert execution) with communication (transferring all experts required for the next MoE block), it suffers from several limitations. First, MoE-Prefetch is not scalable as CPU→GPU expert transfer time can be prohibitive when there exists a large number of experts (e.g., Google’s SwitchTransformer [8] contains up to 256 experts, while Meta’s NLLB-MoE [41] employs 128 experts). Second, at any given MoE block’s execution, the GPU memory must be large enough to store both the current as well as the next MoE block’s entire expert layer parameters, potentially overwhelming the scarce GPU memory. To mitigate the significant GPU memory demands required for transferring entire expert parameters, one could consider prefetching only a subset of experts predicted to be active in the next block. Nonetheless, predicting active experts does not always ensure accuracy, and mispredictions can lead to penalties such as waste of GPU memory and the unavoidable serialization of expert transfer latency, consequently increasing the end-to-end inference latency.

Aside from these prior works focusing on CPU offloading decisions for MoE inference, [14] characterizes MoE

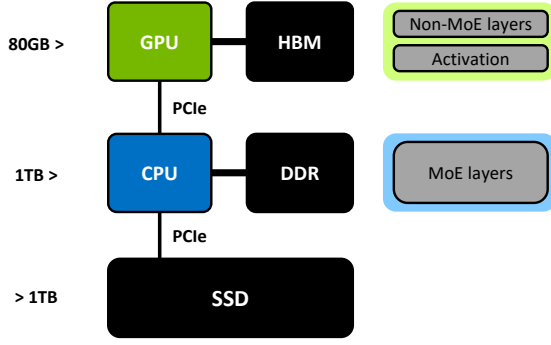


Fig. 4: Pre-gated MoE system for deploying MoE-based LLMs. The memory hungry, sparse MoE parameters are all offloaded to the capacity-optimized CPU memory and are only transferred to the GPU memory when necessary for inference. The rest of the dense, non-MoE parameters are stored locally within the GPU memory.

deployment’s inference latency and its memory usage across different components of the MoE model architecture, suggesting several optimization strategies like dynamic gating and expert buffering. Dynamic gating helps reduce wasted memory allocations for multi-GPU MoE inference and expert buffering is a technique that can help reduce inference latency by caching hot, active experts in GPU memory. In Section VI-D, we further discuss the applicability of expert caching on top of MoE-offload design points.

Overall, we conclude that the inherent sequential dependency between the expert selection and expert execution stage poses several challenges in designing a performance-efficient CPU offloading based MoE system. The key objective of this paper is to exploit the dynamic and sparse nature of MoE models to design a holistic system solution that effectively balances memory efficiency and high performance.

IV. PRE-GATED MOE: CO-DESIGNING ALGORITHM AND SYSTEM FOR FAST & SCALABLE MOE INFERENCE

A. High-level Overview

We propose Pre-gated MoE, an algorithm-system co-design for scalable and high-performance MoE inference. Pre-gated MoE is designed to address the large memory footprint challenge of MoE while also mitigating the dynamic nature of sparse expert activation for performance improvement. These features enable our Pre-gated MoE to deploy large-scale LLM using just a single GPU. Because activated experts are determined dynamically in an input dependent manner, all expert parameters must be preserved at all times, regardless of its actual utilization. To efficiently manage the storage of MoE’s substantial model capacity, Pre-gated MoE carefully considers the model parameter’s actual utility to decide their storage locations. As shown in Figure 4, the dense non-MoE parameters are stored locally within the GPU memory as they are always utilized, regardless of the input values. Meanwhile, the sparse MoE parameters are completely offloaded to the CPU’s DRAM because (1) they account for the majority of LLM’s model capacity so CPU offloading can help signif-

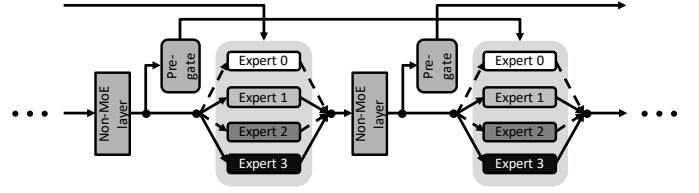


Fig. 5: Two consecutive MoE blocks employing our proposed pre-gate function. For brevity, we only provide a detailed illustration on the MoE blocks (the rest of the non-MoE layers are consolidated into a single block in this figure). The residual paths within a transformer block are not shown. As depicted, a pre-gate function is trained to select which experts to activate for the next MoE block.

icantly save GPU memory, and (2) only a small fraction of the MoE experts that are activated are actually utilized for inference. As we detail in this section, such hierarchical storage of MoE parameters and its deployment proves effective in minimizing the usage of GPU memory while still providing high performance.

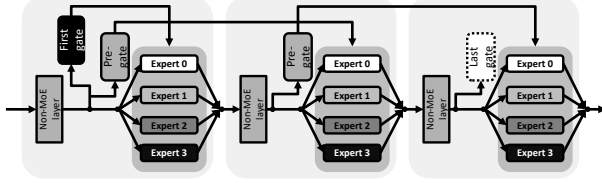
As discussed in Section III-B, prior work has followed two main approaches, MoE-OnDemand and MoE-Prefetch. Because traditional MoE blocks must sequentially execute expert selection followed by expert execution in an input data dependent manner, both MoE-OnDemand and MoE-Prefetch suffer from sub-optimal performance. In particular, MoE-OnDemand directly exposes the CPU→GPU communication latency to migrate activated experts as part of end-to-end inference time. This is because expert execution must always be preceded with the expert selection stage. MoE-Prefetch can hide the communication latency to transfer expert parameters to some extent, but it still suffers from performance loss because *all* expert parameters must be transferred to the GPU, even though only a small fraction of them will actually be utilized for inference.

The key objective of Pre-gated MoE is to mitigate the impact of the MoE block’s dynamically determined sparse expert activation and utilize that property for performance improvement. Specifically, Pre-gated MoE introduces a new gate function that *decouples* the expert selection stage from the expert execution stage. The benefit of decoupling expert selection with expert execution is twofold. First, it enables our system to significantly reduce the latency to migrate experts from CPU to GPU as only the activated experts will be migrated under our proposed design. Second, the performance overhead of migrating the activated experts can be effectively hidden by overlapping it with MoE block’s computation. In the remainder of this section, we detail the two key facets of our algorithm-system co-design.

B. (Algorithm) Pre-gated MoE Architecture

Pre-gate function. In traditional MoE model architectures, each MoE block contains a gate function which selects the experts to activate within the *same* MoE block. Because only those experts that are activated are subject to the subsequent expert execution stage, it is impossible to overlap the expert selection stage with the expert execution stage. In our proposed

Decoder iteration 0



Decoder iteration 1

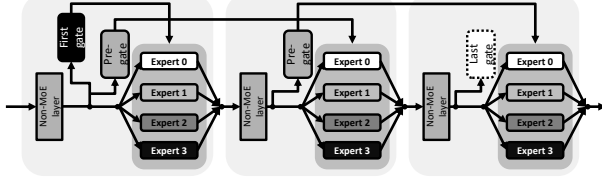


Fig. 6: The sequence of pre-gated MoE block’s execution during the course of two consecutive decoder iterations. We assume the LLM consists of three pre-gated MoE blocks, requiring three MoE block executions for a single iteration of decoding. The first MoE block employs two gate functions (one for the current MoE block and another for the next MoE block) whereas the last MoE block does not utilize any gate function.

