Medical Minds Unite: Amplifying Performance by Merging Specialized LLMs

Aditya Chaloo achaloo@umass.edu

Kedarnath Chimmad kchimmad@umass.edu

Mayank Bumb mbumb@umass.edu

Ojas Raundale oraundale@umass.edu

Siddarth Suresh siddarthsure@umass.edu

1 Problem statement

The field of medicine is witnessing a surge in the application of Large Language Models (LLMs). Trained on vast medical data and literature, these AI systems showcase impressive potential, particularly in facilitating question-answering. LLMs can form the basis of applications, such as virtual medical assistants having the capability of answering complex patient queries, simplifying medical concepts, and providing preliminary diagnoses based on symptoms. While LLMs do possess these capabilities, most of them require to be trained on a huge corpus of data and are proprietary, examples of such models include GPT and BARD. To mitigate these issues with current LLMs, we are proposing two techniques, Model Ensembling and a Mixture of Experts. These two methods have proven to improve the performance of weaker open-source models, empowering them to perform at the level of strong proprietary models.

2 What you proposed vs. what you accomplished

One of the changes we made to our project post-proposal was in the selection of models. Initially, we planned to use three models: BioGPT, BioMistral, and Llama-2. However, we ended up using Mistral exclusively. We created two experts by fine-tuning Mistral on two different datasets to develop a domain expert and a task expert. We utilized Unsloth[1] for quantization and LoRA fine-tuning. The sections below explain this new approach in detail. This adjustment was made after consulting with Prof. Mohit.

- ~~Collect and preprocess dataset~~
- ~~Build and train baseline models like data domain expert and task expert on the collected dataset and examine its performance~~
- ~~Build an MoE and Ensemble-based model for solving MCQ task~~
- *Build a MoE and Ensemble-based models for solving QA task*: We failed to do so because we were not able to achieve better accuracies MCQ task and also we lacked domain knowledge to assess the quality of answers
- ~~Model Analysis and Miscellaneous experiments~~

[1] https://github.com/unslothai/unsloth

3 Related work

Mixture of experts is a technique where specialized models, called ”experts,” are activated based on a gating network that routes input tokens to the appropriate experts. This gating network has trainable parameters updated along with the rest of the model. Inspired by ”The Sparsely-Gated Mixture of Experts Layer” paper by (Shazeer et al., 2017), MoE employs conditional computing, activating only relevant parts of the network based on the input. The MoE layer consists of multiple feed-forward neural networks, with the gating network selecting a sparse subset of experts for each input. This sparsity, achieved through Noisy Top K Gating, ensures computational efficiency by activating only a few experts while others remain inactive.

Mixtral of experts (Jiang et al., 2024). It is one of the most popular research papers. Mistral.ai has managed to create an LLM with 12B parameters with much better performance than Llama 70B and ChatGPT 3.5 using a concept called Mixture of Experts (Shazeer et al., 2017) (1). The paper takes 8 Mistral (Jiang et al., 2023) decoder models. At each layer, there’s a router that passes the

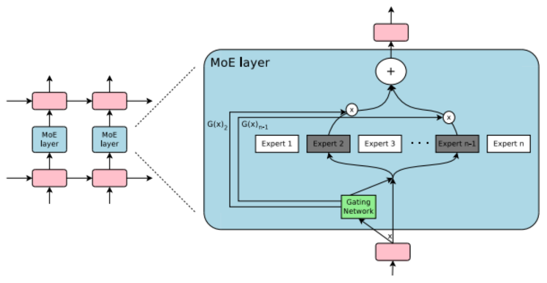


Figure 1: Mixture of Experts (Image from (Shazeer et al., 2017)).

inputs to 2 out of the 8 experts which are decided through a gating network. The gating function involves taking the softmax over the Top-K logits of a linear layer. The outputs from the feedforward network of the 2 experts are combined using the weighted sum method. There are 2 mixtral of experts models. One is built using a pretraining model and the other one uses instruction tuning.

Biomistral (Labrak et al., 2024). It's an open-source LLM tailored for the biomedical domain that uses Mistral-Instruct as its foundation model and has been further pre-trained on the PubMed Central dataset. It also has been trained on 7 languages marking the first large-scale multilingual evaluation of LLMs in the medical domain. The model architecture of biomistral inherits the standard transformer architecture from Mistral, including features such as Grouped-Query Attention (Ainslie et al., 2023), Sliding Window Attention (Beltagy et al., 2020) and Rolling Buffer Cache. Biomistral has 2 quantized models Activation-aware Weight Quantization (AWQ) (Lin et al., 2023) and BitsandBytes (BnB) which uses QLora (Dettmers et al., 2023) technique. The researchers have also instruction-tuned the model on MCQA (Pal et al., 2022) dataset for evaluation that outperforms a lot of same domain models. We have quantized this model to compare the results with other models.

LLM Blender (Jiang et al., 2023)., a popular LLM Ensembler has architectured a new pairwise ranking system. LLM Blender uses multiple open-source LLMs, ranks them by comparing 2 LLMs against each other, and gives rank by using MaxLogits (a technique that computes the superiority of output compared to other outputs). Out of all the outputs, top K outputs along with the input is fed into a T5 (they used Flan-T5-XL (Chung et al., 2022)) to get a superior output. We are taking the motivation from this paper to fuse the outputs of individual models using a Transformer layer/architecture.

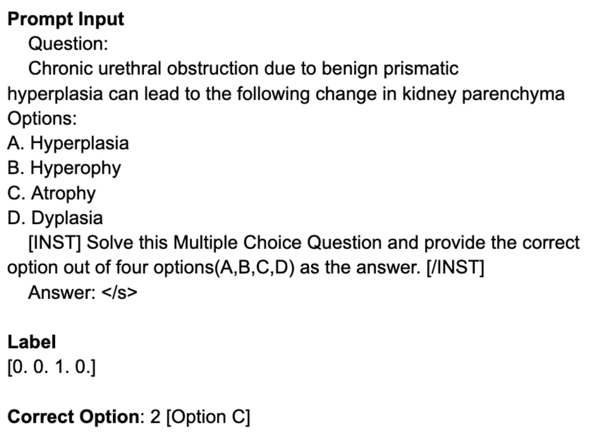
Apart from using a transformer layer/architecture to fuse the model outputs, there are other ways. Traditionally, Ensembling of models were done using Bagging, Boosting, Averaging, Stacking, and many other techniques (Haralabopoulos et al., 2020) (Tasci et al., 2021). Late Fusion is a technique in which multiple models' last layer logits are used to get the final result through ensembling. This is a popular ensembling technique in the field of computer vision and deep learning (Ma et al., 2024).

