REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

ACCELERATED LARGE LANGUAGE MODEL PRE-TRAINING

21011036 — TALHA ÇELİK
20011015 — HÜSEYİN SAİD ARICI

SENIOR PROJECT

Advisor
Res. Asst. Himmet Toprak KESGİN

June, 2025

ACKNOWLEDGEMENTS

We would like to thank Mr. Himmet Toprak Kesgin, who always answered our questions and guided us in the meetings we held throughout the project.

TALHA ÇELİK
HÜSEYİN SAİD ARICI

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

LLM	Large Language Models
NLP	Natural Language Processing
SaLEM	Salient Layers Editing Model
PAFT	Parallel Alignment Fine-Tuning
SFT	Supervised Fine-Tuning
PA	Preference Alignment
stderr	Standard Error

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Accelerated Large Language Model Pre-Training

TALHA ÇELİK
HÜSEYİN SAİD ARICI

Department of Computer Engineering
Senior Project

Advisor: Res. Asst. Himmet Toprak KESGİN

The popularization of large language models has led to major advances in the field of natural language processing. The high processing power and long training times required when working with large data sets have become a problem with these advances. In this project, strategies for splitting the dataset and training the model with different layers were studied to overcome these difficulties brought by vanilla training. The Llama-3.2-1B model was generally used within the scope of the project and experimental studies were conducted on this model. In the training process, three different large datasets, namely Cosmpedia, Gsm8k Train and MetaMathQA, were evaluated. The applied strategies include splitting the dataset into four separate sub-sections and producing sub-models by training each sub-section on specific layers of the main model. The advantage of being able to train these sub-models in parallel aims to significantly shorten the training time. A scenario where the weights of the models were randomly initialized was also tested. The developments of the models were monitored with two validation sets. The success of the models was measured with performance metrics such as ARC, Hellaswag and GSM8K. The main goal of the project is to reduce the training time with this split dataset and layer-based training approach while preserving the success increase of the model to a large extent.

Keywords: Large Language Models, Parallel Training, Training Time Optimization, Layer-Based Fine-tuning, Hellaswag, Gsm8k, Datasets, Natural Language Processing Model Layers

ÖZET

Hızlandırılmş Büyük Dil Modeli Ön Eğitimi

TALHA ÇELİK
HÜSEYİN SAİD ARICI

Bilgisayar Mühendisliği Bölümü
Bitirme Projesi

Danışman: Arş. Gör. Himmet Toprak KESGİN

Büyük dil modellerinin popülerleşmesi, doğal dil işleme alanında büyük ilerlemelere yol açmıştır. Büyük veri setleri ile çalışırken ihtiyaç duyulan yüksek işlem gücü ve uzun eğitim süreleri bu ilerlemelerle birlikte bir sorun haline gelmiştir. Bu projede vanilya eğitimin getirdiği bu zorlukların üstesinden gelmek için, veri kümesini bölme ve modelin farklı katmanlarıyla eğitme stratejileri üzerine çalışılmıştır. Proje kapsamında genel olarak Llama-3.2-1B modeli kullanılmış ve deneysel çalışmalar bu model üzerinde yapılmıştır. Eğitim sürecinde, Cosmpedia, Gsm8k Train ve MetaMathQA olmak üzere üç farklı büyük veri kümesi değerlendirilmiştir. Uygulanan stratejiler, veri kümesini dört ayrı alt bölüme ayırmayı ve her bir alt bölümü, ana modelin belirli katmanlarını eğiterek alt modeller üretmeyi içermektedir. Bu alt modellerin paralel olarak eğitilebilme avantajı, eğitim süresini önemli ölçüde kısaltmayı hedeflemektedir. Modellerinin ağırlıklarının rastgele olarak başlatıldığı bir senaryo da test edilmiştir. Oluşturulan iki adet validasyon seti ile modellerin gelişimleri izlenmiştir. Modellerin başarıları ARC, Hellaswag ve GSM8K gibi performans metrikler ile ölçülmüştür. Projenin temel hedefi, bu bölünmüş veri kümesi ve katman bazlı eğitim yaklaşımıyla eğitim süresini azaltırken, modelin başarı artışını da büyük ölçüde koruyabilmektir.

Anahtar Kelimeler: Büyük Dil Modelleri, Paralel Eğitim, Eğitim Süresi Optimizasyonu, Katman Bazlı Fine-tuning, Hellaswag, Gsm8k, Veri Kümeleri, Doğal Dil İşleme Modeli Katmanları

1
INTRODUCTION

Large language models (LLM), which have pioneered many innovations in natural language processing (NLP) in recent years, cause great losses in terms of training time because they consist of billions of parameters. For this reason, new approaches are being sought in the training of LLMs.

In this project report, the studies carried out to accelerate the pre-training in order to save time in the pre-training phase will be examined. Similar studies will be analyzed and the system established for the project will be explained. The results will be analyzed.

1.1 Purpose

This project aims to use a layer-based random update approach to accelerate and parallelize the pretraining process of LLM. By updating different layers in different training processes, the final model is aimed to perform close to or better than the model that trains the entire dataset and all layers.

1.2 Preliminary Review

There are many studies that are being conducted to accelerate LLM pre-training. One of these is GrowLength, a collaborative effort between researchers from the Agricultural and Mechanical College of Texas, Amazon, and Rice University. GrowLength basically starts training with a low number of tokens and aims to continue training with slightly higher tokens at every opportunity, thus gaining speed in the initial part of the training.[1]

One of the studies conducted on a layer-based basis was included in an article published by Amazon Science. In this study, which took a layer-based approach, the model was updated by selecting the target layers instead of updating it completely.

Training time was saved. SaLEM (Salient Layers Editing Model) Method was used which adopted layer based approach. The SaLEM method is evaluated on six datasets with LLMs fine-tuned for NLP tasks. As a result, it was seen that training the model layer by layer gained speed instead of training the entire model from scratch.[2]

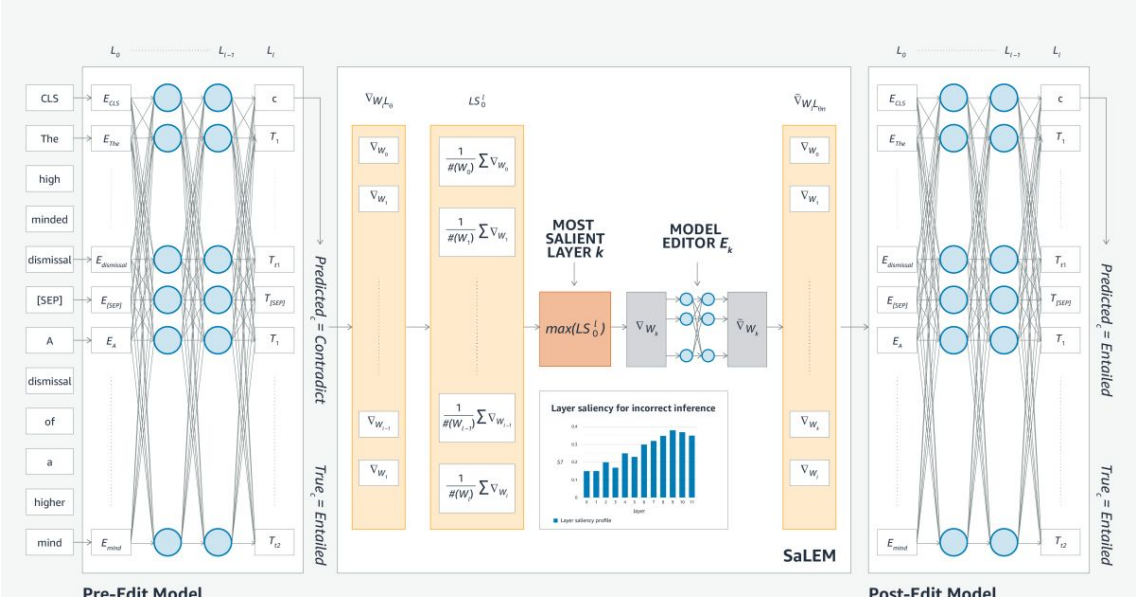

Figure 1.1 saLEM in Action[2]

2
LITERATURE ANALYSIS

The number of studies on layer-based random update method is quite low. Studies directly related to this subject are not yet at the level to be published at academic level. There are similar layer-focused studies.The first study to be mentioned in this report is the Small Model Initialization study conducted by a group of researchers from Apple, which aims to accelerate pre-training by layer-focused. Learning is based on expanding from small to large. HyperCloning is an initialization method that directly transfers the trained weights of a small model to a larger model. It is a method that perfectly fits the approach of going from small to large.The HyperCloning method does not change the training cycle, is simple to implement, and significantly reduces the training cost, accelerating the development of large language models.[3]

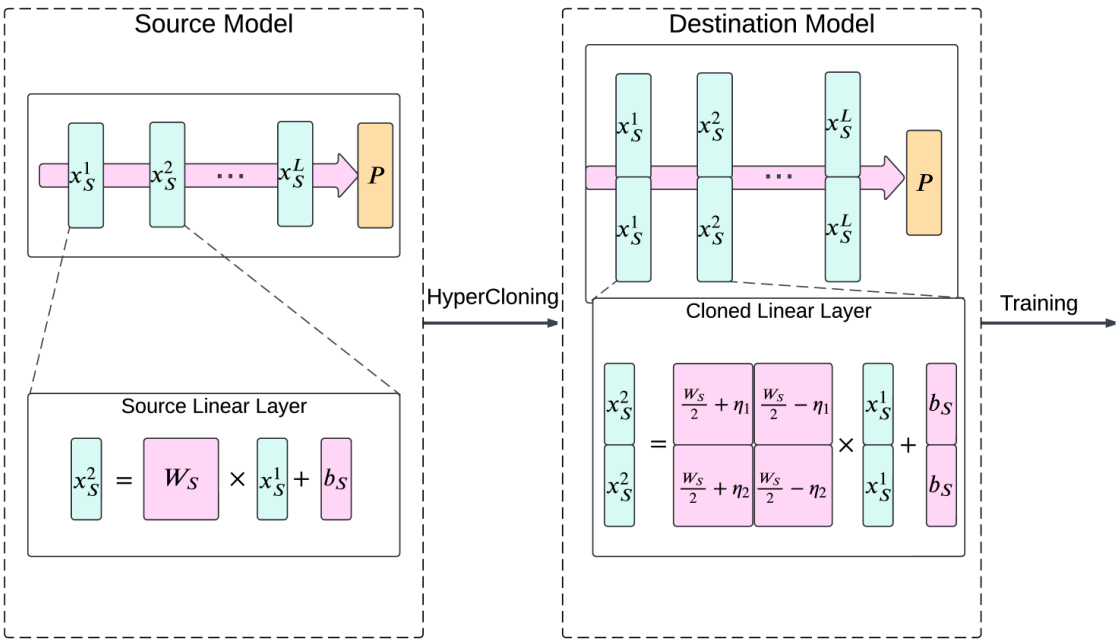

Figure 2.1 Illustration of HyperCloning

The study by Dumitru et al. proposes a new way to reduce the size of Large Language Models. This method changes how much it prunes each layer - it addresses problems

with pruning strategies that always cut a fixed amount.

Instead of cutting a fixed percentage, their method uses a Layer Redundancy score - this score shows how much information a layer adds. It compares the input and output of each layer using cosine similarity. Layers that show high redundancy do not change the input much. The method prunes these layers more - it preserves layers that add more information. That keeps the model working well.

The researchers tested this dynamic pruning on Llama3-8B in addition to Mistral-7B models. The tests show that this method works better than other techniques, such as SliceGPT besides ShortGPT.[4]

There are many studies on the chunking method, which is frequently used in layer separation. One of these is the ChunkRAG method, which is a study by Ishneet Sukhvinder Singh et al. The ChunkRAG method was developed to reduce the common issues of including irrelevant information and producing incorrect answers in knowledge-based text generation systems. In this approach, texts are divided into small, meaningful sections, and each part is individually assessed based on how closely it relates to the user's question. Unnecessary or repetitive content is removed to ensure only the most relevant information is used. The method combines both word-based and meaning-based comparisons to improve the accuracy of responses. Tests have shown that ChunkRAG provides more reliable and accurate answers across various topics, making it a useful solution for applications where factual correctness is essential.[5]

Yuan et al. propose ChunkFlow, a novel approach aimed at improving the efficiency of fine-tuning large language models on long-context tasks. Existing methods often prioritize the small fraction of long sequences in training datasets, leading to underutilization of GPU resources when processing predominantly short sequences. ChunkFlow restructures the input by segmenting long sequences and merging short ones into uniformly sized "chunks," thereby ensuring consistent memory usage and balanced computation. Moreover, the authors introduce a state-aware scheduling algorithm that significantly reduces memory overhead and mitigates pipeline inefficiencies, particularly pipeline bubbles, in distributed training setups. Experimental results demonstrate that ChunkFlow achieves up to a 4.53x speedup over Megatron-LM, highlighting its effectiveness and scalability for training LLMs on datasets with highly variable sequence lengths.[6]

Not only the chunking method but also the parallel training method is an element in this report. In this context, the first study discussed in the report is the study conducted by Salesforce and describing a method called Parallel Alignment Fine-Tuning (PAFT).

The PAFT system involves training the model on a specific task with the Supervised Fine-Tuning (SFT) process and performing the process of adjusting the model's outputs according to human preference as a result of Preference Alignment (PA) in parallel. [7]

Table 2.1 Comparison with state-of-the-art LLMs on Open LLM Leaderboard(Summarized)

LLM	ARC	HellaSwag	MMLU	TruthfulQA	AVERAGE
PAFT (Ein-70B)	0.7986	0.9149	0.7805	0.7514	0.8129
Llama-3-70B-Instruct	0.7142	0.8569	0.8006	0.6181	0.7788
DBRX-132B-Instruct	0.6783	0.8885	0.7372	0.6702	0.7447
Mistral-7B-Instruct-v0.2	0.6314	0.8488	0.6078	0.6826	0.6571
Gemma-7B	0.6109	0.8247	0.6603	0.4491	0.6429

PAFT trained models on HuggingFace LLM Leaderboard:

- Number 1 in 7B/8B category

- Number 1 among all models globally.[7]

Besides the layered approach and parallel training, there are many researches aimed at accelerating the pre-training of LLMs. One of them is Huang et al. from Ant Group and Huazhong University of Science and Technology propose Chain-of-Sight, an innovative vision-language bridging module that significantly accelerates the pre-training of multimodal large language models (MLLMs). Their approach strategically reduces visual tokens during pre-training to optimize computational efficiency while employing a compound token scaling strategy during fine-tuning to enhance model performance. The researchers' multi-scale visual resamplers effectively capture both global and local visual contexts across different spatial resolutions. Experimental results demonstrate that their method achieves a 73 percent reduction in pre-training time while maintaining or surpassing the performance of conventional approaches that use full token sets throughout training. This work by Huang and colleagues presents an important advancement in efficient training methodologies for multimodal AI systems, addressing critical challenges in computational resource utilization for large-scale model development.[8]

3
SYSTEM ANALYSIS AND FEASIBILITY

In this section, the design and operation of the system will be explained. In addition, the results of the feasibility studies conducted for the project are presented.

3.1 System Analysis

The system initially obtains a sub-model by using a base model and data set and closing some training layers. It measures its success. This process continues by closing different training layers and obtaining different models using different data sets.

These models are then merged into a single model with the Merge process. The aim is to obtain a more effective and successful model. The success of this model is measured and attempts are made to improve it.

These processes are performed in the Google Colab environment in Python using the Nvidia A100 GPU, and the models obtained are stored on Google Drive.

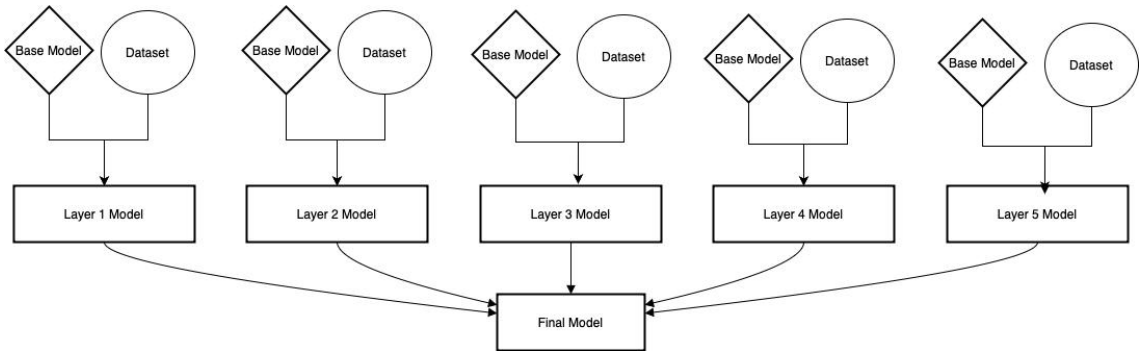

Figure 3.1 System Design

3.2 Feasibility

Feasibility items were determined by considering project needs and requirements in many different metrics. As a result of these items, it was determined that there were

sufficient resources for the realization of the project and that there were no obstacles.

3.2.1 Technical Feasibility

Technical Feasibility was examined by separating software and hardware. The combination of colab and python, which can work in harmony with each other using Nvidia A100 GPU, was selected as appropriate for the project.

3.2.1.1 Hardware Feasibility

Three options were considered regarding hardware. Among these, it was decided to use the Colab option because it was more accessible and much more powerful in terms of GPU. Project was supported with a 40 GB VRAM A100 GPU, 83 GB system RAM, and 112 GB hard disk storage,

3.2.1.2 Software Feasibility

Throughout the project, the Python programming language, which is a leading and easy-to-use language in subjects such as artificial intelligence, natural language processing and machine learning, will be used together with the Google Colab platform, which offers the advantage of high processing power.