MoE model architecture, we introduce the *pre-gate function* which is trained to preemptively select the experts to activate for the *next* MoE block rather than the current MoE block. More concretely, a pre-gate function for the N -th MoE block is trained to generate the activation masks to utilize in the $(N+1)$ -th MoE block to select which experts to activate (Figure 5). Prior work has explored alternative ways to train gate functions, which are fine-tuned for specific objective functions such as enhancing the model accuracy and alleviating the input token’s load-imbalance problem when distributed across multiple GPUs [8], [20], [36], [46], [47]. The approach taken with our Pre-gated MoE is aligned with these prior art but with one important distinction – our pre-gate function is designed to deterministically select and pre-compute which experts to activate for the subsequent MoE block. As we demonstrate in Section VI-C, our pre-gate function has a minimal impact on LLM’s model accuracy and is shown to be highly robust.

Since our pre-gate function is trained to select the active mask for the next MoE block, two important questions remain: (1) How does our Pre-gated MoE architecture select the experts to activate for the *first* MoE block (i.e., the first MoE block does not have a previous MoE block that will select the experts to activate on behalf of the first block)? (2) What is the role of the pre-gate function for the *last* MoE block (i.e., the last MoE block does not have a subsequent MoE block)? We answer these questions using Figure 6. In conventional MoE-based LLMs, a single decoder iteration generates a single output token (word) and multiple iterations of decoding are conducted during a single inference run to generate the final output result (which is a series of tokens). As shown in Figure 6, a single decoder iteration involves the execution of several stacks of MoE blocks. In our proposed MoE design, the first MoE block employs *two* gate functions, the first gate selecting the activated experts for the first MoE

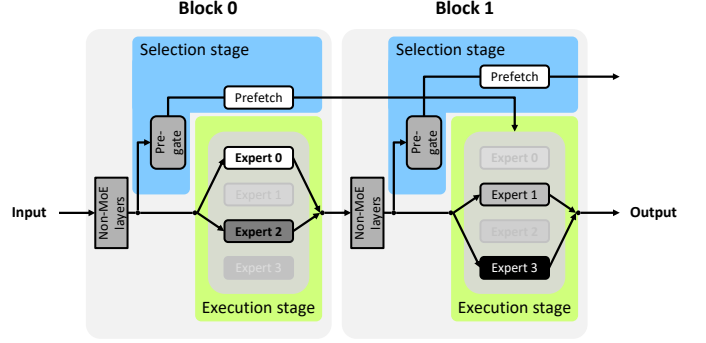


Fig. 7: Our pre-gate function helps eliminate the sequential dependency between an MoE block’s expert selection and expert execution stage. The gate function is implemented as a compact MLP layer having low computation requirement, so the preemptive migration of activated experts right after the gate function (blue) exhibits a PCIe communication-bound behavior. Overall, our Pre-gated MoE enables the compute-bound expert execution stage (green) to concurrently execute with the communication-bound expert selection stage (blue) for all MoE blocks (with the exception of the first MoE block). Example assumes that expert 0 and 2 are activated for the first MoE block while expert 1 and 3 are activated for the second MoE block.

block (identical to conventional MoE architectures) and the second gate (our *pre-gate* function) selecting the experts to activate for the second MoE block. Conversely, because the last MoE block does not have a subsequent MoE block to execute within the *same* decoder iteration, we do not employ a pre-gate function for the last MoE block. In effect, the pre-gate function does not select activated experts *across* different decoder iterations.

Training the pre-gate function. Today’s LLMs are first *pretrained* on vast amounts of textual data which spans a wide variety of languages and application domains. The pretraining stage requires a massive amount of computation power (several tens of thousands of GPUs) and typically takes several months to complete (e.g., GPT-3 is pretrained over hundreds of billions of tokens for more than one month using thousands of GPUs [3]). Once the LLM is pre-trained, it goes through the *fine-tuning* stage with task-specific datasets for specific use cases (e.g., summarization, question answering). Training our Pre-gated MoE does not change how the resource-intensive pretraining stage is conducted, as our pre-gate functions are incrementally trained during the fine-tuning stage. Specifically, we utilize existing pretrained MoE model parameters as-is but change the MoE model architecture to properly accommodate the functionalities of our pre-gate function as well as the augmentations required in the first/last MoE block (see Figure 6). We then go through the fine-tuning stage as required by the downstream task, identical to how conventional MoE models will be fine-tuned in a task specific manner. When our Pre-gated MoE is fine-tuned over the same number of fine-tuning training iterations vs. conventional MoE models, we observe no noticeable degradation in LLM model accuracy, one which we further elaborate in Section VI-C.

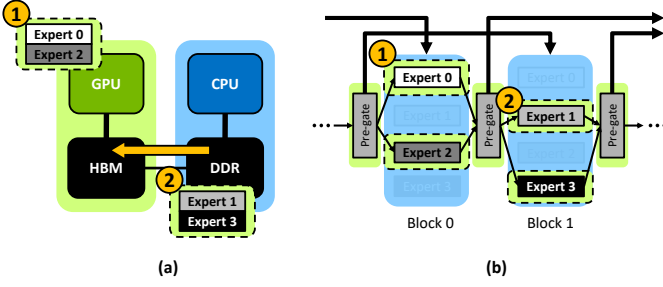


Fig. 8: (a) Illustration of how the example in Figure 7 gets handled over the Pre-gated MoE system where the expert execution in MoE block 0 (using the activated experts 0 and 2) concurrently takes places while the next MoE block 1’s activated experts 1 and 3 are migrated over to the GPU memory. (b) The on-demand migration of just the activated experts (green) allows the entire experts to be stored in CPU memory (blue), significantly saving GPU memory capacity.

C. (System) Preemptive Expert Migration

Our pre-gate function provides MoE models with the ability to determine what experts will be activated in the next MoE block while the current MoE block is being executed, presenting new opportunities for system-level performance optimizations. In particular, with the exception of the first MoE block, all MoE block’s expert execution stage is now completely decoupled from the expert selection stage without any data dependencies, allowing both stages to be concurrently executed (Figure 7)¹. Such feature opens up several opportunities as detailed below.