4 Dataset

Our models were trained and evaluated using several task-specific datasets. To train the domain expert in the Mixture of Expert, we used an instruction-based prompt dataset comprising MedQuad (Ben Abacha and Demner-Fushman, 2019) and The Medical Meadow Wikidoc dataset. MedQuad includes approximately 47,000 question-answer pairs covering a wide spectrum of medical subjects. The Medical Meadow Wikidoc dataset contains question-answer pairs sourced from WikiDoc, an online platform where medical professionals collaboratively contribute and share contemporary medical knowledge. This dataset focuses on medical topics, aligning well with this expert's role. The ARC (AI2 Reasoning Challenge) dataset, containing 7,787 multiple-choice science questions at a grade-school level, was used to train the MCQ Expert for the Mixture of Experts and Ensembling model. Finally, MedMCQA (Pal et al., 2022), a multi-choice question-answering dataset with approximately 194,000 MCQs related to healthcare across 21 topics, was used for two purposes: fine-tuning and evaluating our model's performance in the medical domain. Due to limitations of compute resources, we used approximately 10% of the MedMCQA training data for training and approximately 50% of the validation data for model validation in the medical domain. Due to the unavailability of labels for the test dataset, we use the validation dataset to evaluate our model.

4.1 Data preprocessing

For preprocessing the data, specifically for the MCQ-based dataset, we collated the questions and multiple-choice options into a single prompt. This

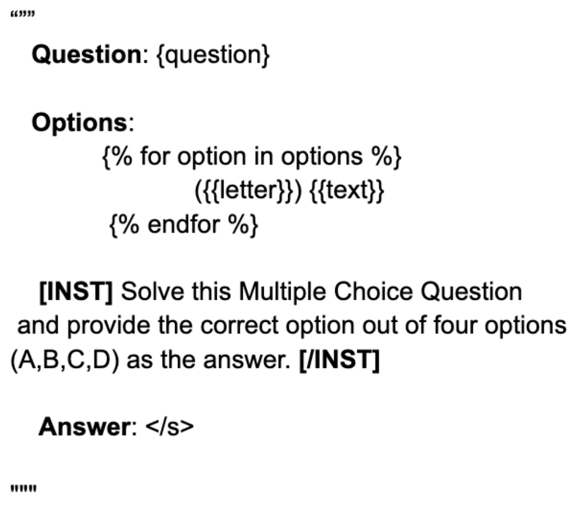


Prompt Input
Question:
Chronic urethral obstruction due to benign prismatic hyperplasia can lead to the following change in kidney parenchyma
Options:
A. Hyperplasia
B. Hyperophy
C. Atrophy
D. Dyplasia
[INST] Solve this Multiple Choice Question and provide the correct option out of four options(A,B,C,D) as the answer. [/INST]
Answer: </s>

Label
[0. 0. 1. 0.]

Correct Option: 2 [Option C]

Figure 2: Example of a Multiple Choice Question from the MedMCQA dataset.



```
“””
  Question: {question}

  Options:
          {% for option in options %}
                  ({{letter}}) {{text}}
          {% endfor %}

   [INST] Solve this Multiple Choice Question
  and provide the correct option out of four options
 (A,B,C,D) as the answer. [/INST]

   Answer: </s>

"""
```

Figure 3: Format of the Prompt used for training on MCQ tasks (AI2arc and MedMCQA)

created an instruction-based prompt dataset used to fine-tune the models. This approach aligns with the way the expert models and end users will interact with the model, providing a question and options and expecting the correct answer. For MedMCQA, the answers to the multiple-choice questions were one-hot encoded, enabling effective classification of the options. To train our model, we initially tested it on a small subset of MedMCQA (1% of the training data) and then gradually scaled up to 10% of the MedMCQA training data.

5 Baselines

The current state of the art on MedMCQA dataset is Med-PaLM 2 (Singhal et al., 2023) model with an accuracy of 0.723. Getting a high accuracy on MedMCQA is quite tough as it is based on

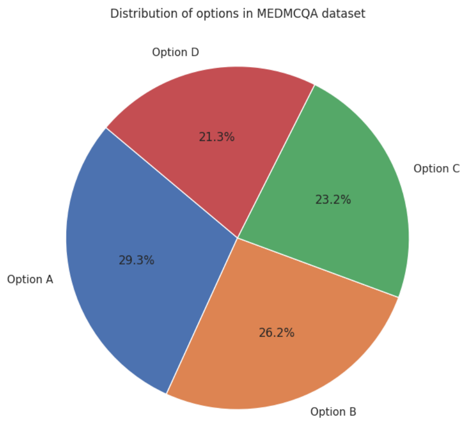


Figure 4: Distribution of Options in the MedMCQA dataset.

Questions from Medical School Entrance Examinations. An accurate model will need to be able to have high medical knowledge and also be able to accurately select the correct answer from the options that may be close by to each other.

Our Mixture of Experts and Ensembling Models use Mistral-based models. So we decided to keep the baselines based on Mistral too. We used the 20k-sized MedMCQA train dataset for all baselines.

The first baseline we tried was to directly see how a vanilla mistral performs on the MedMCQA dataset. It was trained as a next-word prediction model using our prompt dataset to predict the correct option. While the model did train as we saw the loss decreasing 5, the model failed to produce options (A, B, C, or D) as the next word.

So we quickly moved on by adding a linear layer with 4 output dimensions on top of the base model. For the base models we tried 1.) Vanilla Mistral 7B, 2.) Domain Specific Expert (6.1), 3.) Task Specific Expert (6.1), 4.) Biomistral 7B. All of these models were their respective 4-bit quantized versions. We used LORA with $r = 16, \alpha = 8, dropout = 0.05$. This resulted in a total trainable parameters of 0.5% of the 7B parameter base models. Table 2 depicts the accuracies of our baseline.

6 Our approach

Our project aims to enhance the performance of large language models on biomedical domain-specific tasks of multiple choice question answering through a systematic combination of models using various model ensembling and a mixture

DataSet	Used for	Description	Train	Validation	Test
MedMCQA	Objective Dataset - Training for Baselines, MoE and Ensembling Models	Medical MCQ dataset	20k (Mini) & 180k (Mega)	2k	2k
Medical Mistral Instruct	Training Domain Model (Base Model 1)	Instruction supervised finetuning Medical Dataset	26.4k	-	-
Ai2Arc	Training Task Model (Base Model 2)	Grade-school level, multiple-choice science question	7,787	-	-

Table 1: Datasets Used for Baselines, Base Models, and the Objective Training

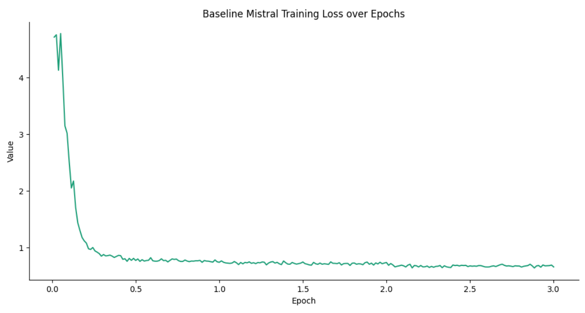


Figure 5: Baseline Next Word Mistral Loss over Epochs

Baseline Models	Accuracy
Baseline Next Word Mistral	Couldn't predict option
Baseline Mistral Classifier	26.03
Baseline Domain Expert Model Classifier	25.93
Baseline Task Expert Model Classifier	24.69
Baseline Biomistral Model Classifier	31.65
SOTA - (Biomistral 7B SLERP)	45.7

Table 2: Baseline Performance

of expert techniques. Leveraging pre-trained and quantized Mistral, we explored various combinations to identify the most effective technique to combine models.

