QoS-Efficient Serving of Multiple Mixture-of-Expert LLMs Using Partial Runtime Reconfiguration

HamidReza Imani [1] **Jiaxin Peng** [1] **Peiman Mohseni** [2] **Abdolah Amirany** [1] **Tarek El-Ghazawi** [1]

https://github.com/hamid-74/Multi-MoE

Abstract

The deployment of mixture-of-experts (MoE) large language models (LLMs) presents significant challenges due to their high memory demands. These challenges become even more pronounced in multi-tenant environments, where shared resources must accommodate multiple models, limiting the effectiveness of conventional virtualization techniques. This paper addresses the problem of efficiently serving multiple fine-tuned MoE-LLMs on a single GPU. We propose a serving system that employs *similarity-based expert consolidation* to reduce the overall memory footprint by sharing similar experts across models. To ensure output quality, we introduce *runtime partial reconfiguration*, dynamically replacing non-expert layers when processing requests from different models. As a result, our approach achieves competitive output quality while maintaining throughput comparable to serving a single model, and incurs only a negligible increase in time-to-first-token (TTFT). Experiments on a server with a single NVIDIA A100 GPU (80GB) using Mixtral-8x7B models demonstrate an 85% average reduction in turnaround time compared to NVIDIA's multi-instance GPU (MIG). Furthermore, experiments on Google's Switch Transformer Base-8 model with up to four variants demonstrate the scalability and resilience of our approach in maintaining output quality compared to other model merging baselines, highlighting its effectiveness.

[1]Department of Electrical and Computer Engineering, The George Washington University [2]Computer Science and Engineering Department, Texas A&M University. Correspondence to: HamidReza Imani <hamidreza@gwu.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

1. Introduction

Incorporating the Mixture-of-Experts (MoE) concept (Jacobs et al., 1991; Jordan & Jacobs, 1994) into transformer-based large language models (LLMs) (He et al., 2024) has enabled an expansion in model parameters, improving the quality of generated outputs across various language tasks (Shazeer et al., 2017; 2018; Fedus et al., 2022; Du et al., 2022; Kim et al., 2021; Zuo et al., 2021; Lin et al., 2021a; Rajbhandari et al., 2022). Rather than using a single block (e.g., a feed-forward layer), MoE-based models employ a gating system and multiple parallel instances of the block (known as experts) with identical structures. During the forward pass calculation, based on the current input, the gating system assigns weights to each expert, and one or more top experts will be selected to produce the block's output. This allows the model to scale in size while imposing negligible computation overhead (in the case of selecting only one expert).

However, such scaling comes with its own drawbacks. MoE LLMs have significantly higher memory footprints and do not fit in most of the recently developed GPU memories. For example, Google's Switch Transformer (Fedus et al., 2022) model has a total of 1.6 trillion parameters which occupies 3.1 TB of memory (16 bits for each parameter). Therefore, for deployment in constrained environments, the model parameters should be partitioned between CPU and GPU memories. Accordingly, in the calculation of forward pass, repetitive host-to-device copy operations will be required which stall the inference pipeline and limit the overall throughput of the system.

At the same time, an increasing number of private entities are adopting customized MoE-based LLMs to meet their specific needs and ensure the required output quality. These LLMs are typically fine-tuned versions of a general-purpose model, trained on specific datasets (Jiang et al., 2024) which should be deployed in local settings for efficiency and customization purposes. However, most of these entities lack access to sufficient GPU memory to accommodate large models, as they typically have only a limited number of GPUs. This issue becomes even more challenging when

multiple entities attempt to perform inference with their customized models simultaneously in the described constrained environment and must share resources.

Current research efforts to improve MoE inference efficiency and performance focus on reducing the overall memory footprint (Lin et al., 2023; Xiao et al., 2023; Huang et al., 2024; Eliseev & Mazur, 2023; Dettmers & Zettlemoyer, 2023) and host-to-device communication overhead (Kamahori et al., 2024; Eliseev & Mazur, 2023; Xue et al., 2024; Suvizi et al., 2024) of a single MoE model. Quantization and mixed precision approaches directly reduce the memory footprint, which allow the server to fit more parameters in GPU memory. In another direction, previously proposed expert caching mechanisms identify the most frequently used experts and prioritize them for loading into GPU memory. As a result, these approaches reduce the communication overhead by increasing the probability of future required experts being available on the GPU. However, these approaches face considerable performance degradation in the case of a system that supports the inference of multiple MoEs.

Conventional virtualization and sharing techniques, such as space and time sharing (El-Araby et al., 2009; Li et al., 2011; Huang et al., 2010), are also inefficient for this class of problem. Space-sharing approaches, like NVIDIA Multi-Instance GPU (MIG) (Choquette et al., 2021), divide the available GPU resources among the requesting users, which amplifies the communication overhead. In contrast, in time sharing approaches, requests for different models cause the currently loaded model to be offloaded and another model to be uploaded onto the GPU, a process that can take up to two minutes.

In this paper, to optimize the performance and availability of a multi-model inference system, we propose a serving system which is expected to achieve a throughput comparable to serving a single model. This feature will be supported at the cost of a slight increase in time-to-first-token (TTFT) and decrease in generated output quality. Although this paper focuses specifically on fine-tuned LLMs with identical architectures and text generation tasks, the proposed technique is applicable to MoEs with different sizes and number of parameters. In summary, we make the following contributions:

- Expert Loading on GPU via Similarity-Based Consolidation: We propose to find a middle point between different fine-tuned models being served. Since experts are the major contributor to the model's large size, we propose to find the most similar experts and load them as common experts in a round-robin approach to reduce the overall memory footprint.
- Runtime Partial Reconfiguration: Despite having a lower number of parameters compared to experts, the contribution of non-expert layers is paramount in the forward pass calculation. Therefore, to serve requests for a certain model, we instantly reconfigure the system by only swapping the non-expert layers loaded on the GPU with non-expert layers of the requested model.
- We implement and deploy our serving approach on a system equipped with a single NVIDIA A100 GPU. To evaluate the impact of our method on generated output quality, we benchmark two model families—Mixtral-8x7B and Google Switch Transformer Base-8, on tasks such as language generation and instruction following. We use a Poisson distribution with varying arrival rates to assess the quality of service (QoS) provided by our approach, and compare it against NVIDIA MIG as a baseline.