3.2.2 Legal Feasibility

The software and operating systems used in the project are licensed. The models used are open source. There is no copyright. There is no legal obligation.

3.2.3 Economic Feasibility

There will be no additional fee for the hardware to be used in the project. The models and software used are free. If needed in Colab, Colab Pro will be purchased for 165 Turkish Liras per month for the duration of the project. Additionally, Google Drive costs 50 Turkish Liras.

3.2.4 Labor and Time Feasibility

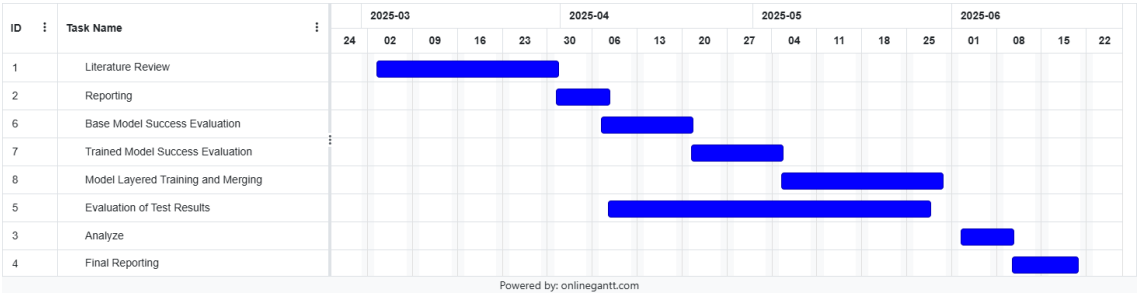

Figure 3.2 Labor and Time Feasibility

4
SYSTEM DESIGN

Many models are trained on Google Colab by closing their layers. The trained layers are merged in the final step. The success rate of the models are measured in every step.

4.0.0.1 Preprocessing Steps

Models to be used selected as Llama-3.2-1B and Llama-3.2-1B-Instruct. The performance of the models before training was measured using Evaluation Harness method. Dataset is saved on Google Drive.

4.0.0.2 Applied Methods

These models were trained using different parts of the dataset and by closing different layers. This training was carried out with PyTorch on Colab. The trained models were saved on Google Drive. Their success after training was measured with Evaluation Harness. Then these models were merged with different methods of the Mergekit library. Success rate was measured again to see results.

4.1 Material and Dataset

The Cosmopedia dataset contains approximately 30 million files and 25 billion tokens. Various parts of the cosmopedia dataset was used to train models.

Table 4.1 Models and Parameters evaulated in English Language

Models	Parameters
EleutherAI/gpt-neo-125M	125M
Locutusque/TinyMistral-248M	248M
openai-community/gpt2	124M
EleutherAI/pythia-410m	410M
Llama-3.2-1B	1B
Llama-3.2-1B-Instruct	1B
Phi-1.5	1.3B
Phi-2	2.7B
Mistral-7B-v0.3	7B
Mistral-7B-Instruct-v0.3	7B
Llama-3-8B-Instruct	8B
Gemma-2B	2B
Gemma-9B	9B

Table 4.2 Models and Parameters evaulated in Turkish Language

Models	Parameters
Turkish-GPT2-Large	750M
Turkish-GPT2-Large-Instruct-v0.1	750M
Gemma-2B	2B
Gemma-9B	9B
MetaMath-Mistral-7B	7B
Llama-3.2-1B	1B

4.2 Method

Selected models trained using PyTorch by closing training layers on Cosmopedia dataset, success rates measured before and after training. Trained models saved on Google Drive. Models merged using Mergekit to create final model.

4.2.1 Training with PyTorch

In the training process implemented using PyTorch, a pre-trained language model was first loaded into memory, and text data was prepared by converting it into numerical representations that the model could process. Each text entry was tokenized and adjusted to a fixed length through truncation or padding. The training data was then divided into small batches, which were iteratively fed into the model. For each batch, the model's predictions were compared to the true outputs to compute a loss value. This loss was then backpropagated through the network, and the model's parameters were updated using an optimization algorithm. The procedure was repeated over multiple training cycles (epochs), allowing the model to gradually

improve its performance. Upon completion, the final state of the model was saved for future use.[9]

4.2.2 Layer Freezing

Layer freezing is a technique used during model training to prevent certain layers from updating their weights. This method focuses on retaining the pre-learned knowledge of selected layers by excluding them from the optimization process. Typically, earlier layers in a model capture more general features, while deeper layers learn task-specific patterns. By freezing the earlier layers, training becomes more efficient and stable, especially when the amount of new data is limited.[10]

4.2.3 Model merging with mergekit

Model merging is a technique developed to consolidate the strengths of different pre-trained language models into a unified architecture. This approach offers an alternative to full retraining by reducing computational costs while aiming to enhance performance. One prominent method in model merging is linear merging, which produces a new model by directly averaging the weights of constituent models using fixed ratios—valued particularly for its simplicity and computational efficiency. In contrast, the spherical linear interpolation (slerp) method blends weights along spherical curves in a high-dimensional space, enabling smoother transitions and more natural integration. The TIES (Task-Informed Expert Selection) method stands out with its layer-wise flexibility; it allows users to assign varying density and weight values to specific parameter groups or layers from different models, enabling more fine-grained and task-specific combinations. MergeKit facilitates the implementation of these methods by providing a YAML-based configuration system that enables detailed control over both local and remote models. As a result, it becomes possible to generate more balanced and optimized models without the need for additional training via transfer learning.[11][12]

4.2.4 Evaluation Harness

Evaluation Harness is a standardized framework designed to systematically and reproducibly assess the capabilities of large language models across a variety of tasks. It enables performance comparison through consistent metrics such as accuracy, exact match, and normalized accuracy, thereby facilitating an objective analysis of model strengths and weaknesses. In this project, the Evaluation Harness was utilized with both English and Turkish task sets to evaluate the multilingual and multitask performance of different models. The assessments were carried out without further

fine-tuning, using a fixed number of examples, and the results were reported alongside standard error (stderr) values to ensure statistical reliability. This methodology allowed for a detailed comparison between base and fine-tuned models, contributing to a more informed understanding of their effectiveness across different linguistic contexts.

5
EXPERIMENTAL RESULTS

In this section, it is examined what kind of performance gain is achieved in training the models by closing the selected layers. ARC Challenge, Hellaswag and validation loss were used to measure the performance of Cosmopedia models. Gsm8k was used in Math models. In one of the Cosmopedia models, the base model was loaded normally, and in the other, the model weights were loaded randomly. In one of the mathematical models, the Gsm8k Train dataset was used, and in the other, the MetaMathQA dataset was used.

5.1 Performance Analysis

In this section, 4 benchmarks used in the evaluation of the models will be discussed and the performance of the models will be examined. As a result of all these metrics, it was appropriate to perform the dataset dividing and layer closing approachs on meta/Llama-3.2-1B.

5.1.1 ARC Challange

ARC Challenge, whose full name is AI2 Reasoning Challenge, was developed by the Allen Institute for AI. It is a benchmark that measures human-like thinking and human reasoning. It highlights concepts such as logic and conceptual analysis. Arc Challenge dataset addresses more difficult questions, unlike Arc Easy. To solve these questions, features such as logical analysis and deep thinking must be strong. The Turkish version of this metric was used to evaluate models.

Table 5.1 ARC-Challange scores of models for English language

Model	Parameter	ARC-Challange Acc	ARC-Challange Acc_Norm	ARC-Challange Stderr
EleutherAI/gpt-neo-125M	125M	0.1911	0.2312	0.0115
Locutusque/TinyMistral-248M	248M	0.1911	0.2287	0.0115
openai-community/gpt2	124M	0.1903	0.2270	0.0115
EleutherAI/pythia-410m	410M	0.2116	0.2432	0.0119
meta/Llama-3.2-1B	1B	0.3140	0.3626	0.0136
meta/Llama-3.2-1B-Instruct	1B	0.3567	0.3788	0.0140
microsoft/phi-1_5	1.3B	0.4462	0.4804	0.0145
microsoft/phi-2	2.7B	0.5299	0.5401	0.0146
mistralai/Mistral-7B-v0.3	7B	0.4872	0.5196	0.0146
mistralai/Mistral-7B-Instruct-v0.3	7B	0.5751	0.5845	0.0144
meta/Llama-3-8B-Instruct	8B	0.5307	0.5674	0.0145
google/gemma-2-2b	2B	0.4680	0.4980	0.0158
google/gemma-2-9b	9B	0.6240	0.6610	0.0150

Table 5.2 ARC-Challange scores of models for Turkish language

Model	Parameter	ARC-Challange TR Acc	ARC-Challange TR Acc_Norm	ARC-Challange Stderr
ytu-ce-cosmos/turkish-gpt2-large	750M	0.2075	0.2400	0.0119
ytu-ce-cosmos/turkish-gpt2-large-750m-instruct-v0.1	750M	0.2203	0.2570	0.0121
google/gemma-2-2b	2B	0.1830	0.2230	0.0122
google/gemma-2-9b	9B	0.8670	0.8770	0.0107
meta-math/MetaMath-Mistral-7B	7B	0.2440	0.2820	0.0136
meta/Llama-3.2-1B	1B	0.2440	0.2841	0.0132


```
{
 "question": {
  "stem": "Which of these animals is a mammal?",
  "choices": [
   {"label": "A", "text": "Lizard"},
   {"label": "B", "text": "Frog"},
   {"label": "C", "text": "Whale"},
   {"label": "D", "text": "Shark"}
  ]
 },
 "answerKey": "C"
}
```


Figure 5.1 ARC Dataset Example

5.1.2 HellaSwag

Hellaswag is a benchmark that evaluates a model's ability to reason like a human with cause-effect relationships. The Turkish translated version of this metric was used to

evaluate models.[13][14][15]

Table 5.3 HellaSwag scores of models for English language

Model	Parameter	HellaSwag Acc	HellaSwag Acc_Norm	HellaSwag Stderr
EleutherAI/gpt-neo-125M	125M	0.2867	0.3040	0.0045
Locutusque/TinyMistral-248M	248M	0.2701	0.2828	0.0044
openai-community/gpt2	124M	0.3375	0.4059	0.0047
EleutherAI/pythia-410m	410M	0.2892	0.3114	0.0045
meta/Llama-3.2-1B	1B	0.4380	0.5570	0.0157
meta/Llama-3.2-1B-Instruct	1B	0.4514	0.6077	0.0049
microsoft/phi-1_5	1.3B	0.4797	0.6261	0.0048
microsoft/phi-2	2.7B	0.5588	0.7376	0.0040
mistralai/Mistral-7B-v0.3	7B	0.6093	0.8031	0.0040
mistralai/Mistral-7B-Instruct-v0.3	7B	0.6485	0.8293	0.0038
meta/Llama-3-8B-Instruct	8B	0.5775	0.7583	0.0043
google/gemma-2-2b	2B	0.4860	0.6350	0.0158
google/gemma-2-9b	9B	0.5360	0.6960	0.0146

Table 5.4 HellaSwag scores of models for Turkish language

Model	Parameter	HellaSwag TR Acc	HellaSwag TR Acc_Norm	HellaSwag TR Stderr
ytu-ce-cosmos/turkish-gpt2-large	750M	0.3210	0.3651	0.0047
ytu-ce-cosmos/turkish-gpt2-large-750m-instruct-v0.1	750M	0.3224	0.3694	0.0048
google/gemma-2-2b	2B	0.2750	0.3050	0.0146
google/gemma-2-9b	9B	0.5360	0.6960	0.0158
meta-math/MetaMath-Mistral-7B	7B	0.3400	0.3640	0.0150
meta/Llama-3.2-1B	1B	0.3050	0.3020	0.0146

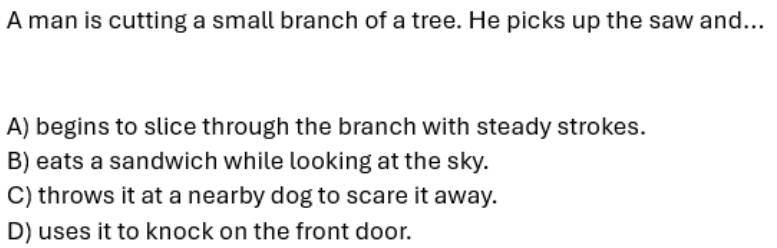

Figure 5.2 HellaSwag Dataset Example

5.1.3 GSM8K

GSM8K (Grade School Math 8K) is a benchmark with a dataset containing high quality mathematic questions for primary school. It contains more than 8500 questions. In the evaluation, models not related to mathematics were not considered.

Table 5.5 GSM8K scores for meta/Llama-3.2-1B

Model	Parameter	GSM8K Flexible Acc	GSM8K Strict Acc	GSM8K Stderr
meta/Llama-3.2-1B-Instruct	1B	0.342	0.342	0.0150
meta/Llama-3.2-1B	1B	0.067	0.067	0.0079

Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May?	Natalia sold 48/2 = <<48/2=24>>24 clips in May. Natalia sold 48+24 = <<48+24=72>>72 clips altogether in April and May. #### 72

Figure 5.3 GSM8K Dataset Example

5.1.4 Validation Loss

In addition to these 3 benchmarks, there is a validation loss dataset, the larger one with 900 samples and the smaller one with 100 samples. Validation loss was strictly followed, especially in the experiments conducted on meta/Llama-3.2-1B.

Table 5.6 Validation loss scores for `meta/Llama-3.2-1B`

Validation Type	Validation Loss
Small Validation	2.4700
Large Validation	8.4196

5.2 Comparative Analysis

In this section, the effects of the layer closure approach on metrics, its benefits in terms of time, and what kind of gains can be achieved with different approaches are discussed. All analysis are made on meta/Llama-3.2-1B. Cosmopedia and metamath datasets were used.

5.2.1 Model Comparasion

As a result of the comparisons made between the models, it was understood that the most suitable model for the project was meta/Llama-3.2-1B. The layer closure approach will be tried on this model.