6.1 Expert Models Preparation

To create our experts, we fine-tuned two mistral models, one on the task-specific dataset (ai2arc from Allen ai) and the other on the domain-specific dataset (combination of medQuad and Medical Meadow Datasets). For finetuning the base Mistral models, we used the library unsloth (2024) library.

Instruction Specific Expert
As the target MCQA dataset of MedMCQA had only 4 options per question, we preprocessed the ai2arc dataset to remove options from questions having more than 4 options, keeping the correct option along with 3 other options.
Following this, prompts were generated of the following format: 3
The base model that was fine-tuned to create the expert was unsloth/mistral-7b-instruct-v0.2-bnb-4bit provided by unsloth (2024). We chose this model because it is quantized to 4-bit. We chose an instruction-tuned model so that the model is pretrained on generic world knowledge and tasks. We then used LoRA for PEFT with LoRA alpha=16 with no dropout and no bias. We also used gradient checkpointing for very long contexts.
The resulting model had 32 decoder layers with each layer consisting of multi-head self-attention, LoRA embedding, rotary embedding, dropout, layer RMS norm, and SiLu activation.
We then used SFTTrainer provided by huggingface (2024) to supervised fine tune the quantized base mistral model with max sequence length set to 256. The datatype for the tensors was BF16 as BF16 is designed to strike a better balance between precision and range. We used the standard learning rate of 2e-4 with AdamW 8bit optimizer. The weight decay was 0.01 and the learning rate scheduler was linear. The model was finetuned on the entire dataset for 3 epochs with 5 warmup steps and a batch size of 16.

For performing this fine-tuning, we referred to this colab notebook provided by unslothSFT (2024a) Loss plot given in figure 6

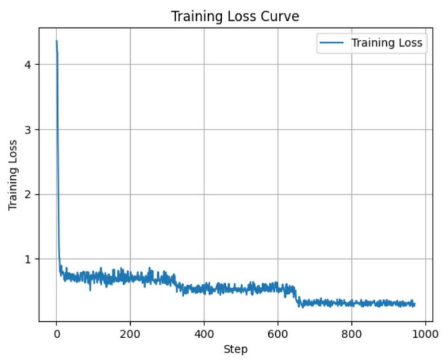


Figure 6: SFT Instruction Specific Expert

Domain Specific Expert As the 2nd expert, we used a domain specialized model. The base model that was fine-tuned to create the expert was unsloth/mistral-7b-instruct-v0.2-bnb-4bit provided by unsloth (2024). We chose this model because it is quantized to 4 bits. We chose an instruction-tuned model so that the model is pretrained on generic world knowledge and tasks. We then fine-tuned this model on domain knowledge using medical mistral instruct (2024). We ran this finetuning for 1 epoch with a dataset size of 26k examples.

The sample Prompt used for fine-tuning is Figure 7

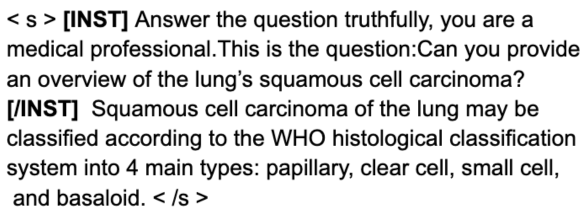

< s > **[INST]** Answer the question truthfully, you are a medical professional.This is the question:Can you provide an overview of the lung's squamous cell carcinoma? **[/INST]** Squamous cell carcinoma of the lung may be classified according to the WHO histological classification system into 4 main types: papillary, clear cell, small cell, and basaloid. < /s >

Figure 7: Sample Prompt used for fine tuning

You can see the SFT trainer parameters here unslothSFT (2024b)
The training loss curve can be seen in figure 8

6.2 Model Ensembling

To build out the ensembling models, we used the Mistral Domain and Mistral Task models that were fine-tuned on datasets as mentioned above. We tried 3 different ways of Ensembling techniques.

1. **Weighted Average** To combine the outputs of two base models using a fusion technique, we employed a weighted sum approach. Here's the process:

   The embeddings from the last layer of each base model, originally of size 32,000, were first reduced to a dimension of 4. We combined these reduced-dimensional embeddings using a weighted sum.

   The resulting combined embeddings were then passed through a linear layer to retain the 4-dimensional structure, which was subsequently fed into the final softmax layer. This method allowed us to effectively integrate the information from the two base models before making the final prediction.

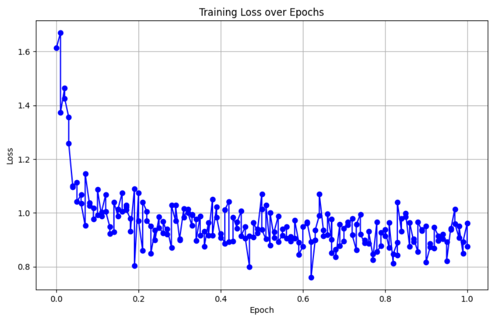


Figure 8: SFT Domain Expert

2. **Transformer Block**

   To combine the outputs of two base models using a fusion technique with a transformer block, we employed unmasked self-attention.

   The embeddings from the last layer of each base model, initially of size 32,000, were first reduced to a dimension of 4. We attempted to reduce the embeddings to intermediate sizes such as 16,000 and 4,000, but this resulted in the overall model size becoming too large to fit in GPU memory. Therefore, we opted for a reduction to 4 dimensions.

   After reducing the embeddings to 4 dimensions, we concatenated the outputs from both models. This concatenated 8-dimensional vector was then passed through a transformer block with unmasked self-attention. The use of unmasked self-attention in the transformer block was intended to enable a more comprehensive understanding of the combined model outputs.

   Finally, the output from the transformer block was fed into a linear layer, which reduced the dimension back to 4, suitable for the final softmax layer. This approach allowed us to leverage the transformer's capabilities to effectively integrate the information from the two base models before making the final prediction.

3. **Late Fusion**

   In our attempt to combine the outputs of two base models using the Late Fusion technique, we experimented with two different approaches.

   Initially, we concatenated the outputs of the last layer embeddings from the two base models and applied random sampling using a multinomial distribution. However, this approach resulted in high loss and low accuracy. We hypothesized that the randomness inherent in the multinomial distribution sampling made it difficult for the subsequent linear layer to effectively learn how to combine the embeddings.

   To address this issue, we replaced the multinomial distribution sampling with a direct application of a linear layer on top of the concatenated model outputs. Although we experimented with using multiple linear layers, this did not yield any significant increase in model accuracy even with a learning rate as low as 1e-6, the loss was too high. Moreover, improper initialization of these layers often led to the loss values becoming NaN occasionally.

   Consequently, we decided to use a single linear layer to predict the output after concatenating the model outputs. This approach provided a more stable and effective method for the linear layer to learn how to combine the embeddings from the two base models.