2. System Overview and Problem Statement

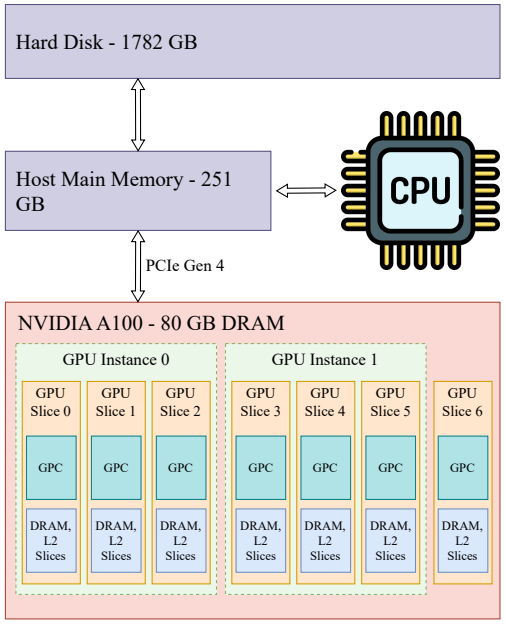

Figure 1. Diagram of an inference system with a single NVIDIA A100 GPU.

The components of a single-GPU inference system are depicted in Figure 1. In this single-GPU system, the models are originally stored in the hard disk. For every ML serving application, the parameters must first be loaded into the host's main memory. If the inference is to be performed on the accelerator, the model parameters will be transferred from the host's main memory to the GPU's memory. After the load is completed, the model will be ready for inference, and for every new request, the inputs are copied to the GPU, processed, and the results are written back to the CPU's

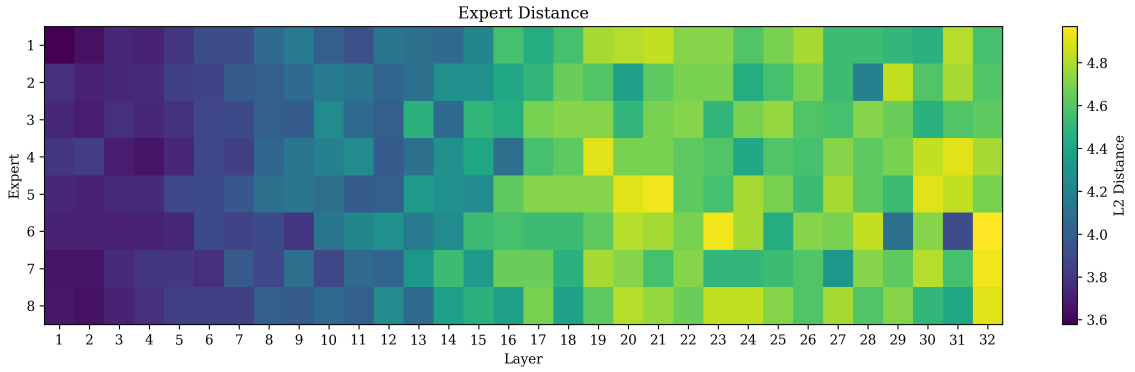

Figure 2. Expert-to-expert distance between Mixtral base and Instruct model: Despite being fine-tuned for different tasks, experts in the same positions exhibit similarity.

main memory.

However, when handling an MoE LLM, the model does not completely fit in the accelerator's memory. Originally proposed in (Eliseev & Mazur, 2023), offloading techniques load the non-expert parameters of the model to GPU initially and fill the remaining space with as many experts as possible. When performing inference for a sparse MoE model, if the required expert is not available on GPU's memory, it must be transferred from the CPU's memory through the PCIe link, which stalls the inference process and slows the token generation speed.

In this paper, we consider a single-GPU inference system that has to support two or more MoE LLMs. For such a problem, accelerator resources must be shared and the conventional virtualization approaches are considered as follows:

Time Sharing. For this approach, there will be a queue that handles requests coming from the users. If the next request's requested model matches with the currently loaded model, the inference will be performed. However, in the case of miss-match, the entire model must be offloaded to the CPU's main memory, and the requested model must be uploaded onto the GPU. Due to the large size of the discussed models and based on our experiments, each model swap can take up to 2 minutes, which introduces a considerable delay in the inference process.

Space Sharing. For space-sharing, the resources on the GPU should be split among the models we are trying to serve. The A100 NVIDIA GPU consists of seven GPU slices and starting with NVIDIA Ampere architecture each slice is capable of operating independently. Each slice comes with its own memory (DRAM and L2 slices), a GPU Processing Cluster (GPC), and host-to-device link bandwidth. On an A100, each GPC consists of 7 Texture Processing Clusters (TPCs), and each TPC is comprised of 2 streaming multiprocessors (SMs). NVIDIA MIG (Choquette et al., 2021) allows us to make different partitions using different numbers of GPU slices. For example, in Figure 1, the system is partitioned into two equal GPU instances which can simultaneously and independently support inference for two models.

3. Approach

3.1. Parameter Layout on GPU: A Consolidated Middle Ground

In the described environment, the parameters of the MoE model can be grouped into three categories: on-device non-expert weights, on-device expert weights, and off-device expert weights. The on-device expert weights constitute the majority of the parameters loaded into GPU memory. Accordingly, in this section, we present the algorithm that generates a unified layout for the on-device expert weights, which is used during the initial model load.

As shown in Figure 2, the expert-to-expert distances between a fine-tuned Instruct model and a base model are not equal. In this context, we flatten the weights of an expert block into a 1D tensor and define the distance as the $L2$ distance between experts from different models. As depicted, the distance between experts generally increases in the deeper layers, which aligns with the experiments conducted in (Shen et al., 2024). While the distances between same expert positions are in the range of 3.6 to 4.9, expert-to-expert distances with different layer and expert numbers can vary from 150 to 250. This observation leads us to propose using the experts interchangeably.