CPU offloading with minimal expert migration overhead. A key limitation of previous CPU offloading solutions is that the latency to migrate the CPU-offloaded expert parameters is either directly exposed as part of the end-to-end inference time (MoE-OnDemand) or the size of the migrated experts are simply too large that, despite its opportunity to overlap expert migration with the expert execution, copying the experts overwhelms the end-to-end performance (MoE-Prefetch). Our Pre-gated MoE, on the other hand, can evaluate which experts will be activated in advance, only migrating the activated experts for the next MoE block while the current MoE block’s experts are being executed. This effectively addresses the dual challenges of prior CPU offloading techniques, namely (a) MoE-OnDemand’s serialization of expert selection and expert execution (resolved by Pre-gated MoE’s concurrent expert migration and expert execution) and (b) MoE-Prefetch’s large expert migration latency (tackled by Pre-gated MoE’s ability to only migrate activated experts). State-of-the-art MoE models employ a large number of experts within an MoE block while only activating a very small subset of them (e.g., Google’s SwitchTransformer [8] contains up to 256 experts but only activates the top-1 expert, while Meta’s

¹The first MoE block is the only exception to this property under our Pre-gated MoE design – due to the lack of a pre-gate function in the first MoE block, we must sequentially execute its expert selection and expert execution stages, identical to conventional MoE models. Because state-of-the-art LLMs typically contain tens of MoE blocks, most of the MoE blocks are able to overlap expert selection with expert execution.

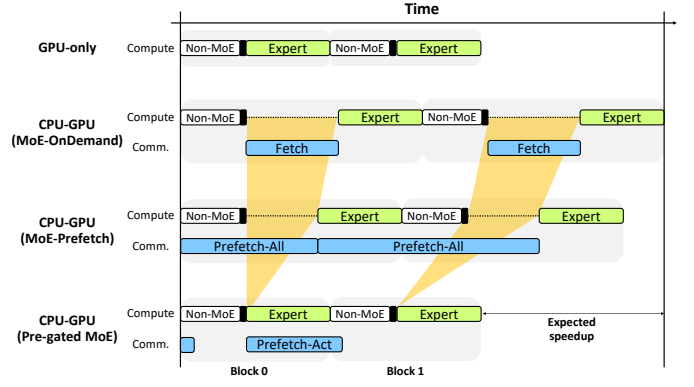


Fig. 9: Execution timeline between our Pre-gated MoE system and three baseline designs (GPU-only, MoE-OnDemand, and MoE-Prefetch). The black bar represents the latency to execute the gate functions. GPU-only is an ideal, oracular design point that has infinite GPU memory capacity, allowing the entire model parameters to be stored within GPU memory (i.e., there is no communication latency to migrate experts from CPU to GPU). In our Pre-gated MoE, the latency to migrate experts can be hidden by both the expert and non-MoE layer’s (e.g., self-attention layer) execution time.

NLLB-MoE [41] employs 128 experts and activates top-2 experts). As depicted in Figure 8, we can clearly see the benefit of how our algorithm-system codesign can effectively address the limitations of existing CPU-offloading solutions, maximizing the opportunity to overlap expert migration latency with expert execution time while also ensuring that the CPU→GPU data transfer size is minimized. Figure 9 points out the limitations of MoE-OnDemand and MoE-Prefetch and how our Pre-gated MoE successfully addresses its shortcomings, potentially reaching the performance of an ideal, *GPU-only* design point when the latency to migrate activated experts can be completely hidden inside the MoE expert execution stage.

$$\forall N, \quad 0 \leq N < \text{Number of MoE blocks}$$

$$\text{Peak_GPU_mem} = \max \left(\text{Non_MoE}_M + \sum_{L=N}^{N+1} \text{Act_Exp}_L \right) \quad (1)$$

Low GPU memory utilization for large LLM deployment. The majority of MoE-based LLM’s model capacity are concentrated around MoE parameters. Since our Pre-gated MoE system offloads the entire MoE parameters to CPU memory and only migrates activated experts over to the GPU, we are able to significantly reduce the *peak* usage of GPU memory. In our Pre-gated MoE system, peak GPU memory usage is primarily dominated by the memory capacity required to store (1) all the non-MoE parameters (which are statically stored in GPU memory) and (2) the active experts for both the current and the subsequent MoE block (dynamically determined at runtime and copied over to the GPU memory). Equation 1 summarizes the peak GPU memory usage to store MoE-based LLM’s model parameters under Pre-gated MoE.

In this equation, Non_MoE_M represents the total size of the non-MoE parameters while $\sum_{L=N}^{N+1} \text{Act_Exp}_L$ represents

TABLE I: Model configuration of Google’s SwitchTransformer.

Model	Experts	Layers	Parameters (B)	Capacity (GB)
Switch-Base	8	12	0.7	2.8
	64	12	3.8	15.2
	128	12	7.5	30.0
Switch-Large	128	24	26.4	105.6

the aggregate size of the active expert parameters over two consecutive (the N -th and $(N+1)$ -th) MoE blocks. Since expert parameters account for the majority of MoE-based LLM’s model size (see Figure 3) and only a small fraction of experts are activated during inference, the peak GPU memory usage in Equation 1 becomes much lower than GPU-only and can also reach the memory consumption level of the memory-optimal MoE-OnDemand design. A key advantage of reducing peak GPU memory usage is that it facilitates the deployment of considerably larger LLMs on systems with limited GPU memory resources (e.g., desktop and edge devices). In Section VI-B, we demonstrate Pre-gated MoE’s scalability and applicability for deploying large-scale LLMs.

V. METHODOLOGY

System configuration. We conducted our evaluation using two system design points, GPU-only and CPU-GPU, which utilize an AMD EPYC 7V12 64-Core CPU with 1.8TB DDR4 memory and a single NVIDIA GPU A100 with 80GB of HBM. The CPU and GPU communicate over a PCIe (gen4) channel with 32 GB/sec of data transfer bandwidth.

The oracular GPU-only design assumes the entire model parameters are stored in GPU memory, so the all computations for inference are conducted on the GPU. Note that multi-GPU solutions leveraging expert parallelism can experience performance loss due to inter-GPU communications and load imbalance issues. For a conservative evaluation, we experiment with our GPU-only system under a single GPU system that can achieve the highest performance. As such, GPU-only represents a performance-optimal, upper-bound MoE inference system that we compare our Pre-gated MoE against.

The CPU-GPU design, on the other hand, utilizes both GPU and CPU memory for storing the model parameters where only the (dense) non-MoE parameters are persistently stored within the GPU memory while the (sparse) MoE parameters are completely offloaded to CPU memory (Figure 4). Our Pre-gated MoE system as well as the two baseline CPU offloading MoE systems (MoE-OnDemand and MoE-Prefetch) employ such CPU-GPU system configuration.

Model and dataset. We use Google’s SwitchTransformer [8] as the baseline MoE for our evaluations, a state-of-the-art large-scale MoE model. The open-sourced pretrained weights available at HuggingFace [43] were utilized to fine-tune both Pre-gated MoE as well as the baseline MoE model for downstream tasks (Table I). As for the training data, we study three datasets covering two distinct downstream tasks: one from the summarization task (Xsum [23]) and two from the closed-book question answering task (CB Web QA [2], SQuAD [32]). The evaluation metrics included Rouge-1 and



Fig. 10: Average latency incurred in executing a single MoE block (normalized to GPU-only). Since GPU-only experiences an out-of-memory (OOM) error in Switch-Large, we normalized the latency of MoE-OnDemand and MoE-Prefetch to Pre-gated MoE. Note that the y-axis in this chart is plotted in log-scale.

