A Model Merging Method

Jisheng Fang
asdfqwer2015@163.com

Hao Mo *
sunshineinautumn@163.com

Qiang Gao
gq15035177217@gmail.com

Abstract

At the NeurIPS 2024 LLM-Merging competition, we successfully developed a simple and effective model merging approach that generates a versatile, generalist model, applicable to a wide range of scenarios. Specifically, this method is easy to implement, prevents significant conflicts among component models, and improves overall model accuracy to some extent. Additionally, it is memory-efficient, allowing for the combination of multiple foundational models, further optimizing resource utilization.

1 Introduction

Current methods for model merging primarily include Parameter Averaging, MoE-based merging, Model Stacking, Model Zipping, and Model Routing [1]. The NeurIPS 2024 LLM-Merging competition aims to create functionally comprehensive models by merging various domain-specific models efficiently. Our approach utilizes a combination of model routing and model stacking, which has yielded significant accuracy improvements. Furthermore, this method is highly adaptable and can be effectively extended to multiple domains, offering both scalability and flexibility for a wide range of applications.

2 Method

Our approach is primarily reflected in three aspects: model selection, model merging, and staged responses.

2.1 Model Selection

For model selection, we approached it from two directions: base models and fine-tuned models. First, we identified a base model by initially screening five options that met the competition's criteria. Using the capabilities of large models listed on Hugging Face's Open LLM Leaderboard2, we narrowed it down to three high-accuracy base models: Meta-Llama-3-8B-Instruct, Phi-3-mini-4k-instruct, and Phi-3-small-8k-instruct. We used the lm-evaluation-harness tool to test each base model's performance on several public datasets. Results showed that Meta-Llama-3-8B-Instruct and Phi-3-small-8k-instruct had their own advantages, displaying similar patterns across our internal test sets. Consequently, we selected Meta-Llama-3-8B-Instruct and Phi-3-small-8k-instruct as our final base models.

For fine-tuned models, we divided task types by knowledge area, including history, medicine, mathematics, physics, and coding. Using datasets integrated within lm-evaluation-harness, combined with our proprietary datasets, we assessed each fine-tuned model's GPU memory usage and accuracy gains over the base models. Ultimately, we selected medical and coding models as our final fine-tuned models, balancing memory efficiency with performance enhancements.

*Contact author. Email: abc20152024@163.com

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

2.2 Model Merging

We observed that using parameter averaging method to merge models often reduces performance on tasks where the base models excel. Therefore, we chose a model combination approach similar to model routing, which typically requires substantial memory. To address this, we merged models with identical architectures by compressing differences, preserving each base model's task-specific strengths while minimizing inference overhead. In our trials, this approach only increased inference costs by around 10%, maintaining efficiency without sacrificing performance.

2.2.1 Weights Merging

In our solution, we combine several fully fine-tuned Llama3 8B models. We assume that the weights of a base model and the fine-tuned models can each be represented as a common component plus a differentiated component. The common component can be obtained through parameter averaging, while the unique components are compressed separately. Besides retaining their own embeddings, lm_head layer weights, and biases (if any) for each model, other layers are merged and compressed. For faster compression, we employed randomized SVD (rSVD) instead of standard SVD. When the compression ratio is set to 0, the result is equivalent to using multiple independent models; at 100%, it corresponds to parameter averaging. Interestingly, after this combined compression, each branch model showed an improvement in general task performance, though due to time constraints, we didn't delve into this further.

The pseudocode for weight merging and compression for a specific layer is as follows:

Algorithm: Weight compression for a layer in models

Input:

$W = \{W_1, W_2, \ldots, W_N\}$

$compress\_rate$

Output:

$scales = \{scale_1, scale_2, \ldots, scale_N\}$

W_{avg}

$compressed\_diff = \{U_1, U_2, \ldots, U_N, V_1, V_2, \ldots, V_N\}$

1. For each weight matrix $W_i \in W$:

$scale_i = \|W_i\|$

$\widehat{W}_i = \frac{W_i}{scale_i}$

Normalize weight matrix W_i.