Therefore, to find the most similar experts and generate a unified layout, we associate a degree of similarity to each expert coordinate specified by layer and expert numbers. In the case of having only two models for serving, the degree of similarity for each location is the expert-to-expert dis-

Algorithm 1 Initial Expert Loader

Input: List of models $\mathcal{M}$, expert capacity C
# Iterate over layers and experts
for $(iL, iE) \in \{1, \cdots, L\} \times \{1, \cdots, E\}$ **do**
  # Iterate over all model pairs
  for $(i, j) \in \mathcal{M} \times \mathcal{M}$ **do**
    `distance`$[iL][iE] \leftarrow$ `distance`$[iL][iE] +$ $\|\mathcal{M}[i]$.`expert`$[iL][iE] - \mathcal{M}[j]$.`expert`$[iL][iE]\|_2$
  end for
end for
# Initialize counter for expert assignments
Initialize `count` $\leftarrow 0$
# Assign experts based on sorted similarity
for (iL, iE) in `sorted(distance)` **while** `count` $< C$ **do**
  `count` $\leftarrow$ `count` + 1
  $idx \leftarrow$ `count` $\mod |\mathcal{M}|$
  `map`$[iL][iE] \leftarrow \mathcal{M}[idx]$.`id`
  Load $\mathcal{M}[idx]$.`expert`$[iL][iE]$ to GPU
end for
# Load non-expert model weights
Set `loadedModel` to the first model id in $\mathcal{M}$
Load non-expert weights of `loadedModel` to GPU

tance for a given $(layer, expert)$ pair. However, for higher number of models, the distance for each expert location is computed as the sum of distances across all model-to-model combinations.

Subsequently, each expert location will be ranked based on the defined similarity measure, with the highest rank indicating the greatest similarity, corresponding to the lowest distance. To generate the initial loader's expert map, starting from the highest rank, each expert location will be assigned to a specific fine-tuned model using a round-robin approach. For instance, if we are serving two models, the expert locations with odd ranks will be assigned to the first model, while those with even ranks will be assigned to the second model. Although the time spent generating this expert map can be substantial and depends on the number of models, this process only needs to be performed once for each combination prior to load time and can be done offline.

This approach sets the granularity of model merging at the expert level and ensures that the loaded parameters are equally representative of all the models being served. After generating the expert map, the parameters should be loaded to the GPU memory. Based on the available GPU memory and the size of non-expert parameters, the total number of experts that can fit on the GPU will be calculated. To start, non-expert parameters of a randomly selected model from our model list will be loaded to GPU's memory. Following this, the loader begins with the highest-ranked expert locations, and transfers their corresponding parameters.

Algorithm 2 Inference Orchestrator

Input: prompt `prompt`, requested model `targetModel`
if `targetModel` is not `loadedModel` **then**
  Load `targetModel` non-experts on GPU
end if
# Tokenize the input prompt and embed it into activations
repeat
  `act` = tokenize and embed `prompt`.
  # Process the token through all transformer layers
  for $iL = 1$ to L **do**
    # Apply the multi-headed attention
    `act` = `layers[`iL`].MultiheadAttention(act)`
    # Determine which expert to use
    (iL, iE) = `layers[`iL`].gate(act)`
    if `map[`iL`][`iE`] is not on GPU` **then**
      # Load expert from CPU if not on GPU
      Load `targetModel.expert[`iL`][`iE`]` to GPU
      `act` = `targetModel.expert[`iL`][`iE`(act)`
    else
      # Use the preloaded expert
      id = `map[`iL`][`iE`]`
      `act` = $\mathcal{M}$`[`id`].expert[`iL`][`iE`(act)`
    end if
    Apply Add & Norm and compute `act` of next layer
  end for
  Predict `nextToken` using `act`
  Concatenate `nextToken` to the end of `prompt`
until `nextToken` is `eos` or required number of output tokens generated

As summarized in Algorithm 1, the loader's inputs are a list of models being served and the total expert capacity. At the beginning of the loading process, all the models' weights are stored on CPU's main memory. By the end of the process, a fraction of each model's parameters is loaded onto the GPU, collectively forming a unified model ready for inference. Moreover, the generated map is a table that keeps track of each expert and its corresponding model id which will route each input in performing forward pass calculations.

3.2. Inference Orchestration

The loader, as explained in Algorithm 1, loads experts from different models. While this approach significantly reduces the memory footprint required to serve all the models, it may decrease the quality of output generation for tasks specific to each model variant.

The Mixtral model evaluated in this paper contains 1,605.64

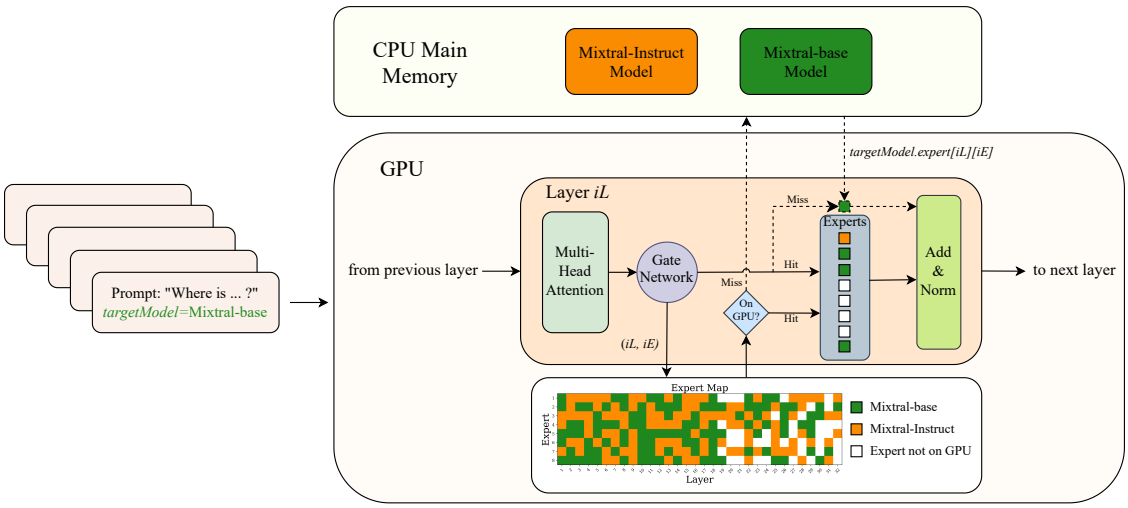

Figure 3. Inference process for each layer of the consolidated model: In the case of an expert hit, the inference is performed using the already loaded expert, which may not necessarily belong to the requested model. However, in the case of a miss, the corresponding expert is loaded from the host's memory to preserve the quality of the generated output.