Rouge-2 scores [22] for summarization, and ExactMatch and F1 scores for question answering.

Model training (fine-tuning). We applied the exact same fine-tuning configurations across all model architectures including Pre-gated MoE and conventional MoE. As discussed in Section IV-B, the fine-tuning stage utilizes the pre-trained weights from the conventional MoE model. We utilize a mini-batch containing 256 sequences, each with a length of 256 tokens, to fine-tune the model for 2,048 steps (i.e., 2^{27} tokens in aggregate). A constant learning rate of 0.0001 is employed.

Software implementation. All of our GPU-only and CPU-GPU systems are implemented using NVIDIA’s FasterTransformer [25], a state-of-the-art high-performance CUDA library widely employed in production inference servers in the industry. Because end-to-end inference performance is less sensitive to what the downstream task the MoE model is trained for, we report performance numbers using the MoE model fine-tuned for the closed-book question answering tasks with the SQuAD dataset. When reporting model accuracy, we use the two downstream tasks as discussed above.

VI. EVALUATION

In this section, we first demonstrate Pre-gated MoE’s effectiveness in improving performance (Section VI-A) and discuss its scalability to large-scale MoE models (Section VI-B). We then quantitatively evaluate Pre-gated MoE’s impact on model accuracy (Section VI-C) and finally present sensitivity studies as a discussion point (Section VI-D), demonstrating the robustness of Pre-gated MoE.

A. Performance

In this section, we primarily focus on single batch inference scenarios because real-world production ML serving systems are optimized for a batch size of 1 [9], [10], [34]. As discussed in Section IV-B, the end-to-end performance of CPU offloading solutions are primarily determined by how well the CPU→GPU communication time (to migrate experts) is hidden inside the MoE block’s execution time. Furthermore, the end-to-end MoE inference time is mostly dominated by a series of (identically sized) MoE block’s execution. As such, we first focus on comparing a single MoE block’s execution time between Pre-gated MoE vs. baseline systems. We then

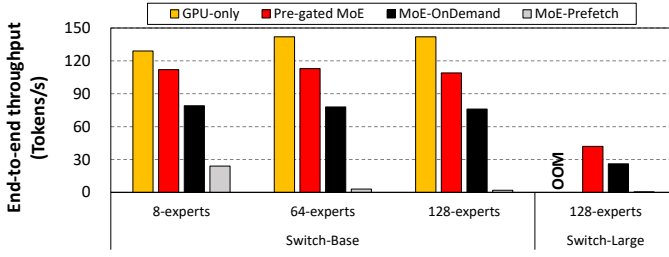


Fig. 11: End-to-end inference throughput. GPU-only experiences an out-of-memory (OOM) error in Switch-Large.

discuss the improvements in end-to-end inference throughput, measured as the number of tokens processed per second.

MoE block latency. Figure 10 summarizes the average latency in executing a single MoE block. Across all configurations, Pre-gated MoE significantly reduces latency by an average $1.7\times$ (max $1.9\times$) and $42\times$ (max $125\times$) vs. MoE-OnDemand and MoE-Prefetch, respectively. Pre-gated MoE also exhibits comparable latency to the performance-optimal GPU-only, incurring only 19% latency overhead across all Switch-Base model configurations. Because MoE-Prefetch must migrate all experts, it suffers from the highest latency where the larger number of experts directly translates into higher performance overheads. MoE-OnDemand does better than MoE-Prefetch, thanks to its ability to only migrate activated experts. However, MoE-OnDemand still suffers from longer latency than Pre-gated MoE due to the serialization of expert selection and expert execution stages.

It is worth pointing out that the performance-optimal GPU-only is unable to run the largest MoE model, i.e., Switch-Large with 128 experts (105.6 GB), due to the limitations in GPU memory capacity, resulting in an out-of-memory (OOM) error. Pre-gated MoE still shows the shortest latency among the three CPU-GPU based designs with Switch-Large, achieving $1.9\times$ and $125\times$ latency reduction than MoE-OnDemand and MoE-Prefetch, respectively.

End-to-end inference throughput. Figure 11 shows the end-to-end inference throughput across all model configurations. Pre-gated MoE achieves an average 111 tokens/sec throughput over all Switch-Base model configurations, an average $1.5\times$ (max $1.6\times$) and $27\times$ (max $55\times$) improvement over MoE-OnDemand and MoE-Prefetch, respectively. Furthermore, Pre-gated MoE is able to achieve 81% of the throughput of oracular GPU-only solution, demonstrating its superior cost-effectiveness. As for the Switch-Large model with 128 experts, Pre-gated MoE achieves 42 tokens/sec of throughput which is $1.6\times$ and $52\times$ higher than MoE-OnDemand and MoE-Prefetch, respectively.

B. Scalability

As discussed in Section II-B, the majority of MoE’s model size is dominated by expert parameters yet only a small fraction of the experts are actually activated for execution. Consequently, judiciously allocating GPU memory for efficient usage becomes vital in minimizing GPU’s peak memory usage which helps deploy large-scale LLMs. Figure 12 compares the

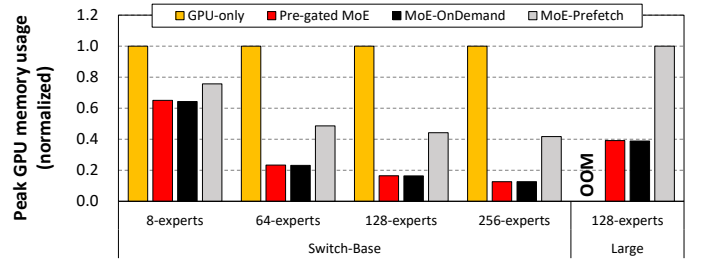


Fig. 12: Peak GPU memory consumption (normalized to GPU-only). We additionally evaluate Switch-Base with 256 experts to further demonstrate Pre-gated MoE’s scalability in deploying larger MoE-based LLMs. For the Switch-Large with 128 experts, GPU-only suffers from an OOM error, so we normalized the memory usage of Pre-gated MoE and MoE-OnDemand to MoE-Prefetch.

peak GPU memory usage of Pre-gated MoE against baseline systems to demonstrate Pre-gated MoE’s scalability.

Among the four designs, GPU-only shows the highest peak memory usage because it solely relies on GPU memory to allocate all of its model parameters and input/output activations. All three CPU-GPU systems are able to significantly reduce peak GPU memory usage, as the memory hungry expert parameters are offloaded to the CPU memory. Also, notice how the GPU memory usage gap between GPU-only and the three CPU offloading based CPU-GPU designs gradually increases as the number of experts are increased. This is because the larger the number of experts are available within an MoE block, the more GPU memory savings the CPU offloading will provide. MoE-Prefetch, however, still consumes an average 51% of GPU-only’s peak GPU memory usage because it always migrates the entire expert parameters to GPU memory. The memory-optimal MoE-OnDemand does much better than MoE-Prefetch as it only migrates activated experts on-demand, showing the lowest peak GPU memory utilization. Our proposed Pre-gated MoE system is able to consume only 23% of GPU-only’s peak GPU memory usage while only incurring 0.2% more GPU memory consumption vs. the memory-optimal MoE-OnDemand.