5.2.1.1 ARC-Challange Comparison

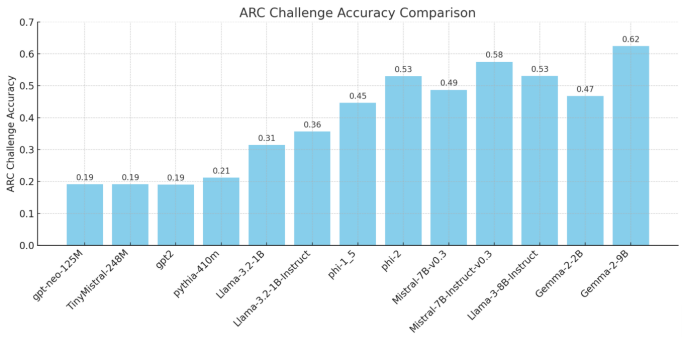

Figure 5.4 ARC-Challange Accuracy Comparision for English Language

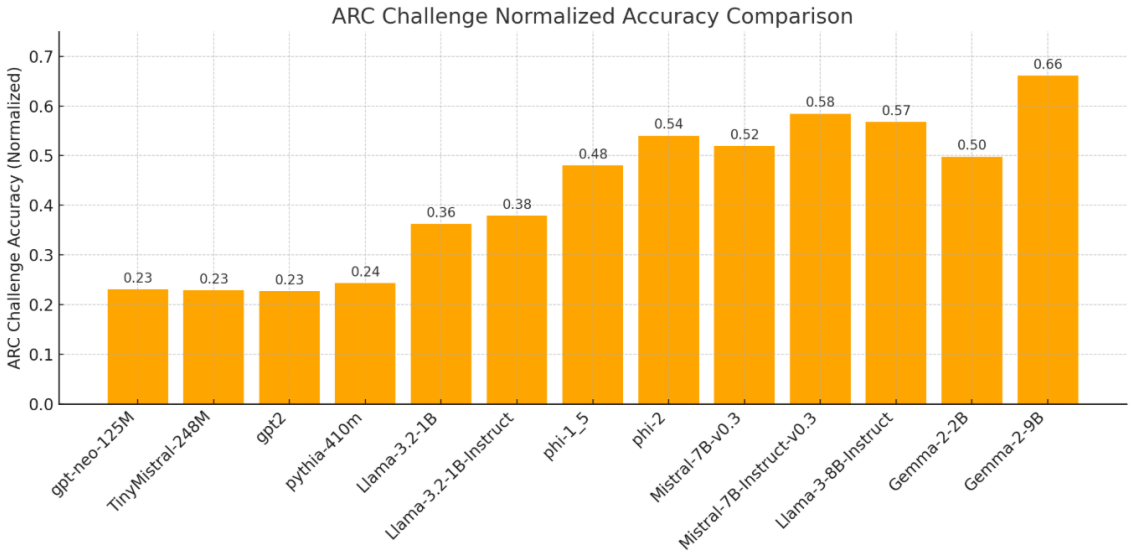

Figure 5.5 ARC-Challange Accuracy Normalized Comparision for English Language

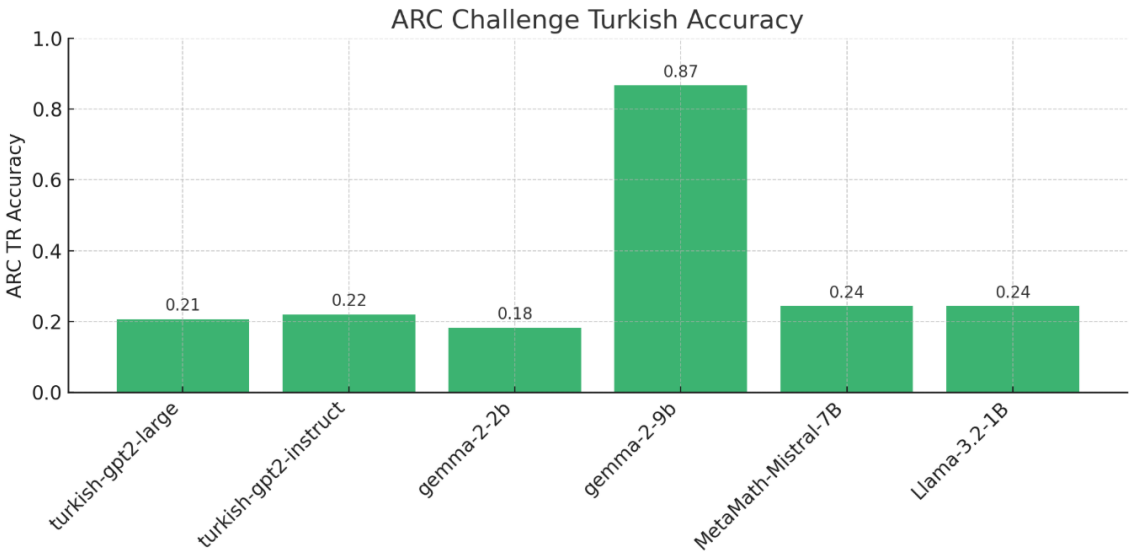

Figure 5.6 ARC-Challange Accuracy Comparision for Turkish Language

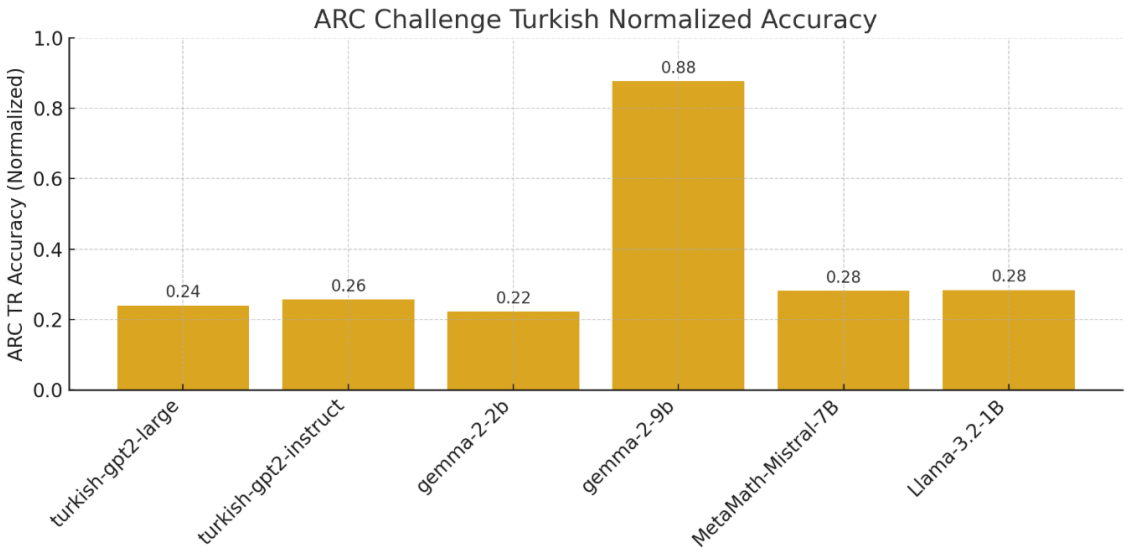

Figure 5.7 ARC-Challange Accuracy Normalized Comparision for Turkish Language

5.2.1.2 HellaSwag Comparison

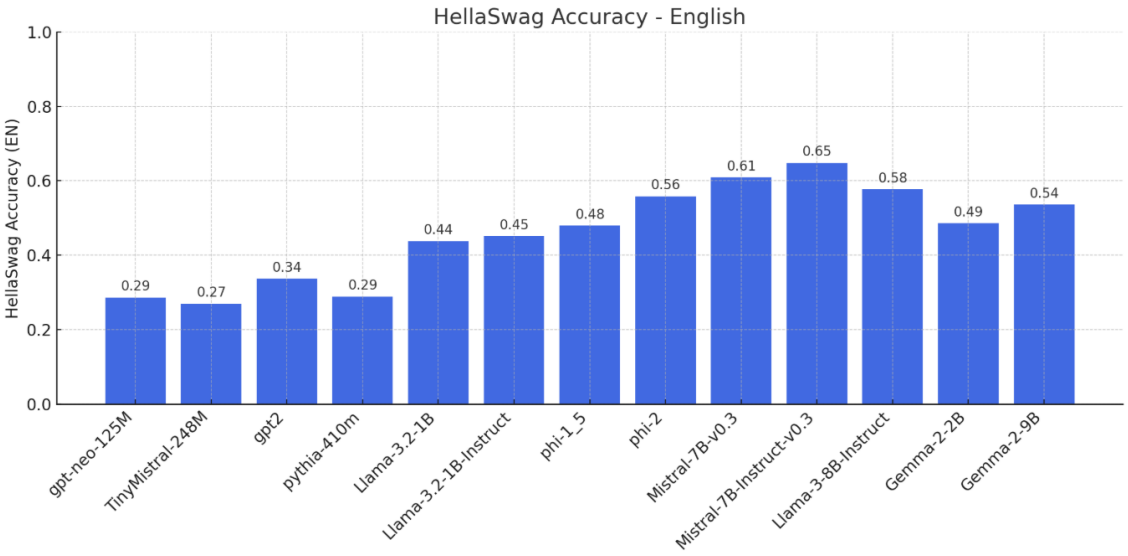

Figure 5.8 HellaSwag Accuracy Comparision for English Language

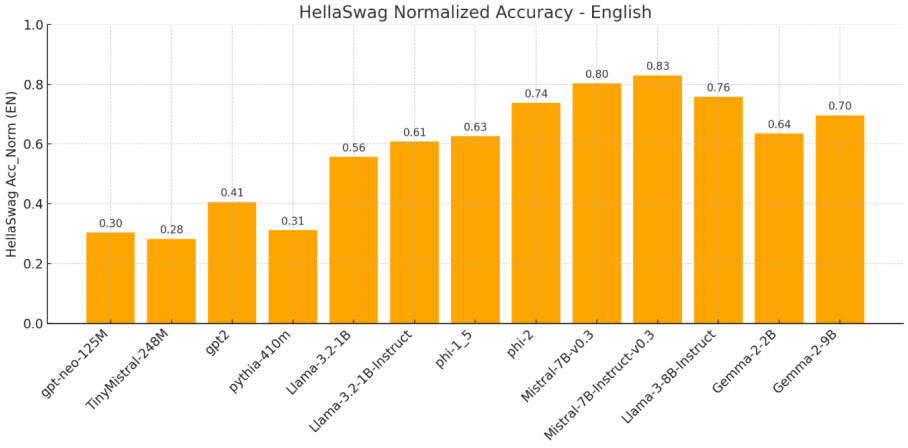

Figure 5.9 HellaSwag Accuracy Normalized Comparision for English Language

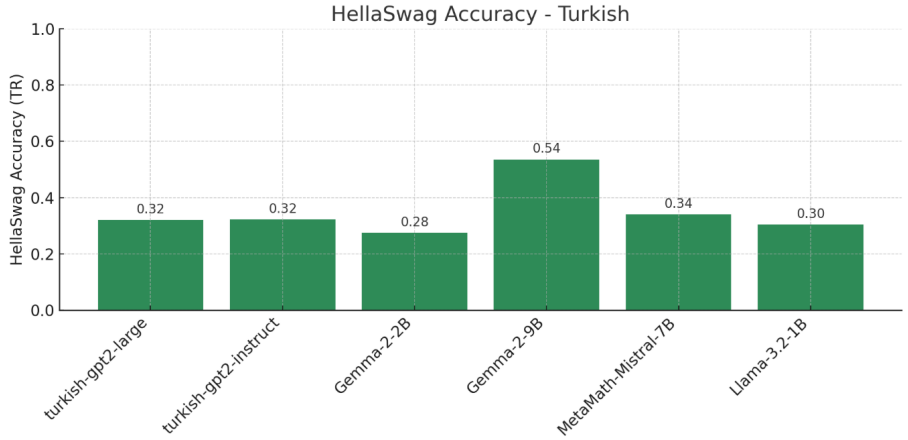

Figure 5.10 HellaSwag Accuracy Comparision for Turkish Language

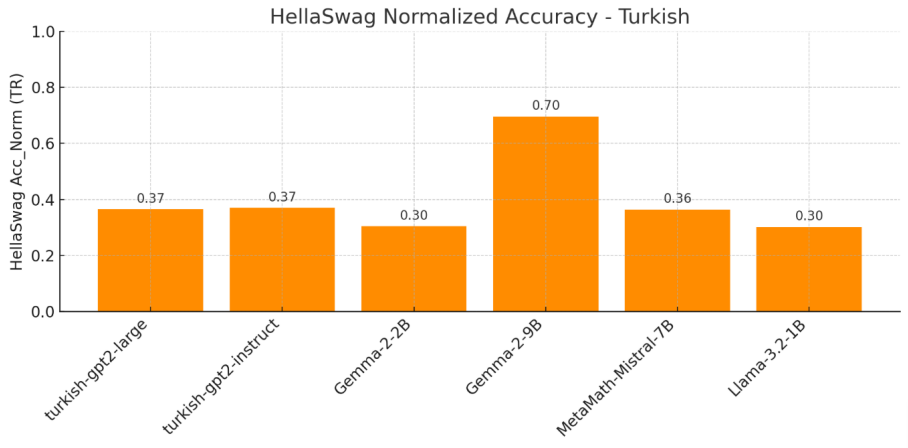

Figure 5.11 HellaSwag Accuracy Normalized Comparision for Turkish Language

5.2.1.3 GSM8K Comparison

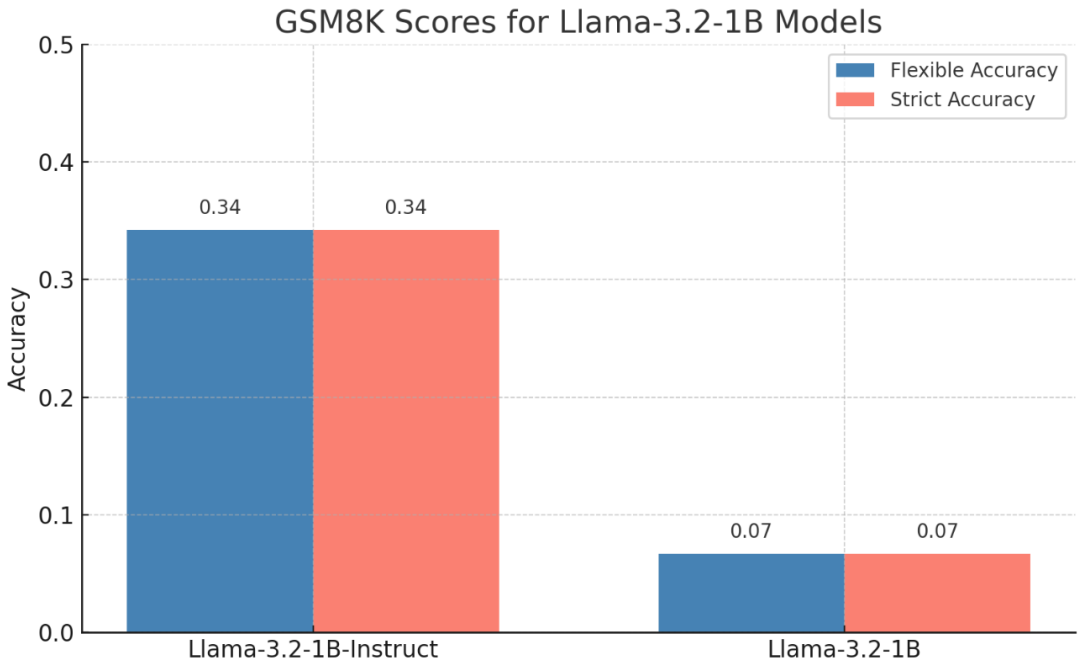

Figure 5.12 GSM8K Accuracy Comparision for meta/Llama-3.2-1B Models

5.3 Layer Closure Examples for Cosmopedia Dataset

9 different layer approaches were used in the experiment. 400mb cosmopedia dataset was divided into 4 parts of 100mb each.

Table 5.7 Layer Configuration Table

SLERP	Part 1	Part 2	Part 3	Part 4
1	0-4	4-8	8-12	12-16
2	0-8	0-8	8-16	8-16
3	0-6	4-9	7-13	11-16
4	0-16	0-16	0-16	0-16
5	0-4	0-8	8-16	12-16
6	0-16	4-16	8-16	12-16
7	0-12	6-12	10-16	4-16
8	0-4	4-16	4-16	4-16
9	0-12	12-16	12-16	12-16

5.3.0.1 Merge Parts

When merging the parts, the process is done as follows:

1-Part 1 and Part 2 are considered. Common layers are shared 50 percent-50 percent. In non-common layers, the desired layer is present on the side that has 80 percent weight. The same process is done for Part 3 and Part 4. It is also done for the last merged Part1-2 and Part 3-4. Thus, a training divided into 4 is simultaneously realized in a shorter time and the loss is minimized. For example, if the combination of part 1 and part 2 in sleep 3 is examined, the layers 4-6, which are common to both, are shared as 50 percent. From part 1, 0-4 is taken as 80 percent, and from part 2, 6-9 is taken as 80 percent. Thus, we obtain the combined part1-2.

5.3.0.2 Validation Loss for SLERPs

Performance of the Llama-3.2-1B base model is recorded as 2.4700 on the small validation set and 8.4196 on the large validation set.

When the standard fine-tuning method "vanilla" is applied, a big improvement in the performance of the model on both validation sets is achieved. The loss on the small validation set is reduced to 2.2424, while the loss on the large validation set is reduced to a very low level of 0.1702.