In our ensemble models, we consistently used a learning rate of 1e-4 and the Adam optimizer for gradient calculations. Automatic Mixed Precision (AMP) was employed to efficiently handle different precisions during training and inference. The loss values for all models ranged between 1 and 2. Attempts to reduce the loss below 1 using a higher learning rate resulted in noisy loss curves, so we maintained the learning rate at 1e-4 for more stable and acceptable loss values. All models were trained with a batch size of 32.

During the initial phases of training, we utilized a smaller dataset to understand the behavior of the expert models and refine the architecture design. At this stage, we used an L4 or T4 GPU in Google Colab, with batch sizes of 4 or 8, as GPU memory constraints limited the batch size. Once we were confident in the model architecture based on results from the small subset of training data, we scaled up the models to the full training set and transitioned to using an A100 GPU for final training.

6.3 Mixture of Experts (MoE)

1. **Routing Architectures**
   For the mixture of experts, we used the two finetuned experts and two different gating architectures.
   The gating network, also known as the router, determines which expert network receives the output for each token from previous layers. This gating layer is present between each layer of the expert so that it can decide which expert's next layer the previous layer's output needs to be sent to. So, as Mistral has 32 layers, there are 32 gating networks in the mixture of experts model.

   Linear Routing This is the basic MoE architecture where we create the model architecture which contains 2 experts, task-based expert, and domain-based expert. For each layer of this expert, we introduce a gating network which is a linear layer that takes output embeddings of each layer and then decides which of the 2 models should be selected for the next iterations. We then pass on the output embeddings of the previous layer to the chosen model and subsequently we get the final output embeddings which are then passed through a linear layer for the classification task.
   We first load the dataset. Then take the embeddings of the data points using tokenizer and max length as 128. Once we have the embeddings we create a train loader and do forward propagation through the model explained above. Since the custom gating network layers use float 32 and a few layers inside the model use float 16 and int8 since it is a quantized model, we use gradscaler which internally takes care of the multiplica-

tion of different datatype. Post that we evaluate the accuracy and store the max validation accuracy model. In addition to this, we also store layer-wise and epoch-wise count of which model did gating network choose.

Noisy Top K Routing We referred to Sooriyarachchi (2024) to implement this architecture. The noisy top k router consists of a linear layer that produces Gaussian noise and another linear layer that performs the routing. Essentially, to prevent all the tokens from being sent to the same set of 'favored' experts, it is helpful to add standard normal noise to the logits from the gating linear layer. This makes training more efficient. The router layer takes input of the form (batch, tokens, embedding size) and converts it to (batch, tokens, number of experts). We then determine the top-k experts to send the tokens to by creating a mask having -inf value for experts which are not in the top-k experts and keeping the values of the top-k experts intact. We then passed this through a softmax layer, eventually creating a sparse representation of the input to be sent to the top-k experts. In our model, we have only 2 experts so top-k is set as 1.
First, the token embeddings and positional embeddings are created for the input token IDs and summed together. Then for each layer of the expert, the generated embedding is sent through the gating layer which decides which tokens need to go to which top-k experts. An expert capacity is calculated (tokens per batch/num experts * capacity factor) to limit the number of tokens being sent to an expert so as to reduce the bias in the selection of experts. Here, the capacity factor is a hyperparameter.
The tokens are then sent to the expert layer whose output is multiplied by the gating weights calculated by the routing layer to give a weighted embedding. Finally, the output from this current layer of mixture of experts model is sent to the next layer. To mitigate the issue of NaN and INF gradients, we have also implemented layer normalization and skip connections in every layer of our MoE model.
The final output embeddings which are then passed through a linear layer for the classification task.

2. **Training**

   Noisy Top K Routing We convert the labels to one hot encoded tensor. To prevent issues with mixed precision training, we have used GradScaler and auto-cast from torch.cuda.amp library PyTorch (2024). The optimizer we used is Adam with a learning rate of 1e-3, weight decay of 1e-3, and epsilon value of 1e-4. The loss function we have used is cross-entropy loss as this is a multiclass classification problem. We ran the model for 8 epochs with a batch size of 8 and a max token length of 300. We chose 300 as this represented around 50% of our inputs and maximized our usage of available compute RAM. Training and validation losses and accuracies can be seen in figures 24 25 15.
   Due to the noise being added explicitly, the training loss curve is not smooth as different expert models are learning differently. The accuracy as well fluctuates during training as different models are picked during training.

   Linear Routing 3 main experiments were done

   (a) **Overfitting (micro dataset)**
   First we ran this model on the micro dataset which consisted of 2k training examples. We ran this on V100-16GB with 8 batch size, max length 128 and 5 epochs. The model overfitted successfully with a training accuracy of 80% after 4 epochs.

   (b) **Mini dataset**
   We then ran this model on mini dataset which consisted of 20k training examples and 2k validation examples. We ran this on RTX8000-48GB with batch size of 16, learning rate 2e-4 (1e-3 after some point of time gave loss. as nan), 10 epochs. The model's maximum validation accuracy was 32%. Training and validation losses and accuracies can be seen in figures. 9 10

   (c) **Mega dataset**
   We then ran this model on Mega dataset which consisted of 180K training examples. We ran this on RTX8000-48GB with batch size of 16, learning rate 2e-4,

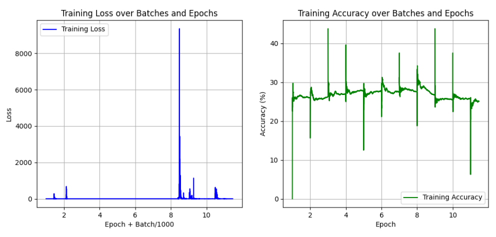


Figure 9: Training Loss and Accuracy on Mini

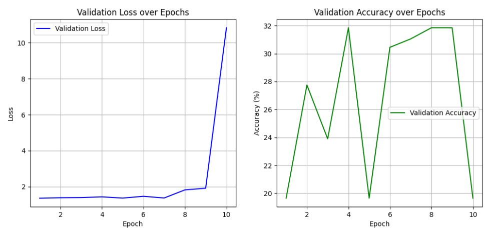


Figure 10: Validation Loss and Accuracy on Mini

1 epoch. The model's validation accuracy was 28%. Training losses and accuracies can be seen in figures. 11

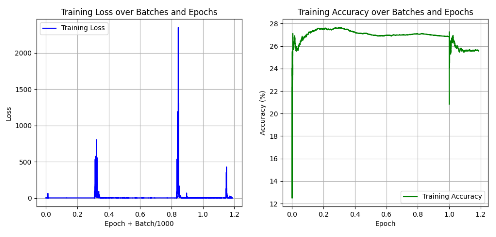


Figure 11: Training Loss and Accuracy on Mega

Environment Details All our models were run in google Colab Pro tier and Unity server. We ran all our experiments on four gpu's: a100, L4, RTX8000-48GB, and T4 on Colab and unity. A100 is the fastest GPU with highest RAM.