Overall, these results demonstrate that Pre-gated MoE is capable of reaching the performance provided with the performance-optimal GPU-only (Figure 11) while also achieving the resource-efficiency of the memory-optimal MoE-OnDemand, achieving high scalability to deploy large LLMs.

C. Model Accuracy

In this subsection, we quantify the impact of our pre-gate function on MoE’s model accuracy. Table II compares the model accuracy of SwitchTransformer with and without our pre-gate function employed for various downstream tasks. In Switch-Base with 8 experts, which is the smallest size among our studied model configurations, Pre-gated MoE consistently exhibits slightly higher model accuracy across all downstream tasks. As the model size is increased with larger number of experts, Pre-gated MoE incurs a small accuracy degradation for some of the downstream tasks, but overall it continues to deliver competitive model accuracy results. Nevertheless, this

TABLE II: Effect of our pre-gate function on the model accuracy of Google’s SwitchTransformer. R1 and R2 represent the Rouge-1 and Rouge-2 scores, respectively. For all score metrics, higher is better.

	Xsum		CB Web QA		SQuAD	
	R1	R2	ExactMatch	F1	ExactMatch	F1
Base-8	34.6	13.0	26.0	30.9	77.4	85.8
Pre-gated	34.7	13.0	28.2	32.6	78.2	86.0
Base-128	38.1	16.6	27.4	33.1	81.7	89.2
Pre-gated	38.0	16.5	25.8	32.2	82.2	89.4
Large-128	40.2	18.8	31.0	36.5	82.4	90.1
Pre-gated	40.1	18.6	30.5	36.2	81.9	90.2

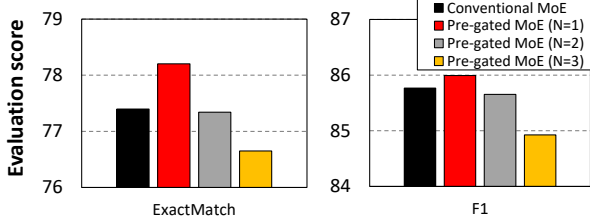


Fig. 13: Effect on model accuracy when changing the activation level of our pre-gate function, from a single MoE block ahead ($N=1$, our default configuration) to 2nd/3rd MoE block ahead ($N=2/3$). Evaluation is conducted over Switch-Base with 8 experts for the closed-book question answering task trained with the SQuAD dataset (ExactMatch (left) and F1 (right) scores are the evaluation metrics for the given task).

magnitude of observed variances in accuracy does not signify a substantial improvement or deterioration in the model’s fundamental capabilities. A detailed analysis on why our pre-gate function improves some of the downstream task’s model accuracy is beyond the scope of this work. In general, Pre-gated MoE’s robust model accuracy observed across different model sizes and different downstream tasks underscores the algorithmic robustness of our proposal.

It is important to emphasize that fine-tuning for both Pre-gated MoE and conventional MoE is done using *the same pre-trained model parameters with the same number of training iterations*. The fact that Pre-gated MoE produces comparable model accuracy under these conditions demonstrates the robustness of our proposal. Furthermore, it also shows that *our proposal can effectively utilize pre-existing resources and training/fine-tuning recipes for deployment* (e.g., pre-trained model parameters from conventional MoE models), enhancing its applicability.

D. Discussion

Pre-gating to activate experts at different blocks. We have so far assumed that our pre-gate function is trained to preemptively select the experts to activate for the next subsequent MoE block. In other words, the pre-gate function’s activation level (N) is a *single* ($N=1$) MoE block ahead of the current MoE block. To explore potential optimizations in the MoE architecture using our pre-gate function, we evaluate the model accuracy of MoE when the pre-gate function is trained to select the experts to activate for the 2nd/3rd subsequent MoE block ahead ($N=2/3$), the result of which is shown in Figure 13.

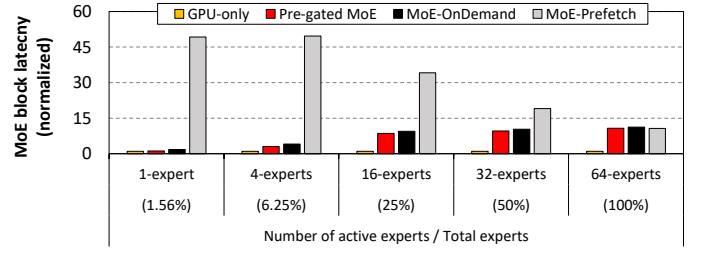


Fig. 14: Effect of the number of activated experts on MoE block latency (normalized to GPU-only). Evaluation is conducted using Switch-Base with 64 experts.

As depicted, our default Pre-gated MoE configuration (pre-gating with activation level-1, i.e., $N=1$) in the Switch-Base model with 8 experts consistently shows the highest model accuracy than the rest of the design points including conventional MoE structure (i.e., selecting experts to activate for the *current* MoE block, $N=0$) as well as pre-gate functions trained to select 2nd/3rd subsequent MoE block ahead ($N=2/3$). Note that the model accuracy gradually decreases as the pre-gate function’s activation level increases (from $N=1$ to 3). We conjecture that the further away the preemptively selected MoE block is from the current pre-gate function, the less likely the current pre-gate function’s input activations will contain useful information to accurately select what experts are most suitable to activate. A detailed evaluation of such is beyond the scope of this work and we leave it as future work.

Number of experts activated. The power of MoE comes from its *sparse* activation of experts (designed to mimic the behavior of the human brain, i.e., specialize regions of the brain tuned for specific tasks), which allows the model architecture to scale its model capacity without proportionally increasing its computational demand. For example, the default model configuration of Google’s SwitchTransformer activates just a single expert (top-1 activation) in a single batch inference, so a SwitchBase model with 64 experts will activate only 1.56% of its experts. For the completeness of our study, we show in Figure 14 the performance of Pre-gated MoE when we manually increase the number of activated experts in Switch-Base with 64 experts from 1 expert (1.56% expert activation) to 64 experts (100% expert activation).