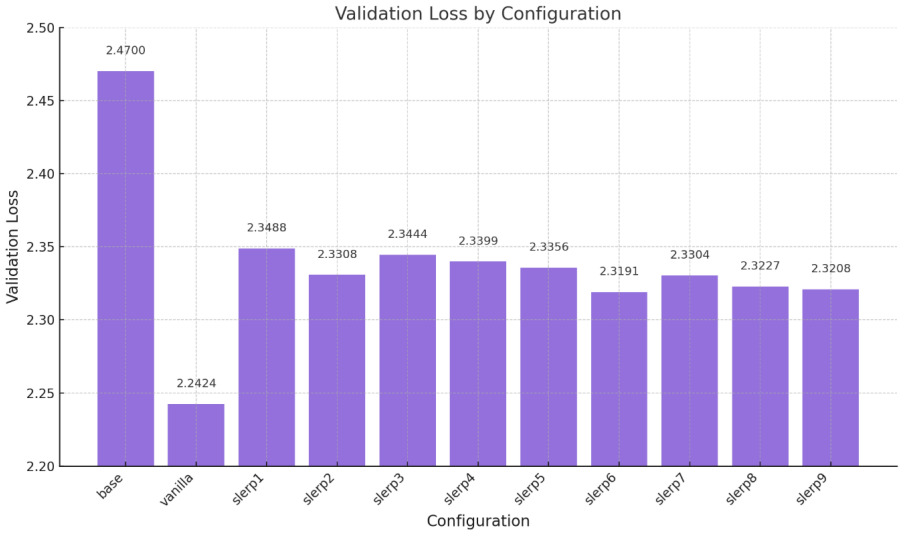

Figure 5.13 Small Validation Loss for SLERPs

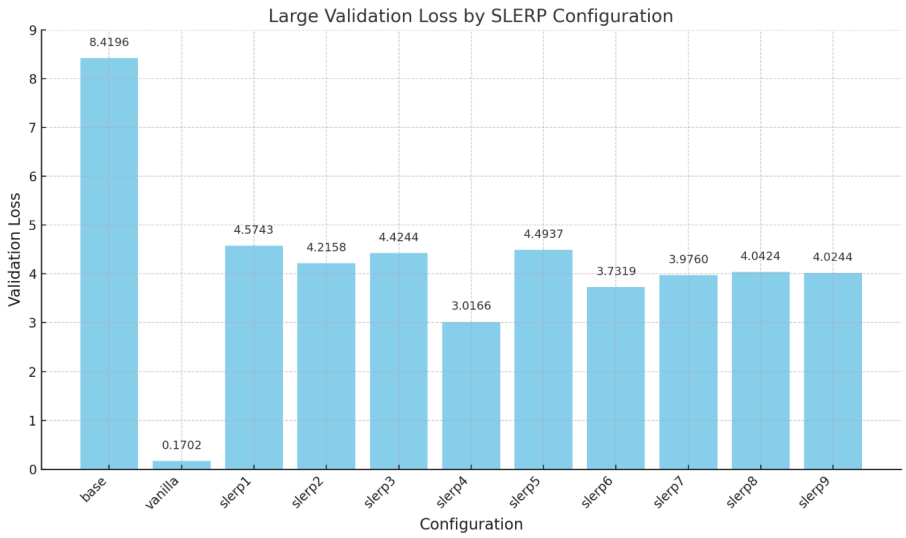

Figure 5.14 Large Validation Loss for SLERPs

5.3.0.3 ARC-Challange Score Comparision

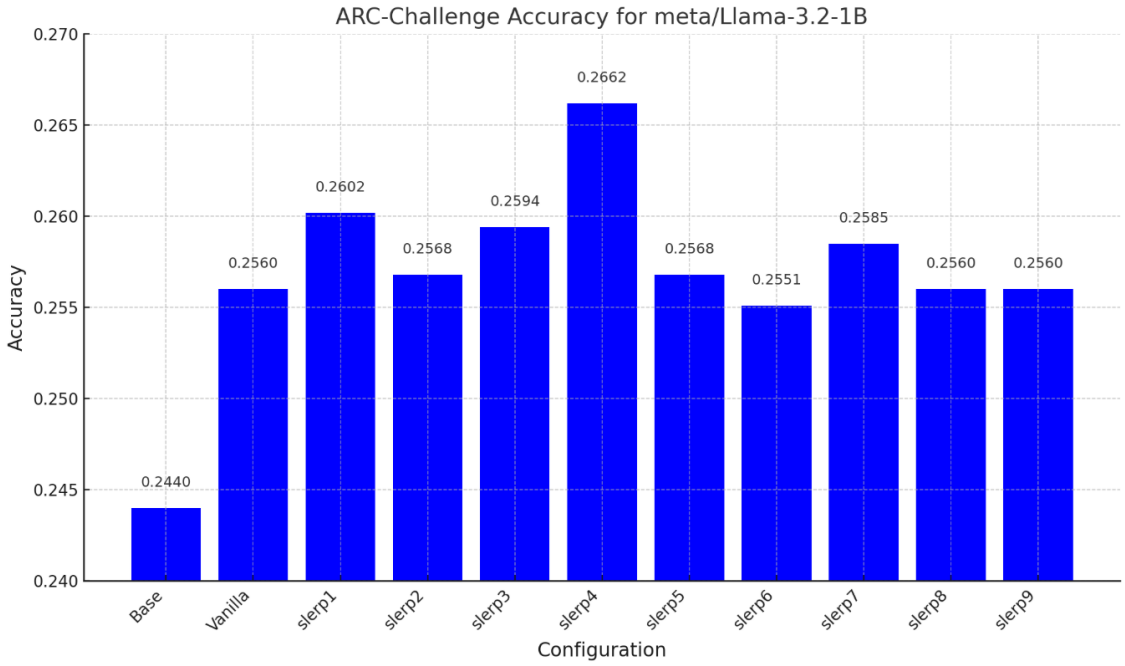

Figure 5.15 ARC Challange Accuracy for SLERPs

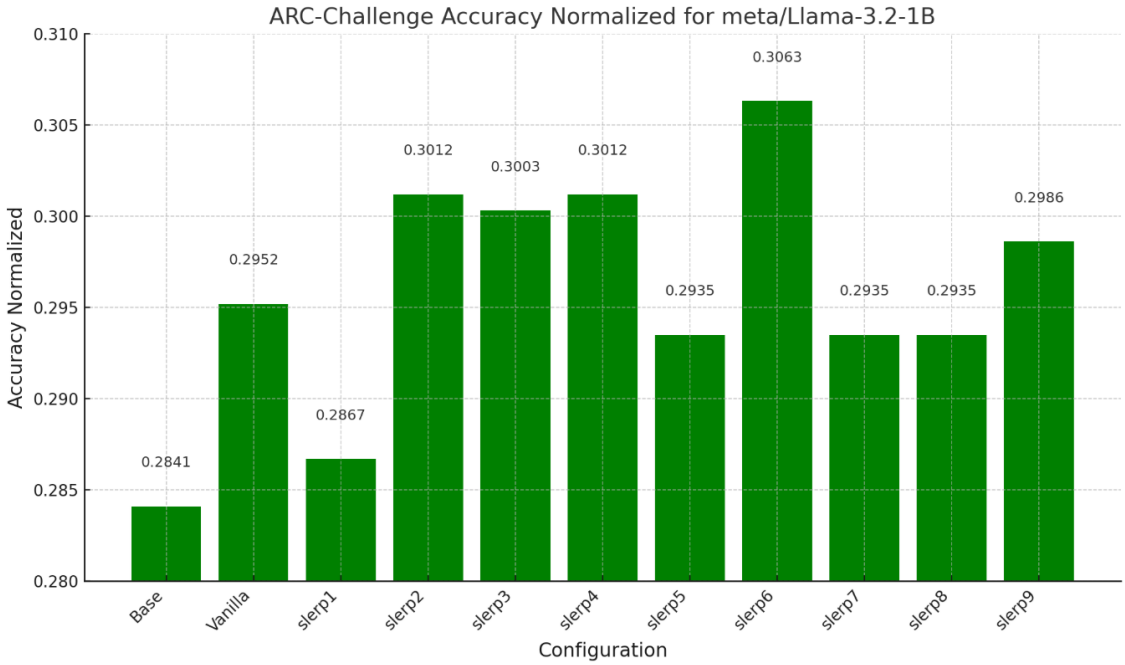

Figure 5.16 ARC Challange Accuracy Normalized for SLERPs

5.3.0.4 HellaSwag Score Comparision

The model's scores on the Hellaswag in addition to ARC benchmarks rose after a test of the Llama-3.2-1B base model's fine tuning output on the Cosmopedia dataset. On Hellaswag, accuracy went up from 0.3090 to 0.3099. ARC accuracy also increased from 0.2440 to 0.2560 with regular fine tuning.

SLERP methods mostly kept these improvements; they also produced good outcomes in the ARC metrics. At times, SLERP outperformed the regular model on Hellaswag acc norm scores, which was 0.3526. For example, the score reached 0.3547 with SLERP 4. This shows that SLERP methods keep or improve the model's output on tasks such as Hellaswag besides ARC.

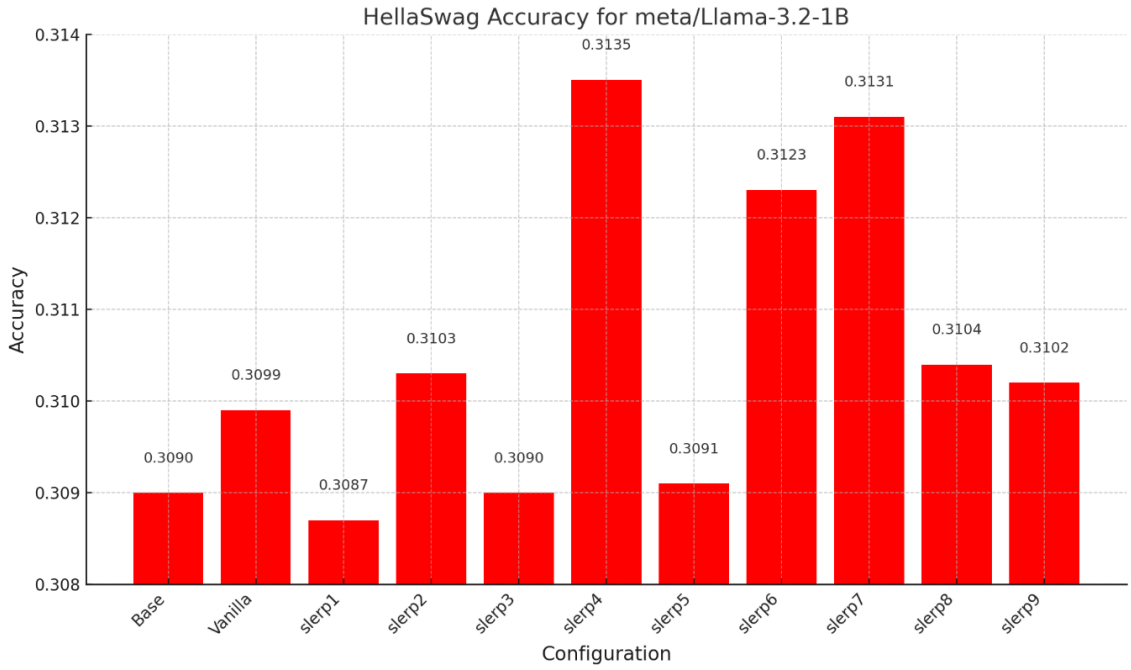

Figure 5.17 HellaSwag Accuracy for SLERPs

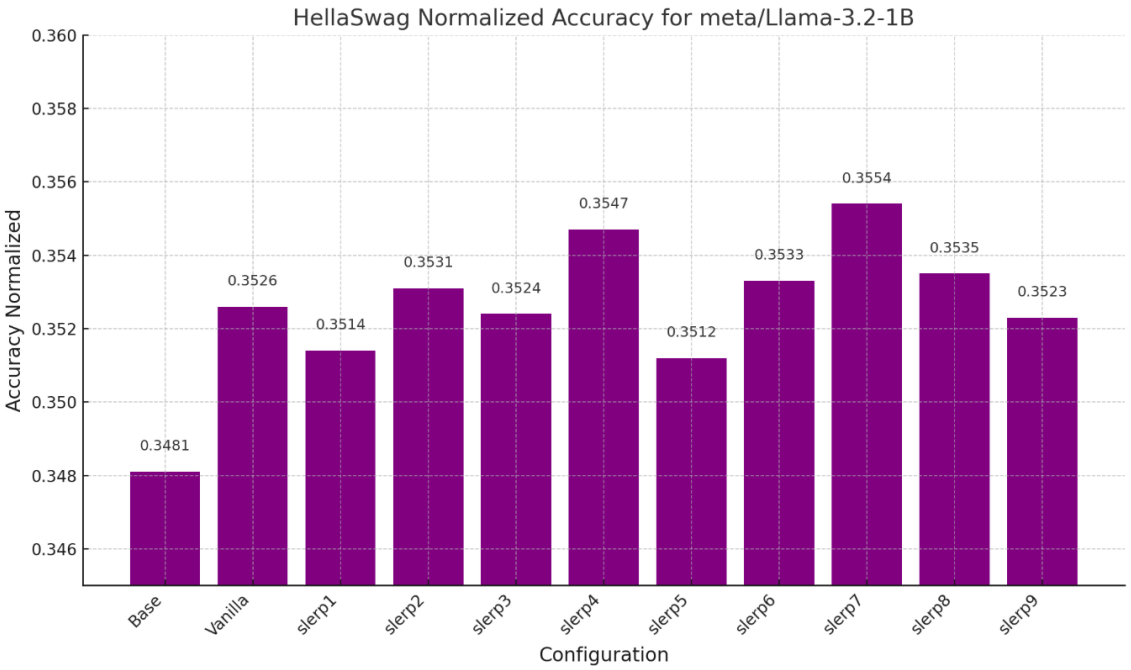

Figure 5.18 HellaSwag Accuracy Normalized for SLERPs

5.3.0.5 Comparision Table

We fine tuned the Llama-3.2-1B model with the Cosmopedia dataset. We worked to shorten the time to train it, but kept its performance. Regular fine tuning lowered the

model's validation loss much, for example, it fell from 8.4196 to 0.1702 on the large set. This took 3 hours and 16 minutes.

Our SLERP plans shortened this time to 19.1 % and 25.3%, which saved a lot of time. In terms of how well it did, SLERP models kept the validation loss low - it fell by 54.4% and 70.9%, especially on the large set. SLERP 4 was the best plan to keep the performance.

SLERP models also did as well as, or better than, the regular model in other tests, like Hellaswag in addition to ARC. SLERP plans shortened the training time a lot; they gave good fine tuning, though the model's performance changed a little.

Strategy	Small Val. Loss	Large Val. Loss	Small Val. Loss Preservation (%)	Large Val. Loss Preservation (%)	Duration (min)	% of Vanilla Duration
SLERP 1	2.3488	4.5743	%47.8	%54.4	37.5	%19.1
SLERP 2	2.3308	4.2158	%49.5	%58.0	42	%21.4
SLERP 3	2.3444	4.4244	%48.2	%55.9	39	%19.9
SLERP 4	2.3399	3.0166	%48.7	%70.9	49.5	%25.3
SLERP 5	2.3356	4.4937	%49.1	%55.2	42	%21.4
SLERP 6	2.3191	3.7319	%50.6	%62.0	49.5	%25.3
SLERP 7	2.3304	3.9760	%49.5	%59.6	45	%23.0
SLERP 8	2.3227	4.0424	%50.3	%59.0	45	%23.0
SLERP 9	2.3208	4.0244	%50.5	%59.2	45	%23.0

Figure 5.19 Comparision Table

5.4 Layer Closure Examples for Cosmopedia Dataset : Random Model Weights Approach

The evaluations in this section are based on the same dataset but consider the meta/Llama-3.2-1B loaded with random weights.

To train large language models, picking the starting weights is an important step - it much changes how the model learns and how well it does in the end. People use weights from models that already learned. This helps them fine tune for new tasks. Starting with random weights offers a way to test a model's ability. It shows if the model can learn from nothing and if it adapts to a certain set of data - this approach helps us know how well the model's design plus the training method work, even without earlier knowledge.

5.4.1 Validation Loss for SLERPs - Random Model Weights

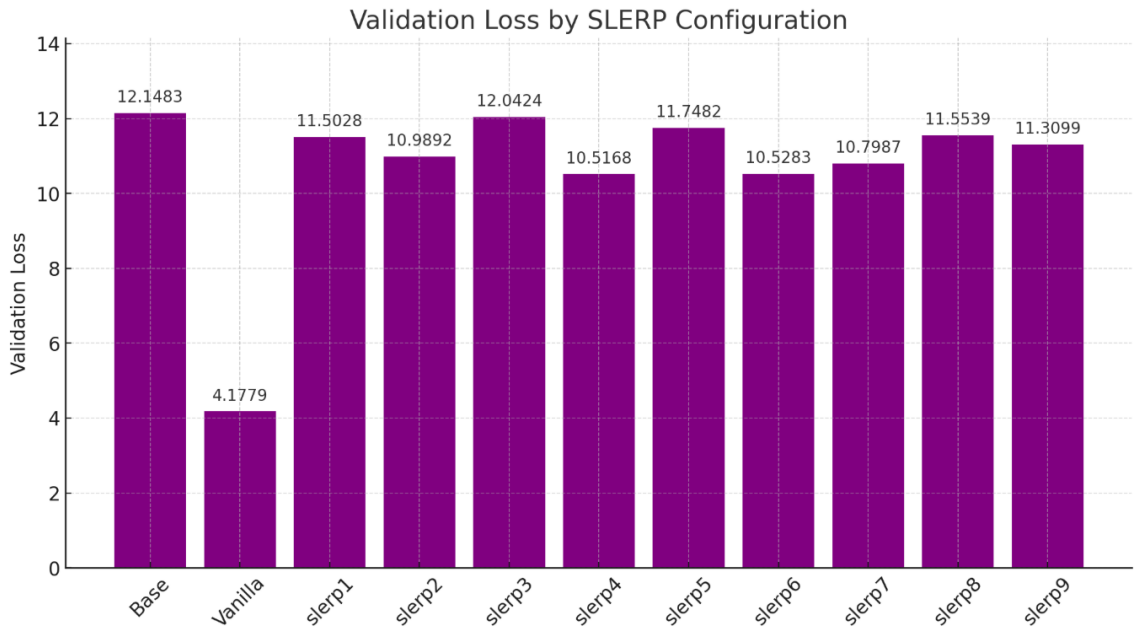

Figure 5.20 Small Validation Loss for SLERPs

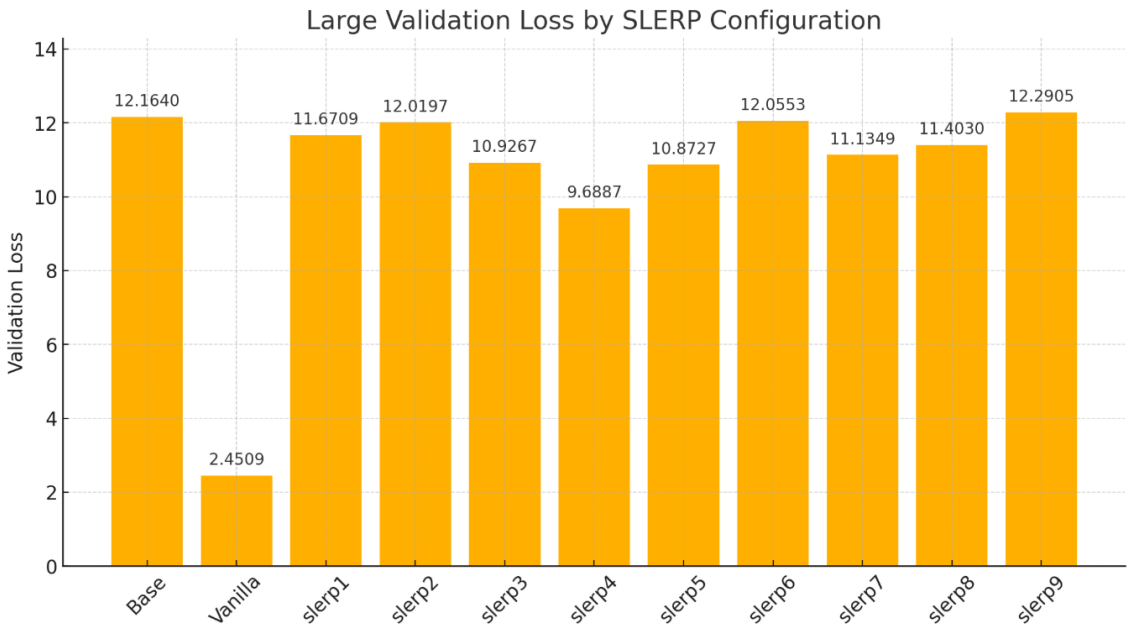

Figure 5.21 Large Validation Loss for SLERPs

5.4.2 ARC-Challange and Validation Score Comparision - Random Model Weights

The base model had a small validation loss of 2.4700 and a large validation loss of 8.4196 in Cosmopedia. For ARC, the accuracy score was 0.2440 and the normalized accuracy score reached 0.2841.