2. $W_{avg} = \frac{1}{N} \sum \widehat{W}_i$

3. For each normalized weight matrix $\widehat{W}_i$:

$U_i, V_i = RSVD(\widehat{W}_i - w_{avg}, compress\_rate)$

4. Return $scales, w_{avg}, compressed\_diff$

Figure 1: Weight compression for a layer in models

The pseudocode for inference in a specific layer of a branch model is as follows:

Algorithm: Inference for Compressed Model Layer

Input:

x

$bias$ # Uncompressed bias

$scales = \{scale_1, scale_2, \ldots, scale_N\}$

W_{avg}

$compressed\_diff = \{U_1, U_2, \ldots, U_N, V_1, V_2, \ldots, V_N\}$

Output:

$y = \{y_1, y_2, \ldots, y_N\}$

1. $y_i = linear(x, w_{avg}) + linear(linear(x, V_i), U_i) * scale_i$

2. If bias is not null:

   $y_i \mathrel{+}= bias_i$

3. Return y # Return the final output.

Figure 2: Inference for Compressed Model Layer

In our approach, we set the compression rate to 95%. We loaded one independent Phi3small model and three fully fine-tuned Llama3 8B models, compressed using this method, into 48GB of VRAM. In practical applications, this method allows for compressing and combining more models as needed.

2.2.2 Router

In our approach, we utilized four branches: one standalone Phi-3-small model and three Llama-3-8B models merged through compression. To determine the appropriate branch for answering each input, we route the question based on relevance. To avoid potential generalization issues and minimize processing time, we did not train a dedicated router. Instead, we designed a set of representative samples for each branch. Therefore, we designed 17 samples that were carefully crafted to cover each branch's focus area and do not overlap with the competition data.

We utilized a decoder-only model and carefully designed our routing method to maximize the router's ability to distinguish between questions from different domains. For each sample or input question, we extract the embedding from the segment "{input}.\n Let's think about what task these questions belong to. These questions belong to the field of" specifically from the phrase "These questions belong to the field of". Due to the attention mechanism, this embedding effectively captures information from the preceding input. Since both input questions and samples rely on this structure, they achieve high alignment. In our experiments, this method significantly improved the router's ability to differentiate between domains compared to extracting the embedding from the input alone.

For classification, we calculate the cosine similarity between input questions and the sample embeddings, with pre-extracted embeddings for each sample to reduce inference time during evaluation.

2.3 Staged Responses

In the competition, we found that achieving high-quality responses requires two key factors: accuracy and clarity. While Llama-3's responses were generally more structured compared to Phi-3-small, Llama-3's accuracy was often weaker. For Phi-3-small (instruct), although prompts can be adjusted to encourage more concise responses, using system-level prompts to enforce brevity sometimes interfered with the model's reasoning, leading to incorrect answers. Conversely, if we aimed for comprehensive reasoning in response to complex questions, the answers tended to be overly detailed (e.g., an SQL question might include additional, unwanted explanations). Moreover, overly complex prompts sometimes unintentionally reduced overall accuracy. This led us to adopt a two-stage response approach.

Unfortunately, due to a time zone miscalculation of the competition deadline, our two-staged solution was only partially submitted, with the full version missing from official scoring. However, in our experiments, the complete two-stage method showed a 6-point improvement in accuracy, compared to only a 2-point gain from early-stage submissions. This section describes the full version of the two-stage approach as a technical demonstration.

In our method, tasks that leveraged Phi-3-small's strengths, like reasoning, were initially answered by Phi-3-small. Llama-3 then refined these answers for clarity and structure through prompt engineering. For some scenarios, we introduced an alternative two-stage response format: Stage 1 generates a guided, heuristic-based answer, while Stage 2 formats the output. Since Stage 2 avoids triggering chain-of-thought processes, it generally produces concise answers without significantly increasing inference time. Notably, Stages 1 and 2 can be handled by the same or different models, depending on the task.

References

[1] Tam D. & Li M. & Yadav P. & et al. (2024) Llm merging: Building llms efficiently through merging. *NeurIPS 2024 Competition Track.*