Issues faced and hacks to resolve them We faced several issues due to the large model size, large dataset size, and minimal compute available.

1. One of the issues we faced was with the availability of compute. We couldn't use the entire training data always as the availability of GPU was for very short times and had to reduce our datasize. To resolve this, we purchased Colab Pro and ran the models overnight when a100 was available.

2. Due to large model size and minimal compute, we had to reduce the dataset size, context length and batch size being fed into the model.

3. Another issue we faced was while training custom model architecture using experts finetuned using unsloth library. The datatypes of the unsloth models and our custom layers during back propagation did not match and we had to make changes in unsloth library to meet our requirements[2]. Unity does not allow this so we made the changes in Colab and restarted the session without deleting the runtime to ensure the changes remained intact.

4. We lost most of our compute in pro tier in the start due to Colab notebooks running even after the execution is complete. So we initially had to manually keep a check on it. We found out a code snippet we can run in the end to Disconnect the runtime.

7 Results

Ensembling From 3, we see that the Late Fusion Ensemble Technique achieved the highest accuracy, followed by the Transformer Ensemble. The Weighted Average method performed worse, likely because the baseline task expert model and domain expert model (2) had similar accuracies, leading the model to assign similar weights to them. The relative superior performance of the other two models is due to their more sophisticated techniques for merging model expertises. The simplicity of the Weighted Average model's architecture pales in comparison to the advanced methods used by the Transformer and Late Fusion-based ensemble techniques.

Model Name	Accuracy
Mixture of Experts(linear)	29.09
Mixture of Experts(TopK)	28.88
Weighted Avg Ensemble	24.55
Late Fusion Ensemble	29.3
Transformer Ensemble	28.74

Table 3: Accuracies of different models.

Mixture of Experts - Noisy TopK Routing

Mixture of Experts - Linear

Following are the figures post-training Mixture of Experts with a linear Gating network. Figures

[2]Refer to fast_lora.py in our code

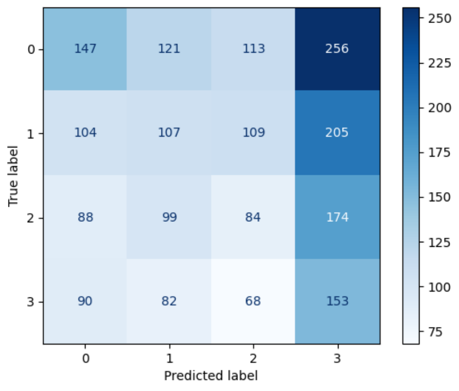


Figure 12: Weighted Avg Ensemble Confusion Matrix

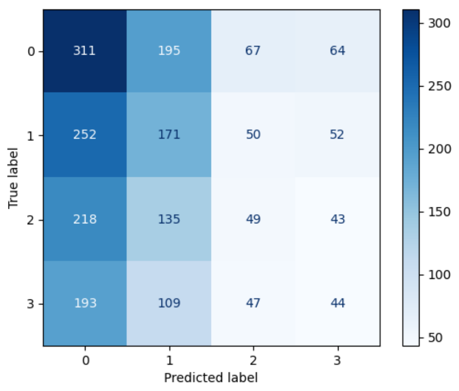


Figure 13: Transformer Ensemble Confusion Matrix

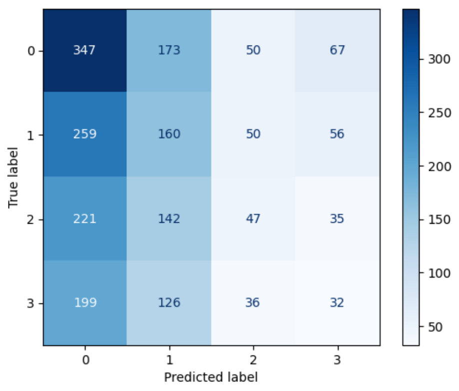


Figure 14: Late Fusion Ensemble Confusion Matrix

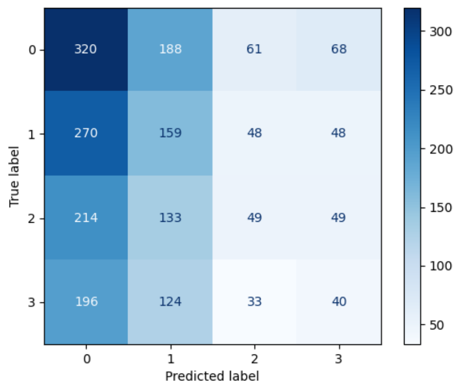


Figure 15: Noisy Top K Inference Confusion Matrix

16 and 11 shows the confusion matrix of the best-trained model on Mini and Mega datasets. 0,1,2,3 denote the options and what the model chooses vs what are the true labels for questions.

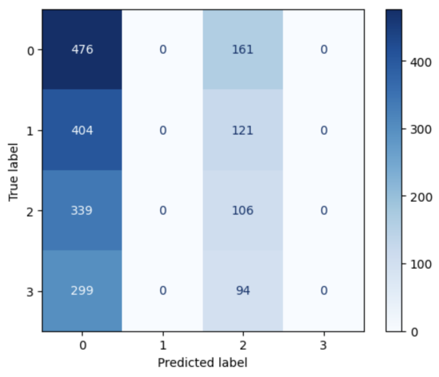


Figure 16: Confusion Matrix for Mini

We then move on to layer-wise analysis. Figure 26 shows which model is getting preference across epochs that is number of times that particular model got chosen by the gating network. Then we did a final analysis that is which layer is getting chosen and how many number of times at the end of the training for both Mini and Mega dataset. Figure 27 and 28 both represent the same.

Figure 29, 30, 31 and 32 represent across epoch analysis across all the epochs.

8 Error analysis

In Table 4, we have annotated the failure data points of our model. After analyzing our dataset, we discovered that our reduced training dataset completely missed Psychiatry subject questions and contained very few questions on Orthopaedics, Anaesthesia, Skin, and a few other subjects. Consequently, our models did not learn

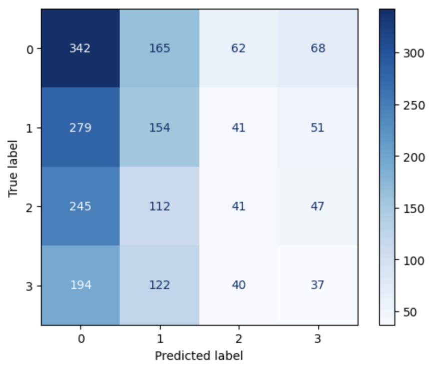


Figure 17: Confusion Matrix for Mega

adequately, leading to incorrect predictions. Before proceeding with training, we checked the dataset distribution with respect to the length of the prompt (including questions and options), and it matched the full-size dataset. However, we overlooked the subject distribution.