The Cosmopedia validation losses in addition to ARC scores of the model that began with random weights showed much lower numbers when compared to a model that had prior training. This shows that transfer learning, which pre trained weights offer, is essential for large language models to fine tune well.

Scientists concluded that a model, which started with random weights, would require more data or more training time to reach a similar performance level from the beginning. The SLERP strategies in our project did not work in models that had closed model weights.

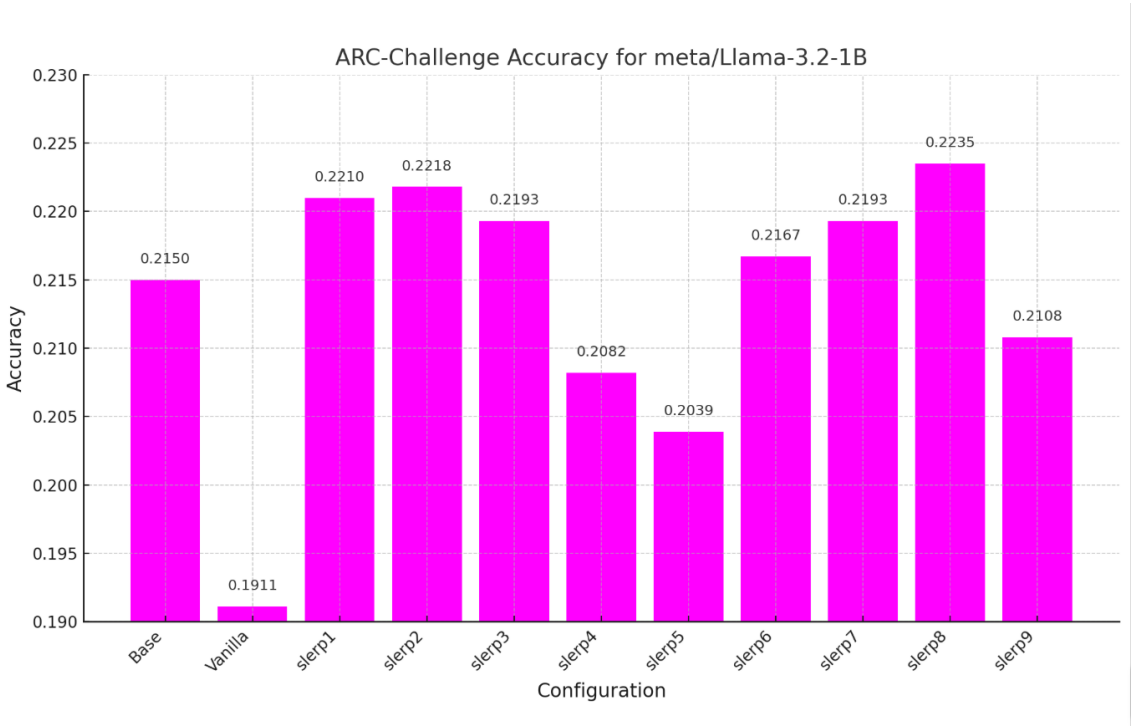

Figure 5.22 ARC Challange Accuracy for SLERPs - Random Model Weights

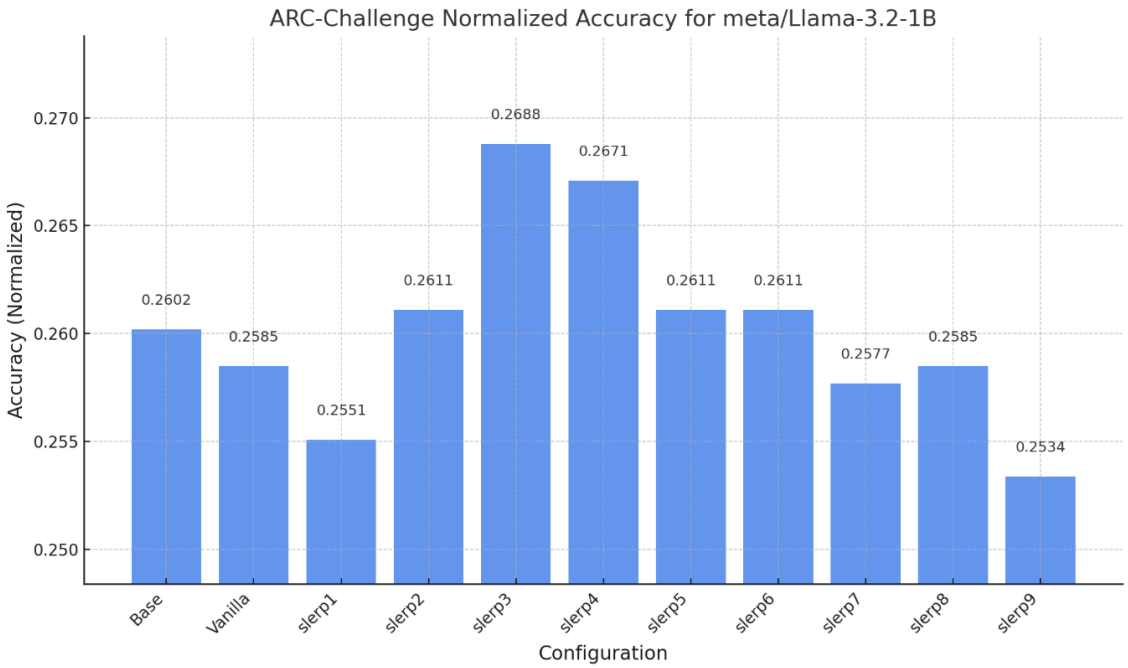

Figure 5.23 ARC Challange Accuracy Normalized for SLERPs - Random Model Weights

5.4.3 HellaSwag Score Comparision - Random Model Weights

This result supports prior findings about ARC and validation losses, it shows that transfer learning, the knowledge from pre trained weights, is important for good results in large language models. Our observations indicate that models, which start from scratch, need much more data or a longer training time to perform as well.

The SLERP strategies work for models that already have knowledge. But these strategies did not help with randomly initialized weights. This confirms that pre trained models are a good place to start fine tuning.

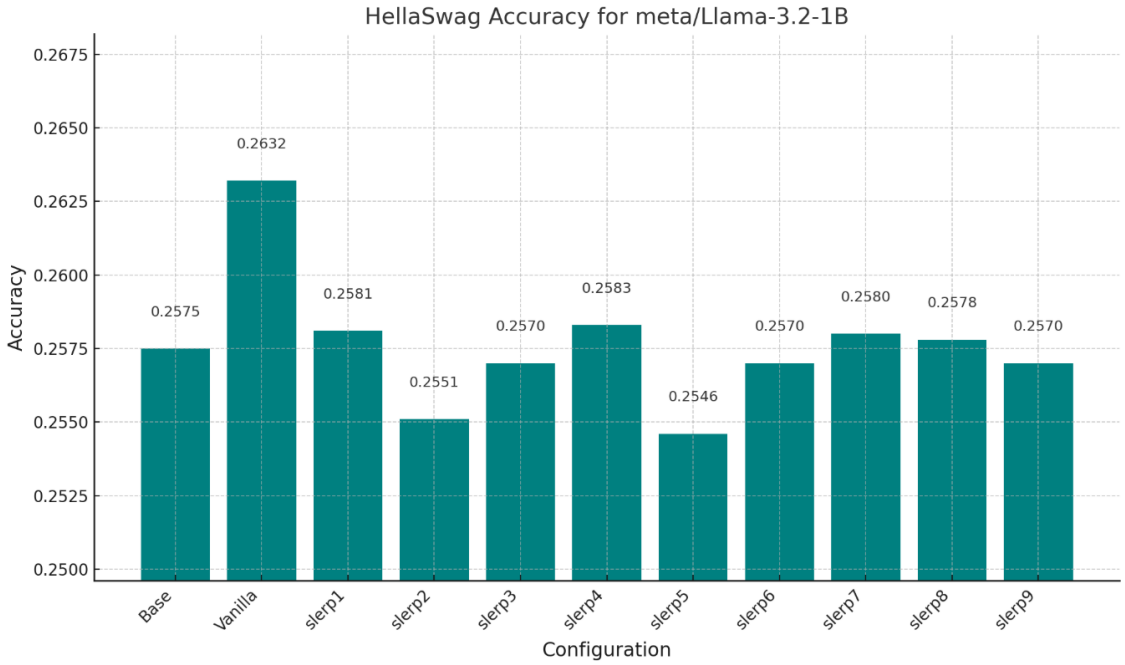

Figure 5.24 HellaSwag Accuracy for SLERPs - Random Model Weights

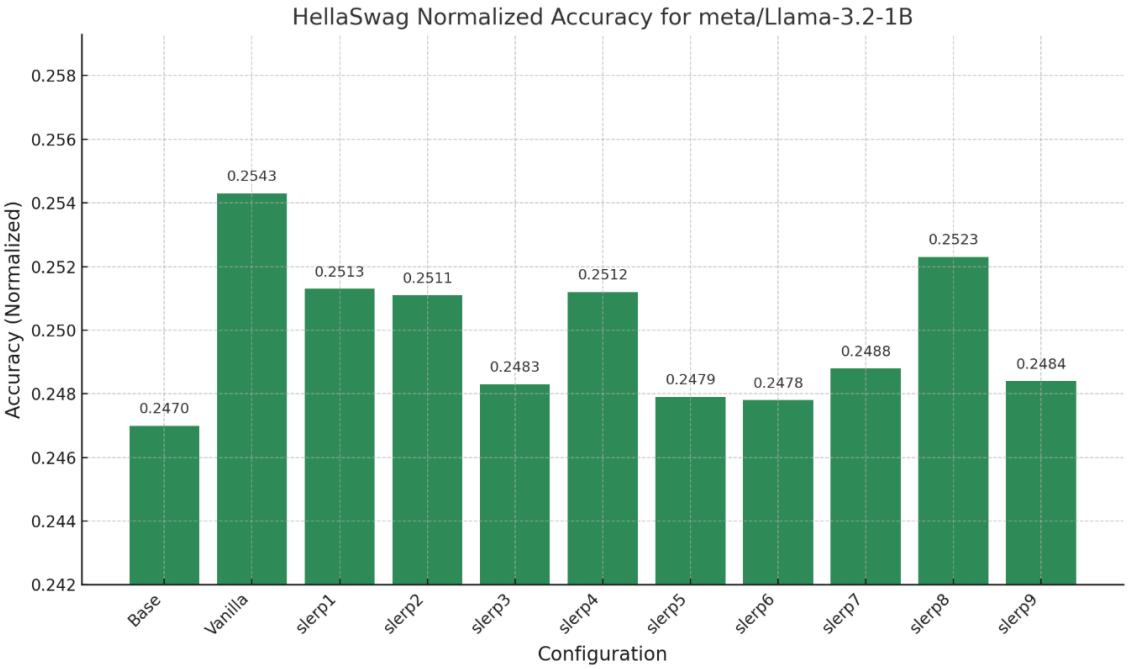

Figure 5.25 HellaSwag Accuracy Normalized for SLERPs - Random Model Weights

5.4.4 Comparision Table - Random Model Weights

The small validation loss protection for the strategies goes from 1.3 % (SLERP 3) to 20.1 % (SLERP 4). This indicates that SLERP struggles to lower the loss as much as vanilla does in the small validation set. SLERP 4 is the best option.

A wider range describes the large validation loss protection. SLERP 4 got a protection rate of 31.8 %, which was the highest. SLERP 3 (15.8 %) and SLERP 5 (16.5 %) also performed well. SLERP 2 (1.8 %) and SLERP 9 (-2.1 %) show low or negative protection rates. A negative protection rate means the strategy raises the loss that vanilla fine tuning gets.

Strategy	Small Val. Loss	Large Val. Loss	Small Val. Loss Preservation (%)	Large Val. Loss Preservation (%)	Duration (min)	% of Vanilla Duration
SLERP 1	11.5028	11.6709	%8.0	%6.1	9	%20.5
SLERP 2	10.9892	12.0197	%14.3	%1.8	9.75	%22.2
SLERP 3	12.0424	10.9267	%1.3	%15.8	9.5	%21.6
SLERP 4	10.5168	9.6887	%20.1	%31.8	11	%25.0
SLERP 5	11.7482	10.8727	%4.9	%16.5	9.75	%22.2
SLERP 6	10.5283	12.0553	%19.9	%1.3	11	%25.0
SLERP 7	10.7987	11.1349	%16.7	%12.3	10.5	%23.9
SLERP 8	11.5539	11.4030	%7.4	%9.4	10.5	%23.9
SLERP 9	11.3099	12.2905	%10.4	%-2.1	10.5	%23.9

Figure 5.26 Comparision Table - Random Model Weights

5.5 Layer Closure Examples for GSM8K Dataset

There is no validation loss metric since the math score is taken into account in the metrics. GSM8K was used as a criterion.

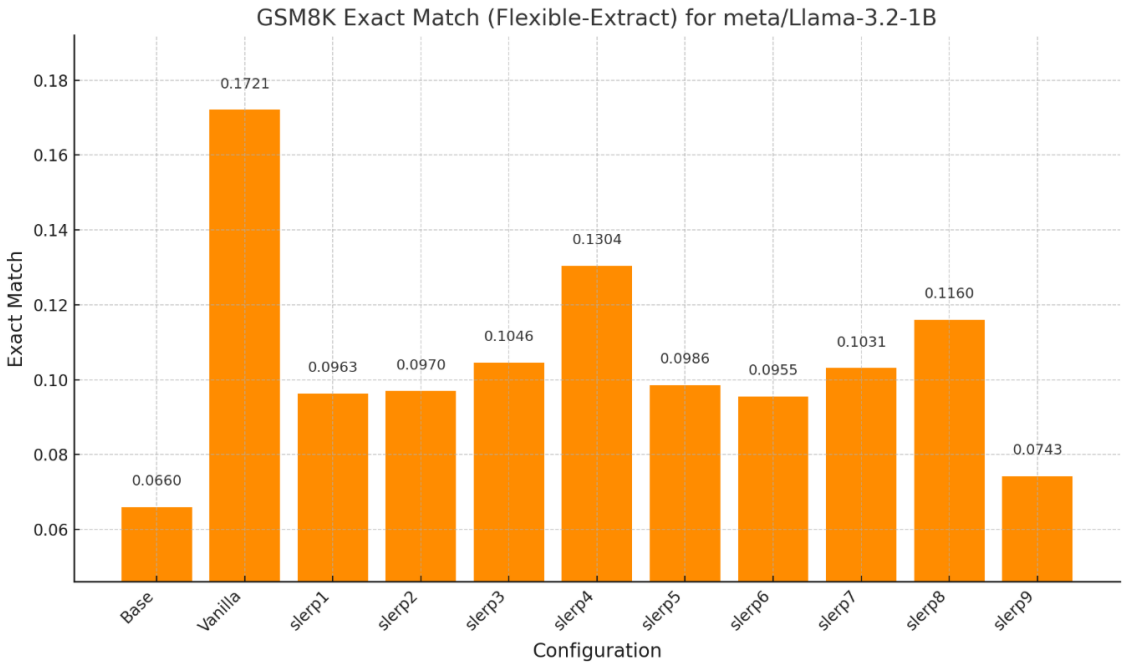

Figure 5.27 GSM8K Flexible-Extract Score for SLERPs

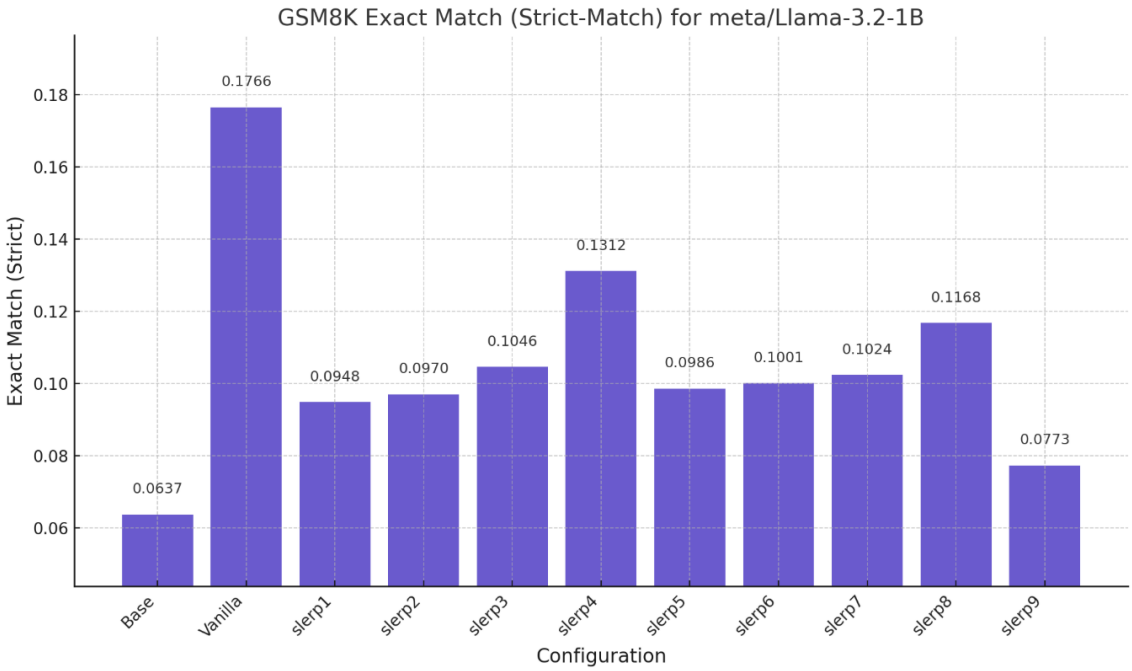

Figure 5.28 GSM8K Strict-Extract Score for SLERPs

5.5.1 Comparision Table - GSM8K Dataset

In finetuning the Llama-3.2-1B model on the GSM8K Train dataset, vanilla training took five minutes. SLERP strategies reduced this time to one to 1.42 minutes, which is around twenty to twenty-eight point three percent. SLERP strategies showed different results in preserving the success in GSM8K scores. In particular, SLERP 4 maintained both the flexible sixty point six percent and the strict sixty one point four percent exact match score increases, and was therefore the best strategy. Other strategies also performed moderately well, ranging from seven point eight percent to forty-seven point seven percent. However, some strategies, such as SLERP 9, were not sufficiently well-preserved. This findings shows that SLERP provides a time benefit for mathematical reasoning tasks.