million non-expert parameters. It has 32 layers, each with 8 experts, and uses top-2 sparse routing. During each forward pass, a total of 1,605.64 million non-expert parameters and $32 \times 2 \times 176.16$ million expert parameters (with each expert contributing 176.16 million parameters) are actively involved in computations. Consequently, the non-expert parameters contribute approximately 14.2% to the generated output.

In contrast to experts, non-expert parameters are always utilized in each forward pass, regardless of activations. Furthermore, their total size is approximately 3 GB (2 bytes per parameter). Based on the described system, it takes approximately a second to transfer all non-expert parameters of a given model via the host-to-device PCIe link. Therefore, for each new request to the serving system, we propose swapping the currently loaded non-expert parameters with those of the $targetModel$, which are transferred to the GPU. Although this process incurs a one-time cost per request and increases the TTFT, it ultimately enhances the quality of the generated output.

Accordingly, the inference orchestrator begins handling each request by comparing the $targetModel$ with the $loadedModel$. If they match, inference is performed immediately. However, if they do not match, the $targetModel$ non-expert parameters are uploaded to the GPU, replacing the currently loaded non-expert parameters before proceeding with inference.

As summarized in Algorithm 2, during the inference phase, each sequence undergoes tokenization and embedding processes before arriving at the first layer of the model. Within each layer, the activations are first processed by a multi-headed attention layer and then passed to the gate network. The gate network generates weighted averaging coefficients for each expert, and in a sparse gating scenario, the top experts are selected to produce the output of the layer.

As depicted in Figure 3, pairs of layer and expert numbers $((iL, iE))$ are generated, and the status of the requested expert is retrieved from the expert map. If the expert is already loaded on the GPU, the corresponding expert's result will be calculated, although it is not guaranteed to belong to the $targetModel$. However, in the case of an expert miss (dashed lines in Figure 3), a load request is sent to the CPU's main memory. Since all models are stored in the CPU's main memory, the requested expert will belong to the $targetModel$, preventing further quality degradation. Finally, after processing all layers, the activations generated by the last layer pass through the *language model head* to produce the output logits.

4. Evaluation Methodology

We evaluate our serving system using the Mixtral-8x7B-v0.1 and Google Switch Transformer Base-8 family of models. The Mixtral model has 32 layers and 8 experts per layer and comes in two variants. The Switch Transformer model comes in four variants and has a total of 96 experts, which are distributed equally across 12 layers (8 experts per layer). Since in our design, both QoS and the application output quality is affected, we evaluate the proposed system through experiments that independently assess each aspect of its

functionality.

4.1. Quality of Service

Since the Switch Transformer Base-8 model has a considerably lower memory footprint (1.5 GB) compared to Mixtral and does not constrain our environment, we use only the Mixtral model family for this experiment. We evaluate our proposed system in a scenario where requests arrive independently for each model we support. We consider two separate Poisson processes to generate requests for each supported model. To demonstrate the system's responsiveness and performance under varying loads, we evaluate the system at different arrival rates (λ). Each request to the system is a prompt consisting of 20 tokens, with the number of requested output tokens set to 25. To compare the QoS of each approach, we measure the average TTFT, average turnaround time, and the total number of processed requests (throughput).

We implement our loader and orchestrator using the PyTorch (Paszke et al., 2019) framework. Our experiments are conducted on a server with an AMD 16-Core MILAN CPU and a single 80GB NVIDIA A100 GPU, connected via a PCIe Gen4 host-to-device interconnect, as outlined in Section 2. QoS metrics are measured from PyTorch's perspective using Python's time package. It is important to note that other inference optimizations, such as Flash Attention (Dao et al., 2022), are complementary to the approach presented in this paper and fall outside the scope of this work. To evaluate the system's performance, we focus on single-batch requests, providing a controlled setting for our experiments.

QoS Baselines. We compare our serving system against two baselines. The first baseline represents a scenario in which a single Mixtral model is served, but with an arrival rate that is twice that of each Poisson process used to evaluate the proposed approach. The second baseline involves partitioning the GPU resources into two using NVIDIA MIG, with each partition assigned to an independent process that handles requests for the base and Instruct models, separately.

4.2. Application Quality

The designed serving system must provide access to multiple models. Therefore, for the Mixtral model, we measure the quality of the generated output using two specific families of benchmarks corresponding to each model variant: the base model and the Instruct model. During the evaluation of each benchmark family, the non-expert layers of the model are set to the respective model variant.

For the base model, we evaluate the quality of text generation using the WikiText2 (Merity et al., 2016), PTB (Marcus et al., 1994), and C4 (Raffel et al., 2020) datasets. For each dataset, we assess the models' generated output by measuring perplexity using 128 samples, with each sample consisting of 2048 tokens.

To demonstrate the quality of output for instruct tasks, we evaluate each model using the MT-Bench (Zheng et al., 2023) benchmarks. MT-Bench is a benchmark specifically designed for instruct models and includes 80 instructions across 8 different categories. Each request prompts the language model to generate an output, and requests may include a follow-up instruction to refine the output and further improve its quality. The quality of the responses is then judged by the GPT-4 (Achiam et al., 2023) model, which assigns each response a grade from 0 to 10.

Moreover, to demonstrate contextual understanding and general effectiveness of each model variant, we assess their performance using HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2020), and TruthfulQA (Lin et al., 2021b) benchmarks. For MMLU and TruthfulQA, we use 5-shot prompts, and for HellaSwag, we use 0-shot prompts. For MMLU and HellaSwag, we evaluate each model with 1000 random samples, and for TruthfulQA, we evaluate them with 812 samples.