1. Interlinking of layers - A lot of times in a given epoch all the validation data chooses the same model for a particular layer. One of the inferences from the layer-wise gating network analysis is possibly the 2 experts haven't interacted with each other properly. Each layer of the 2 experts has learnt something different and it is possible that output embeddings of 1 expert isn't fully able to understand or interact properly with the expected input embeddings of another model. This problem can possibly be solved using 2 methods. We can introduce a factor in the loss function that penalizes the model for choosing the same model for a particular layer every time. Another approach can be to use a transformer-based gating network which can transform the embeddings of one model's output to better suit the input of another model.

2. Weak Prediction - We are getting high predictions for only 2 classes. This can be because of a few possible reasons like dataset wasn't large enough, training epochs were small for a 7b parameter model.

9 Contributions of group members

Each member of the group worked on their individual tasks as listed below. Additionally, everyone was involved in error resolution and model fine-tuning for the other parts.

- Aditya: Data collection, Data processing, and Baseline fine-tuning of models.
- Kedarnath: Worked on building an Ensembling pipeline, Trained all the Ensembling models, Error Analysis.
- Mayank: Mixture of Experts (Linear Gating network), Domain-based Finetuning, Error analysis
- Ojas: Data preprocessing, Baseline fine-tuning of models.
- Siddarth: Mixture of Experts pipeline (Noisy Top-K, Task Specific Mistral Fine Tuning), Error Analysis

10 Conclusion

Training of large models like mistral with 7B parameters take a lot of time. Even after using Unsloth which reduces the trainable model parameters size to 40M is huge given the amount of dataset and computing we had. Both the methods Model Ensembling and Mixture of Experts performed better than the baseline models. Experimenting with LLMs is cost-intensive and takes a lot of time. So we couldn't solve all the problems but tried our best to get the best results. To our surprise, Mixture of Experts model was choosing task specific model more number of times than domain specific model. Perhaps we need to make our gating network more robust.

Future Work

1. Run our models on the entire dataset for a greater number of epochs to check if training is the problem.
2. Update the Gating network to a more complex architecture like a transformer or multilayer perceptron.
3. Implementing Fisher Matrix-based merging using newer approaches that require lesser memory utilization.
4. For the Baseline approach, the linear layer of 4-outputs didn't get trained properly. It could be because of low amounts of data and less computation. The BioMistral paper used the next word generation model and prompted inputs to get the output as the next word. We

Question	Correct Option	Predicted Option	Subject Type
Question: Z tracking technique is used in? Options: A. Administering long-acting antipsychotic B. Lithium monitoring C. Carbamazepine monitoring D. Nicotine patch	D	A	Psychiatry
Question: A patient believes he is the most important person in the world than anyone so his neighbors and family is trying to harm him as they are jealous of him. His wife says otherwise and says he behaves like this recently only before he was working as a school-teacher peacefully and brought to OPD. He is suffering from: Options: A. Delusion of grandiosity B. Delusion of persecution C. Delusion of grandiosity and persecution D. Delusion of grandiosity, persecution and reference	A	D	Psychiatry
Question: Provision of WHO mental action gap are all, except: Options: A. Human rights B. Communication regarding care and career C. Screening family members D. Social support	C	A	Psychiatry
Question: Fish tail deformity in a child is seen after injury to? Options: A. Distal Tibia B. Distal Femur C. Distal humerus D. Distal Radius	C	D	Orthopaedics
Question: Highest concentration of oxygen is delivered through? Options: A. Nasal cannula B. Venturi mask C. Bag and mask D. Mask with reservoir	C	D	Anaesthesia

Table 4: Questions where our models failed

wanted to properly train our baseline model using the next word generation task to see if performance increases.

5. The Long-term goal will be to increase the number of experts which are specialized on different aspects of medical domains like Dental, Surgery, etc.

11 AI Disclosure

- Did you use any AI assistance to complete this proposal? If so, please also specify what AI you used.
  - Yes, we used ChatGPT. We mainly used it for latex codes to prepare this file.

If you answered yes to the above question, please complete the following as well:

- If you used a large language model to assist you, please paste *all* of the prompts that you used below. Add a separate bullet for each prompt, and specify which part of the proposal is associated with which prompt.
  - how to create a table in latex
  - my report is 2 columns based, I want the result to fit in one column, modify ur code
  - getting the error: environment subfigure not defined in latex. Which package am I missing
- **Free response:** For each section or paragraph for which you used assistance, describe your overall experience with the AI. How helpful was it? Did it just directly give you a good output, or did you have to edit it? Was its output ever obviously wrong or irrelevant? Did you use it to generate new text, check your own ideas, or rewrite text?
  - For a few prompts the gpt didn't solve the error that we faced. It gave us the same code error we made.

– Overall it solved most errors we got in latex and resolved our doubts.

References

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. (2023). GQA: Training generalized multi-query transformer models from multi-head checkpoints.

Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The Long-Document transformer.

Ben Abacha, A. and Demner-Fushman, D. (2019). A question-entailment approach to question answering. *BMC Bioinform.*, 20(1):511:1–511:23.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., Webson, A., Gu, S. S., Dai, Z., Suzgun, M., Chen, X., Chowdhery, A., Castro-Ros, A., Pellat, M., Robinson, K., Valter, D., Narang, S., Mishra, G., Yu, A., Zhao, V., Huang, Y., Dai, A., Yu, H., Petrov, S., Chi, E. H., Dean, J., Devlin, J., Roberts, A., Zhou, D., Le, Q. V., and Wei, J. (2022). Scaling instruction-finetuned language models.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023). QLoRA: Efficient finetuning of quantized LLMs.

Haralabopoulos, G., Anagnostopoulos, I., and McAuley, D. (2020). Ensemble deep learning for multilabel binary classification of user-generated content. *Algorithms*, 13(4).

huggingface (2024). huggingface. https://huggingface.co/docs/trl/main/en/sft_trainer. Accessed: 2024-05-17.

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., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M.-A., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T. L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. (2024). Mixtral of experts.

Jiang, D., Ren, X., and Lin, B. Y. (2023). LLM-Blender: Ensembling large language models with pairwise ranking and generative fusion.

Labrak, Y., Bazoge, A., Morin, E., Gourraud, P.-A., Rouvier, M., and Dufour, R. (2024). BioMistral: A collection of open-source pretrained large language models for medical domains.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., Gan, C., and Han, S. (2023). AWQ: Activation-aware weight quantization for LLM compression and acceleration.

Ma, Y., Peri, N., Wei, S., Hua, W., Ramanan, D., Li, Y., and Kong, S. (2024). Long-tailed 3d detection via 2d late fusion.

medical mistral instruct (2024). medical mistral instruct. https://huggingface.co/datasets/Shekswess/medical_mistral_instruct_dataset. Accessed: 2024-05-17.

Pal, A., Umapathi, L. K., and Sankarasubbu, M. (2022). MedMCQA : A large-scale multi-subject multi-choice dataset for medical domain question answering.

PyTorch (2024). Pytorch. https://pytorch.org/docs/stable/index.html. Accessed: 2024-05-17.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). Outrageously large neural networks: The Sparsely-Gated mixture-of-experts layer.