Strategy	Flexible Exact Match	Strict Exact Match	Flexible Score Increase Preservation (%)	Strict Score Increase Preservation (%)	Duration (min)	% of Vanilla Duration
SLERP 1	0.0963	0.0948	%28.5	%27.4	1	%20.0
SLERP 2	0.0970	0.0970	%29.1	%29.4	1.25	%25.0
SLERP 3	0.1046	0.1046	%36.2	%36.5	1.17	%23.3
SLERP 4	0.1304	0.1312	%60.6	%61.4	1.42	%28.3
SLERP 5	0.0986	0.0986	%30.6	%30.8	1.25	%25.0
SLERP 6	0.0955	0.1001	%27.8	%31.8	1.42	%28.3
SLERP 7	0.1031	0.1024	%34.8	%34.6	1.33	%26.7
SLERP 8	0.1160	0.1168	%46.9	%47.7	1.33	%26.7
SLERP 9	0.0743	0.0773	%7.8	%12.4	1.33	%26.7

Figure 5.29 GSM8K Dataset Comparision Table

5.6 Layer Closure Examples for METAMATHQA Dataset

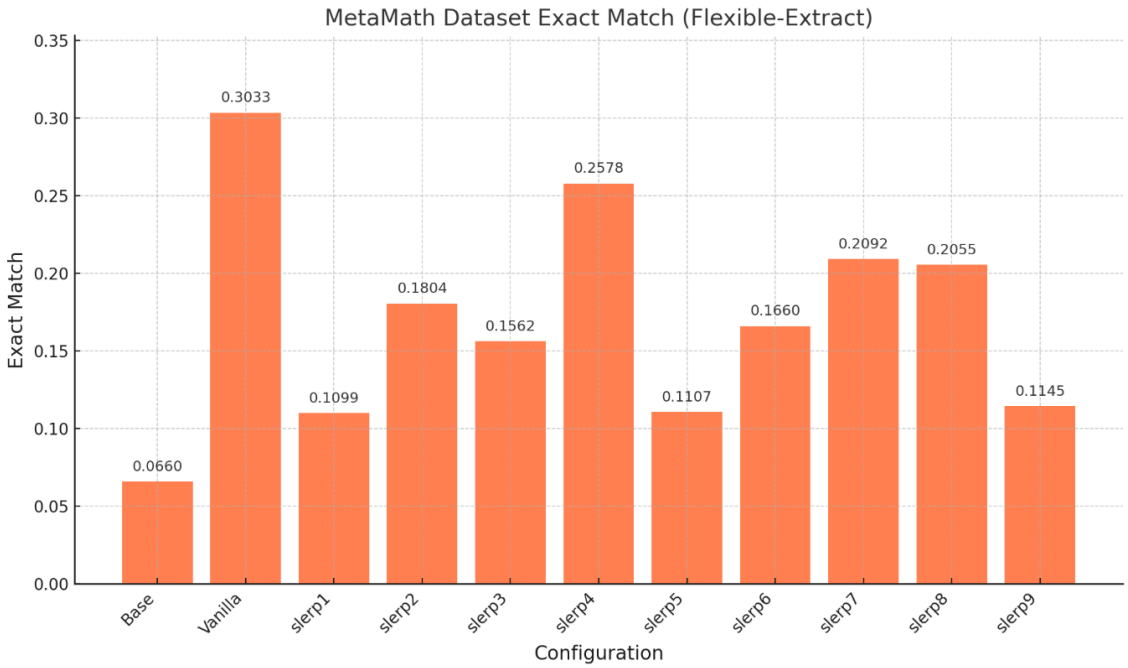

Figure 5.30 Metamath Dataset Flexible-Extract Score for SLERPs

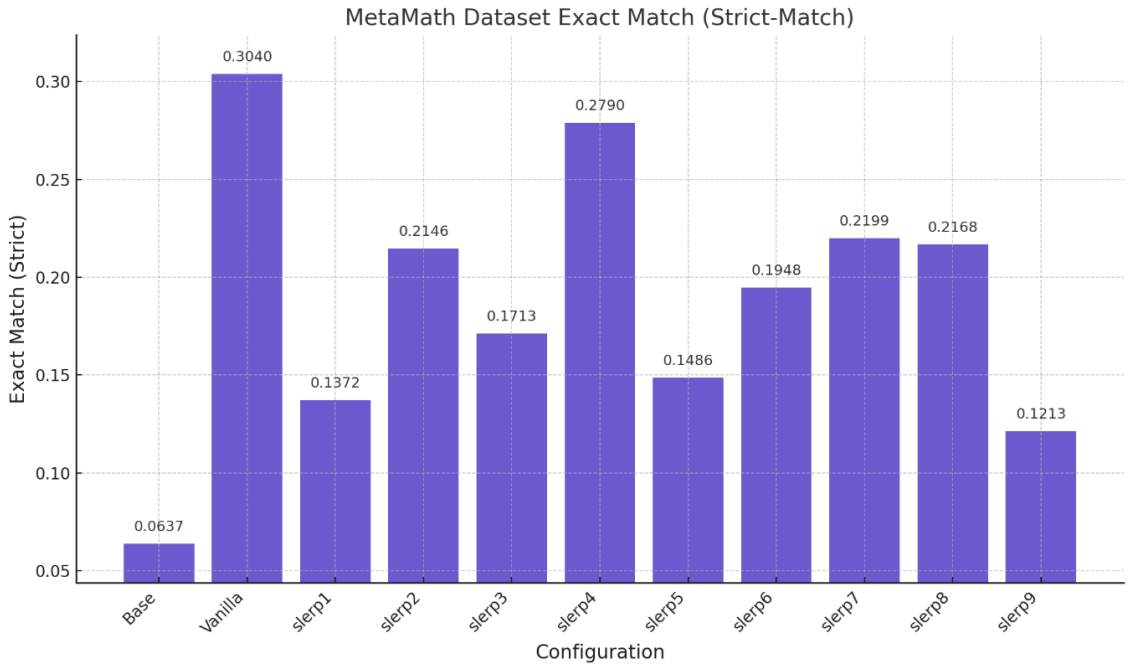

Figure 5.31 Metamath Dataset Strict-Extract Score for SLERPs

5.6.1 Comparision Table - METAMATHQA Dataset

Studies that tuned the MetaMath dataset show that SLERP plans largely maintain the increase in GSM8K exact match scores. Vanilla fine tuning raises performance. SLERP models keep 18.5 % to 80.6 % of the soft exact match scores, they also keep 23.9 % to 89.7 % of the hard exact match scores. People reached these preservation rates because SLERP plans finish quicker. They took 18.6 % to 24.7 % of the time, which meant 56 to 74.5 minutes. Vanilla training took 5 hours and 1.5 minutes. Of the strategies, SLERP 4 performed best , it preserved 80.6 % of soft scores and 89.7 % of hard scores. This success means SLERP 4 keeps almost all mathematical reasoning skills acquired from the MetaMath dataset. It does this much quicker than standard fine tuning.

SLERP 2, SLERP 7 along with SLERP 8 also had good results; they held between 48.1 % plus 60.0 % for the soft score. For the strict score, they retained 62.0% and 64.0%.

Broadly speaking, these findings show that SLERP retains performance also saves time and resources - this occurs when tuning large language models for difficult mathematical reasoning problems like MetaMath. Some strategies, such as SLERP 4, offer a better balance.[16]

Strategy	Flexible Exact Match	Strict Exact Match	Flexible Score Increase Preservation (%)	Strict Score Increase Preservation (%)	Duration (min)	% of Vanilla Duration
SLERP 1	0.1099	0.1372	%18.5	%30.5	56	%18.6
SLERP 2	0.1804	0.2146	%48.1	%62.0	62	%20.6
SLERP 3	0.1562	0.1713	%37.5	%45.0	59	%19.6
SLERP 4	0.2578	0.2790	%80.6	%89.7	74.5	%24.7
SLERP 5	0.1107	0.1486	%18.8	%35.1	62	%20.6
SLERP 6	0.1660	0.1948	%41.7	%53.7	74.5	%24.7
SLERP 7	0.2092	0.2199	%60.0	%64.0	68	%22.6
SLERP 8	0.2055	0.2168	%58.4	%62.7	68	%22.6
SLERP 9	0.1145	0.1213	%20.5	%23.9	68	%22.6

Figure 5.32 METAMATHQA Dataset Comparision Table

5.7 Training-Duration Table

5.7.1 Base, Cosmopedia

3 Epoch, 1e-6 Learning Rate, X means 400 mb training dataset.

Table 5.8 Layer Training Configurations and Durations for cosmopedia dataset

Dataset	Scale	Layers Trained	Duration (min)
cosmopedia	X	16	196
cosmopedia	2.5X	16	500
cosmopedia	X/4	4	37.5
cosmopedia	X/4	6	39
cosmopedia	X/4	8	42
cosmopedia	X/4	12	45
cosmopedia	X/4	16	49.5

5.7.2 Random Model Weights, Cosmopedia

1 Epoch, 2e-4 Learning Rate, X means 400 mb training dataset.

Table 5.9 Layer Training Configurations and Durations for cosmopedia dataset - random model weights

Dataset	Scale	Layers Trained	Duration (min)
cosmopedia	X	16	44.00
cosmopedia	2.5X	16	112.00
cosmopedia	X/4	4	9.00
cosmopedia	X/4	6	9.50
cosmopedia	X/4	8	9.75
cosmopedia	X/4	12	10.50
cosmopedia	X/4	16	11.00

5.7.3 GSM8K

3 Epoch, 1e-6 Learning Rate

Table 5.10 Layer Training Configurations and Durations for GSM8K Train dataset

Dataset	Scale	Layers Trained	Duration (min)
gsm8k	X	16	5.00
gsm8k	X/4	4	1.00
gsm8k	X/4	6	1.10
gsm8k	X/4	8	1.15
gsm8k	X/4	12	1.20
gsm8k	X/4	16	1.25

5.7.4 MetaMath

1 Epoch, 1e-6 Learning Rate

Table 5.11 Layer Training Configurations and Durations for MetaMathQA dataset

Dataset	Scale	Layers Trained	Duration (H:MM:SS)
metamath	X	16	5:01:30
metamath	X/4	4	56:00
metamath	X/4	6	59:00
metamath	X/4	8	1:02:00
metamath	X/4	12	1:08:00
metamath	X/4	16	1:14:30

5.8 Training : Step by Step

In this section, vanilla training of the models is divided into 8 steps, and slerp 4 training is divided into 2 steps. When the 1st step of vanilla training is finished, the 4th step of slerp 4 training, that is, half of it, is finished. The success of the models is compared according to time. While similar data was used as before in the random model weight, gsm8k and metamath sections, a more extensive examination was made with a 1 GB data set in the plain cosmopedia section. There are 7 steps in every vanilla model and the 8th model is the final version.

For normal cosmopedia, total training time is 8 hours 20 minutes = 500 minutes.

Step 1 -> 500/8 * 1 = 62 minutes later

Step 2 -> 500/8 *2 = 124 minutes later

Step 7 -> 500/8 * 7 can be said to be like 437 minutes later.

This comparison will be made on Slerp-4, which is the most successful method as seen in the comparisons. The state of slerp-4 exactly halfway through training is recorded. This is called slerp-mid-4. Slerp-mid-4 is used in comparison.The training time of each sub-model of slerp 4 in the 1GB dataset is 124 minutes. Since the models are trained in parallel, assuming that all of them start training at the same time,

After 62 minutes, Vanilla 1gb finishes its 1st step, slerp-4 1gb finishes its 4th step.

After 124 minutes, Vanilla 1gb finishes its 2nd step, slerp-4 1gb is now completely finished.

Hellaswag accuracy normalized and validation loss were considered as evaluation metrics for 62nd and 124th minutes.