Baselines. We compare the quality of the generated output from the proposed consolidated model against the following models: Mixtral-8x7B-v0.1-base, Mixtral-8x7B-v0.1-Instruct, and Mixtral-8x7B-v0.1-avg. The first two models are official releases by Mistral-AI. However, proposed in (Izmailov et al., 2018), the Mixtral-8x7B-v0.1-avg model is generated by averaging the weights of the first two models. We include Mixtral-8x7B-v0.1-avg in the comparison to evaluate the performance of the proposed model against conventional model-merging approaches. It is important to note that for benchmarks measuring perplexity, the non-expert parameters in our approach are from the base model, while for all other benchmarks, they are from the Instruct model.

4.3. Scalability

To demonstrate the scalability and applicability of our approach to other model architectures with a higher number of fine-tuned variants, we also evaluate our approach against the Averaging baseline (Izmailov et al., 2018) using Google's Switch Transformer Base-8 family of models. For this experiment, we use the base model provided by Google along with three other community-provided fine-tuned variants:

- Model A: `emre/switch-base-8-finetuned-samsum`
- Model B: `google/switch-base-8`
- Model C: `glamprou/switch-base-8-sst2`
- Model D: `glamprou/switch-base-8-mnli`

To evaluate how output quality is affected by the number of merged models, we consider the following serving scenarios:

- Serving models A & B
- Serving models A & B & C
- Serving models A & B & C & D

5. Results

5.1. Performance and Quality of Service

Figure 4 compares the throughput of each serving approach across varying arrival rates. For NVIDIA MIG, the reported throughput represents the combined throughput of two independent GPU instances. At lower arrival rates, all three approaches exhibit low throughput, which is attributed to system under-utilization, meaning they can handle all incoming requests within the experiment's time window. As the arrival rate increases, throughput improves until reaching distinct ridge points for each approach. For both the proposed system and the single-model serving approach, the ridge point occurs at $\lambda = 0.060$, while for NVIDIA MIG, it is observed at $\lambda = 0.040$. This indicates that the proposed system and the single-model approach have higher capacities and can continue handling more requests at higher arrival rates, whereas NVIDIA MIG's performance plateaus earlier.

Table 2 illustrates the average TTFT and turnaround times. Single-model serving and the proposed system achieve significantly lower TTFT (a maximum of 5.86 seconds) and turnaround times (a maximum of 49.67 seconds) compared to NVIDIA MIG, which supports the results depicted in Figure 4. As demonstrated, each NVIDIA MIG instance takes an average of 49 seconds to complete a request.

Moreover, comparing the proposed system against the single-serving approach highlights the overhead of non-expert runtime reconfiguration. As shown in the TTFT section of Table 2, our approach introduces a delay of approximately half a second on average. However, the actual measured latency for swapping non-experts is approximately 1.2 seconds, which does not occur for consecutive requests targeting the same $targetModel$. It is also important to note that the numbers reported in this subsection are specific to the requests defined earlier and may vary for prompts with different lengths and requested numbers of output tokens.

To provide deeper insights into the performance of each approach, the inference latency for each model layer is presented in Table 1. As mentioned in (Austin et al., 2025), during the generation phase of a transformer-based LLM, the attention kernel is memory-bound. This is because the results of prior computations, stored in the KV cache, must be copied from the GPU's GMEM to SMEM. Consequently, each GPU instance in NVIDIA MIG has sufficient compute resources, and we do not observe a significant increase in the latency of the attention layer ($0.72 \rightarrow 0.78$).

Table 1. Latency (milliseconds) of a layer for different hit rates (out of 2 experts).

	Attention	Hit Rate		
		0 expert	1 expert	2 experts
Single/ Proposed	0.72	56.8	29.2	1.2
NVIDIA MIG	0.78	104.3	54.1	1.7

Table 2. Average TTFT and turnaround time (seconds) for the proposed and compared serving approaches.

	TTFT	Turnaround Time
Single	0.89	8.34
Proposed	1.41	8.78
NVIDIA MIG	5.86	49.67

However, the expert block is compute-bound (Austin et al., 2025). Therefore, when both required experts are available in GMEM, the latency increase is more noticeable ($1.2 \rightarrow 1.7$). On the other hand, when the hit rate decreases for the expert block, an overhead (27.8 ms for each expert) is introduced due to copying experts from the CPU's DRAM to the GPU's GMEM via the PCIe link. In our approach, when serving a single model, the full bandwidth of the PCIe link is available to the process. However, using NVIDIA MIG with two GPU instances splits the available PCIe bandwidth between them, nearly doubling the imposed overhead (51.3 ms for each expert).

5.2. Quality of Generated Output

Table 3 presents a comparative analysis of the generated output quality across various models, evaluating perplexity on the WikiText, C4, and PTB datasets, as well as performance metrics on MT-Bench, MMLU, HellaSwag, and TruthfulQA benchmarks. Lower perplexity scores indicate better language modeling capabilities, while higher scores indicate improved performance on the remaining tasks. While the base model serves as the optimal baseline for text generation, the proposed approach outperforms the averaged model and achieves lower perplexity across all datasets, except for PTB, which has been observed to show irregular performance improvements even at higher levels of quantization (Eliseev & Mazur, 2023).