There are two key observations that can be made from this experiment. First, all CPU offloading based solutions (Pre-gated MoE, MoE-OnDemand, and MoE-Prefetch) experience a higher performance degradation vs. GPU-only as the number of activated experts is increased. This is expected because the behavior of MoE becomes similar to a dense LLM model when a larger number of experts are activated (i.e., all model parameters are utilized with 100% activation), rendering CPU offloading solutions less effective. Second, the performance gap between MoE-Prefetch and Pre-gated MoE gradually reduces as the number of activated experts increases. Because MoE-Prefetch migrates the entire expert parameters for every MoE block, a larger number of activated experts reduces the needlessly *overfetched* expert parameters, closing

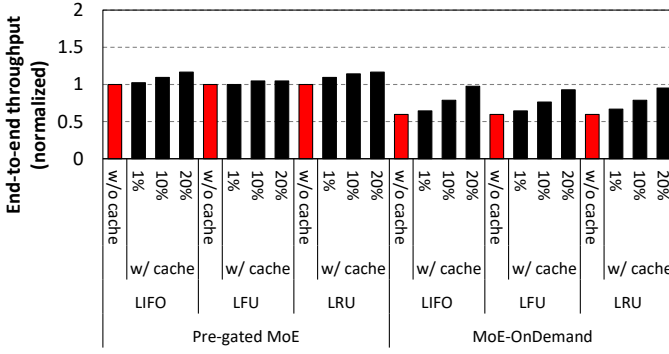


Fig. 15: End-to-end throughput of Pre-gated MoE and MoE-OnDemand when evaluated with the Switch-Large (128 experts) model (normalized to Pre-gated MoE without caching). When caching is enabled, we change the fraction of experts that are cached inside the GPU memory (from 1% to 20%) and compare its effectiveness. For the completeness of our study, we evaluate not only the LIFO policy suggested by [14] but also a least frequently used (LFU) replacement policy [38] and a least-recently used (LRU) replacement policy.

its performance gap against Pre-gated MoE. Nonetheless, for MoE models with sparse expert activations (the most common way of developing an MoE model architecture), Pre-gated MoE demonstrates its robustness and consistently provides superior performance than other CPU-GPU systems.

Caching experts on Pre-gated MoE. Prior work by Huang et al. [14] characterized MoE models for machine translation and language modeling, uncovering the existence of a few hot active experts during inference. Based on such observation, [14] explores *expert buffering* for MoE inference which caches hot, active experts in GPU memory using a last in first out (LIFO) cache replacement policy, while buffering the rest in CPU memory. To evaluate the effectiveness of expert caching [14], [38] on our Pre-gated MoE as well as other CPU offloading based MoE designs, we implement a caching system on top of both Pre-gated MoE and MoE-OnDemand and evaluate its performance.

As shown in Figure 15, caching experts generally provides performance benefits to both Pre-gated MoE and MoE-OnDemand, regardless of the types of the cache replacement policy employed. However, the effectiveness of caching is more pronounced with MoE-OnDemand as the performance overhead incurred with expert migration is more severe under this design point, unlike Pre-gated MoE which is already capable of hiding most of the expert migration latency by overlapping it with expert execution.

Pre-gated MoE with SSD offloading. Prior work [38] evaluates the efficacy of offloading MoE parameters to SSDs as means to deploy even larger LLMs. To evaluate the effectiveness of Pre-gated MoE on top of such design point, we implement Pre-gated MoE and all baseline systems on top of an SSD offloading based MoE serving system, the result of which is summarized in Figure 16. As depicted, the performance benefit of Pre-gated MoE against other baseline systems is decreased compared to a CPU “DRAM” offloaded

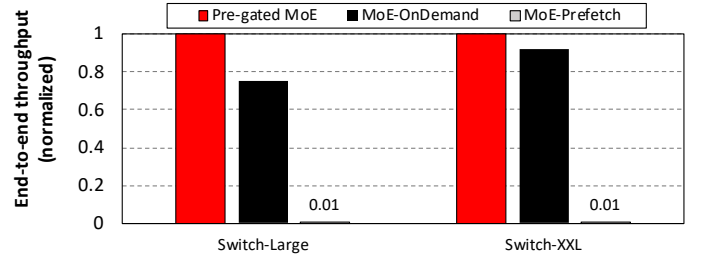


Fig. 16: End-to-end inference throughput of SSD offloading. In this experiment, we additionally evaluate a larger MoE model named Switch-XXL, a SwitchTransformer based model architecture that has the identical configuration as Switch-Large but increases both the feature vector dimension size and the number of heads by 4 \times , amounting to 395 billion parameters (16 \times more than Switch-Large) and 217 GB in model size after quantization is applied. GPU-only suffers from an OOM error, so performance is normalized to Pre-gated MoE.

MoE system. This is because, when the MoE parameters are offloaded to an SSD, the expert migration latency between SSD \rightarrow GPU becomes much longer compared to migrating it from CPU DRAM (due to the much lower slower data transfer bandwidth between SSD vs. CPU DRAM). Consequently, the expert migration latency becomes such a huge end-to-end performance bottleneck that it completely overwhelms the overall system, rendering the effectiveness of any CPU offloading based approaches to become smaller. Nonetheless, Pre-gated MoE consistently delivers higher performance than all other baseline systems demonstrating its robustness.

VII. RELATED WORKS

There exists a large number of prior work exploring ML inference systems for MoE-based LLMs [4], [12], [13], [16], [21], [30], [38], [45]. In this section, we summarize prior work by categorizing them into three different categories: (1) systems for MoE training, 2) systems for MoE inference, and 3) MoE model architectures for efficient MoE deployment.

Systems for the MoE training. Prior work on FastMoE [12] and FasterMoE [13] propose system-level optimizations for multi-GPU solutions, specifically tackling the load-imbalance issue in MoE training. Tutel [16] presents dynamic multi-GPU parallelism and pipelining optimization for distributed MoE training systems. SmartMoE [45] explores efficient search strategies for parallelizing MoE training. TAMoE [4] and Li et al. [21] propose optimizations for MoE training’s all-to-all communication and expert routing. Unlike Pre-gated MoE which focuses on inference, all of these prior works concentrate on MoE training over multi-GPU systems, assuming all model parameters are partitioned across the GPUs allowing each model partition to be stored in GPU memory.

Systems for the MoE inference. DeepSpeed-MoE [30] and Li et al. [21] propose efficient communication optimizations as well as compute kernel optimizations for multi-GPU based MoE inference systems. DeepSpeed-inference [1] proposes to offload memory hungry tensors (e.g., activations, parameters) to the CPU memory and NVMe SSD following ZeRO-offload [35] and ZeRO-infinity [31]. DeepSpeed-inference,

however, did not evaluate their parameter offloading feature to sparse MoE architectures targeting the memory capacity limited expert parameters. HuggingFace Accelerate [15] and SE-MoE [38] respectively implement the MoE-OnDemand and MoE-Prefetch systems we evaluate in this paper, a CPU offloading based MoE inference system.

Efficient MoE model architectures. DeepSpeed-inference [1] proposed PR-MoE and Mixture-of-Student (MoS) architectures, which help significantly compress down the model size of MoE. However, these models require significant modifications to the model architecture based on knowledge distillation and often result in model accuracy degradation. Furthermore, these models are designed for GPU-only configurations, unlike the CPU offloading based Pre-gated MoE. SE-MoE [38] also proposed a compact MoE model architecture based on distillation, compression, and pruning, but it suffers from non-negligible degradation in model accuracy. Our Pre-gated MoE, on the other hand, only requires modest changes to the MoE model architecture without compromising model accuracy.