5.8.1 Vanilla vs Slerp-4 (1Gb Dataset)

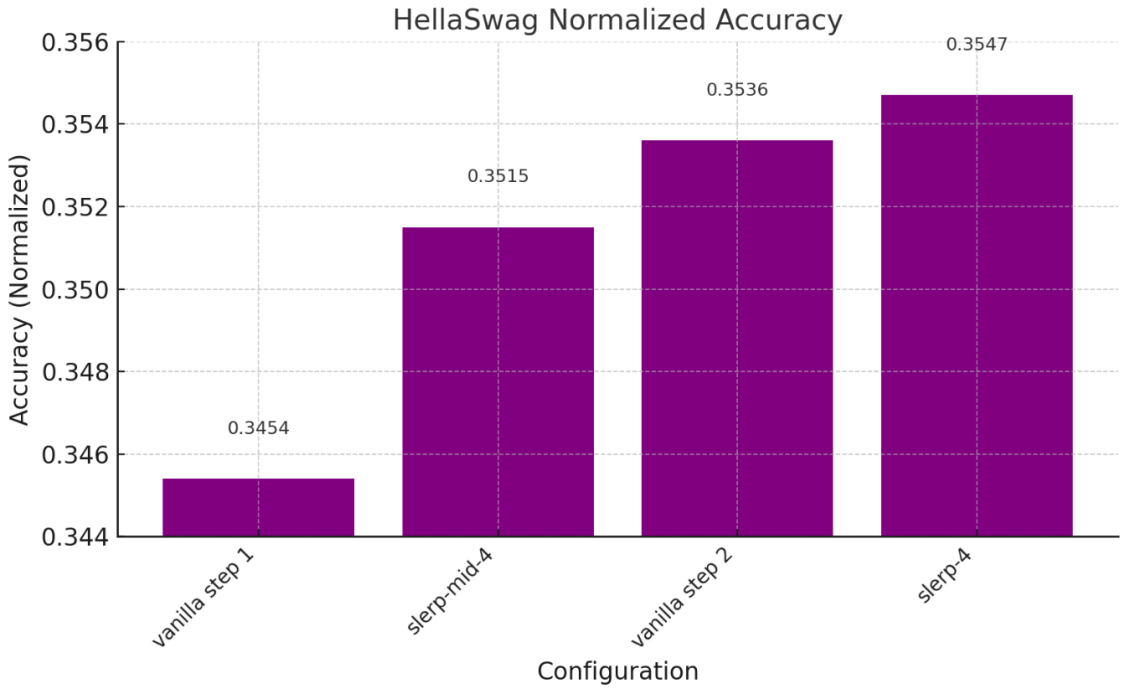

Figure 5.33 HellaSwag Accuracy Normalized - cosmopedia

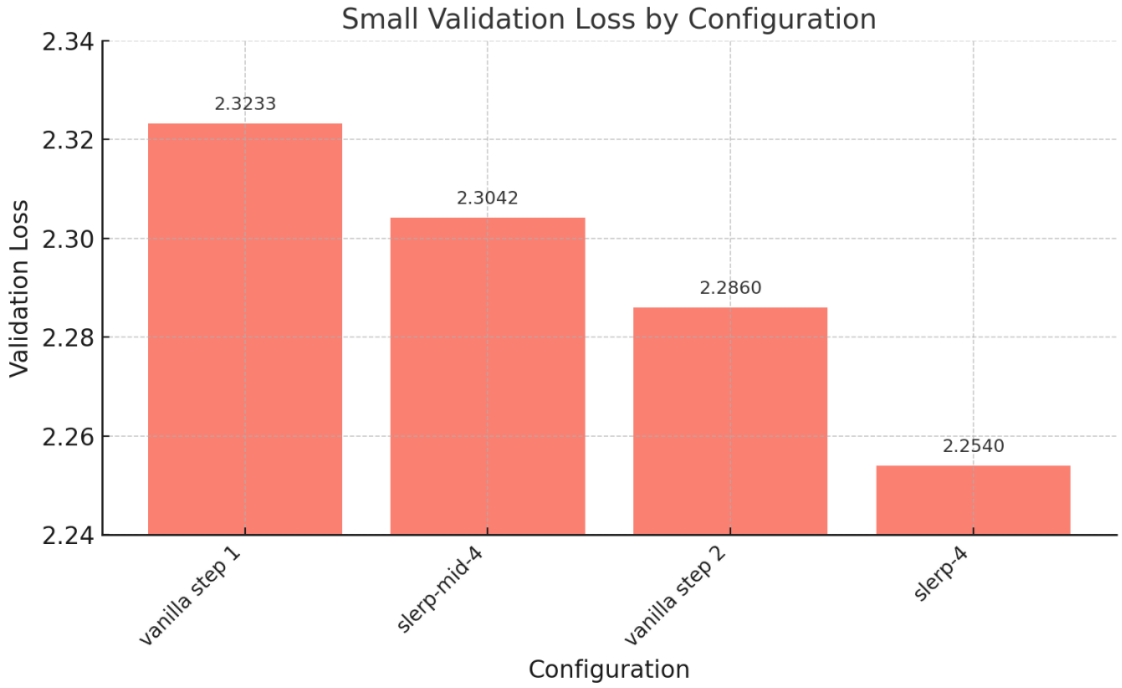

Figure 5.34 Small Validation Loss - cosmopedia

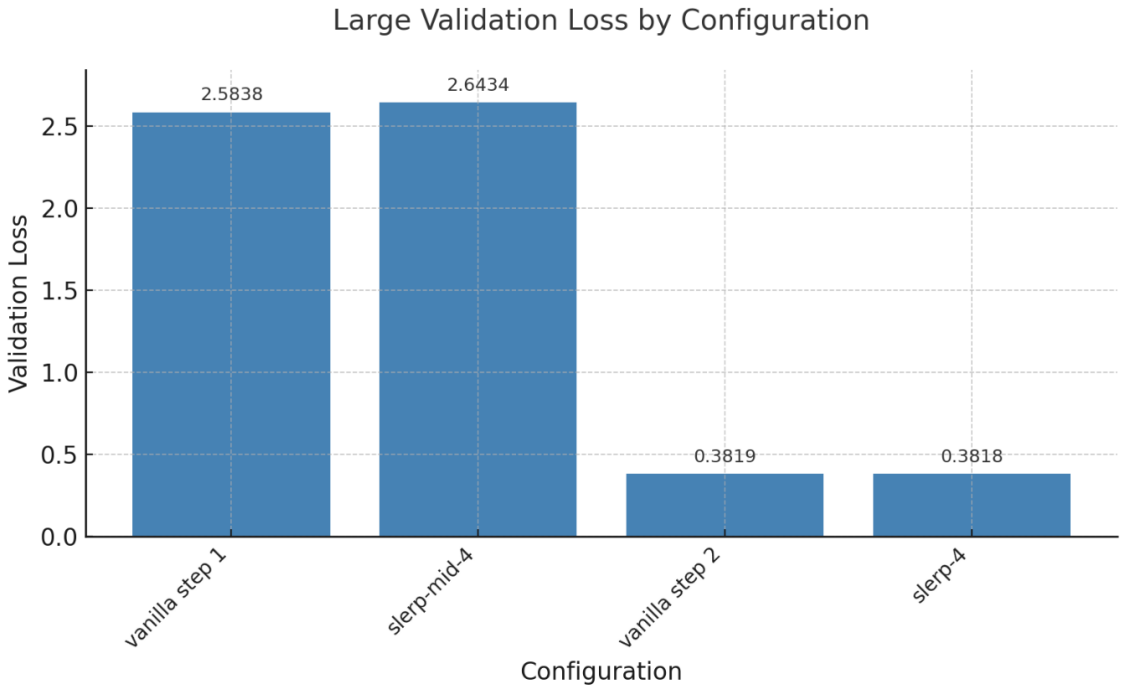

Figure 5.35 Large Validation Loss - cosmopedia

The comparison showed that SLERP-4 ran faster and used less effort than vanilla fine tuning. SLERP-mid-4 (2.3042) finished the first step of vanilla training (2.3233) with a better small validation loss. SLERP-4 (2.2540) then had an even lower small validation

loss as it equaled the second step of vanilla training (2.2860).

For large validation loss, the sharp fall in the second step of vanilla training (0.3819) almost equaled SLERP-4 (0.3818) at the same time. SLERP-4 (0.2512) showed the top value in Hellaswag normalized accuracy scores - it did better than the second step of vanilla training (0.2497). These results confirm that SLERP-4 reaches the performance of vanilla training in less time, needing only two steps instead of eight. It also goes beyond vanilla in some measures, showing that it uses little time and performs well.

5.8.2 Vanilla vs Slerp-4 - Random Model Weights

In this section the training data is 400mb, and the results at the 5.30 and 11.00 will be compared. Vanilla training 44 minutes -> each step $44/8 = 5.30$ minutes

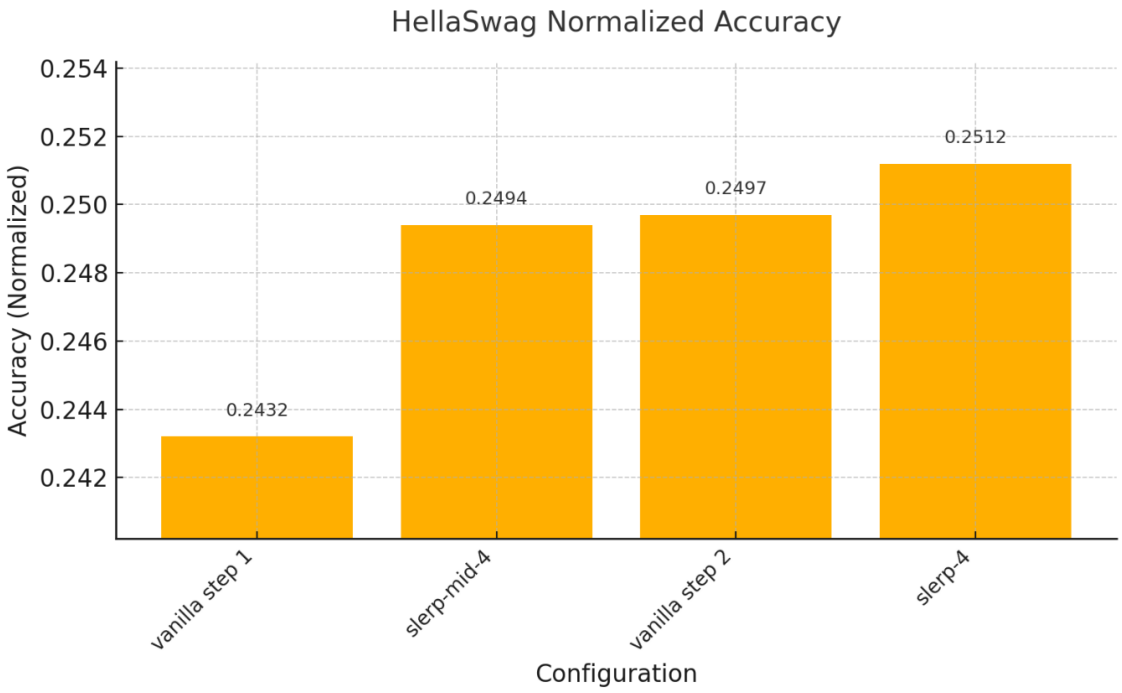

Figure 5.36 HellaSwag Accuracy Normalized - cosmopedia - random model weights

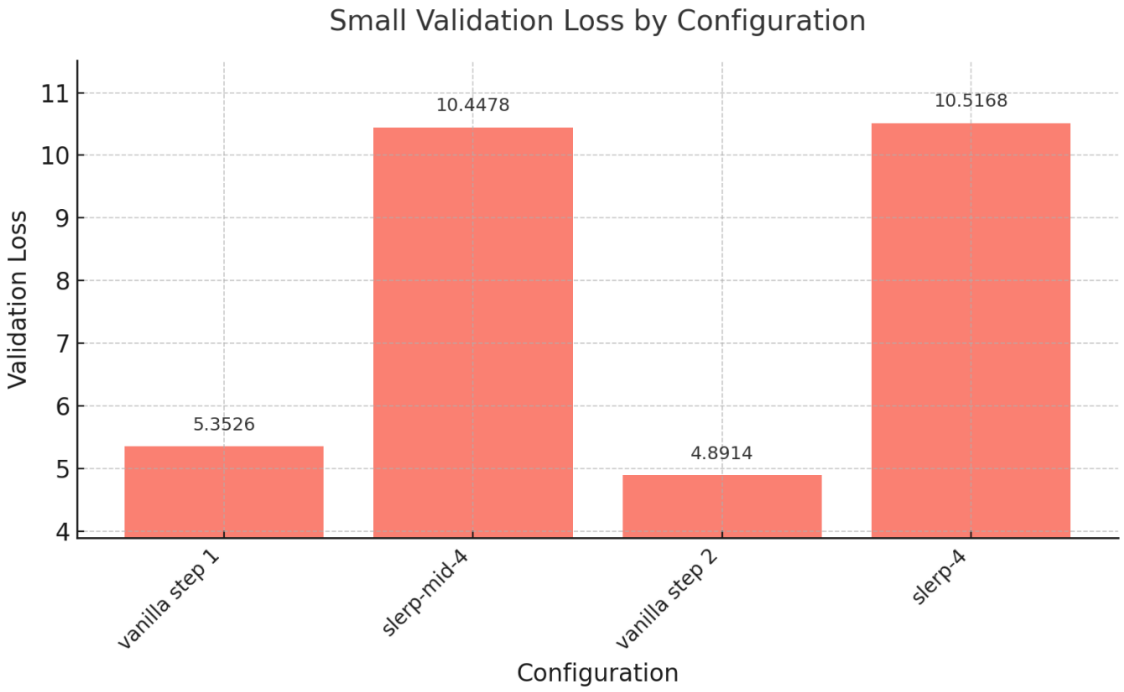

Figure 5.37 Small Validation Loss - cosmopedia - random model weights

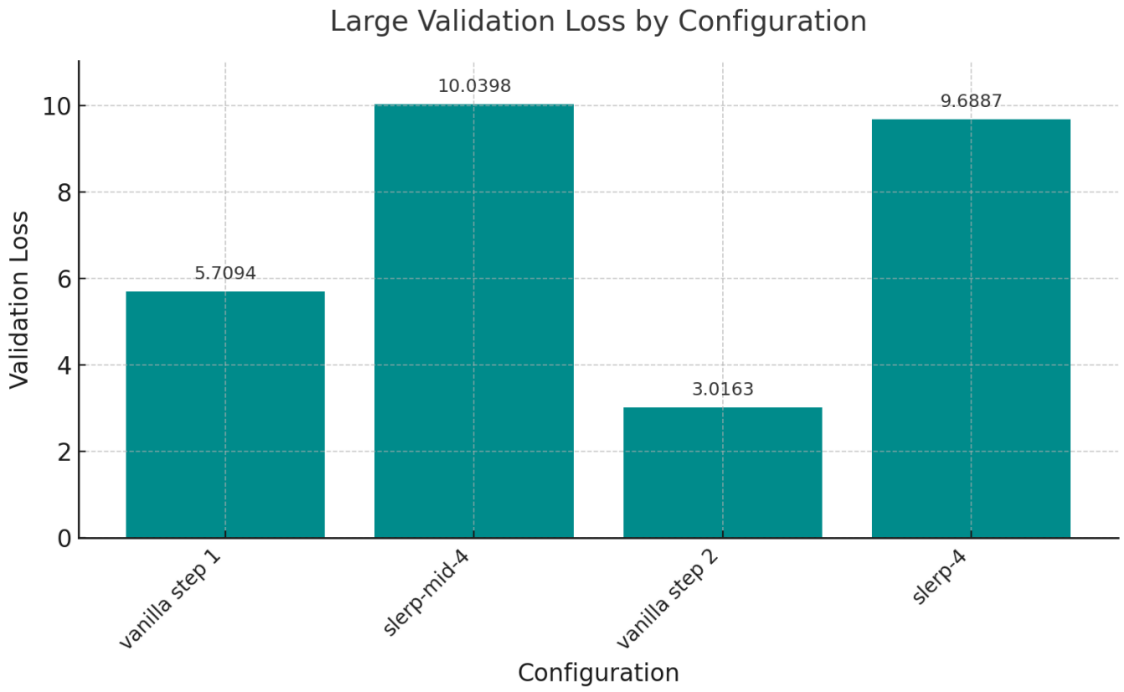

Figure 5.38 Large Validation Loss - cosmopedia - random model weights

A step-by-step analysis of the Llama-3.2-1B model, which began with random weights on the 400MB Cosmopedia dataset, shows something. The SLERP-4 plan performs worse than the vanilla plan when it comes to lowering validation losses. This goes

against what we thought would happen. Small and large validation losses go down as the vanilla plan moves forward. But they go up as the SLERP-4 plan works on them. For example, a small validation loss rises from 4.8914 in vanilla step 2 to 10.5168 in slerp-4.

Hellaswag just barely beats vanilla step 2 with its normalized accuracy scores - it has 0.2512 while vanilla step 2 has 0.2497. The absolute scores that we get are lower than those from models that already had training - these results show that for methods that mix weights, such as SLERP, the model needs to know some things at the start. For a model that begins with random settings, these plans could make the current state worse instead of helping performance.

5.8.3 Vanilla vs Slerp-4 - GSM8K Train

Vanilla training 5 min -> 300 seconds -> Each step 300/8 -> 37 seconds

Vanilla training 1st step -> 37*1 = 37th second

Slerp-4- Mid (4th step)-> 85/2 -> 42nd second

Since it is a mathematical model, there is no validation loss. The comparison was made on flexible-extract and strict-extract.

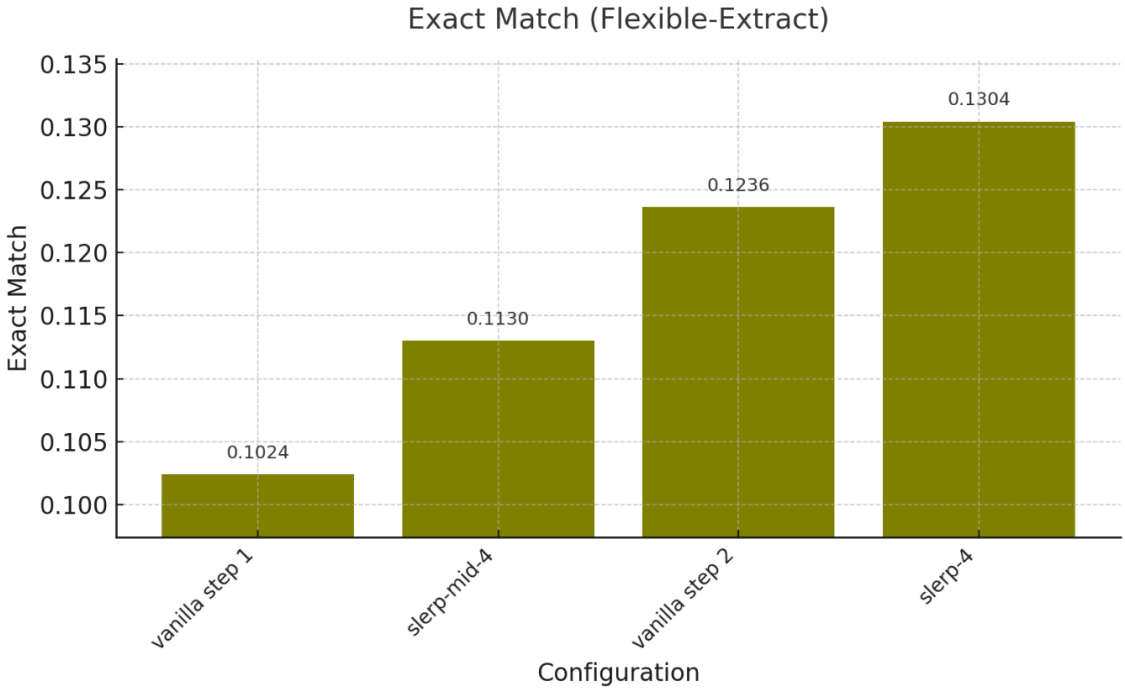

Figure 5.39 flexible-extract - GSM8K

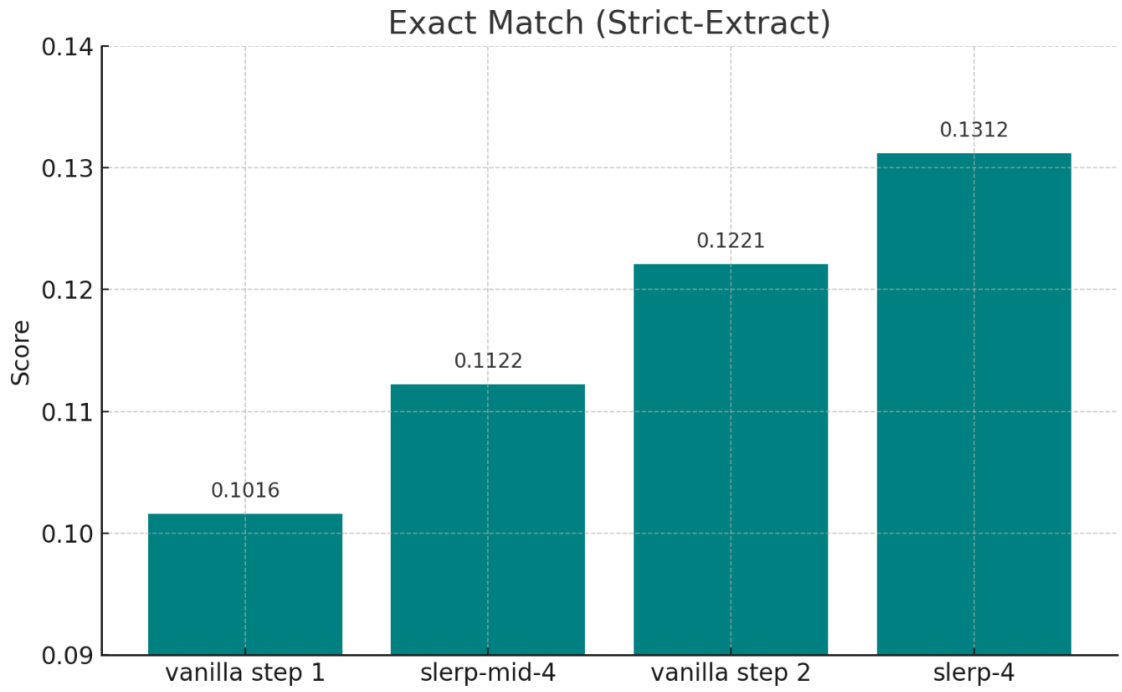

Figure 5.40 strict-extract - GSM8K

A demonstration clearly showed SLERP-4 performed better and used less time than vanilla training. In flexible plus strict exact match scores, the "slerp-mid-4" version of SLERP-4 gained higher scores than vanilla step 1. For example, it received 0.1130 versus 0.1024 in flexible, and 0.1112 versus 0.1016 in strict. The full SLERP-4 showed the highest performance, exceeding the scores vanilla training reached in its second step. For instance, it got 0.1304 versus 0.1236 in flexible, also 0.1312 versus 0.1221 in strict - these results support SLERP-4 as an effective strategy - it offers both quick work and good results, so it goes past vanilla training more quickly when doing complex tasks like mathematical reasoning.