Singhal, K., Tu, T., Gottweis, J., Sayres, R., Wulczyn, E., Hou, L., Clark, K., Pfohl, S., Cole-Lewis, H., Neal, D., Schaekermann, M., Wang, A., Amin, M., Lachgar, S., Mansfield, P., Prakash, S., Green, B., Dominowska, E., Arcas, B. A. y., Tomasev, N., Liu, Y., Wong, R., Semturs, C., Mahdavi, S. S., Barral, J., Webster, D., Corrado, G. S., Matias, Y., Azizi, S., Karthikesalingam, A., and Natarajan, V. (2023). Towards expert-level medical question answering with large language models.

Sooriyarachchi, A. (2024). make moe from scratch. https://huggingface.co/blog/AviSoori1x/makemoe-from-scratch. Accessed: 2024-05-17.

Tasci, E., Uluturk, C., and Ugur, A. (2021). A voting-based ensemble deep learning method focusing on image augmentation and preprocessing variations for tuberculosis detection. *Neural Computing and Applications*, 33(22):15541–15555.

unsloth (2024). unsloth. https://github.com/unslothai/unsloth. Accessed: 2024-05-17.

unslothSFT (2024a). Sft. https://colab.research.google.com/drive/1Dyauq4kTZoLewQ1cApceUQVNcnnNTzg_?usp=sharing. Accessed: 2024-05-17.

unslothSFT (2024b). Sft. https://drive.google.com/file/d/1bKkXxsO2SFUYfXeRTixJdXjIPmFFfN8S/view?usp=sharing. Accessed: 2024-05-17.