On MT-Bench, the proposed model outperforms others with

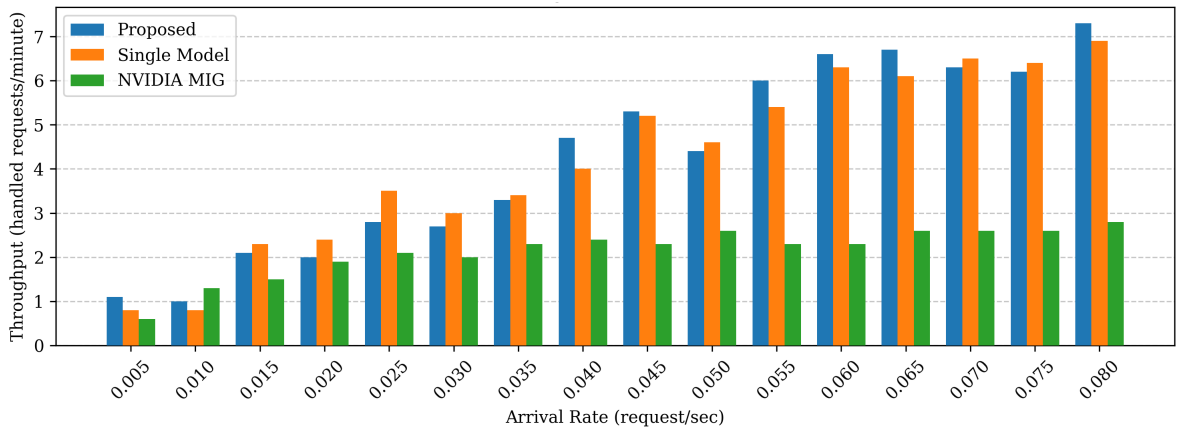

Figure 4. The throughput of each serving approach is measured in completed requests per minute. Since NVIDIA MIG utilizes two independent processes to handle requests for different models, the arrival rate for each instance is half of the value shown on the x-axis, and the reported throughput is the sum of the throughput for both instances. The reported values represent the average over five independent runs.

Table 3. The quality of the generated output, evaluated across different task families, including language generation (Perplexity), instruction following (MT-Bench), and general-purpose multiple-choice questions.

Model	WikiText ↓	C4 ↓	PTB ↓	MT-Bench (out of 10) ↑			MMLU ↑	HellaSwag ↑	TruthfulQA ↑
	(Perplexity)			1st Turn	2nd Turn	Avg	(5-shot)	(0-shot)	(5-shot)
base	3.81	7.24	13.59	3.01	2.16	2.60	70.0%	81.1%	66.6%
Instruct	4.12	7.62	14.60	8.28	7.98	8.13	71.2%	83.0%	73.6%
Avg	3.89	7.38	13.44	8.38	7.61	7.99	72.2%	82.0%	73.0%
Proposed	**3.85**	**7.33**	**13.90**	**8.33**	**7.98**	**8.16**	**71.7%**	**82.2%**	**70.6%**

an average score of 8.16, closely aligning with the high-quality outputs of the Instruct model. Notably, although the averaged model demonstrates superior performance in the first turn, its quality drops significantly when responding to the follow-up refinement prompt and exhibits high variance.

Additionally, the proposed system achieves competitive results on MMLU (71.7%), HellaSwag (82.2%), and TruthfulQA (70.6%) which demonstrates a balanced performance. These results highlight the ability of the proposed system to maintain high output quality at other general-purpose benchmarks.

5.3. Scalability

Presented in Figure 5, we report the ROUGE-1 scores (an N-gram based summarization evaluation metric) for a summarization task on the SAMSum dataset (Gliwa et al., 2019), which Model A is specifically finetuned for. As shown, increasing the number of merged models significantly degrades output quality for the Averaging baseline, with ROUGE-1 scores dropping from $0.49 \rightarrow 0.42 \rightarrow 0.33 \rightarrow 0.25$. However, our approach demonstrates greater resilience, maintaining higher output quality even with more variants, with scores of $0.49 \rightarrow 0.49 \rightarrow 0.46 \rightarrow 0.46$. For more detailed results, including ROUGE-2, ROUGE-L, and ROUGE-Lsum—which follow a similar trend—refer to Table 4 in Appendix A.

6. Related Work

Reducing the overhead of the host-to-device link is crucial for efficiently serving MoE models in constrained environments and efficient expert scheduling. Therefore, model compression techniques such as pruning, and quantization (Lin et al., 2023; Huang et al., 2024; Eliseev & Mazur, 2023; Imani et al., 2024; Dettmers & Zettlemoyer, 2023; Xiao et al., 2023) have been utilized extensively (Liu et al., 2024) to speed up the inference process.

Model Pruning Expert pruning in MoE models aims to reduce the number of parameters while maintaining model accuracy. This process is typically categorized into struc-

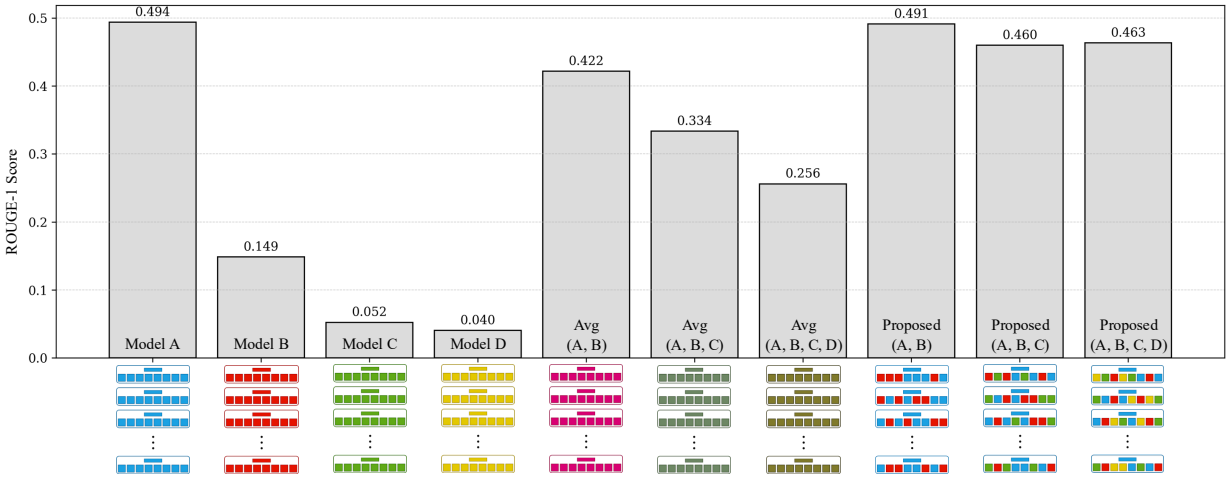

Figure 5. ROUGE-1 scores (higher is better) on SAMSum dataset for individual models and their combinations, comparing the proposed approach against the averaging baseline. Model layouts are illustrated beneath each bar. In our approach, a portion of the expert parameters (squares) and all non-expert parameters (rectangles) are retained from the model specifically trained for the benchmark (Model A, shown in blue). In contrast, the averaging approach modifies all parameters by averaging across the merged models.