5.8.4 Vanilla vs Slerp-4 - Metamath

Vanilla training 5 hours -> 300 min

Each step 300/8 -> 37 min

Slerp 4 all 1 hour 15 min -> 75 min

Slerp-Mid-4 75/2 -> 37 min

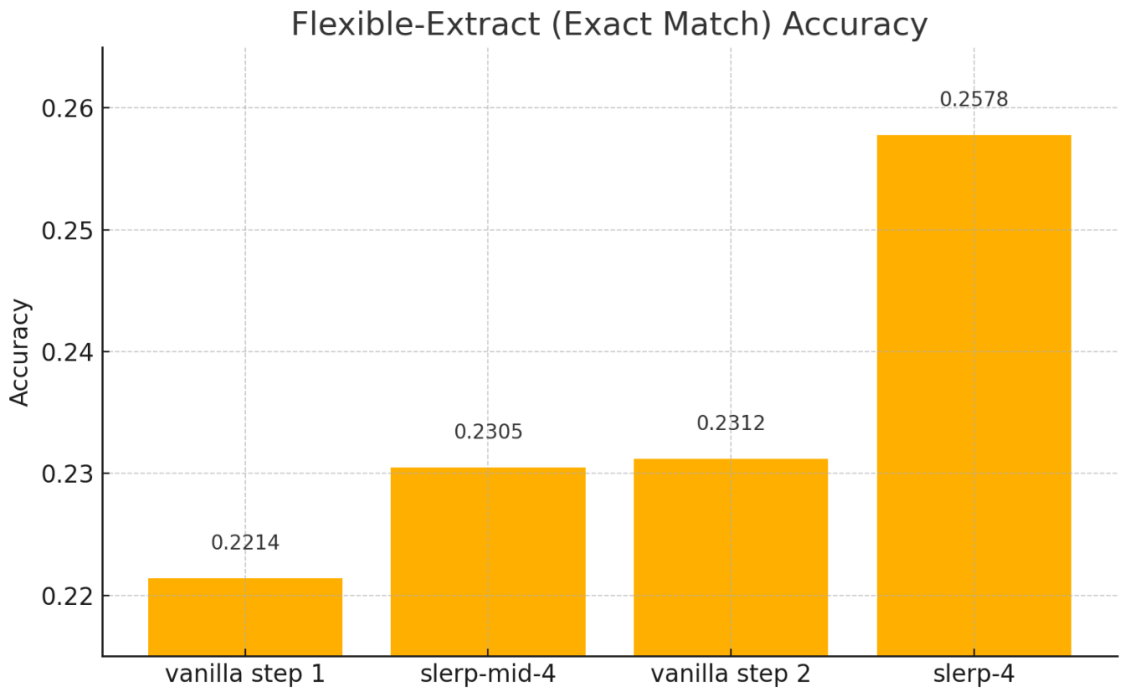

Figure 5.41 flexible-extract - Metamath

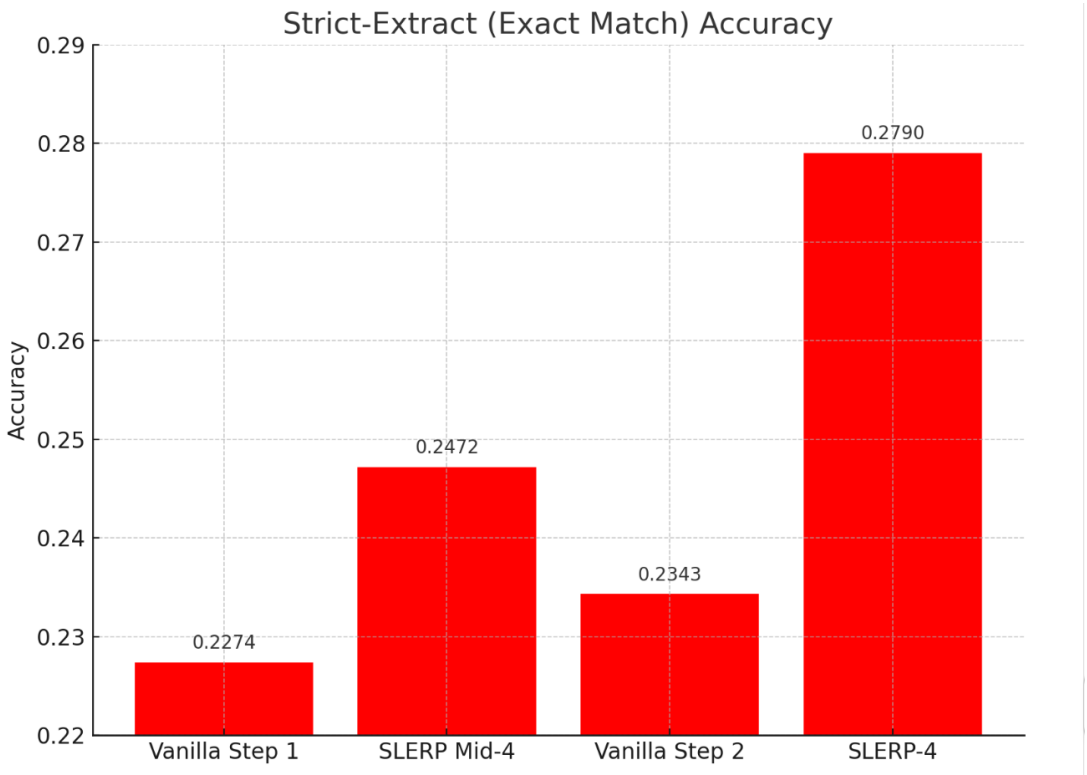

Figure 5.42 strict-extract - Metamath

For both exact match metrics (Flexible-Extract and Strict-Extract), the half-way version of SLERP-4, "slerp-mid-4", achieved higher accuracy scores than the first step of vanilla training (vanilla step 1) (e.g. 0.2305 vs 0.2214 in Flexible, 0.2472 vs 0.2274 in Strict). More strikingly, the full version of SLERP-4 showed the highest performance, significantly exceeding the scores obtained in the second step of vanilla training (vanilla step 2) (e.g. 0.2578 vs 0.2312 in Flexible, 0.2790 vs 0.2343 in Strict). These findings still shows that SLERP-4 is an effective strategy for mathematical reasoning datasets like MetaMath.

5.8.5 1 Gb Cosmopedia Vanilla Steps Review

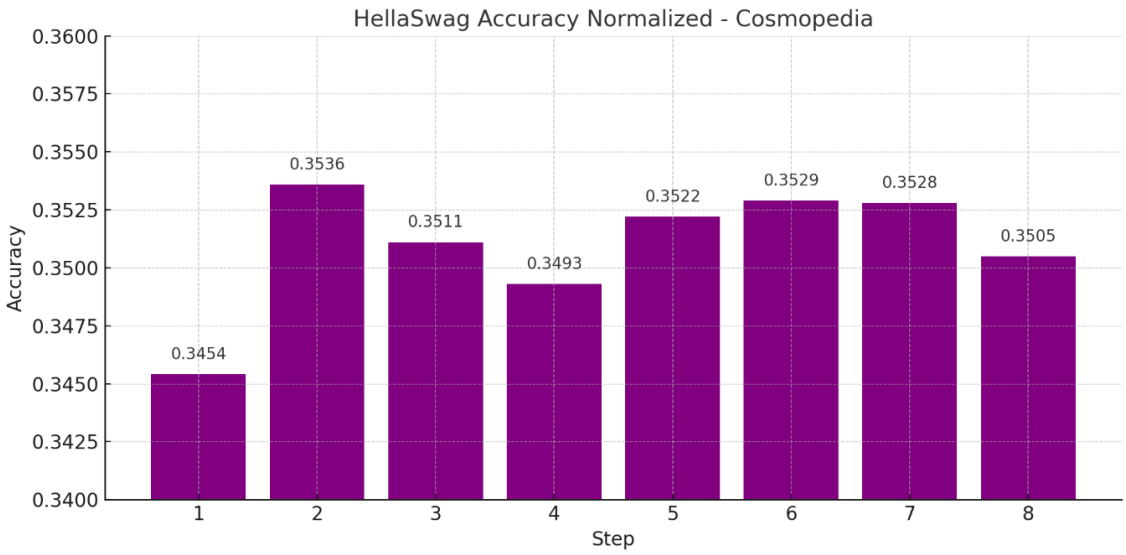

Figure 5.43 Cosmopedia 8 Steps HellaSwag Accuracy Normalized Score

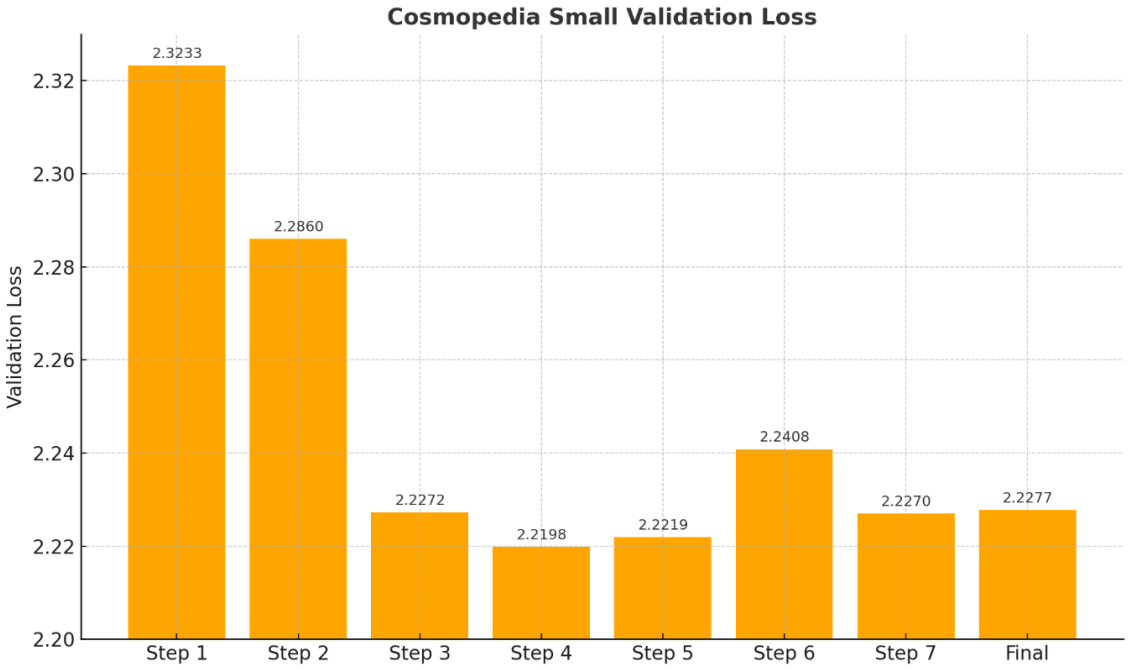

Figure 5.44 Cosmopedia 8 Steps Small Validation Loss

6
Conclusion and Discussion

The project's goal was to develop parallel training strategies by splitting the dataset and training them with different layers of the model. While reducing the training time, the aim were to preserve model's success rate. In the studies experiments were conducted on the Llama-3.2-1B model using various datasets such as Cosmopedia, GSM8K Train and MetaMath. The results show that SLERP strategies provided less training times compared to vanilla fine-tuning. In all studies, training with dividing dataset was completed in an average of 20-25% of the vanilla training time with parallel training.

In terms of performance, SLERP strategies, especially SLERP 4, significantly preserved the success rate achieved by vanilla fine-tuning, and even surpassed it in some cases. Step-by-step analysis on Cosmopedia revealed that SLERP-4 surpassed certain steps of vanilla training in Hellaswag accuracy and validation losses in a shorter time. On GSM8K and MetaMath datasets, SLERP 4 preserved more than 60% of the exact match score increase of vanilla training and up to 80% in MetaMath, proving the success of this strategy in mathematical data. However, the failure of SLERP strategies in models initialized with random weights in our project showed that this strategy is not functional when pre-trained weights are not taken into account. The performance preservation of SLERP in models initialized with random weights was not as strong as in other datasets, which showed the sensitivity of the strategy to certain dataset features.

In general, SLERP strategies have great potential to make the fine-tuning process of large language models faster and more cost-effective. These strategies facilitate complex model adaptation processes, especially when used together with tools such as the model merging tool Mergekit and lm-eval for evaluation. This study has shown that reducing training time does not lead to a loss of performance. Also with the right strategies the model's capabilities can be largely saved.

References

[1] H. Jin, X. Han, J. Yang, Z. Jiang, C.-Y. Chang, and X. Hu, *Growlength: Accelerating llms pretraining by progressively growing training length*, 2023. arXiv: 2310.00576 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2310.00576.

[2] K. Mishra, T. Soliman, A. Ramakrishna, A. Kumar, and A. Galstyan, "Correcting language model outputs by editing salient layers," 2024. [Online]. Available: https://www.amazon.science/publications/correcting-language-model-outputs-by-editing-salient-layers.

[3] M. Samragh *et al.*, *Scaling smart: Accelerating large language model pre-training with small model initialization*, 2024. arXiv: 2409.12903 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2409.12903.

[4] R.-G. Dumitru, P.-I. Clotan, V. Yadav, D. Peteleaza, and M. Surdeanu, *Change is the only constant: Dynamic llm slicing based on layer redundancy*, 2024. arXiv: 2411.03513 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2411.03513.

[5] I. S. Singh *et al.*, *Chunkrag: Novel llm-chunk filtering method for rag systems*, 2025. arXiv: 2410.19572 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2410.19572.

[6] X. Yuan *et al.*, *Efficient long context fine-tuning with chunk flow*, 2025. arXiv: 2503.02356 [cs.DC]. [Online]. Available: https://arxiv.org/abs/2503.02356.

[7] S. K. Pentyala *et al.*, *Paft: A parallel training paradigm for effective llm fine-tuning*, 2024. arXiv: 2406.17923 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2406.17923.

[8] Z. Huang *et al.*, *Accelerating pre-training of multimodal llms via chain-of-sight*, 2024. arXiv: 2407.15819 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2407.15819.

[9] A. Paszke *et al.*, *Pytorch: An imperative style, high-performance deep learning library*, 2019. arXiv: 1912.01703 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1912.01703.

[10] T. Hwang, H. Seo, J. Jung, and S. Jung, *Exploring selective layer freezing strategies in transformer fine-tuning: NLI classifiers with sub-3b parameter models*, 2025. [Online]. Available: https://openreview.net/forum?id=kvBuxFxSLR.

[11] C. Goddard *et al.*, *Arcee's mergekit: A toolkit for merging large language models*, 2025. arXiv: 2403.13257 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2403.13257.

[12] Y. Zhang, "Simple llama merge: What kind of LLM do we need?" In *LLM Merging Competition at NeurIPS 2024*, 2024. [Online]. Available: https://openreview.net/forum?id=VndTgXbAgz.

[13] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, *Hellaswag: Can a machine really finish your sentence?* 2019. arXiv: 1905.07830 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1905.07830.

[14] X. Li *et al.*, *Hellaswag-pro: A large-scale bilingual benchmark for evaluating the robustness of llms in commonsense reasoning*, 2025. arXiv: 2502.11393 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2502.11393.

[15] P. Chizhov, M. Nee, P.-C. Langlais, and I. P. Yamshchikov, *What the hellaswag? on the validity of common-sense reasoning benchmarks*, 2025. arXiv: 2504.07825 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2504.07825.

[16] L. Yu *et al.*, *Metamath: Bootstrap your own mathematical questions for large language models*, 2024. arXiv: 2309.12284 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2309.12284.

Curriculum Vitae

FIRST MEMBER

Name-Surname: TALHA ÇELİK
Birthdate and Place of Birth: 16.07.2003, İstanbul
E-mail: talha.celik1@std.yildiz.edu.tr
Phone: 0537 456 46 90
Practical Training: Turkcell Technology - Artificial Intelligence and Analytic Solutions
Bosch Türkiye - Information Technology

SECOND MEMBER

Name-Surname: HÜSEYİN SAİD ARICI
Birthdate and Place of Birth: 19.02.2002, İstanbul
E-mail: said.arici@std.yildiz.tr
Phone: 0536 073 39 25
Practical Training: SOFT İş Çözümleri A.Ş - Software Department

Project System Informations

System and Software: MacOS, Google Colab, Python
Required RAM: 32GB
Required Disk: 64GB