12 Appendix

12.1 Additional Figures

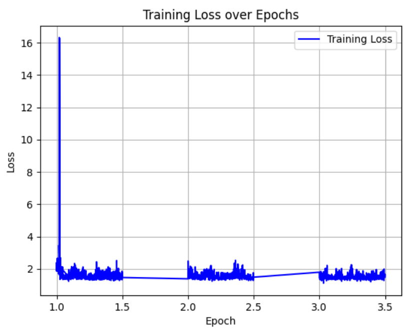


Figure 18: Weighted Avg Ensemble Training Loss Curve

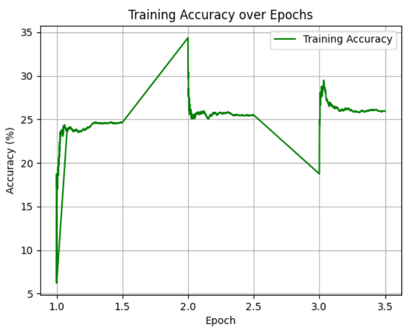


Figure 19: Weighted Avg Ensemble Training Accuracy Curve

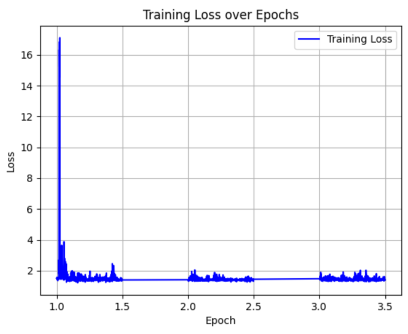


Figure 20: Transformer Ensemble Training Loss Curve

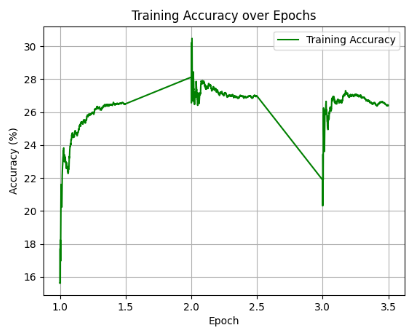


Figure 21: Transformer Ensemble Training Accuracy Curve

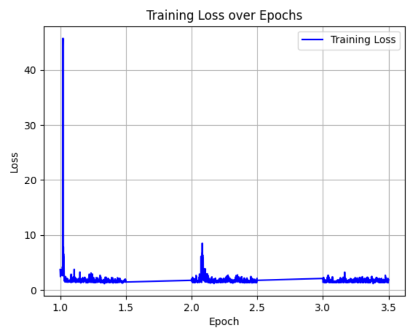


Figure 22: Late Fusion Ensemble Training Loss Curve

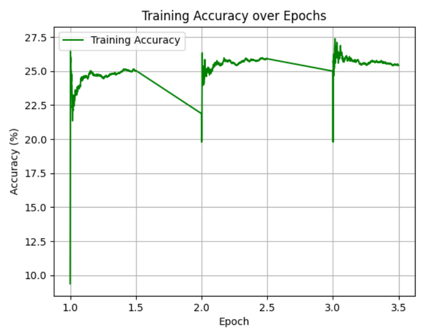


Figure 23: Late Fusion Ensemble Training Accuracy Curve

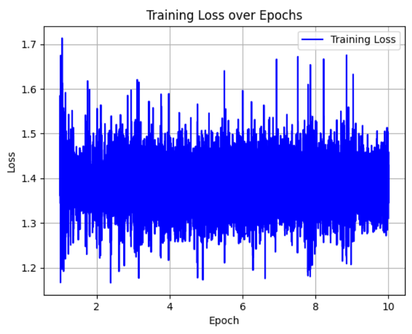


Figure 24: Noisy Top K Training Loss Curve

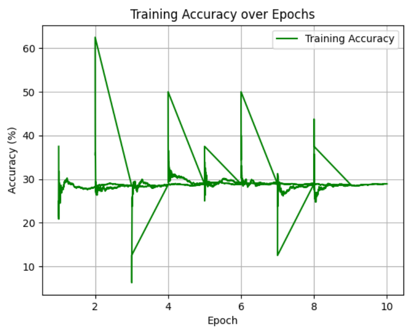


Figure 25: Noisy Top K Training Accuracy Curve

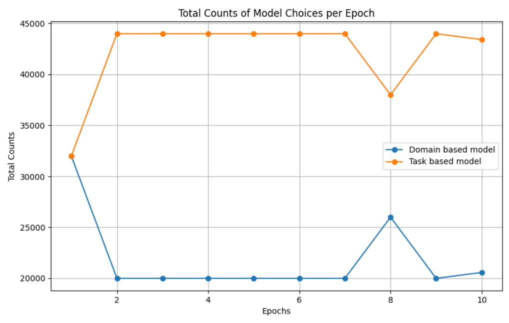


Figure 26: Model preference across epochs

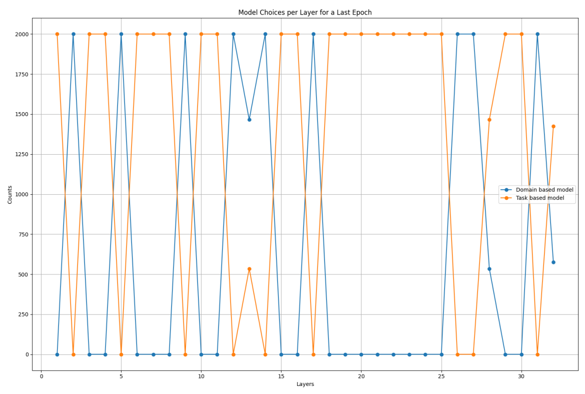


Figure 27: Final model preference for Mini

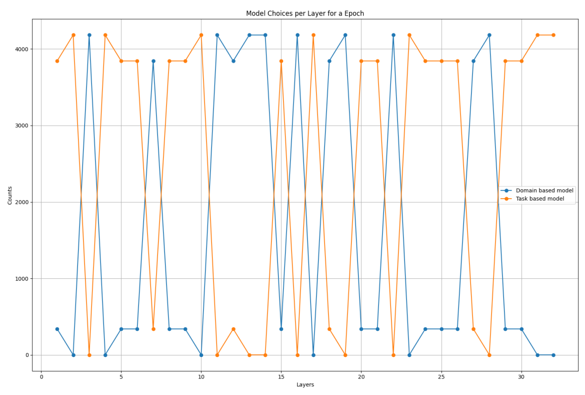


Figure 28: Final model preference for Mega

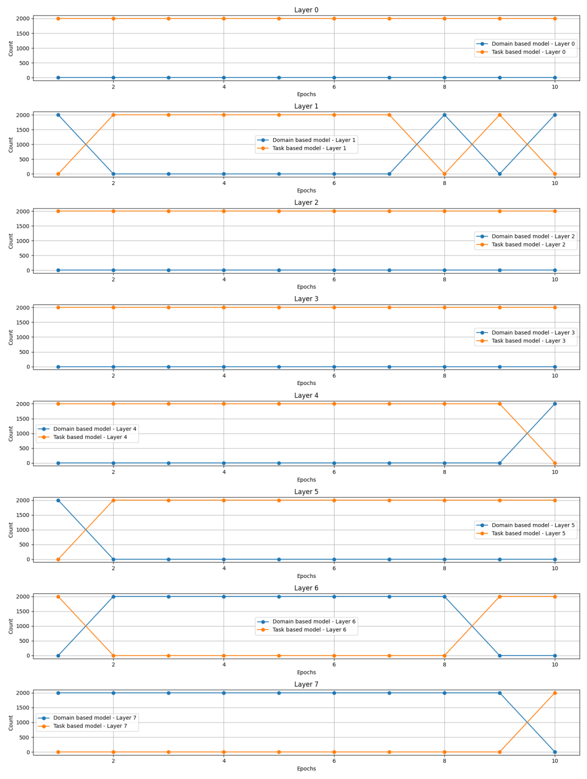


Figure 29: Layer wise preference across epochs part 1

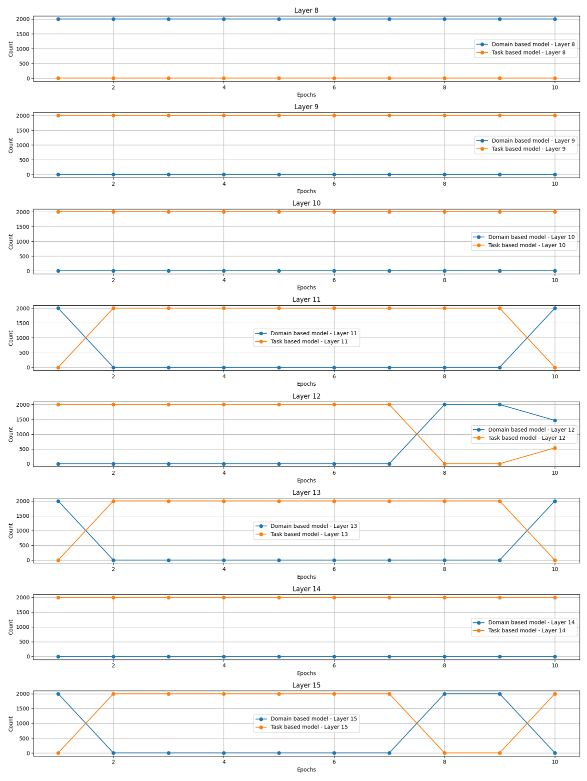


Figure 30: Layer-wise preference across epochs part 2

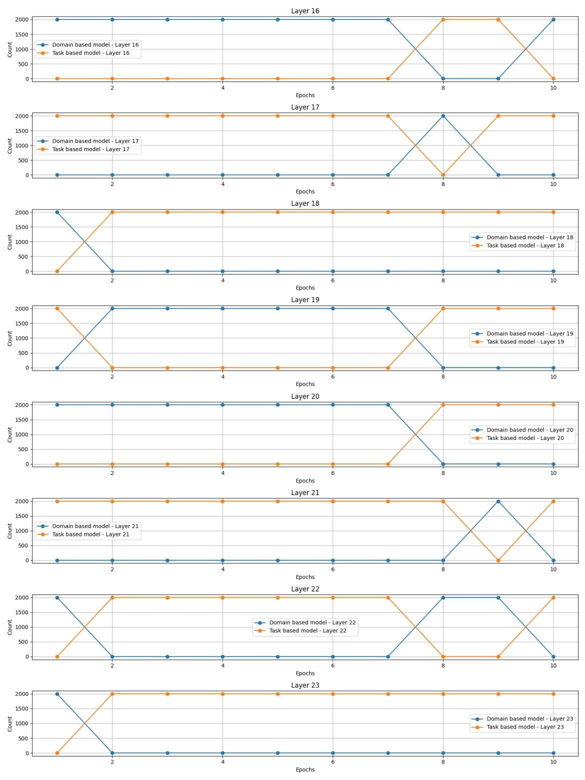


Figure 31: Layer-wise preference across epochs part 3

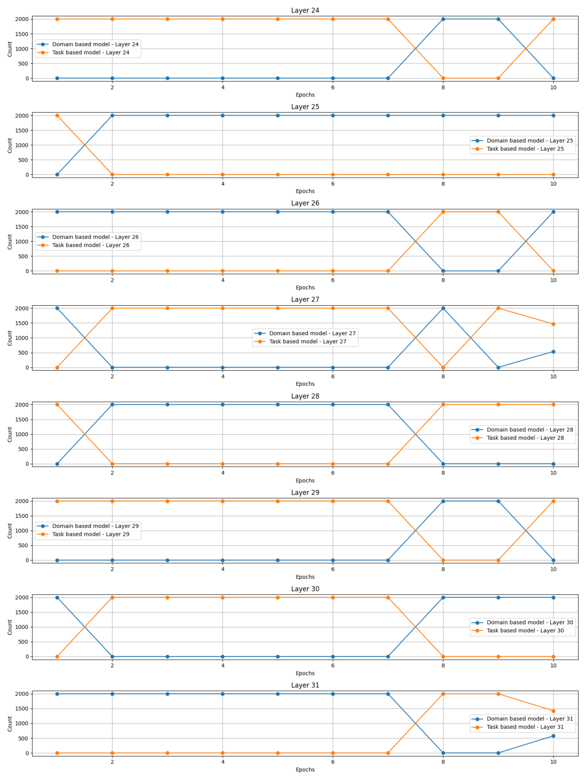


Figure 32: Layer-wise preference across epochs part 4