tured and unstructured pruning. Structured pruning methods focus on reducing the number of experts, with some approaches directly removing unimportant experts and others merging them. Importance of an expert can be defined using different metrics. For example, approaches proposed in (Chen et al., 2022; Muzio et al., 2024; Park, 2024) identify the unimportant experts based on frequency and scarcity and prune non-essential experts while fine-tuning important ones for target tasks. Moreover, similar to our approach, authors in (Chowdhury et al., 2024; Yang et al., 2024a) propose to prune the experts of a fine-tuned model that have lower $L2$ distances from the base model. However, these approaches focus on performance of a single model and do not consider a multi-model multi-task system.

Multi-Task Model Merging Model merging integrates multiple models into a single model by performing weight interpolation at the parameter level, serving as an efficient alternative (Singh & Jaggi, 2020; Yang et al., 2024b; 2023). Existing model merging methods include weighted merging (Matena & Raffel, 2022; Wortsman et al., 2022; Jin et al., 2022), which assigns varying importance to different models, and subspace merging (Du et al., 2024; Suvizi et al., 2025; Yadav et al., 2023), which eliminates unimportant neurons to reduce task interference. However, these approaches use static merging strategies, limiting adaptability. Moreover, techniques such as activation matching, and permutation invariance help minimize discrepancies. Although these approaches demonstrate improved performance on considerably smaller models, their applicability for larger MoE architectures remains unexplored. Additionally, unlike previous methods that rely solely on static merging, our approach enhances both efficiency and effectiveness by integrating dynamic runtime reconfigurability with static merging.

7. Conclusion

In this paper, we presented an efficient serving system designed to address the challenges of deploying multiple fine-tuned MoE-based LLMs in single-GPU resource-constrained settings. By leveraging similarity-based consolidation, our approach reduced the overall memory footprint by sharing similar experts across models. Additionally, runtime partial reconfiguration allowed non-expert layers to be dynamically swapped, maintaining competitive output quality. Our evaluation, conducted on a server with a single NVIDIA A100 GPU, demonstrated the system's ability to significantly reduce turnaround time compared to conventional virtualization approaches like NVIDIA MIG, resulting in comparable throughput to single-model serving with only a minimal increase in TTFT. The results also highlighted that our approach maintained high output quality, as evidenced by strong performance across various benchmarks.

Acknowledgements

This work was supported by the National Science Foundation under Grant No. 2038682. We thank Pedram Akbarian and Rasool Sharifi for the helpful discussions.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are potential societal consequences of our work; however, given the experimental nature of the paper, we are unable to anticipate any negative consequences here.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Austin, J., Douglas, S., Frostig, R., Levskaya, A., Chen, C., Vikram, S., Lebron, F., Choy, P., Ramasesh, V., Webson, A., and Pope, R. How to scale your model. 2025. Retrieved from https://jax-ml.github.io/scaling-book/.

Chen, T., Huang, S., Xie, Y., Jiao, B., Jiang, D., Zhou, H., Li, J., and Wei, F. Task-specific expert pruning for sparse mixture-of-experts. *arXiv preprint arXiv:2206.00277*, 2022.

Choquette, J., Gandhi, W., Giroux, O., Stam, N., and Krashinsky, R. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35, 2021.

Chowdhury, M. N. R., Wang, M., Maghraoui, K. E., Wang, N., Chen, P.-Y., and Carothers, C. A provably effective method for pruning experts in fine-tuned sparse mixture-of-experts. *arXiv preprint arXiv:2405.16646*, 2024.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

Dettmers, T. and Zettlemoyer, L. The case for 4-bit precision: k-bit inference scaling laws. In *International Conference on Machine Learning*, pp. 7750–7774. PMLR, 2023.

Du, G., Lee, J., Li, J., Jiang, R., Guo, Y., Yu, S., Liu, H., Goh, S. K., Tang, H.-K., He, D., et al. Parameter competition balancing for model merging. *arXiv preprint arXiv:2410.02396*, 2024.

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

El-Araby, E., Gonzalez, I., and El-Ghazawi, T. Exploiting partial runtime reconfiguration for high-performance reconfigurable computing. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 1(4):1–23, 2009.

Eliseev, A. and Mazur, D. Fast inference of mixture-of-experts language models with offloading. *arXiv preprint arXiv:2312.17238*, 2023.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Gliwa, B., Mochol, I., Biesek, M., and Wawer, A. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*. Association for Computational Linguistics, 2019. doi: 10.18653/v1/d19-5409. URL http://dx.doi.org/10.18653/v1/D19-5409.

He, J., Vero, M., Krasnopolska, G., and Vechev, M. Instruction tuning for secure code generation. *arXiv preprint arXiv:2402.09497*, 2024.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Huang, M., Narayana, V. K., Simmler, H., Serres, O., and El-Ghazawi, T. Reconfiguration and communication-aware task scheduling for high-performance reconfigurable computing. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 3(4):1–25, 2010.

Huang, W., Liu, Y., Qin, H., Li, Y., Zhang, S., Liu, X., Magno, M., and Qi, X. Billm: Pushing the limit of post-training quantization for llms. *arXiv preprint arXiv:2402.04291*, 2024.

Imani, H., Amirany, A., and El-Ghazawi, T. Mixture of experts with mixture of precisions for tuning quality of service. *arXiv preprint arXiv:2407.14417*, 2024.

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and Wilson, A. G. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

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

Jin, X., Ren, X., Preotiuc-Pietro, D., and Cheng, P. Dataless knowledge fusion by merging weights of language models. *arXiv preprint arXiv:2212.09849*, 2022.

Jordan, M. I. and Jacobs, R. A. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6(2): 181–214, 1994. doi: 10.1162/neco.1994.6.2.181.