VIII. CONCLUSION

This paper presents Pre-gated MoE, our algorithm-system co-design for scalable and high-performance MoE inference. Pre-gated MoE effectively addresses the two main challenges of MoE (its large memory footprint and dynamic nature of sparse expert activation) via our novel pre-gate function, which alleviates the dynamic nature of sparse expert activation, allowing our proposed system to address the large memory footprint of MoEs while also achieving high performance. Compared to state-of-the-art MoE inference systems, Pre-gated MoE improves inference throughput while significantly reducing the GPU memory consumption. Importantly, Pre-gated MoE offers comparable model accuracy across various natural language processing tasks, facilitating its adoption in a wide range of real-world applications.

ACKNOWLEDGMENT

This research was supported by the MSIT (Ministry of Science, ICT), Korea, under the High-Potential Individuals Global Training Program (RS-2022-00155958) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation) and this project is (partially) supported by Microsoft Research Asia.

REFERENCES

- [1] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley, and Y. He, "DeepSpeed-Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2022.
- [2] J. Berant, A. Chou, R. Frostig, and P. Liang, "Semantic Parsing on Freebase from Question-Answer Pairs," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013.
- [3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, 2020.
- [4] C. Chen, M. Li, Z. Wu, D. Yu, and C. Yang, "TA-MoE: Topology-Aware Large Scale Mixture-of-Expert Training," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, 2022.
- [5] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, "PaLM: Scaling Language Modeling with Pathways," in *arxiv.org*, 2022.
- [6] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *arxiv.org*, 2018.
- [7] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat, B. Zoph, L. Fedus, M. Bosma, Z. Zhou, T. Wang, Y. E. Wang, K. Webster, M. Pellat, K. Robinson, K. Meier-Hellstern, T. Duke, L. Dixon, K. Zhang, Q. V. Le, Y. Wu, Z. Chen, and C. Cui, "GLaM: Efficient Scaling of Language Models with Mixture-of-Experts," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.
- [8] W. Fedus, B. Zoph, and N. Shazeer, "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity," *The Journal of Machine Learning Research*, vol. 23, p. 39, 2022.
- [9] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2018.
- [10] E. Frantar, S. Ashkboos, T. Hoeffer, and D. Alistarh, "OPTQ: Accurate Quantization for Generative Pre-trained Transformers," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.
- [11] Google, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," https://storage.googleapis.com/deepmind-media/gemini/gemini_v1_5_report.pdf, 2024.
- [12] J. He, J. Qiu, A. Zeng, Z. Yang, J. Zhai, and J. Tang, "FastMoE: A Fast Mixture-of-Expert Training System," in *arxiv.org*, 2021.
- [13] J. He, J. Zhai, T. Antunes, H. Wang, F. Luo, S. Shi, and Q. Li, "Faster-MoE: Modeling and Optimizing Training of Large-Scale Dynamic Pre-Trained Models," in *Proceedings of the Symposium on Principles and Practice of Parallel Programming (PPOPP)*, 2022.
- [14] H. Huang, N. Ardalani, A. Sun, L. Ke, H.-H. S. Lee, A. Sridhar, S. Bhosale, C.-J. Wu, and B. Lee, "Towards MoE Deployment: Mitigating Inefficiencies in Mixture-of-Expert (MoE) Inference," in *arxiv.org*, 2023.
- [15] HuggingFace, "HuggingFace Accelerate," <https://huggingface.co/docs/accelerate/index>, 2022.
- [16] C. Hwang, W. Cui, Y. Xiong, Z. Yang, Z. Liu, H. Hu, Z. Wang, R. Salas, J. Jose, P. Ram, J. Chau, P. Cheng, F. Yang, M. Yang, and Y. Xiong, "Tutel: Adaptive Mixture-of-Experts at Scale," in *arxiv.org*, 2023.
- [17] 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," in *arxiv.org*, 2020.
- [18] Y. J. Kim, A. A. Awan, A. Muzio, A. F. C. Salinas, L. Lu, A. Hendy, S. Rajbhandari, Y. He, and H. H. Awadalla, "Scalable and Efficient MoE Training for Multitask Multilingual Models," in *arxiv.org*, 2021.
- [19] 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 *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