Kamahori, K., Gu, Y., Zhu, K., and Kasikci, B. Fiddler: Cpu-gpu orchestration for fast inference of mixture-of-experts models. *arXiv preprint arXiv:2402.07033*, 2024.

Kim, Y. J., Awan, A. A., Muzio, A., Salinas, A. F. C., Lu, L., Hendy, A., Rajbhandari, S., He, Y., and Awadalla, H. H. Scalable and efficient moe training for multitask multilingual models. *arXiv preprint arXiv:2109.10465*, 2021.

Li, T., Narayana, V. K., El-Araby, E., and El-Ghazawi, T. Gpu resource sharing and virtualization on high performance computing systems. In *2011 International Conference on Parallel Processing*, pp. 733–742. IEEE, 2011.

Lin, J., Yang, A., Bai, J., Zhou, C., Jiang, L., Jia, X., Wang, A., Zhang, J., Li, Y., Lin, W., et al. M6-10t: A sharing-delinking paradigm for efficient multi-trillion parameter pretraining. *arXiv preprint arXiv:2110.03888*, 2021a.

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Lin, S., Hilton, J., and Evans, O. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*, 2021b.

Liu, J., Tang, P., Wang, W., Ren, Y., Hou, X., Heng, P.-A., Guo, M., and Li, C. A survey on inference optimization techniques for mixture of experts models. *arXiv preprint arXiv:2412.14219*, 2024.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. The penn treebank: Annotating predicate argument structure. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994.

Matena, M. S. and Raffel, C. A. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 35:17703–17716, 2022.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Muzio, A., Sun, A., and He, C. Seer-moe: Sparse expert efficiency through regularization for mixture-of-experts. *arXiv preprint arXiv:2404.05089*, 2024.

Park, S. Learning more generalized experts by merging experts in mixture-of-experts. *arXiv preprint arXiv:2405.11530*, 2024.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21 (140):1–67, 2020.

Rajbhandari, S., Li, C., Yao, Z., Zhang, M., Aminabadi, R. Y., Awan, A. A., Rasley, J., and He, Y. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pp. 18332–18346. PMLR, 2022.

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

Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., et al. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems*, 31, 2018.

Shen, L., Tang, A., Yang, E., Guo, G., Luo, Y., Zhang, L., Cao, X., Du, B., and Tao, D. Efficient and effective weight-ensembling mixture of experts for multi-task model merging. *arXiv preprint arXiv:2410.21804*, 2024.

Singh, S. P. and Jaggi, M. Model fusion via optimal transport. *Advances in Neural Information Processing Systems*, 33:22045–22055, 2020.

Suvizi, A., Subramaniam, S., Lan, T., and Venkataramani, G. Exploring in-memory accelerators and fpgas for latency-sensitive dnn inference on edge servers. In *2024 IEEE Cloud Summit*, pp. 1–6, 2024. doi: 10.1109/Cloud-Summit61220.2024.00007.

Suvizi, A., Ahmadi, R., Zamani, M. S., and Meybodi, M. R. A parallel computational approach for energy-efficient hydraulic analysis of water distribution networks using learning automata. *Sustainable Computing: Informatics and Systems*, pp. 101135, 2025. ISSN 2210-5379.

doi: https://doi.org/10.1016/j.suscom.2025.101135. URL https://www.sciencedirect.com/science/article/pii/S2210537925000563.

Wortsman, M., Ilharco, G., Gadre, S. Y., Roelofs, R., Gontijo-Lopes, R., Morcos, A. S., Namkoong, H., Farhadi, A., Carmon, Y., Kornblith, S., et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pp. 23965–23998. PMLR, 2022.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023.

Xue, L., Fu, Y., Lu, Z., Mai, L., and Marina, M. Moe-infinity: Activation-aware expert offloading for efficient moe serving. *arXiv preprint arXiv:2401.14361*, 2024.

Yadav, P., Tam, D., Choshen, L., Raffel, C., and Bansal, M. Resolving interference when merging models. *arXiv preprint arXiv:2306.01708*, 1, 2023.

Yang, C., Sui, Y., Xiao, J., Huang, L., Gong, Y., Duan, Y., Jia, W., Yin, M., Cheng, Y., and Yuan, B. Moe-i^2: Compressing mixture of experts models through inter-expert pruning and intra-expert low-rank decomposition. *arXiv preprint arXiv:2411.01016*, 2024a.

Yang, E., Wang, Z., Shen, L., Liu, S., Guo, G., Wang, X., and Tao, D. Adamerging: Adaptive model merging for multi-task learning. *arXiv preprint arXiv:2310.02575*, 2023.

Yang, E., Shen, L., Wang, Z., Guo, G., Wang, X., Cao, X., Zhang, J., and Tao, D. Surgeryv2: Bridging the gap between model merging and multi-task learning with deep representation surgery. *arXiv preprint arXiv:2410.14389*, 2024b.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36: 46595–46623, 2023.

Zuo, S., Liu, X., Jiao, J., Kim, Y. J., Hassan, H., Zhang, R., Zhao, T., and Gao, J. Taming sparsely activated transformer with stochastic experts. *arXiv preprint arXiv:2110.04260*, 2021.

A. Complete Results of SAMSum Dataset

Table 4 presents the detailed results for the summarization task on SAMSum dataset. As demonstrated, all metrics—including ROUGE-2, ROUGE-L, and ROUGE-Lsum—follow a similar trend, highlighting the greater resilience of our approach as the number of served models increases.

Table 4. ROUGE scores (higher is better) on SAMSum dataset for individual models and their combinations.

Model	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-Lsum
Model A	0.493	0.250	0.409	0.409
Model B	0.148	0.026	0.128	0.128
Model C	0.052	0.004	0.046	0.046
Model D	0.040	0.009	0.036	0.036
Avg (A, B)	0.422	0.201	0.344	0.344
Avg (A, B, C)	0.333	0.132	0.277	0.277
Avg (A, B, C, D)	0.256	0.079	0.209	0.209
Proposed (A, B)	0.491	0.245	0.407	0.406
Proposed (A, B, C)	0.460	0.229	0.377	0.377
Proposed (A, B, C, D)	0.463	0.229	0.377	0.377