- [20] M. Lewis, S. Bhosale, T. Dettmers, N. Goyal, and L. Zettlemoyer, "BASE Layers: Simplifying Training of Large, Sparse Models," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.
- [21] J. Li, Y. Jiang, Y. Zhu, C. Wang, and H. Xu, "Accelerating Distributed MoE Training and Inference with Lina," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2023.
- [22] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," in *Text Summarization Branches Out*, 2004.
- [23] S. Narayan, S. B. Cohen, and M. Lapata, "Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.
- [24] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.
- [25] NVIDIA, <https://github.com/NVIDIA/FasterTransformer>, 2019.
- [26] OpenAI, "Introducing ChatGPT," <https://openai.com/blog/chatgpt>, 2022.
- [27] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "Carbon Emissions and Large Neural Network Training," in *arxiv.org*, 2021.
- [28] J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, E. Rutherford, T. Hennigan, J. Menick, A. Cassirer, R. Powell, G. van den Driessche, L. A. Hendricks, M. Rauh, P.-S. Huang, A. Glaese, J. Welbl, S. Dathathri, S. Huang, J. Uesato, J. Mellor, I. Higgins, A. Creswell, N. McAleese, A. Wu, E. Elsen, S. Jayakumar, E. Buchatskaya, D. Budden, E. Sutherland, K. Simonyan, M. Paganini, L. Sifre, L. Martens, X. L. Li, A. Kuncoro, A. Nematzadeh, E. Gribovskaya, D. Donato, A. Lazaridou, A. Mensch, J.-B. Lespiau, M. Tsimpoukelli, N. Grigorev, D. Fritz, T. Sottiaux, M. Pajarskas, T. Pohlen, Z. Gong, D. Toyama, C. de Masson d'Autume, Y. Li, T. Terzi, V. Mikulik, I. Babuschkin, A. Clark, D. de Las Casas, A. Guy, C. Jones, J. Bradbury, M. Johnson, B. Hechtman, L. Weidinger, I. Gabriel, W. Isaac, E. Lockhart, S. Osindero, L. Rimell, C. Dyer, O. Vinyals, K. Ayoub, J. Stanway, L. Bennett, D. Hassabis, K. Kavukcuoglu, and G. Irving, "Scaling Language Models: Methods, Analysis & Insights from Training Gopher," in *arxiv.org*, 2022.
- [29] 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," *The Journal of Machine Learning Research*, vol. 21, p. 67, 2020.
- [30] 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 *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.
- [31] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, "ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2021.
- [32] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ Questions for Machine Comprehension of Text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016.
- [33] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-Shot Text-to-Image Generation," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.
- [34] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "MLPerf Inference Benchmark," in *arxiv.org*, 2019.
- [35] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, "ZeRO-Offload: Democratizing Billion-Scale Model Training," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2021.
- [36] S. Roller, S. Sukhbaatar, A. Szlam, and J. E. Weston, "Hash Layers For Large Sparse Models," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, 2021.
- [37] 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 *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.
- [38] L. Shen, Z. Wu, W. Gong, H. Hao, Y. Bai, H. Wu, X. Wu, J. Bian, H. Xiong, D. Yu, and Y. Ma, "SE-MoE: A Scalable and Efficient Mixture-of-Experts Distributed Training and Inference System," in *arxiv.org*, 2023.
- [39] M. Shoenybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism," in *arxiv.org*, 2020.
- [40] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhume, G. Zerveas, V. Korthikanti, E. Zhang, R. Child, R. Y. Aminabadi, J. Bernauer, X. Song, M. Shoenybi, Y. He, M. Houston, S. Tiwary, and B. Catanzaro, "Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model," in *arxiv.org*, 2022.
- [41] N. Team, M. R. Costa-jussà, J. Cross, O. Çelebi, M. Elbayad, K. Heffernan, E. Kalbassi, J. Lam, D. Licht, J. Maillard, A. Sun, S. Wang, G. Wenzek, A. Youngblood, B. Akula, L. Barrault, G. M. Gonzalez, P. Hansanti, J. Hoffman, S. Jarrett, K. R. Sadagopan, D. Rowe, S. Spruit, C. Tran, P. Andrews, N. F. Ayan, S. Bhosale, S. Edunov, A. Fan, C. Gao, V. Goswami, F. Guzmán, P. Koehn, A. Mourachko, C. Ropers, S. Saleem, H. Schwenk, and J. Wang, "No Language Left Behind: Scaling Human-Centered Machine Translation," in *arxiv.org*, 2022.
- [42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, 2017.
- [43] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "HuggingFace's Transformers: State-of-the-art Natural Language Processing," in *arxiv.org*, 2020.
- [44] C.-J. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. A. Behram, J. Huang, C. Bai, M. Gschwind, A. Gupta, M. Ott, A. Melnikov, S. Candido, D. Brooks, G. Chauhan, B. Lee, H.-H. S. Lee, B. Akyildiz, M. Balandat, J. Spisak, R. Jain, M. Rabbat, and K. Hazelwood, "Sustainable AI: Environmental Implications, Challenges and Opportunities," in *arxiv.org*, 2022.
- [45] M. Zhai, J. He, Z. Ma, Z. Zong, R. Zhang, and J. Zhai, "SmartMoE: Efficiently Training Sparsely-Activated Models through Combining Off-line and Online Parallelization," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2023.
- [46] Y. Zhou, T. Lei, H. Liu, N. Du, Y. Huang, V. Y. Zhao, A. M. Dai, Z. Chen, Q. V. Le, and J. Laudon, "Mixture-of-Experts with Expert Choice Routing," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, 2022.
- [47] S. Zuo, X. Liu, J. Jiao, Y. J. Kim, H. Hassan, R. Zhang, T. Zhao, and J. Gao, "Taming Sparsely Activated Transformer with Stochastic Experts," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

APPENDIX

A. Abstract

This artifact evaluation repository contains the base implementation of Pre-gated MoE. The artifact is designed to demonstrate the performance of our Pre-gated MoE compared to the three baselines mentioned in this paper. As detailed in Section V, we built our Pre-gated MoE on top of Google’s SwitchTransformer [8], utilizing NVIDIA’s FasterTransformer [25].

B. Artifact check-list (meta-information)

- **Algorithm:** Pre-gated MoE algorithm
- **Program:** C++, Python
- **Model:** Google’s SwitchTransformer
- **Hardware:** At least one GPU with 40GB of memory and a CPU with 128GB of memory.
- **Output:** Key results of our paper including average MoE block latency, inference throughput, and peak GPU memory consumption.
- **How much disk space required (approximately)?:** Over 100GB for storing the model parameters.
- **How much time is needed to prepare workflow (approximately)?:** 6 hours
- **How much time is needed to complete experiments (approximately)?:** 30 minutes
- **Publicly available?:** Yes
- **Archived:** <https://doi.org/10.5281/zenodo.10976343>

C. Description

1) *How to access:* The artifact is available in archival repositories on Zenodo and GitHub.

- Zenodo: <https://doi.org/10.5281/zenodo.10976343>
- GitHub: https://github.com/ranggihwang/Pregated_MoE

2) *Hardware dependencies:* To reproduce the results presented in the paper, the following hardware is required:

- A CPU with at least 128GB of memory.
- A GPU with at least 40GB of memory. (We recommend using recent GPUs, such as the NVIDIA A100 with 80GB HBM, for optimal performance.)
- Additionally, ensure there is more than 100GB of disk storage available for model parameters.

3) *Software dependencies:* We advise using the Docker image following the description in our repository to circumvent most software-related issues. The repository includes a script that automates the installation of all necessary software dependencies for compiling and running the artifact. Further details are provided in the repository documentation.

4) *Data sets:* To use the artifact, it is necessary to download the model weights for the SwitchTransformer from HuggingFace. Our repository provides detailed instructions and scripts for downloading these model weights.

D. Installation

- 1) Create a directory for model preparation.
- 2) Launch a Docker container using the following command, replacing `${DATA_PATH}` with the path to your model preparation directory:

```
docker run -ti --gpus all --shm-size 5g --name
pregated -v ${DATA_PATH}:/data nvcr.io/nvidia
/pytorch:22.09-py3 bash
```

- 3) Clone the repository and initiate the build process. The `-DSM` parameter should match your GPU’s compute capability. Refer to the documentation to select the appropriate value for your setup.

```
# build on A100
mkdir -p FasterTransformer/build
cd FasterTransformer/build
cmake -DSM=80 -DCMAKE_BUILD_TYPE=Release -
DBUILD_PYT=ON -DBUILD_MULTI_GPU=ON ..
make -j
```

- 4) Install the required Python dependencies:

```
pip install -r ../examples/pytorch/t5/requirement.
txt
```

E. Experiment workflow

- 1) Prepare the models.

```
mkdir /data/ft
cd /workspace/FasterTransformer/
./scripts/convert.sh
```

- 2) Begin the evaluation using the provided script:

```
cd /workspace/FasterTransformer/
# logs will be output here
mkdir logs/
python scripts/eval_all.py
```

F. Evaluation and expected results

The script will generate output files in CSV format including `block_lats.csv`, `throughputs.csv` and `peak_mems.csv`, which contain data on MoE block latencies, inference throughputs, and peak memory usage, respectively. The results are shown in Figure 10, Figure 11, and Figure 12.

G. Experiment customization

To customize the experiment, you can modify the evaluation configuration in `scripts/eval_all.py`

H. Notes

I. Methodology

Submission, reviewing and badging methodology:

- <https://www.acm.org/publications/policies/artifact-review-and-badging-current>
- <http://cTuning.org/ae/submission-20201122.html>
- <http://cTuning.org/ae/reviewing-20201122.html>